Available online at www.sciencedirect.com

ScienceDirect

Neural Networks 20 (2007) 1095–1108

Neural Networks

www.elsevier.com/locate/neunet

Analysis and test of efficient methods for building recursive deterministic perceptron neural networks

David A. Elizondo[a,*], Ralph Birkenhead[a], Mario Góngora[a], Eric Taillard[b], Patrick Luyima[a]

[a] Centre for Computational Intelligence, School of Computing, Faculty of Computing Sciences and Engineering, De Montfort University, Leicester, UK
[b] EIVD, University of Applied Sciences of Western Switzerland, Route de Cheseaux 1, Case postale, CH-1401 Yverdon, Switzerland

Received 20 June 2006; accepted 12 July 2007

Abstract

The Recursive Deterministic Perceptron (RDP) feed-forward multilayer neural network is a generalisation of the single layer perceptron topology. This model is capable of solving any two-class classification problem as opposed to the single layer perceptron which can only solve classification problems dealing with linearly separable sets. For all classification problems, the construction of an RDP is done automatically and convergence is always guaranteed. Three methods for constructing RDP neural networks exist: Batch, Incremental, and Modular. The Batch method has been extensively tested and it has been shown to produce results comparable with those obtained with other neural network methods such as Back Propagation, Cascade Correlation, Rulex, and Ruleneg. However, no testing has been done before on the Incremental and Modular methods. Contrary to the Batch method, the complexity of these two methods is not NP-Complete. For the first time, a study on the three methods is presented. This study will allow the highlighting of the main advantages and disadvantages of each of these methods by comparing the results obtained while building RDP neural networks with the three methods in terms of the convergence time, the level of generalisation, and the topology size. The networks were trained and tested using the following standard benchmark classification datasets: IRIS, SOYBEAN, and Wisconsin Breast Cancer. The results obtained show the effectiveness of the Incremental and the Modular methods which are as good as that of the NP-Complete Batch method but with a much lower complexity level. The results obtained with the RDP are comparable to those obtained with the backpropagation and the Cascade Correlation algorithms.

Keywords: Recursive deterministic perceptron; Batch learning; Incremental learning; Modular learning; Performance sensitivity analysis; Convergence time; Generalisation; Topology

1. Introduction

The single layer perceptron topology (SLPT), introduced by Rosenblatt (1962), was one of the first neural network models shown to be able to learn how to classify patterns. However, Minsky and Papert (1969) showed that this topology is only capable of learning linearly separable patterns. This is a big limitation since most classification problems are non-linearly separable (NLS). The Recursive Deterministic Perceptron feed-forward neural network (Elizondo, 1997; Tajine & Elizondo, 1997) is a multilayer generalisation of this topology, which provides a solution to any two-class classification problem (even if the two classes are NLS). The RDP neural network is guaranteed to converge, does not need any parameters from the user, does not suffer from catastrophic interference, and provides transparent extraction of knowledge as a set of rules (Elizondo & Gongora, 2005). These rules can be generated, using a computational geometry approach, as a finite union of open polyhedral sets. Although this study will focus on two-class classification problems, an m class ($m > 2$) generalisation of the RDP exists (Tajine, Elizondo, Fiesler, & Korczak, 1997).

The approach taken by the RDP is to augment the affine dimension of the input vectors, by adding to these vectors the outputs of a sequence of intermediate neurons as new components. Each intermediate neuron (IN) is an SLPT that can linearly separate an LS subset of points from all the rest of the points in an NLS problem. This allows for additional degrees of freedom for transforming the original NLS problem into a

* Corresponding author.
E-mail addresses: elizondo@dmu.ac.uk (D.A. Elizondo), rab@dmu.ac.uk (R. Birkenhead), mgongora@dmu.ac.uk (M. Góngora), Eric.Taillard@heig-vd.ch (E. Taillard), pluyima@dmu.ac.uk (P. Luyima).

0893-6080/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.07.009

linearly separable (LS) one (two subsets X and Y of $\mathbb{R}^d$ are said to be linearly separable if there exists a hyperplane such that the elements of X and Y lie on the two opposite sides of $\mathbb{R}^d$ delimited by this hyperplane). These intermediate neurons are added progressively, one at each time step. The algorithm stops when the two classes become LS.

Three methods exist for constructing an RDP neural network. These methods are Batch, Incremental, and Modular learning, and were introduced by one of the authors in Tajine and Elizondo (1998). They produce a multilayer topology which, contrary to the SLPT, is capable of solving any classification problem even if the classes considered are NLS. The Batch method chooses at each step, a linearly separable subset of maximum cardinality and produces small topologies. This has been proven to be NP-complete in Elizondo (1997) when the cardinality of both classes and the dimension d are arbitrary. This result was obtained by using the NP-completeness of the Open Hemisphere problem (Johnson & Preparata, 1978). The Batch method has been extensively tested and it has been shown to produce results comparable with those obtained with other neural network methods such as Back Propagation, Cascade Correlation, Rulex, and Ruleneg (Elizondo, 1997). However, no testing has been done on the Incremental and Modular methods which offer polynomial time complexity with slightly larger topologies. A proof of the complexity bound is given in Elizondo (1997). In this paper, for the first time, a comparison study on the performance of the three methods is presented. This study highlights the main advantages and disadvantages of each of the methods in terms of the level of generalisation obtained by using the three methods to build RDP neural networks for three machine learning benchmark classification problems. The results obtained in this research show the potential for using the Incremental and Modular methods for building RDP neural networks and thus help to popularise the use of RDP neural networks for solving classification problems.

For completeness, results obtained on the same datasets used for constructing backpropagation and Cascade Correlation neural networks are also presented.

To illustrate the principle for building an RDP neural network the NLS 2-input Exclusive-OR (XOR) problem can be used. For this illustration, the Batch learning method will be used. This problem consists of classifying the two classes $X = \{(0,0),(1,1)\}$ and $Y = \{(0,1),(1,0)\}$, which are NLS. To perform the NLS to LS transformation, a subset of patterns of the same class which is LS from the rest of the patterns is selected. Fig. 2 shows the four possible subsets which are linearly separable from the rest of the dataset that can be used to build the RDP. Any one of these LS subsets could be equally used. We have decided to use the subset $\{(0,0)\} \subset X \cup Y$ since $\{(0,0)\}$ and $\{(0,1),(1,0),(1,1)\}$ are LS (by the hyperplane $P_t = \{(x_1,x_2) \in \mathbb{R}^2 \mid 2*x_1 + 2*x_2 - 1 = 0\}$) as illustrated in Fig. 2(d). Thus, the intermediate neuron IN1 corresponding to the SLPT of weight vector $\vec{w} = (2,2)$ and threshold $t = -1$ "associated" with the hyperplane P_t is created. The output of IN1 allows us to add, to the input vectors, one column by assigning the value -1 to the input pattern

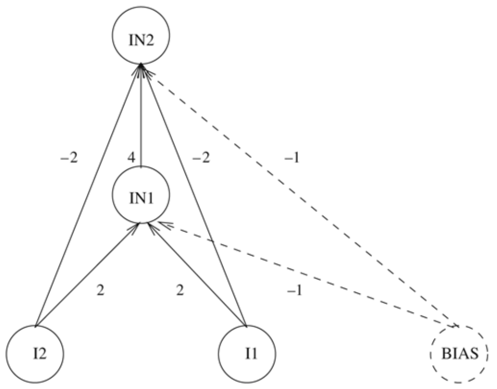

Fig. 1. RDP neural network for solving the XOR classification problem (the intermediate layer contains only one component, IN1, and IN2 corresponds to the output neuron of the RDP).

$\{(0,0)\}$, and the value 1 to the remaining three input patterns $\{(0,1),(1,0),(1,1)\}$. So, this SLPT produces the following sets of augmented input vectors: $X' = \{(0,0,\underline{-1}),(1,1,\underline{1})\}$ and $Y' = \{(0,1,\underline{1}),(1,0,\underline{1})\}$. Now, X' and Y' are LS by the hyperplane $P_2 = \{(x_1,x_2,x_3) \in \mathbb{R}^3 \mid -2*x_1 - 2*x_2 + 4*x_3 - 1 = 0\}$. Next, a second intermediate neuron IN2 (output neuron) which corresponds to the SLPT with the weight vector $\vec{w} = (-2,-2,4)$, and threshold $t = -1$, associated with the hyperplane P_2, is created (Fig. 3). The final result is a two layer RDP neural network solving the XOR classification problem since the output value of this neural network is -1 for the vector patterns $\{(0,0),(1,1)\}$, and 1 for the remaining vector patterns $\{(0,1),(1,0)\}$. Fig. 1 shows a graphical representation of the final RDP neural network which solves the XOR classification problem. Its formal description is $[((2,2),-1),((-2,-2,4),-1)]$. See next section for the formal definition.

Contrary to other learning methods such as backpropagation, the construction of an RDP neural network, with a 100% correct decision boundary on all training datasets, is always guaranteed. The formal mathematical proof for the guarantee of convergence of the RDP can found in Elizondo (1997). For any given classification problem, there are an infinite number of RDP neural networks which provide a solution (all with a 100% correct decision boundary on all training datasets). The choice of a particular RDP will affect the level of generalisation.

The RDP is a linear method that combines, in a cascade fashion, a series of linear single layer perceptrons to build a neural network. Other linear learning methods on LS and NLS data include Support Vector Machines (SVM) (Atiya, 2005; Boser, Guyon, & Vapnik, 1992; Cortes & Vapnik, 1995; Cristianini & Shawe-Taylor, 2003). They are trained by finding a hyperplane that separates the dataset by solving a constrained quadratic programming optimisation problem. In the case of NLS data, the data is mapped into some other Euclidean space. Thus, SVM is still doing a linear separation but in a different space. The user must find a kernel function to be used on an SVM and this can be difficult. In its simplest form a kernel function calculates the dot product of two training vectors. This helps in the evaluation of the correct classification of each training vector. One of the advantages that the RDP has over

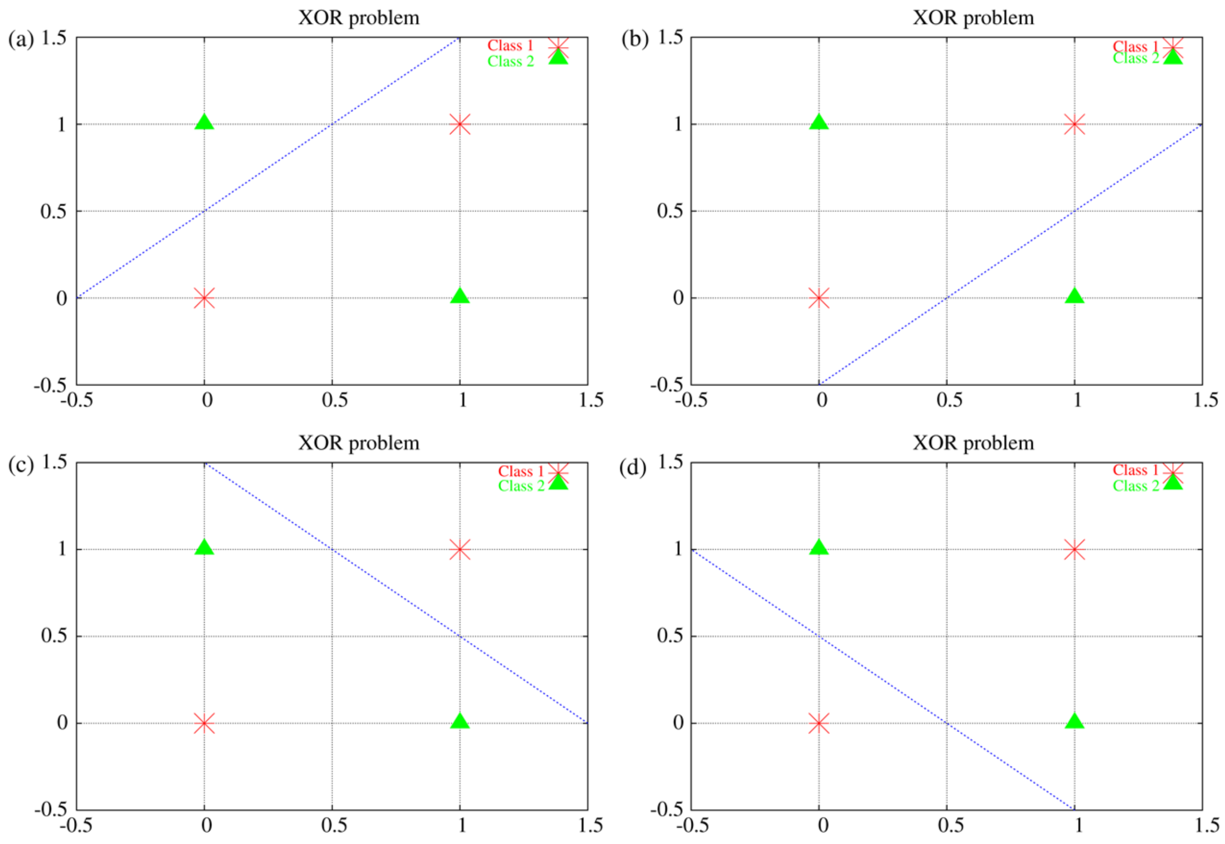

Fig. 2. XOR classification problem and all four possible subsets of the same class which are linearly separable from the rest.

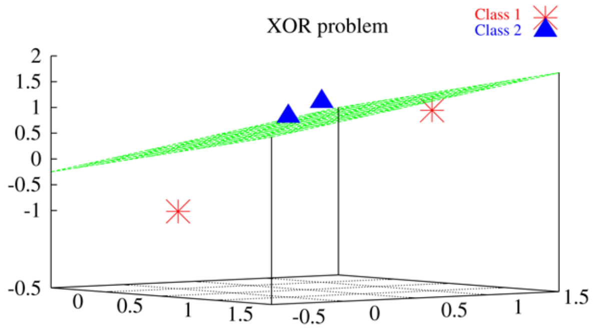

Fig. 3. Hyperplane $P_2 = \{(x_1, x_2, x_3) \in \mathbb{R}^3 \mid -2*x_1 - 2*x_2 + 4*x_3 - 1 = 0\}$ used by the RDP to solve the XOR classification problem.

the SVM, is that its construction is done automatically without the need for any user definable parameters.

This paper is divided into seven sections. The notion of linear separability as well as some of the methods for testing it, are given in Section 2. In this section also, some of the notions used throughout this paper are introduced. Section 3 introduces the three methods for building RDP neural networks. In Section 4, the procedure used to compare the three learning methods is presented. Three machine learning benchmarks (Iris, Soybean, and Wisconsin Breast Cancer) were used (Blake, Newman, Hettich, & Merz, 1998) and datasets were generated using cross validation. The three learning methods are compared in terms of their level of generalisation, the size of their topology, and their convergence time. For completeness purposes, results using the backpropagation and the Cascade Correlation algorithms are also presented. Section 5 presents some results and discussion. A summary and some conclusions are presented in Section 6. In the last section, some future research ideas are proposed.

2. Linear separability

Two subsets X and Y of $\mathbb{R}^d$ are said to be linearly separable (LS) if there exists a hyperplane P of $\mathbb{R}^d$ such that the elements of X and those of Y lie on opposite sides of it. Fig. 4 shows an example of both an LS (a) and an NLS (b) set of points. Squares and circles denote the two classes.

The Recursive Deterministic Perceptron can always distinguish, in a deterministic way, two or more classes (even if the two classes are not linearly separable). The idea behind the construction of an RDP is to augment the affine dimension of the input vector by adding to these vectors the outputs of a sequence of intermediate neurons as new components. Each intermediate neuron corresponds to a single layer perceptron and it is characterised by a hyperplane which linearly separates an LS subset, taken from the non-LS (NLS) original set, and the remaining points in the dataset.

2.1. Background

In this section, some of the standard notions used throughout this paper are introduced, together with some definitions and properties.

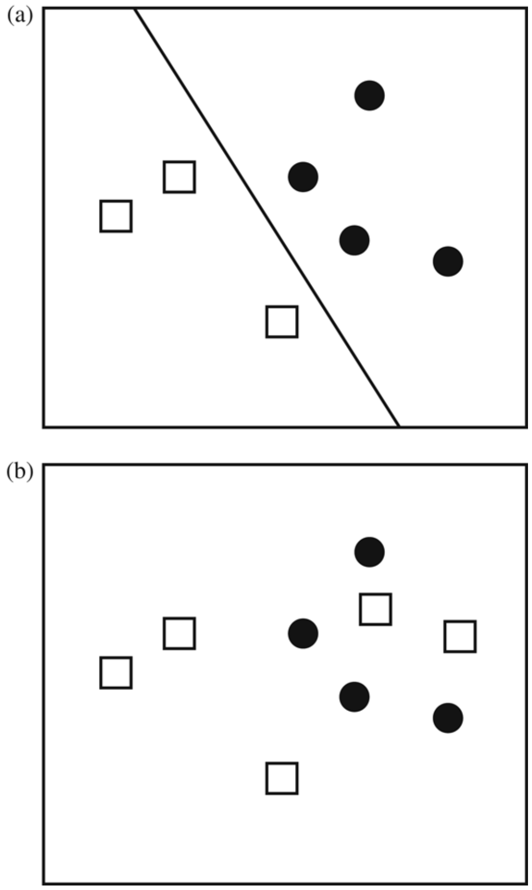
Fig. 4. LS (a) and a non-LS (b) set of points.

2.1.1. Preliminaries

The following standard notions are used: Let $E, F \subset \mathbb{R}^d$,

- $\mathrm{Card}(E)$ stands for the cardinality of a set E. $E \setminus F$ is the set of elements which belong to E and does not belong to F.
- $E \oplus F$ is the set of elements of the form $\vec{e} + \vec{f}$ with $\vec{e} \in E$ and $\vec{f} \in F$.
- $E \ominus F$ stands for $E \oplus -(F)$, i.e. the set of elements of the form $\vec{e} - \vec{f}$ with $\vec{e} \in E$ and $\vec{f} \in F$.
- $\mathrm{Im}(E, G) = \{(x_1, \ldots, x_d, x_{d+1}) \in G \mid (x_1, \ldots, x_d) \in E\}$.

Let $\vec{p}_1, \vec{p}_2$ be the standard position vectors representing two points P_1 and P_2 in $\mathbb{R}^d$,

- The set $\{t\vec{p}_1 + (1-t)\vec{p}_2 \mid 0 \le t \le 1\}$ is called the segment between $\vec{p}_1, \vec{p}_2$ and is denoted by $[\vec{p}_1, \vec{p}_2]$.
- The dot product of two vectors $\vec{u} = (u_1, \ldots, u_d), \vec{v} = (v_1, \ldots, v_d)$ is defined as $\vec{u}\vec{v}^{\mathrm{T}} = u_1v_1 + \cdots + u_dv_d$. $\mathrm{Adj}(\vec{u}, r) = (u_1, \ldots, u_d, r)$ and by extension $\mathrm{Adj}(S, r) = \{\mathrm{Adj}(\vec{x}, r) \mid \vec{x} \in S\}$.
- $\mathcal{P}(\vec{w}, t)$ stands for the hyperplane $\{\vec{x} \in \mathbb{R}^d \mid \vec{w}\vec{x}^{\mathrm{T}} + t = 0\}$ of $\mathbb{R}^d$. $\vec{w}$ is the normal (i.e. is perpendicular), to the hyperplane $\mathbb{P}$. The threshold t is proportional to the distance from the origin to $\mathbb{P}$. $\mathbb{P}$ will stand for the set of all hyperplanes of $\mathbb{R}^d$.

The fact that two subsets X and Y of $\mathbb{R}^d$ are linearly separable is denoted by $X \parallel Y$ or $X \parallel Y(P)$ or $X \parallel_P Y$. Thus if $X \parallel Y(\mathcal{P}(\vec{w}, t))$, then $(\forall \vec{x} \in X, \vec{w}\vec{x}^{\mathrm{T}} + t > 0$ and $\forall \vec{y} \in Y, \vec{w}\vec{y}^{\mathrm{T}} + t < 0)$ or $(\forall \vec{x} \in X, \vec{w}\vec{x}^{\mathrm{T}} + t < 0$ and $\forall \vec{y} \in Y, \vec{w}\vec{y}^{\mathrm{T}} + t > 0)$.

Let $X, Y \subset \mathbb{R}^d$, $\mathbb{P}(X, Y) = \{P \in \mathbb{P} \mid X \parallel Y(P)\}$.

Definition 1. A Recursive Deterministic Perceptron (RDP) P on $\mathbb{R}^d$ is a sequence $[(\vec{w}_0, t_0), \ldots, (\vec{w}_n, t_n)]$ such that $\vec{w}_i \in \mathbb{R}^{d+i}$ and $t_i \in \mathbb{R}$, for $0 \le i \le n$.

$\bullet(\vec{w}_i, t_i)$ for $0 \le i \le n$, is called an Intermediate Neuron (IN) of the RDP P (i.e. an IN of RDP corresponds to an SLPT).

2.2. Methods for testing linear separability

The methods for testing linear separability between two classes can be divided into five groups:

- *The methods based on solving systems of linear equations.* These methods include: the Fourier–Kuhn elimination algorithm, and the Simplex algorithm. The original classification problem is represented as a set of constrained linear equations. If the two classes are LS, the two algorithms provide a solution to these equations.
- *The methods based on computational geometry techniques.* The principal methods include the convex hull algorithm and the class of linear separability method. If two classes are LS, the intersection of the convex hulls of the set of points that represent the two classes is empty. The class of linear separability method consists in characterising the set of points P of $\mathbb{R}^d$ by which it passes a hyperplane that linearly separates two sets of points X and Y.
- *The methods based on neural networks.* The method originally used in the development of the RDP was the perceptron learning algorithm. If the two classes are LS, the perceptron algorithm is guaranteed to converge, after a finite number of steps, and will find a hyperplane that separates them. If the sets are not separable this method will not converge.
- *The methods based on quadratic programming.* These methods can find a hyperplane that linearly separates two classes by solving a quadratic optimisation problem. This is the case for the SVM.
- *The Fisher linear discriminant method.* This method tries to find a linear combination of input variables, $w \times x$, which maximises the average separation of the projections of the points belonging to the two classes C_1 and C_2 while minimising the within class variance of the projections of those points. The resulting hyperplane does not necessarily separates the classes.

These methods are described in detail in Elizondo (2006). Several heuristic methods, to reduce the calculation time while testing for linear separability, are presented in Elizondo (2004a). The original work on the RDP relied on the Perceptron algorithm to find separating hyperplanes, but this method is not guaranteed to produce a result in bounded time. A faster more efficient algorithm for testing linear separability uses the Simplex method. Therefore, this is the selected method for testing linear separability used for this work. The Simplex algorithm has the additional power to confirm that no separating plane exists if that is the case. From the perspective of the RDP, all that is relevant is knowledge of a particular separating hyperplane.

The problem of finding a hyperplane $\mathcal{P}(\vec{w}, t)$ which separates sets X and Y in $\mathbb{R}^d$ [i.e. finding $\vec{w}, t$ where $\vec{w}\vec{x}^{\mathrm{T}} + t > 0$ and $\vec{w}\vec{y}^{\mathrm{T}} + t < 0$ for all $\vec{x} \in X, \vec{y} \in Y$] is equivalent to finding $\vec{w}_1 \in \mathbb{R}^{d+1}$ for which $(\vec{w}_1\vec{s}^{\mathrm{T}} > 0\ \forall \vec{s} \in S)$ where $S = \mathrm{Adj}(X, -1) \cup -\mathrm{Adj}(Y, -1)$. [Given $\vec{w}_1$ which solves the "S problem" in $\mathbb{R}^{d+1}$, the separability hyperplane in $\mathbb{R}^d$ is $\mathcal{P}(\vec{w}, t)$ where $\vec{w}_1 = \mathrm{Adj}(\vec{w}, -t)$. It can be seen that $\vec{w}\vec{x}^{\mathrm{T}} + t > 0$ and $-\vec{w}\vec{y}^{\mathrm{T}} - t > 0 \Rightarrow \vec{w}\vec{y}^{\mathrm{T}} + t < 0$.]

The presented separation algorithm is implemented in $\mathbb{R}^{d+1}$ solving the problem of S. The solution to the original problem for X and Y is then obtained. Since S is finite, the problem of finding $\vec{w}_1$ with $\vec{w}_1\vec{s}^{\mathrm{T}} > 0\ \ \forall \vec{s} \in S$ is equivalent to the problem of finding $\vec{w}_2$ with $\vec{w}_2\vec{s}^{\mathrm{T}} \geq 1\ \ \forall \vec{s} \in S$.

This is a linear programming problem which can be solved by the Simplex method. The set S maps to a set of linear constrains and the calculation of $\vec{w}_2$ reduces to showing that the constraints can all be satisfied simultaneously and choosing one solution. Furthermore, if the constraints cannot be solved simultaneously, then there is no $\vec{w}_2$ and no separating hyperplane to solve the original problem.

The Simplex method is one of the most popular methods used for solving linear programs. A linear program can be seen as a set of variables which are contained in a linear expression called the *objective function*. The goal is to find values to these variables which maximise or minimise the objective function subject to constraints. These constraints linear expressions must be either $\leq$, $\geq$, or $=$ to a given value. There are three possible results when trying to solve a linear program.

1. The model is *solvable*. This means that there exists a set of values for the variables that provide an optimal value to the objective function.
2. The model is *infeasible*. This means that there are no values for the variables which can satisfy all the constraints at the same time.
3. The model is *unbounded*. This means that the value of the objective function can be increased with no limit by choosing values to the variables.

We use in this work the Simplex algorithm for testing linear separability among two classes. The algorithms in Tables 1 and 2 show the Simplex procedure. This algorithm consists in finding the values of q and p to pivot and repeating the process until either an optimum value is obtained, or the linear program is determined to be infeasible. This method can be viewed as a method for organising the procedure so that: a series of combinations of the equations is tried for which the objective function increases (maximisation of the objective function) or decreases (minimisation of the objective function) at each step, and the optimal feasible vector is reached after a number of iterations that is almost always no longer than the order of the number of equations or the number of independent variables, whichever is larger.

We can take, as an example to illustrate the Simplex algorithm, the binary function AND. Let $X = \{(1, 1)\}$ and $Y = \{(0, 0), (1, 0), (0, 1)\}$ represent the input patterns for the two classes, X and Y, which define the AND problem. We want to find a weight vector $\vec{w}$ and a threshold t such that $X \parallel Y(\mathcal{P}(\vec{w}, t))$. We note, from above, $\tilde{X} = \{(1, 1, -1)\}$, $\tilde{Y} = \{(0, 0, -1), (1, 0, -1), (0, 1, -1)\}$, and $S = \{x_0, x_1, x_2, x_3\} = \{(1, 1, -1), (0, 0, 1), (-1, 0, 1), (0, -1, 1)\}$.

Table 1
The simplex algorithm (Source: Sedgewick (1983, chap. 38))

```
**SIMPLEX**($\mathcal{A}$)
– data: an array $\mathcal{A}$ of size **MxN** containing the
constraints linear expressions the pivot column and row
represented by $p$ and $q$ respectively.
– result: a solution if the constraints linear expressions
are solvable, otherwise an *unbounded* or
*infeasible* response.
**Repeat**
  $q := 0$;
  **Repeat**
    $q := q + 1$
  **Until** $(q = M + 1)$ **Or** $(a[0, q] < 0)$;
  $p := 0$;
  **Repeat**
    $p := p + 1$
  **Until** $(p = N + 1)$ **Or** $(a[p, q] > 0)$;
  **For** $i := p + 1$ **To** $N$ **Do**
    **If** $a[i, q] > 0$
    **Then**
      **If** $((a[i, M + 1]/a[i, q]) < (a[p, M + 1]/a[p, q]))$
      **Then**
        $p := i$;
    **If** $(q < M + 1)$ **And** $(p < N + 1)$
    **Then**
      $pivot(p, q)$
**Until**$(q = M + 1)$ **Or** $(p = N + 1)$
```

Table 2
The pivot procedure for the simplex algorithm (Source: Sedgewick (1983, chap. 38))

```
**procedure** pivot(p,q)
– data: the row $p$ and column $q$ of the **MxN** array
containing the constraints linear expressions on which to
perform the pivoting
– result: a $q$ column containing all values equal to zero
except for a 1 in row $q$.
This is obtained by adding multiples of row $p$ to each row
as necessary.
  **Begin**
  **For** $j := 0$ **To** $N$ **Do**
    **For** $k := M + 1$**Downto** 1 **Do**
      **If** $(j <> p)$ **And** $(k <> q)$
      **Then**
          $a[j, k] := a[j, k] - a[p, k] * a[j, q]/a[p, q]$;
  **For** $j := 0$ **To** $N$ **Do**
    **If** $j <> p$
    **Then**
      $a[j, q] := 0$;
  **For** $k := 0$ **To** $M + 1$ **Do**
    **If** $k <> q$
    **Then**
      $a[p, k] := a[p, k]/a[p, q]$;
  $a[p, q] = 1$;
  **End**
```

Thus, to find out if $X \parallel Y$, we need to find a set of values for the weights w_1, w_2, and threshold t such that they simultaneously solve all of the given inequalities.

$$w_1 + w_2 + t \geq 1,$$
$$-t \geq 1,$$
$$-w_1 - t \geq 1.$$
$$-w_2 - t \geq 1.$$

Since the Simplex method limits the values of the variables to being ≥ 0, and a weight value can either be positive or negative, we transform each of our original variables as the difference of two variables. This transformation produces the following variables.

$$\begin{cases} w_1 = w_{11\_aux} - w_{12\_aux}, \\ w_2 = w_{21\_aux} - w_{22\_aux}, \\ t = t_{1\_aux} - t_{2\_aux}. \end{cases}$$

Using the above transformations, our new set of constraints becomes:

$$\begin{cases} (w_{11\_aux} - w_{12\_aux}) + (w_{21\_aux} - w_{22\_aux}) \\ \quad +(t_{1\_aux} - t_{2\_aux}) \geq 1, \\ (-t_{1\_aux} + t_{2\_aux}) \geq 1, \\ (-w_{11\_aux} + w_{12\_aux}) - (t_{1\_aux} + t_{2\_aux}) \geq 1, \\ (-w_{21\_aux} + w_{22\_aux}) - (t_{1\_aux} + t_{2\_aux}) \geq 1. \end{cases}$$

By applying the simplex method, we obtain a feasible solution which gives the following result.

$$w_{11\_aux} = 2, \qquad w_{12\_aux} = 0,$$
$$w_{21\_aux} = 2, \qquad w_{22\_aux} = 0,$$
$$t_{1\_aux} = 0, \qquad t_{2\_aux} = 3.$$

We can thus conclude that the problem is LS. By using these intermediate values we obtain the following values for our original set of variables: $w_1 = 2, w_2 = 2$, and $t = -3$. These variables form the actual hyperplane which separates the two classes.

3. Methods for building RDP neural networks

The three methods for building RDP neural networks are Batch, Incremental and Modular. The Batch method follows a selection strategy based on searching homogeneous LS subsets (i.e., whose elements belong to the same class) from a set of NLS points. With the Incremental approach, all the previous knowledge learned before when adding new knowledge to the RDP does not have to be rediscovered. The modular approach allows the combination of several RDP models, within a single RDP, without having to do any further training.

3.1. The batch method

3.1.1. Description of the method

The Batch method (Table 3) uses an LS subset selection strategy which consists of selecting a set of LS points which belongs to the same class and have maximum cardinality. Essentially, this selection method must utilise some form of exhaustive search explaining the poor complexity measure of the algorithm. This deficiency does not apply to the incremental method discussed later in this section. This set of points is used to compute a hyperplane that separates them from the rest of the points in the dataset. Using this hyperplane, a new intermediate neuron is created and added to the topology. This increases the dimension of the input vector by one. To guarantee convergence, this LS set is marked as used. The process starts again with the remaining set of points until either all points are used, or the problem becomes linearly separable. This algorithm stops after at most $n - 1$ steps, where n corresponds to the number of learning patterns.

Table 3
Batch learning algorithm

```
**Batch** $(X, Y)$
Batch(X, Y)
–data: two disjoint finite subsets $X, Y$ of $\mathbb{R}^d$,
–result: An RDP $P = [(\vec{w}_0, t_0), \ldots, (\vec{w}_{n-1}, t_{n-1})]$
which transforms $X$ and $Y$ into two LS classes. (An RDP linearly
separating $X, Y$, by adding one IN to the RDP constructed by this
algorithm is obtained.
This IN corresponds to the output neuron.)
$i := 0; X_0 := X; Y_0 := Y; X'_0 := X; Y'_0 := Y; S_0 = X \cup Y;$
WHILE $not(X_i \parallel Y_i)$ do
  BEGIN
    SELECT: Select any maximal non-empty subset $Z_i$ from $X'_i$ or from $Y'_i$
      such that $Z_i \parallel (S_i \setminus Z_i)(\mathcal{P}(\vec{w}_i, t_i))$;
      (i.e., $(Z_i \subset X'_i$ or $Z_i \subset Y'_i)$ and $Z_i \parallel (S_i \setminus Z_i)(\mathcal{P}(\vec{w}_i, t_i))$)
    CASE: $Z_i \subset X'_i$
      $S_{i+1} := \mathrm{Adj}(Z_i, -1) \cup \mathrm{Adj}(S_i \setminus Z_i, 1)$;
      $X'_{i+1} := \mathrm{Im}(X'_i, S_{i+1}) \setminus \mathrm{Im}(Z_i, S_{i+1})$;
      $Y'_{i+1} := \mathrm{Im}(Y'_i, S_{i+1})$;
      $X_{i+1} := \mathrm{Im}(X_i, S_{i+1})$;
      $Y_{i+1} := S_{i+1} \setminus X_{i+1}$;
      $i := i + 1$;
    CASE: $Z_i \subset Y'_i$
      $S_{i+1} := \mathrm{Adj}(Z_i, 1) \cup \mathrm{Adj}(S_i \setminus Z_i, -1)$;
      $Y'_{i+1} := \mathrm{Im}(Y'_i, S_{i+1}) \setminus \mathrm{Im}(Z_i, S_{i+1})$;
      $X'_{i+1} := \mathrm{Im}(X'_i, S_{i+1})$;
      $X_{i+1} := \mathrm{Im}(X_i, S_{i+1})$;
      $Y_{i+1} := S_{i+1} \setminus X_{i+1}$;
      $i := i + 1$;
  END;
```

3.1.2. Example

In this subsection, the use of the Batch learning method, presented in Table 3, is illustrated by applying it to an NLS 2D classification toy example. The NLS 2D classification problem consists of two classes A (+), B ($\diamond$) (see Fig. 5).

$$\begin{aligned} A = \{&(3,2), (4,2), (2,3), (3,3), (4,3), (2,4), (3,4), (4,4), \\ &(2,5), (3,5), (4,5), (2,6), (3,6), (4,6), (2,7), (3,7), \\ &(4,7), (5,7), (6,7), (7,7), (4,8), (5,8), (6,8)\}, \end{aligned}$$

$$\begin{aligned} B = \{&(3,0), (4,0), (5,0), (6,0), (3,1), (4,1), (5,1), (6,1), \\ &(6,2), (1,3), (5,3), (6,3), (5,4), (6,4), (5,5), \\ &(6,5), (7,5), (8,5), (5,6), (6,6), (7,6), (8,6), (8,7), \\ &(9,7), (2,8), (7,8), (8,8), (5,9), (6,9), (7,9), (8,9)\}. \end{aligned}$$

After applying the Batch learning method in Table 3 to this problem, an RDP containing seven INS which linearly

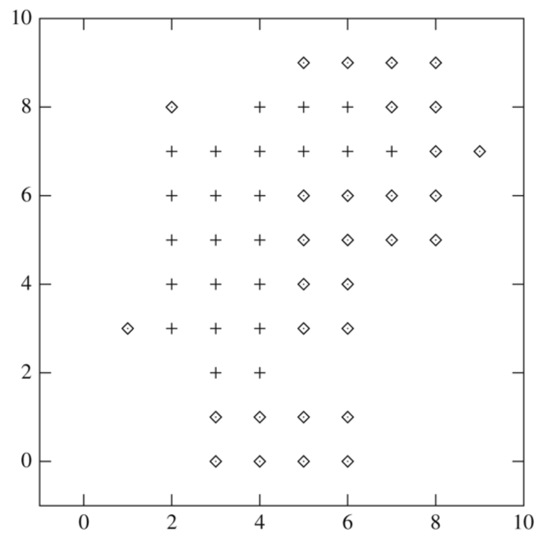
Fig. 5. 2D plot of the two-class classification problem used to illustrate the Batch learning algorithm ($+$ = class A, $\diamond$ = class B).

Table 4
LS subsets selected by executing the Batch learning algorithm described in Table 3 applied to the 2D two-class classification problem

Step #	Selected subset	Class
1	{(3, 0), (4, 0), (5, 0), (6, 0), (4, 1), (5, 1), (6, 1), (5, 3), (6, 3), (6, 4), (6, 5), (7, 5), (8, 5), (7, 6), (8, 6), (9, 7), (8, 7), (8, 8)}	B
2	{(7, 8, 1), (5, 9, 1), (6, 9, 1), (7, 9, 1), (8, 9, 1)}	B
3	{(5, 7, 1, 1), (6, 7, 1, 1), (7, 7, 1, 1), (4, 8, 1, 1)} {(5, 8, 1, 1), (6, 8, 1, 1) }	B
4	{(5, 4, 1, 1, −1), (5, 5, 1, 1, −1), (5, 6, 1, 1, −1), (6, 6, 1, 1, −1) }	A
5	{(3, 2, 1, 1, −1, 1), (4, 2, 1, 1, −1, 1), (3, 3, 1, 1, −1, 1), (4, 3, 1, 1, −1, 1), (3, 4, 1, 1, −1, 1), (4, 4, 1, 1, −1, 1), (3, 5, 1, 1, −1, 1), (4, 5, 1, 1, −1, 1), (3, 6, 1, 1, −1, 1), (4, 6, 1, 1, −1, 1), (3, 7, 1, 1, −1, 1), (4, 7, 1, 1, −1, 1)}	A
6	{(3, 1, 1, 1, −1, 1, −1), (1, 3, 1, 1, −1, 1, −1)}	B

separates A and B is obtained. Table 4 shows the LS subsets selected for each IN (at each step, the selected LS subset was of maximal cardinality). Table 5 shows the weight vectors and thresholds found for each IN. A projection of the selected LS subsets used in the different steps for building the RDP is shown in Fig. 6. Each set shown is a maximal separable set at some stages in the algorithm. Fig. 7 shows the RDP topology found by the learning algorithm described in Table 3 for solving this problem.

3.2. The incremental method

3.2.1. Description of method

In the Incremental or progressive learning, an RDP network is originally trained using only a subset of the training dataset to be correctly classified. The method always guarantees the existence of a subset which can be correctly classified because at least one data vertex in the convex hull of the data will work. The algorithm chooses such a set by inspecting each vertex for suitability and using a suitable vertex which produces the training set with maximum cardinality. Once this network is trained to properly classify this subset, training is continued with the remaining points. Each new point is passed through the existing network. If a new point is not well-classified by the existing RDP, then the new knowledge is captured, without disturbing the previously acquired knowledge, by adding a new intermediate neuron. Training goes on until all the remaining points are classified correctly.

Table 5
RDP weight vectors and threshold values obtained by executing the Batch learning algorithm (Table 3) for each of the INs

i (Step)	$\vec{w}_i$ (Weight vector)	t_i, (Threshold)
1	(−10, 6)	29
2	(−2, −4, −2)	47
3	(2, 4, 6, 8)	−51
4	(−2, 0, −5, −4, 3)	21
5	(14, 2, 48, 43, −34, 26)	−196
6	(2, 2, −9, 0, 0, 0, 0)	0
7	(0.3, 2, −7.85, 0.75, −1.6, −3.55, −0.3)	−6.85

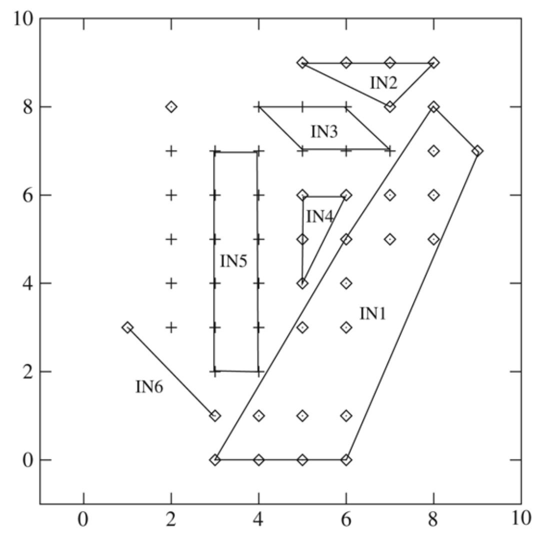

Fig. 6. Projection in a plane of the LS subsets selected by the Batch learning algorithm described in Table 3 to create the INs necessary to construct the RDP. At this point the problem has become separable.

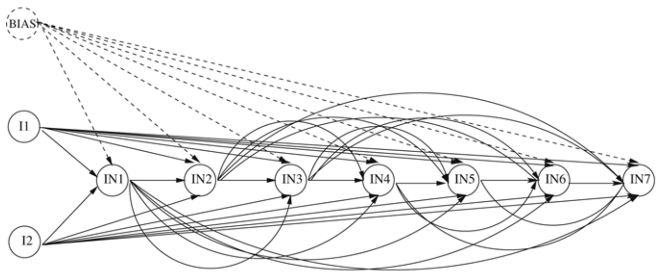

Fig. 7. The topology of the RDP for solving the 2D classification problem obtained by applying the Batch learning algorithm described in Table 3 (IN7 corresponds to the output neuron).

3.2.2. Example

The Incremental learning method is illustrated by applying it to the same problem used to illustrate the Batch learning

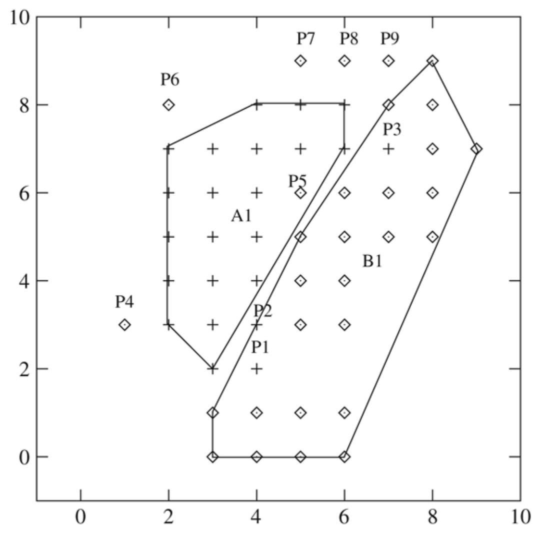

Fig. 8. Cluster subsets A1 and B1 used to build the first RDP and individual points added, one at the time, $p_1, \dots, p_9$ used to illustrate the Incremental learning method.

method (see Fig. 5). In this hand calculation we simply choose appropriate starting subsets by eye. In the actual execution the sets are found by processing each individual point and taking the set which is maximal in the subsequent classification. The choice of this set may affect the classification boundary line for previously unseen data. First a data subset is selected

$$A_1 = \{(3,2), (2,3), (3,3), (2,4), (3,4), (4,4), (2,5), (3,5), (4,5), (2,6), (3,6), (4,6), (2,7), (3,7), (4.7), (5,7), (6,7), (4,8), (5,8), (6,8)\}$$

from set A and

$$B_1 = \{(3,0), (4,0), (5,0), (6,0), (3,1), (4,1), (5,1), (6,1), (6,2), (5.3), (6,3), (5,4), (6,4), (5,5), (6,5), (7,5), (8,5), (6,6), (7,6), (8,6), (8,7), (9,7), (7,8), (8,8), (8,9)\}$$

from B which are linearly separable by the hyperplane $P((-42, 24), -85)$. Therefore, $A_1 \parallel_P B_1$ where $P = [((-42, 24), -85)]$. $A = A_1 \bigcup \{p_1, p_2, p_3\}$ where p_1, p_2 and p_3 correspond respectively to (4, 2), (4, 3), and (7, 7) and $B = B_1 \bigcup \{p_4, p_5, p_6, p_7, p_8, p_9\}$ where p_4, p_5, p_6, p_7, p_8, and p_9 correspond respectively to (1, 3), (5, 6), (2, 8), (5.9), (6, 9), and (7, 9) (Fig. 8). Next, the remaining points $p_1, \dots, p_9$ are "learned" and the RDP containing ten INS shown in Table 6 which separates A and B is obtained.

3.3. The modular method

3.3.1. Description of method

The idea behind modular neural networks is to divide the original problem into smaller subproblems, each of which is to be solved by a subnetwork. These subnetworks are then assembled together into a global network which solves the original problem. The rationale for division into subproblems could be determined by domain knowledge or even be random; the modular RDP combines the RDP subnetworks without loss of previously learned information and with no need for further training. In Section 2, the notation $X \parallel Y(P)$ to indicate that the plane P separated linearly the sets X and Y was introduced. In cases where X and Y are not linearly separable, it was shown in Section 3 that an RDP can be constructed which effectively separates the sets X and Y. The above notation can easily be extended to $X \parallel Y(R)$ to indicate that the RDP R effectively separates X and Y. The following theorem shows how to join the RDP subnetworks.

The RDP networks can clearly be composed so that the output from a set of networks can be passed as the input to another. Since an RDP network essentially just computes a function we use the standard function composition symbol $\circ$ to signify network composition. Hence $Q \circ [P_1, P_2]$ would indicate that Q is a two input RDP composed with RDP networks P_1 and P_2 both of which classify data of the same dimension.

Theorem 1. *Let A_1, A_2, B_1, B_2 be finite subsets of $\mathbb{R}^d$. Let P_1, P_2, P_3, P_4 be RDPs on $\mathbb{R}^d$ such that $A_1 \parallel_{P_1} B_1$, $A_1 \parallel_{P_2} B_2$, $A_2 \parallel_{P_3} B_1$, $A_2 \parallel_{P_4} B_2$*

Let Q, a 4D RDP be defined as

$$Q = [((1, -1, 1, -1), 3), ((3, 5, 2, 5, 5), -3)].$$

Then for the network $P = Q \circ [P_1, P_3, P_2, P_4]$, we have $(A_1 \bigcup A_2) \parallel_P (B_1 \bigcup B_2)$.

The complete proof of the this theorem is given in Tajine and Elizondo (1998). Each RDP P_1, P_2, P_3, and P_4 will produce the outputs described in Table 7 for each of the data subsets:

Let:

$$S_1 = \{(1, -1, 1, -1), (1, -1, 1, 1), (1, 1, 1, -1),$$

Table 6
RDP weight vectors and threshold values obtained by the Incremental learning algorithm for each of the INs

i (Step)	$\vec{w}_i$ (Weight vector)	t_i (Threshold)
1	(−41.9994, 24)	119.997
2	(−83.994, 48, −35.0012)	226.972
3	(−167.988.96, −70.0024, −21.9913)	455.899
4	(335.97, −192, 140.005, 43.98 26, 23.947)	32.095
5	(671.94, −384, 280.01, 87.96 52, 47.894, −919.96)	−551.61
6	(1343.88, −768, 560.02, 175.93, 95.788, −1839.92, −304.161)	480.416
7	(2687.76, −1536, 1120.04, 351.86, 191.576, −3679.84, −608.322, −1887.79)	−94.549
8	(5375.52, −3072, 2240.08, 103.12, 383.152, −7359.68, −1216.54, −3775.58, −832.398)	1154.148
9	(10751, −6144.4480.16, 1407.44, 766.304, −14719.4, −2433.28, −7551.16, −1664.8, −2175.64)	1927.436
10	(−21502, 12288, −8960.32, −2814.88, −1532.61, 29438.8.4866.56, 15102.3, 3329.6, 4351.28, 1796.44)	−2062.32

Table 7
Outputs produced by the RDPs P_1, P_2, P_3, and P_4

		A_1	A_2	B_1	B_2
[p]	P_1	1	± 1	-1	± 1
	P_2	± 1	1	± 1	-1
	P_3	1	± 1	± 1	-1
	P_4	± 1	1	-1	± 1

$$(1, 1, 1, 1), (-1, 1, -1, 1), (-1, 1, 1, 1), (1, 1, -1, 1)\}$$

and

$$S_2 = \{(-1, -1, -1, -1), (-1, -1, 1, -1), (-1, 1, -1, -1), (-1, 1, 1, -1), (-1, -1, -1, 1), (1, -1, -1, -1), (1, -1, -1, 1)\}.$$

Then $S_1 \bigcap S_2 = 0$,

Let $Q = [((1, -1, 1, -1), 3), ((3, 5, 2, 5, 5), -3)]$, then $S_1 \parallel_Q S_2$. Therefore, if $P = Q \circ [P_1, P_2, P_3, P_4]$, then

$$(A_1 \cup A_2) \parallel_P (B_1 \cup B_2).$$

3.3.2. Example

To illustrate the modular construction of an RDP, the same classification problem used to illustrate the Batch and Incremental methods is used. The two original classes are decomposed into the following arbitrarily user selected subclasses:

$$A_1 = \{(3, 2), (4, 2), (2, 3), (3, 3), (4, 3), (2, 4), (3, 4), (4, 4), (2, 5), (3, 5), (4, 5), (2, 6), (3, 6), (4, 6), (2, 7), (3, 7)\},$$

$$A_2 = \{(4, 7), (5, 7), (6, 7), (7, 7), (4, S), (5, 8), (6, 8)\},$$

$$B_1 = \{(3, 0), (4, 0), (5, 0), (6, 0), (3, 1), (4, 1), (5, 1), (6, 1), (5, 2), (6, 2), (1, 3), (5, 3), (6, 3), (5, 4), (6, 4), (5, 5), (6, 5), (7, 5), (8, 5), (5, 6), (6, 6), (7, 6), (8, 6), (8, 7), (9, 7)\},$$

$$B_2 = \{(2, 8), (7, 8), (8, 8), (5, 9), (6, 9), (7, 9), (8, 9)\},$$

as shown in Fig. 9 ($A = A_1 \bigcup A_2$ and $B = B_1 \bigcup B_2$). Next an RDP to linearly separate each subclass from the other subclasses, as shown below, is created:

$$A_1 \parallel_{P_1} B_1, \qquad A_1 \parallel_{P_2} B_2, \qquad A_2 \parallel_{P_3} B_1, \qquad A_2 \parallel_{P_4} B_2$$

where $P_1 = [((-7, -9), -38), ((27, -5, 62), 40)]$, $P_2 = [((0, 10), 76)]$, $P_3 = [((85, -172), -608)]$, and $P_4 = [((-16, 7), -1), ((23, 28, 219), 145)]$. The RDP neural networks can then be created by either using the Batch or the Incremental or a combination of both approaches. In this paper, results are provided with the Modular Batch and the Modular Incremental methods. Once the four RDP modules are created, they can be unified into a single RDP network by using the RDP computed above to linearly separate S_1 and S_2. The final topology of the RDP that combines the four modules for linearly separating classes A and B is shown in Fig. 10. The RDPs P_1, P_2, P_3, P_4 can be constructed in a parallel fashion since their constructions are independent from each other.

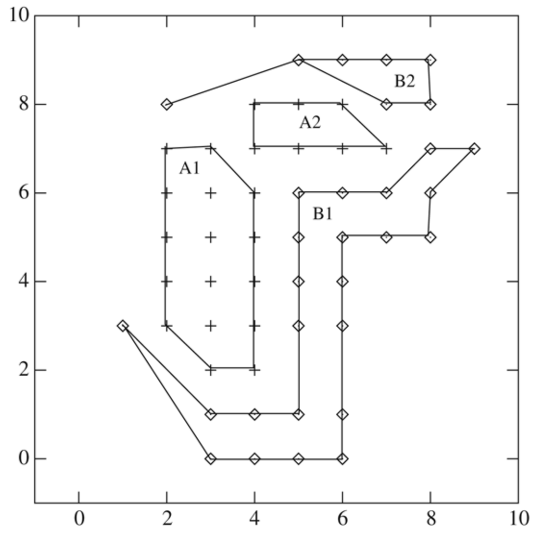

Fig. 9. Clusters of the data subsets A_1, A_2, B_1, and B_2 used to illustrate the modular learning algorithm for constructing RDP neural networks.

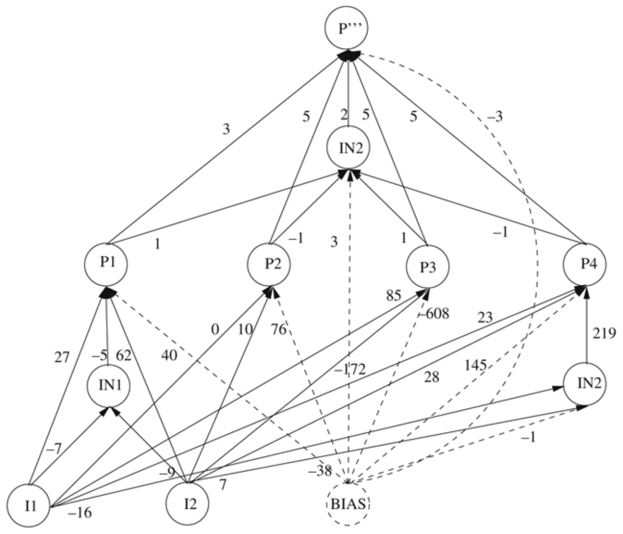

Fig. 10. RDP topology for solving the 2D classification problem obtained by applying the modular learning algorithm.

Table 8
Inputs and outputs used on the IRIS classification problem

Attributes (in cm)	Output	Output classes
Sepal length	Iris plant type	Iris Setosa
Sepal width		Iris Versicolour
Petal length		Iris Virginica
Petal width		

4. Comparison procedure

The three machine learning benchmark datasets used in the comparison study were identified in Section 1.

The IRIS dataset classifies a plant as being an Iris Setosa, Iris Versicolour or Iris Virginica. The dataset describes every iris plant using four input parameters (Table 8). The dataset contains a total of 150 samples with 50 samples for each of the three classes. All the samples of the Iris Setosa class are

Table 9
Inputs and outputs used in the SOYBEAN classification problem

Attributes		Output	Output classes
Date	Leaf-shred	Disease type	Brown-spot
Plant-stand	Stem		Alternaria-leaf-spot
Precipitation	Stem-cankers		Frog-eye-leaf-spot
Temperature	Canker-lesion		
Hail	Fruiting-bodies		
Crop-hist	External decay		
Area-damaged	Fruit-pods		
Severity	Fruit spots		
Seed-tmt	Seed		
Germination	Plant-growth		

Table 10
Inputs and outputs used on the Wisconsin Breast Cancer classification problem

Attributes (1–10)	Output	Output classes
Clump thickness	Class	Benign
Uniformity of cell size		Malignant
Uniformity of cell shape		
Marginal adhesion		
Single epithelial cell size		
Bare nuclei		
Bland chromatin		
Normal nucleoli		
Mitoses		

linearly separable from the rest of the samples (Iris Versicolour and Iris Virginica). Therefore, only the samples belonging to the Iris Versicolour and the Iris Virginica classes were used in this study. Some of the publications that used this benchmark include: Dasarathy (1980), Elizondo (1997), Fisher (1936), Gates (1972).

The SOYBEAN classification problem contains data for the disease diagnosis of the Soybean crop. The dataset describes the different diseases using symptoms. The original dataset contains 19 diseases and 35 attributes. The attribute list was limited to those attributes that had non-trivial values in them (Table 9). Thus there were only 20 out of the 35 attributes that were included in the tests. Since for simplification purposes, this paper uses the two-class RDP as opposed to the *m* class one, only two classes were selected based on the number of samples available and used to build the network. The brown spot, alternaria-leaf-spot and frog-eye-leaf-spot, each having 40 samples were selected and were used. The networks developed were trained to separate the disease class alternaria-leaf-spot from the other two diseases, brown spot and frog-eye-leaf-spot.

The Wisconsin Breast Cancer dataset (Bennett & Mangasarian, 1992; Mangasarian & Wolberg, 1990; Wolberg & Mangasarian, 1990) consists of a binary classification problem to distinguish between benign and malignant breast cancer. The dataset contains 699 instances and 9 attributes (Table 10). The class distribution is: Benign 458 instances (65.5%), and Malignant 241 instances (34.5%).

The technique of cross validation was applied to split the benchmarks into training and testing datasets. The datasets were randomly divided into '*n*' equal sized testing sets that were mutually exclusive (Weiss & Kulikowski, 1991). The remaining samples were used to train the networks. In this study, the classification benchmark datasets were divided into ten equally sized datasets. Sixty percent of the samples were used for training the networks and the remaining forty percent were used for testing purposes. Thus, for each of the three training algorithms, ten different neural networks were developed and tested using different combinations of test sets that were picked up from the equally divided sample sets.

To train the modular networks, each of the cross validation training datasets was further divided into four subsets. The first two subsets, each contained half of the data for class one randomly assigned. The remaining two subsets each contained half of the data for class two, also randomly assigned.

This study was based on the comparison between the convergence time, the level of generalisation, and the topology size with respect to previously unseen data, obtained with each of the three learning algorithms for building RDP neural networks.

Some preliminary results on the comparison of the level of generalisation obtained with the different methods for building RDP neural networks were presented in Elizondo, Birkenhead, and Taillard (2006). The simplex algorithm was used on this study for testing for linear separability. This algorithm was remarkably faster than the Perceptron one when searching for LS subsets. Other algorithms for testing linear separability include the Class of Linear Separability (Elizondo, 2004b) and the Fisher method (see Elizondo (2006) for a survey on methods for testing linear separability).

These results provide a good basis to further develop this study and to compare the topology size (number of intermediate neurons), and convergence time obtained with the three RDP methods. After describing the experimental setup, some conclusions are presented in the next section.

5. Results and discussion

We now give a comparison of the three RDP construction methods based on their time of convergence, their level of generalisation on previously unseen data, and the number of intermediate neurons needed to transform the NLS problem into an LS one. For comparison, results are also presented for the levels of generalisation obtained using the backpropagation and Cascade Correlation algorithms. The IRIS, SOYBEAN and Wisconsin Breast Cancer benchmark datasets were used for constructing neural networks using these methods. Two modular networks were built for each benchmark. The first modular model was trained using the Batch method. The second modular model was trained using the Incremental method. The technique of cross validation was applied to split the benchmarks into training and testing datasets. Some of the training subsets were linearly separable (marked with an asterisk in the tables below). Thus, a single layer perceptron network was used to train them. No incremental or modular networks were developed using these LS subsets. It is likely that the smaller subsets become less complex to learn than the original sets. Therefore, as expected, several of the originally

Table 11
Results obtained with the three learning methods and the three benchmarks in terms of the convergence time

Dataset	Iris				Soybean				Wisconsin breast cancer			
	Batch	Incr	Mod batch	Mod incr	Batch	Incr	Mod batch	Mod incr	Batch	Incr	Mod batch	Mod incr
1	6.7 s	0.2 s	1.4 s	0.2 s	0.0[a]	0.0 s[a]	0.0 s[a]	0.1 s[a]	2.5 h	15.5 s	1.0 h	8.8 s
2	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	15.6 s	0.2 s	0.0 s[a]	0.1 s[a]	2.5 h	17.8 s	1.9 h	8.8 s
3	6.7 s	0.1 s	1.5 s	0.1 s	0.0 s[a]	0.0 s[a]	0.0[a]	0.1 s[a]	2.5 h	16.3 s	1.2 h	7.1 s
4	10.1 s	0.1 s	0.0 s[a]	0.0 s[a]	13.7 s	0.2 s	0.0 s[a]	0.1 s[a]	2.5 h	20.9 s	1.6 h	9.2 s
5	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.1 s[a]	2.5 h	15.9 s	1.8 h	9.0 s
6	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.1 s[a]	2.5 h	16.6 s	1.0 h	8.5 s
7	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	14.1 s	0.2 s	2.7 s	0.2 s	2.6 h	21.0 s	1.2 h	10.8 s
8	10.1 s	0.1 s	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.1 s[a]	2.5 h	20.0 s	1.4 h	10.7 s
9	0.0 s[a]	0.0 s[a]	0.0 s[a]	0.0 s[a]	14.2 s	0.2 s	0.0 s[a]	0.1 s[a]	2.6 h	18.5 s	1.4 h	10.3 s
10	10.1 s	0.1 s	1.4 s	0.1	14.3 s	0.2 s	0.0[a]	0.1 s[a]	2.5 h	18.8 s	1.4 h	10.7 s
Δ[b]	4.3 s	0.1 s	0.5 s	0.0 s	7.2 s	0.1 s	0.3 s	0.1 s	2.5 h	18.1 s	1.4 h	9.4 s

[a] Implies that the two classes on the dataset used for the training of the neural network were linearly separable datasets.
[b] Δ represents the average over the datasets that were not linear separable.

Table 12
Results obtained with the three RDP learning methods, backpropagation, and cascade correlation using the Iris benchmark dataset in terms of the level of generalisation

Dataset	RDP				Back propagation					Cascade correlation				
	Batch	Incr	Mod batch	Mod incr	Mean	Mode	Min	Max	STD	Mean	Mode	Min	Max	STD
1	92.5	92.5	100.0	97.5	95.95	95	95	100	1.65	96.6	95	95	100	2.16
2	95.0[a]	95.0[a]	87.5[a]	87.5[a]	85.8	85	82.5	90	2.18	68.3	67.5	67.5	70	1.25
3	97.5	97.5	95.0	97.5	97.7	100	92.5	100	3.21	98.3	97.5	97.5	100	1.25
4	92.5	90.0	87.5[a]	87.5[a]	85.65	87.5	80	87.5	2.31	85.3	85	75	95	6.05
5	95.0[a]	95.0[a]	95.0[a]	95.0[a]	95.47	95	92.5	97.5	1.26	91.4	92.5	90	92.5	1.32
6	85.0[a]	85.0[a]	90.0[a]	90.0[a]	88.87	87.5	85	92.5	1.68	90.8	87.5	87.5	95	3.06
7	95.0[a]	95.0[a]	92.5[a]	90.0[a]	90.75	90	87.5	92.5	1.48	91.4	92.5	90	92.5	1.32
8	92.5	87.5	97.5[a]	97.5[a]	91.5	92.5	90	95	1.37	89.7	87.5	85	95	3.41
9	95.0[a]	95.0[a]	92.5[a]	90.0[a]	92.67	95	90	95	2.25	91.9	92.5	90	92.5	1.10
10	92.5	92.5	97.5	100	97.6	97.5	97.5	100	0.49	98.3	97.5	97.5	100	1.25
Δ[b]	93.25	92.5	93.5	93.25	92.2	–	–	–	–	90.2	–	–	–	–

[a] Implies that the two classes on the dataset used for the training of the neural network were linearly separable datasets.
[b] Δ represents the average over the datasets that were not linear separable.

NLS training sets became LS after splitting them into smaller subsets to use on training the modular method. This simplified the learning procedure, but had no impact on the validity of the results.

Table 11 shows the convergence time, obtained using the different methods for constructing RDP neural networks. It can be clearly seen that all alternative methods to the Batch give a dramatic improvement in the construction of the RDP. For the Iris dataset, the performance relative to the convergence time, the Incremental method executes 50 times faster, Modular Batch 10 times faster, and the Modular Incremental 30 times. As the size of the dataset increases, the improvement of the incremental methods becomes more apparent, as in the Soybean dataset, the Incremental method executes 65 times faster, the Modular Batch 22 times faster, and the Modular Incremental 60 times. Lastly, the Wisconsin dataset highlights the fact that the NP-complete nature of the Batch learning method (please refer to Tajine and Elizondo (1998) for a formal proof and discussion of the complexities of the three methods) becomes more evident as the size of the datasets increases. In this case the Incremental method executes 506 times faster, the Modular Batch 1.8 times faster, and the Modular Incremental 1000 times. Clearly, as the Modular Batch still inherits the NP-complete nature of the Batch method, this method maintains a variable level of performance with regards to the increments on the dataset size. The incremental method becomes drastically faster as the number of samples grow.

Tables 12–14 show the level of generalisation, in terms of percentage of well-classified samples, obtained using the different methods for constructing RDP neural networks. These results show that both the Batch and the Incremental methods offer comparable performance. The average (Δ) results on the level of generalisation obtained on both methods, using the three benchmarks, only differ by less than 1%. In the case of the Modular Batch and Modular Incremental networks, generalisation levels were slightly higher than those obtained with the Batch or Incremental methods. This is perhaps due to the extra degrees of freedom found on the Modular method.

For comparative purposes, neural networks were also built using the backpropagation and Cascade Correlation algorithms. Ten topologies were developed for the backpropagation networks. Each topology was independently initialised and trained ten times. The first topology had a single hidden unit. The number of hidden units was augmented by one with the last

Table 13
Results obtained with the three RDP learning methods, backpropagation, and cascade correlation using the Soybean benchmark dataset in terms of the level of generalisation

Dataset	RDP				Back propagation					Cascade correlation				
	Batch	Incr	Mod batch	Mod incr	Mean	Mode	Min	Max	STD	Mean	Mode	Min	Max	STD
1	52.1[a]	52.1[a]	52.1[a]	48.0[a]	49.16	47.91	33.33	66.66	7.66	53.54	54.16	52.08	54.16	1.00
2	62.5	79.2	79.1[a]	79.1[a]	83.37	85.41	68.75	89.58	4.16	83.95	85.41	75	91.66	5.97
3	70.8[a]	70.8[a]	72.9[a]	72.9[a]	72.97	75	58.33	85.41	6.38	70.83	68.75	68.75	77.08	2.59
4	83.3	70.8	83.3[a]	85.4[a]	76.93	79.16	62.5	93.75	6.35	75.83	75	70.83	81.25	3.28
5	64.6[a]	64.6[a]	83.3[a]	81.2[a]	76.20	75	68.75	85.41	2.70	73.54	70.83	70.83	81.25	3.26
6	68.7[a]	68.7[a]	77.0[a]	79.0[a]	67.60	68.75	10.41	93.75	13.93	63.75	66.66	60.41	66.66	2.44
7	89.6	79.2	79.1	85.4	76.89	81.25	25	93.75	14.22	78.33	79.16	70.83	83.33	3.28
8	68.7[a]	68.7[a]	72.9[a]	72.9[a]	73.22	72.91	68.75	79.16	2.31	72.91	72.91	72.91	72.91	1.27E − 06
9	83.3	77.1	83.3[a]	83.3[a]	77.29	77.08	60.41	91.66	5.85	76.24	77.08	72.91	81.25	2.23
10	77.1	85.4	85.4[a]	85.4[a]	77.70	75	64.58	89.58	3.91	75.83	75	72.91	81.25	2.97
Δ[b]	72.1	71.6	76.8	77.6	73.14	–	–	–	–	72.48	–	–	–	–

The poor results obtained on the first dataset, with all the methods, are due to artifacts in the data.
[a] Implies that the two classes on the dataset used for the training of the neural network were linearly separable datasets.
[b] Δ represents the average over the datasets that were not linear separable.

Table 14
Results obtained with the three RDP learning methods, backpropagation, and cascade correlation using the Wisconsin Breast Cancer dataset benchmark in terms of the level of generalisation

Dataset	RDP				Back propagation					Cascade correlation				
	Batch	Incr	Mod batch	Mod incr	Mean	Mode	Min	Max	STD	Mean	Mode	Min	Max	STD
1	90.0	93.9	95.4	97.0	96.13	96.97	93.93	98.48	1.12	95.00	95.45	92.42	96.96	1.43
2	97.0	94.2	94.1	95.6	96.63	95.58	94.11	98.52	1.46	93.52	92.64	91.17	95.58	1.58
3	92.7	94.0	97.0	92.5	97.35	97.01	94.03	98.50	1.13	95.52	94.02	94.02	98.50	1.57
4	94.2	97.0	98.5	97.0	96.73	97.05	94.11	100	0.92	95.73	97.05	92.64	98.52	1.76
5	95.7	94.1	95.6	98.5	97.86	98.52	95.58	98.52	0.92	95.88	94.11	94.11	98.52	1.51
6	90.0	90.0	91.4	94.3	90.82	91.42	88.57	92.85	0.81	89.28	88.57	85.71	91.42	1.93
7	97.0	95.5	95.5	94.0	97.08	97.01	94.02	100	0.71	95.22	95.52	92.53	98.50	1.96
8	92.7	94.2	97.1	98.6	98.49	98.55	97.10	100	0.64	95.94	98.55	91.30	98.55	2.71
9	94.2	91.3	94.2	97.1	95.66	95.65	94.20	97.10	0.60	93.33	95.65	86.95	98.55	3.29
10	95.7	92.9	94.3	93.0	95.50	95.77	94.36	97.18	0.95	93.94	94.36	91.54	95.77	1.33
Δ[a]	93.9	93.7	95.3	95.7	96.23	–	–	–	–	94.34	–	–	–	–

[a] Δ represents the average over the datasets that were not linear separable.

topology containing a total of ten hidden units. Both learning rate and momentum were fixed. For the Cascade Correlation, ten neural networks were trained for each of the ten data subsets. The generalisation results are presented in terms of the mean, mode (the most frequently occurring value), min, max and standard deviation obtained for each data subset. No average values are given for the mode, min, max and standard deviation as they are not statistically meaningful in this context.

For the Iris dataset, the maximum of the backpropagation algorithm generalisation results are comparable but slightly better than those attained through the different RDP methods, with respect to the level of generalisation. The performance of the Cascade Correlation algorithm on the Iris dataset on the other hand is slightly poorer compared to the RDP and backpropagation results. Based on the average and the mode results, the RDP seems to obtain slightly better generalisation level compared with both backpropagation and cascade correlation.

For the Soybean dataset, the RDP results for all the construction methods provide less dispersion and in the modular methods appear to offer better generalisation than the backpropagation and Cascade Correlation algorithms. The sixth dataset for backpropagation provides the highest level of generalisation at 93.75%, but with minimum values as low as 10.4%, the overall performance is significantly lower than the Cascade Correlation algorithm. The standard deviation values obtained with the backpropagation model for this datasets are particularly high with an average value of 6.75 and a maximum of 14.22.

In the Wisconsin Breast Cancer dataset, the backpropagation algorithm provides better generalisation performance compared to both the RDP and Cascade Correlation.

Overall, within the RDP construction methods, the Modular Batch and Modular Incremental tend to provide slightly better performance than the Batch and Incremental one, with results that are comparable with those obtained with the backpropagation and the Cascade Correlation algorithms.

Table 15 shows the topology size, in terms of number of intermediate neurons, obtained using the different methods for constructing RDP neural networks.

Table 15
Results obtained with the three learning methods and the Iris benchmark in terms of the number of intermediate neurons needed to transform the original NLS problem into an LS one

Dataset	Iris				Soybean				Wisconsin breast cancer			
	Batch	Incr	Mod batch	Mod incr	Batch	Incr	Mod batch	Mod incr	Batch	Incr	Mod batch	Mod incr
1	1	1	1	2	0[a]	0[a]	0[a]	0[a]	3	9	6	14
2	0[a]	0[a]	0[a]	0[a]	1	2	0[a]	0[a]	3	10	6	14
3	1	1	1	1	0[a]	0[a]	0[a]	0[a]	4	10	6	13
4	2	1	0[a]	0[a]	1	2	0[a]	0[a]	3	12	6	16
5	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	3	10	7	13
6	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	3	9	6	12
7	0[a]	0[a]	0[a]	0[a]	1	2	1	1	3	12	8	18
8	2	1	0[a]	0[a]	0[a]	0[a]	0[a]	0[a]	4	12	8	18
9	0[a]	0[a]	0[a]	0[a]	1	2	0[a]	0[a]	3	9	7	14
10	2	1	1	1	1	2	0[a]	0[a]	3	11	8	18
Δ[b]	0.8	0.5	0.3	0.4	0.5	1	0.1	0.1	3.2	10.4	6.8	15

[a] Implies that the two classes on the dataset used for the training of the neural network were linearly separable datasets.
[b] Δ represents the average over the datasets that were not linear separable.

The differences are not very dramatic in small and relatively simple datasets (Iris and Soybean). In addition, even if differences are observed, they provide a questionable advantage since in some cases the Modular Batch method produces a more efficient topology (less intermediate neurons), while the incremental versions increase the number of intermediate neurons.

Although the topology might be slightly larger, this is compensated by a lower level of complexity given by the incremental and modular methods. This becomes more apparent with larger more difficult datasets, as in the Wisconsin case. While the increase in performance in convergence time of the Incremental method is 506 times, the increase in the topology size is just over 3 times. The same can be said for the Modular Incremental method, where a performance in convergence times shows an increase of 1000 times compared to just an under 6 times increase in the topology.

6. Conclusions

The RDP has been shown to be a useful generalisation of the SLP that has a guaranteed convergence and is capable of solving NLS problems. The initial limitation of this tool was the fact that constructing it implied NP-complete processing. This paper shows the effectiveness of two alternate methods to construct the RDP which reduce dramatically the computing time. The incremental method brings down the complexity of the RDP construction from NP-Complete into $O(n \log n)$ (Elizondo, 1997). The Modular approach breaks down the original problem into several smaller problems that can either be solved by using the Incremental or the Batch methods or a combination of both. This results in a significant reduction in the level of computation required to solve the classification problem. The Incremental method is also best suited for problems of dynamic nature where the network needs to be trained for new data without losing the training already achieved (catastrophic interference). In addition, the modular method provides an additional advantage when parallel processing means are available.With this method the problem can be divided in smaller parts; this not only accelerates the process as a whole, but in addition, enables it to be solved in independent parts. Speed can be increased if the parts are processed concurrently. Although the topology can be larger in some cases, the increase in performance for building the RDP, especially for large datasets, outweighs the increase in the number of intermediate neurons. These results show that the smaller topologies, obtained with the exhaustive Batch method, do not necessarily produce the best results in terms of level of generalisation and that maybe more degrees of liberty are needed to improve the accuracy of the network with respect to previously unseen data. The results obtained are consistent for the three benchmarks and are comparable with the ones shown on recent studies using other learning methods such as SVM (Camastra & Verri, 2005).

Given that the performance difference is not significant in most cases, and better in some others, for the incremental method compared to the Batch method, we can conclude that the former would be the preferred alternative to construct the RDP when a single processor is available. If parallel processing is available, and the Modular approach is used, the modular Batch would provide the best results, but this method would be practical only if the size of the dataset is small. For data set sizes where, due to the NP-complete nature of the problem, the times start to grow beyond the feasibility of the Batch method, the Modular Incremental needs to be used again.

The NP-Completeness of the Batch method, renders it unsuitable for most real world datasets. Thus, either of the Incremental or Modular Incremental methods presented not only enhance, but actually enable the use of the RDP in a much larger range of applications in real world problems.

The results obtained with the RDP are comparable to those obtained with the backpropagation and the Cascade Correlation algorithms. The backpropagation algorithm provides slightly better generalisation results, and the Cascade Correlation gives results which are very close to those obtained with the RDP. These results corroborate the ones presented in Elizondo (1997), where the performance of the RDP was compared with that of other, more classical, methods including

Back Propagation, Cascade Correlation, Rulex, and Ruleneg obtaining comparable results.

7. Future research

The Batch growing method for building RDP neural networks can produce small topologies. This is done by using the approach of maximum cardinality subset on the selection of linearly separable subsets. This approach has been proven to be NP-Complete (Elizondo, 1997). Therefore, heuristic methods can be applied to overcome this complexity problem (Dréo, Pétrowski, Siarry, & Taillard, 2006). For instance, the problem can be simplified by pre-clustering the datasets to obtain a rough solution. This solution can then be refined using a meta heuristic technique. For example, a 'D' point classification strategy can be used for the initial clustering of the data and then a local search strategy like the Tabu Search can be employed to determine the solution to the NP-complete problem.

The implementation of the Incremental method in this study involved the addition of a new intermediate neuron every time a point on the training dataset was not classified correctly by the existing model. The development of an algorithm for adjusting the last hyperplane on the network to classify a new point, before opting to add a new intermediate neuron to the topology, was proposed by Tajine and Elizondo (1998). It would be of interest to see if the number of intermediate neurons on the topology generated by using the Incremental method can be reduced by this method. It will also be of interest to see how the generalisation level is affected by this topology reduction strategy.

It is mentioned in Tajine and Elizondo (1998) that the Incremental algorithm is suited to dynamic classification problems and the Batch method is suited to static classification problems. This needs to be further explored to identify the exact nature of the problems that can be solved by the two methods.

The implementation of the Modular method uses either the Batch or the Incremental method to solve the generated subsets of the original dataset. The use of a combination of the Batch and the Incremental methods to solve subproblems of a single Modular problem can be explored.

To simplify the experiments in this study, the datasets used for the Modular method were randomly split into subsets. It will be interesting to use instead a clustering technique to split the datasets into smaller subsets. The Modular method provides a network of bigger topology whereas at the same time simplifies the problem by breaking it into subproblems and solving them. A heuristic can be devised to identify the number of splits that optimises the compromise between this simplicity of the Modular building algorithm and the topology generated by it. The performance of this method can also be enhanced by using a parallel implementation.

References

Atiya, A. (2005). Learning with kernels: Support vector machines, regularization, optimization, and beyond. *IEEE Transactions on Neural Networks*, *16*(3), 781.

Bennett, K. P., & Mangasarian, O. L. (1992). Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, *1*, 23–34.

Blake, C. L., Newman, D. J., Hettich, S., & Merz, C. J. (1998). *UCI repository of machine learning databases*.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on computational learning theory*.

Camastra, Francesco, & Verri, Alessandro (2005). A novel kernel method for clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(5), 801–805.

Cortes, C., & Vapnik, V. (1995). Support-vector network. *Machine Learning*, *20*, 273–297.

Cristianini, N., & Shawe-Taylor, J. (2003). *An introduction to support vector machines*: *Vol. I*. Cambridge University Press.

Dasarathy, B. W. (1980). Nosing around the neighborhood: A new system structure and classification rule for recognition in partially exposed environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *2*(1), 67–71.

Dréo, J., Pétrowski, A., Siarry, P., & Taillard, E. (2006). *Metaheuristics for hard optimization methods and case studies*. Springer.

Elizondo, D. (1997). The recursive determinist perceptron (RDP) and topology reduction strategies for neural networks. *Ph.D. thesis*. Université Louis Pasteur, Strasbourg, France.

Elizondo, D. (2004a). Searching for linearly separable subsets using the class of linear separability method. In *IEEE-IJCNN* (pp. 955–960).

Elizondo, D. (2004b). Searching for linearly separable subsets using the class of linear separability method. In *Proceedings of the IEEE-IJCNN* (pp. 955–960).

Elizondo, D. (2006). The linear separability problem: Some testing methods. *IEEE Transactions on Neural Networks*, *17*(2), 330–344.

Elizondo, David, Birkenhead, Ralph, & Taillard, Eric (2006). Generalisation and the recursive deterministic perceptron. In *IEEE International joined conference on neural networks*. Vancouver, Canada.

Elizondo, D. A., & Gongora, M. A. (2005). Current trends on knowledge extraction and neural networks. In W. Duch, et al., (Eds.), *Proceedings of the IEEE-ICANN*. Springer.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annual Eugenics*, *7*(II), 179–188.

Gates, G. W. (1972). The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, (May), 431–433.

Johnson, D. S., & Preparata, F. P. (1978). The densest hemisphere problem. *Theoretical Computer Science*, *6*, 93–107.

Mangasarian, O. L., & Wolberg, W. H. (1990). Cancer diagnosis via linear programming. *SIAM News*, *23*(5), 1–18.

Minsky, M., & Papert, S. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

Rosenblatt, F. (1962). *Principles of neurodynamics*. Washington DC: Spartan.

Sedgewick, R. (1983). *Algorithms (p. 508)*. Addison-Wesley Publishing Company.

Tajine, M., & Elizondo, D. (1997). The recursive deterministic perceptron neural network. *Neural Networks*, *11*, 1571–1588.

Tajine, M., & Elizondo, D. (1998). Growing methods for constructing recursive deterministic perceptron neural networks and knowledge extraction. *Artificial Intelligence*, *102*, 295–322.

Tajine, M., Elizondo, D., Fiesler, E., & Korczak, J. (1997). *The international conference on neural networks*. IEEE.

Weiss, S. M., & Kulikowski, C. A. (1991). *Computer systems that learn*. San Mateo, California: Morgan Kaufmann Publishers.

Wolberg, W. H., & Mangasarian, O. L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences*, *87*(December), 9193–9196.