Efficient Perceptron Learning Using Constrained Steepest Descent

Stavros J. Perantonis and Vassilis Virvilis

Institute of Informatics and Telecommunications, National Research Center "Demokritos", 153 10 Agia Paraskevi, Athens, Greece

Requests for reprints should be sent to Dr. Stavros J. Perantonis, Institute of Informatics and Telecommunications, National Research Center "Demokritos", 153 10 Agia Paraskevi, Athens, Greece; E-mail: sper@iit.demokritos.gr

Running Title: Efficient Perceptron Learning

Efficient Perceptron Learning Using Constrained Steepest Descent

Abstract — *An algorithm is proposed for training the single-layered perceptron. The algorithm follows successive steepest descent directions with respect to the perceptron cost function, taking care not to increase the number of misclassified patterns. The problem of finding these directions is stated as a quadratic programming task, to which a fast and effective solution is proposed. The resulting algorithm has no free parameters and therefore no heuristics are involved in its application. It is proved that the algorithm always converges in a finite number of steps. For linearly separable problems, it always finds a hyperplane that completely separates patterns belonging to different categories. Termination of the algorithm without separating all given patterns means that the presented set of patterns is indeed linearly inseparable. Thus the algorithm provides a natural criterion for linear separability. Compared to other state of the art algorithms, the proposed method exhibits substantially improved speed, as demonstrated in a number of demanding benchmark classification tasks.*

Keywords — Perceptron, learning algorithm, quadratic programming, feasible directions.

1 INTRODUCTION

The resurgence of interest in neural network research in the past decade has led to the development of numerous types of architectures and learning algorithms. In particular, feedforward networks have emerged as efficient tools for supervised classification and function approximation tasks. Although recent research has mainly focused on multilayered networks, the single layer perceptron still deserves attention for at least two reasons: Firstly, the design of fast perceptron learning algorithms is important, because such algorithms can form the basis of layer-by-layer learning schemes for multilayer feedforward networks that have received much attention in recent years (Ergenziger & Thompsen, 1995; Hunt & Deller, 1995; Wittner & Denker, 1997). Secondly, many non-linearly separable problems can be cast into linearly separable form by constructing high order polynomial terms of the data. This type of linearization is the main step for constructing high order feed-forward networks that are widely studied and used in many applications.

The most popular stochastic or gradient based algorithms for training the single layered perceptron, e.g. the perceptron learning rule (Rosenblatt, 1962) and the delta rule (Widrow & Hoff, 1988; Rumelhart et al., 1986) can run into serious problems if their parameters (learning rate and momentum) are not chosen correctly. Proper parameter selection usually relies on heuristics. Even if parameters are chosen optimally, in many problems with a highly non-uniform distribution of patterns in the input space learning can be exceptionally slow. This difficulty arises especially in solving non-linear problems which are linearized using high order terms (Telfer & Casasent, 1993). The linearization step not only results in an increase in the input space dimensionality, but also creates a non-homogeneous input space, that many algorithms find difficult to negotiate. Indeed, Volper and Hamson have highlighted this point, by showing that the perceptron rule may need very high order polynomial times even in apparently simple problems with just second-order terms (Volper & Hampson, 1990). The same difficulty arises in the case of layer-by-layer learning in feedforward networks, because in the course of learning many hidden units outputs are forced in the saturation region, so that the space of hidden layer outputs (output layer inputs) quickly becomes highly inhomogeneous. An extra difficulty with layer by layer learning in feedforward network comes from the fact that in intermediate stages of learning the internal pattern representations may not be linearly separable. It is then very important to have a natural termination criterion for the output layer learning algorithm so that inseparability can be detected efficiently (Grossman et al., 1989; Takahashi et al., 1993). This cannot be done using the perceptron rule or the delta rule.

In this paper we propose a novel learning algorithm for a single layer perceptron, which is fast and requires no adjustment of parameters. Learning proceeds by iteratively lowering the value of the perceptron cost function (Barnard & Casasent, 1989; Barnard, 1991) under the constraint that already correctly classified patterns are not to be affected. The perceptron cost function derivative serves as a guide for finding weight vectors with lower cost. The weight vector is found by a line search in the input space, which is terminated when further advancement leads to misclassification of a previously correctly classified pattern. The proper direction for the line search is that of the steepest descent with respect to the perceptron cost function, with additional constraints ensuring that the weight update vector does not intersect hyperplanes corresponding to already correctly classified patterns. The problem of finding the appropriate search direction can be stated as a quadratic programming task, to which a fast and effective solution is proposed.

It is proved in the paper that the resulting algorithm always converges in a finite number of steps. For linearly separable problems, it always finds a hyperplane that completely separates patterns belonging to different categories. In the case of non-linearly separable problems, the algorithm detects the inseparability in a finite number of steps and terminates, having usually found a good separation hyperplane. Thus, it provides a natural criterion for linear separability or inseparability. Experimental results show that the proposed algorithm finds the solution to large scale linearly separable problems much faster than the perceptron rule. Its fast convergence is not hindered by inhomogeneities in the distribution of training patterns. Moreover, it exhibits a decisive learning speed advantage over other algorithms involving no adjustable parameters.

The paper is organized as follows: In section 2, basic terminology is established and background information is introduced concerning characteristics of the perceptron weight space and cost functions. The design and control flow of or algorithm algorithm is discussed in section 3. In section 4 a study of the convergence properties of the algorithm is presented. In section 5, the problem of finding optimal search directions for the implementation of the algorithm is discussed in detail. In section 6 simulation results are presented for various classification problems. Finally, conclusions are drawn in section 7.

2 TERMINOLOGY AND BACKGROUND

2.1 Weight Space

Let us assume that we wish to distinguish between two linearly separable classes of P patterns $\mathbf{x_p}$, each of dimension N, by employing a single layer perceptron with a hard limiter activation function. We want to find a vector $\boldsymbol{W}$ such that

$$\theta(\boldsymbol{W}^T \boldsymbol{X}_p) = T_p, \quad p = 1, \ldots, P \tag{1}$$

where T_p is the target for pattern p, equal to either zero or one, and θ is the step function. The vector $\boldsymbol{X}_p$ consists of the input pattern $\mathbf{x_p}$ of dimension N plus an extra component equal to one and $\boldsymbol{W}$ consists of the weight vector $\mathbf{w}$ augmented by the threshold w_0.

In the $N+1$ dimensional weight space, the vector $\boldsymbol{W}$ is represented by a point. Each of the patterns $\boldsymbol{X}_p$ is represented by a hyperplane which passes through the origin and divides the weight space into 2 subspaces. Hyperplanes corresponding to different patterns $\boldsymbol{d}_p$ segment R^{N+1} into several convex polytopes, whose common boundaries are hyperplane segments. For a given $\boldsymbol{W}$ each pattern hyperplane is classified as Bit Right (BR) if the quantity $O_p = \theta(\boldsymbol{W}^T \boldsymbol{X}_p)$ is equal to T_p, and Bit Wrong (BW) if O_p is not equal to T_p.

A useful observation is that the vector $\boldsymbol{d}_p = (2T_p - 1)\boldsymbol{X}_p$ always points towards the side of the pattern hyperplane that corresponds to the correct classification of pattern $\boldsymbol{X}_p$. Thus, patterns classified as BR are characterized by a positive value of the the quantity $Z_p = \boldsymbol{W}^T \boldsymbol{d}_p$. To see this, suppose first that $\boldsymbol{W}^T \boldsymbol{X}_p > 0$. In this case T_p must be equal to 1 because $\boldsymbol{X}_p$ is classified as BR. Therefore $2T_p - 1 = 1 > 0$ and Z_p is positive. On the other hand, if $\boldsymbol{W}^T \boldsymbol{X}_p < 0$, T_p must be equal to zero. Therefore $2T_p - 1 < 0$ and Z_p is positive. Similarly, we see that patterns classified as BW by $\boldsymbol{W}$ are characterized by a negative value of Z_p. In the sequel, we shall characterize patterns by the vectors $\boldsymbol{d}_p$ instead of the original vectors $\boldsymbol{X}_p$. In terms of these vectors, the original perceptron problem of eqn (1) becomes:

$$\theta(\boldsymbol{W}^T \boldsymbol{d}_p) = 1, \quad p = 1, \ldots, P \tag{2}$$

2.2 Degeneracies

For reasons that will become apparent in subsequent sections, we shall consider non-degenerate cases, in which any $N+1$ equations of the form $\boldsymbol{W}^T \boldsymbol{d}_p = 0$ have only one solution, namely the origin $\boldsymbol{W} = \mathbf{0}$. In other words, the matrix with elements d_{ip} has rank $N+1$. If degeneracies are present in the

original training set, they can be lifted e.g. by adding random numbers of small magnitude to the training vector components (Strang, 1988). Note that even under these conditions, the origin is still common to all pattern hyperplanes. Again, to lift this last degeneracy, various methods can be followed. A popular technique (Bobrowski & Niemiro, 1984) is to solve the problem always keeping w_0 equal to a constant. In the space of the remaining N variables, any $N+1$ pattern hyperplanes have no common points. This method has the disadvantage that the two cases $w_0 > 0$ and $w_0 < 0$ need to be considered separately in two independent learning passes. A second technique that will be adopted here involves solving

$$\theta(\boldsymbol{W}^T \boldsymbol{d}_p - \epsilon_p) = 1, \quad p = 1, 2, \ldots P \tag{3}$$

instead of eqn (2), where ϵ_p are positive random numbers. Any solution of eqn (3) is obviously also a solution of eqn (2). Following this amendment, our remarks about BR and BW patterns have to be modified accordingly:

- If pattern $\boldsymbol{d}_p$ is classified as BR by $\boldsymbol{W}$, then $\boldsymbol{W}^T \boldsymbol{d}_p - \epsilon_p > 0$.
- If pattern $\boldsymbol{d}_p$ is classified as BW by $\boldsymbol{W}$, the $\boldsymbol{W}^T \boldsymbol{d}_p - \epsilon_p < 0$.

2.3 Cost Functions

There are two cost functions related to our problem:

- The perceptron cost function is defined by

$$E = \sum_{p=1}^{P} (T_p - O_p) \boldsymbol{W}^T \boldsymbol{X}_p = - \sum_{p=BW} \boldsymbol{W}^T \boldsymbol{d}_p \tag{4}$$

  where the second sum runs over all patterns that are classified as BW by $\boldsymbol{W}$. According to the last remark in section 2.1, this cost function is positive. It is also piecewise linear in R^{N+1} and has constant gradient $\boldsymbol{\Delta W}$ in each polytope given by

$$-\boldsymbol{\Delta W} = \sum_{p=1}^{P} (T_p - O_p) \boldsymbol{X}_p = - \sum_{p=BW} \boldsymbol{d}_p \tag{5}$$

  Performing gradient descent using this cost function gives the offline (batch) version of Rosenblatt's perceptron learning rule.

- The squared error cost function

$$E_{SE} = \sum_{p=1}^{P} \Big(\theta(\boldsymbol{W}^T \boldsymbol{d}_p) - 1\Big)^2 \tag{6}$$

counts the number of BW for a certain $\boldsymbol{W}$ and obviously takes on a constant value for each polytope.

3 TRAINING STRATEGY

We initialize the training procedure with the weight vector $\boldsymbol{W}$ in the interior of a certain polytope. Our primary aim is to reach the minimum of cost function E_{SE}. Therefore, we devise a strategy that will gradually move $\boldsymbol{W}$ to polytopes of lower E_{SE}. Our strategy involves updating $\boldsymbol{W}$ along successive search directions, each characterized by a vector $\boldsymbol{P}$. To determine these search directions, we shall use information related to the gradient $\boldsymbol{\Delta W}$ of the perceptron cost function E. The plausibility of this dual strategy (using gradient information of one cost function in order to keep decreasing the other) will be established in the next section, where its convergence properties will be studied in detail.

For the first epoch, we choose the vector $\boldsymbol{P} = \boldsymbol{\Delta W}$ and we update $\boldsymbol{W}$ according to:

$$\boldsymbol{W}_{new} = \boldsymbol{W} + n\boldsymbol{P} \tag{7}$$

where n is the learning rate. This is reminiscent of the perceptron learning rule, where n remains constant throughout learning. In our case, the rule of calculation of n is constructed by the requirement to cross over as many BW patterns as possible, without crossing over any BR. In this way, maximum decrease of E_{SE} can be achieved.

Noting that the classification decision for a pattern $\boldsymbol{d}_p$ changes at the point where $\boldsymbol{W}_{new}^T \boldsymbol{d}_p - \epsilon_p = 0$, i.e. $n_p = -(\boldsymbol{W}^T \boldsymbol{d}_p - \epsilon_p)/(\boldsymbol{P}^T \boldsymbol{d}_p)$, we consider the following cases:

1. Suppose that at least one pattern can be found along our search direction which is classified as BR by $\boldsymbol{W}$. Let n_R be the smallest (positive) n_p corresponding to such a BR pattern.

   (a) If BW patterns exist with smaller n_p than n_R, let n_W be the largest among the n_p of these BW patterns. Maximum decrease in E_{SE} will be achieved if n is chosen so that $n_W < n < n_R$. In practice, we always chose $n = (n_W + n_R)/2$. This procedure of moving over

all BW patterns without crossing any BR pattern will be called "Fast Moving". This type of movement is illustrated in Figure 1.

(b) On the other hand, if no BW patterns exist with smaller n_p that n_R, we cannot decrease E_{SE} by following the gradient direction. In this case, the first pattern encountered in the $\boldsymbol{\Delta W}$ direction is a BR pattern. The best we can do is keep E_{SE} constant by moving close to the BR pattern. This pattern is added to an internal "list of active patterns" (i.e., patterns $\boldsymbol{d}_p$ for which the equation $\boldsymbol{W}^T\boldsymbol{d}_p - \epsilon_p = 0$ holds) so that the next move will be parallel to the hyperplane of the specified pattern. This procedure of moving to zero distance from a BR pattern will be called "Moving Near", and is also illustrated in Figure 1.

At this point, note the usefulness of lifting degeneracies, as explained in section 2.2. If more that $N + 1$ pattern hyperplanes were allowed to have common points, our search directions could intersect two or more pattern hyperplanes with the same value of n, rendering problematical the "Moving Near" weight update.

2. If no BR pattern can be found along our search direction, then:

(a) If BW patterns can be found, we only have to cross over all BW patterns to solve the problem (last Fast Moving update). Thus n must be chosen larger than all n_p.

(b) If neither BR nor BW patterns can be found, then $\boldsymbol{P} = \boldsymbol{0}$ and the algorithm terminates.

The first epoch of the algorithm has now been completed. Weight updating in subsequent epochs is performed as follows:

1. If in the previous epoch the weight vector was updated using the "Moving Near" process, the weight vector still resides in the same polytope as before. Obviously, E_{SE} is the same as in the previous epoch. In this case, we update the weight vector using a new search direction, in the hope that it will lead us to a BW pattern hyperplane, so that the "Fast Moving" procedure can be used in the present epoch.

To select the search direction $\boldsymbol{P}$, we use the perceptron cost function E. We wish to update $\boldsymbol{W}$ so that the new vector $\boldsymbol{W}_{new}$ remains in the current polytope and $E(\boldsymbol{W}_{new}) < E(\boldsymbol{W})$. In fact, if we wish E to decrease locally at the fastest possible rate, we can choose the search direction of steepest descent that has common points with the current

polytope (steepest feasible search direction). If there are currently K active patterns, this search direction is the feasible search direction ($\boldsymbol{P}^T \boldsymbol{d}_p \geq 0$) which forms the smallest possible angle ϕ with the gradient $\boldsymbol{\Delta W}$ of E. A method for finding this direction will be studied in detail in section 5. At this point, it suffices to say that in general this search direction will be parallel to some of the pattern hyperplanes, so that after finding it, we have to update the list of active patterns. Once the appropriate search direction, characterized by vector $\boldsymbol{P}$ has been found, $\boldsymbol{W}$ is updated using eqn (7) with n determined as in cases 1 or 2 above.

2. On the other hand, if the previous epoch weight update was performed using the "Fast Moving" procedure, the weight vector now resides in a new polytope of lower E_{SE} than in the previous epoch. We must now use the new E, corresponding to the new polytope to find the steepest feasible search direction. Once again, $\boldsymbol{W}$ is updated according to eqn (7) with n determined as in cases 1 or 2 above.

Termination and convergence properties of the algorithm are discussed in detail in the next section.

We note that our algorithm provides a natural way of performing steepest descent in the perceptron weight space. In this sense it is related to the algorithm proposed by Bobrowski and Niemiro (Bobrowski & Niemiro, 1984; Bobrowski, 1991), whose algorithm is designed to chose steepest polytope edges, rather than steepest feasible directions. We shall refer to this method as the BN method.

4 PROOF OF CONVERGENCE

The purpose of this section is to prove the following statements:

1. The proposed algorithm always terminates in a finite number of steps (epochs).

2. Upon termination, the proposed algorithm correctly classifies all the patterns in linearly separable problems.

An immediate corollary of the above statements is the following: If, upon termination, which always occurs in a finite number of steps, there are still misclassified patterns, the pattern classification problem presented to the network is not linearly separable. Thus, the proposed algorithm can detect linear inseparability in a finite number of steps.

4.1 Finiteness

The following Lemma introduces an ordering of subsequent points reached by the algorithm inside a polytope, in terms of the angles formed by the subsequent search directions and $\boldsymbol{\Delta W}$. This ordering is then used to prove the main theorem.

Lemma 1: Let A and B be points in a certain polytope, which are reached by the algorithm in two subsequent epochs. If ϕ_A and ϕ_B are the angles formed by the steepest feasible directions $\boldsymbol{P_A}$ and $\boldsymbol{P_B}$ at A and B respectively, then $\phi_A < \phi_B$.

Proof: Let us assume, for the sake of the argument, that $\phi_B < \phi_A$ (equality of the angles is not an option, because they would correspond to the same feasible direction at A). Consider the vector $\boldsymbol{P}$ where $\boldsymbol{P} = \boldsymbol{P_A} + \boldsymbol{P_B}$ and let ϕ_C be the angle formed by $\boldsymbol{P}$ and $\boldsymbol{\Delta W}$. Since $\phi_B < \phi_A$, we have $\cos\phi_B > \cos\phi_A$. The following consecutive relations hold:

$$\cos\phi_C = \frac{\boldsymbol{P}^T\boldsymbol{\Delta W}}{|\boldsymbol{P}|\,|\boldsymbol{\Delta W}|} = \frac{\boldsymbol{P_A}^T\boldsymbol{\Delta W} + \boldsymbol{P_B}^T\boldsymbol{\Delta W}}{|\boldsymbol{P_A}+\boldsymbol{P_B}|\,|\boldsymbol{\Delta W}|} =$$
$$\frac{|\boldsymbol{P_A}|\cos\phi_A + |\boldsymbol{P_B}|\cos\phi_B}{|\boldsymbol{P_A}+\boldsymbol{P_B}|} > \frac{(|\boldsymbol{P_A}|+|\boldsymbol{P_B}|)\cos\phi_A}{|\boldsymbol{P_A}+\boldsymbol{P_B}|} > \cos\phi_A \qquad (8)$$

with the last relation following from the triangular inequality $|\boldsymbol{P_A}| + |\boldsymbol{P_B}| > |\boldsymbol{P_A} + \boldsymbol{P_B}|$. Thus, $\phi_C < \phi_A$. Since the polytope is convex and $\boldsymbol{P_A}$, $\boldsymbol{P_B}$ are feasible directions at consecutive points A and B, $\boldsymbol{P}$ is also feasible at

A. From eqn (8) it follows that $\boldsymbol{P}$ is feasible at A and steeper than $\boldsymbol{P_A}$. This contradicts the original hypothesis, according to which $\boldsymbol{P_A}$ is the steepest feasible direction at A. The contradiction originated from the false assumption that $\phi_B < \phi_A$, which means that the statement $\phi_B > \phi_A$ is true.

Theorem 1: The algorithm will always terminate in a finite number of epochs.

Proof: The number of successive polytopes negotiated by the algorithm is at most equal to the initial number of wrong bits, which is obviously bounded by the total number of patterns P. Therefore, to prove the theorem, it suffices to show that a finite number of epochs is spent by the algorithm in a certain polytope. The angle ϕ formed by the steepest feasible direction at a certain point of a polytope and $\boldsymbol{\Delta W}$, is determined by the number of active patterns at this point. Since there are at most P pattern hyperplanes forming the boundary of the polytope, there is a finite number of active constraint combinations and, consequently, a finite number of possible values of ϕ. Moreover, according to Lemma 1, the sequence of successive ϕ is strictly increasing, so that each of the possible values of ϕ can be attained by the algorithm at most once. Thus the number of epochs spent in a polytope is bounded by the finite number of possible values of ϕ.

4.2 Linearly Separable Problems

To prove that the algorithm will always find a solution to linearly separable problems, we must ensure that it can always escape from polytopes corresponding to incorrect classification of some patterns. In particular, it is important to show that it cannot terminate at points corresponding to minima of E in polytopes with non-zero E_{SE}. This is ensured by the important Lemma 3, which states that for linearly separable problems the minimum of the cost function E in a certain polytope $\mathcal{R}$ occurs at points belonging to at least one BW pattern hyperplane. In turn, the idea of the proof in Lemma 3 is to show that for every point in the interior of $\mathcal{R}$, there exists a point with lower E, which lies on the boundary of $\mathcal{R}$ and belongs to a BW pattern, so Lemma 3 is preceded by Lemma 2, which shows how to construct such a point.

Lemma 2: Consider a set of patterns $\{\boldsymbol{d}_p\}$ that are linearly separable, so that the system of inequalities (3) has at least one solution $\boldsymbol{W}_s$. Moreover, consider a weight vector $\boldsymbol{W}$ belonging to the interior of a polytope $\mathcal{R}$ characterized by a positive number of wrong bits. Then, there exists a pattern

$\boldsymbol{d}_B$ belonging to the boundary of $\mathcal{R}$ classified as BW by $\boldsymbol{W}$ and a vector $\boldsymbol{W}_B$ that satisfies

$$\boldsymbol{W}_B^T \boldsymbol{d}_B = \epsilon_B \quad \text{and} \quad \boldsymbol{W}_B = \boldsymbol{W}_s + t_B(\boldsymbol{W} - \boldsymbol{W}_s) \quad \text{with} \quad 0 < t_B < 1 \tag{9}$$

Proof: Let us consider the straight line (t) passing through $\boldsymbol{W}$ and $\boldsymbol{W}_s$, which is parametrized by

$$\boldsymbol{W}_t = \boldsymbol{W}_s + t(\boldsymbol{W} - \boldsymbol{W}_s). \tag{10}$$

Figure 2 illustrates the geometry involved in the simple two-dimensional case. It is essentially required to show that the line segment with end points at $\boldsymbol{W}$ and $\boldsymbol{W}_s$ cannot be intersected by BR hyperplanes. It is only intersected by BW pattern hyperplanes and the pattern $\boldsymbol{d}_B$ mentioned in the Lemma is the BW pattern which is intersected first, as we move from $\boldsymbol{W}$ to $\boldsymbol{W}_s$.

Indeed, given a pattern $\boldsymbol{d}_p$ classified as BW by $\boldsymbol{W}$, its point of intersection with line (t) is characterized by a weight vector $\boldsymbol{W}_p$ satisfying:

$$\boldsymbol{W}_p^T \boldsymbol{d}_p = \epsilon_p = \boldsymbol{W}_s^T \boldsymbol{d}_p + t_p(\boldsymbol{W} - \boldsymbol{W}_s)^T \boldsymbol{d}_p \tag{11}$$

i.e.

$$t_p(\boldsymbol{W}_s - \boldsymbol{W})^T \boldsymbol{d}_p = \boldsymbol{W}_s^T \boldsymbol{d}_p - \epsilon_p \tag{12}$$

Since $\boldsymbol{W}_s$ is a solution of eqn (3), all patterns $\boldsymbol{d}_p$ are classified as BR by $\boldsymbol{W}_s$, and therefore $\boldsymbol{W}_s^T \boldsymbol{d}_p - \epsilon_p > 0$. Since $\boldsymbol{d}_p$ is classified as BW by $\boldsymbol{W}$, then $\boldsymbol{W}^T \boldsymbol{d}_p - \epsilon_p < 0$, so that

$$(\boldsymbol{W}_s - \boldsymbol{W})^T \boldsymbol{d}_p > \boldsymbol{W}_s^T \boldsymbol{d}_p - \epsilon_p > 0 \tag{13}$$

Therefore we can express t_p as

$$t_p = \frac{\boldsymbol{W}_s^T \boldsymbol{d}_p - \epsilon_p}{(\boldsymbol{W}_s - \boldsymbol{W})^T \boldsymbol{d}_p}, \tag{14}$$

and obviously it follows from relation (13) that $0 < t_p < 1$.

On the other hand, if $\boldsymbol{d}_p$ is classified as BR by $\boldsymbol{W}$, then $\boldsymbol{W}^T \boldsymbol{d}_p - \epsilon_p > 0$ and it follows from eqn (14) that

$$t_p \begin{cases} < 0, & \text{if } (\boldsymbol{W}_s - \boldsymbol{W})^T \boldsymbol{d}_p < 0 \\ > 1, & \text{if } (\boldsymbol{W}_s - \boldsymbol{W})^T \boldsymbol{d}_p > 0 \end{cases} \tag{15}$$

Let us now consider the pattern $\boldsymbol{d}_B$ with the largest t_p satisfying $0 < t_p < 1$. According to our analysis, this pattern is classified as BW by $\boldsymbol{W}$

and its intersection $\boldsymbol{W}_B$ with the straight line (ξ) can be written in the form required by relations (9). There remains to prove that $\boldsymbol{W}_B$ lies on the boundary of $\mathcal{R}$. Since $\boldsymbol{W}_B^T \boldsymbol{d}_B = \epsilon_B$, it suffices to prove that all other patterns except $\boldsymbol{d}_B$ are classified by $\boldsymbol{W}_B$ in the same way they are classified by $\boldsymbol{W}$. Given a pattern $\boldsymbol{d}_q \neq \boldsymbol{d}_B$, we have

$$\boldsymbol{W}_B^T \boldsymbol{d}_q = \boldsymbol{W}_s^T \boldsymbol{d}_q + t_B (\boldsymbol{W} - \boldsymbol{W}_s)^T \boldsymbol{d}_q \tag{16}$$

If pattern $\boldsymbol{d}_q$ is classified as BW, then $0 < t_q < t_B$ and $(\boldsymbol{W} - W_s)^T \boldsymbol{d}_q < 0$, so that:

$$\boldsymbol{W}_B^T \boldsymbol{d}_q < \boldsymbol{W}_s^T \boldsymbol{d}_q + t_q (\boldsymbol{W} - \boldsymbol{W}_s)^T \boldsymbol{d}_q \tag{17}$$

Substituting t_q from eqn (14) we find that $\boldsymbol{W}_B^T \boldsymbol{d}_q < \epsilon_q$, so that $\boldsymbol{d}_q$ is classified as BW by $\boldsymbol{W}_B$.

Similarly, if $\boldsymbol{d}_q$ is classified as BR by $\boldsymbol{W}$, we must examine two cases, according to relations (15). Thus, if $(\boldsymbol{W}_s - \boldsymbol{W})^T \boldsymbol{d}_q < 0$, then $t_q < 0 < t_B$. If, on the other hand, $(\boldsymbol{W}_s - \boldsymbol{W})^T \boldsymbol{d}_q > 0$, then $0 < t_B < 1 < t_q$. In both cases, we can use eqn (16) to write:

$$\boldsymbol{W}_B^T \boldsymbol{d}_q > \boldsymbol{W}_s^T \boldsymbol{d}_q + t_q (\boldsymbol{W} - \boldsymbol{W}_s)^T \boldsymbol{d}_q. \tag{18}$$

Substituting from eqn (14) we find that $\boldsymbol{W}_B^T \boldsymbol{d}_q > \epsilon_q$. Thus, in both cases $\boldsymbol{d}_q$ is classified as BR by $\boldsymbol{W}_B$ and the proof is completed.

Lemma 3: Consider a set of linearly separable patterns and a polytope $\mathcal{R}$ corresponding to $E_{SE} \neq 0$. Then, the minimum of E in $\mathcal{R}$ occurs at points belonging to at least one BW pattern hyperplane.

Proof: Given a vector $\boldsymbol{W}$ belonging to the interior of $\mathcal{R}$, we construct the corresponding vector $\boldsymbol{W}_B$ given by Lemma 2. From eqn (9), taking the inner product with any of the vectors $\boldsymbol{d}_p$, we obtain:

$$-\boldsymbol{W}_B^T \boldsymbol{d}_p = (t_B - 1)(\boldsymbol{W}_s^T \boldsymbol{d}_p) + t_B (-\boldsymbol{W}^T \boldsymbol{d}_p) \tag{19}$$

Since $\boldsymbol{W}_s$ is a solution of the linearly separable problem, all patterns are classified as BR by $\boldsymbol{W}_s$ and therefore $\boldsymbol{W}_s^T \boldsymbol{d}_p > 0$. Also, $t_B - 1 < 0$ according to Lemma 2, so that:

$$-\boldsymbol{W}_B^T \boldsymbol{d}_p < t_B (-\boldsymbol{W}^T \boldsymbol{d}_p) \tag{20}$$

Summing over patterns classified as BW by $\boldsymbol{W}$ and using eqn (4) we conclude that:

$$E(\boldsymbol{W}_B) = t_B E(\boldsymbol{W}) < E(\boldsymbol{W}) \tag{21}$$

because $0 < t_B < 1$. From the last equation and from Lemma 2, it follows that for every weight vector $\boldsymbol{W}$ in the interior of $\mathcal{R}$ there exists a vector

with a lower value of E that belongs to the boundary of $\mathcal{R}$ and satisfies $\boldsymbol{W}_B^T \boldsymbol{d}_p = \epsilon_p$ for at least one pattern $\boldsymbol{d}_p$ classified as BW by vectors in $\mathcal{R}$. Obviously, the minimum of E in $\mathcal{R}$ is to be found among all vectors $\boldsymbol{W}_B$, and the proof is completed.

Theorem: Upon termination, the proposed algorithm correctly classifies all the patterns in linearly separable problems.

Proof: Let us consider a polytope corresponding to a positive number of BW patterns. At points of the polytope not corresponding to the minimum of E, there are always available feasible directions and the algorithm cannot terminate, even if it cannot follow a "Fast Moving" trajectory. On the other hand, let us consider a point corresponding to the minimum of E. The algorithm cannot terminate at this point either. Indeed, according to Lemma 3, at least one of the pattern hyperplanes that surround this point corresponds to a BW pattern and the algorithm will follow a "Fast Moving" trajectory passing through the point and continue in another polytope. It follows that the algorithm can only terminate in the polytope where all patterns are correctly classified.

5 FINDING THE SEARCH DIRECTION

5.1 General Remarks

As illustrated in the previous section, our algorithm decomposes the original classification problem into a series of successive subproblems, whereby the steepest feasible search direction must be found. In this section we show that the problem of finding the steepest feasible search direction can be formulated as a quadratic programming problem. An interesting question is whether we can compute these successive feasible directions (a task requiring the successive application of a quadratic programming algorithm) with complexity that outperforms related methods that can be used for training the perceptron. An immediate adversary is the BN method, whereby steepest polytope edges, rather than steepest feasible directions, are used and therefore no quadratic programming task needs to be solved. In this section, we propose a method for solving the quadratic programming task. In section 6 it is demonstrated that using the algorithm proposed in section 3 in conjunction with this method, it is possible to solve classification problems much faster than other well known perceptron training algorithms (including the perceptron rule and the BN algorithm).

5.2 Mathematical Formulation of the Problem

Let us assume that $\boldsymbol{W} \in \Re^{N+1}$ resides at the intersection of $K \leq N+1$ hyperplanes with normal vectors $\boldsymbol{d}_p$. Without loss of generality, we may reorder the patterns, so that the K active patterns are numbered from 1 to K: $p \in \mathcal{N}_k = \{1 \ldots K\},\ K \leq N+1$. We wish to find a direction $\boldsymbol{P}$ which is feasible in the current polytope ($\boldsymbol{P}^T \boldsymbol{d}_p \geq 0$) and forms the smallest possible angle ϕ with the gradient $\boldsymbol{\Delta W}$ of E. Note that in 3 dimensions, this problem has a simple mechanical equivalent, namely the problem of finding the path followed by a particle falling under the influence of gravity starting from the intersection of $K \leq 3$ planes.

To find the steepest feasible direction, it is easy to see that we must solve the following quadratic programming problem:

$$\begin{aligned} \text{Minimize} \quad F &= \frac{1}{2}|\boldsymbol{\Delta W} - \boldsymbol{P}|^2 \\ \text{subject to} \quad \boldsymbol{P}^T \boldsymbol{d}_p &\geq 0, \quad p = 1 \ldots K \end{aligned} \tag{22}$$

Indeed, given any feasible direction, minimum Euclidean distance between a vector along this direction and $\boldsymbol{\Delta W}$ is achieved if $\boldsymbol{P}$ is the projection of $\boldsymbol{\Delta W}$. Therefore, solution of the program (22) should be sought among projections of $\boldsymbol{\Delta W}$. It follows that $\boldsymbol{P}^T \boldsymbol{\Delta W} = |\boldsymbol{P}|^2$. Using this relation, and taking into account that $\cos\phi = \frac{\boldsymbol{\Delta W}^T \boldsymbol{P}}{|\boldsymbol{\Delta W}|\,|\boldsymbol{P}|}$ we obtain $F = \frac{1}{2}|\boldsymbol{\Delta W}|^2 \sin^2\phi$, so that minimization of F also means minimization of ϕ.

The program (22) is a special form of the generic quadratic programming problem, for which various solutions have been proposed in the past (Pang, 1983; Rao, 1984; Bazaraa et al., 1993 and references cited therein). Note that in program (22) the number of constraints is always less or equal to the dimensionality $N+1$ of the input plus bias space and the objective function F is a hyperspherical quadratic form (there are no terms involving products of the form $P_i P_j$ with $i \neq j$). For this type of problem, the new approach proposed in this paper is a feasible directions method based on preliminary (a priori) knowledge about the solution of the problem. Suppose, for the sake of the argument, that we have been able to determine the active constraints ($\boldsymbol{P}^T \boldsymbol{d}_i = 0, i \in \mathcal{M} \subseteq \mathcal{N}_k$) for the solution of the program (22). Let $L \leq K$ be the number of active constraints. Among all vectors belonging to the space S defined by $\boldsymbol{P}^T \boldsymbol{d}_i = 0,\ i \in \mathcal{M}$, the vector whose distance from $\boldsymbol{\Delta W}$ is minimum is the projection of $\boldsymbol{\Delta W}$ upon S. Reordering the patterns, so that the L active constraints are numbered from 1 to L, the projection can be readily obtained using the well known Gramm-Smidt procedure (see, e.g.,

Strang, 1988):

$$\boldsymbol{P} = \boldsymbol{\Delta W} - \sum_{i=1}^{L} \frac{\boldsymbol{\Delta W}^T \boldsymbol{u}_i}{|\boldsymbol{u}_i|^2}\ \boldsymbol{u}_i \text{ with } \boldsymbol{u}_i = \boldsymbol{d}_i - \sum_{j=1}^{i-1} \frac{\boldsymbol{d}_i{}^T \boldsymbol{u}_j}{|\boldsymbol{u}_j|^2}\ \boldsymbol{u}_j \tag{23}$$

The final solution can be found by an exhaustive search among the projections of $\boldsymbol{\Delta W}$ on the spaces S obtained for all possible subsets of $\mathcal{N}_k$. From these projections, non-feasible ones are eliminated. From the rest, the projection that yields the minimum value of F is the final solution. However, this procedure takes exponential time with K and is unacceptable. Still, the fact that $\boldsymbol{P}$ is a projection is useful information, and will play an important role in forming an efficient algorithm for solving program (22).

5.3 Transformation to Orthogonal Constraints

To facilitate solution of the program (22), we firstly transform it to an equivalent problem with mutually orthogonal constraint hyperplanes. The reason for this transformation is that given an initial non-feasible search direction, we can easily form a feasible search direction by simply eliminating non-feasible components.

The K vectors $\boldsymbol{d}_p$ define a subspace in $\Re^K$ where the solution should be searched for. Let us decompose the vectors $\boldsymbol{\Delta W}$ and $\boldsymbol{P}$ into components respectively parallel and perpendicular to $\Re^K$.

$$\boldsymbol{\Delta W} = \boldsymbol{\Delta W}_\perp + \boldsymbol{\Delta W}_\parallel \text{ and } \boldsymbol{P} = \boldsymbol{\Delta W}_\perp + \boldsymbol{Q} \tag{24}$$

The influence of the constraints is limited to the $\boldsymbol{Q}$ part of the final solution $\boldsymbol{P}$. Using the relations $\boldsymbol{\Delta W}_\perp^T \boldsymbol{\Delta W}_\parallel = 0$ and $\boldsymbol{\Delta W}_\perp^T \boldsymbol{Q} = 0$, we can rewrite the first line of eqn (22) as $F = \frac{1}{2}|\boldsymbol{\Delta W}_\parallel - \boldsymbol{Q}|^2$. Note that $\boldsymbol{Q}$ and $\boldsymbol{\Delta W}_\parallel$ are vectors with $N+1$ components but are lying in the $\Re^K$ subspace. It is therefore possible to write them as linear combinations of K linear independent vectors of this subspace.

In the $\Re^K$ subspace, the intersection of any $K-1$ pattern hyperplanes is a straight line. In all, there are K such straight lines. It is convenient to use the K direction vectors $\boldsymbol{v}_p$ of these lines as the (generally orthogonal) basis for the $\Re^K$ subspace. Each of the $\boldsymbol{v}_p$ vectors is the projection of $\boldsymbol{d}_p$ on the intersection of all other normal vectors and is formed following the Gramm-Smidt technique, so that $\frac{\boldsymbol{v}_i{}^T \boldsymbol{d}_j}{\boldsymbol{v}_i{}^T \boldsymbol{v}_i} = \delta_{ij}$. Now we can write $\boldsymbol{Q}$ and $\boldsymbol{\Delta W}_\parallel$ as:

$$\boldsymbol{Q} \ = \ \sum_{i=1}^{K} q_i \frac{\boldsymbol{v}_i}{|\boldsymbol{v}_i|^2},\ q_j = \boldsymbol{P}^T \boldsymbol{d}_j \geq 0 \tag{25}$$

$$\boldsymbol{\Delta W}_{\parallel} \;=\; \sum_{i=1}^{K} a_i \frac{\boldsymbol{v}_i}{|\boldsymbol{v}_i|^2},\; a_j = \boldsymbol{\Delta W}^T \boldsymbol{d}_j \tag{26}$$

In the same spirit, any $N+1$ dimensional vector $\boldsymbol{X}$ belonging to the $\Re^K$ subspace can be transformed to yield a K-dimensional vector $\boldsymbol{x}$ and vice versa, using the relations:

$$\boldsymbol{X} = \mathbf{V}\boldsymbol{x} \Rightarrow \boldsymbol{x} = \mathbf{D}\boldsymbol{X} \tag{27}$$

where $\mathbf{V}$ is a $(N+1) \times K$ matrix with the vector $\frac{\boldsymbol{v}_i}{|\boldsymbol{v}_i|^2}$ in the i-th column and $\mathbf{D}$ is a $K \times (N+1)$ matrix with the vector $\boldsymbol{d}_i$ in the i-th row.

We can now proceed to rewrite F and the constraints using the new vectors:

$$F = \frac{1}{2}(\boldsymbol{a} - \boldsymbol{q})^T \mathbf{R} (\boldsymbol{a} - \boldsymbol{q}), \quad \boldsymbol{q} \geq \mathbf{0} \tag{28}$$

where $\mathbf{R}$ is symmetric and equal to $\mathbf{V}^T\mathbf{V}$.

It follows that the original problem has been transformed from minimizing a hyperspherical quadratic form (22) subject to non orthogonal constraints to minimizing a hyperelliptical quadratic form subject to orthogonal constraints.

5.4 Double Search Technique

In order to solve program (22), we shall employ an iterative algorithm based on minimization of F along successive search directions. Given an initial position vector $\boldsymbol{q}$ in the feasible region and a feasible search direction $\boldsymbol{\Delta q}$, we can find the position where F attains its minimum value in the feasible region along this search direction. The unconstrained minimum resides at $\boldsymbol{q}' = \boldsymbol{q} + \eta_g \boldsymbol{\Delta q}$ where η_g is found by linear minimization across the search direction:

$$\eta_g = -\frac{\nabla {F_q}^T \boldsymbol{\Delta q}}{|\boldsymbol{\Delta Q}|^2},\; \text{where } \boldsymbol{\Delta Q} = \boldsymbol{V}\boldsymbol{\Delta q} \tag{29}$$

and the gradient ∇F_q of F is given by:

$$\nabla F_q = -\mathbf{R}(\boldsymbol{a} - \boldsymbol{q}) = \boldsymbol{V}^T(\boldsymbol{P} - \boldsymbol{\Delta W}) \tag{30}$$

Note that ∇F_q can be calculated in terms of vectors in the original $N+1$ dimensional space.

The position vector characterized by η_g may of course lie outside the feasible region. In this case, we have to take into account the constraints $q_i \geq 0, i \in \mathcal{N}_k$. The position of lowest F along our search direction, that lies

on the boundary of the feasible region, is given by $\boldsymbol{q}' = \boldsymbol{q} + \eta_c \boldsymbol{\Delta q}$, where η_c is the minimum among all positive $\eta_i = -q_i/\Delta q_i$:

$$\eta_c = \min\{\eta_i = -\frac{q_i}{\Delta q_i} : \eta_i > 0 \ \text{ and } \ i \in \mathcal{N}_k\} \tag{31}$$

In all cases, the new point $\boldsymbol{q} + \eta \boldsymbol{\Delta q}$ that yields the lowest value of F is characterized by $\eta = \min\{\eta_c, \eta_g\}$.

Of course, if our initial position $\boldsymbol{q}$ satisfies $q_i = 0$ for some $i \in \mathcal{N}_k$, a given search direction $\boldsymbol{\Delta q^A}$ may not be feasible. However, the orthogonality of the constraints allows us to find a new feasible search direction by starting from $\boldsymbol{\Delta q^A}$ and removing (setting to zero) non-feasible components. Formally, the appropriate search direction is given by a vector $\boldsymbol{\Delta q}$, with components:

$$\Delta q_i = \begin{cases} 0, & \text{if } \ q_i = 0 \ \text{ and } \ \Delta q_i^A < 0 \\ \Delta q_i^A, & \text{otherwise} \end{cases} \tag{32}$$

The proposed algorithm uses two search directions exploiting advantages from both. The first is the *gradient search direction*

$$\boldsymbol{\Delta q^A} = -\nabla F_q \tag{33}$$

which always leads to lower values of the cost function. However, it is well known that always following this direction may lead to zig-zag paths and slow down convergence. The second direction, which we shall call *projection search direction*, points to the projection $\boldsymbol{\Delta W}_{GS}$ of $\boldsymbol{\Delta W}$ on the zero space of the currently active constraints:

$$\boldsymbol{\Delta q}^A = \boldsymbol{D}(\boldsymbol{\Delta W}_{GS} - \boldsymbol{P}) \tag{34}$$

If the active constraints of the solution of program (22) had already been found, this would locate the solution in just one step. However, if only this direction is followed, the algorithm may terminate before the minimum has been found.

To find the solution in a small number of steps, both gradient and projection information must be combined in the same iterative algorithm. In each iteration the proposed Double Search Algorithm tests both gradient and projection search directions, and selects the one leading to the lower final value of F. Iterations of the Double Search algorithm will be called "internal epochs", to distinguish them from the epochs of the main algorithm discussed in section 5. The algorithm avoids zig-zag paths, and our experience shows that it locates the exact solution in a few iterations as demonstrated in the experimental section.

There follows a full description of the algorithm:

Initialization: Set $\boldsymbol{Q} = \mathbf{0}$ (Equivalently $\boldsymbol{q} = \mathbf{0}$). Initialize the list of currently active constraints to contain all $i \in \mathcal{N}_k$.
Internal epoch update: At each epoch of the algorithm follow the following steps:

1. Perform the following operations using the gradient search direction $\boldsymbol{\Delta q^A}$ given by eqn (33):
   (a) Calculate the feasible direction $\boldsymbol{\Delta q}$ using eqn (32).
   (b) Calculate η_g using eqn (29) and η_c using eqn (31). Find $\eta = \min(\eta_c, \eta_g)$.
   (c) Calculate the cost function change ΔF between points $\boldsymbol{q} + \eta\boldsymbol{\Delta q}$ and $\boldsymbol{q}$.
2. Repeat steps a-c above for the projection search direction given by eqn (34).
3. Compare the two resulting ΔF for the gradient and projection search directions. Find the more negative ΔF of the two and note the corresponding value of η.
4. Using the search direction $\boldsymbol{\Delta q}$ that led to the most negative ΔF and the corresponding value of η, update $\boldsymbol{q}$ as $\boldsymbol{q}' = \boldsymbol{q} + \eta\boldsymbol{\Delta q}$.
5. Update the list of currently active constraints.

Termination: The algorithm terminates when no further move is possible, i.e. when $\Delta q_i = 0$ for all i.

In Figure 3 we show the steps followed by the algorithm in a simple two-dimensional problem, illustrating how zig-zag paths are avoided and rapid convergence is achieved.

In short, our algorithm achieves a decrease in error at each iteration and avoids zig-zagging by employing the projection search direction when needed. We have not been able to provide a formal proof of convergence in a finite number of steps. However, in our simulations, the exact minimum, that satisfies the Kuhn-Tucker conditions, has always been found in a finite number of steps in more that 9000 quadratic programming problems whose solution was required in the various benchmarks. Thus, the existence of a rigorous proof is not ruled out and is left for future work.

6 SIMULATIONS

6.1 Classification Benchmarks

The following classification problems are studied:

1. *Linearly separable problem with random distribution of points:* Points are randomly distributed in a N dimensional cube and a randomly chosen hyperplane forms the decision region between two classes. Three problems with different values of P and N are considered.

2. *Elliptical discrimination problem:* In this problem, it is required to discriminate between points lying inside and outside a hyperellipse embedded in M-dimensional space, whose points are characterized by the equation $\sum_{i=1}^{M}(x_i - c_i)^2/a_i^2 = 1$. Obviously, the problem can be made linearly separable, if second order terms in x_i are used. Thus input vectors of dimensionality $N = 2M$ are formed by x_i and $y_i = x_i^2$ $(i = 1, 2, \ldots, M)$. This task is closely related to Casasent type networks (Block, 1988; Telfer & Casasent, 1993). The special case of a hypersphere has been studied by Volper and Hamson who have shown that the perceptron rule requires $O(P^3N^8)$ weight adaptations (Volper & Hampson, 1990). Again, three such problems with different values of P and N are considered.

3. *Sonar target recognition problem:* This is the well known problem of distinguishing between the reflected sonar signals from two kinds of submarine objects: rocks and metal cylinders. We use the original data set studied by Gorman and Sejnowski (1988a, 1988b), that consists of 208 input vectors, each with 60 components. In this problem, Gorman and Sejnowski reported only 85% success for the single layered perceptron, rising to 100% only after introducing 12 hidden units into their network architecture.

4. *Ionospheric data:* This is a task regarding the classification of radar returns from the ionosphere (Sigillito et al., 1989). It consists of 350 input vectors, each with 34 components.

5. *Invariant character recognition using 3rd order correlations:* From a number of character images digitized on a 25x25 pixel screen, 32 features approximately invariant in scaling and rotation are extracted using third order correlations according to the method of Perantonis and Lisboa (1992). Two benchmark tasks are included in the simulations. The first task (OCR1) involves a small set consisting of 160 character

images to be classified in 32 character categories, for which a network with 32 inputs and 32 outputs is used. The data set for this task is linearly separable. The second task (OCR2) involves 240 character images corresponding to 12 letters of the alphabet. Each of the characters is transformed using 3 different scaling and 5 different rotation factors, so that a data set of 240x15=3240 patterns is formed. A network with 32 inputs and 12 outputs is used to discriminate between character classes. The corresponding problem is not entirely linearly separable. In similar experiments described in (Perantonis and Lisboa, 1992) it was reported that single layered perceptrons had difficulties in solving this kind of problem, and consequently a hidden layer of nodes was employed to improve classification accuracy. Here we examine the problem again under the light of our analysis of the perceptron presented in this paper.

6.2 Learning Speed

In order to assess the learning speed of the algorithm proposed in this paper, we compare it with two other algorithms used to train the single layered perceptron with hard limiter activation function. These are the BN method of following polytope edges for lowering the perceptron cost function at each epoch and Rosenblatt's perceptron rule. If the requirement for a hard limiter activation function is relaxed, improved methods based on gradient information can also be used. One of the most successful of these methods is conjugate gradient (CG) (Johansson, Dowla, & Goodman, 1992). For the sake of completeness, we have also used the Polak-Ribiére variant of this method to solve our benchmarks.

Results regarding speed of convergence are presented in Tables 1 and 2. In Table 1, the performance of each algorithm is shown in terms of the total CPU time, with the corresponding number of epochs shown in parentheses. The percentage of correctly classified patterns achieved upon termination of the algorithm is shown in Table 2.

All results are averages over ten trials starting from different initial weights selected from a random uniform distribution between -0.5 and 0.5. The algorithm proposed in this paper and the BN algorithm are executed until normal termination, whereby no further weight update is possible. The termination criterion for the perceptron rule and the CG method is one hour of execution. All simulations were performed using a locally developed neural network simulator (billnet) on a Pentium computer at 133 MHz.

Compared to the perceptron rule, our method exhibits a definitive advantage in terms of learning speed (see Table 1). The advantage is already

apparent in the small scale benchmarks, but becomes more pronounced in medium and large scale problems. Figure 4 shows a typical learning session for our algorithm for the sonar data problem. In Figure 4a, the number of wrong bits is plotted against the number of epochs. The corresponding curve for the perceptron rule is also plotted for comparison. In Figure 4b, the number of active patterns K is plotted against the number of epochs for our algorithm. Note that at the beginning of learning the number of wrong bits drops at a relatively slow rate (in comparison with the perceptron rule). As learning progresses, building up of the active patterns list helps locate more efficient search directions, and convergence is achieved in a few epochs, while the perceptron rule is still far from the solution and its learning curve is almost flat.

Moreover, note that our algorithm has correctly classified all patterns in the linearly separable problems, while the perceptron rule has not been able to separate the patterns in the larger scale synthetic benchmarks (uniform linearly separable and elliptically separable problem) in the allocated time. For the sonar data problem, our algorithm has completely separated the two classes. Thus, the sonar data problem is linearly separable, a fact that, to the best of our knowledge, has not been pointed out in the literature. [1] Note that the perceptron rule required more than 100,000 epochs to separate the patterns for the sonar data problem.

There is also a definitive learning speed advantage over the BN algorithm, as is also evident in Table 1. This advantage originates from two sources:

- Firstly, from the ability of our algorithm to find better search directions, since it is not limited by weight updates performed along polytope edges, but rather finds the true feasible steepest descent direction. In most problems, this allows our algorithm to reach termination in a smaller number of epochs than the BN technique, as is evident in Table 1.

- Secondly, the fact that the BN method follows only edges after the first $N+1$ epochs means that the number of active patterns soon saturates at the value $N+1$. In our case, the number K of active patterns remains lower than $N+1$. A related issue is the behaviour, in terms of complexity, of the double search algorithm employed to solve the problem of determining the direction of steepest descent at each epoch.

[1] While the manuscript of this paper was under review, it came to the authors' attention that Torres Moreno & Gordon (1998) had come to the same conclusion using a different classification method.

In our benchmarks, we observed that the average number of internal epochs required by the double search algorithm was much lower than the current number of active patterns K. This means that the average time spent on determining iteratively the steepest descent direction is less than the time needed to calculate the basis vectors $\boldsymbol{v}_i$, which is of the order NK^3. Therefore, the total time spent by the double search algorithm is less than $2NK^3$. Thus our method has a definitive advantage over the BN technique, which needs time of order N^4 to determine the edges.

Figure 5 displays the maximum and average number of internal epochs required by the double search algorithm, plotted against the number of active patterns for the direction of steepest descent found by the algorithm. Cumulative data from all benchmarks are shown, so that values of K up to 50 dimensions are included. Note the definitely sublinear trend of the plots. The average number of epochs is always much smaller than K (even the maximum number of epochs is lower in most cases). An interesting conclusion drawn from Figure 5 is that the ratio of the average CPU time required by the quadratic search algorithm to the average time spent to calculate the basis vectors $\boldsymbol{v}_i$ is generally small (and decreases with increasing K). Therefore, no significant improvement to the learning speed would be possible if a more efficient algorithm were used to solve eqn (28) instead of the double search method.

Finally, our method has a learning speed advantage over the CG method. Lower average training times were recorded using our method for all problems, with the exception of the P=20000 N=40 ellipse benchmark whereby a slightly larger training time was required by our method. Note, moreover, that the CG algorithm was unable to separate the patterns corresponding to the two real world linearly separable datasets (sonar data and OCR1 problem) in the allocated time limit of one hour of CPU time.

6.3 Detection of Linear Inseparability

The OCR2 and ionospheric data classification problems are not linearly separable, as confirmed by the fact that our algorithm reaches termination with a non-zero number of wrong bits. Of course, our method and the BN method exit upon termination having detected inseparability, while for the perceptron rule and the CG algorithm there is no natural termination criterion. In both linearly inseparable problems, a better solution than that obtained using the perceptron rule is reached in much less CPU time. Indeed, upon termination, our algorithm has correctly classified 99.72% of the patterns in

the OCR2 problem, which compares favorably with 97.47% obtained by the perceptron rule. We note that this level of performance is the same as that obtained by a network with one hidden layer and 20 hidden nodes using the back-propagation rule. Similarly, in the ionospheric data problem our algorithm has correctly classified 96.35% of the data, which is to be compared with 92.22% obtained by the perceptron rule. From Table 2 it is also evident that the performance of our algorithm upon termination is also better than the performance of the BN algorithm, possibly as a result of the better search directions followed by our algorithm during learning. The solution found by the CG algorithm is considerably worse than that of our method in the ionospheric data problem, and slightly better in the OCR2 problem. In short, in linearly inseparable problems our method has the combined advantages of a natural termination criterion for promptly detecting inseparability having at the same time reached good solutions (in terms of wrong bits) upon termination.

6.4 Generalization Ability

We have also conducted experiments concerning the generalization ability of our method and the other two learning algorithms (perceptron rule and BN method) that utilize the same perceptron architecture (step activation function). To assess generalization ability, each dataset was partitioned into a training set consisting of 80 % of the available input vectors and a test set consisting of the remaining 20% of the data. Ten different partitions were chosen at random and for each partition 10 different restarts of the algorithms were performed with different initial weights selected from a random uniform distribution between -0.5 and 0.5. The same termination criteria as before were used. Generalization ability results are given in Table 3 as average percentages of correctly classified bits in the test sets of the resulting 100 training sessions per benchmark and algorithm. In four of the benchmarks our method exhibits better generalization ability than the other methods, while in the remaining six its generalization ability is second best. Hence, we observe that generalization ability is not compromised by the improved speed offered by our method.

7 CONCLUSIONS

In this paper, we have introduced an efficient learning algorithm for the single layered perceptron. The algorithm proceeds by lowering the perceptron cost function following the direction of steepest descent, taking at the same

time care not to increase the number of wrongly classified patterns. In this way the perceptron training task is decomposed in a succession of small scale quadratic programming problems whose solution determines the appropriately constrained direction of steepest descent.

The main contributions of the paper are:

- the proof that this strategy terminates in a finite number of epochs regardless of the nature of the problem (linearly separable or not) and thus provides a natural criterion for linear separability
- the proof that our algorithm always leads to the desired solution in linearly separable classification problems
- the experimental demonstration (in linearly and non-linearly separable classification problems) that by using an efficient quadratic programming algorithm (double search method) for finding the steepest descent directions it is possible to train the single layered perceptron much faster than other perceptron training schemes.

Related issues currently under investigation include:

- a more detailed study of the complexity of our method. Our simulation results are compatible with the hypothesis that the algorithm converges to the solution of linearly separable problems in polynomial time, and it would be very important if this could be proved. Techniques for further lowering the complexity of the double search technique are currently under investigation, as is a comparison with other quadratic programming methods
- the extension of our method to multilayered perceptrons with hard limiter formal neuron activations. In this type of networks it is difficult to implement gradient descent based techniques, because of the non-differentiability of the activation functions. However, our method provides a natural way to perform gradient descent in single layered networks, and the scheme can be extended to two-layered networks using a layer-by-layer optimization methodology. In particular, it is possible to break down the problem of training a multilayered perceptron to perform classification tasks by introducing suitable single layered perceptron cost functions for the output and hidden layers and applying the methodology of this paper to each of these cost functions. In this way, a family of algorithms for training the multilayered perceptron can be developed, which ensure that the number of wrong bits never increases during learning and do not suffer from the slow training speed

of the back propagation algorithm. Results from this line of research will be presented in a forthcoming paper.

REFERENCES

Barnard, E., & Casasent, D. (1989). A comparison between criterion functions for linear classifiers, with an application to neural nets. *IEEE Transactions on Systems, Man, and Cybernetics,* **19**, 1030-1041.

Barnard, E. (1991). Performance and generalization of the classification figure of merit criterion function. *IEEE Transactions on Neural Networks,* **2**, 322-325.

Bazaraa, M. S., Sherali, H. D., & Shetty, C. M. (1993). *Nonlinear Programming Theory and Algorithms.* New York: John Wiley and Sons.

Block, H. D. (1988). The perceptron: A model for brain functioning. *Reviews of Modern Physics*, **34,** 123-135, 1962. Reprinted in J. A. Anderson & E. Rosenfeld (Eds.) *Neurocomputing: Foundations of Research.* Cambridge: MIT Press, 1988.

Bobrowski, L. (1991). Design of piecewise linear classifiers from formal neurons by a basis exchange technique. *Pattern Recognition,* **24**, 863-870.

Bobrowski, L., & Niemiro, W. (1984). A method of synthesis of linear discriminant function in the case of nonseparability. *Pattern Recognition,* **17**, 205-210.

Ergenziger, S., & Thompsen, E. (1995). An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer. *IEEE Transactions on Neural Networks,* **6**(1), 31-42.

Gorman, R. P., & Sejnowski, T. J. (1988a). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks,* **1**, 75-89.

Gorman, R. P., & Sejnowski, T. J. (1988b). Learned classification of sonar targets using a massively parallel network. *IEEE Transactions on Acoustics, Speech and Signal Processing,* **36**, 1135-1140.

Grossman, T., Meir, R., & Domany, E. (1989). Learning by choice of internal representations. *Advances in Neural Information Processing Systems,* **1**, 73-80.

Hunt, S. D., & Deller, J. R. (1995). Selective training of feedforward artificial neural networks using matrix perturbation theory. *Neural Networks*, **8**, 931-944.

Johansson, E. M., Dowla, F. U., & Goodman, D. M. (1992). Backpropagation learning for multilayer feedforward networks using the conjugate gradient method. *International Journal of Neural Systems,* **2**(4), 291-301.

Pang, J.-S. (1983). Methods for quadratic programming: a survey. *Computers and Chemical Engineering,* **7**, 583-594.

Perantonis, S. J., & Lisboa, P. J. G. (1992). Invariant pattern recognition using higher-order networks and moment classifiers. *IEEE Transactions on Neural Networks,* **3**(2), 241-251.

Rao, S. S. (1984). *Optimization Theory and Applications*. Wiley Eastern.
Rosenblatt, F. (1962). *Principles of Neurodynamics*. New York: Spartan.
Rumelhart, D. E., Hinton J. E., & Williams, R. J. (1986).Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McLelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, 1, *Foundations*. Cambridge, MA: MIT Press, 318-362.
Strang, G. (1988). *Linear Algebra and its Applications*. Harcourt Brace Jovanovich.
Takahashi, H., Tomita, E., & Kawabata, T. (1993). Separability of internal representations in multilayer perceptrons with application to learning. *Neural Networks*, **6**, 689-703.
Telfer, B. A., & Casasent, D. P. (1993). Minimum-cost associative processor for piecewise-hyperspherical classification. *Neural Networks,* **6**, 1117-1130.
Torres Moreno, J. M. & Gordon, M. B. (1998). Characterization of the sonar signals benchmark. *Neural Processing Letters,* **7**, 1-4.
Sigillito, V. G., Wing, S. P., Hutton, L. V., & Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. Johns Hopkins APL Technical Digest, **10**, 262-266.
Volper, D. J., & Hampson, S.E. (1990). Quadratic function nodes: Use, structure and training. *Neural Networks,* **3**, 93-107.
Widrow, B., & Hoff, M. E. (1988). Adaptive switching circuits. In *1960 IRE WESCON Convention Record,* New York, 1960, vol. 4, 96-104. Reprinted in J. A. Anderson & E. Rosenfeld (Eds.), *Neurocomputing: Foundations of Research.* Cambridge: MIT Press, 1988.
Wittner, B. S., & Denker, J. S. (1997). Strategies for teaching layered networks classification tasks. In *Neural Information Processing Systems* (pp. 850-859) Denver.

Table Captions

Table 1 : Average CPU time (in secs) and number of epochs (given in parentheses) required by the proposed method, the BN algorithm, the perceptron rule and the CG algorithm for various classification problems. All problem data sets but the last two (ionospheric data and OCR2) are linearly separable.
Table 2 : Average classification performance (percentage of correctly classified bits) achieved by the proposed algorithm, the BN method, the perceptron rule and the CG algorithm for various classification problems.
Table 3 : Average generalization ability (percentage of correctly classified bits in the test set) achieved by the proposed algorithm, the BN method and the perceptron rule.

Figure Captions

Figure 1: An arrangement of hyperplanes (lines) in 2-dimensional space is shown, corresponding to a classification problem. In each polytope the number of BW is displayed. Starting from point I, our algorithm finds the solution in one step following a "Fast Moving" weight update. If started from point II, the algorithm finds the solution in two steps, performing a "Moving Near" update up to point IIa followed by a "Fast Moving" update.

Figure 2: Geometry involved in the proof of Lemma 2. The line segment with end-points at the current weight vector $\boldsymbol{W}$ and the solution weight vector $\boldsymbol{W}_s$ intersects the current polytope at a BW hyperplane.

Figure 3: Solution of a quadratic programming problem with mutually orthogonal constraint hyperplanes using the double search method. This method finds the minimum at point S in two steps (solid curve), while gradient descent displays an oscillatory behaviour (dotted curve).

Figure 4: (a) Number of wrong bits versus number of epochs for our method (solid curve) and the perceptron rule (dotted curve) in the sonar data classification problem. (b) Number of active patterns versus number of epochs for our algorithm.

Figure 5: Plot of the number of internal epochs required for convergence of the double search algorithm as a function of the number of active patterns. Both the maximum (dotted curve) and average number of epochs (solid curve) are shown.

Table 1

	Proposed	BN	Perceptron	CG
Uniform				
P=100 N=2	0.006 (4.6)	0.015 (6.5)	0.435 (581)	0.326 (12.6)
P=1000 N=4	0.243(16.4)	0.527 (32.0)	1.379 (157)	4.466 (18.5)
P=20000 N=10	33.0 (76.9)	108.6 (205)	3600 (14439)	184.1 (208)
Ellipse				
P=100 N=4	0.023 (13.4)	0.031 (19.5)	0.059 (63.1)	0.227 (9.0)
P=10000 N=20	74.3 (214.0)	259.4 (567)	2661.4 (14749)	328.2 (104.5)
P=20000 N=40	647.6 (488.7)	4554 (2162)	3600 (6082)	582.9 (77.8)
Sonar	117.8 (223.8)	405.9 (190.5)	1174.9 (146047)	3600 (123549)
OCR1	4.75 (16.0)	5.65 (17.9)	42.38 (94.4)	3600 (2855)
Ionosphere	22.1 (155.5)	31.8 (142.8)	3600 (430735)	3600 (101658)
OCR2	308.3 (294.4)	1337.7 (533.9)	3600 (4554)	3600 (589.3)

Table 2

	Proposed	BN	Perceptron	CG
Uniform				
P=100 N=2	100	100	100	100
P=1000 N=4	100	100	100	100
P=20000 N=10	100	100	99.91	100
Ellipse				
P=100 N=4	100	100	100	100
P=10000 N=20	100	100	99.90	100
P=20000 N=40	100	100	99.72	100
Sonar	100	100	100	94.13
OCR1	100	100	100	97.51
Ionosphere	96.35	94.87	92.22	97.07
OCR2	99.72	99.64	97.47	94.36

Table 3

	Proposed	BN	Perceptron
Uniform			
P=100 N=2	97.13	96.93	96.27
P=1000 N=4	99.90	99.94	99.89
P=20000 N=10	99.96	99.95	99.85
Ellipse			
P=100 N=4	96.93	96.53	97.60
P=10000 N=20	99.82	99.85	99.80
P=20000 N=40	99.77	87.47	99.57
Sonar	75.00	74.03	76.61
OCR1	98.40	97.99	99.44
Ionosphere	86.77	86.87	85.75
OCR2	99.20	99.11	98.94

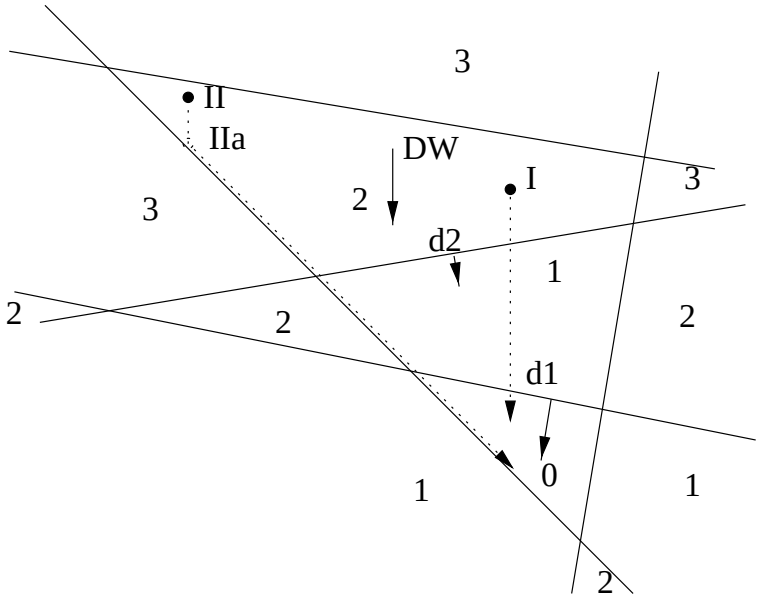

Figure 1:

Table 1:

Table 2:

Table 3:

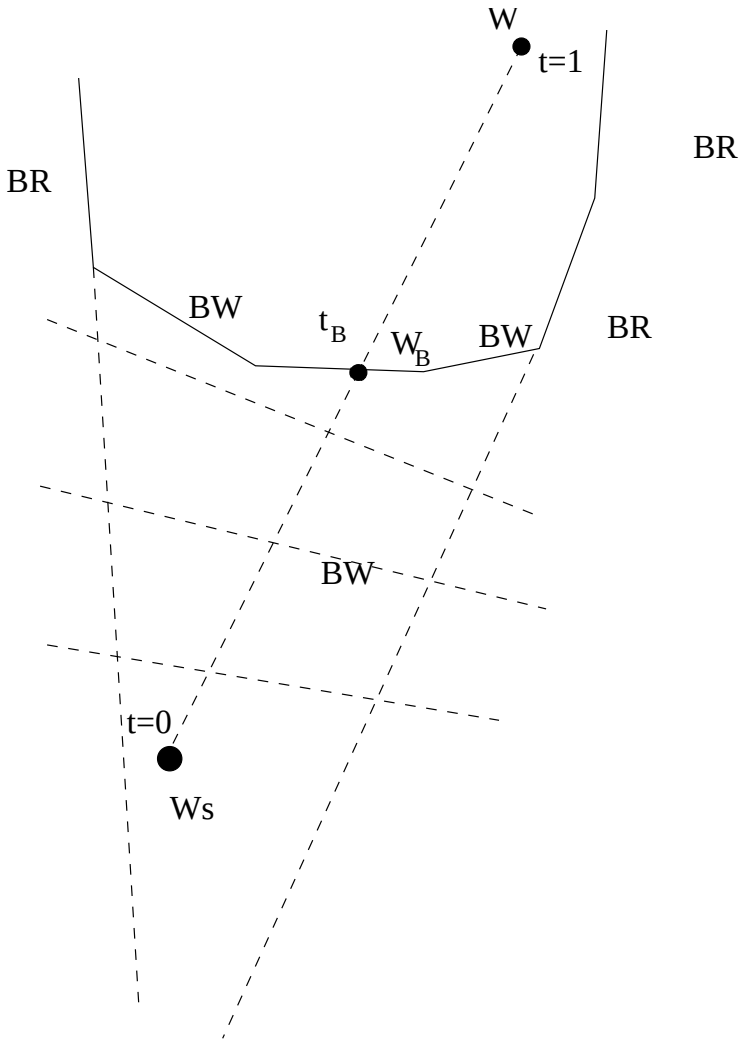

Figure 2:

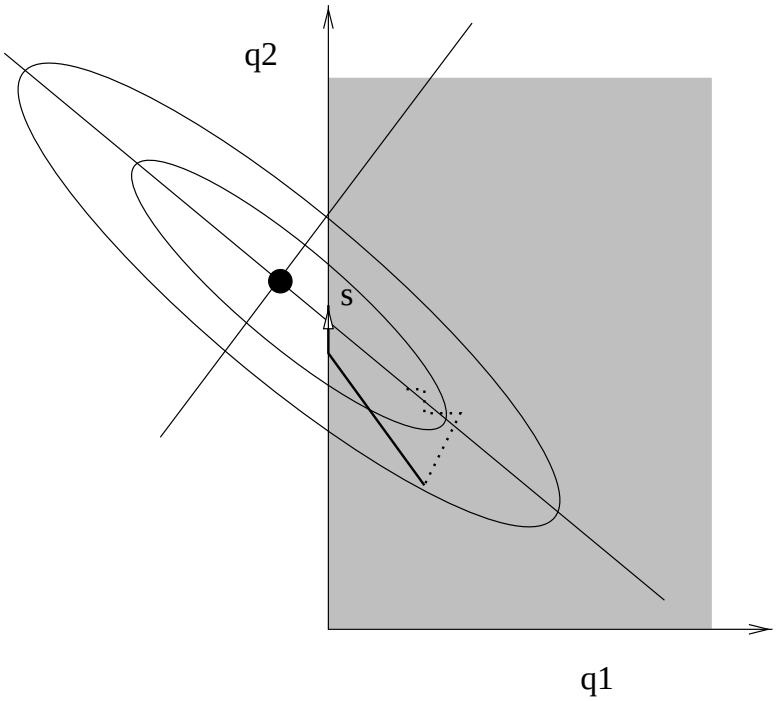

Figure 3:

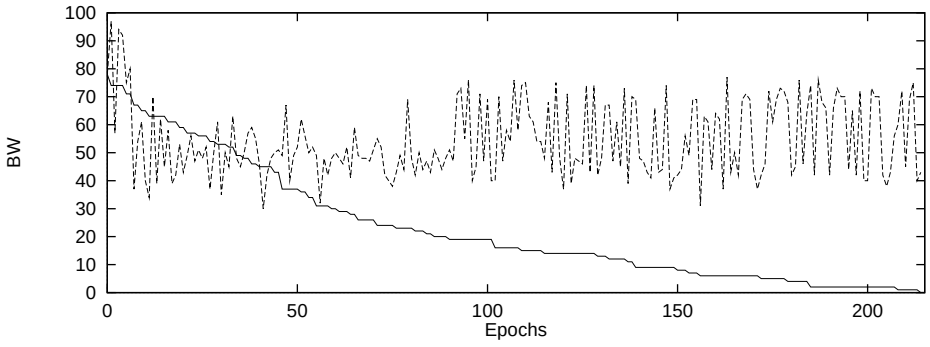

Figure 4:

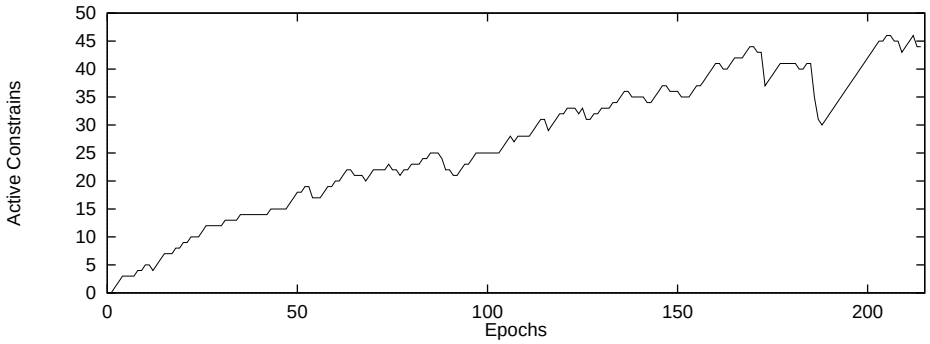
Active Constrains
Epochs
0
5
10
15
20
25
30
35
40
45
50
0
50
100
150
200

Figure 5: