A Fast Method for Calculating the Perceptron with Maximal Stability

Pál Ruján[*]

Fachbereich 8 Physik, Postfach 2503

Carl-von-Ossietzky Universität

D-26111 Oldenburg

In Memoriam R. Rammal

Abstract

For the class of linearly separable two class (boolean) functions the Perceptron with maximal stability defines in the space of all possible input configurations the direction along which the distance between the two classes is minimal. This solution has several advantages: it is unique, it is robust, and has the best generalization probability among all known linear discriminants. I present here an active set approach to the dual problem, finding the minimal connector between two disjoint convex hulls. If N is the number of the input units and M is the number of examples, this algorithm runs in $O(MN^2)$ steps and requires the storage of a symmetric $(N+3) \times (N+3)$ matrix.

1 Introduction

R. Rammal was a physicist with many faces. He was interested in *problems*, which he solved with whatever methods he found useful, analytical or numerical. For example, he was not afraid to learn from computer scientists how to compute effectively the ground state of two dimensional spin glasses. This paper is dedicated to his memory: although all statements made below are subsequently proved, this is by no means a mathematical paper. My goal was to give a simple 'picture' of why, what, and how the things have been done.

Many pattern recognition algorithms use linear discriminant surfaces. This is also typically the case for neural networks, where most learning algorithms

[*]as published in J. de Phys. I France **3** (1993) 277-290, with minor changes

[1, 2, 3, 4] are based on the Perceptron learning rule[5] or its variants [6, 7, 8]. Real world problems would require networks with more hundreds of input variables, trained on databases consisting of several thousands of examples. In such situations, as expected from algorithmic complexity results [9, 10], the bottleneck is the long learning time. Even for linearly separable problems, maximizing the generalization probability of the network, either by queries [11, 12], or by the Bayes criterion [13], involves rerunning repeatedly the Perceptron algorithm.

Hence, it was of immediate practical interest to address the problem of how to implement a reliable, fast, and economic Perceptron learning rule. As a result, an algorithm for determining the Perceptron with maximal stability (PMS) is presented in this paper. Among the linear discriminant learning rules the PMS has several distinguished features[14]:

- Uniqueness: if the problem is linearly separable, there is only one PMS solution;
- Robustness: by construction, the PMS solution is particularly stable against threshold fluctuations;
- Generalization probability: among all types of single unit learning rules it has the best generalization ability [15]. Note that the optimal Bayes-rule requires *many* Perceptrons [13].

The PMS problem can be viewed primarily as the search of a direction in the space of all possible input configurations such that the gap between two parallel hyperplanes separating the set of positive from the set of negative examples is maximal. This geometric view was – to my knowledge – introduced first by Lambert[16] under the pictorial name 'maximal dead zone' and has been independently used also by the Russian school [17]. In the physics literature a slightly different formulation has been introduced by Gardner [18], here one maximizes the distance between the origin and the closest point in the *union* of the negative and positive example sets. The dual[1] problem of the PMS is to minimize the distance between the convex hull of the positive and negative examples, respectively, and is called the *minimal connector* problem. Mathematically, the PMS problem consists thus of minimizing a quadratic function subject to linear constraints — a quadratic programming problem. As shown in [28], this problem can be solved in polynomial time as a function of problem size and accuracy.

An iterative algorithm for the PMS has been given by Krauth and Mézard (the minimal overlap or 'MinOver' algorithm)[19] and has been later improved through a combination of the Adaline and Perceptron algorithm (called 'AdaTron') by Anlauf and Biehl[20]. Both are constrained gradient descent approaches and rather simple to implement. For reasonable speed, however, one needs to store the correlation (overlap) matrix, whose size is $M \times M$, where M is the number

[1]Dual in mathematical programming sense.

of examples. Even then, the convergence is rather slow, especially for large or almost not linear separable problems.

On the other hand, the quadratic programming problem is the basic routine for solving nonlinear equations and has accordingly received a lot of attention in numerical analysis. For example, the IMSL 10.0 version contains the QPROG routine, based on the dual method of Goldfarb and Idnani[24]. The (dis)advantage of canned routines is that usually they are rather general and it is not easy to tailor them to specific needs.

The approach introduced here uses besides the main ideas of Goldfarb and Idnani the additional symmetries of the Perceptron problem. The algorithm is in many ways equivalent to a matrix inversion. It runs in $O(MN^2)$ steps and requires storing a symmetric $N \times N$ matrix. The paper is organized as follows: in Section 2 the primal and its dual problem are formulated using a geometrical interpretation. The active set strategy and the proof for convergence is given in Section 3. The actual implementation of the algorithm is explained in more detail in Section 4. Section 5 concludes the paper by presenting numerical results, comparisons with other methods, and remarks on numerical stability.

2 The maximal dead zone and the minimal connector

Consider a class of boolean functions defined on N variables (inputs) $f \in \mathcal{F}$, $f(s_1, s_2, \ldots, s_N) = \pm 1$. The inputs might be binary, $s_i = \pm 1$, integer, or real values lying on a compact interval. Only the class of linearly separable functions is considered here, $\mathcal{F} = \mathcal{LS}$:

$$f(\vec{s}) = f(\{\vec{w}, \delta\}) = \mathrm{sgn}\,[(\vec{w}, \vec{s}) - \delta] \tag{1}$$

Here $\vec{s} = (s_1, s_2, \ldots, s_N)$ is the input vector, $\vec{w} = (w_1, w_2, \ldots, w_N)$ is the weight vector, and $(\vec{w}, \vec{s})$ denotes their scalar (dot) product. A boolean function is linearly separable if the hyperplane

$$(\vec{w}, \vec{s}) = \delta \tag{2}$$

separates the input configuration regions where $f = 1$ and $f = -1$, that is

$$\begin{array}{rcll} (\vec{w}, \vec{s}) & \geq & \alpha & \text{if } f(\vec{s}) = +1 \\ (\vec{w}, \vec{s}) & \leq & \beta & \text{if } f(\vec{s}) = -1 \\ \alpha & > & \beta & \end{array} \tag{3}$$

The corresponding network is a *Perceptron* and $\vec{w}$ is the vector whose element w_i is the weight connecting input unit i to the output unit. Note that $\vec{w}$ is normal to the separating hyperplane, its length is denoted by $W = \sqrt{(\vec{w}, \vec{w})}$.

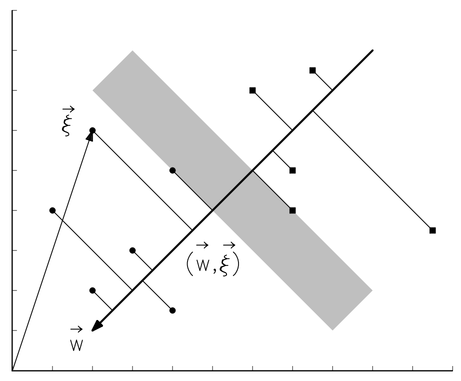

Figure 1: *Map and dead zone (dashed area). Filled circles and squares represent positive and negative examples, respectively.*

According to the usual learning protocol, the network is trained on a set of M input configurations $\{\vec{\xi}^{(\nu)}\}_{\nu=1}^{M}$ generated randomly and independently from some $P(\vec{s})$ probability distribution. For all training examples the corresponding output $f(\vec{\xi}^{(\nu)}) = \sigma_\nu$. I will denote by I_+ and I_- the set of indices corresponding to an output value $\sigma_\nu = \pm 1$ *iff* $\nu \in I_\pm$, respectively. The *maximal dead zone* discriminant surface is obtained by maximizing the gap[16]

$$G = \max_{\vec{w}} \frac{\alpha - \beta}{W} \tag{4}$$

and requiring that all inequalities (4) are fulfilled. The normalization factor W is needed because otherwise multiplying both sides of the inequalities with $\lambda >> 1$ does not change the direction of vector w but makes the gap G arbitrarily large.

The Kuhn—Tucker conditions for the optimum and the algebraic derivation of the dual problem can be found in the original paper of Lambert[16]. Here I use a simple geometric argument to derive the dual problem. The convex hulls $S_\pm$ of the set of positive and negative examples are defined as a convex combination of the *extremal vectors (vertices)* of those sets:

$$\vec{x} \in S_\pm \ \text{ if } \vec{x} = \sum_{\nu \in I_\pm} a_\nu^{(\pm)} \vec{\xi}^{(\nu)} \tag{5}$$

$$\sum_{\nu \in I_\pm} a_\nu^{(\pm)} = 1; \quad a_\nu^{(\pm)} \geq 0 \tag{6}$$

Extremal vectors, corresponding to the vertices of the convex polyhedron Eq. (5-6), cannot be expressed as linear independent combination of other vectors in

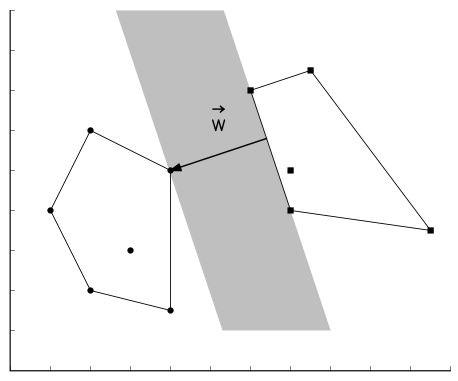

Figure 2: *The maximal dead zone and the minimal connector between the convex hulls of positive and negative examples.*

the set. Note that if the inputs are binary, *all* possible configuration vectors correspond to the vertices of an N-dimensional hypercube and are thus extremal. The two convex hulls of positive and negative examples form two convex polyhedra. They are disjoint for a linear separable function but intersect if the target function is not linear separable. Searching for the maximal dead zone can be thought as moving a parallel set of hyperplanes separating the two polyhedra such that the distance between the two planes is maximal (the 'dead' zone, where no examples are found, is maximal). From Fig. 1 it is clear that the dot product $(\vec{w}, \vec{\xi})$ corresponds to the projection of the input vector $\vec{\xi}$ into the direction $\vec{w}$. The set of dot products

$$h_\nu = (\vec{w}, \vec{\xi}^{(\nu)}) \quad \nu = 1, \ldots, M \tag{7}$$

is accordingly called a *map* formed by the projections of N-dimensional vectors $\vec{\xi}^{(\nu)}$ into the one-dimensional direction defined by $\vec{w}$.

When the hyperplane defined by $\vec{w}$ separates, then there is a gap between the projections of all positive and all negative examples

$$\tilde{G} = \min_{\nu \in I_+} h_\nu - \max_{\nu \in I_-} h_\nu \tag{8}$$

From Fig. 2 it is also clear that this gap is maximal when the weight vector $\vec{w}$ points into the direction along which the distance between the two polyhedra, L, is minimal. Thus, we can reformulate our problem as

$$L^2 = \min_{\{a_\nu^\pm;\ \alpha,\beta\}} (\vec{X}_+ - \vec{X}_-)^2 \tag{9}$$

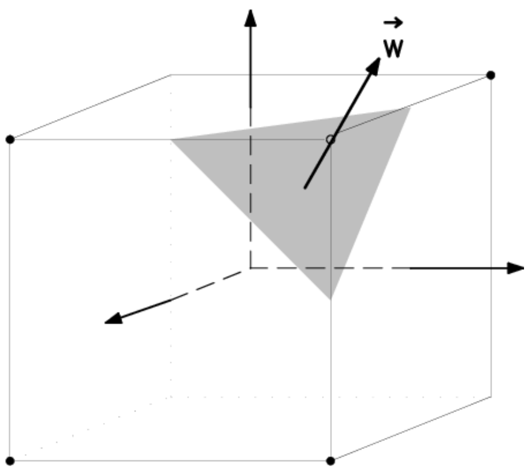

Figure 3: *An example with $N = 3$ binary inputs. The only negative example $(1, 1, 1)$ is represented by an open circle. Positive examples are shown by full circles. After the reflection of the negative example the problem is not linearly separable.*

where $\vec{X}_\pm$ belongs to $S_\pm$, respectively. Thus, $\vec{w} \sim \vec{X}_+ - \vec{X}_-$, or

$$\vec{w} = \sum_{\nu \in I_+} a_\nu^+ \vec{\xi}^{(\nu)} - \sum_{\nu \in I_-} a_\nu^- \vec{\xi}^{(\nu)} \tag{10}$$

The convex coefficients $a_\nu^\pm$ (called also 'embedding strengths') are the Lagrange multipliers corresponding to the different constraints Eq. (4). The minimal connector problem is the dual of the maximal dead zone problem (primal problem), and

$$G = \frac{\tilde{G}}{W} \leq W \tag{11}$$

and $G = L = W$ only at optimality. If the problem is not linearly separable, $G < 0,\ L = 0$.

In the physical literature, the input variables are chosen as $s_i = \pm 1$ binary variables because their resemblance to magnetic $\frac{1}{2}$ spin variables. h_ν corresponds to the total magnetic field acting on the output spin from the different input spins. It is usual to multiply the input variables with their desired output, so that all input configurations will have output 1. Given a separating hyperplane passing through the origin, this transformation corresponds to reflecting the negative examples to the other side of the plane. The maximal stability is thus defined as the maximal distance between the origin and the *union* of the positive examples with the mirror images of the negative examples. This gauge transformation is exact when the separating plane passes through the origin. For very large networks and unbiased examples (the total magnetization vanishes), the gauge transformation is still valid in a probabilistic sense (with probability one)[2]. However, for special cases, like the one shown in Fig. 3, the physicist's definition is more restrictive than the usual one.

3 The active set approach and its implementation

One is thus seeking to minimize the length of the vector

$$L^2 = W^2_{\min} = \min_{\{\vec{a}^{\pm};\ \alpha,\beta\}} (\vec{w}, \vec{w}) \tag{12}$$

where

$$\vec{w} = \sum_{\nu \in I_+} a^+_\nu \vec{\xi}^{(\nu)} - \sum_{\nu \in I_-} a^-_\nu \vec{\xi}^{(\nu)} \tag{13}$$

subject to the inequalities

$$\begin{aligned} (\vec{w}, \vec{\xi}^{(\nu)}) - \alpha &\geq 0 \text{ if } \nu \in I_+ &(14)\\ -(\vec{w}, \vec{\xi}^{(\nu)}) + \beta &\geq 0 \text{ if } \nu \in I_- &(15)\\ a^+_\nu &\geq 0 \text{ if } \nu \in I_+ &(16)\\ a^-_\nu &\geq 0 \text{ if } \nu \in I_- &(17)\end{aligned}$$

and the equalities

$$\sum_{\nu \in I_+} a^+_\nu = \sum_{\nu \in I_-} a^-_\nu = 1 \tag{18}$$

Thus, every input configuration $\vec{\xi}^{(\nu)}$ in the example set defines a constraint indexed by the same index ν.

One can see this problem as having two hollow polyhedra corresponding to the convex hull of the positive and negative examples, respectively. Within each, a demon keeps one end of a magic rubber rope. The demons can move anywhere within their polyhedra but cannot leave it. Starting from some original position $\vec{X}^+_0$ and $\vec{X}^-_0$ within their territories, the demons move towards each other, driven by the rubber rope. Eventually, they hit the boundary walls and travel further on the different faces of the polyhedra. When they both stop, the length of rope is minimal and its direction defines $\vec{w}$. If the two polyhedra intersect (nonlinear separable problem) the demons will 'fall' rather freely towards each other and annihilate on impact. This simple picture summarizes the algorithm I am going now to describe.

First, let us count the independent variables of our problem. The goal is to calculate the N components of weight vector $\vec{w}$ and the two thresholds, α and

[2]For the vast majority of possible configurations

β. Since the norm of $\vec{w}$ is arbitrary, fixing it leaves us with $N+1$ independent variables. The dual problem Eq. (12) depends apparently on $M+2$ variables (the components $a_\nu^\pm$), where usually M is larger than N. This apparent contradiction is easily resolved by remarking that the hyperplane normal direction $\vec{w}$ is fully determined by a set of N linearly independent *difference* vectors $\vec{\xi}^{(\alpha)} - \vec{\xi}^{(\beta)}$, requiring actually at most $N+1$ input vectors. This means that at any instance of the calculation, at most $N+1$ variables $a_\nu^\pm$ are nonzero. Adding the variables α, β and subtracting two degrees of freedom needed to satisfy Eqs. (16-17) one recovers the same number of free variables as for the primal problem.

The constraints corresponding to the set of positive $a_\nu^\pm$ are called *active constraints.* If a constraint is active, its corresponding inequality Eqs. (14-15) is satisfied as an *equality.* The set of active constraints define thus a linear set of equations which can be written in the following matrix form:

$$\sum_\beta Q_{\alpha\beta} x_\beta = b_\alpha \tag{19}$$

In more detail,

$$\left|\begin{array}{c|c|ccc|ccc|}
0 & 0 & 1 & 1 & \cdots & 0 & 0 & \cdots \\ \hline
0 & 0 & 0 & 0 & \cdots & 1 & 1 & \cdots \\ \hline
1 & 0 & q_{11}^{++} & q_{12}^{++} & \cdots & q_{11}^{+-} & q_{12}^{+-} & \cdots \\
1 & 0 & q_{21}^{++} & q_{22}^{++} & \cdots & q_{21}^{+-} & q_{22}^{+-} & \cdots \\
\vdots & \vdots & & & & & & \\
1 & 0 & q_{M_+1}^{++} & q_{M_+2}^{++} & \cdots & q_{M_+1}^{+-} & q_{M_+2}^{+-} & \cdots \\ \hline
0 & 1 & q_{11}^{-+} & q_{12}^{-+} & \cdots & q_{11}^{--} & q_{12}^{--} & \cdots \\
0 & 1 & q_{21}^{-+} & q_{22}^{-+} & \cdots & q_{21}^{--} & q_{22}^{--} & \cdots \\
\vdots & \vdots & & & & & & \\
0 & 1 & q_{M_-1}^{-+} & q_{M_-2}^{-+} & \cdots & q_{M_-1}^{--} & q_{M_-2}^{--} & \cdots
\end{array}\right|
\left|\begin{array}{c}
-\alpha \\ \hline \beta \\ \hline a_1^+ \\ a_2^+ \\ \vdots \\ a_{M_+}^+ \\ \hline -a_1^- \\ -a_2^- \\ \vdots \\ -a_{M_-}^-
\end{array}\right|
=
\left|\begin{array}{c}
+1 \\ \hline -1 \\ \hline 0 \\ 0 \\ \vdots \\ 0 \\ \hline 0 \\ 0 \\ \vdots \\ 0
\end{array}\right| \tag{20}$$

where the element $q_{\alpha\beta}^{\sigma_\alpha\sigma_\beta} = (\vec{\xi}^{(\alpha)}, \vec{\xi}^{(\beta)})$. The parameterization shown in Eq. (20) corresponds to extending the input space by adding a 'header' of $(1, 0, \vec{\xi})$ and $(0, 1, \vec{\xi})$ to the positive and negative example vectors, respectively. Hence, the matrix $\mathbf{Q}$ is a symmetric extension of the overlap matrix. All its diagonal elements, except for the first two (the convex conditions Eq. (18)), are large, positive numbers: for ± 1 binary inputs $Q_{ii} = N$, $i = 3, \ldots, N+3$. This allows the use of a simple non pivoting procedure for locally updating the inverse of $\mathbf{Q}$.

Any set of active constraints satisfying (20) and fulfilling $a_\nu^\pm > 0$ satisfies the Kuhn—Tucker conditions. In general, this will be the solution of some *subproblem.* The *active set strategy* is a method of searching for the optimal active constraints by a systematic improvement based on solving such subproblems. In practice this is achieved by adding and removing constraints to and from the actual active set, such that each exchange improves (decreases) the actual value of W. Given a subsolution $\vec{w}^{(n)}$ and the thresholds $\alpha^{(n)}$, $\beta^{(n)}$, the most violated constraint is obtained by calculating

$$mvc \; : \; \arg\min_{\{\nu,\mu\}} \{(\vec{w}^{(n)}, \vec{\xi}^{(\nu)}) - \alpha^{(n)}; \; \beta^{(n)} - (\vec{w}^{(n)}, \vec{\xi}^{(\mu)})\} \tag{21}$$

for all *inactive* constraints such that $\nu \in I_+$ and $\mu \in I_-$. The most violated inequality corresponds thus to the worst classified example.

Following Goldfarb and Idnani [24], the algorithm is formulated as following:

1. Initialize $\vec{X}_+$ and $\vec{X}_-$ using two (possibly close) examples from each class. Initialize the list of active constraints, $\vec{w}$, the thresholds and the inverse of the $\mathbf{Q^{-1}}$ matrix.

2. Generate most violated constraint. If no violated constraint is found, the program converged.

3. Add most violated constraint to the active set. If necessary, remove some constraints from the set. Update correspondingly the $\mathbf{Q^{-1}}$, the $\vec{x}$ solution vector and the list of active constraints. Check W: if $W = 0$ the problem is not linearly separable. Otherwise go to step 2.

I have shown schematically in Figs 4a-b how the algorithm is supposed to work in a two dimensional case ($N = 2$, the input variables are real numbers). The form of the extremal vertices codes for the desired output (full circles for 1 and full squares for -1).

I present now the proof that adding the *mvc* to the active set will always reduce the length of $\mathbf{w}$. The following notations correspond to those of Figs. 5a–5b. The present solution is given by the vector $\vec{w} = \vec{X}^+ - \vec{X}^-$. After adding the *mvc* shown here as $\vec{\xi}^{(\text{new})}$, the solution will change to $\vec{w}' = \vec{X}^{+'} - \vec{X}^-$ (see Fig. 5b). I assume that only the position of the demon on the positive polyhedral set will change. The case when the other side ($\vec{X}^-$) changes its position can be easily reproduced following the same line of thought. When both positions change, the motion can be seen as a succession of positive and negative moves.

The motion of $\vec{X}^+$ can be split into a series of simple moves, $\vec{X}^+ \to \vec{X}_1^+ \to \vec{X}_2^+ \ldots \to \vec{X}^{+'}$ so that the length of the vector $\vec{w}$, denoted by W, decreases as $W > W_1 > W_2 \ldots W_n \to W'$. The basic step is shown in Fig. 5a. Remember that $\vec{X}^+ = \sum_{\nu \in I_+} a_\nu^+ \vec{\xi}^{(\nu)}$ is a convex combination of the positive examples in the active set. This set is extended now to contain the *mvc* as

$$\vec{x} = (1 - \lambda)\vec{X}^+ + \lambda \xi^{(\text{new})} \tag{22}$$

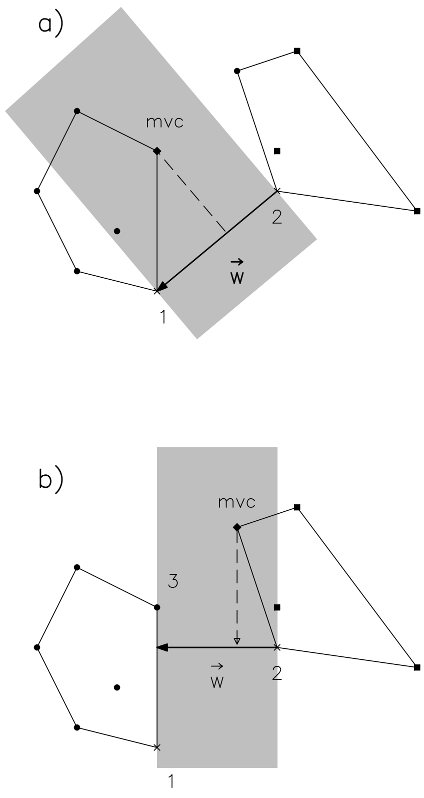

Figure 4: *Schematic steps made by the dual algorithm in two dimensions when starting from the vertices 1 and 2. In every step the most violated constraint (mvc) is added to the map (filled diamond) and eventually active constraints are removed (crosses). Performing the move as indicated in b) leads to the optimal solution shown in Fig. 2.*

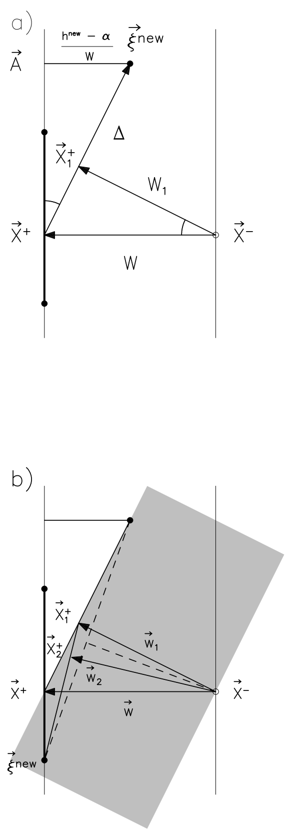

Figure 5: *Adding a violated constraint: a) Constraining motion along the vector* $\vec{\Delta}$ *leads to* $W_1 = W\frac{h^{\text{new}}-\alpha}{W\Delta} < W$ *(see text). b) The iteration of this construction as* $\vec{w} \to \vec{w}_1 \to \vec{w}_2 \to \ldots$ *converges to the* $\vec{w}'$*, shown by a dashed line.*

The vector $\vec{x}$ moves thus on the line connecting $\vec{X}^+$ to $\vec{\xi}^{(\text{new})}$. Denote by $\vec{\Delta} = \vec{\xi}^{(\text{new})} - \vec{X}^+$ and its length by Δ. With this notation

$$\vec{x}(\lambda) = \vec{w} + \lambda\vec{\Delta} \tag{23}$$

In order to get $\vec{w}_1$ one has to minimize the length of $\vec{x}$ according to the free parameter $0 \leq \lambda \leq 1$. Noting that $(\vec{w}, \vec{\Delta}) = h_{\text{new}} - \alpha$, one obtains after an elementary calculation that

$$W_1^2 = (\vec{w}_1, \vec{w}_1) = \min_{\lambda}(\vec{x}, \vec{x}) = W^2 - \frac{(h_{\text{new}} - \alpha)^2}{\Delta^2} < W^2 \tag{24}$$

The same result can be obtained by noting the similarity between the two triangles $(\vec{\xi}^{(\text{new})}, \vec{X}^+, \vec{A})$ and $(\vec{X}^-, \vec{X}^+, \vec{X}_1^+)$ in Fig. 5a. As already mentioned, $\vec{X}_1^+$ lies *inside* the convex combination of the old active set plus the $\vec{\xi}^{(\text{new})}$. This procedure can be iterated as indicated in Fig. 5b, where a local[3] *mvc* is determined by using $\vec{w}_1$ and $\alpha_1 = (\vec{w}_1, \vec{X}_1^+)$. Note that no constraint is ever removed. Only in the large iteration limit will eventually approach the convex coefficient of some active constraint to zero. Eq. (24) shows that using *any* violated constraint will decrease W but the largest improvement is obtained by adding the most violated constraint.

4 Implementation

The most often used computational step is solving the set of linear equations Eq. (20). This suggests carrying the inverse of the matrix $\mathbf{Q}$. Since the vector $\vec{b}$ has only two nonzero elements, the solution of a subproblem is given simply by

$$x_i = Q^{-1}_{2i} - Q^{-1}_{1i} \tag{25}$$

The vector $\vec{w}$ is calculated from $\vec{x}$ (Eq. (13)). It is also useful to introduce a list containing the indices of the active constraints as stored in the i^{th} row and column of $\mathbf{Q^{-1}}$ or indicating that the row (column) is a free slot.

Having set up the data structure, the implementation of the main steps defined above follows.

1. When adding or removing a constraint the changes in $\mathbf{Q}$ and $\mathbf{Q^{-1}}$ are both local. In fact, it is useful to initialize $\mathbf{Q}$ and $\mathbf{Q^{-1}}$ as an $(N+3)\times(N+3)$ unit matrix. The first four rows and columns are calculated from two initial examples, one from the positive, one from the negative set. The first two rows (columns) correspond to the normalization conditions Eq. (18). The following two rows are filled as in Eq. (20). The resulting 4×4 block is easily inverted by hand. The solution vector, the weight vector, and the list of active constraint is initialized accordingly.

2. The most violated constraint is determined once the map of inactive examples has been calculated. Note that due to rounding errors and to possible presence of degenerate faces, it is recommended to require *mvc* values larger than some small $\epsilon > 0$ value.

3. Adding (or removing) a constant corresponds to adding (removing) a row and a column. The updating of the inverse matrix is implemented following the Sherman–Morrison formula described in Demidovich & Maron [21] or in *Numerical Recipes* [22] for hypermatrices. In our case one has to add only a row and a column to a matrix whose inverse is known. Assume that a 'free slot' has index j — meaning that $Q_{ij} = Q^{-1}_{ji} = \delta_{i,j}$ before adding the

[3]Only points on the active set are considered

constraint. Here $\delta_{i,j}$ stands for the Kronecker–delta, which is unity if $i = k$ and zero otherwise. Adding a constraint entails the following operations: First, one has to form the row vector $\vec{u}$ to be added to $\mathbf{Q}$. It consisting of the 'header', coding for the class (10 or 01,for $\sigma = \pm 1$, respectively), followed by the elements

$$u_i = (\vec{\xi}^{(i)}, \vec{\xi}^{(j)}) \quad \forall i \in \text{occupied slots} \tag{26}$$

and $u_j = 0$. Next, one calculates the vector

$$z_i = \sum_{k \in \text{occupied slots}} Q^{-1}_{ik} u_k \tag{27}$$

and the 'determinant' d

$$d = (\vec{\xi}^{(j)}, \vec{\xi}^{(j)}) - (\vec{u}, \vec{z}) \tag{28}$$

With these vectors the inverse $\mathbf{Q^{-1}}$ is updated as

$$Q^{-1}_{ik} \leftarrow Q^{-1}_{ik} + \frac{1}{d} z_i z_k - \frac{1}{d} z_i(\delta_{k,j} + \delta_{j,k}) + \frac{1}{d} \delta_{i,j}\delta_{k,j} \tag{29}$$

Likewise, deleting a constraint corresponds to removing an outer product of two vectors (diad) from $\mathbf{Q^{-1}}$

$$Q^{-1}_{ik} \leftarrow Q^{-1}_{ik} - \frac{1}{Q^{-1}_{jj}} y_i y_k + \delta_{i,k} \tag{30}$$

where $y_i = Q^{-1}_{ij}$.

As it is, the solution of the linear equations Eq. (20) might well provide solutions *outside* the convex hulls. In such cases one or more convex coefficients are non positive, $a^{\pm}_i \leq 0$ and the corresponding constraints are removed from the active set. If more than one constraint must be removed, the result might depend on the order of removal. Assume that the most violated constraint is $\vec{\xi}^{(\text{new})}$ and consider $\lambda\vec{\xi}^{(\text{new})}$, λ being an arbitrary variable. Increasing λ from zero, the old solutions $a^{\pm}_i$ will change gradually until for some $\lambda < 1$ value one of them vanishes. This is the variable to be removed first. After removal, the process is iterated until no convex coefficient becomes negative as λ is increased to 1. It is important to recognize that the removal of constraints is done *first*, and only then is the most violated constraint added to the set.

There are two cases when removing constraints is unavoidable: a) the number of active variables is already $N+1$. Any further active constraint will be linearly dependent of the already present vectors and the matrix $\mathbf{Q}$ is singular. b) The most violated constraint is (almost) a linear combination of the already present active vertices. The matrix $\mathbf{Q}$ is then badly conditioned, a fact reflected by a very small d value. This is a sure sign that some constraint *must* be removed before attempting to add the most violated constraint, or that this particular *mvc* can be 'eliminated' from the example set.

As an independent check for convergence and accuracy, the gap Eq. (8) is calculated and compared to L Eq. (12). According to the duality arguments[25] the two values are equal only at optimality.

5 Numerical results and comparison to other algorithms

Looking at the load of this algorithm, one recognizes that the most expensive operations are finding the most violated constraint (this boils down to M scalar products of real N dimensional vectors) and adding a constraint, which involves the calculation of $\vec{u}$ (N scalar products) and the matrix-vector multiplication to obtain $\vec{z}$, both $O(N^2)$. For $M > N$ the load is dominated by finding the *mvc*.

How many exchanges of active constraints are needed? My experience shows that for large enough problems, the number of constraint changes increases rather slowly with the value of $\frac{M}{N}$, being typically on the range of $1.5 - 4N$. This is expected, since as $\frac{M}{N}$ increases the gap becomes smaller and the 'demons' have a harder task in finding their way through the many face(t)s of the polyhedra.

The total number of operations involved in the algorithm is thus of order cMN^2, where c is a slowly increasing function of $\frac{M}{N}$. The running time could be further improved by using block schemes[23] but I am not sure the additional bookkeeping is worthwhile unless some very large scale problems come along. As it stands, the algorithm is fully vectorizable and the code gives the user the freedom to use its own optimized version of a scalar product and a matrix-vector product.

Table I shows real CPU time on an scalar Apollo-HP series 400 machine (68040 processor) for a number of random linear separable problems: after generating at random the components of a hyperplane normal, n_i, in N dimensions, randomly generated $\xi_i^{(\nu)} = \pm 1$ vectors are assigned output ± 1 depending on which side of the plane $\vec{n}$ they fall, $\sigma_\nu = \text{sgn}\ (\vec{n}, \vec{\xi}^{(\nu)})$. Results are presented for 5 independent runs keeping $N = 128$ constant and changing the number of examples $M = 1024,\ 2048,\ 4096,\ 8192$ and by keeping $M = 1024$ constant and varying the number of input variables $N = 128,\ 256,\ 512$. Note the faster convergence for nonlinear separable problems, due to the 'free fall' towards $L = 0$.

Table I

Run	N	M	lin. sep.	not lin. sep.	CPU [s]	Iterations [steps]
1	128	1024	$\surd$	-	140.5	257
2					136.9	241
3					124.7	232
4					125.7	233
5					136.2	249
1	128	1024	-	$\surd$	49.7	127
2					51.1	12
1	128	2048	$\surd$	-	283.1	321
2					310.5	346
3					284.5	323
4					315.5	352
5					281.0	322
1	128	2048	-	$\surd$	88.2	124
2					86.6	126
1	128	4096	$\surd$	-	651.3	528
2					616.0	409
3					636.5	649
4					586.0	389
5					581.1	387
1	128	4096	-	$\surd$	153.5	115
2					154.1	114
1	128	8192	$\surd$	-	1453.2	530
2					1253.3	471
3					1553.9	568
4					1453.1	542
5					1383.7	512
1	128	8192	-	$\surd$	281.1	107
2					286.1	110
1	256	1024	$\surd$	-	404.2	336
2					409.2	340
3					439.3	354
4					408.1	333
5					440.4	349
1	512	1024	$\surd$	-	1447.8	499
2					1494.4	503
3					1537.8	482
4					1486.2	497
5					1545.6	511

Average running time for randomly generated linear separable and non linear separable functions for different values of M and N: see text for explanation.

Table II shows comparisons between the MinOver, AdaTron, and the present implementation for 5 different values of M and N. This comparison is somewhat difficult because a) the MinOver algorithm depends on the precision ϵ required for the solution as $1/\epsilon$, b) the AdaTron algorithm is sometimes instable when the threshold value is not zero, c) both were implemented by storing the *full* $M \times M$ overlap matrix, while the quadratic programming algorithm was devised exactly to avoid this feature. I opted (rather arbitrarily) for a precision of three significant digits (MinOver and AdaTron), test functions with a zero threshold, and $M = 500$ (so that the overlap matrix fits into the core). These conditions present a rather large handicap for quadratic programming algorithm, which was run in its original implementation. As shown in Table II, the AdaTron algorithm fared best when the number of input units was rather large ($N = 400, 200$) but had difficulties when the gap became smaller. The MinOver algorithm (implemented as in [26]) was not far behind but this depends strongly on the required accuracy. Despite the unfavorable test conditions, the quadratic programming algorithm was by far the best for difficult problems with a small gap. It provides 'sharp', accurate solutions even for problems where the distance between the two example sets is only several times machine accuracy small. Such solutions are practically undetectable with gradient descent methods. The numerical stability of the algorithm relies on the fact that the overlap matrix $\mathbf{Q}$ has large positive diagonal elements. For problems where this is not the case, numerical instabilities may show up. They can be avoided either by performing linear transformations on the example set (like moving the origin, rescaling, etc.), or by using the more stable LU decomposition of $\mathbf{Q}$ based on pivoting, as in the original work of Goldfarb and Idnani [24].

In conclusion, the combination of numerical stability and the ability of handling large sets of examples in a reasonable amount of time makes the quadratic programming algorithm the routine of choice for creating compact feedforward networks for real-world applications[27].

Table II

Algorithm	N = 50	N = 100	N = 200	N = 400
MinOver	247.5	174.3	184.3	282.6
	157.2	162.4	188.6	279.3
AdaTron	no conv.	766.2	240.6	225.7
	1748.0	558.3	221.1	220.8
QuadProg	13.1	49.8	147.3	505.0
	11.5	42.7	172.5	441.1

Running time for $M = 500$ randomly generated linear sets (in seconds). Comparison between MinOver [19], AdaTron [20], and the present algorithm (QuadProg).

Acknowledgments

Helpful discussions with H.J. Schellnhuber and support from the German - Israeli Foundation are kindly acknowledged. A C-code implementation of the algorithm is available on this site (http://www.neuro.uni-oldenburg.de/~rujan - software).

References

[1] M. Marchand, M. Golea, and P. Rujan *Europhys. Lett.* **11** (1989) 487-492 **27** 1134 (1984)

[2] M. Mézard and J.-P. Nadal *J. Phys. A* **22** (1989) 2191-2203

[3] S. Knerr, L. Personnaz, G. Dreyfus in *Neurocomputing* F. Fogelman Soulié and J. Hérault (Eds.) NATO ASI Series, Springer-Verlag 1990 pp.41-50

[4] S. L. Gallant *Neural Networks* **3** (1990) 191-201

[5] Rosenblatt, F. *Psychoanalytic Review* **65** (1958) 386

[6] R. O. Duda and P. E. Hart *Pattern Classification and Scene Analysis* J. Wiley & Sons 1973

[7] S. Watanabe *Pattern Recognition: Human and Mechanical* pp. 363-378 J. Wiley & Sons, 1985

[8] S. I. Gallant 'Optimal Linear Discriminants' *Eight International Conference on Pattern Recognition* Vol. 2, 849, Paris, 1986

[9] Judd S. *Proc. IEEE First Conference on Neural Networks San Diego 1987* Vol. II pp. 685-692 (IEEE Cat. No. 87TH0191-7)

[10] A. Blum and R. Rivest *Proc. of the 1988 Workshop on Computational Learning Theory* pp. 9-18 Haussler D. and Pitt L. (Eds.) Morgan Kaufman, San Mateo, Ca.

[11] W. Kinzel and P. Ruján *Europhys. Lett.* **13** (1990) 473-477

[12] E. B. Baum *IEEE Trans on Neural Networks* **2** (1991) 5-19

[13] D. Haussler and M. Opper *Phys. Rev. Lett.* **66** (1991) 2677

[14] W. Kinzel in *Statistical Mechanics of Neural Networks*, L. Garrido, (Ed.), Lecture Notes in Physics **368**,Springer Verlag, 1990

[15] M. Opper, W. Kinzel, J. Kleinz and R. Nehl *J. Phys. A* **23** L581 (1990)

[16] P. F. Lambert in *Methodologies of Pattern Recognition*, M. S. Watanabe (Ed.) Academic Press, New York, 1969

[17] D. Vapnik *Estimation of Dependencies from Empirical Data* see Appendices Springer Verlag, 1982

[18] E. Gardner *J. Phys. A* **21** 257 (1988)

[19] W. Krauth and M. Mézard *J. Phys. A* **20** L745 (1987)

[20] J. K. Anlauf and M. Biehl *Europhys. Lett.* **10** (1990) 687-692

[21] B. P. Demidovich and I. A. Maron *Computational Mathematics* Mir Publishers, Moscow, 1976 pp 260-265

[22] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, Second Edition, Cambridge University Press, 1995

[23] G. H. Golub and C. F. van Loan *Matrix Computations* The J. Hopkins University Press, 1989

[24] D. Goldfarb and A. Idnani *Mathematical Programming* **27** 1-33 (1983)

[25] W. S. Dorn *Quarterly of Applied Mathematics* **18** (1960) 155-162

[26] P. Ruján in *Statistical Mechanics of Neural Networks*, L. Garrido, (Ed.), Lecture Notes in Physics **368**, Springer Verlag, 1990

[27] U. Ramacher and P. Ruján: "A Neural Network Approach to Defect Recognition in Designed Chip Masks", unpublished, 1991

[28] M. K. Kozlov, S. P. Tarasov, and L. P. Khachian, *Soviet. Math. Dokl* **20** (1979) 1108-1111