Efficient Linear Discriminant Analysis Using a Fast Quadratic Programming Algorithm

S. J. Perantonis and V. Virvilis

Institute of Informatics and Telecommunications,
National Center for Scientific Research "DEMOKRITOS",
Athens, Greece

Abstract-An algorithm is proposed for performing linear discriminant analysis using a single-layered feedforward network. 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 algorithm has no free parameters and therefore no heuristics are involved in its application. Its efficiency in terms of speed of convergence is demonstrated in a number of pattern classification problems.

I. Introduction

In recent years, artificial neural networks have been widely applied to pattern recognition problems. In particular, feedforward networks have emerged as efficient tools for supervised discrimination of patterns belonging to two or more categories. Although recent research has mainly focused on multilayered networks, the single layer feedforward network still deserves attention for at least two reasons:

Firstly, the design of fast learning algorithms for this type of network is important, because such algorithms can form the basis of layer-by-layer learning schemes for multilayer feedforward networks [1]. 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 a single layered feedforward network, e.g. the perceptron learning rule [2] and the delta rule [3] 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 [4]. 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 [5].

In this paper we propose a learning algorithm for a single layer network, which is fast and requires no adjustment of parameters. Learning proceeds by taking extra precautions to ensure that during training the algorithm won't increase the number of misclassified patterns. The above requirement leads to a quadratic programming problem, for which a fast method of solution is proposed. The algorithm can find the solution to large scale linearly separable problems much faster that the perceptron rule. Its fast convergence is not hindered by inhomogeneities in the distribution of training patterns.

II. Terminology and Background

We wish to distinguish between two linearly separable classes of P patterns $\mathbf{x_p}$, each of dimension N. 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 divides the weight space into 2 subspaces. Hyperplanes corresponding to different patterns $\boldsymbol{d_p}$ segment R^{N+1} into several convex regions (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 BitWrong (BW) if O_p is not equal to T_p.

Instead of $\boldsymbol{X_p}$, it is useful to describe the patterns using the vectors $\boldsymbol{d_p} = (2T_p - 1)\boldsymbol{X_p}$, which always point towards the side of the pattern hyperplane that corresponds to correct classification. Thus, patterns

classified as BR (BW) are characterized by a positive (negative) value of the the quantity $\boldsymbol{W}^T\boldsymbol{d_p}$. The original perceptron problem (1) now becomes:

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

There are two cost functions related to our problem:

- The perceptron cost function [4] is defined by

$$E = \sum_{p=1}^{P}(T_p - O_p)\boldsymbol{W}^T\boldsymbol{X_p} = -\sum_{p=BW}\boldsymbol{W}^T\boldsymbol{d_p} \tag{3}$$

  where the second sum runs over all patterns that are classified as BW by $\boldsymbol{W}$. This cost function is piecewise linear in R^{N+1} and has constant gradient $\boldsymbol{\Delta W}$ in each polytope given by

$$-\boldsymbol{\Delta W} = \sum_{p=1}^{P}(T_p - O_p)\boldsymbol{X_p} = -\sum_{p=BW}\boldsymbol{d_p} \tag{4}$$

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

- The squared error cost function

$$E_{SE} = \sum_{p=1}^{P}\Big(\theta(\boldsymbol{W}^T\boldsymbol{d_p}) - 1\Big)^2 \tag{5}$$

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

III. Training strategy

We initialize the training procedure with the weight vector $\boldsymbol{W}$ in the interior of a certain polytope $\mathcal{R}_{\mathcal{I}}$. Our aim is to reach a minimum of the number of BW. Our strategy involves updating $\boldsymbol{W}$ along successive search directions, each characterized by a vector $\boldsymbol{P}$. To determine these directions, we shall use information related to the gradient $\boldsymbol{\Delta W}$ of E. In particular, at each epoch, the search direction is the direction of steepest descent with respect to the vector $\boldsymbol{\Delta W}$, subject to the constraints that no BR patterns are crossed over. For the first epoch, we choose the vector $\boldsymbol{P} = \boldsymbol{\Delta W}$ and we update $\boldsymbol{W}$ according to:

$$\boldsymbol{W}_{new} = \boldsymbol{W} + n\boldsymbol{P} \tag{6}$$

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 $\boldsymbol{W}_{new}^T\boldsymbol{d_p} = 0$, i.e. $n_p = -(\boldsymbol{W}^T\boldsymbol{d_p})/(\boldsymbol{P}^T\boldsymbol{d_p})$, 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".
   (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" so that the next move will be parallel to the hyperplane of the specified pattern. This procedure of moving practically to zero distance from a BR pattern will be called "Moving Near".
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 $\boldsymbol{P} = 0$ 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 $\mathcal{R}$ 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, we use E again.

   We update $\boldsymbol{W}$ so that the new vector $\boldsymbol{W}_{new}$ remains in $\mathcal{R}$ so that and E decreases locally at the fastest possible rate. We therefore choose the search direction of steepest descent that has common points with R. Finding this direction is a difficult problem that will be studied in detail in

section IV.. 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 (6) 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 $\mathcal{R}_{\mathcal{N}}$ of lower E_{SE} than in the previous epoch. We must now use the new E, corresponding to the polytope $\mathcal{R}_{\mathcal{N}}$ to find the direction of steepest descent that has common points with $\mathcal{R}_{\mathcal{N}}$. Once again, $\boldsymbol{W}$ is updated according to (6) with n determined as in cases 1 or 2 above.

The algorithm will terminate when $\boldsymbol{P} = 0$, which can happen in the following situations:

1. if $E_{SE} = 0$, in which case the classification problem was linearly separable and our algorithm has reached a solution.
2. if the minimum of the cost function E has been reached in a certain polytope following the "Moving Near" procedure. In this case, no further move is possible, since the algorithm has no way of crossing over a BW pattern, thus further lowering E_{SE}.

Importantly, it can be proved that for linearly separable problems the process described above will always separate the patterns in a finite number of epochs. Moreover, for non-linearly separable problems, the algorithm will always terminate in a finite number of steps as in case 2 above, having done its best to minimize the number of BW. Thus, termination of the procedure when there are still BW patterns left means that the given classes of patterns were non-linearly separable, so that the procedure can be used as a test of linear separability. For brevity reasons, the proofs of these important statements are omitted and will be given elsewhere.

We note in passing that Bobrowski and Niemiro [6] have proposed a similar algorithm that follows search directions related to polytope edges in order to minimize E. This was a successful algorithm that, in our opinion, has attracted less attention than it deserved in the literature.

IV. Steepest descent direction

A. Mathematical Formulation of the problem

Let us assume that $\boldsymbol{W} \in R^{N+1}$ resides at the intersection of $K \leq N$ 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: $p \in \mathcal{N}_k = \{1 \dots K\},\ K \leq N+1$. We wish to find a vector $\boldsymbol{W} + \boldsymbol{P}$ that belongs to the feasible region ($\boldsymbol{W}^T \boldsymbol{d_p} \geq 0$) and lies in the direction of steepest descent for our original cost function E given by (3).

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

$$\begin{aligned} \text{Minimize } F &= \frac{1}{2}||\boldsymbol{\Delta W} - \boldsymbol{P}||^2 \qquad (7) \\ \text{subject to } \boldsymbol{P}^T \boldsymbol{d_p} &\geq 0 \ \ \forall p = 1 \dots K \end{aligned}$$

Suppose, for the sake of the argument, that we have been able to determine the list of active constraints ($\boldsymbol{P}^T \boldsymbol{d_i} = 0, i \in \mathcal{M} \subseteq \mathcal{N}_k$) for the solution of the program defined by (7). 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 and can be readily obtained using the well known Gramm-Smidt procedure. This is useful information, and will play an important role in forming an efficient algorithm for solving (7). Before proceeding to describe the algorithm, it is necessary to simplify (7) by converting it to an equivalent problem with mutually orthogonal constraint hyperplanes.

B. Transformation to a problem with orthogonal constraints

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 $\Re^K$.

$$\boldsymbol{\Delta W} = \boldsymbol{\Delta W}_{\perp} + \boldsymbol{\Delta W}_{\parallel} \text{ and } \boldsymbol{P} = \boldsymbol{\Delta W}_{\perp} + \boldsymbol{Q} \qquad (8)$$

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 (7) 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.

We shall form the basis of the R^K subspace by the vertices $\boldsymbol{v_p}$ by which the hypercorner is formed. 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:

$$\boldsymbol{Q} = \sum_{i=1}^{K} q_i \frac{\boldsymbol{v_i}}{||\boldsymbol{v_i}||^2},\ q_j = \boldsymbol{P}^T \boldsymbol{d_j} \geq 0 \qquad (9)$$

$$\boldsymbol{\Delta W}_{\parallel} = \sum_{i=1}^{K} a_i \frac{\boldsymbol{v_i}}{||\boldsymbol{v_i}||^2},\ a_j = \boldsymbol{\Delta W}^T \boldsymbol{d_j} \qquad (10)$$

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

$$\boldsymbol{X} = \mathbf{V}\boldsymbol{x} \Rightarrow \boldsymbol{x} = \mathbf{D}\boldsymbol{X} \tag{11}$$

where $\mathbf{V}$ is a $(N+1)\times K$ matrix with the vector $\frac{\boldsymbol{v_i}}{||\boldsymbol{v_i}||^2}$ in the ith column and $\mathbf{D}$ is a $K\times(N+1)$ matrix with the vector $\boldsymbol{d_i}$ in the ith row.

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

$$F = \frac{1}{2}(\boldsymbol{a}-\boldsymbol{q})^T\mathbf{R}(\boldsymbol{a}-\boldsymbol{q}), \quad \boldsymbol{q} \geq 0 \tag{12}$$

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 (7) subject to non orthogonal constraints to minimizing a hyperelliptical quadratic form subject to orthogonal constraints.

C. Double Search Technique

In order to solve (7), we shall reside on 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 search direction $\Delta\boldsymbol{q}$, we can find the position where F attains its minimum value in the feasible space along this search direction. The unconstrained minimum resides at $\boldsymbol{q'} = \boldsymbol{q} + \eta_g\Delta\boldsymbol{q}$ where η_g is found by linear minimization across the search direction:

$$\eta_g = -\frac{\nabla {F_q}^T \Delta\boldsymbol{q}}{||\boldsymbol{\Delta Q}||^2}, \text{ where } \boldsymbol{\Delta Q} = \boldsymbol{V}\Delta\boldsymbol{q} \tag{13}$$

and the gradient ∇F_q of F is given by:

$$\nabla F_q = -\mathbf{R}(\boldsymbol{a}-\boldsymbol{q}) = \boldsymbol{V}^T(\boldsymbol{P}-\boldsymbol{\Delta W}) \tag{14}$$

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 $\boldsymbol{q'} = \boldsymbol{q} + \eta_c\Delta\boldsymbol{q}$, where η_c is the minimum among all positive $\eta_i = -q_i/\Delta q_i$:

$$\eta_c = \min\{\eta_i = -\frac{q_i}{\Delta q_i} : \eta_i > 0 \text{ and } i \in \mathcal{N}_k\} \tag{15}$$

In all cases, the new point $\boldsymbol{q} + \eta\Delta\boldsymbol{q}$ 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 $\Delta\boldsymbol{q^A}$ may not be feasible. However, the orthogonality of the constraints allows us to find a new feasible search direction by starting from $\Delta\boldsymbol{q^A}$ and removing (setting to zero) those components that lead outside the feasible region. Formally, the appropriate search direction is given by a vector $\Delta\boldsymbol{q}$, with components:

$$\Delta q_i = \begin{cases} 0, & \text{if } q_i = 0 \text{ and } \Delta q_i^A < 0 \\ \Delta q_i^A, & \text{otherwise} \end{cases} \tag{16}$$

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

$$\Delta\boldsymbol{q^A} = -\nabla F_q \tag{17}$$

which always leads to lower values of the cost function. However, 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:

$$\Delta\boldsymbol{q}^A = \boldsymbol{D}(\boldsymbol{\Delta W}_{GS} - \boldsymbol{P}) \tag{18}$$

If the active constraints of the solution of (7) had already been found, this would locate the solution in just one step. However, following only this direction we have no guarantee that the algorithm will not terminate before the minimum has been found.

To guarantee convergence *and* find the solution in a small number of steps, both gradient and projection information must be combined in the same algorithm. In each internal epoch the proposed Double Search Algorithm tests both gradient and projection search directions, and selects the one leading to the lower final value of F. The algorithm avoids zig-zag paths, and our experience shows that it locates the exact solution in a few iterations.

There follows a full description of the algorithm:

Initialization: Set $\boldsymbol{Q}$ =0 (Equivalently $\boldsymbol{q}$ =0). Initialize the list of currently active constraints to contain all $i \in \mathcal{N}_k$.

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 (17):
   (a) Calculate the feasible direction $\Delta\boldsymbol{q}$ using (16).
   (b) Calculate η_g using (13) and η_c using (15). 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 (18).
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 $\Delta \boldsymbol{q}$ that led to the most negative ΔF and the corresponding value of η, update $\boldsymbol{q}$ as $\boldsymbol{q}' = \boldsymbol{q} + \eta \Delta \boldsymbol{q}$.
5. Update the list of currently active constraints.

Termination: The algorithm terminates when no further move is possible. That means $\Delta q_i = 0$ or $\Delta q_i < 0$ and $q_i = 0$.

V. Simulations

The following pattern classification problems are studied:

1. *Linearly separable problem with uniform 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:* It is required to discriminate between points lying inside and outside a hyperellipse embedded in M-dimensional space, whose points are characterized by $\sum_{i=1}^{M}(x_i - c_i)^2/a_i^2 = 1$. The problem is made linearly separable by forming input vectors consisting of x_i and $y_i = x_i^2$ $(i = 1, 2, \ldots, M)$. 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 [7], that consists of 208 input vectors, each with 60 components.

Three algorithms are used to solve all benchmark tasks, namely the algorithm proposed in this paper (section III), Bobrowski and Niemiro's (BN) [6] suggestion of following polytope edges for lowering the perceptron cost function at each epoch, and, finally, Rosenblatt's perceptron rule.

Results regarding speed of convergence are presented in Tables 1 and 2. The performance of each algorithm is presented in terms of average number of epochs and average CPU time over ten trials starting from different randomly selected initial weights. The algorithm proposed in this paper and BN algorithm are executed until normal termination. The termination criterion for perceptron was 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 and the BN method, our algorithm exhibits an advantage in terms of learning speed, which is less pronounced in the small scale benchmarks, but becomes definitive in medium and large scale problems. Note that the perceptron rule was not able to solve the large scale "uniform" and "ellipse" problems in the allocated time of one hour. For the sonar data problem, our algorithm has managed to separate the data completely in 215 epochs. Thus this problem is linearly separable, a fact that eluded Gorman and Sejnowski, who report only 85% success rate using a single layered perceptron. This is not surprising, since the perceptron rule needs more than 100,000 epochs to solve the problem.

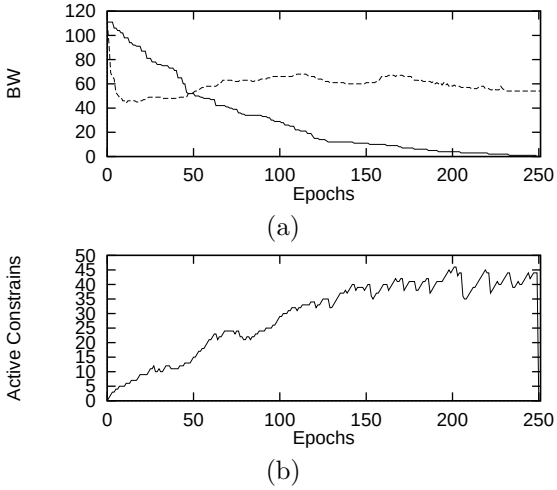

Figure 1: (a) MSE versus number of epochs for our method (solid curve) and the perceptron rule (dotted curve). (b) Number of active constraints versus number of epochs for our algorithm.

	Proposed	BN	Perceptron
Uniform			
P=100 N=2	0.019	0.022	0.8192
P=1000 N=4	0.334	0.539	2.127
P=10000 N=20	43.38	133.70	3600 (∗)
Ellipse			
P=100 N=4	0.054	0.056	0.067
P=10000 N=20	99.36	361.23	2750.13
P=20000 N=40	1047.03	9066.63	3600 (∗)
Sonar	235.21	777.14	1574.48

Table 1: CPU time in seconds. Asterisks denote that the corresponding algorithm failed to solve the problem in the allocated CPU time.

Figure 1 shows a typical learning session for our algorithm. In figure 1a, the squared error E_{SE} is plotted against the number of epochs for the sonar data problem. The corresponding curve for the perceptron rule is also plotted for comparison. In figure 1b, the number of active patterns K is plotted against the number of epochs for our algorithm. Note that at the beginning of learning E_{SE} drops at a relatively slow rate (in comparison with the perceptron rule).

	Proposed	BN	Perceptron
Uniform			
P=100 N=2	8.7	9.6	925.1
P=1000 N=4	20.6	33.2	246.7
P=10000 N=20	83.5	216.5	13179 (∗)
Ellipse			
P=100 N=4	18.2	22.5	57.8
P=10000 N=20	206.6	553.21	12504.6
P=20000 N=40	496.7	1998.1	4683.3 (∗)
Sonar	215.3	182.3	145639

Table 2: Number of epochs needed to solve the classification tasks of Table 1.

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. The advantage in terms of CPU time compared to the BN method (as shown in Table 2) mainly comes from the fact that the BN method follows only edges after the first $N+1$ epochs and therefore the number of active constraints soon saturates at the value $N+1$. In our case, the number of active constraints remains lower than $N+1$, leading to lower computational complexity.

VI. Conclusions

In this paper, an algorithm for fast training of the single layered feedforward network was proposed. Our algorithm is based on constrained optimization methods that utilize the gradient of the perceptron cost function and the position of pattern hyperplanes in weight space, so that the number of misclassified patterns never increases during training. Key features of the algorithm are the absence of free parameters and a natural termination criterion. Simulation results in pattern classification problems demonstrate the advantages of our method over other perceptron learning rules. We consider our method as a basis for future development of layer-by-layer training methods for multilayered feedforward networks with hard limiter activation functions.

References

[1] S. D. Hunt and J. R. Deller, "Selective training of feedforward artificial neural networks using matrix perturbation theory," *Neural Networks*, vol. 8, pp. 931-944, 1995.

[2] F. Rosenblatt, *Principles of Neurodynamics*. New York: Spartan, 1962.

[3] D. E. Rumelhart, J. E. Hinton and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, D. E. Rumelhart and J. L. McLelland, Eds. Cambridge, MA: MIT Press, ch. 8, 1986, pp. 318-362.

[4] B. A. Telfer and D. P. Casasent, "Minimum-cost associative processor for piecewise-hyperspherical classification," *Neural Networks,* vol. 6, pp. 1117-1130, 1993.

[5] D. J. Volper and S. E. Hampson, "Quadratic function nodes: Use, structure and training," *Neural Networks,* vol. 3, pp. 93-107, 1990.

[6] L. Bobrowski and W. Niemiro, "A method of synthesis of linear discriminant function in the case of nonseparability," *Pattern Recognition,* vol. 17, pp. 205-210, 1984.

[7] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks,* vol. 1, pp. 75-89, 1988.