Computer Science Journal of Moldova, vol.**13**, no.3(39), 2005

A Deterministic and Polynomial Modified Perceptron Algorithm

Olof Barr

Abstract

We construct a modified perceptron algorithm that is deterministic, polynomial and also as fast as previous known algorithms. The algorithm runs in time $O(mn^3 \log n \log(1/\rho))$, where m is the number of examples, n the number of dimensions and ρ is approximately the size of the margin. We also construct a non-deterministic modified perceptron algorithm running in time $O(mn^2 \log n \log(1/\rho))$.

1 A Deterministic and Polynomial Modified Perceptron Algorithm

1.1 Historical and Technological Exposition

The Perceptron Algorithm was introduced by Rosenblatt in [12] and has been well-studied by mathematicians and computer scientists since then. For convenience, we will in this paper discuss the version of the algorithm that, given a set of points (constraints) $A = \cup_i a_i$ from $\mathbb{R}^n$ finds, if any, a normal z to a hyperplane through origo such that $z \cdot a_i > 0$ for every i. (Note that we do not have any zero rows in the matrix A). This is so to say, a hyperplane through origo such that all points are on the very same side of the hyperplane.

The original algorithm can easily be described as follows:

Algorithm 1.1 *The Perceptron Algorithm*

Input: A set of points (constraints) $A = \cup_i a_i$ from $\mathbb{R}^n$.

Output: A normal z to a hyperplane such that $z \cdot a_i > 0$ for every i, if there is such a solution.

1. *Let z=0.*

2. *If there is a point $a_i \in A$ such that $z \cdot a_i \leq 0$, then $z \leftarrow z + a_i$*

3. *Repeat step 2 until no such point is found and output z.*

Obviously this algorithm will never halt if there is no solution to the problem. On the other hand Novikoff proved that if there is a solution to the problem, the algorithm will halt in a finite number of steps, even if the number of constraints is infinite.

Theorem 1.1 *The number of mistakes made by the on-line perceptron algorithm, on a set A that has a solution to the problem, is at most $(2R/\gamma)^2$, where $R = \max \|a_i\|$ and γ is the size of the margin.*

Now, since the margin can be any positive number close to zero, this upper bound of the performance does not say very much. And even though there are many results showing that the algorithm runs much faster in the "usual" case, there are constructions of constraint sets such that the behaviour of the algorithm is exponential in terms of the number of constraints [2].

1.1.1 Linear Programs

The problem solved by this Perceptron Algorithm is the homogenized form of a feasible standard form of a linear program: Find x such that $Ax \geq 0$ and $x \neq 0$. Here, A is the matrix with a_i as row i and $x = z^T$, where z is the normal of the hyperplane described above. This problem is of great importance, since it solves $\max c^T x$, $Ax \leq b$, $x \geq 0$ by iterative solving the homogenized version and performing a binary search.

Due to the importance of the problem, many different methods have been evolved to find solutions to it. To mention here, those are the simplex algorithm, the interior point method, ellipsoid methods, the perceptron algorithm and the modified perceptron algorithm.

1.1.2 The Modified Perceptron Algorithm

The first polynomial algorithm using the perceptron algorithm was created by Dunagan and Vempala in [7]. Even though the algorithm they constructed was not as fast as other algorithms mentioned above, it must be said it was a break through for the Perceptron Algorithm. After this result, the algorithm could not be counted out, it could be competitive.

Algorithm 1.2 *The Modified Perceptron Algorithm (Dunagan & Vempala)*

Input: An $m \times n$ matrix A.
Output: A point x such that $Ax \geq 0$ and $x \neq 0$.

1. *Let $B = I$, $\sigma = 1/(32n)$.*
2. *(Perceptron)*
   (a) Let x be the origin in $\mathbb{R}^n$.
   (b) Repeat at most $16n^2$ times: If there exists a row a such that $a \cdot x \leq 0$, set $x = x + \bar{a}$.
   Here $\bar{a}$ denotes the normalized vector $a/\|a\|$.
3. *If $Ax \geq 0$, then output Bx as a feasible solution and stop.*
4. *(Perceptron Improvement)*
   (a) Let x be a random unit vector in $\mathbb{R}^n$.
   (b) Repeat at most $(\ln n)/\sigma^2$ times: If there exists a row a such that $\bar{a} \cdot \bar{x} < -\sigma$, set $x \leftarrow x - (\bar{a} \cdot x)\bar{a}$. If $x = 0$, go back to step (a). (This is to assure us of not having the vector x set to zero)
   (c) If there still exists a row a such that $\bar{a} \cdot \bar{x} < -\sigma$, restart at step (a). (This takes care of the situation of a bad choice of the randomly chosen vector x)
5. *If $Ax \geq 0$, then output Bx as a feasible solution and stop.*

6. *(Rescaling)*
   Set $A \leftarrow A(I + \bar{x}\bar{x}^T)$ *and* $B \leftarrow B(I + \bar{x}\bar{x}^T)$.

7. *Go back to step 2.*

It is not evident that this algorithm will terminate. But Dunagan and Vempala prove that the margin of the Linear Program will increase in mean, when many rescalings are made inside the algorithm. This will in turn make the margin so large such that the Perceptron part of the algorithm will return a solution to the problem of the Linear Program.

Beside the size of the running time ($O(mn^4 \log n \log(1/\rho)$, where ρ is approximately the size of margin γ), the algorithm they presented was not deterministic. This meant that the result was given in the mentioned time with very high probability. This is bad, since exceeding the specified time will not always imply that no solution exists.

Dunagan and Vempala put an open question whether or not there was a deterministic version of their algorithm.

1.2 A Deterministic and Polynomial Modified Perceptron Algorithm

In this paper, we answer the above stated question in the affirmative. Also, the constructed algorithm presented has the a running time a factor $O(n)$faster than the one by Dunagan and Vempala. As a consequence of the construction, we also get a non-deterministic algorithm running a factor $O(n^2)$ faster than the algorithm constructed by Dunagan and Vempala.

1.2.1 How to Make it Deterministic

First of all we can conclude that it is due to the random choice of a unit vector inside the algorithm that makes the algorithm of Dunagan and Vempala a non-deterministic one. The question to put is if there is a way of choosing appropriate vectors so that we can keep control of the number of iterations being made inside the algorithm.

What the algorithm wants to choose is a unit vector that has an inner product of at least $1/\sqrt{n}$ with a feasible solution z to the posed problem. But since we do not know a solution to the problem, we can ask us if we can have a set V of vectors, where at least one vector $v \in V$ has the mentioned property. The answer to this is positive. $V = \cup_{i=1}^{n}\{e_i, -e_i\}$, where $\cup_{i=1}^{n}\{e_i\}$ constitutes an ON-basis for $\mathbb{R}^n$, will do according to the following proposition.

Proposition 1.1 *Let $V = \cup_{i=1}^{n}\{e_i, -e_i\}$, where $\cup_{i=1}^{n}\{e_i\}$ constitutes an ON-basis for $\mathbb{R}^n$. Now, for every unit vector $w \in \mathbb{R}^n$,*

$$\max_{v \in V} w \cdot v \geq \frac{1}{\sqrt{n}}.$$

Proof: First assume that e_i is the vector with a 1 in the ith coordinate and zero elsewhere. Now let $w = (w_1, \ldots, w_n)$. At least one w_i must have an absolute value of at least $1/\sqrt{n}$, otherwise $\|w\| < 1$. This yields that there exists at least one vector $v \in V$ such that $w \cdot v \geq 1/\sqrt{n}$ as stated above. To generalize this statement for any ON-basis, we only have to consider the rotation symmetry of $\mathbb{R}^n$. □

To make the algorithm deterministic, we will now run the algorithm in $2n$ parallel tracks. And instead of using the origin in step 2(a) and a random unit vector in step 4(a), we will use vectors from our set V and update them for each iteration in the algorithm. Since one of parallel tracks will come closer and closer to a solution for each round, this specific track will terminate in the time mentioned above. But, we are running $2n$ tracks, and the total running time will be a factor $2n$ greater than before.

In practice, this is an advantage since the algorithm will tell us how to make use of paralell processors in a practical situation, making the algorithm fast when implemented.

1.2.2 How to Speed Up the Algorithm

In order to speed up the algorithm, a deep analysis of all estimates done by Dunagan and Venpala has been done. As a result of this σ can

be enlarged to $1/(32\sqrt{n})$ and the $16n^2$ in step 2(b) can be reduced to $4n$. These alterations will speed up the process with a factor of order $O(n)$. New estimates in the margin growth causes another factor of order $O(n)$.

1.3 The Deterministic Modified Perceptron Algorithm

Now, we are ready to go into details with the topic of this paper.

Below we can study the general structure of the algorithm, breaking it up into smaller parts that will be presented further on. Important is though that we are running the algorithm in $2n$ parallel tracks. This does not imply that we have to run the algorithm on parallel processors, only that we do each step for every single track before going to the next step.

Algorithm 1.3 *The Modified Perceptron Algorithm*

Input: An $m \times n$ matrix A.

Output: A vector x such that $Ax \geq 0$ and $x \neq 0$ or "No solution exists".

1. *Initials*

   Choose $R = n$ (the dimension of the Linear Program) and put $\sigma = \frac{1}{32\sqrt{n}}$.

   Let $B = I$ and $V = \cup_{i=1}^{n}\{e_i, -e_i\}$, where $\cup_{i=1}^{n}\{e_i\}$ constitutes an ON-basis for $\mathbb{R}^n$.

2. *Wiggle Phase*

   Let $U = \emptyset$. For each vector $v \in V$, run the Wiggle Algorithm, collecting all returned corresponding vectors u in U.

   $V \leftarrow U$.

3. *Check for no solution*

   If $V = \emptyset$, then output "No solution exists" and stop.

4. *Rescaling Phase Normalize every vector in V. That is, for every $v \in V$, let $v \leftarrow \frac{v}{\|v\|}$*

   For each vector v in V (at most $2n$), together with its corresponding matrices A and B, run the Rescaling Algorithm.

5. *Perceptron Phase*

   For each vector $v \in V$, run the Perceptron Algorithm for at most R rounds with v as the initial vector.

6. *If no feasible solution was obtained in the last iteration of the algorithm, scale every vector $v \in V$ such that $\|v\| = 1$ and go back to step 2.*

Note that there are more things to show than only to describe the smaller algorithms used inside the larger structure. We have to show that they work, to calculate the complexity and to prove that the main algorithm always will return a correct answer in the time mentioned above.

First we describe the Wiggle Algorithm:

Algorithm 1.4 *Wiggle Algorithm*

Input: An $m \times n$ matrix A, a vector $v \in \mathbb{R}^n$ and a set of vectors U.

Output: One of the following three: A solution x to $Ax \geq 0$, an updated vector v that will be put in U or the empty set $\emptyset$ (also to be put in U).

1. *If $\frac{a \cdot v}{\|a\|\|v\|} < -\sigma$ for some row $a \in A$,*

   then $v \leftarrow v - \left(\frac{a}{\|a\|} \cdot v\right) \frac{a}{\|a\|}$.

2. *Repeat step 1 at most $(\log n)/\sigma^2$ times.*

3. *If $\frac{a \cdot v}{\|a\|\|v\|} \geq -\sigma$ for every row $a \in A$, then $U \leftarrow U \cup v$*

4. *If $Av \geq 0$, then output Bv as a feasible solution and stop.*

This algorithm is shown in [5], to output a vector v in at most $(\log n)/\sigma^2$ steps, if the input vector v satisfies $v \cdot z \geq 1/\sqrt{n}$, where z is a unit vector that solves $Ax \geq 0$. Thus, we have to insure us that at least one of our starting vectors in V does have this property. But this was shown in the earlier proposition presented above.

Also we must calculate the complexity of running through the Wiggle Algorithm once:

Proposition 1.2 *The number of iterations inside the Wiggle Algorithm is of order* $O(mn^2 \log n)$.

Proof: The inner loop of the algorithm requires at most one matrix-vector multiplication, time $O(mn)$, and a constant number of vector manipulations, time $O(n)$. This is repeated at most $(\log n)/\sigma^2 = 32^2 n \log n$ times. So, the overall time bound is $O(mn^2 \log n)$. □

The following rescaling procedure is a simple matrix-matrix multiplication, being of order $O(n^2)$.

Algorithm 1.5 *Rescaling Phase*
Input: a vector v *and its*
corresponding matrices A *and* B.
Output: rescaled matrices A *and* B.

1. $A \leftarrow A(I + \frac{v}{\|v\|}\frac{v}{\|v\|}^T)$

2. $B \leftarrow B(I + \frac{v}{\|v\|}\frac{v}{\|v\|}^T)$

The important thing to prove for this part, is that for at least one of our $2n$ parallel processes, the matrices will be stretched in a direction such that the margin increases in the new problem $Ax \geq 0$. The proof of this follows substantially the proof in [7], but using some other indata. The reason for letting $\rho \leq 1/(2\sqrt{n})$ in the following theorem is that if ρ would be larger, the algorithm will halt later on in the Perceptron phase.

Theorem 1.2 *Suppose* $\rho \leq 1/(2\sqrt{n})$ *and* $\sigma = 1/(32\sqrt{n})$. *Let* A' *be obtained from* A *by one iteration of the algorithm (where the problem is not solved). Let* ρ' *and* ρ *be the margins of the problems* $A'x \geq 0$ *and* $Ax \geq 0$ *respectively. Also assume that we are studying one of those processes running parallel, where* $v \cdot z \geq 1/\sqrt{n}$ *and* z *is a feasible solution, of length one, to* $Ax \geq 0$. *Then* $\rho' \geq (1 + \frac{1}{6})\rho$.

Proof: Let a_i, $i = 1, \ldots, m$ be the rows of A at the beginning of some iteration, for one of the parallel processes having $v \cdot z \geq 1/\sqrt{n}$. (Below we drop the index i, and usually denote a row a_i only with a). Let z be the unit vector satisfying $\rho = \min_i \frac{a}{\|a\|} \cdot z$, and let $\sigma_i = \frac{a}{\|a\|} \cdot v$. After the wiggle phase, we get a vector v such that $\frac{a}{\|a\|} \cdot v = \sigma_i \geq -\sigma$ for every i.

As described in the algorithm, let A' be the matrix obtained after the rescaling step, i.e. $a'_i = a_i + k(a_i \cdot v)v$. Finally define $z' = z + \alpha(z \cdot v)v$, where

$$2\alpha + 1 = \rho\sqrt{n}$$

or to put it another way

$$\alpha = (\rho\sqrt{n} - 1)/2.$$

Even though z' might not be an optimal choice, it is enough to consider this one element to lower bound ρ'. We have $\rho' \geq \min_j \frac{a'}{\|a'\|} \cdot \frac{z'}{\|z'\|}$.

We will first prove that $\frac{a'}{\|a'\|} \cdot z'$ cannot be too small.

$$\frac{a'}{\|a'\|} \cdot z' = \frac{\frac{a}{\|a\|} + (\frac{a}{\|a\|} \cdot v)v}{\|\frac{a}{\|a\|} + (\frac{a}{\|a\|} \cdot v)v\|} \cdot z' = \frac{[\frac{a}{\|a\|} + (\frac{a}{\|a\|} \cdot v)v][z + \alpha(z \cdot v)v]}{\sqrt{1 + 3(\frac{a}{\|a\|} \cdot v)^2}},$$

since the vector v is normalized before the rescaling is done. Now, in the case of a positive σ_i,

$$= \frac{\rho + \sigma_i(z \cdot v)(1 + 2\alpha)}{\sqrt{1 + 3\sigma_i^2}} \geq \rho\frac{1 + \sigma_i(z \cdot v)\sqrt{n}}{\sqrt{1 + 3\sigma_i^2}} \geq \rho\frac{1 - \sigma}{\sqrt{1 + 3\sigma^2}},$$

where the last inequality follows from that $(z \cdot v) \geq 1/\sqrt{n}$ and $\sigma_i \in [-\sigma, 1]$, with some hard work using the method of Lagrange. On the other hand, if σ_i is negative we get that the expression is at least

$$\frac{31}{32\sqrt{1+3\sigma^2}}.$$

Now we are about to bound $\|z'\|$ from above, and for convenience we study the square of it, $\|z'\|^2$. We know that

$$\|z'\|^2 = \|z + \alpha(z \cdot v)v\|^2 = 1 + (\alpha^2 + 2\alpha)(v \cdot z)^2$$

Inserting our known $\alpha = (\rho\sqrt{n} - 1)/2$ we get that

$$\|z'\|^2 = 1 + ((\alpha+1)^2 - 1)(v \cdot z)^2 \leq 1 - \frac{7}{16} = \frac{9}{16}$$

since $\alpha + 1 \leq 3/4$.

Using the identity

$$\frac{1}{\sqrt{1+\beta}} \geq 1 - \frac{\beta}{2}$$

for $\beta \in (-1, 1)$, we find that the total estimate for the new margin is

$$\rho' \geq \rho\left(1 - \frac{1}{2^5\sqrt{n}}\right)\left(1 - \frac{3}{2^{11}n}\right)\left(\frac{4}{3}\right) \geq \rho\left(\frac{7}{6}\right)$$

when σ_i is positive. Otherwise, when σ_i is negative we get that

$$\rho' \geq \rho\left(\frac{31}{32}\right)\left(1 - \frac{3}{2^{11}n}\right)\left(\frac{4}{3}\right) \geq \rho\left(\frac{7}{6}\right).$$

This estimate will be enough to fulfill the demand we have on the margin to increase with a certain proportion, such that we also guarantee a convergence in the case of the non-deterministic algorithm. For further details, see [3] and [7]. □

Now we describe the classical Perceptron Algorithm, but reduced to at most n steps, being an important component of the modified algorithm.

Algorithm 1.6 *Perceptron Algorithm*
Input: A starting vector v from V.
Output: A feasible solution Bv or a new updated vector v.

1. *If there is a row a in A such that $v \cdot a \leq 0$, then $v \leftarrow v + a/\|a\|$.*
2. *If $Av \geq 0$, then output Bv as a feasible solution and stop.*
3. *Repeat the two steps above at most $R = 4n$ times.*

The behaviour of this algorithm is well-studied, and we know from, for example, [6] that it produces a feasible solution if the problem has a margin ρ of size at least $1/\sqrt{R} = 1/(2\sqrt{n})$. Also we can conclude that:

Proposition 1.3 *The number of iterations inside the Perceptron Algorithm is of order $O(mn^2)$.*

Proof: The algorithm will perform at most one matrix-vector multiplication in its inner loop (made at time $O(mn)$) and a constant number of vector manipulations (in time $O(n)$). This is done at most $2n$ times. So we get $O(mn^2)$. □

Lemma 1.1 *The number of times we repeat step 2-5 in Algorithm 1.1 is at most of order $O(n \log(1/\rho))$.*

Proof: As we have seen above, at least one of our $2n$ parallel processes will start with a vector v satisfying $v \cdot z \geq 1/\sqrt{n}$ where z is a feasible unit vector. So, after the wiggling phase, the resulting vector will be a proper direction for rescaling, this enlarges the radius ρ with a factor of size at least $(1+1/6)$. But since the perceptron stage will terminate, yielding a feasible solution if $\rho \geq 1/(2\sqrt{n})$, we know that our algorithm will terminate after k proper rescalings when

$$\rho \left(1 + \frac{1}{6}\right)^k \geq \frac{1}{2\sqrt{n}}.$$

This yields that $k = O(\log(1/\rho))$ □

Summing up the information we have got, we get the complexity of the algorithm in total.

Theorem 1.3 *The modified Perceptron Algorithm returns an answer in time* $O(mn^3 \log n \log(1/\rho))$.

Proof: The algorithm runs in $2n$ parallel processes. The wiggle algorithm runs for at most $2^{10} n \log n$ times, each round taking time $O(mn)$. The rescale process takes $O(n^2)$ and the Perceptron algorithm runs for at most n times, each round taking time $O(mn)$. All these three stages are repeated at most $O(\log(1/\rho))$ times. So we get that the total time is

$$O\left(2nO\left(\log\left(\frac{1}{\rho}\right)\right)(2^{10}mn^2 \log n + O(n^2) + mn^2)\right) =$$

$$= O\left(mn^3 \log n \log\left(\frac{1}{\rho}\right)\right)$$

□

1.3.1 A Fast Non-Deterministic Polynomial Modified Perceptron

The results in the previous section can be used to strengthen the results made by Dunagan and Vempala in [7]. In the non-deterministic case we are not longer in need for our $2n$ parallel processes anymore. Now, taking away the condition about having a deterministic process, we can speed it up a factor $2n$.

In general, one could follow the proof made by Dunagan and Vempala in [7] to prove the behaviour of the non-deterministic algorithm. The only changes made are the size of R, σ and k inside the algorithm.

But we do want to point out a statement made in the article: a statement that says that the probability of two random unit vectors have inner product at least $1/\sqrt{n}$ is at least $1/8$ can be shown by a standard computation. The statement is true, but we have not found a standard argument proving this statement. A detailed analysis can be

found in [3] showing that the probability is at least $(1-\mathrm{erf}(1/\sqrt{2})/2 \geq 1/8$ where $\mathrm{erf}(x)$ is the errorfunction

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{dt}.$$

Anyhow, we get the following:

Corollary 1.1 *There is a polynomial non-deterministic modified perceptron algorithm that terminates in time* $O(mn^2 \log n \log(1/\rho))$.

References

[1] N. Alon and A. Naor, Approximating the Cut-Norm via Grothendieck's Inequality, *submitted.* Available at http://www.math.tau.ac.il/~nogaa/PDFS/publications.html

[2] M. Anthony and J. Shawe-Taylor, Using the Perceptron Algorithm to Find Consistent Hypotheses *Combinatorics, Probability and Computing* (1993) 2:pp. 385-387.

[3] O. Barr and O. Wigelius, *New Estimates Correcting an Earlier Proof of the Perceptron Algorithm to be Polynomial*, ISSN 1403-9338, LUTFMA-5041-2004.

[4] A. Blum and J. Dunagan, Smoothed Analysis of the Perceptron Algorithm for Linear Programming, in *SODA'02*, 2002; 905–914.

[5] A. Blum, A. Frieze, R. Kannan and S. Vempala, A Polynomial-time Algorithm for Learning Noisy Linear Threshold Functions *Algorithmica,* **22**(1/2):35–52, 1997.

[6] N. Cristianini and J. Shawe-Taylor, *Support Vector Machines,* Cambridge, 2000.

[7] J. Dunagan and S. Vempala, *A Polynomial-time Rescaling Algorithm for Solving Linear Programs,* Microsoft Research, Redmond.

[8] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization,* Springer Verlag, Berlin Heidelberg, 1988.

[9] D. G. Luenberger, *Linear and Nonlinear Programming,* Addison-Wesley, Reading, Massachusetts, 1984.

[10] M. E. Muller, A Note on a Method for Generating Points Uniformly on N-Dimensional Spheres *Comm. Assoc. Comput. Mach.***2**, 19-20, 1959.

[11] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming* Studies in Applied Mathematics, Vol. 13, Philadelphia, 1994.

[12] F. Rosenblatt, *Principles of Neurodynamics.* Spartan Books, 1962.

[13] N. Z. Shor, *Minimization Methods for Non-Differentiable Functions,* Springer-Verlag, Berlin, Heidelberg, 1985.

[14] D. Spielman and S. Teng, Smoothed Analysis of Termination of Linear Programming Algorithms, in *Mathematical Programming, Series B,* Vol. **97**, 2003

[15] D. Spielman and S. Teng, Smoothed Analysis: Why The Simplex Algorithm Usually Takes Polynomial Time, in *Proc. of the 33rd ACM Symposium on the Theory of Computing,*, 296–305, 2001.

Olof Barr,

Received December 9, 2005

Centre for Mathematical Sciences
Lund University
Sweden
E–mail: *barr@maths.lth.se*