An Example with Decreasing Largest Inscribed Ball for Deterministic Rescaling Algorithms

Dan Li, Tamás Terlaky

Abstract Recently, Peña and Sohèili presented a deterministic rescaling perceptron algorithm and proved that it solves a feasible perceptron problem in $O(m^2n^2\log(\rho^{-1}))$ perceptron update steps, where ρ is the radius of the largest inscribed ball. The original stochastic rescaling perceptron algorithm of Dunagan and Vempala is based on systematic increase of ρ, while the proof of Peña and Sohèili is based on the increase of the volume of a so-called cap. In this note we present a perceptron example to show that with this deterministic rescaling method, ρ may decrease after one rescaling step.

Furthermore, inspired by our previous work on the duality relationship between the perceptron and the von Neumann algorithms, we propose a deterministic rescaling von Neumann algorithm which is a direct transformation of the deterministic rescaling perceptron algorithm. Though the complexity of this algorithm is not proved yet, we show by constructing a von Neumann example that ρ does not increase monotonically for the deterministic rescaling von Neumann algorithm either. The von Neumann example serves as the foundation of the perceptron example. This example also shows that proving the complexity of the rescaling von Neumann algorithm cannot be based on monotonic expansion of ρ.

At last, we present computational results of the deterministic rescaling von Neumann algorithm. The results show that the performance of the rescaling algorithm is improved compared with the original von Neumann algorithm when solving the test problems.

Keywords Rescaling perceptron algorithm, the largest inscribed ball, von Neumann algorithm, linear feasibility problem

1 Introduction

Dunagan and Vempala [6] proposed a stochastic rescaling perceptron algorithm. It solves linear feasibility problems in the form of

$$(PPb): \quad {}^{p}\!A^T y \geq 0,\ y \neq 0,$$

where ${}^{p}\!A \in \mathbb{R}^{m\times n}$ with its column vectors ${}^{p}a_1, {}^{p}a_2, \ldots, {}^{p}a_n \in \mathbb{R}^m$ and $y \in \mathbb{R}^m$. Without loss of generality, we may assume that $\|{}^{p}a_i\|_2 = 1$ for all $i = 1, 2, \ldots, n$. Problem (PPb) is also called perceptron problem. In order to prove the effectiveness of rescaling and the algorithm's complexity, Dunagan and Vempala showed that with high probability, ${}^{p}\rho$ increases by at lease a fixed ratio after each rescaling, where ${}^{p}\rho$ is the radius of the largest inscribed ball contained in the feasible cone and centered on the unit sphere. The radius ${}^{p}\rho$ is also called the width of the feasible cone [13]. Since the complexity of the classical perceptron algorithm [14, 12] is $O\left({}^{p}\rho^{-2}\right)$ perceptron updates, as ${}^{p}\rho$ increases periodically, the perceptron algorithm is accelerated to $O\left(m^2 \log m \log\left({}^{p}\rho^{-1}\right)\right)$ with high probability. To make the complexity for different algorithms comparable, all the complexity is measured in the perceptron update steps. Due to the fact that ${}^{p}\rho$ is guaranteed to increase with a certain probability, this rescaling perceptron algorithm has the property of randomness.

Recently, Peña and Sohèili [13] proposed a deterministic rescaling perceptron algorithm with complexity of $O\left(m^2 n^2 \log\left({}^{p}\rho^{-1}\right)\right)$ iteration complexity. The deterministic rescaling perceptron algorithm applies linear transformation

$$ {}^{p}\!A' = \left(I - \frac{1}{2}{}^{p}a_j {}^{p}a_j^T\right){}^{p}\!A \tag{1} $$

on matrix ${}^{p}\!A$ at each rescaling step, where ${}^{p}a_j$ is one of the column vectors, called a rescaling vector. Compare these two versions of the rescaling perceptron algorithms. The one by Peña and Sohèili has a weaker but deterministic complexity.

To prove the complexity of the deterministic rescaling perceptron algorithm, Peña and Sohèili did not mention how ${}^{p}\rho$ changes with rescaling. Instead, they utilized a spherical cap, which is contained in the intersection of the feasible cone and the unit sphere. In their paper, they showed that the volume of this spherical cap increases monotonically after each rescaling. Since the spherical cap is always contained in a hemisphere, they concluded that the algorithm will terminate in finite number of rescaling steps.

After comparing their proofs, the following question naturally arises: does ${}^{p}\rho$ also increase monotonically in the deterministic rescaling perceptron algorithm? In order to answer this question, we construct an example to show that ${}^{p}\rho$ may be smaller after one rescaling step.

The von Neumann problem [4] is a linear feasibility problem with the following form.

$$ \begin{aligned} (vNPb): \quad {}^{v}\!Ax &= 0, \\ e^T x &= 1, \\ x &\geq 0, \end{aligned} $$

where ${}^{v}\!A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, and $e \in \mathbb{R}^n$ is the vector of all ones. As the alternative system to the perceptron problem, the von Neumann problem can be solved by the von Neumann algorithm [3, 4] published by Dantzig. Since the two problems (PPb) and (vNPb) are a pair of alternative systems to each other, our previous work [11] revealed the duality relationship between the perceptron and the von Neumann algorithms. Based on the duality relationship, variants of the perceptron algorithm can be interpreted as variants of the von Neumann algorithm and vice versa, as well as one can transit the complexity results from one family to the other. The complexity of the von Neumann algorithm linearly depends on the multiplicative inverse of ${}^{v}\!\rho^2$ [7], where ${}^{v}\!\rho$ is the radius of the largest inscribed ball in the convex hull of ${}^{v}\!A$ centered at the origin. Since ${}^{v}\!\rho$ could be exponential in the input size of the problem, the von Neumann algorithm is not a polynomial algorithm. For the purpose of improving its complexity and hopefully to obtain a polynomial version, we propose a deterministic rescaling von Neumann algorithm. When trying to prove its complexity, we raise the analogous question for the deterministic rescaling von Neumann algorithm to the one we have asked for the deterministic rescaling perceptron algorithm: will ${}^{v}\!\rho$ increase monotonically with rescaling? We construct a von Neumann example in $\mathbb{R}^3$ to give a negative answer. Since it is visualized to construct a von Neumann example in $\mathbb{R}^3$, the perceptron example is actually derived from this von Neumann example. We present the major steps of transforming the von Neumann example to the perceptron example at the end of this paper.

We use notations following several rules: (1) the superscript on the left indicates which problem the notation is used for, e.g., p for the perceptron problem and v for the von Neumann problem; (2) prime denotes the corresponding notation after rescaling; (3) the superscript on the right is either the iteration counter or an arithmetic operation depending on the context; (4) positive number subscript is the index of vectors, points, or coordinates.

The paper is structured as follows. In Section 2, we present a perceptron example to show that in the deterministic rescaling perceptron algorithm, ${}^{p}\!\rho$ may not increase after one rescaling step. The construction of a von Neumann example is introduced in Section 3 and its validity is verified in Section 3.3. In Section 4, we present the procedure of deriving the perceptron example from the von Neumann example.

2 A decreasing ball example for rescaling perceptron algorithms

Since this example is derived from a von Neumann example which is in the dual perspective, we will present and verify the example in this section first, then introduce how it is developed in a later section.

2.1 The example

The example is as follows:

$$(Eg.p) \quad {}^{p}\!A^T y \geq 0, \;\; y \neq 0,$$

where ${}^{p}\!A \in \mathbb{R}^{4\times 9}$ and

$${}^{p}\!A^T = \begin{bmatrix} -0.000003029342674 & -2.019699173751262 & -0.000004999001640 & 0.020000000000000 \\ -0.019798999974999 & 0.019997999899990 & -0.019997999899990 & 0.020000000000000 \\ 0.001431631766736 & 0.019997999899990 & 0 & 0.020000000000000 \\ -0.000003028146773 & -1.973134679085590 & 0.183149852715338 & 0.020000000000000 \\ 0.019737351052173 & 0.000000950063128 & 0.020002002579065 & 0.020000000000000 \\ -0.052561097586474 & 1.592477159015729 & -0.091573429520343 & 0.024000000000000 \\ -0.052561703455009 & 1.188537324265476 & -0.091574429320671 & 0.028000000000000 \\ -0.052560491717939 & 1.996416993765981 & -0.091572429720015 & 0.020000000000000 \\ 0.050728859951203 & 1.996416993765981 & -0.091572429720015 & 0.020000000000000 \end{bmatrix}.$$

Each column vector ${}^{p}a_i$ of ${}^{p}\!A$ defines a hyperplane in $\mathbb{R}^4$ and there are nine hyperplanes in total. We have the following claims.

Claim 1. *For the perceptron problem (Eg.p), the radius of the largest inscribed ball ${}^{p}\rho$ will decrease if the problem is rescaled by (1) using ${}^{p}a_1$ as the rescaling vector.*

Claim 2. *The perceptron phase of the rescaling perceptron algorithm [13] will identify column ${}^{p}a_1$ as the rescaling vector when the algorithm is applied to problem (Eg.p).*

2.2 Verification

In order to verify these two claims, we implement the example and the algorithm in MATLAB using IEEE double precision arithmetic. The unit roundoff error is $O(10^{-16})$.

We have the following observations. The initial ${}^{p}\rho = 0.00999988$. After running the rescaling perceptron algorithm [13], the perceptron phase does not solve ${}^{p}\!A^T y \geq 0, y \neq 0$. It identifies column ${}^{p}a_1$ as the rescaling vector, which is nearly perpendicular to the feasible cone. In the rescaling phase, ${}^{p}a_1$ is used to rescale the matrix ${}^{p}\!A$. The radius of the largest inscribed ball after rescaling becomes ${}^{p}\rho' = 0.00961856$, which yields a factor of $O(10^{-4})$ decrease.

Verify Claim 1: the correctness of ${}^p\rho$ and ${}^p\rho'$ are checked first by solving

$$ {}^p\rho = \max_{\|y\|=1, {}^pA^T y \geq 0} \min_i \{ {}^pa_i^T y \} \tag{2} $$

in MATLAB using the *fminmax* function. The *fminmax* function uses a Sequential Quadratic Programming method [2] and might only return a local optimal solution. To clarify the situation, we also verify the results by the following steps.

Step 1. Identify the hyperplanes that touch/support the current ball (${}^p\rho$ or ${}^p\rho'$).

Step 2. Project the normal vectors of the hyperplanes found in Step 1 to a three dimensional subspace. Denote these three-dimension vectors as ${}^la_i$.

Step 3. Employ Dantzig's method [3] to solve the von Neumann problem

$$\begin{aligned} {}^lAx &= 0, \\ e^T x &= 1, \\ x &\geq 0, \end{aligned}$$

where ${}^lA$ is composed by the vectors ${}^la_i$ as its columns. If an exact solution is found, then this von Neumann problem is feasible, which proves that there is no direction in which the ball would grow. Dantzig's method yields to run the von Neumann algorithm multiple times and solve a linear equation system to obtain an exact solution to the von Neumann problem. The von Neumann algorithm is presented as the von Neumann phase of Algorithm 1 in Section 3.1. The main arithmetical operations in the process of verification involve vector normalization, matrix-vector multiplication, and solving linear equation systems. At each iteration of the von Neumann algorithm, the column vectors ${}^la_s$ which has the largest angle with the current iterate ${}^lAx^k$ is chosen for update, where k is the iteration counter. The inner product values of ${}^la_i^T({}^lAx^k)$ are compared for all i. The minimal difference between ${}^la_s^T({}^lAx^k)$ and all the other ${}^la_i^T({}^lAx^k)$ values is $O(10^{-5})$ versus the numerical error is $O(10^{-16})$ in the double precision arithmetic. Thus, we recognize that the vectors ${}^la_s$ are chosen correctly due to sufficient separation between the vectors. Regarding solving the linear equation systems, since the systems for our example are 4×4 dimensional, we use decomposition methods to solve them. Both LU and QR factorizations are applied to test our results. Though LU factorization is commonly used and needs less computation, QR factorization is more reliable in numerical computations. The accuracy of QR factorization is enough for most purposes [15]. The results of our experiment show that the values of ${}^p\rho$ and ${}^p\rho'$ are consistent while using different methods and factorizations. Therefore, executing Steps 1-3, we verify that ${}^{p}\rho = 0.00999988$ and ${}^{p}\rho' = 0.00961856$ are the radius of the largest inscribed balls before and after rescaling, respectively. Rescaling using ${}^{p}a_1$ makes the ball shrink. which verifies Claim 1.

Verify Claim 2: we have already noticed that the perceptron phase of the rescaling algorithm is actually the same as the perceptron algorithm. The minimal difference between ${}^{p}a_1^T({}^{p}\!Ax^k)$ and all the other ${}^{p}a_i^T({}^{p}\!Ax^k)$ values is in the order of $O(10^{-5})$, which is much larger than the numerical error $O(10^{-16})$. Therefore, the vector ${}^{p}a_1$ is chosen correctly as the rescaling vector after running the rescaling perceptron algorithm. Claim 2 is verified.

3 The von Neumann side

Paper [11] on the duality between the perceptron and the von Neumann algorithms discusses the duality relationship between these two algorithms; and consequently interpreted variants of the perceptron algorithm as variants of the von Neumann algorithm and vice versa. This relationship leads us to formalize an analogous deterministic rescaling von Neumann algorithm according to the deterministic rescaling perceptron algorithm [13].

3.1 A deterministic rescaling von Neumann algorithm

Recall that as the dual [4, 11] of the perceptron problem (PPb), the von Neumann problem is in the form of (vNPb). Without loss of generality [1], we can assume that matrix ${}^{v}\!A$ has the same properties as ${}^{p}\!A$ in problem (PPb). For a pair of dual problems we have ${}^{v}\!A = {}^{p}\!A$, and they are usually denoted by the same letter A. However, in this paper, we construct two different examples, therefore superscripts are used for the purpose of clarification. For convenience, we first introduce the following notation.

- $\mathrm{conv}({}^{v}\!A)$ – the convex hull of all column vectors ${}^{v}a_i$ of ${}^{v}\!A$.
- ${}^{v}\rho$ – the radius of the largest inscribed ball in $\mathrm{conv}({}^{v}\!A)$ and centered at the origin, i.e.,

$$ {}^{v}\rho = \inf\{\|h\| : h \in \partial(\mathrm{conv}({}^{v}\!A))\}. \tag{3} $$

- $x(y) := \mathrm{argmin}\ \{y^T\, {}^{v}\!Ax \mid e^T x = 1, x \geq 0\}$.

Therefore, we have ${}^{v}\!Ax(y) = {}^{v}a_s$ if and only if ${}^{v}a_s^T y = \min\ \{{}^{v}a_i^T y \mid i = 1, \cdots, n\}$.

3.1.1 The deterministic rescaling von Neumann algorithm

Analogous to the rescaling perceptron algorithm, we propose the following rescaling variant of the von Neumann algorithm.

Algorithm 1. ***The Deterministic Rescaling von Neumann Algorithm***

Let $N = 6nm^2$, $D = I$, and let $t = 0$.

I. **The von Neumann Phase** [3, 4] (**Run** the von Neumann algorithm for N iterations)

Choose any $x^0 \in \Delta_n$.

Let $b^0 = {}^v\!Ax^0$ and $k = 0$.

For $k = 0, 1, \cdots, N-1$

1. **If** $\|b^k\| \leq \frac{\epsilon}{2^t}$, then STOP, return

$$x^* = \frac{Dx^k}{\sum_{i=1}^{n}(d_i x_i^k)} \tag{4}$$

as an ϵ-solution, where x_i^k is the i-th coordinate of x^k and d_i is the i-th diagonal entry of D.

2. **Else**, find ${}^v a_s$ which makes the largest angle with the vector b^k, i.e., ${}^v a_s = {}^v\!Ax(b^k)$.
3. Let $\nu^k = {}^v a_s^T b^k$.

   If $\nu^k > 0$, then STOP, problem (vNpb) is infeasible.
4. **Else**, let e_s be the unit vector corresponding to index s. Let

$$\begin{aligned} \lambda &= \frac{1-\nu^k}{\|b^k\|^2 - 2\nu^k + 1}, \\ x^{k+1} &= \lambda x^k + (1-\lambda)e_s, \\ b^{k+1} &= {}^v\!Ax^{k+1}, \\ k &= k+1. \end{aligned}$$

End For

II. **The Rescaling Phase**

Let $j = \underset{i=1,\cdots,n}{\operatorname{argmax}} \{e_i^T x^N\}$.

Utilize ${}^v a_j$ as the rescaling vector, then formula (1) yields

$${}^v\!A = \left(I - \frac{1}{2}{}^v a_j {}^v a_j^T\right){}^v\!A. \tag{5}$$

Let

$$D = D\text{diag}\left(\frac{1}{\|{}^v a_1\|}, \frac{1}{\|{}^v a_2\|}, \cdots, \frac{1}{\|{}^v a_n\|}\right),$$

where $\text{diag}(d_1, d_2, \cdots, d_n)$ means an $n \times n$ diagonal matrix whose diagonal entries are $d_1, d_2, \cdots, d_n$.

Normalize each column of ${}^v\!A$ back to the unite sphere and let $t = t + 1$.

III. Go back to I, to the von Neumann Phase.

Polynomial complexity of this algorithm need to be proved. So far could not complete this project. In order to get closer to this result, we ask the same question as for the rescaling perceptron algorithm: can the complexity be proved based on the increase of ${}^v\rho$ as was done in the proof of the rescaling perceptron algorithm by Dunagan and Vempala [6]? Or else, analogous to the deterministic rescaling perceptron algorithm, is it possible to identify some increasing cap? Towards answering these questions, we construct an example of the von Neumann problem in the next section. This example not only shows that ${}^v\rho$ is not going to increase monotonically after each rescaling, but also helps us to generate an analogous perceptron example as presented in Section 2.

3.1.2 The precision of solutions

Before introducing the example of the von Neumann problem, we first discuss how rescaling steps effect the precision of a solution.

Lemma 1. *Run Algorithm 1 on a von Neumann problem (vNPb). Assume that starting from this original von Neumann problem, the algorithm has done t times rescaling steps (rescaling phase) and the current iterator in the von Neumann phase is b^k. If $\|b^k\| \le \frac{\epsilon}{2^t}$, then x^* calculated by (4) is an ϵ-solution to the original von Neumann problem, i.e., $\|{}^v\!Ax^*\| \le \epsilon$.*

Proof. For one single rescaling step, the matrix ${}^v\!A$ is rescaled by formula (5) and then each column is normalized back to the unit sphere. Let $B = I - \frac{1}{2}{}^v a_j {}^v a_j^T$ and $D = diag\left(\frac{1}{\|{}^v a_1'\|}, \frac{1}{\|{}^v a_2'\|}, \cdots, \frac{1}{\|{}^v a_n'\|}\right)$. We have ${}^v\!A' = B{}^v\!AD$, where ${}^v\!A'$ is the matrix after rescaling. Assume that after rescaling $\|{}^v\!A'x\| = \|b\| \le \epsilon$ and x is on the unit simplex. The matrix B is invertible and its inverse can be computed according to the Sherman-Morrison formula [8]

$$B^{-1} = (I - \frac{1}{2}{}^v a_j {}^v a_j^T)^{-1} = I + {}^v a_j {}^v a_j^T.$$

Since ${}^v\!A'x = B{}^v\!ADx = b$, we have

$$\|{}^v\!ADx\| = \|B^{-1}b\| = \|(I + {}^v a_j {}^v a_j^T)b\| \le \|b\| + \|({}^v a_j {}^v a_j^T)b\| \le 2\|b\| \le 2\epsilon. \tag{6}$$

It means that x is a solution of $\|{}^v\!ADx\| \le 2\epsilon$. In order to recover a solution for the original problem, we need to bound $\|{}^v\!Ax^*\|$ above. Notice that x^* is also on the unit simplex and

$$\|{}^v\!Ax^*\| = \left\|\sum_{i=1}^{n} {}^v a_i x_i^*\right\| = \left\|\frac{\sum_{i=1}^{n} {}^v a_i d_i x_i}{\sum_{i=1}^{n}(d_i x_i)}\right\| = \frac{\|\sum_{i=1}^{n} {}^v a_i d_i x_i\|}{\sum_{i=1}^{n}(d_i x_i)}. \tag{7}$$

Since we also have the fact that

$$\frac{1}{d_i} = \|{}^v a_i'\| = \left\|{}^v a_i - \frac{1}{2}({}^v a_i^T {}^v a_j){}^v a_j\right\| = \sqrt{1 - \frac{3}{4}\|{}^v a_i^T {}^v a_j\|} \le 1, \tag{8}$$

which shows that rescaling always shrinks the length of column vectors of ${}^v\!A$. Combine (4), (7), and (8), we have after one rescaling step

$$\|{}^v\!Ax^*\| = \frac{\|{}^v\!ADx\|}{\sum_{i=1}^{n}(d_i x_i)} \le \|{}^v\!ADx\| \le 2\epsilon.$$

Therefore, ϵ needs to be reduced by a factor $\frac{1}{2}$ after each rescaling phase in order to keep the final solution x^* as an ϵ-solution to the original problem. If the total number of calling the rescaling phase is t, then in the worst case we need to reduce ϵ to $\frac{\epsilon}{2^t}$. This lemma is proved. □

3.2 Construction of the matrix ${}^v\!A$

For an example that ${}^v\rho$ is not increasing monotonically, the constraint matrix ${}^v\!A$ has to satisfy the following properties.

Property 1. *Among all column vectors ${}^v a_i$, there is at least one ${}^v a_j$ such that after applying (5), ${}^v\rho' < {}^v\rho$.*

Property 2. *After running the von Neumann algorithm, ${}^v a_j$ has the largest weight in the returned linear combination, i.e., the largest coordinate of x is corresponding to ${}^v a_j$.*

In order to obtain these two properties, the example is generated according to the following idea. First, create an initial convex hull with a known ${}^v\rho_0$, where ${}^v\rho_0$ is a small positive number. Second, identify vectors ${}^v a_j$ from the columns of ${}^v\!A$ which shrink ${}^v\rho$ after rescaling. If no such column vector exist, then add new columns to ${}^v\!A$. As a result, matrix A satisfies Property 1. At last, if ${}^v a_j$ obtained in the previous step does not satisfy Property 2, then add new perturbed points around ${}^v a_i$ which have larger weight after running the von Neumann algorithm but would increase ${}^v\rho$ with rescaling. The function of these new perturbed points is to introduce perturbation by creating more small facets around the corner of those ${}^{v}a_i$ and evenly share(distribute) the large weight when running the von Neumann algorithm, and consequently make Property 2 holds for ${}^{v}a_j$.

We construct an example ${}^{v}A \in \mathbb{R}^3$ with 13 column vectors. Each column vector represents a point on the unit sphere. Let $\epsilon = 0.01$, which gives the initial ${}^{v}\rho_0$. Let $\alpha_1 = \sqrt{1-2\epsilon^2}$ to simplify expressions. First, we construct a symmetric convex hull defined by eight points(columns) as follows:

$$[p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8] = \begin{bmatrix} \alpha_1 & \alpha_1 & \alpha_1 & \alpha_1 & -\alpha_1 & -\alpha_1 & -\alpha_1 & -\alpha_1 \\ \epsilon & \epsilon & -\epsilon & -\epsilon & \epsilon & \epsilon & -\epsilon & -\epsilon \\ \epsilon & -\epsilon & \epsilon & -\epsilon & \epsilon & -\epsilon & \epsilon & -\epsilon \end{bmatrix}.$$

These points are symmetrically distributed on four hyperplanes. The distances between the origin and these four hyperplanes are all equal to $\epsilon = 0.01$. Figure 1 shows the positions of these initial points. For better illustration, the distances in Figure 1 are not drawn to scale. The unit sphere is presented for scale, while the four sub-circles are pushed much further away from the origin. The real distance is much smaller. We call these eight points major points.

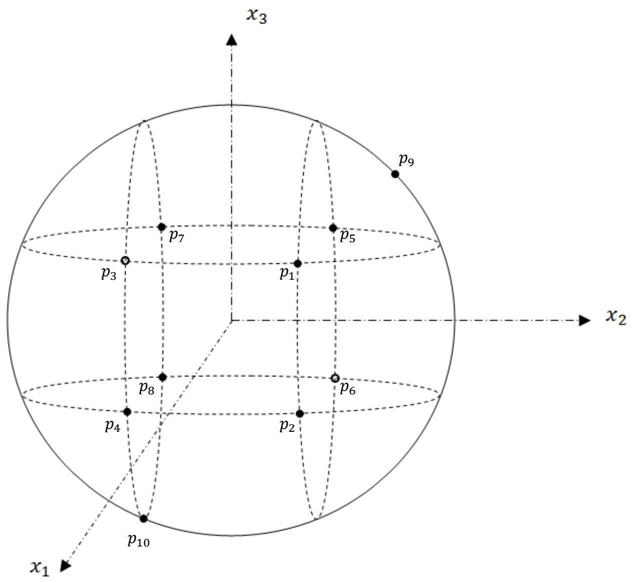


Figure 1: Illustration of initial major points, p_9, and p_{10}.

In order to obtain a rescale vector ${}^{v}a_j$, we add two points

$$[p_9, p_{10}] = \begin{bmatrix} 0 & 0 \\ \sqrt{1 - \left(\frac{2\epsilon}{3}\right)^2} & -\epsilon \\ \frac{2\epsilon}{3} & -\sqrt{1-\epsilon^2} \end{bmatrix}.$$

Figure 1 also illustrates these two points. Computational experiment shows that with points p_9 and p_{10}, ${}^{v}\rho$ will decrease after rescaling by p_9. However, after running the von Neumann algorithm, p_9 does not take the largest weight. Therefore, we need more points (columns).

The goal of adding new points is to decrease large weights on other points so that after running the von Neumann algorithm p_9 has the largest weight. A point with larger weight indicates that the point has been used more for updating the iterate. Thus, after identifying those major points with large weight, we consider to add perturbed points near them. The perturbed points will disturb the update process so that instead of the major points the perturbed points are used to update at some iterations. As a result, the perturbed points share weight with the major points. To prevent that the new perturbed points are dominated, which means the new perturbed points take all the weights from the major points, the perturbation should be small enough compared to the distance among major points, which is $O(10^{-2})$. We set the magnitude of perturbation to $\delta = 10^{-6}$.

We present two methods to perturb the major point. The first method is to move the point along certain small circle on the unit sphere. We perturb point p_7 by this method to obtain p_{12} and p_{13} as follows.

$$p_{12} = \begin{bmatrix} -\sqrt{1 - 2\epsilon^2 - \frac{17}{16}\delta^2 - \frac{5}{2}\epsilon\delta} \\ -\epsilon - \delta \\ \epsilon + \frac{\delta}{4} \end{bmatrix},$$

$$p_{13} = \begin{bmatrix} -\sqrt{1 - 2\epsilon^2 - \frac{17}{16}\delta^2 - \frac{3}{2}\epsilon\delta} \\ -\epsilon - \delta \\ \epsilon - \frac{\delta}{4} \end{bmatrix}.$$

Observe that the second coordinates of p_{12} and p_{13} are more negative than the one of p_7, which means the direction of perturbation is pointing away from the initial convex hull. This is also the rule when we for the rest of perturbations. The second method to generate perturbed point is to move point along a given direction d with a step length $\delta = 10^{-6}$, then normalize it back to the unit sphere. Points p_3, p_4, and p_{12}

are perturbed by the second method to generate p_{11}, p_{14}, and p_{15} respectively.

$$p_{11} = p_3 + \delta d_3 = \begin{bmatrix} x_1 \\ -\epsilon \\ \epsilon \end{bmatrix} + \delta \begin{bmatrix} 0 \\ -4 \\ 1 \end{bmatrix},$$

$$p_{14} = p_4 + \delta d_4 = \begin{bmatrix} x_1 \\ -\epsilon \\ -\epsilon \end{bmatrix} + \delta \begin{bmatrix} -10^{-4} \\ 0 \\ -0.01 \end{bmatrix},$$

$$p_{15} = p_{12} + \delta d_{12} = \begin{bmatrix} -\sqrt{1 - 2\epsilon^2 - \frac{17}{16}\delta^2 - \frac{3}{2}\epsilon\delta} \\ -\epsilon - \delta \\ \epsilon - \frac{\delta}{4} \end{bmatrix} + \delta \begin{bmatrix} -10^{-4} \\ 0 \\ -0.01 \end{bmatrix}.$$

After normalization and rearrangement, the new perturbed points can be expressed as follows.

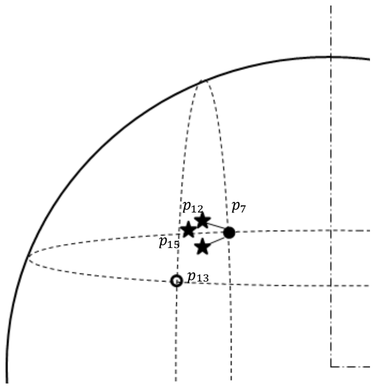
$$p_{11} = \left(1 + 10\epsilon\delta + 17\delta^2\right)^{-\frac{1}{2}} \begin{bmatrix} \alpha_1 \\ -\epsilon - 4\delta \\ \epsilon + \delta \end{bmatrix},$$

$$p_{14} = \left(1 - 2 \times 10^{-4}\delta\alpha_1 + (10^{-4} + 10^{-8})\delta^2 + 2 \times 10^{-2}\epsilon\delta\right)^{-\frac{1}{2}} \begin{bmatrix} \alpha_1 - 10^{-4}\delta \\ -\epsilon \\ -\epsilon - 10^{-2}\delta \end{bmatrix},$$

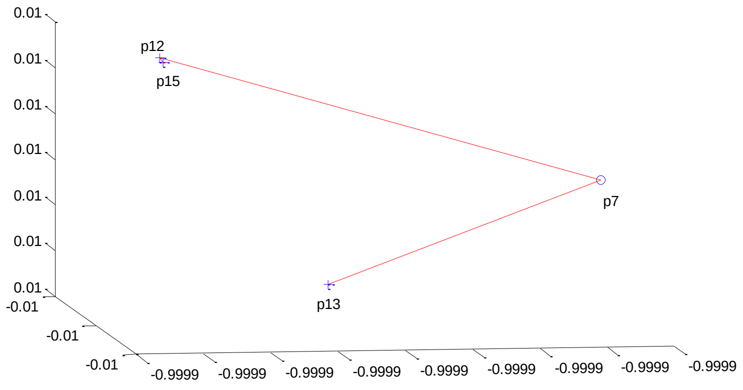
$$p_{15} = \left(1 + \left(10^{-8} - \frac{49}{1.6 \times 10^5}\right)\delta - 2 \times 10^{-2}\epsilon\delta - 2 \times 10^{-4}\delta\sqrt{1 - 2\epsilon^2 - \frac{17}{16}\delta^2 - \frac{5}{2}\epsilon\delta}\right)^{-\frac{1}{2}}$$
$$\times \begin{bmatrix} -\sqrt{1 - 2\epsilon^2 - \frac{17}{16}\delta^2 - \frac{5}{2}\epsilon\delta} + 10^{-4}\delta \\ -\epsilon - \delta \\ \epsilon + \frac{6}{25}\delta \end{bmatrix}.$$

Figure 2 illustrates the perturbation of point p_7. After removing p_3 and p_6, we obtain our example

$(Eg.vN)$: problem $(vNPb)$ with ${}^{v}\!A = [{}^{v}a_1, {}^{v}a_2, \cdots, {}^{v}a_{13}] = [p_1, p_2, p_4, p_5, p_7, p_8, \cdots, p_{15}]$. The reason that we remove points p_3 and p_6 is that p_9 will have the largest weight without them.



(a) Relative positions.



(b) Drawn to scale.

Figure 2: Illustration of the perturbations on point p_7.

We have the following two claims on this example $(Eg.vN)$ and they will be verified in Section 3.3.

Claim 3. *For the von Neumann problem (Eg.vN), the radius of the largest inscribed ball* ${}^{v}\rho$ *will decrease if the problem is rescaled by (5) using* ${}^{v}a_7$ *as the rescaling vector.*

Claim 4. *The von Neumann phase of the deterministic rescaling von Neumann algorithm will identify* ${}^{v}a_7$ *as the rescaling vector when applying the algorithm on (Eg.vN).*

3.3 Verification

In this section, we will verify Claim 3 in Section 3.3.1 – 3.3.2, and Claim 4 in Section 3.3.3 both theoretically and numerically.

3.3.1 The initial ${}^{v}\rho \geq 0.01$

To estimate ${}^{v}\rho$, we start from an initial convex hull comprised by the following ten columns p_i, where $i = 1, 2, \cdots, 10$. Figure 3 shows this initial convex hull. By construction, $p_1, p_2, \cdots, p_8$ compose a cube with an edge length of 0.02. It is easy to check that the radius of the largest inscribed ball in this initial convex hull is ${}^{v}\rho_0 = 0.01$. Then for the radius ${}^{v}\rho$, we have the following conclusions.

Lemma 2. *(a) The quantity* ${}^{v}\rho_0$ *is a lower bound of the largest inscribed ball in conv(*$[{}^{v}a_1, {}^{v}a_2, \cdots, {}^{v}a_9]$*). Then (b) it also provides a lower bound for* ${}^{v}\rho$*, i.e.,* ${}^{v}\rho_0 \leq {}^{v}\rho$*.*

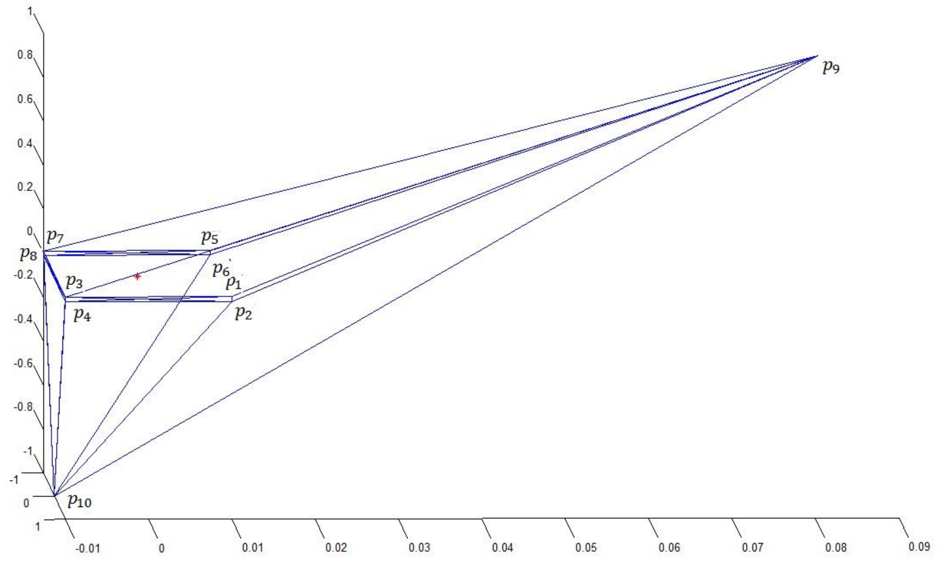


Figure 3: Illustration of initial convex hull.

Proof. The lemma can be proved by the procedure of construction, which is based on the initial convex hull shown in Figure 3.

(a) Due to the special positions of p_9 and p_{10}, removing p_6 only causes trivial changes of some inessential facets which compared with 0.01 have relatively larger distance to the origin. Thus, removing p_6 from the convex hull does not effect ${}^v\rho$. However, removing p_3 will generate a new facet defined by p_4, p_7, and p_9. This facet is closer to the origin than 0.01. Thus, we continue our constrcution with replacing p_3 by p_{11} instead of removing p_3 directly.

Recall that in the previous section, p_{11} is generated by perturbing p_3 along the direction of $[0; -4; 1]$ with a step size 10^{-6}. Point p_{11} is very close to p_3 compare the distance among the facets and the origin. In the original convex hull, the facets containing p_3 as vertex are (p_3, p_1, p_4), (p_3, p_4, p_7), (p_3, p_7, p_9), and (p_3, p_1, p_9). Replacing p_3 by p_{11} rotate facets (p_3, p_4, p_7), (p_3, p_7, p_9), and (p_3, p_1, p_9) towards outside of the original convex hull and generates new facets with p_8 and p_{10}. Since (p_3, p_4, p_7) is the facet which defines ${}^v\rho$ in the original convex hull, the rotation relaxes this constraint and makes ${}^v\rho$ larger than 0.01. The replacement also brings the facet (p_3, p_1, p_4) closer to the origin. However, the original distance from this facet to the origin is almost one and the change is in the magnitude of 10^{-6}. Thus, it does not have effect on ${}^v\rho$. Figure 4 illustrates this replacement without drawing to scale.

Therefore, after removing p_6 and replacing p_3 by p_{11}, we obtain a convex set comprised by nine columns $\operatorname{conv}([{}^va_1, {}^va_2, \cdots, {}^va_9]) = \operatorname{conv}([p_1, p_2, p_4, p_5, p_7, p_8, p_9, p_{10}, p_{11}])$ and ${}^v\rho_0$ is a lower bound for the radius of

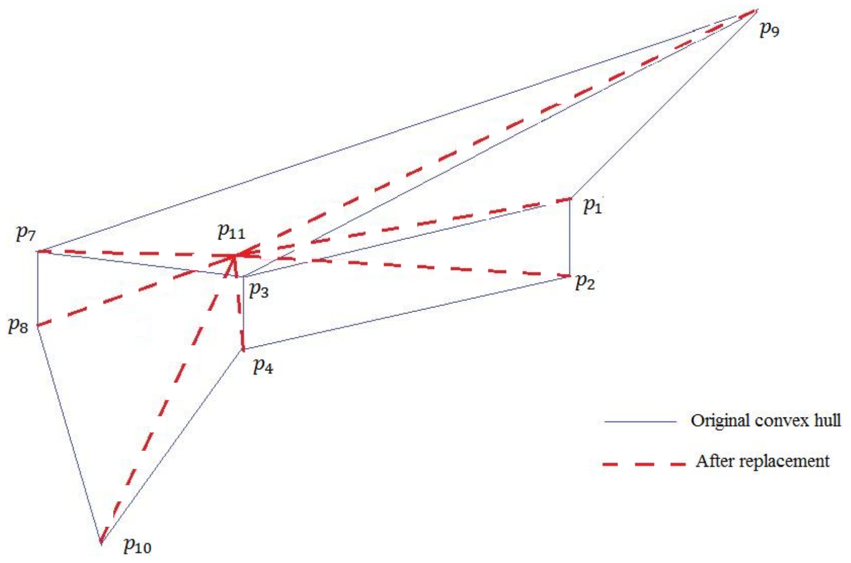


Figure 4: Illustration of replacing p_3 by p_{11}.

the largest inscribed ball.

(b) Since $\|{}^v a_i\| = 1$ for all i, conv$([{}^v a_1, {}^v a_2, \cdots, {}^v a_9])$ is in the interior of the unit ball except nine vertexes ${}^v a_1, {}^v a_2, \cdots, {}^v a_9$. All the new points $p_{12}, \cdots, p_{15}$ are on the unit sphere and different from ${}^v a_1, {}^v a_2, \cdots, {}^v a_8$. Introducing them to matrix ${}^v\!A$ will expand the convex hull, i.e., conv$([{}^v a_1, {}^v a_2, \cdots, {}^v a_9])\subset$conv$({}^v\!A)$. Therefore, ${}^v\rho \geq {}^v\rho_0 = 0.01$. □

To confirm this conclusion, we have calculated ${}^v\rho$ with double precision arithmetic as described in Section 2. Numerical calculation returns ${}^v\rho = 0.010002475$, which confirms that initially ${}^v\rho$ is larger than 0.01.

3.3.2 After one rescaling we have ${}^v\rho' < 0.01$

Numerical experiment shows that rescaling by using ${}^v a_7$ will decrease the size of the inscribed ball, i.e., ${}^v a_j = {}^v a_7 = p_9$ in (5). The hyperplane defined by ${}^v a_8, {}^v a_9, {}^v a_{10}$ restricts the ball. The distance from the origin to this hyperplane is 0.009964594, which gives an upper bound for ${}^v\rho'$, i.e., ${}^v\rho' \leq 0.009964594$.

We verify this number by multiple methods. First we solve the problem

$$ {}^v\rho' = \min_{\|z\|=1} \max_i \{-z^T (a'_v)_i\} \tag{9}$$

by the *fminmax* function in MATLAB. The solution returned is also ${}^v\rho' = 0.009964594$. By the reasons stated in Section 2 for the function *fminmax*, we numerically enumerate all the facets of the convex hull and calculate the minimal distance from the origin to those facets and use both of LU and QR factorizations to solve linear equation systems in the process of calculating the hyperplanes. All the calculations are done in double precision arithmetic. The returned results are within the same order of $O(10^{-15})$ precision. The difference between ${}^{v}\rho$ and ${}^{v}\rho'$ is on the order of $O(10^{-5})$, which is much larger than the roundoff errors. Thus we claim that ${}^{v}\rho > {}^{v}\rho'$. Therefore, utilizing ${}^{v}a_7$ to rescale this von Neumann problem $(Eg.vN)$ will shrink the inscribed ball. Claim 3 is verified.

3.3.3 Weights after running the von Neumann algorithm

In the desired rescaling von Neumann algorithm, we run the von Neumann algorithm to identify the rescaling vector, which need to be ${}^{v}a_7$ in this example. The rescaling vector should be the corresponding column vector of the largest coordinate of x when the von Neumann algorithm stops after $6mn^2$ iterations [13]. Let ξ_i be the i-th coordinate of x, so $x = [\xi_1, \xi_2, \cdots, \xi_{13}]$. In our example, the von Neumann algorithm initiated with $x^0 = e_7$, where e_7 is the unit vector corresponding to index 7. After 3042 iterations, the numerical experiment shows that ${}^{v}a_7$ and ${}^{v}a_8$ have the same largest weight $\xi_7 = \xi_8 = \xi_{\max}$. Now we verify ξ_7 and ξ_8 are theoretically equal. Since we start from $\xi_7^0 = 1, \xi_8^0 = 0$. The superscript denotes the iteration counter. At the first iteration, ${}^{v}a_8$ is utilized to update x. Thus, $\xi_7^1 = \xi_8^1 = 0.5$. After that, neither of ${}^{v}a_7$ and ${}^{v}a_8$ are used again to update. Throughout the following 3041 iterations, ${}^{v}a_7^T({}^{p}\!Ax^k)$ is always positive, and the minimal difference between ${}^{v}a_7^T({}^{p}\!Ax^k)$ and ${}^{v}a_s^T({}^{p}\!Ax^k)$ are in the order of $O(10^{-5})$ versus the numerical error is $O(10^{-15})$ for double digit accuracy. Thus, we recognize that there is enough separation between ${}^{v}a_7$ and ${}^{v}a_s$, and ${}^{v}a_7$ is not overlooked. Consequently, ξ_7 and ξ_8 have exactly the same updates starting from the second iteration [9]

$$\xi_7^k = \xi_8^k = \lambda^k \xi_7^{k-1} = \cdots = \frac{1}{2}\prod_{i=2}^{k} \lambda^i.$$

Therefore, ξ_7 and ξ_8 remain equal, thus we can choose ${}^{v}a_7$ as the rescaling vector. The computational experiment also shows that ${}^{v}a_7$ is chosen when the von Neumann phase terminates. Claim 4 is confirmed.

4 From the von Neumann example to the perceptron example

In this section, we explain how we derived the example for the rescaling perceptron algorithm which is stated in Section 2. Since constructing a von Neumann example in dimension three can be visualized, we start from the von Neumann example in spite the fact that the complexity of the deterministic rescaling von Neumann algorithm has not been proved yet. After obtaining an example for the von Neumann algorithm for which the inscribed ball decrease, we adopt the following steps.

1. Identify all the facets of conv($^v\!A$) and calculate their normal vectors.

2. Lift these normal vectors to a one dimension higher space. Since we already know that conv($^v\!A$) only contains a small ball inside, lifting will lead to a narrow feasible cone.

3. Run the perceptron algorithm and remove the redundant constraints that are not used during updates.

4. Identify a constraint which can shrink $^p\rho$ when rescaling is done by its normal vector ^pa_j.

5. Analogous to constructing the von Neumann example, adding perturbed constraints to balance the weight among all vectors ^pa_i so that ^pa_j is used the most frequently during the preceptron updates.

With the above five steps, we obtain the example presented in Section 1.

5 Computational results for the rescaling von Neumann algorithm

The description of the deterministic rescaling von Neumann algorithm is given in Section 3.1. As we state, the theoretical complexity result of this algorithm is not proved yet. Regardless, we present some computation results in this section to show that the performance of the von Neumann algorithm is improved after applying the rescaling phase.

To generate ill-conditioned von Neumann problems which have small $^v\rho$, we adapt the tube generator [5, 10].It places $n-1$ points on the spherical cap concentrated around $[0\ 0\ \ldots 0\ 1]^T$ or $[0\ 0\ \ldots 0\ -1]^T$. The n-th point is generated as a positive combination of the antipodes of the first $n-1$ points, so that it is on the opposite spherical cap. This generator guarantees that the von Neumann problem is feasible. At the mean time, since all the n points lie in the tube around the last coordinate axis, we can control $^v\rho$ by adjusting the width of the tube. The performance of the deterministic rescaling von Neumann algorithm is compared with the original von Neumann algorithm. For each size of A, we randomly generated 20 von Neumann problems using tube generator. The results in Table 1 are obtained by using Matlab R2014a on a

Table 1: Comparison of the performance of Algorithm 1 and the original von Neumann algorithm,

Size:$m\times n$	Original		Deterministic rescaling			
	Sec.	No.update	Sec.	No.update	No.rescaling	$^v\rho'/^v\rho$
5×10	9.8102	3.77E+5	0.3542	1.22E+4	4.05	63.1851
10×20	10.3687	3.98E+5	1.8770	6.00E+4	2.5	47.9988
20×40	16.0625	5.90E+5	10.2361	2.30E+5	1.2	4.2875

Windows 7 desktop (Intel(R) Xeon(R) CPU, 3.07GHz) with 4Gb RAM.

6 Conclusions

Peña and Sohèili presented a deterministic rescaling perceptron algorithm. The main result of the paper is the construction of an example which shows that even though the algorithm eventually expands the feasible cone, ${}^{p}\rho$ may decrease after one rescaling step. By the duality relationship between the perceptron and the von Neumann algorithms, we apply the Peña-Sohèili rescaling method to the von Neumann algorithm. Driven by the desire of proving its complexity, we explore how ${}^{v}\rho$ will change after rescaling. We construct an example in $\mathbb{R}^3$ to show that there is no guarantee of monotonic increasing of ${}^{v}\rho$. Therefore, the complexity cannot be proved by increasing ${}^{v}\rho$ and another methods e.g., a proof analogous to the one presented in [13] need to be discovered. The computational results show that the deterministic rescaling von Neumann algorithm solves the test problems significantly faster then the original von Neumann algorithm.

References

[1] I. Bárány and S. Onn. Colourful linear programming and its relatives. *Mathematics of Operations Research*, 22(3), 1997.

[2] R. K. Brayton, S. W. Director, G. Hachtel, and L.Vidigal. A new algorithm for statistical circuit design based on quasi-newton methods and function splitting. *IEEE Trans. Circuits and Systems*, 26:784–794, 1979.

[3] G. B. Dantzig. Converting a converging algorithm into a polynomially bounded algorithm. Technical Report SOL 91-5, Stanford University, 1991.

[4] G. B. Dantzig. An ϵ-precise feasible solution to a linear program with a convexity constraint in $1/\epsilon^2$ iterations independent of problem size. Technical Report SOL 92-5, Stanford University, 1992.

[5] A. Deza, S. Huang, T. Stephen, and T. Terlaky. The colourful feasibility problem. *Discrete Applied Mathematics*, 156:2166–2177, 2008.

[6] J. Dunagan and S. Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. In *Proceedings of STOC'04*, pages 315–320. ACM Press, 2004.

[7] M. A. Epelman and R. M. Freund. Condition number complexity of an elementary algorithm for resolving a conic linear system. *Mathematical Programming*, 88(3):451–485, 2000.

[8] G. H. Golub and C. F. V. Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore and London, third edition, 1996.

[9] J. P. M. Gonçalves, R. Storer, and J. Gondzio. A family of linear programming algorithms based on an algorithm by von Neumann. *Optimization Methods and Software*, 24(3):461–478, 2009.

[10] S. Huang. Colourful feasibility: algorithms, bounds and implications. Master's thesis, Computing and Software, McMaster University, Hamilton,Ontario, 2007.

[11] D. Li and T. Terlaky. The duality between the perceptron algorithm and the von neumann algorithm. In L. F. Zuluaga and T. Terlaky, editors, *Modeling and Optimization: Theory and Applications*, volume 62 of *Springer Proceedings in Mathematics and Statistics*, pages 113–136. Springer New York, 2013.

[12] M. Minsky and S. A. Papert. *Perceptrons: An Introduction To Computational Geometry*. MIT Press, 1969.

[13] J. Peña and M. Sohèili. A deterministic rescaled perceptron algorithm. *Mathematical Programming*, pages 1–14, January 2015.

[14] F. Rosenblatt. The perceptron–a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[15] L. N. Trefethen and D. Bau III. *Numerical Linear Algebra*. SIAM, 1997.