A smooth perceptron algorithm

Negar Soheili* Javier Peña†

September 22, 2011

Abstract

The perceptron algorithm, introduced in the late fifties in the machine learning community, is a simple greedy algorithm for finding a solution to a finite set of linear inequalities. The algorithm's main advantages are its simplicity and noise tolerance. The algorithm's main disadvantage is its slow convergence rate. We propose a modified version of the perceptron algorithm that retains the algorithm's original simplicity but has a substantially improved convergence rate.

Key words: perceptron algorithm, smoothing technique, condition number

1 Introduction

The perceptron algorithm was introduced by Rosenblatt [18] in 1958 for solving classification problems in machine learning. This algorithm solves the polyhedral feasibility problem

$$A^{\mathrm{T}}y > 0 \tag{1}$$

via a sequence of simple greedy updates. The main advantages of the perceptron algorithm are its simplicity and robustness to noise. Its main limitation is its slow rate of convergence. Block [2] and Novikoff [13] showed that the perceptron algorithm finds a solution to (1) after at most $\frac{1}{\rho(A)^2}$ iterations, where $\rho(A)$ is the following measure of thickness of the cone of solutions to (1):

$$\rho(A) := \max_{\|z\|=1} \{r : \mathbb{B}(z, r) \subseteq \{y \mid A^{\mathrm{T}}y \geq 0\}\}. \tag{2}$$

*Tepper School of Business, Carnegie Mellon University, USA, nsoheili@andrew.cmu.edu

†Tepper School of Business, Carnegie Mellon University, USA, jfp@andrew.cmu.edu

In (2) and throughout the paper $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^m$ and $\mathbb{B}(z,r)$ denotes the Euclidean ball of radius r centered at z. The main contribution of this paper is a modified version of the perceptron algorithm that retains its original simplicity but terminates in $\mathcal{O}\left(\frac{\sqrt{\log(n)}}{\rho(A)}\right)$ iterations, where n is the dimension of the column space of A (see Theorem 1).

The perceptron algorithm has been the subject of recent renewed interest in the machine learning, computer science, and optimization communities. In particular, Dunagan and Vempala [5] proposed a randomized rescaled version of the perceptron algorithm that terminates in $\mathcal{O}\left(n\log\left(\frac{1}{\rho(A)}\right)\right)$ iterations with high probability. Belloni, Freund and Vempala [1] extended Dunagan and Vempala's approach to more general conic feasibility problems. Furthermore, the measure of thickness $\rho(A)$ defined in (2) is related to a certain *condition number* of the matrix A used by Goffin [9], and further studied and extended by Cheung et al. [3, 4]. Related condition and geometric measures have also been used for the analysis of algorithms for more general feasibility problems in a line of work introduced by Renegar [15–17], and further investigated by a number of researchers [6–8, 14, 20].

A key insight for our work is the observation that the perceptron algorithm can be seen as a first-order algorithm for a certain canonical optimization problem associated to the feasibility problem (1). This observation leads to a modified version of the perceptron algorithm with the aforementioned properties. In contrast to Dunagan and Vempala's, our algorithm is deterministic. Our approach is based on an interpretation of the perceptron algorithm as a subgradient method, and a subsequent refinement via a smoothing technique proposed by Nesterov [12]. We note that similar improvements on convergence rate could be achieved via other accelerated first-order techniques such as the mirror-prox method of Nemirovski [11] or other accelerated first-order methods for saddle-point problems such as those described in [10, 19]. The approach that we have followed enables us to achieve the improvement on convergence rate with only fairly minor modifications on the original algorithm. In particular, our modified algorithm retains most of the perceptron's original simplicity. In addition, although our approach is based on Nesterov's smoothing technique [12], we provide a succinct and self-contained proof of the algorithm's convergence rate.

The paper is organized as follows. In Section 2 we recall the perceptron algorithm and present our proposed modification, namely a *smooth* perceptron algorithm. In Section 3 we discuss the key observation that the perceptron algorithm can be seen as a first-order algorithm for a canonical

non-smooth concave optimization problem associated to (1). In Section 4 we prove our main theorem, namely the improved rate of convergence of the smooth perceptron algorithm. Finally, in Section 5 we report some numerical experiments that illustrate the behavior of the classical and the smooth perceptron algorithms. The experiments demonstrate that both the number of iterations and total CPU time required by the smooth perceptron algorithm are indeed lower than those required by the classical perceptron algorithm in a way that is commensurate with our main theoretical result.

2 Smooth perceptron algorithm

We next describe our modification of the perceptron algorithm. To that end, we first recall the perceptron algorithm in its *classical* version and in an equivalent *normalized* version. We subsequently tweak the steps in the main loop of the normalized version to obtain a new *smooth* version of the perceptron algorithm. The smooth version has the improved convergence rate stated in Theorem 1 below.

To solve problem (1), the perceptron algorithm starts with a trial solution (usually zero). At each iteration it moves the current trial solution one unit in the direction normal to one of the violated constraints (if any). This procedure is repeated as long as the trial solution is not solution to (1). For convenience of notation we make the following assumption on the input matrix A.

Assumption 1 $A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \in \mathbb{R}^{m\times n}$, *where* $\|a_j\| = 1$ *for* $j = 1, \ldots, n$.

Notice that this assumption can be made without lost of generality since the matrix A can always be pre-processed accordingly.

Classical Perceptron Algorithm
begin
  $y_0 := 0$;
  for $k = 1, 2, \ldots$
    $a_j^{\mathrm{T}} y_k := \min\limits_{i=1,\ldots,n} a_i^{\mathrm{T}} y_k$;
    $y_{k+1} := y_k + a_j$;
  end
end

We next introduce some convenient notation. Let $\Delta_n := \{x \geq 0 : \|x\|_1 = 1\}$, and for $y \in \mathbb{R}^m$, let $x(y) \in \Delta_n$ denote an arbitrary point in the set

$\operatorname{argmin}_{x\in\Delta_n} \langle A^{\mathrm{T}}y, x\rangle$. Observe that for a given $y \in \mathbb{R}^m$, we have $a_j^{\mathrm{T}}y = \min_i a_i^{\mathrm{T}}y$ if and only if $a_j = Ax(y)$. It follows that at iteration k the classical perceptron algorithm generates an iterate y_k with $y_k = Ax_k$ for some $x_k \geq 0$ with $\|x_k\|_1 = k$. This observation leads to the following normalized version of the perceptron algorithm.

Normalized Perceptron Algorithm
begin
  $y_0 := 0$;
  for $k = 0, 1, 2, \ldots$
    $\theta_k := \frac{1}{k+1}$;
    $y_{k+1} := (1-\theta_k)y_k + \theta_k Ax(y_k)$;
  end
end

It is easy to see that the k-th iterate generated by the normalized perceptron algorithm is exactly the same as the k-th iterate generated by the classical perceptron algorithm divided by k. In particular, the k-th iterate y_k generated by the normalized perceptron algorithm satisfies $y_k = Ax_k$ for some $x_k \in \Delta_n$. Indeed, the main loop in the normalized perceptron algorithm can also be written in the following fashion to maintain both y_k and x_k:

for $k = 0, 1, 2, \ldots$
  $\theta_k := \frac{1}{k+1}$;
  $y_{k+1} := (1-\theta_k)y_k + \theta_k Ax(y_k)$;
  $x_{k+1} := (1-\theta_k)x_k + \theta_k x(y_k)$;
end

Now define the following *smooth* version of the map $y \mapsto x(y)$. Given $\mu > 0$ let $x_\mu : \mathbb{R}^m \to \Delta_n$ be defined as

$$x_\mu(y) := \frac{e^{\frac{-A^{\mathrm{T}}y}{\mu}}}{\|e^{\frac{-A^{\mathrm{T}}y}{\mu}}\|_1}. \tag{3}$$

In this expression we are using *vectorized* notation. In other words, $e^{\frac{-A^{\mathrm{T}}y}{\mu}}$ denotes the n-dimensional vector

$$e^{\frac{-A^{\mathrm{T}}y}{\mu}} := \begin{bmatrix} e^{\frac{-a_1^{\mathrm{T}}y}{\mu}} \\ \vdots \\ e^{\frac{-a_n^{\mathrm{T}}y}{\mu}} \end{bmatrix}.$$

We will write $\mathbf{1}$ to denote the n-dimensional vector of all ones. We are now ready to present our smooth perceptron algorithm.

Smooth Perceptron Algorithm
begin
$y_0 := \frac{A\mathbf{1}}{n}$; $\mu_0 := 1$; $x_0 := x_{\mu_0}(y_0)$;
for $k = 0, 1, 2, \ldots$
$\theta_k := \frac{2}{k+3}$;
$y_{k+1} := (1-\theta_k)(y_k + \theta_k A x_k) + \theta_k^2 A x_{\mu_k}(y_k)$;
$\mu_{k+1} := (1-\theta_k)\mu_k$;
$x_{k+1} := (1-\theta_k)x_k + \theta_k x_{\mu_{k+1}}(y_{k+1})$;
end
end

We can now formally state our main theorem.

Theorem 1 *Assume $A \in \mathbb{R}^{m\times n}$ satisfies Assumption 1 and problem* (1) *is feasible. Then the smooth perceptron algorithm terminates in at most*

$$\frac{2\sqrt{\log(n)}}{\rho(A)} - 1$$

iterations.

We present the proof of Theorem 1 in Section 4 below.

3 Perceptron algorithm as a first-order algorithm

We next show that the normalized perceptron algorithm can be seen as first-order algorithm for the canonical maximization problem (5) below associated to (1). The smooth perceptron algorithm in turn is a smooth first-order algorithm for (5). For ease of exposition, we make the following assumption throughout the rest of the paper.

Assumption 2 *Problem* (1) *is feasible.*

Under Assumption 2 the thickness parameter defined in (2) satisfies

$$\begin{aligned} \rho(A) &= \max_{\|y\|=1} \min_{i=1,\ldots,n} a_i^{\mathrm{T}} y \\ &= \max_{\|y\|\le 1} \min_{i=1,\ldots,n} a_i^{\mathrm{T}} y \\ &= \max_{\|y\|\le 1} \psi(y), \end{aligned} \tag{4}$$

where $\psi(y) := \min_{x \in \Delta_n} \langle A^{\mathrm{T}} y, x \rangle$.

Observe that a point $y \in \mathbb{R}^m$ with $\|y\| \leq 1$ is a solution to (1) if and only if $\psi(y) > 0$, which in turn holds if and only if $\psi(y)$ is within $\rho(A)$ of its maximum on $\{y : \|y\| \leq 1\}$.

Consider now the function $\varphi : \mathbb{R}^m \to \mathbb{R}$ defined as

$$\varphi(y) := -\frac{1}{2}\|y\|^2 + \psi(y) = -\frac{1}{2}\|y\|^2 + \min_{x \in \Delta_n} \langle A^{\mathrm{T}} y, x \rangle .$$

It readily follows that

$$\frac{1}{2}\rho(A)^2 = \max_{y \in \mathbb{R}^m} \varphi(y). \tag{5}$$

Furthermore, a point $y \in \mathbb{R}^m$ solves (1) if $\varphi(y) > 0$, that is, if $\varphi(y)$ is within $\frac{1}{2}\rho(A)^2$ of its maximum.

Recall the main step in the normalized perceptron algorithm:

$$y_+ := (1-\theta)y + \theta A x(y) = y + \theta(-y + Ax(y)).$$

Observe that $-y + Ax(y) \in \partial\varphi(y)$. Hence the normalized perceptron can be seen as a subgradient algorithm applied to the maximization problem (5).

4 Proof of the main theorem

The specific steps in the smooth perceptron algorithm as well as the proof of Theorem 1 are based on applying Nesterov's excessive gap technique [12] to (5). For $\mu > 0$ define the smooth approximation φ_μ of φ as:

$$\varphi_\mu(y) := -\frac{1}{2}\|y\|^2 + \min_{x \in \Delta_n} \{\langle A^{\mathrm{T}} y, x \rangle + \mu d(x)\}, \tag{6}$$

where $d(x)$ is the entropy prox-function on Δ_n, that is,

$$d(x) := \sum_{j=1}^{n} x_j \log(x_j) + \log(n).$$

It is easy to see that the minimizer in (6) is precisely the point $x_\mu(y)$ defined in (3). Hence

$$\varphi_\mu(y) = -\frac{1}{2}\|y\|^2 + \langle A^{\mathrm{T}} y, x_\mu(y) \rangle + \mu d(x_\mu(y)).$$

Theorem 1 is a consequence of the following two lemmas.

Lemma 1 *For any given $x \in \Delta_n$ and $y \in \mathbb{R}^m$, we have*

$$\rho(A) \leq \|Ax\|, \tag{7}$$

and

$$\varphi_\mu(y) \leq \varphi(y) + \mu \log(n). \tag{8}$$

Proof: From (4) we get $\rho(A) \leq \max_{\|y\|=1} \left\langle A^{\mathrm{T}} y, x \right\rangle = \|Ax\|$ so (7) follows. Inequality (8) follows from the construction of φ_μ and $\max_{x \in \Delta_n} d(x) = \log(n)$. ■

Lemma 2 *The iterates $x_k \in \Delta_n$, $y_k \in \mathbb{R}^m$, $k = 0, 1, \ldots$ generated by the smooth perceptron algorithm satisfy the following excessive gap condition*

$$\frac{1}{2}\|Ax_k\|^2 \leq \varphi_{\mu_k}(y_k). \tag{9}$$

Proof of Theorem 1: Putting Lemma 1 and Lemma 2 together, we get

$$\tfrac{1}{2}\rho(A)^2 \leq \tfrac{1}{2}\|Ax_k\|^2 \leq \varphi_{\mu_k}(y_k) \leq \varphi(y_k) + \mu_k \log(n).$$

In our algorithm $\mu_k = \frac{2}{(k+1)(k+2)} < \frac{2}{(k+1)^2}$ so $\varphi(y_k) > 0$, and consequently $A^{\mathrm{T}} y_k > 0$, as soon as

$$k \geq \frac{2\sqrt{\log(n)}}{\rho(A)} - 1.$$

■

Before we prove Lemma 2, recall that for $z, x \in \Delta_n$ the Bregman distance $h(z, x)$ is defined as $h(z, x) = d(z) - d(x) - \langle \nabla d(x), z - x \rangle$ and satisfies

$$h(z, x) \geq \frac{1}{2}\|z - x\|_1^2. \tag{10}$$

Observe also that by Assumption 1 we have

$$\|A\|_{1,2} = \max\{\|Ax\|_2 : \|x\|_1 = 1\} = \max\{\|a_1\|_2, \ldots, \|a_n\|_2\} = 1.$$

Proof of Lemma 2: We proceed by induction. For $k = 0$ we have:

$$\begin{aligned}
\tfrac{1}{2}\|Ax_0\|^2 &= \tfrac{1}{2}\|A\tfrac{\mathbf{1}}{n}\|^2 + \left\langle A\tfrac{\mathbf{1}}{n}, A(x_0 - \tfrac{\mathbf{1}}{n}) \right\rangle + \tfrac{1}{2}\|A(x_0 - \tfrac{\mathbf{1}}{n})\|^2 \\
&\leq \tfrac{1}{2}\|A\tfrac{\mathbf{1}}{n}\|^2 + \left\langle A\tfrac{\mathbf{1}}{n}, A\left(x_0 - \tfrac{\mathbf{1}}{n}\right) \right\rangle + \tfrac{1}{2}\|x_0 - \tfrac{\mathbf{1}}{n}\|_1^2 \\
&\leq -\tfrac{1}{2}\|A\tfrac{\mathbf{1}}{n}\|^2 + \left\langle A^{\mathrm{T}} A\tfrac{\mathbf{1}}{n}, x_0 \right\rangle + d(x_0) \\
&= -\tfrac{1}{2}\|y_0\|^2 + \left\langle A^{\mathrm{T}} y_0, x_{\mu_0}(y_0) \right\rangle + d(x_{\mu_0}(y_0)) \\
&= \varphi_{\mu_0}(y_0).
\end{aligned}$$

The first inequality above follows from $\|A\|_{2,1} = 1$. The second inequality follows from (10) and $d(\frac{\mathbf{1}}{n}) = 0$.

Now we will show that if (9) holds for k then it also holds for $k+1$. To ease notation, drop the index k and write y_+, x_+, μ_+ for y_{k+1}, x_{k+1}, μ_{k+1} respectively. Also, let $\hat{x} = (1-\theta)x + \theta x_\mu(y)$ so that $y_+ = (1-\theta)y + \theta A\hat{x}$. We have

$$\begin{aligned}
\varphi_{\mu_+}(y_+) &= -\tfrac{\|y_+\|^2}{2} + \left\langle A^{\mathrm{T}} y_+, x_{\mu_+}(y_+)\right\rangle + \mu_+ d(x_{\mu_+}(y_+)) \\
&= -\tfrac{\|(1-\theta)y+\theta A\hat{x}\|^2}{2} + (1-\theta)\big[\left\langle A^{\mathrm{T}} y, x_{\mu_+}(y_+)\right\rangle + \mu d(x_{\mu_+}(y_+))\big] \\
&\quad + \theta\left\langle A^{\mathrm{T}} A\hat{x}, x_{\mu_+}(y_+)\right\rangle \\
&\geq (1-\theta)\left[-\tfrac{\|y\|^2}{2} + \left\langle A^{\mathrm{T}} y, x_{\mu_+}(y_+)\right\rangle + \mu d(x_{\mu_+}(y_+))\right]_1 \\
&\quad + \theta\left[-\tfrac{\|A\hat{x}\|^2}{2} + \left\langle A^{\mathrm{T}} A\hat{x}, x_{\mu_+}(y_+)\right\rangle\right]_2 .
\end{aligned} \tag{11}$$

The last inequality follows from the concavity of the function $y \mapsto -\frac{\|y\|^2}{2}$. We can now estimate the expression in the first bracket in (11) as follows:

$$\begin{aligned}
\big[.\big]_1 &= -\tfrac{1}{2}\|y\|^2 + \left\langle A^{\mathrm{T}} y, x_\mu(y)\right\rangle + \mu d(x_\mu(y)) + \mu(d(x_{\mu_+}(y_+)) - d(x_\mu(y))) \\
&\qquad + \left\langle A^{\mathrm{T}} y, x_{\mu_+}(y_+) - x_\mu(y)\right\rangle \\
&= \varphi_\mu(y) + \mu\big(d(x_{\mu_+}(y_+)) - d(x_\mu(y)) - \left\langle \nabla d(x_\mu(y)), x_{\mu_+}(y_+) - x_\mu(y)\right\rangle\big) \\
&= \varphi_\mu(y) + \mu h(x_{\mu_+}(y_+), x_\mu(y)) \\
&\geq \tfrac{1}{2}\|Ax\|^2 + \mu h(x_{\mu_+}(y_+), x_\mu(y)) \\
&\geq \tfrac{1}{2}\|A\hat{x}\|^2 + \left\langle A^{\mathrm{T}} A\hat{x}, x - \hat{x}\right\rangle + \mu h(x_{\mu_+}(y_+), x_\mu(y)).
\end{aligned} \tag{12}$$

The second step above follows from the optimality conditions for (6). The third step follows from the definition of Bregman distance h. The fourth step follows from the induction hypothesis (9).

The expression in the second bracket in (11) can be written as

$$\big[.\big]_2 = \frac{1}{2}\|A\hat{x}\|^2 + \left\langle A^{\mathrm{T}} A\hat{x}, x_{\mu_+}(y_+) - \hat{x}\right\rangle. \tag{13}$$

Observe also that

$$\begin{aligned}
x_+ - \hat{x} &= (1-\theta)x + \theta x_{\mu_+}(y_+) - (1-\theta)x - \theta x_\mu(y) \\
&= \theta(x_{\mu_+}(y_+) - x_\mu(y)).
\end{aligned} \tag{14}$$

Plugging (12) and (13) into (11), we finish the proof as follows:

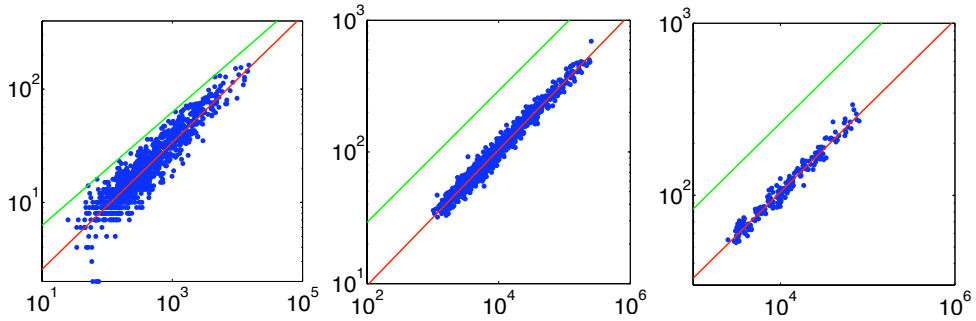
$$\begin{aligned}
\varphi_{\mu_+}(y_+) &= (1-\theta)\big[\tfrac{1}{2}\|A\hat{x}\|^2 + \big\langle A^{\mathrm{T}}A\hat{x}, x-\hat{x}\big\rangle + \mu h(x_{\mu_+}(y_+), x_\mu(y))\big] \\
&\quad +\theta\big[\tfrac{1}{2}\|A\hat{x}\|^2 + \big\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+)-\hat{x}\big\rangle\big] \\
&= \tfrac{1}{2}\|A\hat{x}\|^2 + \theta\big\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+) - x_\mu(y)\big\rangle + (1-\theta)\mu h(x_{\mu_+}(y_+), x_\mu(y)) \\
&\geq \tfrac{1}{2}\|A\hat{x}\|^2 + \theta\big\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+) - x_\mu(y)\big\rangle + \tfrac{1}{2}\theta^2\|x_{\mu_+}(y_+) - x_\mu(y)\|_1^2 \\
&= \tfrac{1}{2}\|A\hat{x}\|^2 + \big\langle A^{\mathrm{T}}A\hat{x}, x_+ - \hat{x}\big\rangle + \tfrac{1}{2}\|x_+ - \hat{x}\|_1^2 \\
&\geq \tfrac{1}{2}\|A\hat{x}\|^2 + \langle A\hat{x}, A(x_+ - \hat{x})\rangle + \tfrac{1}{2}\|A(x_+ - \hat{x})\|^2 \\
&= \tfrac{1}{2}\|Ax_+\|^2.
\end{aligned}$$

The second step above follows because $\hat{x} = (1-\theta)x + \theta x_\mu(y)$. The third step follows from (10) and the fact that at iteration k we have $\frac{\theta^2}{2(1-\theta)} = \frac{2}{(k+3)^2} \leq \frac{2}{(k+1)(k+2)} = \mu$. The fourth step follows from (14). The fifth step follows from $\|A\|_{1,2} = 1$. ■

5 Numerical experiments

We next report results of some numerical experiments comparing the performance of the smooth perceptron algorithm and the classical perceptron algorithm. The theoretical convergence rates of the smooth and the classical perceptron algorithms suggest the relationship $Y = 2\sqrt{\log(n)}X^{\frac{1}{2}}$ between the number of iterations Y taken by the smooth perceptron algorithm, and the number of iterations X taken by the classical perceptron algorithm to find a feasible solution to (1). We test this relationship by estimating the parameter β in the model $Y = cX^\beta$ for a collection of empirically generated values of the pair (X, Y). To estimate the exponent β via linear regression, we apply a logarithmic transformation and estimate the slope β in the linearized model $\log(Y) = \log(c) + \beta\log(X)$.

We implemented the classical and the smooth perceptron algorithms in MATLAB and ran them in collections of randomly generated matrices of sizes 10×50, 10×500, and 200×1000 with a wide spectrum of thickness parameter $\rho(A)$. We recorded the number of iterations and CPU time required by each of these algorithms to find a solution to (1). Figure 1 displays log-log scatter plots of the number of iterations taken by the smooth perceptron algorithm versus those taken by the classical perceptron algorithm. Each plot also shows the fitted regression line in red. In addition, we plot the theoretical linear relationship $\log(Y) = \log(2\sqrt{\log(n)}) + \frac{1}{2}\log(X)$ in green.



(a) $A \in \mathbb{R}^{10\times 50}$ (b) $A \in \mathbb{R}^{100\times 500}$ (c) $A \in \mathbb{R}^{200\times 1000}$

Figure 1: Scatter plot of the number of iterations taken by smooth and classical versions of the perceptron algorithm.

Table 1: Regression results for number of iterations

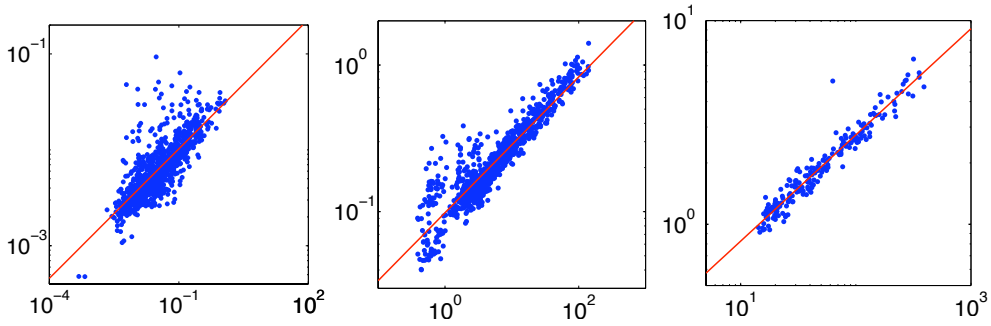
size 10×50		size 100×500		size 200×1000	
$\hat{\beta}$	95% CI	$\hat{\beta}$	95% CI	$\hat{\beta}$	95% CI
0.5597	$[0.5462, 0.5732]$	0.5156	$[0.5104, 0.5207]$	0.5009	$[0.4880, 0.5137]$

Table 1 displays the regression estimates of β. For example, the linear regression yields the point estimate $\hat{\beta} = 0.5009$ with 95% confidence interval $= [0.4880, 0.5137]$ for the instances with size 200×1000. As Table 1 shows, the slope of the regression line is close to 0.5 in all cases. This is also verified by Figure 1, where the green and red lines are approximately parallel in all cases.

The experimental results confirm that the number of iterations required by the smooth perceptron algorithm is indeed lower than those required by the classical perceptron algorithm. However, each iteration of the smooth perceptron algorithm involves more computational work. It is then natural to explore what type of relationship exists between the computational times taken by the two algorithms. Figure 2 and Table 2 respectively display log-log scattered plots with regression lines and summaries of the regression estimates for the CPU times taken by the smooth perceptron algorithm and by the classical perceptron algorithm. The scatter plots show that the total CPU time taken by the smooth perceptron algorithm is also substantially lower than that taken by the classical perceptron algorithm. Furthermore, Table 2 shows that the slopes of the regression lines in these figures are also close to 0.5.

Table 2: Regression results for CPU times

size 10×50		size 100×500		size 200×1000	
$\hat{\beta}$	95% CI	$\hat{\beta}$	95% CI	$\hat{\beta}$	95% CI
0.4498	$[0.4272, 0.4724]$	0.4631	$[0.4511, 0.4752]$	0.5216	$[0.5029, 0.5402]$



(a) $A \in \mathbb{R}^{10\times 50}$ (b) $A \in \mathbb{R}^{100\times 500}$ (c) $A \in \mathbb{R}^{200\times 1000}$

Figure 2: Scatter plot of CPU time taken by smooth and classical versions of the perceptron algorithm.

References

[1] A. Belloni, R. Freund, and S. Vempala. An efficient rescaled perceptron algorithm for conic systems. *Math. Oper. Res*, 34(3):621–641, 2009.

[2] H. D. Block. The perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34:123–135, 1962.

[3] D. Cheung and F. Cucker. A new condition number for linear programming. *Math. Program.*, 91:163–174, 2001.

[4] D. Cheung, F. Cucker, and J. Peña. On strata of degenerate polyhedral cones I: Condition and distance to strata. *Europ. J. Oper. Res*, 198:23–28, 2009.

[5] J. Dunagan and S. Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. *Math. Program.*, 114(1):101–114, 2006.

[6] M. Epelman and R. M. Freund. Condition number complexity of an elementary algorithm for computing a reliable solution of a conic linear system. *Math. Program.*, 88:451–485, 2000.

[7] R. Freund. Complexity of convex optimization using geometry-based measures and a reference point. *Math Program.*, 99:197–221, 2004.

[8] R. Freund and J. Vera. Condition-based complexity of convex optimization in conic linear form via the ellipsoid algorithm. *SIAM J. Optim.*, 10:155–176, 1999.

[9] J. Goffin. The relaxation method for solving systems of linear inequalities. *Math. Oper. Res.*, 5:388–414, 1980.

[10] G. Lan, Z. Lu, and R. Monteiro. Primal-dual first-order methods with $\mathcal{O}(1/\epsilon)$ iteration-complexity for cone programming. *Math. Programming.*, Series A:DOI 10.1007/s10107–008–0261–6, 2009.

[11] A. Nemirovski. Prox-method with rate of convergence $\mathcal{O}(1/t)$ for variational inequalities with Lipschitz-continuous monotone operators and smooth convex-concave saddle point problems. *SIAM J. Optim.*, 15(1):229–251, 2004.

[12] Y. Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM J. Optim.*, 16(1):235–249, 2005.

[13] A. B. J. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume XII, pages 615–622, 1962.

[14] J. Peña and J. Renegar. Computing approximate solutions for convex conic systems of constraints. *Math Program.*, 87:351–383, 2000.

[15] J. Renegar. Incorporating condition measures into the complexity theory of linear programming. *SIAM J. Optim.*, 5:506–524, 1995.

[16] J. Renegar. Linear programming, complexity theory and elementary functional analysis. *Math. Program.*, 70:279–351, 1995.

[17] J. Renegar. Condition numbers, the barrier method, and the conjugate-gradient method. *SIAM J. Optim.*, 6:879–912, 1996.

[18] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Cornell Aeronautical Laboratory, Psychological Review*, 65(6):386–408, 1958.

[19] P. Tseng. On accelerated proximal gradient methods for convex-concave optimization. *SIAM J. Optim. (Submitted)*, 2008.

[20] J. Vera, J. Rivera, J. Peña, and Y. Hui. A primal-dual symmetric relaxation for homogeneous conic systems. *Journal of Complexity*, 23:245–261, 2007.