Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12–17, 2007

Optimizing 0/1 Loss for Perceptrons by Random Coordinate Descent

Ling Li and Hsuan-Tien Lin

***Abstract*—The 0/1 loss is an important cost function for perceptrons. Nevertheless it cannot be easily minimized by most existing perceptron learning algorithms. In this paper, we propose a family of random coordinate descent algorithms to directly minimize the 0/1 loss for perceptrons, and prove their convergence. Our algorithms are computationally efficient, and usually achieve the lowest 0/1 loss compared with other algorithms. Such advantages make them favorable for nonseparable real-world problems. Experiments show that our algorithms are especially useful for ensemble learning, and could achieve the lowest test error for many complex data sets when coupled with AdaBoost.**

I. Introduction

The perceptron was first introduced by Rosenblatt [1] as a probabilistic model for information processing in the brain. It is simply a linear threshold classifier, which can be thought as a hyperplane in the input space.

Given a set of examples with binary labels, an important task of perceptron learning is to find a hyperplane that classifies the examples with the smallest number of mislabeling errors. A data set that can be classified by some perceptron without any mislabeling errors is called *linearly separable*, or simply *separable*. For a separable set, the task of learning is relatively easy and can be carried out by many existing algorithms. For example, the perceptron learning rule [2] is guaranteed to converge to a separating hyperplane in a finite number of iterations. The hard-margin support vector machine (SVM) can even find the separating hyperplane that maximizes the minimal example margin [3].

However, these algorithms behave poorly when the data set is nonseparable, which is a more common situation in real-world problems. In such a situation, the perceptron learning rule will not converge, and is very unstable in the sense that the learned hyperplane might change from an optimal one to a worst-possible one within a single trial [4]. The optimization problem of the hard-margin SVM becomes infeasible, and hence cannot be solved without modifications. To tackle the nonseparable cases, many different algorithms have been proposed (see Section II). Although those algorithms appear quite different, they usually try to minimize some cost functions of example margins. Note that the number of mislabeling errors is proportional to a specific cost function, the 0/1 loss.

The 0/1 loss is an important criterion for perceptron learning. It captures the discrete nature of the binary classification (correct or incorrect), and partially indicates the prediction power. A good 0/1 loss minimization algorithm can be used either to obtain standalone perceptrons, or to build more complex classifiers with many perceptrons. However, the 0/1 loss cannot be directly minimized by most existing perceptron algorithms, which is both because the minimization problem is NP-complete [5], and because the loss is neither convex nor smooth.

In this paper, we propose a family of new perceptron algorithms to directly minimize the 0/1 loss. The central idea is random coordinate descent, i.e., iteratively searching along randomly chosen directions. An efficient update procedure is used to exactly minimize the 0/1 loss along the chosen direction. Both the randomness and the exact minimization procedure help escape from local minima. Theoretical analyses indicate that our algorithms globally minimize the 0/1 loss with arbitrarily high probability under simple settings, and perform random search towards an optimal hyperplane efficiently. Experimental results further demonstrate that our algorithms achieve the best 0/1 loss most of the time when compared with other perceptron algorithms, and are thus favorable base learners for ensemble learning methods such as AdaBoost [6].

The paper is organized as follows. We discuss some of the existing algorithms in Section II. We then introduce our random coordinate descent algorithms, as well as their convergence analyses, in Section III. Our algorithms are compared with existing ones, both as standalone learners and as base learners of AdaBoost, in Section IV. Finally, we conclude in Section V.

II. Related Work

We assume that the input space is a subset of $\mathbb{R}^m$. A perceptron consists of a weight vector $(w_1, \ldots, w_m)$ and a bias term b (i.e., the negative threshold). To avoid treating them separately, we use the notation $\mathbf{w} = (w_0, w_1, \ldots, w_m)$ with $w_0 = b$, and accordingly expand the original input vector to $\mathbf{x} \in \mathbb{R}^{m+1}$ with $x_0 = 1$. Then, the perceptron performs classification by the sign of the inner product between $\mathbf{w}$ and $\mathbf{x}$, i.e., $\operatorname{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$.

For a given training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with $y_i \in \{-1, 1\}$ being the labels, let $\rho_i = y_i \langle \mathbf{w}, \mathbf{x}_i \rangle$ be the unnormalized margin of the i-th example. Most perceptron algorithms try to minimize some cost function based on ρ_i:

$$C(\mathbf{w}) = \sum_{i=1}^{N} \varphi_i c(\rho_i), \tag{1}$$

where φ_i is the sample weight, and $c(\rho)$ is a margin cost function. Some common margin cost functions are listed

The authors are with the Learning Systems Group, Computer Science Option, California Institute of Technology, Pasadena, CA 91125, USA (email: {ling, htlin}@caltech.edu).

TABLE I
SOME COMMON MARGIN COST FUNCTIONS

cost function	$c(\rho)$
perceptron criterion	$\max\{0, -\rho\}$
SVM hinge loss	$\max\{0, 1-\rho\}$
modified least-squares	$(\max\{0, 1-\rho\})^2$
0/1 loss	$[\rho \le 0]$

in Table I. Note that many margin cost functions, including the first three in Table I, can be viewed as monotonic and continuous approximations of the 0/1 loss [7], which is a direct measure of the classification performance.

The well-known perceptron learning rule (PLR) [2] performs gradient descent on the perceptron criterion associated with some individual $(\mathbf{x}_i, y_i)$. It updates the weight vector $\mathbf{w}$ when it predicts wrongly on $\mathbf{x}_i$, i.e., when $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \le 0$,

$$\mathbf{w} \Leftarrow \mathbf{w} + y_i \mathbf{x}_i. \tag{2}$$

This update rule is applied repeatedly to every example in the training set. If the training set is separable, the perceptron convergence theorem [2] guarantees that a separating hyperplane can be found in finite time. However, if the training set is nonseparable, the algorithm will never converge and there is no guarantee for obtaining good perceptrons in terms of the 0/1 loss.

The pocket algorithm [4] solves the convergence problem of PLR at the price of much more computation. It runs PLR while keeping "in its pocket" the best-till-now weight vector in terms of the 0/1 loss. Although it can find an optimal weight vector that minimizes the 0/1 loss with arbitrarily high probability, the number of epochs required is prohibitively large in practice [4]. This is partly because the algorithm aims at minimizing the 0/1 loss, but only adopts (2) from PLR to update the weight vector. Since PLR is very unstable when the training set is nonseparable, much computation is wasted on bad weight vectors.

In contrast to the pocket algorithm, which uses only the best weight vector, Freund and Schapire [8] suggested combining all the weight vectors that occurred in the trials of PLR by a majority vote. One variant, which uses averaging instead of voting, can produce a single perceptron. Since it was shown that the voted- and the averaged-perceptron models perform similarly in practice [8], we will only consider the averaged-perceptron algorithm in this paper. The averaged-perceptron algorithm operates with the perceptron criterion, but does not explicitly minimize any cost functions. Thus, the obtained perceptron is usually not the best minimizer of either the perceptron criterion or the 0/1 loss.

When the margin cost function $c(\rho)$ satisfies certain smoothness assumptions, which is the case of all cost functions in Table I except the 0/1 loss, the stochastic gradient descent algorithm (SGD) [9] can be used to minimize the associated $C(\mathbf{w})$. For example, PLR is just a special case of SGD with the perceptron criterion. However, because the 0/1 loss has sharp transitions at $\rho = 0$, and zero gradients elsewhere, it cannot be directly minimized by SGD.

Minimizing the 0/1 loss for perceptrons is a challenging task. The problem is NP-complete [5], and hence deterministic optimization is likely to be inefficient. In addition, general numerical optimization cannot work because of the zero gradients, non-convexity, and non-smoothness. Practically, smooth and preferably convex approximations of the 0/1 loss are usually considered instead.

Nevertheless, the 0/1 loss is important because it captures the discrete nature of the binary classification (correct or incorrect). It is thus interesting to see whether a decent minimizer of the 0/1 loss could outperform minimizers of other cost functions. In addition, a good 0/1 loss minimizer can be useful in some practical cases. For example, adaptive boosting (AdaBoost), one of the most popular ensemble learning algorithms, expects an efficient and decent 0/1 loss minimizer as the base learner [6]. Such a good base learner could help AdaBoost in training speed and algorithmic convergence. However, existing perceptron algorithms mentioned above usually cannot be good base learners (see Section IV), because of slowness (e.g., pocket) and/or cost function mismatch (e.g., averaged-perceptron).

III. RANDOM COORDINATE DESCENT

From now on, we will focus on the 0/1 loss for perceptrons and propose a family of algorithms to directly minimize it. The notation $E(\mathbf{w})$, or the word *error*, will be used to specifically represent $C(\mathbf{w})$ with the 0/1 loss. Similar to (2), our algorithms updates the perceptron weight vector iteratively by choosing an update direction $\mathbf{d}$ and a proper descent step α,

$$\mathbf{w} \Leftarrow \mathbf{w} + \alpha \mathbf{d}. \tag{3}$$

We first show how to determine α for a given direction $\mathbf{d}$, and then discuss how we could choose $\mathbf{d}$.

A. Finding Optimal Descent Step

Assume that an update direction $\mathbf{d}$ has been chosen. To have the most decrease of $E(\mathbf{w})$ along the direction $\mathbf{d}$, we determine the best step size α^* by solving

$$\min_{\alpha \in \mathbb{R}} E(\mathbf{w} + \alpha \mathbf{d}) = \sum_{i=1}^{N} \varphi_i \left[y_i \langle \mathbf{w} + \alpha \mathbf{d}, \mathbf{x}_i \rangle \le 0 \right]. \tag{4}$$

We can take a closer look at the error that $(\mathbf{w} + \alpha \mathbf{d})$ makes on an individual example $(\mathbf{x}_i, y_i)$. Let $\delta_i = \langle \mathbf{d}, \mathbf{x}_i \rangle$.

- When $\delta_i \ne 0$, $\langle \mathbf{w} + \alpha \mathbf{d}, \mathbf{x}_i \rangle = \delta_i \left(\delta_i^{-1} \langle \mathbf{w}, \mathbf{x}_i \rangle + \alpha \right)$. The error of $(\mathbf{w} + \alpha \mathbf{d})$ on $(\mathbf{x}_i, y_i)$ is the same as the error of a 1-D decision stump [10] with bias α on the example $\left(\delta_i^{-1} \langle \mathbf{w}, \mathbf{x}_i \rangle, y_i \operatorname{sign}(\delta_i) \right)$.
- When $\delta_i = 0$, $\langle \mathbf{w} + \alpha \mathbf{d}, \mathbf{x}_i \rangle = \langle \mathbf{w}, \mathbf{x}_i \rangle$. Thus, the error does not change with α.

There exists a deterministic and efficient algorithm for minimizing the training error for decision stumps [10]. Hence, we can transform all training examples with $\delta_i \ne 0$ using $(\mathbf{x}_i, y_i) \mapsto \left(\delta_i^{-1} \langle \mathbf{w}, \mathbf{x}_i \rangle, y_i \operatorname{sign}(\delta_i) \right)$, and then apply the decision stump learning algorithm on the transformed training set to decide the optimal descent step α^*. Such an update procedure is illustrated in Fig. 1. Note that α^* is not restricted to be positive, and hence the direction $\mathbf{d}$ does not need to be strictly descent.

Input: A training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, its sample weight $\{\varphi_i\}_{i=1}^N$, the current $\mathbf{w}$, and an update direction $\mathbf{d}$

for $i = 1$ to N **do** {generate the 1-D training set}
  $\delta_i \Leftarrow \langle \mathbf{d}, \mathbf{x}_i \rangle$
  if $\delta_i \neq 0$ **then**
    $x_i' \Leftarrow \delta_i^{-1} \langle \mathbf{w}, \mathbf{x}_i \rangle$, $y_i' \Leftarrow y_i \operatorname{sign}(\delta_i)$
  end if
end for
Find the optimal decision stump only for those $\{(x_i', y_i')\}$ and $\{\varphi_i\}$ with $\delta_i \neq 0$:

$$\alpha^* \Leftarrow \operatorname*{argmin}_{\alpha \in \mathbb{R}} \sum_{i:\ \delta_i \neq 0} \varphi_i \left[y_i' \cdot \operatorname{sign}\left(x_i' + \alpha\right) \le 0 \right]$$

$\mathbf{w} \Leftarrow \mathbf{w} + \alpha^* \mathbf{d}$

Fig. 1. The update procedure that solves (4)

Input: A training set, sample weight, and the number of epochs T
Initialize $\mathbf{w}^{(1)}$
for $t = 1$ to T **do**
  Generate a random update direction $\mathbf{d}^{(t)} \in \mathbb{R}^{m+1}$
  Update $\mathbf{w}^{(t)}$ to $\mathbf{w}^{(t+1)}$ with the algorithm in Fig. 1
end for
return $\mathbf{w}^{(T+1)}$ as the perceptron weight vector

Fig. 2. Random coordinate descent (RCD) for perceptrons

B. Choosing Update Directions

There are many ways to choose an update direction. The simplest way might be the cyclic coordinate descent (CCD). It focuses on one basis vector in each epoch, and rotates through the $(m + 1)$ basis vectors repeatedly. Such a technique is commonly used when the cost function is not differentiable.

To avoid getting stuck in some local minima potentially caused by using fixed directions in a fixed order, we may choose the update directions randomly. This method, together with the update procedure, is called *random coordinate descent* (RCD) and is depicted in Fig. 2. Note that CCD can be thought as a degenerate variant of RCD. Below we will discuss some other representative variants of RCD algorithms.[1]

The simplest variant is called RCD-plain, in which the update directions are independent and identically-distributed random vectors. We have used two common distributions for generating the random vectors. The first one, called the *uniform random vectors*, picks each component of the vector from a uniform distribution spanned over the corresponding feature range. The other one uses Gaussian distribution estimated from the corresponding feature statistics, and is called the *Gaussian random vectors*.

Note that the final value of the bias, w_0, can be in quite different ranges from the other components of $\mathbf{w}$, due to the setting $x_0 = 1$. Thus, it might be helpful to have an update direction devoted to adjusting w_0 only. If the zeroth basis vector is additionally adopted as the update direction every $(m + 1)$ epochs, RCD-plain becomes RCD-bias.

Inspired by what PLR does, we may also use the gradient of the perceptron criterion associated with a randomly picked example $(\mathbf{x}_i, y_i)$ as the update direction. We call this variant RCD-grad. The difference between PLR and RCD-grad is that PLR does not pursue the optimal descent step.

C. Convergence of RCD

Next we analyze the convergence properties of RCD algorithms, which follows from random search techniques in optimization [12]. Note that both $E(\mathbf{w})$ and RCD algorithms are invariant to the magnitude of vectors $\mathbf{w}$ or $\mathbf{d}$. For easier analysis, we assume that every nonzero $\mathbf{w}^{(t)}$ or $\mathbf{d}^{(t)}$ is normalized to unit length.

First, we show a general result of global convergence that applies to RCD-plain and RCD-bias, with either uniform or Gaussian random vectors.

Definition 1: We call $\left\{\mathbf{d}^{(t)}\right\}_{t=1}^\infty$ *sufficiently random* if $\prod_{t=1}^\infty \left(1 - \Pr\left[\mathbf{d}^{(t)} \in A\right]\right) = 0$ for all $A \subseteq S \cup \{\mathbf{0}\}$ with nonzero measure, where $S = \{\mathbf{w} \colon \|\mathbf{w}\| = 1\}$.

Theorem 1: For an RCD algorithm with sufficiently random directions $\left\{\mathbf{d}^{(t)}\right\}_{t=1}^\infty$,

$$\lim_{t \to \infty} \Pr\left[\mathbf{w}^{(t)} \text{ minimizes } E(\mathbf{w})\right] = 1. \tag{5}$$

Proof: We extend from the global search convergence theorem [12, page 20], which, together with the property that $E(\mathbf{w})$ is lower bounded and evaluates to discrete values, implies the correctness of (5) under the following conditions:

1) $S \cup \{\mathbf{0}\}$ is a measurable set; $E(\mathbf{w})$ is measurable.
2) The directions $\left\{\mathbf{d}^{(t)}\right\}_{t=1}^\infty$ are sufficiently random.
3) The sequence $\left\{E(\mathbf{w}^{(t)})\right\}_{t=1}^\infty$ is non-increasing, and $E(\mathbf{w}^{(t+1)}) \le E(\mathbf{d}^{(t)})$.

The proof relies on the third condition, which holds because α^* is the minimizer of $E(\mathbf{w}^{(t)} + \alpha \mathbf{d}^{(t)})$ in Fig. 1. ■

We can easily verify that the assumption of Theorem 1 is satisfied by RCD-plain and RCD-bias, with either the uniform or the Gaussian random vectors. However, Theorem 1 does not fully reveal their efficiency. Next, we take RCD-plain with the Gaussian random vectors $\mathbf{d}^{(t)} \sim \mathcal{N}(\mathbf{0}, 1)$ as an example, and show how it performs efficient random search.

Define $B(\mathbf{w}^*, R)$ as a hypersphere in $\mathbb{R}^{m+1}$ with center $\mathbf{w}^*$ and radius R. Let

$$A(\mathbf{w}^*, R) = \{\mathbf{w} / \|\mathbf{w}\| : \mathbf{w} \in B(\mathbf{w}^*, R) \setminus \{\mathbf{0}\}\} \subset S.$$

An integration on S shows that when $\|\mathbf{w}^*\| = 1$, the probability of the direction $\mathbf{d}^{(t)} \in A(\mathbf{w}^*, R)$ is

$$m \cdot v_m \int_0^R r^{m-2}\, d\sqrt{1 - r^2} \ge v_m R^m, \tag{6}$$

where v_m is some constant [13]. Note that in the space of weight $\mathbf{w}$, the N hyperplanes $\langle \mathbf{x}_i, \mathbf{w} \rangle = 0$ cut the surface S into different regions, each of which represents a specific prediction pattern on the training set. Following the analysis of Dunagan and Vempala [14], each region can be approximately modeled by some $A(\mathbf{w}^*, R)$ with $\|\mathbf{w}^*\| = 1$.[2] Assume that $\mathbf{w}^*$ is from a region where $E(\mathbf{w}^*) < E(\mathbf{w}^{(t)})$.

[1]More variants can be found in our earlier technical report [11].

[2]Within the region, we choose a $\mathbf{w}^*$ such that R is maximized.

Equation (6) gives a lower bound for locating the region in a naive random search. However, RCD-plain can perform the search more efficiently from the current $\mathbf{w}^{(t)}$:

Theorem 2: For the piece $A(\mathbf{w}^*, R)$ defined above, if the angle between $\mathbf{w}^*$ and $\mathbf{w}^{(t)}$ is $\phi \in [0, \frac{\pi}{2}]$, then RCD-plain with the standard Gaussian random vectors satisfies

$$\Pr\left[E(\mathbf{w}^{(t+1)}) \leq E(\mathbf{w}^*)\right] \geq v_m R^m \big/ \sin^m \phi.$$

Proof: The scale invariance of $E(\mathbf{w})$ implies that $E(\mathbf{w}) = E(\mathbf{w}^*)$ for any $k > 0$ and $\mathbf{w} \in B(k\mathbf{w}^*, kR)$. Consider $\mathbf{d}^{(t)}$ such that the line $\mathbf{w}^{(t)} + \alpha \mathbf{d}^{(t)}$ intersects $B(k\mathbf{w}^*, kR)$, which is equivalent to $\mathbf{d}^{(t)} \in A_k$ with $A_k = A(k\mathbf{w}^* - \mathbf{w}^{(t)}, kR)$. Thus, for any $k > 0$,

$$\Pr\left[E(\mathbf{w}^{(t+1)}) \leq E(\mathbf{w}^*)\right] \geq \Pr\left[\mathbf{d}^{(t)} \in A_k\right].$$

The theorem is proved by setting $k = 1/\cos\phi$, which results in a piece A_k with the maximum measure. ∎

Similar analysis can be carried out for the case of the uniform or other Gaussian random vectors. When $\mathbf{w}^{(t)}$ and $\mathbf{w}^*$ are close, the angle ϕ between them is small, and RCD-plain greatly improves the lower bound of the probability for decreasing $E(\mathbf{w})$. In other words, RCD-plain performs fast local search for a better $\mathbf{w}$ nearby. Note that because of the structure of the $\mathbf{w}$-space, a region with global minima is close to regions with small training errors. With the implicit use of the structure, and the ability to escape from local minima, RCD-plain (and similarly RCD-bias) can hence perform global minimization efficiently.

IV. Experiments

We first compare the performance of different RCD variants in minimizing the training error. Some better variants are further compared with the existing perceptron algorithms. We use nine real-world data sets[3] from the UCI machine learning repository [15], with 80% of randomly chosen examples for training and the rest for testing. Three artificial data sets[4] are also randomly generated, with 600 examples for training and 4400 for testing. The 0/1 loss (error) is measured over 500 runs on the training and testing sets, and is presented using the mean and the standard error.

a) Data Preprocessing.: The features in the training set are first linearly normalized to $[-1, 1]$ solely based on the training examples. Then the test examples are normalized accordingly.

b) Initial Seeding.: We initialize $\mathbf{w}^{(1)}$ with either the zero vector or the Fisher's linear discriminant (FLD, see for example [17]). For the latter case, any singular estimate of the within-class covariance matrix is regularized with an eigenvalue shrinkage parameter of 10^{-10} [18].

A. Comparisons within RCD Variants

In Fig. 3, we compare different RCD variants using their training errors on the pima data set. The results shown are based on uniform random vectors, while similar results have been observed for Gaussian random vectors. Some important observations are:

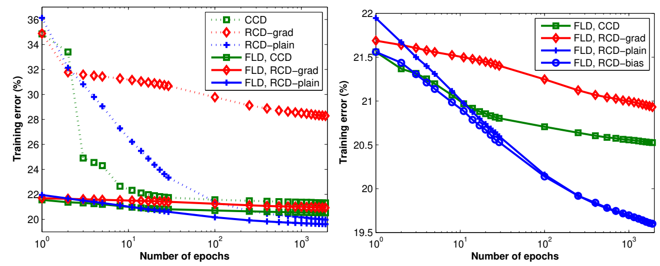

Fig. 3. Training errors of RCD algorithms on pima

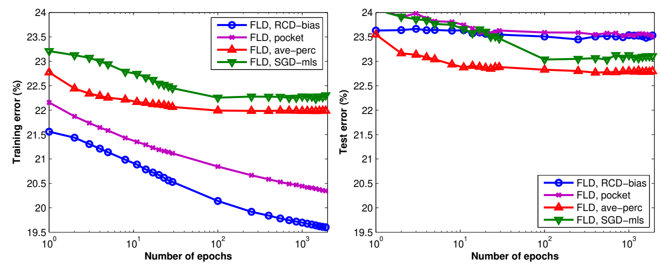

Fig. 4. Training and test errors of stochastic algorithms on pima

- With the same number of epochs, training errors of RCD algorithms using the FLD seeding are significantly lower than those using the zero seeding.
- Both CCD and RCD-grad do not work as well as RCD-plain and RCD-bias. This result confirms that sufficiently random directions are important for RCD.
- RCD-bias is better than RCD-plain, especially at the beginning. However, the edge gets smaller with more training epochs.

We obtain similar observations in some other data sets. Thus, RCD-plain and RCD-bias, with the FLD seeding, are more promising. Next, we compare them with some existing perceptron algorithms. For a fair comparison, we also equip the other algorithms with the FLD seeding.

B. Comparisons as Standalone Learners

We consider the pocket algorithm with ratchet (denoted as pocket) [4], an improved variant of the averaged-perceptron algorithm, in which examples are presented completely randomly during training (ave-perc) [8, 11], SGD with a learning rate 0.002 on the modified least-squares (SGD-mls) [9], and the linear soft-margin SVM with parameter selection (soft-SVM) [19, 20]. The first three stochastic algorithms can be directly compared to RCD algorithms by allowing the same $T = 2000$ epochs, and the last one is included for reference.

Fig. 4 presents the performance of the stochastic algorithms on the pima data set.[5] In the competition for low training errors, RCD-bias is clearly the best, and pocket

[3] They are australian, breast, cleveland, german, heart, ionosphere, pima, sonar, and votes84. See [11] for details.

[4] They are ringnorm, threenorm [16], and yinyang. See [11] for details.

[5] The curves of RCD-plain are very close to those of RCD-bias, and are thus not shown.

TABLE II
TRAINING ERRORS (%) OF PERCEPTRON ALGORITHMS USING THE FLD SEEDING

data set	RCD-plain	RCD-bias	pocket	ave-perc	SGD-mls	soft-SVM
australian	10.12 ± 0.03	$\mathbf{9.98 \pm 0.03}$	10.81 ± 0.03	12.19 ± 0.03	12.70 ± 0.04	14.33 ± 0.03
breast	$\mathbf{1.68 \pm 0.01}$	$\mathbf{1.68 \pm 0.01}$	1.86 ± 0.01	2.87 ± 0.02	2.77 ± 0.02	2.70 ± 0.02
cleveland	$\mathbf{10.57 \pm 0.05}$	$\mathbf{10.62 \pm 0.05}$	12.07 ± 0.05	14.40 ± 0.06	14.48 ± 0.06	14.74 ± 0.05
german	19.16 ± 0.04	$\mathbf{18.80 \pm 0.03}$	21.10 ± 0.03	21.31 ± 0.04	22.18 ± 0.05	21.48 ± 0.04
heart	$\mathbf{9.48 \pm 0.05}$	$\mathbf{9.49 \pm 0.05}$	11.22 ± 0.05	13.64 ± 0.06	13.82 ± 0.06	14.20 ± 0.06
ionosphere	3.88 ± 0.04	3.97 ± 0.04	$\mathbf{3.41 \pm 0.05}$	4.92 ± 0.06	5.14 ± 0.05	6.95 ± 0.10
pima	$\mathbf{19.60 \pm 0.04}$	$\mathbf{19.60 \pm 0.03}$	20.34 ± 0.03	21.99 ± 0.04	22.25 ± 0.04	22.09 ± 0.04
ringnorm	27.61 ± 0.07	$\mathbf{27.36 \pm 0.08}$	30.46 ± 0.07	35.49 ± 0.11	34.52 ± 0.13	31.82 ± 0.09
sonar	2.56 ± 0.04	2.62 ± 0.04	$\mathbf{0.00 \pm 0.00}$	0.37 ± 0.02	1.42 ± 0.06	11.58 ± 0.20
threenorm	$\mathbf{11.41 \pm 0.06}$	$\mathbf{11.39 \pm 0.06}$	13.53 ± 0.06	14.43 ± 0.06	14.51 ± 0.06	14.47 ± 0.06
votes84	$\mathbf{1.32 \pm 0.02}$	$\mathbf{1.31 \pm 0.02}$	1.46 ± 0.02	2.42 ± 0.03	2.48 ± 0.03	3.02 ± 0.04
yinyang	$\mathbf{15.33 \pm 0.05}$	$\mathbf{15.36 \pm 0.05}$	15.61 ± 0.05	19.10 ± 0.07	19.03 ± 0.07	18.89 ± 0.08

(results within one standard error of the best are marked in bold)

TABLE III
TEST ERRORS (%) OF PERCEPTRON ALGORITHMS USING THE FLD SEEDING

data set	RCD-plain	RCD-bias	pocket	ave-perc	SGD-mls	soft-SVM
australian	14.24 ± 0.12	13.92 ± 0.12	14.31 ± 0.12	$\mathbf{13.64 \pm 0.12}$	13.87 ± 0.12	14.78 ± 0.12
breast	3.65 ± 0.07	3.61 ± 0.07	3.43 ± 0.06	3.36 ± 0.06	$\mathbf{3.28 \pm 0.06}$	$\mathbf{3.22 \pm 0.06}$
cleveland	18.68 ± 0.22	18.57 ± 0.21	18.49 ± 0.21	$\mathbf{16.74 \pm 0.20}$	$\mathbf{16.76 \pm 0.20}$	$\mathbf{16.72 \pm 0.20}$
german	24.45 ± 0.12	23.70 ± 0.13	25.24 ± 0.13	$\mathbf{23.24 \pm 0.12}$	24.05 ± 0.13	23.64 ± 0.12
heart	18.13 ± 0.21	18.20 ± 0.22	17.63 ± 0.20	$\mathbf{16.51 \pm 0.20}$	$\mathbf{16.49 \pm 0.20}$	$\mathbf{16.45 \pm 0.20}$
ionosphere	13.91 ± 0.17	14.72 ± 0.18	12.87 ± 0.18	12.76 ± 0.18	$\mathbf{12.63 \pm 0.18}$	$\mathbf{12.57 \pm 0.17}$
pima	23.79 ± 0.14	23.50 ± 0.14	23.50 ± 0.14	$\mathbf{22.79 \pm 0.14}$	23.07 ± 0.14	23.19 ± 0.14
ringnorm	35.83 ± 0.04	$\mathbf{35.65 \pm 0.04}$	36.59 ± 0.04	39.27 ± 0.08	38.38 ± 0.10	35.70 ± 0.05
sonar	25.98 ± 0.29	26.20 ± 0.29	25.20 ± 0.25	25.09 ± 0.26	24.90 ± 0.28	$\mathbf{23.89 \pm 0.27}$
threenorm	16.82 ± 0.03	16.86 ± 0.03	17.65 ± 0.04	16.14 ± 0.02	16.18 ± 0.02	$\mathbf{16.08 \pm 0.02}$
votes84	5.21 ± 0.09	5.00 ± 0.10	5.24 ± 0.10	4.52 ± 0.10	4.70 ± 0.11	$\mathbf{4.39 \pm 0.09}$
yinyang	$\mathbf{17.71 \pm 0.02}$	17.75 ± 0.02	17.74 ± 0.02	19.25 ± 0.02	19.21 ± 0.02	19.21 ± 0.02

(results within one standard error of the best are marked in bold)

follows. However, when the test error is concerned, the other three methods, especially ave-perc, are the winners. The final performance of all perceptron algorithms are shown in Tables II and III. Similarly, RCD-plain and RCD-bias achieve the lowest training errors for most of the data sets. On the other hand, soft-SVM and ave-perc, which are known to be regularized, could usually obtain better test errors. Thus, the 0/1 loss itself may not be the best cost function, and overfitting without regularization shall explain the inferior test performance of RCD algorithms.

We also observe that pocket is much slower than RCD-bias, because with nonseparable data sets, checks on whether a new weight vector should replace the in-pocket one may happen very often. In addition, pocket usually also needs more epochs in order to achieve the same level of training error as RCD-bias.

C. Comparisons as AdaBoost Base Learners

AdaBoost expects its base learners to efficiently find a hypothesis with low weighted training error. The details for plugging the sample weights φ_i into existing perceptron algorithms are listed in our technical report [11]. We use the reweighting technique for pocket and ave-perc. Then, we set $T = 200$ for all perceptron algorithms, which seems to be sufficient for all the data sets, and apply them as AdaBoost base learners. We run AdaBoost for up to 200 iterations, and report the results with the zero seeding in Table IV, while similar results have been obtained with the FLD seeding.

We observe that ave-perc, SGD-mls, and soft-SVM, which are not designed for the weighted training error, usually fail to return a decent perceptron, and cause AdaBoost to stop at some early iteration [11]. That is, their performance is mainly associated with a few regularized perceptrons rather than with AdaBoost. On the other hand, AdaBoost with RCD-plain, RCD-bias, or pocket always runs through all 200 iterations, and hence their performance is fully connected to AdaBoost.

To further compare perceptron algorithms as a base learners to AdaBoost, we mark the entries with * in Table IV to denote a significant improvement from the best entry of Table III. There are some data sets without *, which means that they are simple enough, and can be modeled sufficiently well with one single perceptron (e.g., australian). For those data sets, ave-perc, SGD-mls, and soft-SVM have good performance both as standalone learners and with AdaBoost. However, since they already perform well as standalone learners, there is no need in binding them with AdaBoost on those data sets.

On the other hand, for complex data sets that can be better modeled by AdaBoost with perceptrons (e.g., yinyang), base learners that aim at minimizing the 0/1 loss (RCD-plain, RCD-bias, or pocket) perform significantly better than others. In addition, RCD-plain and RCD-bias are usually better than pocket in terms of both the training speed and the performance. Thus, RCD-plain and RCD-bias fit the needs

TABLE IV
TEST ERRORS (%) OF ADABOOST WITH PERCEPTRON ALGORITHMS USING THE ZERO SEEDING

data set	RCD-plain	RCD-bias	pocket	ave-perc	SGD-mls	soft-SVM
australian	15.45 ± 0.12	15.49 ± 0.12	15.75 ± 0.12	$\mathbf{13.61 \pm 0.12}$	14.00 ± 0.12	15.65 ± 0.13
breast	$\mathbf{3.21 \pm 0.06}$	3.34 ± 0.06	3.41 ± 0.07	3.35 ± 0.06	$\mathbf{3.24 \pm 0.06}$	$\mathbf{3.20 \pm 0.06}$
cleveland	18.00 ± 0.21	18.22 ± 0.21	18.95 ± 0.20	$\mathbf{16.81 \pm 0.20}$	$\mathbf{16.74 \pm 0.20}$	$\mathbf{16.69 \pm 0.21}$
german	25.17 ± 0.13	25.37 ± 0.12	25.57 ± 0.13	$\mathbf{23.25 \pm 0.12}$	23.96 ± 0.13	23.48 ± 0.12
heart	17.60 ± 0.21	17.58 ± 0.22	18.94 ± 0.21	$\mathbf{16.55 \pm 0.20}$	$\mathbf{16.54 \pm 0.20}$	$\mathbf{16.57 \pm 0.21}$
ionosphere	$\mathbf{10.36 \pm 0.16}^*$	$\mathbf{10.30 \pm 0.16}^*$	$11.65 \pm 0.17^*$	13.21 ± 0.17	12.67 ± 0.17	$10.83 \pm 0.16^*$
pima	24.87 ± 0.14	24.79 ± 0.14	25.15 ± 0.14	$\mathbf{22.77 \pm 0.14}$	23.01 ± 0.14	23.06 ± 0.13
ringnorm	$8.60 \pm 0.05^*$	$12.22 \pm 0.07^*$	$\mathbf{7.12 \pm 0.04}^*$	39.29 ± 0.08	38.32 ± 0.09	$15.49 \pm 0.17^*$
sonar	$16.44 \pm 0.25^*$	$\mathbf{16.06 \pm 0.25}^*$	25.02 ± 0.27	25.77 ± 0.27	25.37 ± 0.27	$21.52 \pm 0.25^*$
threenorm	$\mathbf{14.51 \pm 0.02}^*$	$15.34 \pm 0.03^*$	$14.95 \pm 0.02^*$	$16.14 \pm 0.02^*$	$16.17 \pm 0.02^*$	$15.97 \pm 0.02^*$
votes84	$\mathbf{4.25 \pm 0.09}^*$	$\mathbf{4.24 \pm 0.09}^*$	4.54 ± 0.10	4.74 ± 0.10	4.68 ± 0.10	4.82 ± 0.10
yinyang	$\mathbf{3.95 \pm 0.03}^*$	$\mathbf{3.98 \pm 0.03}^*$	$4.87 \pm 0.02^*$	19.25 ± 0.02	19.23 ± 0.02	19.14 ± 0.02

(results within one standard error of the best entry in the row are marked in bold)
(results better than the best of Table III by one standard error are marked with *)

of AdaBoost, and can almost always achieve the best test error in this situation. The results demonstrate the usefulness of RCD algorithms as base learners of AdaBoost.

V. CONCLUSION

We have proposed a family of new learning algorithms to directly optimize the 0/1 loss for perceptrons. The main ingredients are random coordinate descent (RCD) and an update procedure to exactly minimize the 0/1 loss along the update direction. We have proved the convergence of our RCD algorithms, and have also analyzed their speedup heuristic. Experimental results have confirmed that RCD algorithms are efficient, and usually achieve the lowest in-sample 0/1 loss compared with several existing perceptron learning algorithms.

RCD algorithms can be used not only as standalone learners, but also as favorable base learners for ensemble learning. In terms of the improvement for the out-of-sample performance, our experimental results have demonstrated that AdaBoost works better with RCD algorithms than with other existing perceptron algorithms.

On some data sets on which a single perceptron works well, our results indicate that RCD-based 0/1 loss minimizer may not be the best choice. On the other hand, regularized algorithms such as the averaged-perceptron and the soft-margin SVM could achieve better out-of-sample performance. Future work will be focused on RCD algorithms for some regularized 0/1 loss.

ACKNOWLEDGMENTS

We wish to thank Yaser Abu-Mostafa and Amrit Pratap for many valuable discussions. This work was mainly supported by the Caltech SISL Graduate Fellowship.

REFERENCES

[1] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386–408, 1958.

[2] ——, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, 1962.

[3] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[4] S. I. Gallant, "Perceptron-based learning algorithms," *IEEE Transactions on Neural Networks*, vol. 1, pp. 179–191, 1990.

[5] P. Marcotte and G. Savard, "Novel approaches to the discrimination problem," *Mathematical Methods of Operations Research*, vol. 36, pp. 517–545, 1992.

[6] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Machine Learning: Proceedings of the Thirteenth International Conference (ICML '96)*, L. Saitta, Ed., 1996, pp. 148–156.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, ser. Springer Series in Statistics. Springer-Verlag, 2001.

[8] Y. Freund and R. E. Schapire, "Large margin classification using the perceptron algorithm," *Machine Learning*, vol. 37, pp. 277–296, 1999.

[9] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *ICML 2004: Proceedings of the Twenty-First International Conference on Machine Learning*, R. Greiner and D. Schuurmans, Eds., 2004.

[10] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine Learning*, vol. 11, pp. 63–91, 1993.

[11] L. Li, "Perceptron learning with random coordinate descent," California Institute of Technology, Pasadena, CA, Computer Science Technical Report CaltechCSTR:2005.006, 2005.

[12] F. J. Solis and R. J.-B. Wets, "Minimization by random search techniques," *Mathematics of Operations Research*, vol. 6, pp. 19–30, 1981.

[13] S. Axler, P. Bourdon, and W. Ramey, *Harmonic Function Theory*, 2nd ed., ser. Graduate Texts in Mathematics. Springer-Verlag, 2001.

[14] J. Dunagan and S. Vempala, "A simple polynomial-time rescaling algorithm for solving linear programs," in *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, L. Babai, Ed., 2004, pp. 315–320.

[15] S. Hettich, C. L. Blake, and C. J. Merz, "UCI repository of machine learning databases," 1998, available at http://www.ics.uci.edu/~mlearn/MLRepository.html.

[16] L. Breiman, "Arcing classifiers," *The Annals of Statistics*, vol. 26, pp. 801–824, 1998.

[17] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[18] J. H. Friedman, "Regularized discriminant analysis," *Journal of the American Statistical Association*, vol. 84, pp. 165–175, 1999.

[19] C.-C. Chang and C.-J. Lin, *LIBSVM: A library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[20] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification," National Taiwan University, Tech. Rep., 2003.