A simpler unified analysis of Budget Perceptrons

Ilya Sutskever ILYA@CS.UTORONTO.CA
University of Toronto, 6 King's College Rd., Toronto, Ontario, M5S 3G4, Canada

Abstract

The kernel Perceptron is an appealing online learning algorithm that has a drawback: whenever it makes an error it must increase its support set, which slows training and testing if the number of errors is large. The Forgetron and the Randomized Budget Perceptron algorithms overcome this problem by restricting the number of support vectors the Perceptron is allowed to have. These algorithms have regret bounds whose proofs are dissimilar. In this paper we propose a unified analysis of both of these algorithms by observing that the way in which they remove support vectors can be seen as types of L_2-regularization. By casting these algorithms as instances of online convex optimization problems and applying a variant of Zinkevich's theorem for noisy and incorrect gradient, we can bound the regret of these algorithms more easily than before. Our bounds are similar to the existing ones, but the proofs are less technical.

1. Introduction

Traditional batch learning algorithms update their parameters after processing every datapoint in the training set. In contrast, online learning algorithms update their parameters based on small batches of training examples. When the training set is large enough, batch algorithms will learn more slowly than online algorithms (e.g., (Bottou & Bousquet, 2008; Shalev-Shwartz & Srebro, 2008)). For example, if the training set is redundant, online learning algorithms can finish learning before completing their first pass over the training set, which is not possible with batch algorithms.

Appearing in *Proceedings of the 26^{th} International Conference on Machine Learning*, Montreal, Canada, 2009.

1.1. Perceptrons

The Perceptron (Rosenblatt, 1958) is the oldest and one of the most widespread online learning algorithms. It classifies n-dimensional real vectors into one of two possible classes (which are 1 and -1), and stores its knowledge in the form of an n-dimensional weight vector w. It predicts the class $p = \text{sign}(w^\top x)$ for the input vector x, and updates its parameters by the equation

$$w_{i+1} = w_i + x_i \cdot (y_i - p_i)/2 \tag{1}$$

where $y_i \in \{1, -1\}$ is the label of the example x_i, and $p_i = \text{sign}(w_i^\top x_i)$ is the Perceptron's prediction on the example x_i. This update is nonzero only when the Perceptron makes a mistake.

There are several variants on the Perceptron learning algorithm (e.g., (Shalev-Shwartz & Singer, 2005; Shalev-Shwartz et al., 2007; Crammer et al., 2006; Littlestone & Warmuth, 1989); see references in (Cesa-Bianchi & Lugosi, 2006)). They are similar as algorithms and have formal performance guarantees. The oldest guarantee of this kind is Novikoff's theorem (Novikoff, 1963), which states that the number of errors that the Perceptron makes is at most $1/\gamma^2$, where γ is the margin of a set of weights w^* that makes no errors on the data (in this setting, the margin is the distance between the hyperplane defined by this Perceptron and the point closest to it; the theorem assumes that $\|x\| \leq 1$).[1]

1.2. Kernels

Despite its simplicity and appeal, the Perceptron can represent a fairly limited set of hypotheses, and there are many practical problems for which the Perceptron simply cannot obtain low *training* error. This difficulty was first overcame by the Support Vector Machine (SVM) (Cortes & Vapnik, 1995), which is a nonlinear Perceptron that works very well in practice. The strength of the SVM comes from two main sources. First, it uses the kernel trick (Scholkopf & Smola, 2002), which is a way of dramatically enhancing the set of hypothesis representable by the Perceptron. Consider mapping a training example x to an expanded

[1] In this paper, all norms are L_2.

set of features $\phi(x)$; for example, $\phi((x_1, \ldots, x_n)) = \left(\prod_{j \in S} x_j\right)_{S \subseteq \{1,\ldots,n\}}$ (a 2^n-dimensional vector). If, instead of classifying x, we classify $\phi(x)$ with a Perceptron, we could represent a much richer set of hypotheses (simply because Perceptrons that classify $\phi(x)$ have 2^n parameters while Perceptrons that classify x have only n). Using $\phi(x)$ directly is obviously infeasible due to its size; however, the inner product $\langle \phi(x), \phi(y) \rangle = \prod_{i=1}^{n} (1 + x_i \cdot y_i)$ is efficiently computable, and the kernel trick makes essential use of this efficiency: notice that if we run the Perceptron on examples of the form $\phi(x)$, then

$$w = \sum_{i=1}^{M} \phi(x_i) \alpha_i \tag{2}$$

where $\alpha_i \in \pm 1$, M is the number of errors, and x_i are the errors themselves. When w has this form, we can compute the Perceptron's predictions by the equation

$$\operatorname{sign} \langle w, \phi(x) \rangle = \operatorname{sign} \sum_{i=1}^{M} \langle \phi(x_i), \phi(x) \rangle \, \alpha_i \tag{3}$$

which is a dramatically more efficient than working with the $\phi(x)$'s directly. The inner product $\langle \phi(x_i), \phi(x) \rangle$ is represented by the kernel function $k(x, y)$. Following (Dekel et al., 2008), we call $\{x_1, \ldots, x_M\}$ the active set of w.

In addition to the kernel trick, the SVM uses regularization and minimizes the hinge loss, which prevents the SVM from overfitting and causes it to find sparse solutions– vectors w that can be expressed as a linear combination of a small number of $\phi(x_i)$'s.

The hinge loss function L is defined by the equation

$$L(w; (x, y)) = \max(0, 1 - \langle w, x \rangle \, y) \tag{4}$$

and given a training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, the SVM minimizes the cost function

$$\sum_{i=1}^{n} L(w; (x_i, y_i)) + \lambda \cdot \|w\|^2 / 2 \tag{5}$$

An important property of the hinge loss is that if w does not classify x correctly (i.e., $\operatorname{sign} \langle w, x \rangle \neq y$), then $L(w; (x, y)) > 1$, implying that the total hinge loss of the training set upper bounds the number of errors in the training set (e.g., (Shalev-Shwartz, 2007)).

1.3. Budget

SVMs are usually trained with batch algorithms, but it is tempting to apply the plain Perceptron to the vectors $\phi(x)$, as described in the previous sections, in order to obtain an online learning algorithm for the Kernel Perceptron. The main computational challenge in doing so is computing the inner products $\langle w, \phi(x) \rangle$. To do so, we must store the active set $\{x_1, \ldots, x_M\}$ and evaluate the kernel function M times, so the time to compute $\langle w, \phi(x) \rangle$ grows linearly with the size of the active set. If the Perceptron makes a large number of errors during training, the size of its active set will become large (see eq. 2), which will significantly slow both training and testing.

There have been a number of attempts to fix the problem of large active sets in the online kernel Perceptron (Crammer et al., 2004; Weston et al., 2005; Dekel et al., 2008; Cavallanti et al., 2007), the last two of which have formal performance guarantees. These algorithms enforce a strict upper bound B (the budget) on the active set, making sure that these Perceptrons are efficient. They do so by removing a vector from the active set whenever the active set becomes too large.

We will now describe the two algorithms relevant to this work. The Forgetron (Dekel et al., 2008) is the first budget Perceptron that had a formal performance guarantee. It enforces a strict bound on the active set by removing vectors as follows: first, it reduces the weight α_i of every vector in its active set; then, it discards the oldest vector in the active set, which has the smallest weight. This is done only when the size of the active set exceeds the budget B, so applying this removal procedure on every error will ensure that the size of the active set will not exceed the budget. Due to the repeated weight reductions, the oldest vector will have small weight, so removing it will not change the Perceptron significantly. The factor by which the Forgetron reduces weight is not constant and differs from error to error.

The Randomized Budget Perceptron (RBP) of (Cavallanti et al., 2007) is considerably simpler than the Forgetron: whenever the Perceptron makes an error and the size of the active set exceeds the budget, RBP discards a random vector from its active set.

Both algorithms have regret bounds whose proofs are different and are somewhat nontrivial. In this paper, we present a view that lets us reprove similar regret bounds for the RBP and for a simplified Forgetron (one that reduces the weights by the same factor on each error) with less effort using the same idea. The idea is that both algorithms can be seen as doing online gradient descent on the hinge loss with L_2-regularization on their errors, where the RBP algorithm adds a special kind of zero-mean noise to the gradient and the Forgetron has a small deterministic error in its gradient. Thus, we will view these algorithms as performing online convex programming on the points where the algorithms err, and apply a version of Zinkevich's theorem (Zinkevich, 2003) that accounts for a changing target and errors in the gradients, which we will prove in the next section. This will let us bound the total number of errors

with the hinge loss of the best slowly-changing sequence of hypotheses. As a result, obtain similar proofs for a regret bound of the RBP and the simplified Forgetron. The regret bound that we prove for the RBP has a better dependence on the budget size than (Cavallanti et al., 2007), but it is competitive against hypotheses of a slightly smaller norm.

An unusual aspect of the regret bound of the RBP is that it is shown to be competitive not against a fixed Perceptron, but against a sequence of slowly-changing Perceptrons. Our bounds for both the Forgretron and the RBP will also be able to cope with a sequence of slow-changing Perceptrons, thanks to lemma 1 below.

2. Online convex programming with errors in the gradients

Online convex programming (OCP) is the setting where in each round, a convex function f is chosen; we choose a point x without knowing f and experience $f(x)$ loss. Our goal is to have a cumulative loss that is not much greater than the cumulative loss of *any* fixed point x^*. Zinkevich's theorem states that this goal is attained with simple gradient descent. We will apply Zinkevich's theorem, where the function f is an L_2-regularized hinge loss of an unknown datapoint.

Our analysis requires a variant of Zinkevich's theorem that accounts both for tracking—i.e., is competitive against a slowly-changing sequence of hypotheses—and for errors in the gradient. A tracking version of Zinkevich's theorem is already known (Zinkevich, 2003); however, the existing variants do not account for errors in the gradient. We also need the result to hold for noisy gradients, but this is done with the application of the idea of lemma 2 in (Flaxman et al., 2005) (see subsection 3.1).

Lemma 1:

Let C be a closed convex set of diameter U.

Let $f_1, \ldots, f_T$ be a sequence of arbitrary convex functions (in the analysis below, each f_i will be an L_2-regularized hinge loss).

Let $E_1, \ldots, E_T$ be an arbitrary sequence of vectors (the gradient errors), and let $E = \sum_{i=1}^{T} \|E_i\|$.

Let $\eta > 0$ be the learning rate, and let $w_1 \in C$ be an arbitrary initial point.

Define $w_{i+1} = \pi_C(w_i - \eta \cdot \partial_i)$, where $\partial_i = f_i'(w_i) + E_i$, and $f_i'(w_i)$ is any subgradient of f_i at w_i. The variable ∂_i is a gradient with error, and $\pi_C(x)$ is the closest point of C to x.

Let G be such that $\|\partial_i\| \leq G$ holds for all i.

Let $w_0', \ldots, w_T' \in C$ be an arbitrary sequence of comparison vectors, and let $S = \sum_{i=1}^{T} \|w_{i-1}' - w_i'\|$ be their total shift.

Then

$$\sum_{i=1}^{T} (f_i(w_i) - f_i(w_i')) \leq \frac{U^2}{2\eta} + \frac{3 \cdot US}{2\eta} + \frac{T \cdot \eta \cdot G^2}{2} + U \cdot E \tag{6}$$

Note that if there is no shift ($S = 0$) and there are no errors in the gradient ($E = 0$), then we get the original form of Zinkevich's theorem. The variable w_0' is introduced only to make the proof shorter and can be eliminated by setting $w_0' = w_1'$, causing the shift to be $S = \sum_{i=1}^{T-1} \|w_i' - w_{i+1}'\|$. Note, also, that the ability to cope with a changing optimal solution is inversely proportional to the learning rate: the smaller the learning rate, the harder it is to keep track of change, which makes intuitive sense.

Proof:

Our proof is standard and is very similar to (Zinkevich, 2003). It measures the speed with which we approach the competing points.

For $i = 1, \ldots, T$ let $D_i = \|w_i - w_{i-1}'\|^2 - \|w_{i+1} - w_i'\|^2$. Then

$$\begin{aligned}
D_i &= \|w_i - w_{i-1}'\|^2 - \|w_{i+1} - w_i'\|^2 \\
&= \|w_i - w_{i-1}'\|^2 - \|\pi_C(w_i - \eta \cdot \partial_i) - w_i'\|^2 \\
&\geq_1 \|w_i - w_{i-1}'\|^2 - \|w_i - \eta \cdot \partial_i - w_i'\|^2 \\
&= \|w_i - w_{i-1}'\|^2 - \\
&\quad \|(w_i - w_{i-1}') + (w_{i-1}' - w_i') - \eta \cdot \partial_i\|^2 \\
&= \|w_i - w_{i-1}'\|^2 - \|w_i - w_{i-1}'\|^2 - \\
&\quad \|w_{i-1}' - w_i'\|^2 - \eta^2 \cdot \|\partial_i\|^2 + \\
&\quad 2\eta(f_i'(w_i) + E_i)^\top (w_i - w_i') - \\
&\quad 2(w_i - w_{i-1}')^\top (w_{i-1}' - w_i') \\
&\geq_2 -\|w_{i-1}' - w_i'\|^2 - \eta^2 \cdot G^2 + \\
&\quad 2\eta(f_i(w_i) - f_i(w_i')) - \\
&\quad 2\eta\|E_i\| \cdot \|w_i - w_i'\| - \\
&\quad 2\|w_i - w_{i-1}'\| \cdot \|w_{i-1}' - w_i'\| \\
&\geq_3 -U \cdot \|w_{i-1}' - w_i'\| - \eta^2 \cdot G^2 + \\
&\quad 2\eta(f_i(w_i) - f_i(w_i')) - \\
&\quad 2\eta\|E_i\| \cdot U - 2 \cdot U \cdot \|w_{i-1}' - w_i'\|
\end{aligned}$$

The three inequalities are justified as follows:

- In ineq. 1 used $\|\pi_C(a) - b\| \leq \|a - b\|$ for all $b \in C$ and all a, since C is convex.
- In ineq. 2 we used $f_i'(w_i)^\top (w_i - w_i') \geq f_i(w_i) - f_i(w_i')$ since f_i is convex and $f_i'(w_i)$ is a subgradient; we also applied Cauchy-Swartz twice.

- In ineq. 3 we used the fact that all the points are in C and that its diameter is U.

We can finish the proof by noticing that $\sum_{i=1}^{T} D_i \leq U^2$; rearranging and dividing by 2η, we get the stated bound.

3. Bounding the regret

In this section we will explain the main idea of the unified analysis.

Consider both the Forgetron and the RBP when the active set of w already contains B vectors, and suppose that they make an error on vector x. Then both algorithms will remove a vector from the active set, and add x to the active set with an appropriate value for its weight α. In this section, we will show that both algorithms follow the gradient of an L_2-regularized hinge loss with corrupted gradients.

Adding a vector to the active set according to the Perceptron's learning rule is equivalent to adding the gradient of the hinge loss to w on input x on which the Perceptron errs. This means that if (x, y) is a datapoint and $p = \text{sign}\,\langle w, x\rangle$ is an incorrect prediction ($p \neq y$), then $L'(w; (x, y)) = -(y - p)/2 \cdot x$.

Removing a vector from the active set can be seen as a form of L_2-regularization for both algorithms, where the gradients are corrupted in different ways. Consider first the Forgetron. Recall that it scales down all the weights α's by a constant on each error, which *is* L_2-regularization with some weight and some learning rate. However, in addition to the scaling, the Forgetron also removes the vector with the smallest weight. If the weight of the removed vector is small, then the Forgetron can be seen as doing L_2-regularization with a small amount of error (where the error removes the oldest point), so we can apply lemma 1.

Next, consider the RBP, which removes a random vector from its active set. Let

$$\begin{aligned} w &= \sum_{i=1}^{B} \phi(x_i)\alpha_i \\ I &\sim \text{Unif}(1, \ldots, B) \\ s(w) &= \phi(x_I)\alpha_I \end{aligned}$$

Removing a random vector from the active set is equivalent to replacing w with $w - s(w)$. However, notice that $\mathbb{E}[s(w)] = w/B$; at the same time, replacing w with $w - w/B$ is what we get from following the gradient of an L_2-regularizer. Hence, replacing w with $w - s(w)$ is no different from performing L_2-regularization where the gradient is corrupted with a certain type of zero-mean noise that depends on w.[2]

[2]We defined $s(w)$ when w is represented as the sum of B terms. If w is represented as the sum of less than B terms, then we will add zero terms to make sure that w is represented as the sum of B terms. For example, if $w = \phi(x_1) + \phi(x_2)$, then $s(w) = 0$ with probability $1 - 2/B$.

Algorithm 1 The modified Randomized Budget Perceptron.

Input: data $\{(x_1, y_1), \ldots, (x_T, y_T)\}$, budget B, the convex domain C, the learning rate η.
Initialize $w_1 \leftarrow 0$.
for $i = 1$ **to** T **do**
  Set $p_i \leftarrow \text{sign}(w_i^\top \phi(x_i))$ (the prediction)
  if $p_i = y_i$; i.e., if w_i predicts x_i correctly **then**
    Set $w_{i+1} \leftarrow w_i$
  else
    Set $w_{i+1} \leftarrow \pi_C(w_i + \eta \cdot (y_i - p_i)/2 \cdot \phi(x_i) - s(w_i))$
  end if
end for

In what follows, we will cast the Forgetron and the RBP as algorithms that perform online convex optimization with an L_2-regularized hinge loss, where the Forgetron has errors in its gradients and the RBP is given noisy gradients. We will run the online convex optimization only on the datapoints on which the Perceptron makes a mistake, as in (Shalev-Shwartz, 2007, ch. 2), which is straightforward in the Forgetron's case. However, it is less straightforward in the case of the RBP, because the set of vectors on which the Perceptron makes a mistake is random; nonetheless, we will show that the idea of lemma 2 of (Flaxman et al., 2005) still applies in the next subsection.

Both proofs have analogous structure; they differ in their choice of the parameters U (the diameter of the set of vectors we compete against), λ (the weight decay), η (the learning rate), and the manner in which the gradient is corrupted.

Finally, we will see that our algorithms use large learning rates; however, because of that, we will get that both Perceptrons are able to track a changing hypothesis (a direct consequence of lemma 1)–which was known for the RBP but not for the Forgetron.

We will assume that the vectors $\phi(x_i)$ are distinct and that $\|\phi(x)\| \leq 1$ for all x, which implies the following fact.

Fact 1:

If we run Algorithms 1 and 2, then $w_t = \sum_{j=1}^{B} \phi(x_j)\alpha_j$ *where* $|\alpha_j| \leq \eta$ *for all* j.

Thus, the weight of every vector never exceeds the learning rate, which follows from the form of the gradient of the hinge loss and from the fact that all the vectors are distinct.

3.1. The Randomized Budget Perceptron

We will now cast the RBP as an instance of online convex programming. The modified RBP (Algorithm 1) differs from the original RBP in order to be more compatible with lemma 1 in two ways. First, it prevents the weight vector from being too large (using the projection π_C); and second, it will occasionally trim the active set even when its size is smaller than B, because $s(w)$ is subtracted even when the active set is smaller than B. However, when the active set is small, then $s(w)$ is likely to be zero.

Here is the setup. We are given B, the budget; we will determine λ, η, and U as the proof goes on. We are also given an arbitrary sequence of points $(x_1, y_1), \ldots, (x_T, y_T)$ that satisfy $\|\phi(x_i)\| \leq 1$ for every i.

First, the convex domain C is $\{w : \|w\| \leq U/2\}$, so the diameter of C is U.

Second, the functions to be minimized are $f_i(w) = L(w; (\phi(x_i), y_i)) + \lambda \cdot \|w\|^2/2$. We will write $L_i(w)$ for $L(w; (\phi(x_i), y_i))$.

Consider Algorithm 1. Once we choose λ, the choice of the learning rate η is constrained. Indeed, we wish to replace updates of the form

$$\Delta_i = -\eta \cdot f_i'(w_i) = -\eta \cdot L_i'(w) - \lambda\eta \cdot w$$

with the random update

$$\Delta_i' = -\eta \cdot L_i'(w) - s(w)$$

In order for Δ_i' to satisfy $\mathbb{E}[\Delta_i'] = \Delta_i$, we must have

$$1/B = \lambda\eta \tag{7}$$

since $\mathbb{E}[s(w)] = w/B$.

We will now show how to apply an idea of (Flaxman et al., 2005) to our setting in a way that will consider only a subset of the training points.

Consider a sequence of noise variables $n_1, \ldots, n_T$ that are defined as the noise in our gradients. Specifically, as we run the modified RBP on the a-priori known sequence of training points,[3] we will have noise in our gradients whenever we make an error. If the algorithm makes a mistake on example x_i, then let $n_i = -\Delta_i'/\eta + \Delta_i/\eta$ be the noise; that is, n_i is exactly the amount of noise we add to the gradients $f_i'(w_i)$ in order to obtain noisy gradient $-\Delta_i'/\eta$ (note that $-\Delta_i/\eta = f_i'(w_i)$). If the algorithm makes no error, let $n_i = 0$. As a result, n_i satisfies $\mathbb{E}[n_i|n_{<i}] = 0$, and hence $\mathbb{E}[n_i] = \mathbb{E}_{n_{<i}}[\mathbb{E}[n_i|n_{<i}]] = 0$.

Let M be the number of errors an and let $m(1), \ldots, m(M)$ be the indicies where the modified RBP made its errors. The variables M, $m(j)$ and w_i are random.

[3] It is known only for the purposes of the analysis, but not to the algorithm.

For each i, define the function $h_i(w) = f_i(w) + (w - w_i)^\top n_i$. Then $h_i(w_i) = f_i(w_i)$ and $\mathbb{E}[h_i(w)] = f_i(w)$ for all w; the functions h_i are convex and $h_i'(w) = f_i'(w) + n_i$. Running the modified RBP is equivalent to running online gradient descent with projections π_C on the functions $h_{m(j)}$ that correspond to the examples on which the RBP made an error, so lemma 1 applies to $h_{m(j)}$ for $1 \leq j \leq M$ with the vectors $w_{m(j)}$.

Using the fact that the hinge loss ≥ 1 whenever the algorithm makes an error, as well as applying lemma 1, we get

$$\begin{aligned} M &\leq \sum_{j=1}^{M} f_{m(j)}(w_{m(j)}) \\ &= \sum_{j=1}^{M} h_{m(j)}(w_{m(j)}) \\ &\leq \sum_{j=1}^{M} h_{m(j)}(w'_{m(j)}) + \frac{U^2}{2\eta} + \frac{3 \cdot US_M}{2\eta} + \\ &\quad \frac{M \cdot \eta \cdot G^2}{2} + U \cdot E \end{aligned}$$

where the second inequality is due lemma 1, S_M is the shift of the vectors $\{w'_{m(j)}\}_{j=1}^{M}$ and G is greater than $\|h'_{m(j)}(w_{m(j)})\|$ for all j. But we know that $h_i(w) = L_i(w) + \lambda \cdot \|w\|^2/2 + (w - w_i)^\top n_i$, that $\|w\|^2 \leq U^2/4$, and that $E = 0$. Using this knowledge, we can upper bound the number of errors by

$$\begin{aligned} M &\leq \sum_{i=1}^{T} L_i(w_i') + \sum_{j=1}^{M} (w'_{m(j)} - w_{m(j)})^\top n_{m(j)} + \\ &\quad M \cdot \frac{\lambda \cdot U^2}{8} + \frac{U^2}{2\eta} + \frac{3 \cdot US}{2\eta} + \frac{M \cdot \eta \cdot G^2}{2} \end{aligned}$$

where we added the hinge losses of the comparison sequence at all of the points (which are always nonnegative). Here S is the shift of all the comparison vectors w_i'; we used $S_M \leq S$, which holds by the triangle inequality. Finally, by averaging the inequality, we get

$$\begin{aligned} \mathbb{E}[M] &\leq \sum_{i=1}^{T} L_i(w_i') + \mathbb{E}[M] \cdot \frac{\lambda \cdot U^2}{8} + \\ &\quad \frac{U^2}{2\eta} + \frac{3 \cdot US}{2\eta} + \frac{\mathbb{E}[M] \cdot \eta \cdot G^2}{2} \end{aligned} \tag{8}$$

which is a deterministic bound on the average number of errors. The $n_{m(i)}$ terms disappear since w_i is determined by $n_{<i}$ and $\mathbb{E}[n_i|n_{<i}] = 0$. This can be made clearer by defining an indicator variable Err_i which is 1 if the algorithm errs on example i and 0 otherwise. Using Err_i, we get

$$\sum_{j=1}^{M} (w'_{m(j)} - w_{m(j)})^\top n_{m(j)} = \sum_{i=1}^{T} (w_i' - w_i)^\top n_i \cdot Err_i$$

Now, consider $E[(w'_i - w_i)^\top n_i \cdot Err_i | n_{<i}]$. Given $n_{<i}$, w_i is fixed and Err_i is known, and $E[n_i | n_{<i}] = 0$; this shows that each term in the above sum is zero in expectation.[4]

This bound is meaningful only when multiplicative factor of $\mathbb{E}[M]$ on the RHS of eq. 8 is less than 1, which is $\lambda \cdot U^2/8 + G^2\eta/2$. This constrains the choice of the parameters.

Before we find the parameters, we observe that $G^2 \leq 4$, which can be seen by noticing that the gradient $h'_i(w_i)$ has the form $\pm\phi(x_j) \pm c\phi(x_k)$ for some j and k and some $c \leq 1$, which follows from fact 1; since $\phi(x) \leq 1$, the norm $\|h'_i(w_i)\| \leq 2$, and hence $G^2 \leq 4$.

Finding a set of reasonable parameters is straightforward. By recalling that $1/(\lambda B) = \eta$ and choosing $\lambda = 1/U^2, B = 16U^2$, $\eta = 1/16$ makes sure that the multiplicative factor of $\mathbb{E}[M]$ on the RHS of eq. 8 is $1/4$, yielding the following:

Theorem 1: *For all sequences of points satisfying $\|\phi(x_i)\| \leq 1$, and for all sequences of comparison vectors w'_i satisfying $\|w'_i\| \leq \sqrt{B}/8$, the expected number of mistakes Algorithm 1 makes is bounded by*

$$\frac{3}{4}\mathbb{E}[M] \leq \sum_{i=1}^{T} L_i(w'_i) + \frac{B}{2} + 6 \cdot S \cdot \sqrt{B} \qquad (9)$$

where $S = \sum_{i=2}^{T} \|w'_i - w'_{i-1}\|$, and Algorithm 1 uses $\eta = 1/16$, the domain $C = \{w : \|w\| \leq \sqrt{B}/8\}$, and the budget B.

As in (Cavallanti et al., 2007), it is possible to select other constants that can increase the multiplicative constant of $\mathbb{E}[M]$ (which is $3/4$) closer to 1 at the expense of being competitive with vectors of smaller norms.

3.2. The simplified Forgetron

We perform an analogous analysis of the simplified Forgetron (Algorithm 2), which turns out to be simpler than that of Algorithm 1 because lemma 1 applies directly.

Let the datapoints (x_i, y_i) and the functions f_i be exactly as in the previous section: $f_i(w) = L_i(w) + \lambda\|w\|^2/2$, where $L_i(w) = L(w; (\phi(x_i), y_i))$.

We will run online convex programming on the functions f_i with errors in the gradient, where the errors make sure that the update

$$w_{i+1} \leftarrow \pi_C(w_i - \eta \cdot L'_i(w_i) - \lambda\eta \cdot w_i - \eta \cdot E_i) \qquad (10)$$

removes the oldest member of the active set, when the active set has B members. This is arranged by setting

[4] We cannot simply use $\mathbb{E}[n_i] = 0$ because w_i is also random, so there might be correlations causing $\mathbb{E}[n_i^\top w_i]$ to be nonzero.

Algorithm 2 The simplified Forgetron.

Input: data $\{(x_1, y_1), \ldots, (x_T, y_T)\}$, budget B, the convex domain C, the learning rate η, and the weight decay λ.
Initialize $w_1 \leftarrow 0$.
for $i = 1$ **to** T **do**
  Set $p_i \leftarrow \text{sign}(w_i^\top \phi(x_i))$ (the prediction)
  if $p_i = y_i$; i.e., if w_i predicts x_i correctly **then**
    Set $w_{i+1} \leftarrow w_i$
  else
    Set $w_{i+1} \leftarrow w_i$
    If w_{i+1}'s active set is equal to B, then remove the vector with the smallest weight from w_{i+1}'s active set.
    Set $w_{i+1} \leftarrow w_{i+1} + \eta \cdot (y_i - p_i)/2 \cdot \phi(x_i) - \lambda\eta \cdot w_{i+1}$
    Set $w_{i+1} \leftarrow \pi_C(w_{i+1})$
  end if
end for

$E_i = c_i\phi(x_{\text{old}(i)})$ where c_i/η is the weight of the oldest example in w_i multiplied by $(1 - \lambda\eta)$. The need for this multiplication is from eq. 10, which first scales the oldest vector in the active set by $1 - \lambda\eta$ before removing it with $-\eta \cdot E_i$.

We know that the weight of each example is reduced by a factor of $(1 - \lambda\eta)$ on each error, and if an example is removed from the active set then its weight was removed B times, so $|c_i| \leq (1 - \lambda\eta)^B$. Furthermore, since $\|\phi(x_{\text{old}(i)})\| \leq 1$, we have $\|E_i\| \leq c_i$. Note that when the active set is sufficiently small, we do not need to remove the oldest vector, in which case $\|E_i\| = 0 \leq (1 - \lambda\eta)^B$

As in the previous section, we can apply lemma 1 on the errors of the simplified Forgetron; this is considerably more straightforward than for the RBP since the algorithm is deterministic. At once, lemma 1 gives the bound

$$\begin{aligned}
M &\leq \sum_{j=1}^{M} f_{m(j)}(w_{m(j)}) \\
&\leq \sum_{j=1}^{M} f_{m(j)}(w'_{m(j)}) + \frac{U^2}{2\eta} + \frac{3 \cdot US_M}{2\eta} + \\
&\quad \frac{M \cdot \eta \cdot G^2}{2} + U \cdot E \\
&\leq \sum_{j=1}^{M} L_{m(j)}(w'_{m(j)}) + M \cdot \frac{\lambda \cdot U^2}{8} + \frac{U^2}{2\eta} + \\
&\quad \frac{3 \cdot US_M}{2\eta} + \frac{M \cdot \eta \cdot G^2}{2} + U \cdot M \cdot (1 - \lambda\eta)^B \\
&\leq \sum_{i=1}^{T} L_i(w'_i) + M \cdot \frac{\lambda \cdot U^2}{8} + \frac{U^2}{2\eta} + \frac{3 \cdot US}{2\eta} +
\end{aligned}$$

$$\frac{M \cdot \eta \cdot G^2}{2} + U \cdot M \cdot (1 - \lambda\eta)^B$$

where S_M is the shift of the vectors $w'_{m(j)}$. We used $E = \sum_{j=1}^{M} E_{m(j)} \leq M \cdot (1 - \lambda\eta)^B$ and that $f_i(w'_i) = L_i(w'_i) + \lambda \cdot \|w'_i\|^2/2 \leq L_i(w'_i) + \lambda U^2/8$ in the third inequality, and the non-negativity of the hinge loss in the last inequality. We also used the fact that $S_M \leq S$ as in the previous section.

Before we choose the parameters to make sure that the multiplicative factor of M on the RHS is less than 1, we need to bound G. We know that the gradient has the form of $\pm\phi(x_i) - \lambda w - E_i$, whose norm can be bounded with the triangle inequality by $2 + U\lambda$ since $\|\phi(x_i)\|$ and $\|E_i\|$ are both ≤ 1, so $G^2 \leq (2 + U\lambda)^2$. If we choose $\lambda = 1/U^2$ and ensure that $U \geq 1$, we will get that $G^2 \leq 9$.

As before, we need to choose the parameters to make sure that the multiplicative factor of M on the RHS is small. It will be sufficient for the parameters to satisfy

$$\frac{\lambda \cdot U^2}{8} + \frac{\eta \cdot G^2}{2} + U \cdot (1 - \lambda\eta)^B \leq \frac{1}{2} \qquad (11)$$

We begin by choosing $\lambda = 1/U^2$, which causes the first term to be $1/8$. By choosing $\eta = 1/32$ and using $G^2 \leq 9 \leq 16$, we get that the second term $\leq 1/4$. Finally, we need to choose B so that $U(1 - 1/(32U^2))^B \leq 1/8$. Since we enforce $U \geq 1$, we have $(1 - 1/(32U^2))^{32U^2} \leq 1/2$. So the third term $\leq 1/8$ if $B \geq 32U^2 \log_2(8U)$, which is satisfied if

$$U^2 = \frac{B}{32 \log_2 8B} \qquad (12)$$

Note that if $B \geq 500$, then $U \geq 1$.

Combining the above, we get the following:

Theorem 2: *Given a budget $B \geq 500$, let $\eta = 1/32$, U be as above, and $\lambda = 1/U^2$. Consider Algorithm 2 with $C = \{w : \|w\| \leq U/2\}$. If we run Algorithm 2 on an arbitrary sequence of points satisfying $\|\phi(x_i)\| \leq 1$, then the number of errors made by the simplified Forgetron is less than*

$$\frac{1}{2}M \leq \sum_{i=1}^{T} L_i(w'_i) + 16 \cdot U^2 + 48 \cdot U \cdot S \qquad (13)$$

where w'_i is any sequence of comparison vectors satisfying $\|w'_i\| \leq U/2$, and S is the shift of w'_i.

As the original Forgetron regret bound, this bound does not apply against comparison vectors whose norm is of order $\Theta(\sqrt{B})$.

4. Conclusions and Discussions

In this paper we bounded the regret of two budget Perceptron algorithms using the observation that both algorithms can be seen as performing a type of L_2-regularization.

We can analyze the original RBP (recall that it differs from the modified RBP in that it does not remove any vector from the active set if there are fewer than B of them) by making sure that first B cost functions are the hinge loss without L_2-regularization. While reasonably straightforward, this analysis is slightly more cumbersome to describe. It is also straightforward to keep track of $\|w\|^2 = \sum_{1 \leq i,j \leq B} = \alpha_i \alpha_j k(x_i, x_j)$ efficiently, which is needed to implement the projection π_C. It may even be possible to remove the need for the projection π_C altogether using the lazy-projection variant of Zinkevich's theorem (Zinkevich, 2004).

Finally, the assumption that the vectors are all distinct is not strong since we can easily introduce tiny perturbations which will not change the algorithm to a noticeable extent yet will make sure that no two vectors are equal.

Acknowledgements

We thank anonymous reviewers for useful comments and suggestions. This research was partially supported by the Ontario Graduate Scholarship.

References

Bottou, L., & Bousquet, O. (2008). The tradeoffs of large scale learning. In J. Platt, D. Koller, Y. Singer and S. Roweis (Eds.), *Advances in neural information processing systems*, vol. 20, 161–168. NIPS Foundation (http://books.nips.cc).

Cavallanti, G., Cesa-Bianchi, N., & Gentile, C. (2007). Tracking the Best Hyperplane with a Simple Budget Perceptron. *Machine Learning*, *69(2/3)*, 143–167.

Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, Learning, and Games*. Cambridge University Press.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online Passive-Aggressive Algorithms. *The Journal of Machine Learning Research*, *7*, 551–585.

Crammer, K., Kandola, J., & Singer, Y. (2004). Online classification on a budget. In S. Thrun, L. Saul and B. Schölkopf (Eds.), *Advances in neural information processing systems 16*, 225–232. Cambridge, MA: MIT Press.

Dekel, O., Shalev-Shwartz, S., & Singer, Y. (2008). The Forgetron: A Kernel-Based Perceptron on a Budget. *SIAM Journal on Computing*, *37(5)*, 1342–1372.

Flaxman, A. D., Kalai, A. T., & Mcmahan, B. H. (2005). Online convex optimization in the bandit setting: gradient descent without a gradient. *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 385–394). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Littlestone, N., & Warmuth, M. (1989). The weighted majority algorithm. *30th Annual Symposium on the Foundations of Computer Science.*, 256–261.

Novikoff, A. B. (1963). On convergence proofs for perceptrons. *Proceedings of the Symposium on the Mathematical Theory of Automata*, *12*, 615–622.

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

Scholkopf, B., & Smola, A. (2002). *Learning with kernels*. MIT Press Cambridge, Mass.

Shalev-Shwartz, S. (2007). *Online Learning: Theory, Algorithms, and Applications*. Doctoral dissertation, The Hebrew University of Jerusalem.

Shalev-Shwartz, S., & Singer, Y. (2005). A new perspective on an old perceptron algorithm. *Proceedings of the Sixteenth Annual Conference on Computational Learning Theory* (pp. 264–278).

Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. *The 24th international conference on Machine learning*, 807–814.

Shalev-Shwartz, S., & Srebro, N. (2008). SVM optimization: inverse dependence on training set size. *Proceedings of the 25th international conference on Machine learning* (pp. 928–935).

Weston, J., Bordes, A., & Bottou, L. (2005). Online (and offline) on an even tighter budget. 413–420. Society for Artificial Intelligence and Statistics. (Available electronically at http://www.gatsby.ucl.ac.uk/aistats/).

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. *Proceedings of the International Conference on Machine Learning* (pp. 928–936).

Zinkevich, M. (2004). *Theoretical Guarantees for Algorithms in Multi-Agent Settings*. Doctoral dissertation, Carnegie Mellon University.