Minimizing the Maximal Loss: How and Why

Shai Shalev-Shwartz SHAIS@CS.HUJI.AC.IL
School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel.

Yonatan Wexler YONATAN.WEXLER@ORCAM.COM
Orcam

Abstract

A commonly used learning rule is to approximately minimize the *average* loss over the training set. Other learning algorithms, such as AdaBoost and hard-SVM, aim at minimizing the *maximal* loss over the training set. The average loss is more popular, particularly in deep learning, due to three main reasons. First, it can be conveniently minimized using online algorithms, that process few examples at each iteration. Second, it is often argued that there is no sense to minimize the loss on the training set too much, as it will not be reflected in the generalization loss. Last, the maximal loss is not robust to outliers. In this paper we describe and analyze an algorithm that can convert any online algorithm to a minimizer of the maximal loss. We prove that in some situations better accuracy on the training set is crucial to obtain good performance on unseen examples. Last, we propose robust versions of the approach that can handle outliers.

1. Introduction

In a typical supervised learning scenario, we have training examples, $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$, and our goal is to learn a function $h : \mathcal{X} \to \mathcal{Y}$. We focus on the case in which h is parameterized by a vector $w \in \mathcal{W} \subset \mathbb{R}^d$, and we use h_w to denote the function induced by w. The performance of w on an example (x, y) is assessed using a loss function, $\ell : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \to [0, 1]$. A commonly used learning rule is to approximately minimize the average loss, namely,

$$\min_{w \in \mathcal{W}} L_{\text{avg}}(w) := \frac{1}{m} \sum_{i=1}^{m} \ell(w, x_i, y_i) \; . \tag{1}$$

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

Another option is to approximately minimize the maximal loss, namely,

$$\min_{w \in \mathcal{W}} L_{\max}(w) := \max_{i \in [m]} \ell(w, x_i, y_i) \; . \tag{2}$$

Obviously, if there exists $w^* \in \mathcal{W}$ such that $\ell(w^*, x_i, y_i) = 0$ for every i then the minimizers of both problems coincide. However, approximate solutions can be very different. In particular, since $L_{\max}(w) \geq L_{\text{avg}}(w)$ for every w, the guarantee $L_{\max}(w) < \epsilon$ is stronger than the guarantee $L_{\text{avg}}(w) < \epsilon$. Furthermore, for binary classification with the zero-one loss, any vector for which $L_{\max}(w) < 1$ must predict all the labels on the training set correctly, while the guarantee $L_{\text{avg}}(w) < 1$ is meaningless.

Some classical machine learning algorithms can be viewed as approximately minimizing $L_{\max}$. For example, Hard-SVM effectively solves $L_{\max}$ with respect to the loss function $\ell(w, x_i, y_i) = \lambda \|w\|^2 + 1[y_i \langle w, x_i \rangle < 1]$. However, minimizing L_{avg} is a more popular approach, especially for deep learning problems, in which w is the vector of weights of a neural network and the optimization is performed using variants of stochastic gradient descent (SGD). There are several reasons to prefer L_{avg} over $L_{\max}$:

1. If m is very large, it is not practical to perform operations on the entire training set. Instead, we prefer iterative algorithms that update w based on few examples at each iteration. This can be easily done for L_{avg} by observing that if we sample i uniformly at random from $[m]$, then the gradient of $\ell(w, x_i, y_i)$ with respect to w is an unbiased estimator of the gradient of $L_{\text{avg}}(w)$. This property, which lies at the heart of the SGD algorithm, does not hold for $L_{\max}$.

2. Our ultimate goal is not to minimize the loss on the training set but instead to have a small loss on unseen examples. As argued before, approximately minimizing $L_{\max}$ can lead to a smaller loss on the training set, but it is not clear if this added accuracy will also be reflected in performance on unseen examples. Formal

arguments of this nature were given in (Bousquet & Bottou, 2008; Shalev-Shwartz & Srebro, 2008).

3. The objective $L_{\max}$ is not robust to outliers. It is easy to see that even a single outlier can make the minimizer of $L_{\max}$ meaningless.

In this paper we tackle the aforementioned disadvantages of $L_{\max}$, and by doing so, we show cases in which $L_{\max}$ is preferable. In particular:

1. We describe and analyze a meta algorithm that can take any online learner for w and convert it to a minimizer of $L_{\max}$. A detailed description of our meta algorithm, its analysis, and a comparison to other approaches, are given in Section 2.

2. The arguments in (Bousquet & Bottou, 2008; Shalev-Shwartz & Srebro, 2008) rely on a comparison of upper bounds. We show that these upper bounds are not tight in many cases. Furthermore, we analyze the sample complexity of learning in situations where the training examples are divided to "typical" scenarios and "rare" scenarios. We argue that in many practical cases, our goal is to have a high accuracy on both typical and rare examples. We show conditions under which minimizing even few rare examples suffice to guarantee good performance on unseen examples from the rare scenario. In other words, few examples can have a dramatic effect on the performance of the learnt classifier on unseen examples. This is described and analyzed in Section 3.

3. Finally, in Section 4 we review standard techniques for generalizing the results from realizable cases to scenarios in which there might be outliers in the data.

To summarize, we argue that in some situations minimizing $L_{\max}$ is better than minimizing L_{avg}. We address the "how" question in Section 2, the "why" question in Section 3, and the issue of robustness in Section 4. Finally, in Section 5 we provide some empirical evidence, showing the effectiveness of our algorithm on real world learning problems.

2. How

In this section we describe and analyze an algorithmic framework for approximately solving the optimization problem given in (2).

Denote by $\mathcal{S}_m = \{p \in [0,1]^m : \|p\|_1 = 1\}$ the probabilistic simplex over m items. We also denote by $\Lambda : \mathcal{W} \to [0,1]^m$ the function defined by

$$\Lambda(w) = (\ell(w, x_1, y_1), \ldots, \ell(w, x_m, y_m)) \ .$$

The first step is to note that the optimization problem given in (2) is equivalent to

$$\min_{w \in \mathcal{W}} \max_{p \in \mathcal{S}_m} \langle p, \Lambda(w) \rangle \ . \tag{3}$$

This is true because for every w, the p that maximizes the inner optimization is the all zeros vector except 1 in the coordinate for which $\ell(w, x_i, y_i)$ is maximal.

We can now think of (3) as a zero-sum game between two-players. The p player tries to maximize $\langle p, \Lambda(w) \rangle$ while the w player tries to minimize $\langle p, \Lambda(w) \rangle$. The optimization process is comprised of T game rounds. At round t, the p player defines $p_t \in \mathcal{S}_m$ and the w player defines $w_t \in \mathcal{W}$. We then sample $i_t \sim p_t$ and define the value of the round to be $\ell(w_t, x_{i_t}, y_{i_t})$.

To derive a concrete algorithm we need to specify how player p picks p_t and how player w picks w_t. For the w player one can use any online learning algorithm. We specify the requirement from the algorithm below.

Definition 1 (Mistake bound for the w player) *We say that the w player enjoys a mistake bound of C if for every sequence of indices $(i_1, \ldots, i_T) \in [m]^T$ we have that*

$$\sum_{t=1}^{T} \ell(w_t, x_{i_t}, y_{i_t}) \ \leq \ C \ . \tag{4}$$

Example 1 *Consider a binary classification problem in which the data is linearly separable by a vector w^* with a margin of 1. Let the loss function be the zero-one loss, namely, $\ell(w, x, y) = 1[y\langle w, x \rangle \leq 0]$, where $1[\text{boolean expression}]$ is 1 if the boolean expression holds and 0 otherwise. We can use the online Perceptron algorithm as our w learner and it is well known that the Perceptron enjoys the mistake bound of $C = \|w^*\|^2 \max_{i \in [m]} \|x_i\|^2$ (for a reference, see for example (Shalev-Shwartz, 2011)).*

For the p player, we use the seminal work of (Auer et al., 2002). In particular, recall that the goal of the p player is to maximize the loss, $\ell(w_t, x_{i_t}, y_{i_t}$, where $i_t \sim p_t$. The basic idea of the construction is therefore to think of the m examples as m slot machines, where at round t the gain of pulling the arms of the different machines is according to $\Lambda(w_t) \in [0,1]^m$. Crucially, the work of (Auer et al., 2002) does not assume that $\Lambda(w_t)$ are sampled from a fixed distribution, but rather the vectors $\Lambda(w_t)$ can be chosen by an adversary. As observed in Auer et al. (2002, Section 9), this naturally fits zero-sum games, as we consider here.

In (Auer et al., 2002) it is proposed to rely on the algorithm EXP3.P.1 as the strategy for the p-player. The acronym EXP3 stands for **Exp**loration-**Exp**loitation-**Exp**onent, because the algorithm balances between exploration and ex-

ploitation and rely on an exponentiated gradient framework. The "P" in EXP3.P.1 stands for a regret bound that holds with high probability. This is essential for our analysis because we will later apply a union bound over the m examples. While the EXP3.P.1 algorithm gives the desired regret analysis, the runtime per iteration of this algorithm scales with m. Here, we propose another variant of EXP3 for which the runtime per iteration is $O(\log(m))$.

To describe our strategy for the p player, recall that it maintains $p_t \in \mathcal{S}_m$. We will instead maintain another vector, $q_t \in \mathcal{S}_m$, and will set p_t to be the vector such that $p_{t,i} = \frac{1}{2}q_{t,i} + \frac{1}{2m}$. That is, p_t is a half-half mix of q_t with the uniform distribution. While in general such a strong mix with the uniform distribution can hurt the regret, in our case it only affects the convergence rate by a constant factor. On the up side, this strong exploration helps us having an update step that takes $O(\log(m))$ per iteration.

Recall that at round t of the algorithm, we sample $i_t \sim p_t$ and the value of the round is $\ell(w_t, x_{i_t}, y_{i_t})$. Denote $z_t = -\frac{\ell_{i_t}(w_t)}{p_{i_t}} e_{i_t}$, then it is easy to verify that $\mathbb{E}_{i_t \sim p_t}[z_t] = -\Lambda(w_t)$. Therefore, applying gradient descent with respect to the linear function $\langle \cdot, z_t \rangle$ is in expectation equivalent to applying gradient descent with respect to the linear function $-\langle \cdot, \Lambda(w_t) \rangle$, which is the function the p player aims at minimizing. Instead of gradient descent, we use the exponentiated gradient descent approach which applies gradient descent in the log space, namely, the update can be written as $\log(q_{t+1}) = \log(q_t) + \eta z_t$.

A pseudo-code of the resulting algorithm is given in Section 2.3. Observe that we use a tree structure to hold the vector q, and since all but the i_t coordinate of z_t are zeros, we can implement the update of q in $O(\log(m))$ time per iteration. The following theorem summarizes the convergence of the resulting algorithm.

Theorem 1 *Suppose the we have an oracle access to an online algorithm that enjoys a mistake bound of C with respect to the training examples $(x_1, y_1), \ldots, (x_m, y_m)$. Fix ϵ, δ, and suppose we run the FOL algorithm with T, k such that $C/T \leq \epsilon/8$, $T = \Omega(m \log(m/\delta)/\epsilon)$, and $k = \Omega(\log(m/\delta)/\epsilon)$, and with $\eta = 1/(2m)$. Then, with probability of at least $1 - \delta$,*

$$\max_i \frac{1}{k} \sum_{j=1}^{k} \ell(w_{t_j}, x_i, y_i) \;\leq\; \epsilon \,.$$

The proof of the theorem is given in Appendix **??**.

The above theorem tells us that we can find an ensemble of $O(\log(m)/\epsilon)$ predictors, such that the ensemble loss is smaller than ϵ for all of the examples.

We next need to show that we can construct a single predictor with a small loss. To do so, we consider two typical scenarios. The first is classification settings, in which $\ell(w, x, y)$ is the zero-one loss and the second is convex losses in which $\ell(w, x, y)$ has the form $\phi_y(h_w(x))$, where for every y, ϕ_y is a convex function.

2.1. Classification

In classification, $\ell(w, x, y)$ is the zero-one loss, namely, it equals to zero if $h_w(x) = y$ and it equals to 1 if $h_w(x) \neq y$. We will take ϵ to be any number strictly smaller than $1/2$, say 0.499.

Observe that Theorem 1 tells us that the average loss of the classifiers $w_{t_1}, \ldots, w_{t_k}$ is smaller than $\epsilon = 0.499$. Since the values of the loss are either 1 or 0, it means that the loss of more than $1/2$ of the classifiers is 0, which implies that the majority classifier has a zero loss.

Corollary 1 *Assume that $\ell(w, x, y)$ is the zero-one loss function, namely, $\ell(w, x, y) = 1[h_w(x) \neq y]$. Apply Theorem 1 with $\epsilon = 0.49$. Then, with probability of at least $1 - \delta$, the majority classifier of $h_{w_{t_1}}, \ldots, h_{w_{t_k}}$ is consistent, namely, it makes no mistakes on the entire training set.*

Example 2 *Consider again the linear binary classification problem given in Example 1, where we use the online Perceptron algorithm as our w learner, and its mistake bound is C as given in Example 1. Then, after $\tilde{O}(m + C)$ iterations, we will find an ensemble of $O(\log(m))$ halfspaces, whose majority vote is consistent with all the examples. In Section 2.4 we compare the runtime of the method to state-of-the-art approaches. Here we just note that to obtain a consistent hypothesis using SGD one needs order of $m\,C$ iterations, which is significantly larger in most scenarios.*

2.2. Convex Losses

Consider now the case in which $\ell(w, x, y)$ has the form $\phi_y(h_w(x))$, where for every y, ϕ_y is a convex function. Note that this assumption alone does not imply that ℓ is a convex function of w (this will be true only if $h_w(x)$ is an affine function).

In the case of convex ϕ_y, combining Theorem 1 with Jensen's inequality we obtain:

Corollary 2 *Under the assumptions of Theorem 1, if $\ell(w, x, y)$ has the form $\phi_y(h_w(x))$, where for every y, ϕ_y is a convex function, then the predictor $h(x) = \frac{1}{k}\sum_{j=1}^{k} h_{w_{t_j}}(x)$ satisfies $\forall i, \quad \phi_{y_i}(h(x_i)) \leq \epsilon$. If we further assume that $h_w(x)$ is an affine function of w, and let $w = \frac{1}{k}\sum_{j=1}^{k} w_{t_j}$, then we also have that*

$$\forall i, \quad \phi_{y_i}(h_w(x_i)) \;\leq\; \epsilon \,.$$

2.3. Pseudo-code

Below we describe a pseudo-code of the algorithm. We rely on a tree data structure for maintaining the probability of the p-player. It is easy to verify the correctness of the implementation. Observe that the runtime of each iteration is the time required to perform one step of the online learner plus $O(\log(m))$ for sampling from p_t and updating the tree structure.

`Focused Online Learning (FOL)`

Input:
Training examples $(x_1, y_1), \ldots, (x_m, y_m)$
Loss function $\ell : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \to [0, 1]$
Parameters η, T, k
Oracle access to online learning algorithm OLA
Initialization:
Tree.initialize(m) (see the Tree pseudo-code)
$w_1 =$ OLA.initialize()
Loop over $t \in \{1, \ldots, T\}$**:**
$(i_t, p_{i_t}) =$ Tree.sample($1/2$)
OLA.step(x_{i_t}, y_{i_t})
Tree.update($i_t, \exp(\eta\, \ell(w_t, x_{i_t}, y_{i_t})/p_{i_t})$)
Output:
Sample $(t_1, \ldots, t_k)$ indices uniformly from $[T]$
Output Majority/Average of $(h_{w_{t_1}}, \ldots, h_{w_{t_k}})$

`Tree`

initialize(m)
Build a full binary tree of height $h = \lceil \log_2(m) \rceil$
Set value of the first m leaves to 1 and the rest to 0
Set the value of each internal node to be
the sum of its two children
Let q_i be the value of the i'th leaf divided by
the value of the root
sample(γ)
Sample $b \in \{0, 1\}$ s.t. $\mathbb{P}[b = 0] = \gamma$
If $b = 0$
Sample i uniformly at random from $[m]$
Else
Set v to be the root node of the tree
While v is not a leaf:
Go to the left/right child by sampling
according to their values
Let i be the obtained leaf
Return: $(i, \gamma/m + (1 - \gamma)q_i)$
update(i, f)
Let v be the current value of the i'th leaf of the tree
Let $\delta = f\, v - v$
Add δ to the values of all nodes on
the path from the i'th leaf to the root

2.4. Related Work

As mentioned before, our algorithm is a variant of the approach given in Auer et al. (2002, Section 9), but has the advantage that the update of the p player at each iteration scales with $\log(m)$ rather than with m. Phrasing the max-loss minimization as a two players game has also been proposed by (Clarkson et al., 2012; Hazan et al., 2011). These works focus on the specific case of binary classification with a linear predictor, namely, they tackle the problem $\min_{w \in \mathbb{R}^d : \|w\|_2 \leq 1} \max_{p \in \mathcal{S}_m} \sum_i p_i \langle w, x_i \rangle$. Assuming the setup of Example 1, (Clarkson et al., 2012) presents an algorithm that finds a consistent hypothesis in runtime of $\tilde{O}((m + d) \cdot C)$. For the same problem, our algorithm (with the Perceptron as the weak learner) finds a consistent hypothesis in runtime of $\tilde{O}((m + C) \cdot d)$. Furthermore, if the instances are $\bar{d}$-sparse (meaning that the number of non-zeros in each x_i is at most $\bar{d}$), then the term d in our bound can be replaced by $\bar{d}$. In any case, our bound is sometimes better and sometimes worse than the one in (Clarkson et al., 2012). We note that we can also use AdaBoost (Freund & Schapire, 1995) on top of the Perceptron algorithm for the same problem. It can be easily verified that the resulting runtime will be identical to our bound. In this sense, our algorithm can be seen as an online version of AdaBoost.

Finally, several recent works use sampling strategies for speeding up optimization algorithms for minimizing the average loss. See for example (Bengio & Senécal, 2008; Bouchard et al., 2015; Zhao & Zhang, 2014; Allen-Zhu & Yuan, 2015).

3. Why

In this section we tackle the "Why" question, namely, why should we prefer minimizing the maximal loss instead of the average loss. For simplicity of presentation, throughout this section we deal with binary classification problems with the zero-one loss, in the realizable setting. In this context, minimizing the maximal loss to accuracy of $\epsilon < 1$ leads to a consistent hypothesis[1]. On the other hand, minimizing the average loss to any accuracy of $\epsilon > 1/m$ does not guarantee to return a consistent hypothesis. Therefore, in this context, the "why" question becomes: why should we find a consistent hypothesis and not be satisfied with a hypothesis with $L_{\text{avg}}(h) \leq \epsilon$ for some $\epsilon > 1/m$.

In the usual PAC learning model (see (Shalev-Shwartz & Ben-David, 2014) for an overview), there is a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ and the training examples are assumed to be

[1] Recall that a consistent hypothesis is a hypothesis that makes no mistakes on the training set. We also use the term Empirical Risk Minimization (ERM) to describe the process of finding a consistent hypothesis, and use ERM(S) to denote any hypothesis which is consistent with a sample S.

sampled i.i.d. from $\mathcal{D}$. The goal of the learner is to minimize $L_{\mathcal{D}}(h) := \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(h,x,y)] = \mathbb{P}_{(x,y)\sim\mathcal{D}}[h(x) \neq y]$. For a fixed $h \in \mathcal{H}$, the random variable $L_{\text{avg}}(h)$ is an unbiased estimator of $L_{\mathcal{D}}(h)$. Furthermore, it can be shown (Boucheron et al. (2005, Section 5.1.2)) that with probability of at least $1-\delta$ over the choice of the sample $S \sim \mathcal{D}^m$ we have that:

$$\forall h \in \mathcal{H},\ \ L_{\mathcal{D}}(h) \leq L_{\text{avg}}(h) + \tilde{O}\left(\sqrt{L_{\text{avg}}(h)\frac{\text{VC}(\mathcal{H}) - \log(\delta)}{m}} + \frac{\text{VC}(\mathcal{H}) - \log(\delta)}{m}\right)$$

where $\text{VC}(\mathcal{H})$ is the VC dimension of the class $\mathcal{H}$ and the notation $\tilde{O}$ hides constants and logarithmic terms.

From the above bound we get that any h with $L_{\text{avg}}(h) = 0$ (i.e., a consistent h) guarantees that $L_{\mathcal{D}}(h) = \tilde{O}\left(\frac{\text{VC}(\mathcal{H})+\log(1/\delta)}{m}\right)$. However, we will obtain the same guarantee (up to constants) if we will choose any h with $L_{\text{avg}}(h) \leq \epsilon$, for $\epsilon = \tilde{O}\left(\frac{\text{VC}(\mathcal{H})+\log(1/\delta)}{m}\right)$. Based on this observation, it can be argued that it is enough to minimize L_{avg} to accuracy of $\epsilon = \tilde{O}\left(\frac{\text{VC}(\mathcal{H})+\log(1/\delta)}{m}\right) > \frac{1}{m}$, because a better accuracy on the training set will in any case get lost by the sampling noise.

Furthermore, because of either computational reasons or high dimensionality of the data, we often do not directly minimize the zero-one loss, and instead minimize a convex surrogate loss, such as the hinge-loss. In such cases, we often rely on a margin based analysis, which means that the term $\text{VC}(\mathcal{H})$ is replaced by B^2, where B is an upper bound on the norm of the weight vector that defines the classifier. It is often the case that the convergence rate of SGD is of the same order, and therefore there is no added value of solving the ERM problem over performing a single SGD pass over the data (or few epochs over the data). Formal arguments of this nature were given in (Bousquet & Bottou, 2008; Shalev-Shwartz & Srebro, 2008).

Despite of these arguments, we show below reasons to prefer the max loss formulation over the average loss formulation. The first reason is straightforward: arguments that are based on worst case bounds are problematic, since in many cases the behavior is rather different than the worst case bounds. In subsection 3.1 we present a simple example in which there is a large gap between the sample complexity of SGD and the sample complexity of ERM, and we further show that the runtime of our algorithm will be much better than the runtime of SGD for solving this problem.

Next, we describe a family of problems in which the distribution from which the training data is being sampled is a mix of "typical" examples and "rare" examples. We show that in such a case, few "rare" examples may be sufficient for learning a hypothesis that has a high accuracy on both the "typical" and "rare" examples, and therefore, it is really required to solve the ERM problem as opposed to being satisfied with a hypothesis for which $L_{\text{avg}}(h)$ is small.

3.1. A Simple Example of a Gap

Consider the following distribution. Let $z_1 = (\alpha, 1)$ and $z_2 = (\alpha, -2\alpha)$ for some small $\alpha > 0$. To generate an example $(x, y) \sim \mathcal{D}$, we first sample a label y uniformly at random from $\{\pm 1\}$, then we set $x = yz_1$ with probability $1-\epsilon$ and set $x = yz_2$ with probability ϵ. The hypothesis class is halfspaces: $\mathcal{H} = \{x \to \text{sign}(\langle w, x\rangle) : w \in \mathbb{R}^2\}$.

The following three lemmas, whose proofs are given in the appendix, establish the gap between the different approaches.

Lemma 1 *For every $\delta \in (0,1)$, if $m \geq \frac{2\log(4/\delta)}{\epsilon}$ then, with probability of at least $1-\delta$ over the choice of the training set, $S \sim \mathcal{D}^m$, any hypothesis in ERM(S) has a generalization error of 0.*

Lemma 2 *Suppose we run SGD with the hinge-loss and any $\eta > 0$ for less than $T = \Omega(1/(\alpha\epsilon))$ iterations. Then, with probability of $1 - O(\epsilon)$ we have that SGD will not find a solution with error smaller than ϵ.*

Lemma 3 *Running FOL (with the Perceptron as its w player) takes $\tilde{O}\left(\frac{1}{\epsilon} + \frac{1}{\alpha}\right)$ iterations.*

3.2. Typical vs. Rare Distributions

To motivate the learning setting, consider the problem of face detection, in which the goal is to take an image crop and determine whether it is an image of a face or not. An illustration of typical random positive and negative examples is given in Figure 1 (top row). By having enough training examples, we can learn that the discrimination between face and non-face is based on few features like "an ellipse shape", "eyes", "nose", and "mouth". However, from the typical examples it is hard to tell whether an image of a watermelon is a face or not — it has the ellipse shape like a face, and something that looks like eyes, but it doesn't have a nose, or a mouth. The bottom row of Figure 1 shows some additional "rare" examples.

Such a phenomenon can be formally described as follows. There are two distributions over the examples, $\mathcal{D}_1$ and $\mathcal{D}_2$. Our goal is to have an error of at most ϵ on **both** distributions, namely, we would like to find h such that $L_{\mathcal{D}_1}(h) \leq \epsilon$ and $L_{\mathcal{D}_2}(h) \leq \epsilon$. However, the training examples that we observe are sampled i.i.d. from a mixed distribution, $\mathcal{D} = \lambda_1\mathcal{D}_1 + \lambda_2\mathcal{D}_2$, where $\lambda_1, \lambda_2 \in (0,1)$ and $\lambda_1 + \lambda_2 = 1$. We assume that $\lambda_2 \ll \lambda_1$, namely, typical examples in the training set are from $\mathcal{D}_1$ while examples from $\mathcal{D}_2$ are rare.

Fix some ϵ. If $\lambda_2 < \epsilon$, then a hypothesis with $L_{\text{avg}}(h) \leq \epsilon$

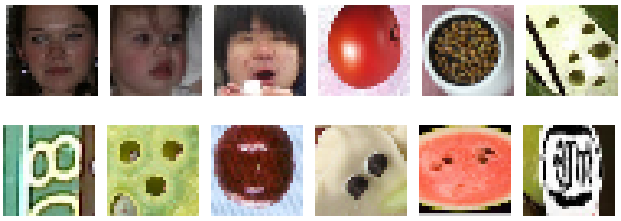

Figure 1. Top: typical positive (left) and negative (right) examples. Bottom: rare negative examples.

might err on most of the "rare" examples, and is therefore likely to have $L_{\mathcal{D}_2}(h) > \epsilon$. If we want to guarantee a good performance on $\mathcal{D}_2$ we must optimize to a very high accuracy, or put another way, we would like to minimize $L_{\max}$ instead of L_{avg}. The question is how many examples do we need in order to guarantee that a consistent hypothesis on S will have a small error on both $\mathcal{D}_1$ and $\mathcal{D}_2$. A naive approach is to require order of $\text{VC}(\mathcal{H})/(\lambda_2\epsilon)$ examples, thus ensuring that we have order of $\text{VC}(\mathcal{H})/\epsilon$ examples from both $\mathcal{D}_1$ and $\mathcal{D}_2$. However, this is a rough estimate and the real sample complexity might be much smaller. Intuitively, we can think of the typical examples from $\mathcal{D}_1$ as filtering out most of the hypotheses in $\mathcal{H}$, and the goal of the rare examples is just to fine tune the exact hypothesis. In the example of face detection, the examples from $\mathcal{D}_1$ will help us figure out what is an "ellipse like shape", what is an "eye", and what is a "mouth" and a "nose". After we understand all this, the rare examples from $\mathcal{D}_2$ will tell us the exact requirement of being a face (e.g., you need an ellipse like shape and either eyes or a mouth). We can therefore hope that the number of required "rare" examples is much smaller than the number of required "typical" examples. This intuition is formalized in the following theorem.

Theorem 2 *Fix $\epsilon, \delta \in (0,1)$, distributions D_1, D_2, and let $D = \lambda_1 D_1 + \lambda_2 D_2$ where $\lambda_1 + \lambda_2 = 1, \lambda_1, \lambda_2 \in [0,1]$, and $\lambda_2 < \lambda_1$. Define $\mathcal{H}_{1,\epsilon} = \{h \in \mathcal{H} : L_{D_1}(h) \leq \epsilon\}$ and $c = \max\{c' \in [\epsilon, 1) : \forall h \in \mathcal{H}_{1,\epsilon}, L_{D_2}(h) \leq c' \Rightarrow L_{D_2}(h) \leq \epsilon\}$. Then, if*

$$m \geq \Omega\left(\frac{\text{VC}(\mathcal{H})\log(1/\epsilon) + \log(1/\delta)}{\epsilon} + \frac{\text{VC}(\mathcal{H}_{1,\epsilon})\log(1/c) + \log(1/\delta)}{c\,\lambda_2}\right)$$

we have, with probability of at least $1-\delta$ over the sampling of a sample $S \sim D^m$:

$$L_{D_1}(\text{ERM}(S)) \leq \epsilon \quad \textit{and} \quad L_{D_2}(\text{ERM}(S)) \leq \epsilon$$

The proof of the theorem is given in the appendix. The first term in the sample complexity is a standard VC-based sample complexity. The second term makes two crucial improvement. First, we measure the VC dimension of a reduced class ($\mathcal{H}_{1,\epsilon}$), containing only those hypotheses in $\mathcal{H}$ that have a small error on the "typical" distribution. Intuitively, this will be a much smaller hypothesis class compared to the original class. Second, we apply an analysis of the sample complexity similar to the "shell analysis" of (Haussler et al., 1996), and assume that the error of all hypotheses in $\mathcal{H}_{1,\epsilon}$ on $\mathcal{D}_2$ is either smaller than ϵ or larger than c, where we would like to think of c as being significantly larger than ϵ. Naturally, this will not always be the case. But, Theorem 2 provides data dependent conditions, under which a much smaller number of examples from $\mathcal{D}_2$ is sufficient. As a motivation, consider again Figure 1, and suppose $H_{1,\epsilon}$ contains conjunctions over all subsets of the features "has eyes", "has nose", "has mouth", "has skin color". Let h^* be the conjunction of all these 4 features. It is reasonable to assume that examples in $\mathcal{D}_2$ lack one of these features. Let us also assume for simplicity that each lacking feature takes at least $1/8$ of the mass of $\mathcal{D}_2$. Hence, the error of all "wrong" functions in $\mathcal{H}_{1,\epsilon}$ on $\mathcal{D}_2$ is at least $1/8$, while the error of h^* is 0. We see that in this simple example, $c = 1/8$.

All in all, the theorem shows that a small number of "rare" examples in the training set can have a dramatic effect on the performance of the algorithm on the rare distribution $\mathcal{D}_2$. But, we will see this effect only if we will indeed find a hypothesis consistent with all (or most) examples from $\mathcal{D}_2$, which requires an algorithm for minimizing $L_{\max}$ and not L_{avg}.

4. Robustness

In the previous section we have shown cases in which minimizing $L_{\max}$ is better than minimizing L_{avg}. However, in the presence of outliers, minimizing $L_{\max}$ might lead to meaningless results — even a single outlier can change the value of $L_{\max}$ and might lead to a trivial, non-interesting, solution. In this section we describe two tricks for addressing this problem. The first trick replaces the original sample with a new sample whose examples are sampled from the original sample. The second trick relies on slack variables. We note that these tricks are not new and appears in the literature in various forms. See for example (Huber & Ronchetti, 2009; Maronna et al., 2006). The goal of this section is merely to show how to apply known tricks to the max loss problem.

Recall that in the previous section we have shown that a small amount of "rare" examples can have a dramatic effect on the performance of the algorithm on the "rare" distribution. Naturally, if the number of outliers is larger than the number of rare examples we cannot hope to enjoy the benefit of rare examples. Therefore, throughout this section we assume that the number of outliers, denoted k, is smaller

than the number of "rare" examples, which we denote by m_2.

4.1. Sub-sampling with repetitions

The first trick we consider is to simply take a new sample of n examples, where each example in the new sample is sampled independently according to the uniform distribution over the original m examples. Then, we run our algorithm on the obtained sample of n examples.

Intuitively, if there are k outliers, and the size of the new sample is significantly smaller than m/k, then there is a good chance that no outliers will fall into the new sample. On the other hand, we want that enough "rare" examples will fall into the new sample. The following theorem, whose proof is in the appendix, shows for which values of k and m_2 this is possible.

Theorem 3 *Let k be the number of outliers, m_2 be the number of rare examples, m be the size of the original sample, and n be the size of the new sample. Assume that $m \geq 10k$. Then, the probability that the new sample contains outliers and/or does not contain at least $m_2/2$ rare examples is at most* $0.01 + 0.99kn/m + e^{-0.1\,nm_2/m}$.

For example, if $n = m/(100k)$ and $m_2 \geq 1000 \log(100)\, k$, then the probability of the bad event is at most 0.03.

4.2. Slack variables

Another common trick, often used in the SVM literature, is to introduce a vector of slack variables, $\xi \in \mathbb{R}^m$, such that $\xi_i > 0$ indicates that example i is an outlier. We first describe the ideal version of outlier removal. Suppose we restrict ξ_i to take values in $\{0, 1\}$, and we restrict the number of outliers to be at most K. Then, we can write the following optimization problem:

$$\min_{w \in \mathcal{W}, \xi \in \mathbb{R}^m} \max_{i \in [m]} (1 - \xi_i)\, \ell(w, x_i, y_i) \;\text{ s.t. } \xi \in \{0,1\}^m,\ \|\xi\|_1 \leq K\ .$$

This optimization problem minimizes the max loss over a subset of examples of size at least $m - K$. That is, we allow the algorithm to refer to at most K examples as outliers.

Note that the above problem can be written as a max-loss minimization:

$$\min_{\bar{w} \in \bar{\mathcal{W}}} \max_i \bar{\ell}(\bar{w}, x_i, y_i) \quad \text{where}$$
$$\bar{\mathcal{W}} = \{(w, \xi) : w \in \mathcal{W}, \xi \in \{0,1\}^m, \|\xi\|_1 \leq K\} \;\text{ and}$$
$$\bar{\ell}((w, \xi), x_i, y_i) = (1 - \xi_i)\ell(w, x_i, y_i)$$

We can now apply our framework on this modified problem. The p player remains as before, but now the $\bar{w}$ player has a more difficult task. To make the task easier we can perform several relaxations. First, we can replace the non-convex constraint $\xi \in \{0,1\}^m$ with the convex constraint $\xi \in [0,1]^m$. Second, we can replace the multiplicative slack with an additive slack, and re-define: $\bar{\ell}((w, \xi), x_i, y_i) = \ell(w, x_i, y_i) - \xi_i$. This adds a convex term to the loss function, and therefore, if the original loss was convex we end up with a convex loss. The new problem can often be solved by combining gradient updates with projections of ξ onto the set $\xi \in [0,1]^m, \|\xi\|_1 \leq K$. For efficient implementations of this projection see for example (Duchi et al., 2008). We can further replace the constraint $\|\xi\|_1 \leq K$ with a constraint of $\|\xi\|_2^2 \leq K$, because projection onto the Euclidean ball is a simple scaling, and the operation can be done efficiently with an adequate data structure (as described, for example, in (Shalev-Shwartz et al., 2011)).

5. Experiments

In this section we demonstrate several merits of our approach on the well studied problem of face detection. Detection problems in general have a biased distribution as they are often expected to detect few needles in a haystack. Furthermore, a mix of typical and rare distributions is to be expected. For example, users of smartphones won't be in the same continent as the manufacturers who collect data for training. This domain requires weighting of examples, and therefore is a good playground to examine our algorithm.

To create a dataset we downloaded $30k$ photos from Google images that are tagged with "face". We then applied an off-the-shelf face detector, and it found $32k$ rectangles that aligned on faces. This was the base of our positive examples. For negative examples we randomly sampled $250k$ rectangles in the same images that do not overlap faces. Each rectangle was cropped and scaled to 28×28 pixels. Using a fixed size simplifies the experiments so we can focus on the merits of our method rather than justify various choices that are not relevant here, such as localization and range of scale.

Recall that our FOL algorithm relies on an online algorithm as the w player. In the experiments, w is the vector containing all the weights of a convolutional neural network (CNN) with a variant of the well known LeNet architecture. The layers of the network are as follows. Convolution with a 5×5 kernel, stride of 1, and 40 output channels, followed by ReLU and max-pooling. Then a convolution with a 5×5 kernel, stride of 1, and 80 output channels, followed by ReLU and max-pooling. Then a convolution with a 7×7 kernel, stride of 1, and 160 output channels, followed by ReLU. Finally, a linear prediction over the resulting 160 channels yields the binary prediction for the input 28×28

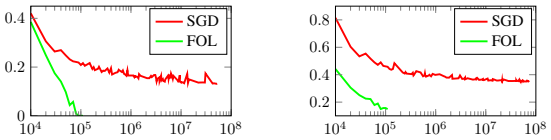

Figure 2. Comparing the error percentage of FOL vs. SGD as a function of the number of iterations. Left: Train Error. Right: Test Error.

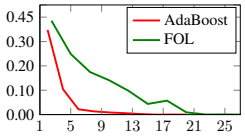

Figure 3. Train error of FOL vs. AdaBoost as a function of the number of epochs.

image crop. Overall the model has $710,642$ weights. We denote by $\mathcal{H}$ the resulting hypothesis class.

In the comparison, we focus on the case in which the data is realizable by $\mathcal{H}$. To guarantee that, we first used vanilla SGD to find a network in $\mathcal{H}$. We then kept only samples that were labeled correctly by the network. This yielded $28k$ positive examples and $246k$ negative examples. This set was then randomly mixed and split $9:1$ for train and test sets.

For the w player we used the SGD algorithm with Nesterov's momentum, as this is a standard solver for learning convolutional neural networks. The parameters we used are a batch size of 64, an ℓ_2 regularization of 0.0005, momentum parameter of 0.9, and a learning rate of $\eta_t = 0.01(1+0.0001\,t)^{-0.75}$. We used the logistic loss as a surrogate loss for the classification error.

We performed two experiments to highlight different properties of our algorithm. The first experiment shows that FOL is much faster than SGD, and this is reflected both in train and test errors. The second experiment compares FOL to the application of AdaBoost on top of the same base learner.

Experiment 1: Convergence speed In this experiment we show that FOL is faster than SGD. Figure 2 shows the train and test errors of SGD and FOL. Both models were initialized with the same randomly selected weights. As mentioned before, FOL relies on SGD as its w player. Observe that FOL essentially solves the problem (zero training error) after 30 epochs, whereas SGD did not converge to a zero training error even after $14,000$ epochs and achieved 0.1313% error. While the logarithmic scale in the figure shows that SGD is still improving, it is doing so at a decreasing rate. This is reflected by our theoretical analysis.

To understand why SGD slows down, observe that when the error of SGD is as small as here (0.13%), only one example in 769 is informative. Even when at classification error of 0.4% (left-side of the graph), only 4 in $1,000$ examples are informative. Hence, even with batch size of 64, SGD picks one useful sample only once every fifteen iterations. FOL expects an average of 32 useful samples in every iteration so every iteration is informative. In our case, since the training set size is $246k$, only 984 examples are informative and FOL makes sure to focus on these rather than waste time on solved examples.

As can be seen in Figure 2, the faster convergence of FOL on the training set is also translated to a better test error. Indeed, FOL achieves a test error of 0.14% (after 27 epochs) whereas even after $14k$ epochs SGD results 0.35% error.

Experiment 2: Comparison to AdaBoost As mentioned in Section 2.4, FOL can be seen as an online version of AdaBoost. Specifically, we can apply the AdaBoost algorithm while using SGD as its weak learner. For concreteness and reproducibility of our experiment, we briefly describe the resulting algorithm. We initialize a uniform distribution over the m examples, $p = (1/m, \ldots, 1/m)$. At iteration t of AdaBoost, we run one epoch of SGD over the data, while sampling examples according to p. Let h_t be the resulting classifier. We then calculate $h_t(x_i)$ over all the m examples, calculate the averaged zero-one error of h_t, with weights based on p, define a weight $\alpha_t = 0.5\log(1/\epsilon_t - 1)$, and update p such that $p_i \propto p_i \exp(-\alpha_t y_i h_t(x_i))$. We repeat this process for T iterations, and output the hypothesis $h(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x))$.

Observe that each such iteration of AdaBoost is equivalent to 2 epochs of our algorithm. In Figure 3 we show the train error of AdaBoost and FOL as a function of the number of epochs over the data. The behavior on the test set shows a similar trend. As can be seen in the figure, AdaBoost finds a consistent hypothesis after 20 epochs, while FOL requires 27 epochs to converge to a consistent hypothesis. However, once FOL converged, its last hypothesis has a zero training error. In contrast, the output hypothesis of AdaBoost is a weighted majority of T hypotheses ($T = 10$ in this case). It follows that at prediction time, applying AdaBoost's predictor is 10 times slower than applying FOL's predictor. Often, we prefer to spend more time during training, for the sake of finding a hypothesis which can be evaluated faster at test time. While based on our theory, the output hypothesis of FOL should also be a majority of $\log(m)$ hypotheses, we found out that in practice, the last hypothesis of FOL converges to a zero classification error at almost the same rate as the majority classifier.

Acknowledgements: S. Shalev-Shwartz is supported by ICRI-CI and by the European Research Council (Theo-

ryDL project).

References

Allen-Zhu, Zeyuan and Yuan, Yang. Even faster accelerated coordinate descent using non-uniform sampling. *arXiv preprint arXiv:1512.09103*, 2015.

Auer, Peter, Cesa-Bianchi, Nicolo, Freund, Yoav, and Schapire, Robert E. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

Bengio, Yoshua and Senécal, Jean-Sébastien. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *Neural Networks, IEEE Transactions on*, 19(4):713–722, 2008.

Bouchard, Guillaume, Trouillon, Théo, Perez, Julien, and Gaidon, Adrien. Accelerating stochastic gradient descent via online learning to sample. *arXiv preprint arXiv:1506.09016*, 2015.

Boucheron, Stéphane, Bousquet, Olivier, and Lugosi, Gábor. Theory of classification: A survey of some recent advances. *ESAIM: probability and statistics*, 9:323–375, 2005.

Bousquet, Olivier and Bottou, Léon. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pp. 161–168, 2008.

Clarkson, Kenneth L, Hazan, Elad, and Woodruff, David P. Sublinear optimization for machine learning. *Journal of the ACM (JACM)*, 59(5):23, 2012.

Duchi, John, Shalev-Shwartz, Shai, Singer, Yoram, and Chandra, Tushar. Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pp. 272–279. ACM, 2008.

Fan, Xiequan, Grama, Ion, and Liu, Quansheng. Hoeffding's inequality for supermartingales. *Stochastic Processes and their Applications*, 122(10):3545–3559, 2012.

Freund, Yoav and Schapire, Robert E. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pp. 23–37. Springer, 1995.

Haussler, David, Kearns, Michael, Seung, H Sebastian, and Tishby, Naftali. Rigorous learning curve bounds from statistical mechanics. *Machine Learning*, 25(2-3):195–236, 1996.

Hazan, Elad, Koren, Tomer, and Srebro, Nati. Beating sgd: Learning svms in sublinear time. In *Advances in Neural Information Processing Systems*, pp. 1233–1241, 2011.

Huber, Peter J. and Ronchetti, Elvezio M. *Robust Statistics (second edition)*. J. Wiley, 2009.

Kivinen, J. and Warmuth, M. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–64, January 1997.

Maronna, Ricardo A, Martin, R Douglas, and Yohai, Victor J. *Robust Statistics: Theory and Methods*. J. Wiley, 2006.

Shalev-Shwartz, Shai. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2011.

Shalev-Shwartz, Shai and Ben-David, Shai. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge university press, 2014.

Shalev-Shwartz, Shai and Singer, Yoram. On the equivalence of weak learnability and linear separability: New relaxations and efficient boosting algorithms. *Machine learning*, 80(2-3):141–163, 2010.

Shalev-Shwartz, Shai and Srebro, Nathan. Svm optimization: inverse dependence on training set size. In *Proceedings of the 25th international conference on Machine learning*, pp. 928–935. ACM, 2008.

Shalev-Shwartz, Shai, Singer, Yoram, Srebro, Nathan, and Cotter, Andrew. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

Zhao, Peilin and Zhang, Tong. Stochastic optimization with importance sampling. *arXiv preprint arXiv:1401.2753*, 2014.