Better Parameter-free Stochastic Optimization with ODE Updates for Coin-Betting

Keyi Chen[1] **John Langford**[2] **Francesco Orabona**[1]

[1] Boston University
[2] Microsoft Research
keyichen@bu.edu, jcl@microsoft.com, francesco@orabona.com

Abstract

Parameter-free stochastic gradient descent (PFSGD) algorithms do not require setting learning rates while achieving optimal theoretical performance. In practical applications, however, there remains an empirical gap between tuned stochastic gradient descent (SGD) and PFSGD. In this paper, we close the empirical gap with a new parameter-free algorithm based on continuous-time Coin-Betting on truncated models. The new update is derived through the solution of an Ordinary Differential Equation (ODE) and solved in a closed form. We show empirically that this new parameter-free algorithm outperforms algorithms with the "best default" learning rates and almost matches the performance of finely tuned baselines without anything to tune.

1 Introduction

Most machine learning algorithms require solving an optimization problem, $\min_{\boldsymbol{x}\in\mathbb{R}^d} F(\boldsymbol{x})$. To solve this problem, first-order stochastic optimization algorithms are the de-facto choice for machine learning due to their speed across large datasets and simplicity. These Stochastic (sub)Gradient Descent (SGD) algorithms start from an initial solution $\boldsymbol{x}_1$, iteratively update a vector $\boldsymbol{x}_t$ moving in the negative direction of a stochastic (sub)gradient $\boldsymbol{g}_t$ such that $\mathbb{E}[\boldsymbol{g}_t] \in \partial F(\boldsymbol{x}_t)$: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \boldsymbol{g}_t$, where $\eta_t > 0$ is the *learning rate* or *step size*. Learning rates are the big caveat of SGD.

How do we set the learning rate? Intuitively, the learning rates must become arbitrarily small to converge to the minimum of the function. This is clear considering minimizing the function $F(x) = |x - 10|$ with SGD. In addition, the step size must be large enough that not too many updates are required to move from the initial to the optimal solution.

We can formalize the above intuitions with the standard convergence rate of SGD with constant step size η after T iterations with stochastic subgradients $\boldsymbol{g}_t$ bounded by 1 in L_2 norm (Zinkevich 2003):

$$\mathbb{E}\left[F\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}_t\right)\right] - F(\boldsymbol{x}^\star) \leq \frac{\|\boldsymbol{x}_1 - \boldsymbol{x}^\star\|^2}{2\eta T} + \frac{\eta T}{2}\,. \quad (1)$$

From the above, we have that the optimal worst-case step size is $\eta = \frac{\|\boldsymbol{x}_1 - \boldsymbol{x}^\star\|_2}{\sqrt{T}}$ implying the optimal step size is inversely proportional to the square root of the number of iterations and proportional to the distance between the initial point and the optimal one $\boldsymbol{x}^\star$. Unfortunately, we do not know in advance the distance from the initial point to the optimal solution nor we can expect to reliably estimate it—again, consider the function $F(x) = |x - 10|$. This lack of information about $\|\boldsymbol{x}_1 - \boldsymbol{x}^\star\|_2$ is the primary difficulty of choosing the learning rate in the stochastic setting.

From a practical point of view, this failure of the theory to provide a way to automatically set the learning rates means that most of the time the best way to achieve the best convergence rate is to treat the learning rate as a hyperparameter and exhaustively search for the best one. However, the computational cost of this search can be huge, basically multiplying the entire learning process by the number of different learning rates we have to try.

However, a new class of *parameter-free* algorithms has been recently proposed (e.g. McMahan and Orabona 2014; Orabona 2014; Cutkosky and Boahen 2016; Orabona and Pal 2016; Cutkosky and Boahen 2017; Foster, Rakhlin, and Sridharan 2018; Cutkosky and Orabona 2018; Kotłowski 2019; Kempka, Kotłowski, and Warmuth 2019; Cutkosky and Sarlos 2019). These algorithms do not have a learning rate parameter at all, while achieving essentially the same theoretical convergence rate you would have obtained tuning the learning rate in (1). The simplest parameter-free algorithm (Orabona and Pal 2016) has an update rule of

$$\boldsymbol{x}_{t+1} = \frac{-\sum_{i=1}^{t}\boldsymbol{g}_i}{L(1+t)}\left(1 - \sum_{i=1}^{t}\langle \boldsymbol{g}_i, \boldsymbol{x}_i\rangle\right)\,, \quad (2)$$

where L is the Lipschitz constant of F. These algorithms basically promise to trade-off a bit of accuracy for the removal of tuning the learning rate. However, empirically they still have a big gap with tuned optimization algorithms.

Contributions. In this paper, we greatly reduce this gap with a new class of parameter-free stochastic optimization algorithms that performs better than SGD, AdaGrad, and Adam with the "best default" parameter, see Figure 1. We achieve it by designing a parameter-free algorithm that is more aware of the geometry of the loss functions by avoiding overshooting their minima. In particular, we modify the optimization algorithm based on coin-betting in (2) to make an *infinite number of infinitesimally small updates* on a *truncated linear model for optimization* (Asi and Duchi 2019).

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

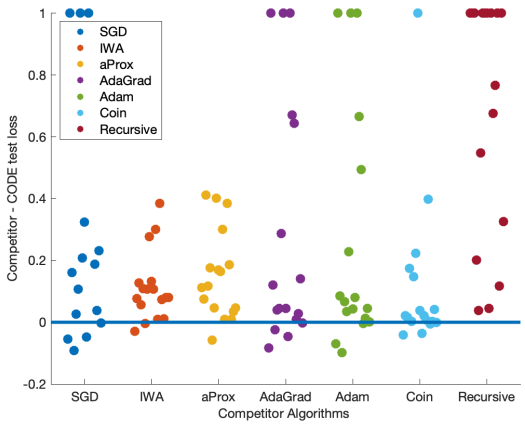

Figure 1: Difference between the normalized competitor and CODE test losses on 17 regression datasets using "best default" parameters. Each point represents a dataset. Points located above the line $y = 0$ represent datasets on which CODE outperforms competitors.

The final update is a closed formula solution of an Ordinary Differential Equation (ODE), hence we call our algorithm CODE: Coin-betting ODE.

Related Work. Parameter-free learning algorithms are discussed in Section 2. The main inspiration here is the Importance Weight Aware updates (Karampatziakis and Langford 2011) which make infinitely many infinitesimal SGD updates for each sampled loss. We provide two improvements: making the approach work for a parameter-free update rule and generalizing the set of losses. In particular, we consider any convex Lipschitz function instead of an expectation of losses of the form satisfying $\ell_t(\boldsymbol{x}_t) = \ell(\langle \boldsymbol{z}_t, \boldsymbol{x}_t \rangle, y_t)$. Achieving closed-form solutions requires using a truncated linear model (Asi and Duchi 2019). The Importance Weight Aware updates (Karampatziakis and Langford 2011) are also close to the ones in the truncated model from Asi and Duchi (2019), perfectly coinciding in some cases. Both these approaches are also similar to Passive-Aggressive online learning algorithms (Crammer et al. 2006), implicit updates (Kivinen and Warmuth 1997; Kulis and Bartlett 2010), and the proximal point method (Rockafellar 1976).

There is a long history relating ODE to optimization. The corresponding ODEs of a numerical optimization algorithm are established by taking infinitesimal step sizes so that the trajectory converges to a curve modeled by the ODE. The continuous trajectory provides new insights into numerical optimization such as the continuous-time interpretation of Nesterov's method (Su, Boyd, and Candes 2015) and the accelerated Mirror Descent inspired by continuous-time analysis (Krichene, Bayen, and Bartlett 2015). This paper is the first attempt to combine a parameter-free algorithm with an ODE approach. The technique is general and we believe it points out an interesting new direction for research.

Organization of the paper. In Section 2, after introducing some definitions, we briefly review the theory behind parameter-free optimization algorithms through the simplified lens of coin-betting algorithms. Then, in Section 3 we introduce our algorithm CODE. Finally, in Section 4 we present empirical results and we conclude with a discussion and future work in Section 5.

2 Preliminaries

In this section, we introduce the needed mathematical background and the basic idea of parameter-free coin-betting optimization algorithms and truncated linear models.

Notation. We denote vectors by bold letters and matrices by capital letters, e.g. $\boldsymbol{x} \in \mathbb{R}^d$ and $A \in \mathbb{R}^{d \times m}$. We denote by subscript t a variable that changes in a discrete way, while using a function of t for a variable that changes over time in a continuous way, e.g., $\boldsymbol{x}_t$ and $\boldsymbol{x}(t)$. $\mathbf{1}[E]$ denotes the indicator function of an event E, i.e., $\mathbf{1}[E]$ is equal to 1 if E is true and 0 otherwise.

Convex Analysis. We denote by $\|\cdot\|$ the L_2 norm in $\mathbb{R}^d$. Let $f : \mathbb{R}^d \to \mathbb{R} \cup \{\pm\infty\}$, the *Fenchel conjugate* of f is $f^\star : \mathbb{R}^d \to \mathbb{R} \cup \{\pm\infty\}$ with $f^\star(\boldsymbol{\theta}) = \sup_{\boldsymbol{x} \in \mathbb{R}^d} \boldsymbol{\theta}^\top \boldsymbol{x} - f(\boldsymbol{x})$. A vector $\boldsymbol{x}$ is a *subgradient* of a convex function f at $\boldsymbol{v}$ if $f(\boldsymbol{v}) - f(\boldsymbol{u}) \leq (\boldsymbol{v} - \boldsymbol{u})^\top \boldsymbol{x}$ for any $\boldsymbol{u} \in \operatorname{dom} f$. The *differential set* of f at $\boldsymbol{v}$, denoted by $\partial f(\boldsymbol{v})$, is the set of all the subgradients of f at $\boldsymbol{v}$. If f is also differentiable at $\boldsymbol{v}$, then $\partial f(\boldsymbol{v})$ contains a single vector, $\nabla f(\boldsymbol{v})$, which is the *gradient* of f at $\boldsymbol{v}$.

Betting on a coin. We describe here how to reduce subgradient descent to betting on the outcome of a binary event (i.e. a coin flip). This is not an exhaustive review of the topic—interested readers are referred to, e.g., Orabona and Pal (2016) and Orabona (2019, Chapter 9).

We consider a gambler making repeated bets on the outcomes of adversarial coin flips. The gambler starts with \$1. In each round t, he bets on the outcome of a coin flip $c_t \in \{-1, 1\}$, where $+1$ denotes heads and -1 denotes tails. We do not make any assumption on how c_t is generated.

The gambler can bet any amount, although no additional money may be borrowed. We encode the gambler's bet in round t by $x_t \in \mathbb{R}$, where $sign(x_t)$ encodes whether the bet is on heads or tails and $|x_t|$ encodes the betted amount. When the bet succeeds, the gambler wins $x_t c_t$, otherwise, $x_t c_t$ is lost. We define Wealth_t as the gambler's wealth at the end of round t, that is

$$\text{Wealth}_t = 1 + \sum_{i=1}^{t} x_i c_i \ . \tag{3}$$

We enforce $x_t = \beta_t \text{Wealth}_{t-1}$ for some betting fraction $\beta_t \in [-1, 1]$ implying that the gambler cannot borrow money. We also slightly generalize the problem by allowing the outcome of the coin flip $\boldsymbol{c}_t$ to be a vector in $\mathbb{R}^d$ with L_2 norm bounded by 1, with the definition of the wealth in (3) generalized through inner products.

Now, we give a proof sketch of how it is possible to reduce optimization to a coin-betting algorithm. Consider the function $F(x) := |x - 10|$ and the optimization problem $\min_x F(x)$. We set the outcome of the coin flip c_t to be equal to the negative subgradient g_t of F in x_t, that is $c_t = -g_t \in \partial[-F(x_t)] \in \{-1, 1\}$, where x_t is the bet. Let's

also assume that there exists a function $H(\cdot)$ such that our betting strategy guarantees that Wealth_T is at least $H(\sum_{t=1}^T g_t)$ for any arbitrary sequence $g_1, \cdots, g_T$.

We claim that the average of the bets, $F\left(\frac{1}{T}\sum_{t=1}^T x_t\right)$, converges to the minimum, $F(x^\star)$, with a rate dependent on how good our betting strategy is. In fact, we have

$$\begin{aligned} F\left(\frac{1}{T}\sum_{t=1}^T x_t\right) - F(x^\star) &\leq \frac{1}{T}\sum_{t=1}^T F(x_t) - F(x^\star) \\ &\leq \frac{1}{T}\sum_{t=1}^T g_t x^\star - \frac{1}{T}\sum_{t=1}^T g_t x_t \\ &\leq \tfrac{1}{T} + \tfrac{1}{T}\left(\sum_{t=1}^T g_t x^\star - H\left(\sum_{t=1}^T g_t\right)\right) \\ &\leq \tfrac{1}{T} + \tfrac{1}{T}\max_v \; vx^\star - H(v) = \tfrac{H^\star(x^\star)+1}{T}, \end{aligned}$$

where in the first inequality we used Jensen's inequality, in the second the definition of subgradients, in the third our assumption on H, and in the last equality the definition of Fenchel conjugate of H.

In words, we can use any gambling algorithm to find the minimizer of a non-smooth objective function by accessing its subgradients. Note that the outlined approach works in any dimension for any convex objective function, even with stochastic subgradients (Orabona and Pal 2016). In particular, using the Krichevsky-Trofimov (KT) betting strategy that sets the signed betting fraction at time t equal to $\beta_t = \frac{\sum_{i=1}^{t-1} c_t}{t}$ (Krichevsky and Trofimov 1981), Orabona and Pal (2016) obtained the optimal parameter-free algorithm in (2).

Truncated Linear Models. It is well-known that first-order optimization algorithms for convex functions can be thought as minimizing surrogate lower bounds to the objective function F, instead of directly minimizing it. That is, at step t instead of trying to minimize $F(\boldsymbol{x})$, we minimize the first order approximation of F in $\boldsymbol{x}_t$, i.e., $F(\boldsymbol{x}_t)+\langle \boldsymbol{g}_t, \boldsymbol{x}-\boldsymbol{x}_t\rangle$ where $\boldsymbol{g}_t \in \partial F(\boldsymbol{x}_t)$. Of course, this is a linear function with a minimum at infinity. Hence, we minimize the function *constrained to a local neighborhood of $\boldsymbol{x}_t$ w.r.t. the L_2 norm*. Using a Lagrangian multiplier, we obtain exactly the subgradient descent algorithm.

This view of subgradient descent as minimizing local lower bounds to the function immediately gives us a way to obtain new algorithms. For example, changing the metric of the constraint, we go from subgradient descent to Mirror Descent (Beck and Teboulle 2003). More recently, Asi and Duchi (2019) proposed to substitute the linear approximation with a *truncated linear approximation*, that is

$$\tilde{F}_t(\boldsymbol{x}) = \max[F(\boldsymbol{x}_t) + \langle \boldsymbol{g}_t, \boldsymbol{x} - \boldsymbol{x}_t\rangle, F_-], \qquad (4)$$

where F_- is a lower bound to the value of the function. For example, we can set F_- to 0 if we know that the objective function is non-negative. Coupling this local lower bound with the constraint of staying not too far from $\boldsymbol{x}_t$ in a L_2 sense, Asi and Duchi (2019) obtained a new class of optimization algorithms, aProx, that are more robust to the setting of the learning rate in low-noise regimes.

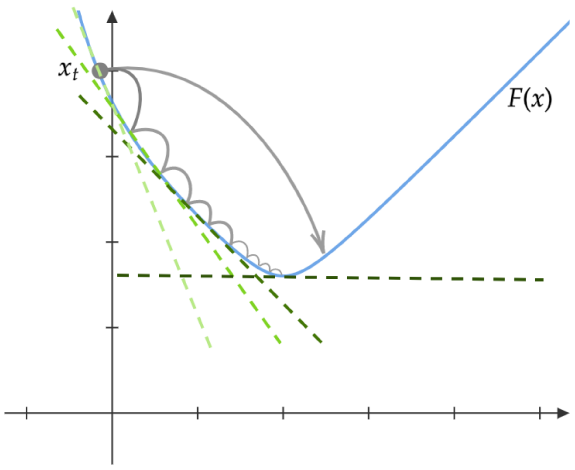

Figure 2: ODE updates versus standard updates.

3 ODE Updates for Coin-Betting Optimization

In this section, we describe our Coin-betting ODE (CODE) algorithm. The main idea is to discretize each update of the parameter-free coin betting algorithm (2) in an infinite series of infinitely small updates. Moreover, thanks to the use of the truncated linear model described in the previous section, we are still able to recover a closed-form solution for each update. It is worth noting that a direct application of the truncation method would require a mirror descent optimization algorithm, but betting does not belong to the mirror descent family. Indeed, there are no direct ways to merge truncation and betting techniques, other than the ODE's approach we propose in the following.

We proceed in stages towards the closed-form update. First, note that parameter-free algorithms require the stochastic gradients $\boldsymbol{g}_t$ to be bounded, see lower bound in (Cutkosky and Boahen 2017). For simplicity, we set this bound to be equal to 1. Now, we introduce a straightforward improvement of the update rule (2):

$$\boldsymbol{x}_{t+1} = \frac{-\sum_{i=1}^t \boldsymbol{g}_i}{1+\sum_{i=1}^t \mathbf{1}[\boldsymbol{g}_i \neq \mathbf{0}]}\left(1 - \sum_{i=1}^t \langle \boldsymbol{g}_i, \boldsymbol{x}_i\rangle\right). \qquad (5)$$

This update gives an improved convergence rate that depends on $O(T^{-1}\sqrt{\sum_{t=1}^T \mathbf{1}[\boldsymbol{g}_t \neq \mathbf{0}]})$, where $\boldsymbol{g}_t \in \partial F(\boldsymbol{x}_t)$, rather than $O(T^{-1/2})$. We defer the proof of this result to the Appendix.

Second, let's explain the idea behind having infinitely many steps. Consider an online convex optimization algorithm that at time t predicts $\boldsymbol{x}_t$, receives the subgradient, and then updates its prediction as $\boldsymbol{x}_{t+1}$. What if we instead make a step of 1/10th of the original size, we receive a subgradient, update again with 1/10 of the step, and so on for 10 times? In subgradient descent, the update in the first case can end up far from the minimum of the original objective function due to overshooting. With 10 updates of 1/10th size, we instead expect the optimization algorithm to land much closer to the minimum of the objective function using the informational

advantage of 10 subgradients with 10 small steps. See Figure 2 for a graphic representation of this idea. The exact same phenomenon occurs in coin-betting optimization algorithms. In CODE, by using truncated linear models, we get additional gradient information at the corner of the hinge.

However, in the above example, it is clear that considering 10 updates with weights 1/10 is arbitrary. Indeed, we could push this reasoning to the limit and have $1/\delta$ updates over the losses $\delta\ell_t$ when δ goes to zero. In this case, the algorithm follows a *continuous trajectory* rather than a discrete one. While this reasoning is compelling, calculating the trajectory along the gradient flow is computationally difficult. For example, with 10 updates of 1/10th size, the algorithm will request 10 gradients.

A first idea to solve this problem was proposed by Karampatziakis and Langford (2011), considering only objective functions of the form $F(\boldsymbol{x}) = \mathbb{E}_t[\ell_t(\boldsymbol{x})]$, where $\ell_t(\boldsymbol{x}) = \ell(\langle \boldsymbol{z}_t, \boldsymbol{x}\rangle, y_t)$. In this special case, only the derivative of ℓ changes under updates, not the direction of the gradient. Taking advantage of this, it is possible to calculate the final $\boldsymbol{x}_{t+1}$ without computing the entire trajectory with additional information at the truncated linear models' corners. In other words, we do not need additional gradients to derive $\boldsymbol{x}_{t+1}$.

So, the last key component is to use the truncated linear models described in Section 2. In particular, if we use the updates in (5) on the function $\tilde{F}_t$ in (4), the subgradient of $\tilde{F}_t$ always has the same direction and it is zero when we are in the flat part of the function. Hence, using (5) with infinitesimally small weights of the function $\tilde{F}_t$ the update could never go beyond the point where $\tilde{F}_t(\boldsymbol{x}) = F_-$. Moreover, a closed-form solution along the continuous trajectory becomes possible.

Now that we have all the pieces, we state the closed-form solution that gives the final update rule.

Theorem 1. *Set* $\mathrm{Wealth}_1 = 1$ *and let* $F(\boldsymbol{x})$ *be a 1-Lipschitz convex function. Define* $\boldsymbol{\theta}_1 = \mathbf{0} \in \mathbb{R}^d$, $H_1 = 1$, *and* $\boldsymbol{g}_t \in \partial F(\boldsymbol{x}_t)$ *for* $t = 1, \dots, T$. *Then, the limit as* δ *approaches 0 of running* (5) *over* $1/\delta$ *updates with the surrogate functions* $\delta\tilde{F}_t$ *gives the update rule* $\boldsymbol{x}_{t+1} = \boldsymbol{\psi}(t, h_t)$, *where*

$$\boldsymbol{\psi}(t,h) := \frac{\mathrm{Wealth}_t\, e^{-\langle \boldsymbol{g}_t, \boldsymbol{\theta}_t\rangle \ln\left(1+\frac{h}{H_t}\right)+\|\boldsymbol{g}_t\|^2\left(h+H_t \ln \frac{H_t}{H_t+h}\right)}}{H_t+h}\left(\boldsymbol{\theta}_t - h\boldsymbol{g}_t\right), \tag{6}$$

$$h_t := \min(\tilde{h}_t, 1),$$

the following quantities are defined recursively as

$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - h_t\boldsymbol{g}_t, \qquad H_{t+1} := H_t + h_t,$$

$$\mathrm{Wealth}_{t+1} := \mathrm{Wealth}_t\, e^{-\langle \boldsymbol{g}_t, \boldsymbol{\theta}_t\rangle \ln\left(1+\frac{h_t}{H_t}\right)+\|\boldsymbol{g}_t\|^2\left(h_t+H_t \ln \frac{H_t}{H_t+h_t}\right)},$$

and $\tilde{h}_t$ *is the zero of the function*

$$\phi(h) := F(\boldsymbol{x}_t) + \langle \boldsymbol{g}_t, \boldsymbol{\psi}(t,h) - \boldsymbol{x}_t\rangle - F_- \,. \tag{7}$$

Hence, to find the closed formula of the update, we only need to find the value of h_t in each round. This depends on the zero of a one-dimensional function, hence it can be found with any standard algorithm, like bisection or the Newton algorithm. Given the closed-form update, we can now state our CODE algorithm, Algorithm 1. Note that, given the value of h_t, the computational complexity of each update is $O(d)$, like in SGD.

Algorithm 1: Coin-betting ODE (CODE) Algorithm

Initialize: $\mathrm{Wealth}_0 = 1$, $H_1 = 1$, $\boldsymbol{\theta}_1 = \mathbf{0} \in \mathbb{R}^d$
for $t = 1, \dots, T$ **do**
  Query point $\boldsymbol{x}_t = \frac{\mathrm{Wealth}_t}{H_t}\boldsymbol{\theta}_t$
  Receive $\boldsymbol{g}_t$ such that $\mathbb{E}[\boldsymbol{g}_t] \in \partial F(\boldsymbol{x}_t)$, $\|\boldsymbol{g}_t\| \le 1$
  Calculate $h_t = \min(1, \tilde{h}_t)$, where $\tilde{h}_t$ is the zero of the function ϕ in (7)
  Update $\mathrm{Wealth}_{t+1} = \mathrm{Wealth}_t\, e^{-\langle \boldsymbol{g}_t, \boldsymbol{\theta}_t\rangle \ln\left(1+\frac{h_t}{H_t}\right)+\|\boldsymbol{g}_t\|^2\left(h_t+H_t \ln \frac{H_t}{H_t+h_t}\right)}$
  Update $H_{t+1} = H_t + h_t$
  Update $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - h_t\boldsymbol{g}_t$
end for

Proof of the update rule. To obtain the closed form solution in Theorem 1, we break each update of the coin-betting optimization algorithm into $1/\delta$ "mini-updates" over the truncated surrogate functions $\delta\tilde{F}_i$, then we take the limit for δ that goes to 0 and derive the final update rule in Theorem 1. This means that we have $1/\delta$ mini-updates between $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$, that give us the points $\boldsymbol{x}_{t+\delta}, \boldsymbol{x}_{t+2\delta}, \dots, \boldsymbol{x}_{t+1-\delta}, \boldsymbol{x}_{t+1}$. The following lemma fully characterizes these mini-updates.

Lemma 2. *Assume that at time* t *the Wealth of the coin-betting optimization algorithm is* Wealth_t, *the number of non-zero subgradients received is* H_t, *and the sum of the previous weighted subgradients is* $\boldsymbol{\theta}_t$. *Receive the subgradient* $\boldsymbol{g}_t$, *where* $\|\boldsymbol{g}_t\| \le 1$. *Then, breaking the update in* (5) *in* $1/\delta$ *mini-updates with weights* δ *over the truncated linear model* $\tilde{F}_t(\boldsymbol{x}_t)$ *is equivalent to the updates*

$$\mathrm{Wealth}_{t+1} = \mathrm{Wealth}_t + w_{t+1},$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \delta\boldsymbol{g}_t \sum_{j=\delta}^{1-\delta} s_j, \qquad H_{t+1} = H_t + \delta \sum_{j=\delta}^{1-\delta} s_j,$$

where for any $j = 0, \delta, 2\delta, \dots, 1-\delta$, *we define* $w_t := 0$, $s_j := \mathbf{1}[\tilde{F}_t(\boldsymbol{x}_{t+j}) \neq F_-]$, *and* $w_{t+j+\delta} := w_{t+j} - \delta s_j \langle \boldsymbol{g}_t, \boldsymbol{x}_{t+j}\rangle$.

Proof. We use the fact that $\tilde{F}_t$ is differentiable everywhere but in $\boldsymbol{x}_t$ where we choose as subgradient the zero vector. So, overall the subgradient can only assume the value $\boldsymbol{g}_t$ and $\mathbf{0}$. In particular, the subgradient is the null vector iff $s_j = 0$. This proves the updates of w_t and $\boldsymbol{\theta}_t$, while $\boldsymbol{\theta}_t$ and H_t are updated accordingly to the update rules in (5). □

We now consider the case when δ goes to 0 and prove the Theorem 1.

Proof of Theorem 1. We now consider the limit of the previous mini-updates in Lemma 2 when δ goes to zero. We immediately obtain that

$$
\begin{aligned}
w_t'(j) &:= \lim_{\delta\to 0} \frac{w_{t+j+\delta} - w_{t+j}}{\delta} \\
&= \lim_{\delta\to 0} -\frac{\delta \mathbf{1}[\tilde{F}_t(\boldsymbol{\psi}(t,j)) \neq F_-]\langle \boldsymbol{g}_t, \boldsymbol{\psi}(t,j)\rangle}{\delta} \\
&= -\mathbf{1}[\tilde{F}_t(\boldsymbol{\psi}(t,j)) \neq F_-]\langle \boldsymbol{g}_t, \boldsymbol{\psi}(t,j)\rangle, \qquad (8)
\end{aligned}
$$

where $\boldsymbol{\psi}$ is some function that represents the continuous trajectory of the iterates. In particular, $\boldsymbol{x}_t = \boldsymbol{\psi}(t,0)$. Later, we will prove that $\boldsymbol{\psi}$ coincides with the one we defined in (6). Also, defining h_t as

$$
h_t := \lim_{\delta\to 0} \delta \sum_{j=\delta}^{1-\delta} s_j = \int_0^1 \mathbf{1}[\tilde{F}_t(\boldsymbol{\psi}(t,j)) \neq F_-]\,\mathrm{d}j,
$$

we have

$$
\lim_{\delta\to 0}\ \boldsymbol{\theta}_t + \delta \sum_{j=\delta}^{1-\delta} s_j \boldsymbol{g}_t = \boldsymbol{\theta}_t + h_t \boldsymbol{g}_t,
$$

$$
\lim_{\delta\to 0}\ H_t + \delta \sum_{j=\delta}^{1-\delta} s_j = H_t + h_t\ .
$$

Hence, using the above results in (5), we obtain the trajectory of $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$ is described by

$$
\boldsymbol{\psi}(t,j) = \frac{\mathrm{Wealth}_t + w_t(j)}{H_t + j}(\boldsymbol{\theta}_j - j\boldsymbol{g}_t).
$$

Together with (8), this implies that $w_t'(j) = 0$ for $j \geq h_t$, while for $j \leq h_t$ we have

$$
w_t'(j) = -\left\langle \boldsymbol{g}_t, \frac{\mathrm{Wealth}_t + w_t(j)}{H_t + j}(\boldsymbol{\theta}_j - j\boldsymbol{g}_t) \right\rangle\ . \qquad (9)
$$

To simplify the notation, denote by

$$
P(j) = \frac{1}{H_t + j}\langle \boldsymbol{g}_t, \boldsymbol{\theta}_t - j\boldsymbol{g}_t\rangle
$$

and

$$
Q(j) = -\frac{\mathrm{Wealth}_t}{H_t + j}\langle \boldsymbol{g}_t, \boldsymbol{\theta}_t - j\boldsymbol{g}_t\rangle\ .
$$

Note that $Q(j) = -\mathrm{Wealth}_j\, P(j)$. Hence, we can rewrite (9) as $w_t'(j) + w_t(j)P(j) = Q(j)$. Solving this first-order, linear, inhomogeneous ODE, we get

$$
\begin{aligned}
w_t'(j) e^{\int P(j)\,\mathrm{d}j} &= \left[\int e^{\int P(j)\,\mathrm{d}j} Q(j)\,\mathrm{d}j + C\right] \\
&= -\mathrm{Wealth}_t \int e^{\int P(j)\,\mathrm{d}j} d\left(\int P(j)\,\mathrm{d}j\right) + C \\
&= -\mathrm{Wealth}_t\, e^{\int P(j)\,\mathrm{d}j} + C,
\end{aligned}
$$

where C is a constant. Next, we need to solve for C. Consider that

$$
\begin{aligned}
&\int_0^{h_t} P(j)\,\mathrm{d}j = \int_0^{h_t} \frac{1}{H_t + j}\langle \boldsymbol{g}_t, \boldsymbol{\theta}_t - j\boldsymbol{g}_t\rangle\,\mathrm{d}j \\
&= \langle \boldsymbol{g}_t, \boldsymbol{\theta}_t\rangle \int_0^{h_t} \frac{1}{H_t + j}\,\mathrm{d}j - \|\boldsymbol{g}_t\|^2 \int_0^{h_t} \frac{j}{H_t + j}\,\mathrm{d}j \\
&= \langle \boldsymbol{g}_t, \boldsymbol{\theta}_t\rangle \ln(H_t + h_t) - \|\boldsymbol{g}_t\|^2 (h_t - H_t \ln(H_t + h_t))\ .
\end{aligned}
$$

Hence, we have

$$
\begin{aligned}
w_t(h_t) &= -\mathrm{Wealth}_t \\
&\quad + C e^{-\langle \boldsymbol{g}_t, \boldsymbol{\theta}_t\rangle \ln(H_t + h_t) + \|\boldsymbol{g}_t\|^2 (h_t - H_t \ln(H_t + h_t))}.
\end{aligned}
$$

Since $w_t(0) = 0$, we have

$$
C = \mathrm{Wealth}_t\, e^{\langle \boldsymbol{g}_t, \boldsymbol{\theta}_t\rangle \ln(H_t) + \|\boldsymbol{g}_t\|^2 H_t \ln(H_t))}.
$$

Finally, we have

$$
\frac{w_t(h_t)}{\mathrm{Wealth}_t} = e^{-\langle \boldsymbol{g}_t, \boldsymbol{\theta}_t\rangle \ln\left(1 + \frac{h_t}{H_t}\right) + \|\boldsymbol{g}_t\|^2 \left(h_t + H_t \ln \frac{H_t}{H_t + h_t}\right)} - 1\ .
$$

Using the fact that $\frac{\mathrm{Wealth}_{t+1}}{\mathrm{Wealth}_t} = 1 + \frac{w_t(h_t)}{\mathrm{Wealth}_t}$, we have the closed form expression of the wealth. This also provides an expression of the evolution of $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$, that coincides with $\boldsymbol{\psi}(t,j)$ in (6). □

4 Empirical Evaluation

Here, we compare CODE with SGD, SGD with truncated models (aProx) (Asi and Duchi 2019), SGD with Importance Weight Aware updates (IWA) (Karampatziakis and Langford 2011), AdaGrad (Duchi, Hazan, and Singer 2011), Adam (Kingma and Ba 2015), the coin-betting algorithm in (2) (Coin) (Orabona and Pal 2016) and the recursive coin-betting algorithm (Recursive) (Cutkosky and Sarlos 2019). For SGD, aProx and IWA, we use the optimal worst-case step size for stochastic convex optimization: $\eta_k = \eta_0/\sqrt{k}$, and tune the initial step size η_0. In the adaptive learning rate methods, AdaGrad and Adam, we tune the initial step size η_0. CODE, Coin and Recursive do not have learning rates.

4.1 Train/Test on Real Datasets

We test the ability of CODE to get a good generalization error. Hence, we perform experiments with 21 different machine learning binary classification datasets and 17 regression datasets from the LIBSVM website (Chang and Lin 2011) and OpenML(Vanschoren et al. 2013). We implement extensive experiments on a large number of datasets to verify the significance of our results. We pre-process the samples normalizing them to unit norm vectors. We shuffle the data and use 70% for training, 15% for validation, and hold out 15% for testing. Given the lack of a regularizer, all the algorithms pass on the training set once to avoid overfitting (see, e.g., Section 14.5.1 Shalev-Shwartz and Ben-David 2014). We evaluate algorithms with 0-1 loss for classification tasks and absolute loss for regression tasks, normalizing the scores by the performance of the best constant predictor. In this way, each dataset is weighted equally independently by how hard it is. Otherwise, a single hard dataset would dominate the average loss. All the experiments are repeated 3 times and we take the mean of the 3 repetitions. See Appendix for more details on datasets, experiments, and numerical values.

Best Fixed Learning Rates. Clearly, the process of tuning hyperparameters is computationally expensive. For example, if we want to try 2 different learning rates, we have to run SGD twice per dataset. Hence, to have a fair comparison in terms of computational cost, here we consider this setting:

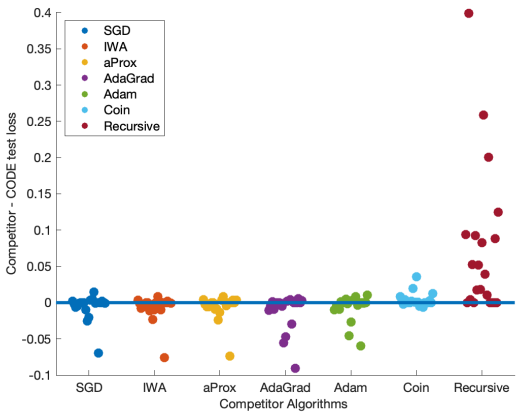

Figure 3: Difference between normalized competitor and CODE test losses using the "best default" parameter on 21 classification datasets.

Algorithm	Single Learning Rate	
	Mean normalized absolute loss	Mean normalized 0-1 loss
SGD	1.1440	0.1816
IWA	0.8787	0.1804
aProx	0.9192	0.1812
AdaGrad	1.0941	**0.1757**
Adam	1.2362	0.1806
Coin	0.9294	0.1906
Recursive	10.0576	0.2601
CODE	**0.7668**	0.1870

Table 1: Average normalized test set accuracies on 17 regression and 21 classification datasets with best fixed learning rates.

we test the common belief that many optimization algorithms have a "default" learning rate that works on every dataset. If this were true, tuning would not be an extra cost. To test this scenario, we tune the learning rate of the baselines to achieve the best *average of normalized performance over all datasets directly on the test sets*. In other words, we choose the "best default" parameter of each algorithm to minimize the numbers in Table 1. This is strictly better for the baselines than just choosing some fixed default parameters for each of them.

First, we compare all the algorithms on linear regression problems with the absolute loss. We summarize the results in Figure 1 in the Introduction and in Table 1. In the figure, each point represents a baseline algorithm (x-axis) vs. the normalized test loss difference between the algorithm and CODE (y-axis) on one dataset. So, points located above $y = 0$ represent datasets where CODE outperforms the baseline algorithm. We can see that CODE on average is superior to all other algorithms. The mean of normalized absolute loss of SGD, AdaGrad, Adam, Coin, and Recursive is greater than 1,

Algorithm	One Learning Rate per Dataset	
	Mean normalized absolute loss	Mean normalized 0-1 loss
SGD	0.7150	0.1745
IWA	0.7133	0.1775
aProx	0.0.7201	0.1778
AdaGrad	**0.7000**	**0.1544**
Adam	0.7022	0.1596
Coin	0.9294	0.1906
Recursive	10.0576	0.2601
CODE	0.7668	0.1870

Table 2: Average normalized test set accuracies on 17 regression and 21 classification datasets with tuned learning rates.

indicating that these baseline algorithms perform worse than the best constant predictor on average. The reason is clear: on these datasets, no single learning rate can work on all of them. Furthermore, CODE wins Coin by ~ 0.16, which proves that the ODE updates boost the performance of the parameter-free algorithm. CODE also wins Recursive significantly. Overall, CODE essentially guarantees the best performance *without any parameter tuning*.

We also test the performance of all algorithms on classification problems. In Figure 3 and in Table 1, we can see that all algorithms except Recursive attained similar performance. CODE is worse than AdaGrad on average ~ 0.0113 and is better than Coin and Recursive. We performed two-sample paired t-test between CODE and competitors and failed to reject the null hypothesis that the performance of CODE on average is as good as the AdaGrad at the significant level $\alpha = 0.05$.

Tuned Learning Rates. We now turn to the case in which we ignore the computational complexity and we tune all the learning rates for SGD, IWA, aProx, AdaGrad, and Adam. For each repetition and dataset, we use the validation set to select the best learning rate, train using that learning rate, test on the test set and report the average of normalized loss. Results are summarized in Figure 4 and Table 2.

As just said, this is a very expensive procedure and not a fair comparison for parameter-free algorithms in terms of computational cost. Yet, in both regression and classification tasks, the performance of CODE and other algorithms except Recursive are close to each other. Remember the fact that CODE achieves this performance without any tuning. As a parameter-free algorithm, CODE only loses over the best algorithm on average ~ 0.0668 on regression problems and ~ 0.0326 on classification problems. The difference on average is not significant in statistics at the significance level 0.05. We believe that there are many settings where such loss of accuracy would be negligible compared to what we gain from removing the need to tune the learning rate. It is also instructive to notice how much the performance of the baselines improves when we move from a "default" learning rate to a tuned one. In other words, to achieve the best opti-

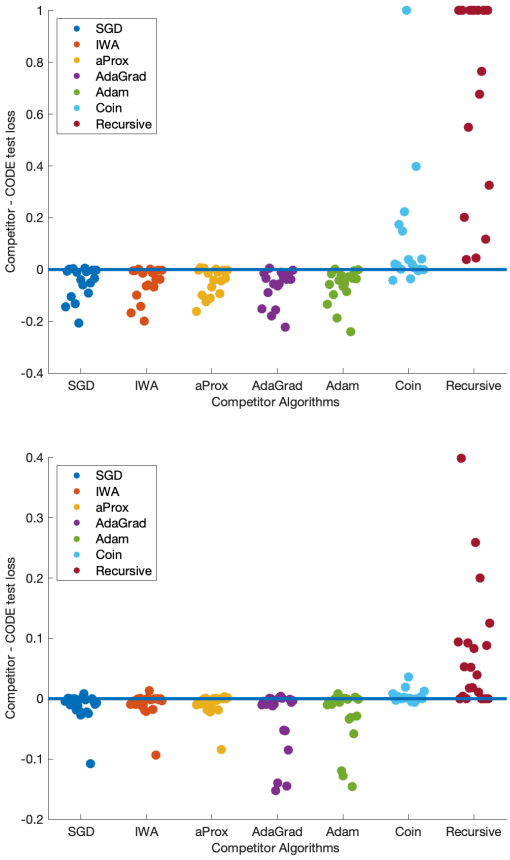

Figure 4: Difference between normalized competitor and CODE test losses using a tuned learning rate on 17 regression (1) and 21 classification (2) datasets.

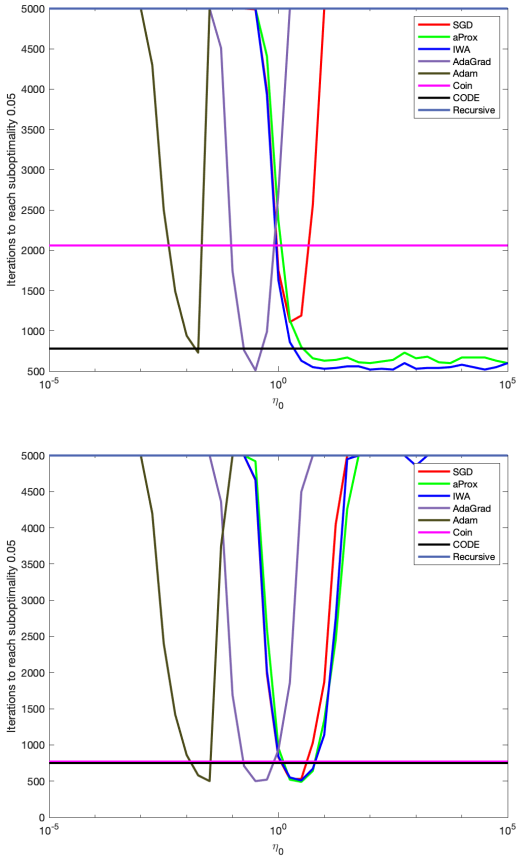

Figure 5: Synthetic dataset with absolute loss. Number of iteration to reach 0.05 suboptimality gap versus initial step sizes η_0. (1) noiseless setting, (2) $\sigma = 1/2$.

mal performance with, for example, AdaGrad the parameter tuning cannot be avoided, at least on regression problems.

4.2 Sub-Optimality Gap on Synthetic Datasets

We also generate synthetic data and test the algorithms following the protocol in Asi and Duchi (2019), to observe the sensitivity of the algorithms to the setting of the step sizes. Of course, the parameter-free ones do not have any step size to set. For $A \in \mathbb{R}^{m \times d}$ and $\boldsymbol{y} \in \mathbb{R}^{m \times 1}$ the objective function we want to minimize is $F(\boldsymbol{x}) = \frac{1}{m}\|A\boldsymbol{x} - \boldsymbol{y}\|_1$, which corresponds to a regression problem with the absolute loss. In each experiment, we generate $\boldsymbol{x}^\star \sim N(0, I_d) \in \mathbb{R}^d$, and set $\boldsymbol{y} = A\boldsymbol{x}^\star + \sigma \boldsymbol{v}$ for $\boldsymbol{v} \sim N(0, I_d)$, where σ controls the amount of noise. We generate A with uniformly random orthogonal columns, with $m = 1000$ and $d = 40$. Then, we normalize the L_2 norm of each sample. We repeat the above data generation process 10 times and show the average results in the plots. We also consider the classification setting, see similar experiments and plots in the Appendix.

As in Asi and Duchi (2019), we study how many iterations are needed to reach a suboptimality gap of 0.05, that is we plot the smallest k such that $F(\boldsymbol{x}_k) - F(\boldsymbol{x}^\star) \leq 0.05$. In Figure 5, we show the results. As expected, the performance of SGD, Adam, and AdaGrad is extremely sensitive to the setting of the step size. We really need to find the right one, otherwise the convergence slows down catastrophically. Instead, IWA and aProx have a much better range of acceptable step sizes for the noise-free case. However, their advantage almost disappears in the noisy case. On the other hand, we can see that the parameter-free algorithms, CODE, perform very well, with *CODE achieving essentially optimal performance in both regimes*. Moreover, CODE inherits the better performance of aProx in the noise-free case, gaining a big advantage over Coin, but still maintaining a closed-form solution.

5 Discussion

We have presented a new parameter-free method called CODE, which is the first work combining the truncated linear approximation and continuous updates for Coin-Betting optimization algorithms. The empirical results show that CODE can outperform algorithms with a "default" learning rate and be very competitive with finely-tuned ones. In future work, we plan to investigate theoretical guarantees for CODE.

Acknowledgements

This material is based upon work supported by the National Science Foundation under grants no. 1925930 "Collaborative Research: TRIPODS Institute for Optimization and Learning", no. 1908111 "AF: Small: Collaborative Research: New Representations for Learning Algorithms and Secure Computation", and no. 2046096 "CAREER: Parameter-free Optimization Algorithms for Machine Learning".

References

Asi, H.; and Duchi, J. C. 2019. Stochastic (approximate) proximal point methods: Convergence, optimality, and adaptivity. *SIAM Journal on Optimization*, 29(3): 2257–2290.

Beck, A.; and Teboulle, M. 2003. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3): 167–175.

Chang, C.-C.; and Lin, C.-J. 2011. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

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

Cutkosky, A.; and Boahen, K. 2017. Online Learning Without Prior Information. In *Proc. of the 2017 Conference on Learning Theory*, volume 65 of *Proc. of Machine Learning Research*, 643–677. Amsterdam, Netherlands: PMLR.

Cutkosky, A.; and Boahen, K. A. 2016. Online convex optimization with unconstrained domains and losses. In *Advances in Neural Information Processing Systems*, 748–756.

Cutkosky, A.; and Orabona, F. 2018. Black-Box Reductions for Parameter-free Online Learning in Banach Spaces. In *Proc. of the Conference on Learning Theory (COLT)*.

Cutkosky, A.; and Sarlos, T. 2019. Matrix-Free Preconditioning in Online Learning. arXiv:1905.12721.

Duchi, J. C.; Hazan, E.; and Singer, Y. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12: 2121–2159.

Foster, D. J.; Rakhlin, A.; and Sridharan, K. 2018. Online Learning: Sufficient Statistics and the Burkholder Method. In *Proc. of the Conference on Learning Theory*.

Karampatziakis, N.; and Langford, J. 2011. Online Importance Weight Aware Updates. In *Proc. of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'11, 392—-399. Arlington, Virginia, USA: AUAI Press.

Kempka, M.; Kotłowski, W.; and Warmuth, M. K. 2019. Adaptive Scale-Invariant Online Algorithms for Learning Linear Models. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proc. of the 36th International Conference on Machine Learning*, volume 97 of *Proc. of Machine Learning Research*, 3321–3330. Long Beach, California, USA: PMLR.

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Kivinen, J.; and Warmuth, M. 1997. Exponentiated Gradient versus Gradient Descent for Linear Predictors. *Information and Computation*, 132(1): 1–63.

Kotłowski, W. 2019. Scale-invariant unconstrained online learning. *Theoretical Computer Science*.

Krichene, W.; Bayen, A.; and Bartlett, P. L. 2015. Accelerated Mirror Descent in Continuous and Discrete Time. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28*, 2845–2853. Curran Associates, Inc.

Krichevsky, R.; and Trofimov, V. 1981. The performance of universal encoding. *IEEE Trans. on Information Theory*, 27(2): 199–207.

Kulis, B.; and Bartlett, P. L. 2010. Implicit Online Learning. In *ICML*, 575–582.

McMahan, H. B.; and Orabona, F. 2014. Unconstrained Online Linear Learning in Hilbert Spaces: Minimax Algorithms and Normal Approximations. In *Proc of the Annual Conference on Learning Theory, COLT*.

Orabona, F. 2014. Simultaneous Model Selection and Optimization through Parameter-free Stochastic Learning. In *Advances in Neural Information Processing Systems 27*.

Orabona, F. 2019. A Modern Introduction to Online Learning. *arXiv preprint arXiv:1912.13213*.

Orabona, F.; and Pal, D. 2016. Coin Betting and Parameter-Free Online Learning. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*, 577–585. Curran Associates, Inc.

Rockafellar, R. T. 1976. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5): 877–898.

Shalev-Shwartz, S.; and Ben-David, S. 2014. *Understanding Machine Learning: From Theory to Algorithms*. New York, NY, USA: Cambridge University Press.

Su, W.; Boyd, S.; and Candes, E. J. 2015. A Differential Equation for Modeling Nesterov's Accelerated Gradient Method: Theory and Insights. arXiv:1503.01243.

Vanschoren, J.; van Rijn, J. N.; Bischl, B.; and Torgo, L. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations*, 15(2): 49–60.

Zinkevich, M. 2003. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *Proc. of ICML*, 928–936.