Scaling Machine Learning via Prioritized Optimization

Tyler B. Johnson

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2018

Reading Committee:

Carlos Guestrin, Chair

Maryam Fazel, Chair

Sham Kakade

Program Authorized to Offer Degree:
Electrical Engineering

University of Washington

Abstract

Scaling Machine Learning via Prioritized Optimization

Tyler B. Johnson

Co-Chairs of the Supervisory Committee:
Carlos Guestrin
Department of Computer Science & Engineering

Maryam Fazel
Department of Electrical Engineering

To learn from large datasets, modern machine learning applications rely on scalable training algorithms. Typically such algorithms employ stochastic updates, parallelism, or both. This work develops scalable algorithms via a third approach: prioritized optimization.

We first propose a method for prioritizing challenging tasks when training deep models. Our robust approximate importance sampling procedure (RAIS) speeds up stochastic gradient descent by sampling minibatches non-uniformly. By approximating the ideal sampling distribution using robust optimization, RAIS provides much of the benefit of exact importance sampling with little overhead and minimal hyperparameters.

In the second part of this work, we develop strategies for prioritizing optimization when solving convex problems with piecewise linear structure. Our BlitzWS working set algorithm offers unique theoretical guarantees and solves several classic machine learning problems very efficiently in practice. We also propose a closely related safe screening test, BlitzScreen, which is state-of-the-art for safe screening in multiple ways.

Our final contribution is a "stingy update" rule for coordinate descent. Our StingyCD algorithm prioritizes optimization variables by eliminating provably useless computation. StingyCD requires only simple changes to CD and results in significant speed-ups in practice.

DEDICATION

to Amanda

ACKNOWLEDGMENTS

Completing a PhD program is one of my proudest accomplishments. I depended on many friends to help me reach this goal.

I feel fortunate to have worked with a supportive and brilliant advisor, Carlos Guestrin. I appreciate many things about working with Carlos: the freedom and encouragement in choosing projects, his honest and useful feedback, and the delicate balance of urgency and humor during our meetings. Most of all, I thank Carlos for consistently demanding my best effort. My abilities have grown more than I anticipated, and I am grateful for that.

I found Maryam Fazel's convex optimization course more interesting and more challenging than any other course I encountered at UW. My research relies heavily on topics from this class, and I am thankful for opportunities to both take and help instruct the course. I also appreciate Maryam taking an interest in my research and helping me develop these ideas. I also thank the other members of my supervisory committee, Sham Kakade and Ali Shojaie, for their additional feedback and support.

Many fellow students and colleagues also deserve acknowledgment for setting me up for success. I especially thank Joseph Bradley and Shingo Takamatsu, who mentored me during my initial years in the program. While my research has changed immensely since then, the good habits that I learned from Joseph and Shingo have remained the same. I also thank Hyokun Yun, my mentor during an internship with Amazon's Core ML team. Yun's deep curiosity for machine learning led to many enjoyable conversations, and my perspective of the field expanded greatly as a result. Finally, for the thoughtful discussion and advice, I thank many additional friends from UW, including Marco Ribeiro, Tianqi Chen, Sameer Singh, Brian Dolhansky, Dennis Meng, David Perlmutter, Philip Cho, Sachin Mehta, Rahul

Kidambi, John Halloran, Chris Xie, Chris Aicher, Alex Tank, Jay Garlapati, and Johan Ugander, just to name a few.

Past research advisors and teachers helped prepare me for graduate school. I greatly appreciate that Clayton Scott introduced me to machine learning research when I studied at the University of Michigan. During my earlier years as an undergraduate, Richard Yamada and Akram Boukai also patiently advised me on research projects, and I feel fortunate to have worked with them as well. I also thank Christine Deyo and Andy Lamkin, teachers at Rochester Adams High School, who encouraged my interest in challenging math and physics problems. In retrospect, the experience of trying—and often failing—to solve such problems prepared me well for the years ahead.

Last and most of all, I thank lifelong friends and family members for their support. Without Ian McDonald and Jason Meng, it is possible I never would have graduated, and it seems certain I never would have learned to climb. I also thank Mark Rudolf for his perspective from a distance. I find it poetic that Mark and I have pursued PhDs on opposite ends of the country, although Mark (a poetry champion) can judge that much better than I. I also especially thank my parents, Carol and Dick, for raising, teaching, and always supporting me. Finally, I especially thank my wife, Amanda. I feel extremely grateful to have married such a wonderful and supportive person.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Training a model is a fundamental task in machine learning. We often perform this task using *optimization*. We define a set of models for consideration and a function that scores each model. Training amounts to computing—in most cases approximately—the model with the most desirable score.

In order to model data, the scoring function incorporates a data fitting term, which quantifies the discrepancy between a training dataset and the model. This training setup is simple yet powerful. By training on large, informative datasets and optimizing over expressive classes of models, machine learning has produced admirable results for many applications. Notable examples include image classification (Krizhevsky et al., 2012), ad click prediction (McMahan et al., 2013), content recommendation (Koren et al., 2009), language translation (Sutskever et al., 2014), as well as many others.

As the training set size and model complexity increase, however, computational requirements also grow. In some cases, training algorithms require days or even weeks to learn a useful model. For this reason, it is important to design learning algorithms that are computationally efficient. Faster algorithms enable consideration of more data, models, or training configurations, which in turn expand the capabilities of machine learning.

Parallelism and stochastic updates are standard strategies for scaling training. Parallel algorithms divide the training task among multiple processing elements (Niu et al., 2011; Bradley et al., 2011; Low et al., 2012; Dean et al., 2012), decreasing training times by increasing computational power. Stochastic algorithms learn models incrementally via small updates, which are noisy yet computationally inexpensive. Applied sequentially, such updates result in very efficient algorithms for machine learning (Zinkevich, 2003; Duchi et al.,

2011; Shalev-Shwartz and Tewari, 2011; Kingma and Ba, 2015).

We take a third approach to scaling optimization for machine learning, which is complementary to parallelism and stochastic updates. In particular, we design algorithms that prioritize computational resources on important components of the problem. We aim to propose strategies that are both useful in practice as well as principled (in many cases backed by theoretical guarantees). Formally, our thesis statement is as follows:

Thesis statement: Standard machine learning algorithms allocate equal computational resources to all parts of the model and training set during training. This is inefficient. By prioritizing resources in a practical and principled way, we can dramatically speed up training.

At a high level, our focus on prioritization follows from the fact that an optimization problem's components—i.e., the different training examples or parts of the model—can vary greatly. Therefore, we should not necessarily allocate resources uniformly during training. Instead, we can often learn faster by training in a prioritized way.

1.1 Examples of prioritization in this work

Depending on the problem, we prioritize optimization differently. We next describe specific examples that we consider in this work.

1.1.1 Easy training examples

A stochastic algorithm trains models iteratively. Typically the algorithm initializes model parameters randomly, at which point the model handles poorly most examples in the training set. As training continues, the model's performance improves. If training works well, the model correctly handles most examples upon training completion.

At any point during training, we can roughly divide the training set into two categories: "easy" examples, which the model handles correctly, and "difficult" examples, which the

model struggles with.

Standard stochastic algorithms train on each example in the training set with equal priority, regardless of whether the example is easy or difficult. Intuitively, this is inefficient. The model learns little by reconsidering examples it handles correctly. If possible, we should prioritize computational resources on tasks that the model has not yet learned.

In Chapters 2, 3, and 4 we propose several strategies for speeding up training by exploiting easy training examples. In Chapter 2, we develop an importance sampling procedure to train deep models faster with stochastic gradient descent. In Chapters 3 and 4, we propose strategies for prioritizing training examples in order to efficiently train support vector machines (as well as other types of models).

1.1.2 Model sparsity

Sparse models make predictions using a small fraction of the model's inputs. Remaining features have no impact on the model's output.

When training a sparse model, training algorithms initially do not know whether a feature is relevant to the final model. For this reason, standard algorithms consider each feature with equal priority during training, regardless of the feature's importance to the model.

Intuitively, we can do much better. Due to sparsity, most features have *zero* impact on the final model, and yet considering such features requires significant computational resources during training. Ideally, we should design algorithms that concentrate resources on features with greatest importance and ignore features that are irrelevant to the final model. In Chapters 3 and 4, we propose several methods based on this idea.

1.1.3 Piecewise linear structure

As it turns out, some of the examples described so far—specifically easy training examples in SVMs as well as model sparsity—can be unified using the concept of "piecewise linear structure." We explain this in detail in Chapter 3. To exploit piecewise linear structure, we define the optimization objective as a sum of many piecewise terms. Each piecewise term

is comprised of simpler subfunctions, some of which we assume to be linear. Prioritizing optimization amounts to selectively replacing piecewise terms in the objective with simpler subfunctions. This results in a modified problem that can be much simpler to solve.

In Chapter 3, we propose two strategies for prioritizing piecewise linear structure—a working set algorithm and a safe screening test. By considering this general type of structure, our strategies apply simultaneously to many classic problems, including training SVMs, learning sparse models, and minimizing strongly convex objectives subject to many constraints.

1.2 Summary of contributions

We organize the remainder of this dissertation as follows. Chapters 2, 3, and 4 describe in detail our proposed strategies. In each chapter, we also provide background material and explain the relation between our contributions and prior work. In Chapter 5, we provide general conclusions, noting especially some limitations of our ideas. In this chapter, we also suggest promising directions for future work.

We summarize the main contributions from Chapters 2, 3, and 4 as follows:

- *Robust approximate importance sampling for SGD (Chapter 2):* We propose simple modifications to stochastic gradient descent in order to train deep models faster. The procedure, named RAIS, speeds up training by sampling training examples non-uniformly. Unlike prior approximate importance sampling methods, RAIS uses robust optimization to ensure convergence. As a result, RAIS depends minimally on hyper-parameters and leads to consistent and significant improvements in training time.

- *BlitzWS working set algorithm (Chapter 3):* Working set algorithms speed up training by reducing optimization to a sequence of smaller subproblems. Typically working set algorithms choose subproblems using heuristics. We propose BlitzWS, a unique working set algorithm that guarantees a specified amount of convergence progress during each iteration. This result inspires a method for adaptively selecting algorithmic parameters, which leads to very fast convergence times in practice.

- *BlitzScreen safe screening test (Chapter 3):* Safe screening tests simplify the optimization objective without affecting the problem's solution. We derive a state-of-the-art safe screening test called BlitzScreen, which is closely related to BlitzWS. BlitzScreen provides a novel link between working set algorithms and safe screening. In addition, BlitzScreen dominates prior screening tests in terms of usefulness.

- *Piecewise problem formulation (Chapter 3):* BlitzWS and BlitzScreen apply to instances of a general piecewise problem formulation. This problem formulation encompasses many classic machine learning problems, including ℓ_1-regularized learning, group lasso, and linear SVM problems. Prior to BlitzScreen, screening tests required substantial new derivations for each of these problems. In contrast, BlitzWS and BlitzScreen apply to many problems simultaneously.

- *Stingy coordinate decent updates (Chapter 4):* Coordinate descent is a simple and good algorithm for solving lasso and linear SVM optimization problems. Even so, CD can be inefficient for these problems, since many iterations result in zero improvement to the model. We propose StingyCD, a modified CD algorithm that skips over many of these "zero updates." StingyCD only skips iterations that it guarantees are useless, and since the changes introduce little overhead, StingyCD can reduce training times considerably.

Chapter 2

PRIORITIZING CHALLENGING EXAMPLES TO SPEED UP DEEP LEARNING

This chapter details our robust approximate importance sampling procedure—RAIS—for training models faster with stochastic gradient descent. Importance sampling is a procedure that, in theory, can speed up SGD by prioritizing training examples. In practice, the cost of computing importances greatly limits the impact of importance sampling.

By approximating the ideal sampling distribution using robust optimization, RAIS provides much of the benefit of exact importance sampling with drastically reduced overhead. Since RAIS still introduces some additional computational cost, the procedure is best-suited for training deep models, in which case the overhead is relatively small. Empirically, we find RAIS-SGD and standard SGD train models that achieve similar performance, but RAIS trains faster, achieving speed-ups of at least 20% and sometimes much more. RAIS requires only simple changes to SGD, and the procedure depends minimally on hyperparameters.

The work presented in this chapter will appear at the 2018 Conference on Neural Information Processing Systems (Johnson and Guestrin, 2018b).

2.1 Introduction

Deep learning models perform excellently on many tasks. Training such models is resource-intensive, however, as stochastic gradient descent algorithms can require days or weeks to train effectively. After a relatively short period training, models usually perform well on some—or even most—training examples. As training continues, frequently reconsidering such "easy" examples slows further improvement.

Importance sampling prioritizes training examples for SGD in a principled way. The

technique suggests sampling example i with probability proportional to the norm of loss term i's gradient. This distribution both prioritizes challenging examples and minimizes the stochastic gradient's variance.

SGD with optimal importance sampling is impractical, however, since computing the sampling distribution requires excessive time. Zhao and Zhang (2015) and Needell et al. (2014) analyze importance sampling for SGD and convex problems; practical versions of these algorithms sample proportional to fixed constants. For deep models, other algorithms attempt closer approximations of gradient norms (Alain et al., 2016; Katharopoulos and Fleuret, 2017, 2018). But these algorithms are not inherently robust. Without carefully chosen hyperparameters or additional forward passes, these algorithms do not converge, let alone speed up training.

We propose RAIS, an importance sampling procedure for SGD with several appealing qualities. First, RAIS determines each sampling distribution by solving a robust optimization problem. As a result, each sampling distribution is minimax optimal with respect to an uncertainty set. Since RAIS trains this uncertainty set in an adaptive manner, RAIS is not sensitive to hyperparameters.

In addition, RAIS maximizes the benefit of importance sampling by adaptively increasing SGD's learning rate—an effective yet novel idea to our knowledge. This improvement invites the idea that one RAIS-SGD iteration equates to more than one iteration of conventional SGD. Interestingly, when plotted in terms of "epochs equivalent," the learning curves of the algorithms align closely.

RAIS applies to any model that is trainable with SGD. Since RAIS introduces slight computational overhead, we focus on training deep models, in which case the overhead is insignificant compared to the cost of backpropagation. Because of its relative simplicity, RAIS combines nicely with standard "tricks," including data augmentation, dropout, and batch normalization. We show this empirically in §2.6. In this section, we also demonstrate that RAIS consistently improves training times, achieving speed-ups of at least 20% and sometimes much more. We include qualitative results from these experiments in Figure 2.1.

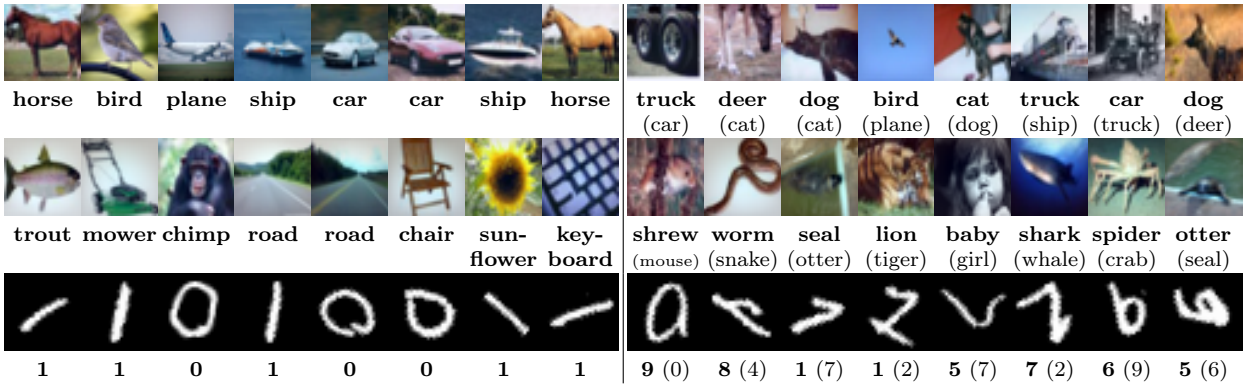


Figure 2.1: Nonpriority and priority training examples for image classification. *Left:* Examples that RAIS samples infrequently during training. *Right:* Examples that RAIS prioritizes. Bold denotes the image's label. Parentheses denote another class that the model considers likely. Datasets are CIFAR-10 (top), CIFAR-100 (middle), and rotated MNIST (bottom).

2.2 Problem formulation

Given loss functions $f_1, f_2, \ldots, f_n$ and a parameter $\lambda \in \mathbb{R}_{\geq 0}$, our task is to efficiently solve

$$\underset{\mathbf{w}\in\mathbb{R}^d}{\text{minimize}}\; F(\mathbf{w})\,, \quad \text{where} \quad F(\mathbf{w}) = \tfrac{1}{n}\textstyle\sum_{i=1}^n f_i(\mathbf{w}) + \tfrac{\lambda}{2}\left\|\mathbf{w}\right\|^2\,. \tag{P}$$

Here each f_i is a data fitting term that corresponds to example i of a training dataset. The weights $\mathbf{w}$ are model parameters, which we learn by solving (P).

A standard algorithm for solving (P) is stochastic gradient descent. Let $\mathbf{w}^{(t)}$ denote the weights when iteration t begins. SGD updates these weights via

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta^{(t)}\mathbf{g}^{(t)}\,. \tag{2.1}$$

Above, $\eta^{(t)} \in \mathbb{R}_{>0}$ is a learning rate, specified by a schedule: $\eta^{(t)} = \texttt{lr\_sched}\,(t)$. The vector $\mathbf{g}^{(t)}$ is an unbiased stochastic approximation of the gradient $\nabla F(\mathbf{w}^{(t)})$. SGD computes $\mathbf{g}^{(t)}$ by sampling a minibatch of $|\mathcal{M}|$ indices from $\{1, 2, \ldots, n\}$ uniformly at random (or approximately so). Denoting this minibatch by $\mathcal{M}^{(t)}$, SGD defines the stochastic gradient as

$$\mathbf{g}^{(t)} = \tfrac{1}{|\mathcal{M}|}\textstyle\sum_{i\in\mathcal{M}^{(t)}} \nabla f_i(\mathbf{w}^{(t)}) + \lambda\mathbf{w}^{(t)}\,.$$

In this chapter, we assume an objective function, learning rate schedule, and batch size, and we propose a modified algorithm called RAIS-SGD. RAIS prioritizes examples by sampling minibatches *non-uniformly*, allowing us to train models with less iterations and time.

2.3 Relation to prior work

The SGD algorithm described in §2.2 is standard for training deep models. There also exist many tricks that improve upon this algorithm, such as data augmentation, dropout, and momentum. Later in §2.5.5, we discuss how RAIS combines nicely with these tricks

In this section, we provide background on prior approaches that attempt to improve SGD by prioritizing training examples. In general terms, we also explain how RAIS avoids some of the shortcomings of these approaches.

2.3.1 Prior theoretical works on importance sampling for SGD

RAIS is not the first procedure to use importance sampling with SGD. Zhao and Zhang (2015) and Needell et al. (2014) first analyzed SGD with importance sampling for the case that F is convex. Zhao and Zhang showed that to optimize convergence bounds, we should sample example i for minibatch $\mathcal{M}^{(t)}$ with probability proportional to $\left\|\nabla f_i(\mathbf{w}^{(t)})\right\|$. As we will confirm in §2.4.2, this distribution minimizes the variance of the stochastic gradient.

Sampling in this optimal way is impractical, however, since computing $\left\|\nabla f_i(\mathbf{w}^{(t)})\right\|$ for all training examples requires excessive time. In practice, Zhao and Zhang and Needell et al. sample each example i with probability proportional to a Lipschitz constant L_i, which upper-bounds $\|\nabla f_i(\mathbf{w})\|$ for all $\mathbf{w}$. This procedure also relates closely to leverage score sampling, a technique that Boutsidis et al. (2009), Mahoney (2011), Ma et al. (2014), and others have used to speed up matrix approximation algorithms.

Sampling according to Lipschitz constants can lead to some improvement, especially if L_i values vary greatly among training examples. However, the gains are also quite limited, as the sampling distribution does not depend on the model (instead the distribution remains fixed throughout training). Furthermore, this approach has little use when training deep

models. Because of a deep model's many layers, similar Lipschitz bounds are weaker and less intuitive in this case.

2.3.2 Approximate importance sampling strategies for deep learning

Compared to the convex setting, computing the stochastic gradient $\mathbf{g}^{(t)}$ requires much more time when training deep models. Because of this bottleneck, *approximating* gradient norms—i.e., estimating $\left\|\nabla f_i(\mathbf{w}^{(t)})\right\|$ for all training examples—is practical.

Alain et al. (2016) propose an approximation that distributes the computation of gradient norms across a cluster of machines. In parallel with regular training, Katharopoulos and Fleuret (2017) train a miniature neural network for the purpose of predicting importance values. Katharopoulos and Fleuret (2018) approximate importance values using additional forward passes. Schaul et al. (2016) and Horgan et al. (2018) apply importance sampling to prioritize experience replay for reinforcement learning.

With the exception of (Katharopoulos and Fleuret, 2018) (which requires a large amount of time overhead to compute gradient norms), all of these prior strategies are sensitive to errors in gradient norm estimates. For this reason, all require user-specified smoothing parameters to converge. Such smoothing moves the sampling distribution toward a uniform distribution, arbitrarily decreasing the benefit of prioritization. Worse, if the smoothing parameter is too small, the algorithms diverge. In contrast, RAIS elegantly addresses uncertainty in importance values using robust optimization. RAIS adapts to the state of the algorithm, so RAIS does not require careful tuning of hyperparameters to perform well.

2.3.3 Other prioritization strategies

Besides importance sampling, researchers have considered additional ways to prioritize training examples for deep learning. Bengio et al. (2009) train on examples in order of increasing difficulty. Other researchers prioritize challenging training examples (Shrivastava et al., 2016; Shalev-Shwartz and Wexler, 2016). And yet others prioritize examples closest to the model's decision boundary (H.-S. Chang, 2017). Unlike RAIS, the primary goal of these approaches

is improved model performance, not optimization efficiency. These methods use non-uniform sampling in order to implicitly modify the objective F, and perhaps we could use importance sampling in conjunction with these methods to speed up optimization.

There also exist ideas for sampling minibatches non-uniformly outside the context of deep learning. Zhang et al. (2017, 2018) consider sampling diverse minibatches via repulsive point processes to reduce the stochastic gradient's variance. Perhaps RAIS could combine with this concept to sample minibatches that are both diverse and important. Another approach uses side information, such as class labels, to speed up optimization with importance sampling (Gopal, 2016). With RAIS, we can use side information in the same way by choosing appropriate features for our RAIS models. We note that unlike RAIS, Gopal's algorithm requires an additive smoothing parameter to converge.

In addition, Stich et al. (2017b) and Borsos et al. (2018) also consider adaptive sampling strategies. These algorithms rely on convex structure, so it is unclear if these algorithms translate to training deep learning models. In addition, Stich et al.'s procedure provides little improvement when applied to SGD, and Borsos et al.'s algorithm requires an additive smoothing hyperparameter.

2.4 SGD with oracle importance sampling

We now introduce an SGD algorithm with "oracle" importance sampling, which prioritizes examples using exact knowledge of importance values. RAIS-SGD is an approximation of this algorithm.

The oracle algorithm, which we refer to as O-SGD, makes two changes to uniform SGD, which we refer to as U-SGD. First, O-SGD samples training examples non-uniformly in a way that minimizes the variance of the stochastic gradient. This first change is not new—see (Zhao and Zhang, 2015), for example. Second, to compensate for the first improvement, O-SGD adaptively increases the learning rate. This second change, which is novel to our knowledge, can be essential for obtaining large speed-ups.

2.4.1 Progress attributable to each iteration

Given $\mathbf{w}^{(t)}$, let us define the expected convergence progress attributable to iteration t as

$$\begin{aligned}\mathbb{E}\Delta^{(t)} &= \|\mathbf{w}^{(t)} - \mathbf{w}^\star\|^2 - \mathbb{E}\left[\|\mathbf{w}^{(t+1)} - \mathbf{w}^\star\|^2\right] \\ &= 2\eta^{(t)}\langle \nabla F(\mathbf{w}^{(t)}), \mathbf{w}^{(t)} - \mathbf{w}^\star\rangle - [\eta^{(t)}]^2\mathbb{E}\left[\|\mathbf{g}^{(t)}\|^2\right] . \end{aligned} \tag{2.2}$$

Here $\mathbf{w}^\star$ denotes the solution to (P), and the expectation is with respect to minibatch $\mathcal{M}^{(t)}$. The equality follows from plugging in (2.1) and applying the fact that $\mathbf{g}^{(t)}$ is unbiased.

O-SGD improves upon uniform SGD by increasing $\mathbb{E}\Delta^{(t)}$. To do so, O-SGD decreases $\mathbb{E}[\|\mathbf{g}^{(t)}\|^2]$ and increases $\eta^{(t)}$ compared to the same quantities in U-SGD. With these changes, we expect O-SGD will train quality models in fewer iterations than U-SGD.

2.4.2 Maximizing progress with oracle importance sampling

Unlike U-SGD, O-SGD samples minibatches non-uniformly. In particular, O-SGD prioritizes training examples in order to decrease $\mathbb{E}[\|\mathbf{g}_{\mathrm{O}}^{(t)}\|^2]$. During iteration t, O-SGD defines a discrete distribution $\mathbf{p}^{(t)} \in \mathbb{R}^n_{\geq 0}$, where $\sum_i p_i^{(t)} = 1$. O-SGD constructs minibatch $\mathcal{M}^{(t)}$ by sampling independently $|\mathcal{M}|$ examples according to $\mathbf{p}^{(t)}$. The resulting stochastic gradient is

$$\mathbf{g}_{\mathrm{O}}^{(t)} = \tfrac{1}{|\mathcal{M}|}\textstyle\sum_{i\in\mathcal{M}^{(t)}} \tfrac{1}{np_i^{(t)}}\nabla f_i(\mathbf{w}^{(t)}) + \lambda\mathbf{w}^{(t)} . \tag{2.3}$$

Scaling the ∇f_i terms by $(np_i^{(t)})^{-1}$ ensures $\mathbf{g}_{\mathrm{O}}^{(t)}$ remains an unbiased approximation of $\nabla F(\mathbf{w}^{(t)})$.

O-SGD defines $\mathbf{p}^{(t)}$ as the sampling distribution that maximizes (2.2):

Proposition 2.1 (Oracle sampling distribution)**.** *In order to minimize* $\mathbb{E}[\|\mathbf{g}_{\mathrm{O}}^{(t)}\|^2]$*, O-SGD samples each example* i *with probability proportional to the* i*th "gradient norm." That is,*

$$p_i^{(t)} = \|\nabla f_i(\mathbf{w}^{(t)})\| \big/ \textstyle\sum_{j=1}^n \|\nabla f_j(\mathbf{w}^{(t)})\| .$$

Proof sketch. Defining $\bar{f}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^n f_i(\mathbf{w})$, we write this second moment as

$$\mathbb{E}\left[\|\mathbf{g}_{\mathrm{O}}^{(t)}\|^2\right] = \tfrac{1}{n^2|\mathcal{M}|}\textstyle\sum_{i=1}^n \tfrac{1}{p_i^{(t)}}\|\nabla f_i(\mathbf{w}^{(t)})\|^2 - \tfrac{1}{|\mathcal{M}|}\|\nabla \bar{f}(\mathbf{w}^{(t)})\|^2 + \|\nabla F(\mathbf{w}^{(t)})\|^2 . \tag{2.4}$$

Finding the distribution $\mathbf{p}^{(t)}$ that minimizes (2.4) is a problem with a closed-form solution. The solution is the distribution defined by Proposition 2.1, which we show in Appendix A.1. □

The oracle sampling distribution is quite intuitive. Training examples with largest gradient norm are most important for further decreasing F, and these examples receive priority. Examples that the model handles correctly have smaller gradient norm, and O-SGD deprioritizes these examples.

2.4.3 Adapting the learning rate

Prior approximate importance sampling strategies for SGD (Alain et al., 2016; Gopal, 2016; Katharopoulos and Fleuret, 2017, 2018) use a learning rate schedule identical to U-SGD's learning rate schedule. Because importance sampling reduces the gradient's variance—possibly by a large amount—we find it better to adaptively increase O-SGD's learning rate.

For O-SGD, we propose a learning rate that depends on the "gain ratio" $r_{\mathrm{O}}^{(t)} \in \mathbb{R}_{\geq 1}$:

$$r_{\mathrm{O}}^{(t)} = \mathbb{E}\left[\|\mathbf{g}_{\mathrm{U}}^{(t)}\|^2\right] \Big/ \mathbb{E}\left[\|\mathbf{g}_{\mathrm{O}}^{(t)}\|^2\right]. \tag{2.5}$$

Above, $\mathbf{g}_{\mathrm{U}}^{(t)}$ is the stochastic gradient defined by uniform sampling. O-SGD adjusts the learning rate so that according to (2.2), one iteration of O-SGD results in as much progress as $r_{\mathrm{O}}^{(t)}$ iterations of U-SGD. Defining the edge case $r_{\mathrm{O}}^{(0)} = 1$, this learning rate depends on the "effective iteration number"

$$\hat{t}_{\mathrm{O}}^{(t)} = \textstyle\sum_{t'=1}^{t} r_{\mathrm{O}}^{(t'-1)}.$$

Since the gain ratio exceeds 1, we have $\hat{t}_{\mathrm{O}}^{(t)} \geq t$ for all t. Using this effective iteration number, O-SGD defines the learning rate as

$$\eta_{\mathrm{O}}^{(t)} = r_{\mathrm{O}}^{(t)} \texttt{lr\_sched}(\hat{t}_{\mathrm{O}}^{(t)}).$$

We justify this choice of learning rate schedule with the following proposition:

Proposition 2.2 (Equivalence of gain ratio and expected speed-up)**.** *Given* $\mathbf{w}^{(t)}$*, define* $\mathbb{E}\Delta_{\mathrm{U}}^{(t)}$ *as the expected progress from iteration* t *of U-SGD with learning rate* $\eta_{\mathrm{U}}^{(t)} = \texttt{lr\_sched}\,(t)$*. For comparison, define* $\mathbb{E}\Delta_{\mathrm{O}}^{(t)}$ *as the expected progress from iteration* t *of O-SGD with learning rate* $\eta_{\mathrm{O}}^{(t)} = r_{\mathrm{O}}^{(t)}\eta_{\mathrm{U}}^{(t)}$*. Then* $\mathbb{E}\Delta_{\mathrm{O}}^{(t)} = r_{\mathrm{O}}^{(t)}\mathbb{E}\Delta_{\mathrm{U}}^{(t)}$*. That is, relative to U-SGD, O-SGD multiplies the expected progress by* $r_{\mathrm{O}}^{(t)}$*.*

Proof. Using (2.2), we have

$$\mathbb{E}\Delta_{\mathrm{U}}^{(t)} = 2\eta_{\mathrm{U}}^{(t)}\langle \nabla F(\mathbf{w}^{(t)}), \mathbf{w}^{(t)} - \mathbf{w}^{\star}\rangle - [\eta_{\mathrm{U}}^{(t)}]^2\mathbb{E}\left[\|\mathbf{g}_{\mathrm{U}}^{(t)}\|^2\right] .$$

For O-SGD, we expect progress

$$\begin{aligned}\mathbb{E}\Delta_{\mathrm{O}}^{(t)} &= 2\eta_{\mathrm{O}}^{(t)}\langle \nabla F(\mathbf{w}^{(t)}), \mathbf{w}^{(t)} - \mathbf{w}^{\star}\rangle - [\eta_{\mathrm{O}}^{(t)}]^2\mathbb{E}\left[\|\mathbf{g}_{\mathrm{O}}^{(t)}\|^2\right] \\ &= 2r_{\mathrm{O}}^{(t)}\eta_{\mathrm{U}}^{(t)}\langle \nabla F(\mathbf{w}^{(t)}), \mathbf{w}^{(t)} - \mathbf{w}^{\star}\rangle - r_{\mathrm{O}}^{(t)}[\eta_{\mathrm{U}}^{(t)}]^2\mathbb{E}\left[\|\mathbf{g}_{\mathrm{U}}^{(t)}\|^2\right] = r_{\mathrm{O}}^{(t)}\mathbb{E}\Delta_{\mathrm{U}}^{(t)} .\end{aligned}$$

□

We remark that the purpose of this learning rate adjustment is not necessarily to speed up training—whether the adjustment results in speed-up depends greatly on the original learning rate schedule. Instead, the purpose of this rescaling is to make O-SGD (and hence RAIS-SGD) suitable as a drop-in replacement for U-SGD. We confirm that this is the case in §2.6. In this section, we show empirically that RAIS-SGD and U-SGD follow similar learning curves when plotted in terms of *effective* iterations. That is, training with RAIS-SGD for t iterations equates approximately to training with U-SGD for $\hat{t}^{(t)}$ iterations (where $\hat{t}^{(t)}$ is the effective iteration number for RAIS-SGD).

2.5 Robust approximate importance sampling (RAIS)

Determining $\mathbf{p}^{(t)}$ and $r_{\mathrm{O}}^{(t)}$ in O-SGD depends on knowledge of many gradient norms ($\left\|\nabla f_i(\mathbf{w}^{(t)})\right\|$ for all examples, $\left\|\nabla \bar{f}(\mathbf{w}^{(t)})\right\|$, and $\left\|\nabla F(\mathbf{w}^{(t)})\right\|$). Computing these norms requires a time-consuming pass over all training data. To make importance sampling more practical, we propose RAIS-SGD.

2.5.1 Determining a robust sampling distribution

Like O-SGD, RAIS selects the tth minibatch by sampling indices from a discrete distribution $\mathbf{p}^{(t)}$. We denote the stochastic gradient by $\mathbf{g}_{\mathrm{R}}^{(t)}$, which takes the same form as $\mathbf{g}_{\mathrm{O}}^{(t)}$ in (2.3).

Let $v_i^* = \|\nabla f_i(\mathbf{w}^{(t)})\|$ and $\mathbf{v}^* = [v_1^*, v_2^*, \ldots, v_n^*]^T$. Since computing $\mathbf{v}^*$ is impractical, RAIS defines $\mathbf{p}^{(t)}$ by approximating these values. Naïve algorithms approximate $\mathbf{v}^*$ with a point estimate $\hat{\mathbf{v}}$. The sampling distribution becomes a multiple of $\hat{\mathbf{v}}$. Alain et al. (2016), Gopal (2016), and Katharopoulos and Fleuret (2017) all propose algorithms based on similar point estimation strategies.

The drawback of the point estimation approach is extreme sensitivity to differences between $\hat{\mathbf{v}}$ and $\mathbf{v}^*$. In fact, without preventive measures, divergence becomes *likely*. For this reason, Alain et al., Gopal, and Katharopoulos and Fleuret all incorporate additive smoothing. They introduce a hyperparameter, which we denote by δ, and sample example i with probability proportional to $\hat{v}_i + \delta$. This approach to robustness is very unconvincing, however, since performance becomes critically dependent on a user-specified parameter. A small value of δ risks divergence, while a large value greatly decreases the benefit of importance sampling (since $\mathbf{p}^{(t)}$ moves toward uniform).

RAIS takes a much different approach to make approximate importance sampling robust. Instead of a point estimate, RAIS approximates $\mathbf{v}^*$ with an uncertainty set $\mathcal{U}^{(t)} \subset \mathbb{R}^n_{\geq 0}$, which we expect contains (or nearly contains) $\mathbf{v}^*$. Given $\mathcal{U}^{(t)}$, RAIS defines $\mathbf{p}^{(t)}$ by minimizing the worst-case value of $\mathbb{E}[\|\mathbf{g}_{\mathrm{R}}^{(t)}\|^2]$ over all gradient norm possibilities in the uncertainty set. Noting that $\mathbb{E}[\|\mathbf{g}_{\mathrm{R}}^{(t)}\|^2] \propto \sum_i \frac{1}{p_i^{(t)}}(v_i^*)^2 + c$ for some $c \in \mathbb{R}$ (according to (2.4)), this means that $\mathbf{p}^{(t)}$ is the solution to the following robust optimization problem:

$$\mathbf{p}^{(t)} = \operatorname{arginf} \left\{ \max\left\{ \textstyle\sum_{i=1}^n \frac{1}{p_i} v_i^2 \mid \mathbf{v} \in \mathcal{U}^{(t)} \right\} \,\middle|\, \mathbf{p} \in \mathbb{R}^n_{>0}, \textstyle\sum_{i=1}^n p_i = 1 \right\}. \tag{PRC}$$

Such problems are common for making decisions with data uncertainty (Ben-Tal et al., 2009).

It turns out that (PRC) is straightforward to solve because the minimax theorem applies to (PRC) (we prove this in Appendix A.2.1, assuming our specific definition of $\mathcal{U}^{(t)}$ in §2.5.2).

As a result, we can solve (PRC) by first defining $\mathbf{p}^{(t)}$ to minimize (PRC)'s objective and then maximizing the result over $\mathcal{U}^{(t)}$. Given $\mathbf{v}$, we solve (PRC) by defining $p_i = v_i(\sum_{j=1}^n v_j)^{-1}$. Plugging this value back into (PRC)'s objective results in the simplified problem

$$\mathbf{v}^{(t)} = \operatorname{argmax}\left\{ \left(\sum_{i=1}^n v_i\right)^2 \mid \mathbf{v} \in \mathcal{U}^{(t)} \right\}. \tag{PRC'}$$

During each iteration t, RAIS solves (PRC'). After doing so, RAIS recovers the minimax optimal sampling distribution by defining $p_i^{(t)} \propto v_i^{(t)}$ for all training examples.

2.5.2 Modeling the uncertainty set

To define $\mathcal{U}^{(t)}$, RAIS relies on features of the algorithm's state that are predictive of the true gradient norms. For each example i, we define a feature vector $\mathbf{s}_i^{(t)} \in \mathbb{R}^{d_{\mathrm{R}}}_{>0}$. A useful feature for $\mathbf{s}_i^{(t)}$ is the gradient norm $\|\nabla f_i(\mathbf{w}^{(t')})\|$, where t' is the most recent iteration for which $i \in \mathcal{M}^{(t')}$. Since RAIS-SGD computes $\nabla f_i(\mathbf{w}^{(t')})$ during iteration t' (in order to compute $\mathbf{g}_{\mathrm{R}}^{(t')}$), constructing this feature during iteration t adds little overhead.

Given $\mathbf{s}_i^{(t)}$ for all examples, RAIS defines the uncertainty set as an axis-aligned ellipsoid. Since $\mathbf{v}^* \geq 0$, RAIS also intersects this ellipsoid with the positive orthant. RAIS parameterizes this uncertainty set with two vectors, $\mathbf{c} \in \mathbb{R}^{d_{\mathrm{R}}}_{\geq 0}$ and $\mathbf{d} \in \mathbb{R}^{d_{\mathrm{R}}}_{\geq 0}$. These vectors map features $\mathbf{s}_{1:n}^{(t)}$ to parameters of the ellipsoid. Specifically, RAIS defines the uncertainty set as

$$\mathcal{U}_{\mathbf{cd}}^{(t)} = \left\{ \mathbf{v} \in \mathbb{R}^n_{\geq 0} \;\middle|\; \tfrac{1}{n}\textstyle\sum_{i=1}^n Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, v_i) \leq 1 \right\}, \quad \text{where} \quad Q_{\mathbf{cd}}(\mathbf{s}, v) = \tfrac{(\langle \mathbf{c}, \mathbf{s}\rangle - v)^2}{\langle \mathbf{d}, \mathbf{s}\rangle}.$$

Here we denote the uncertainty set by $\mathcal{U}_{\mathbf{cd}}^{(t)}$ to emphasize the dependence of $\mathcal{U}^{(t)}$ on $\mathbf{c}$ and $\mathbf{d}$.

We include an illustration of this uncertainty set in Figure 2.2. With this definition of $\mathcal{U}_{\mathbf{cd}}^{(t)}$, (PRC') has a simple closed-form solution:

Proposition 2.3 (Solution to robust counterpart)**.** *For all i, the solution to (PRC') satisfies*

$$v_i^{(t)} = \langle \mathbf{c}, \mathbf{s}_i^{(t)} \rangle + k \langle \mathbf{d}, \mathbf{s}_i^{(t)} \rangle, \quad \textit{where} \quad k = \sqrt{n \Big/ \textstyle\sum_{j=1}^n \langle \mathbf{d}, \mathbf{s}_j^{(t)} \rangle}.$$

Proof. Since $\mathbf{v} \geq 0$, minimizing $(\sum_i v_i)^2$ is equivalent to minimizing $\sum_i v_i$. For some $\nu \geq 0$,

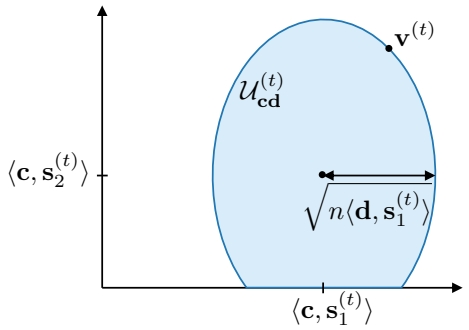


Figure 2.2: Illustration of uncertainty set. The set $\mathcal{U}_{\mathbf{cd}}^{(t)} \in \mathbb{R}^n$ is the intersection of an axis-alligned ellipsoid and the positive orthant. Each axis corresponds to a training instance. Parameters $\mathbf{c}$ map state $\mathbf{s}_{1:n}^{(t)}$ onto the ellipsoid's center. Parameters $\mathbf{d}$ control the ellipsoid's size. RAIS defines the sampling distribution as a multiple of $\mathbf{v}^{(t)}$, which maximizes $(\sum_{i=1}^n v_i)^2$ over $\mathcal{U}_{\mathbf{cd}}^{(t)}$.

the maximizer of $\sum_i v_i$ subject to $\mathbf{v} \in \mathcal{U}_{\mathbf{cd}}^{(t)}$ satisfies

$$\nabla \textstyle\sum_i v_i = \nu \nabla \frac{1}{n} \sum_i Q_{\mathbf{cd}}(v_i) \quad \text{and} \quad \nu(\frac{1}{n} \textstyle\sum_i Q_{\mathbf{cd}}(v_i) - 1) = 0\,.$$

The solution given by the proposition satisfies these KKT conditions when $\nu = \frac{n}{2k}$. □

If we consider $\langle \mathbf{c}, \mathbf{s}_i^{(t)} \rangle$ an estimate of v_i^* and $\langle \mathbf{d}, \mathbf{s}_i^{(t)} \rangle$ a measure of the uncertainty in this estimate, then Proposition 2.3 is quite interpretable. RAIS samples example i with probability proportional to $\langle \mathbf{c}, \mathbf{s}_i^{(t)} \rangle + k\langle \mathbf{d}, \mathbf{s}_i^{(t)} \rangle$. The first term accounts for our expectation of v_i^*, while the $k\langle \mathbf{d}, \mathbf{s}_i^{(t)} \rangle$ term adds robustness by compensating for uncertainty.

We note that we could consider modeling the uncertainty set using other shapes. For example, we could define $\mathcal{U}^{(t)}$ as an axis-aligned box rather than an axis-aligned ellipsoid. In this scenario, the uncertainty set takes the form

$$\mathcal{U}_{\text{box}}^{(t)} = \left\{ \mathbf{v} \in \mathbb{R}_{\geq 0}^n \mid \max_{i \in [n]} \{Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, v_i)\} \leq 1 \right\}.$$

Compared to the axis-aligned ellipsoid, this uncertainty set limits the *maximum* value of $Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, v_i)$ rather than the mean. The disadvantage of this shape is that the uncertainty set contains points with worst-case estimation error for *all* training examples. In other words, $\mathcal{U}_{\text{box}}^{(t)}$ contains points that satisfy $Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, v_i) = 1$ for all i. This is unnatural, since in practice,

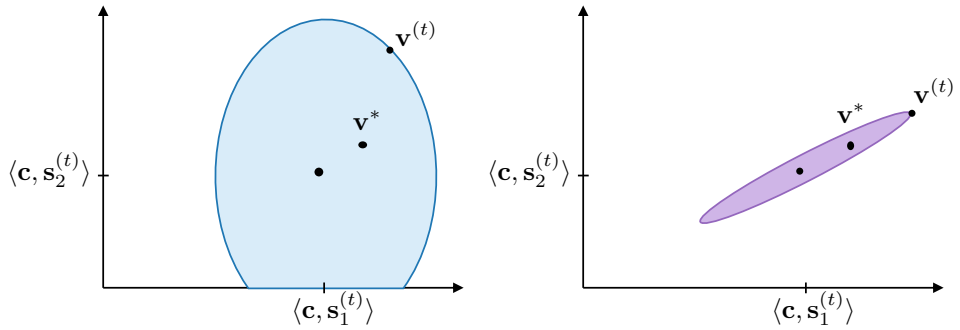


Figure 2.3: Potential benefit of uncertainty set with general ellipsoid shape. If errors in importance value estimates are strongly correlated, it may be possible to use a general ellipsoid (not axis-aligned) to improve RAIS. In the simple example above, $\mathbf{v}^{(t)}$ better approximates $\mathbf{v}^*$ in the correlated case (right), which results in greater prioritization. We consider only axis-aligned uncertainty sets in this work, since doing so greatly simplifies the procedures for solving (PRC) and training uncertainty set parameters.

the average estimation error tends to be much smaller than the worst-case error. For this reason, a box uncertainty set may be much larger than needed.

Another possibility is to model the uncertainty set using a general ellipsoid rather than an axis-aligned ellipsoid. Given $\boldsymbol{\mu} \in \mathbb{R}^n_{\geq 0}$ and positive definite $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$, we define

$$\mathcal{U}^{(t)}_{\text{general-ellipsoid}} = \left\{ \mathbf{v} \in \mathbb{R}^n_{\geq 0} \mid (\mathbf{v} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{v} - \boldsymbol{\mu}) \leq 1 \right\} .$$

This uncertainty set accounts for correlation among errors in importance value estimates. This seems sensible—for example, errors for training examples with identical class labels may be positively correlated. Modeling such correlation may lead to a smaller uncertainty set and ultimately improved prioritization. We illustrate such a scenario in Figure 2.3.

To parameterize $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ with vectors $\mathbf{c}$ and $\mathbf{d}$, we could define

$$\mu_i = \langle \mathbf{c}, \mathbf{s}_i^{(t)} \rangle \quad \text{and} \quad \boldsymbol{\Sigma} = \sum_k d_k \mathbf{S}_k^{(t)} .$$

Here each $\mathbf{S}_k^{(t)}$ is a positive semidefinite feature matrix. As an example, entry ij of $\mathbf{S}_1^{(t)}$ could take value 1 if examples i and j have the same class label and 0 otherwise. To ensure $\boldsymbol{\Sigma}$ is

positive semidefinite, we could constrain each d_k to be nonnegative.

Greater computational complexity is the biggest drawback of using a general ellipsoid uncertainty set. In contrast to the simple solution from Proposition 2.3, solving the robust counterpart with $\mathcal{U}^{(t)}_{\text{general-ellipsoid}}$ requires considering off-diagonal elements of a large $n \times n$ matrix. Depending on $\mathbf{\Sigma}$, the minimax theorem may also no longer apply, which would introduce further difficulties when solving (PRC). Depending on the choice of features for $\mathbf{\Sigma}$, there could also be greater risk of overfitting when learning $\mathbf{c}$ and $\mathbf{d}$. For these reasons, we use only axis-aligned uncertainty sets in this work. Such uncertainty sets are still effective yet much simpler.

2.5.3 Learning the uncertainty set

The uncertainty set parameters $\mathbf{c}$ and $\mathbf{d}$ greatly influence the performance of RAIS. If $\mathcal{U}^{(t)}_{\mathbf{cd}}$ is a small region near $\mathbf{v}^*$, then RAIS's sampling distribution is similar to O-SGD's sampling distribution. If $\mathcal{U}^{(t)}_{\mathbf{cd}}$ is less representative of $\mathbf{v}^*$, the variance of the stochastic gradient could become much larger.

In order to make $\mathbb{E}[\|\mathbf{g}^{(t)}_{\mathrm{R}}\|^2]$ small but still ensure $\mathbf{v}^*$ likely lies in $\mathcal{U}^{(t)}_{\mathbf{cd}}$, RAIS adaptively defines $\mathbf{c}$ and $\mathbf{d}$. To do so, RAIS minimizes the size of $\mathcal{U}^{(t)}_{\mathbf{cd}}$ subject to a constraint that encourages $\mathbf{v}^* \in \mathcal{U}^{(t)}_{\mathbf{cd}}$. Specifically, RAIS solves

$$\mathbf{c}, \mathbf{d} = \operatorname{arginf} \left\{ \textstyle\sum_{i=1}^n \langle \mathbf{d}, \mathbf{s}_i^{(t)} \rangle \;\middle|\; \mathbf{c}, \mathbf{d} \in \mathbb{R}^{d_{\mathrm{R}}}_{\geq 0}, \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \tilde{w}_i Q_{\mathbf{cd}}(\tilde{\mathbf{s}}_i, \tilde{v}_i) \leq 1 \right\} . \tag{PT}$$

Here we have defined the "size" of $\mathcal{U}^{(t)}_{\mathbf{cd}}$ as the sum of $\langle \mathbf{d}, \mathbf{s}_i^{(t)} \rangle$ values. In (PT), the right-most constraint assumes weighted training data, $(\tilde{w}_i, \tilde{\mathbf{s}}_i, \tilde{v}_i)_{i=1}^{|\mathcal{D}|}$. So that this constraint encourages $\mathbf{v}^* \in \mathcal{U}^{(t)}$, RAIS must define this training set so that

$$\tfrac{1}{|\mathcal{D}|} \textstyle\sum_{i=1}^{|\mathcal{D}|} \tilde{w}_i Q_{\mathbf{cd}}(\tilde{\mathbf{s}}_i, \tilde{v}_i) \approx \tfrac{1}{n} \sum_{i=1}^n Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, \|\nabla f_i(\mathbf{w}^{(t)})\|) .$$

That is, for any parameter vectors $\mathbf{c}$ and $\mathbf{d}$, the average value of $Q_{\mathbf{cd}}(\tilde{\mathbf{s}}, \tilde{v}_i^*)$ over the weighted training set should approximately equal the average value of $Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, v_i^*)$, which depends on

Algorithm 2.1 RAIS-SGD

input objective function $F(\cdot)$, minibatch size $|\mathcal{M}|$, learning rate schedule `lr_sched`$(\cdot)$
input RAIS training set size $|\mathcal{D}|$, exponential smoothing parameter α for gain estimate
initialize $\mathbf{w}^{(1)} \in \mathbb{R}^d$; $\hat{t}^{(1)} \leftarrow 1$; $\mathbf{c} \leftarrow \mathbf{0}$; $\mathbf{d} \leftarrow \infty \cdot \mathbf{1}$; `r_estimator` $\leftarrow$ `GainEstimator(`α`)`
for $t = 1, 2, \ldots, T$ **do**
  $\mathbf{v}^{(t)} \leftarrow \operatorname{argmax}\big\{ \left(\sum_{i=1}^n v_i\right)^2 \mid \mathbf{v} \in \mathcal{U}_{\mathbf{cd}}^{(t)} \big\}$ *# see Proposition 2.3 for closed-form solution*
  $\mathbf{p}^{(t)} \leftarrow \mathbf{v}^{(t)} / \|\mathbf{v}^{(t)}\|_1$
  $\mathcal{M}^{(t)} \leftarrow$ `sample_indices_from_distribution(`$\mathbf{p}^{(t)}$, `size` $= |\mathcal{M}|$`)`
  $\mathbf{g}_{\mathrm{R}}^{(t)} \leftarrow \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}^{(t)}} \frac{1}{n p_i^{(t)}} \nabla f_i(\mathbf{w}^{(t)}) + \lambda \mathbf{w}^{(t)}$
  `r_estimator.record_gradient_norms(`$\|\mathbf{g}_{\mathrm{R}}^{(t)}\|$, $(\|\nabla f_i(\mathbf{w}^{(t)})\|, p_i^{(t)})_{i \in \mathcal{M}^{(t)}}$`)`
  $\hat{r}^{(t)} \leftarrow$ `r_estimator.estimate_gain_ratio()` *# see §2.5.4*
  $\eta^{(t)} \leftarrow \hat{r}^{(t)} \cdot$ `lr_sched(`$\hat{t}^{(t)}$`)`
  $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta^{(t)} \mathbf{g}_{\mathrm{R}}^{(t)}$
  $\hat{t}^{(t+1)} \leftarrow \hat{t}^{(t)} + \hat{r}^{(t)}$
  if $\operatorname{mod}(t, \lceil |\mathcal{D}|/|\mathcal{M}| \rceil) = 0$ **and** $t \geq (n + |\mathcal{D}|)/|\mathcal{M}|$ **then**
    $\mathbf{c}, \mathbf{d} \leftarrow$ `train_uncertainty_model()` *# see §2.5.2*
return $\mathbf{w}^{(T+1)}$

current (unknown) gradient norms.

To achieve this, RAIS uses gradients from the most recently sampled minibatches. To define entry j of this training set, RAIS considers a pair (i, t') for which $i \in \mathcal{M}^{(t')}$ and $t' < t$. RAIS then defines $\tilde{\mathbf{s}}_j = \mathbf{s}_i^{(t')}$, $\tilde{v}_j = \|\nabla f_i(\mathbf{w}^{(t')})\|$, and $\tilde{w}_j = (n p_i^{(t')})^{-1}$. Determining these values introduces little overhead, since the algorithm computes $\nabla f_i(\mathbf{w}^{(t')})$ during iteration t' to determine $\mathbf{g}_{\mathrm{R}}^{(t')}$.

The idea behind this training set is that the average value of $Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, \|\nabla f_i(\mathbf{w}^{(t)})\|)$ over training examples tends to change gradually with t. Thus, the weighted distribution of the training set approximates the distribution of $Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, \big\|\nabla f_i(\mathbf{w}^{(t)})\big\|)$ values for the current iteration. This is because in normal training scenarios, the dynamics of SGD change slowly between iterations.

2.5.4 Approximating the gain ratio

In addition to the sampling distribution, RAIS must estimate the gain ratio to approximate O-SGD. Define $\mathbf{g}_{\mathrm{R1}}^{(t)}$ as a stochastic gradient of the form (2.3) using minibatch size 1 and RAIS sampling. Define $\mathbf{g}_{\mathrm{U1}}^{(t)}$ in the same way but with uniform sampling. From (2.4), we can deduce that the gain ratio satisfies

$$\mathbb{E}\left[\|\mathbf{g}_{\mathrm{U}}^{(t)}\|^2\right] \Big/ \mathbb{E}\left[\|\mathbf{g}_{\mathrm{R}}^{(t)}\|^2\right] = \frac{\mathbb{E}[\|\mathbf{g}_{\mathrm{R}}^{(t)}\|^2] + \frac{1}{|\mathcal{M}|}\left(\mathbb{E}[\|\mathbf{g}_{\mathrm{U1}}^{(t)}\|^2] - \mathbb{E}[\|\mathbf{g}_{\mathrm{R1}}^{(t)}\|^2]\right)}{\mathbb{E}[\|\mathbf{g}_{\mathrm{R}}^{(t)}\|^2]}\,. \tag{2.6}$$

To approximate the gain ratio, RAIS estimates the three moments on the right side of this equation. RAIS estimates $\mathbb{E}[\|\mathbf{g}_{\mathrm{R}}^{(t)}\|^2]$ using an exponential moving average of $\|\mathbf{g}_{\mathrm{R}}^{(t)}\|^2$ from recent iterations:

$$\mathbb{E}[\|\mathbf{g}_{\mathrm{R}}^{(t)}\|^2] \approx \alpha\left[\|\mathbf{g}_{\mathrm{R}}^{(t)}\|^2 + (1-\alpha)\|\mathbf{g}_{\mathrm{R}}^{(t-1)}\|^2 + (1-\alpha)^2\|\mathbf{g}_{\mathrm{R}}^{(t-2)}\|^2 + \dots\right].$$

RAIS approximates $\mathbb{E}[\|\mathbf{g}_{\mathrm{R1}}^{(t)}\|^2]$ and $\mathbb{E}[\|\mathbf{g}_{\mathrm{U1}}^{(t)}\|^2]$ in a similar way. After computing gradients for minibatch t, RAIS estimates $\mathbb{E}[\|\mathbf{g}_{\mathrm{R1}}^{(t)}\|^2]$ and $\mathbb{E}[\|\mathbf{g}_{\mathrm{U1}}^{(t)}\|^2]$ using appropriately weighted averages of $\|\nabla f_i(\mathbf{w}^{(t)})\|^2$ for each $i \in \mathcal{M}^{(t)}$ (for $\mathbb{E}[\|\mathbf{g}_{\mathrm{R1}}^{(t)}\|^2]$, RAIS weights terms by $(np_i^{(t)})^{-2}$; for $\mathbb{E}[\|\mathbf{g}_{\mathrm{U1}}^{(t)}\|^2]$, RAIS weights terms by $(np_i^{(t)})^{-1}$). Using the exponential averaging parameter α, RAIS averages these estimates from minibatch t with estimates from prior iterations.

RAIS approximates the gain ratio by plugging these moment estimates into (2.6). We denote the result by $\hat{r}^{(t)}$. Analogous to O-SGD, RAIS uses learning rate $\eta^{(t)} = \hat{r}^{(t)}\texttt{lr\_sched}\left(\hat{t}^{(t)}\right)$, where $\hat{t}^{(t)}$ is the effective iteration number $\hat{t}^{(t)} = \sum_{t'=1}^{t} \hat{r}^{(t'-1)}$.

2.5.5 Practical considerations

We have now fully described the algorithmic components of RAIS, which we summarize in Algorithm 2.1. We next consider several important practical details.

Solving (PRC') and (PT) In view of Proposition 2.3, computing $\mathbf{p}^{(t)}$ requires a small number of length n operations. This typically adds insignificant computational cost. For

small models, small $|\mathcal{M}|$, or large n, solving (PRC') introduces relatively more overhead. If necessary, it may be possible to approximate RAIS in such cases—for example by considering only a subset of training examples during each iteration. We do not consider such approximations in this work.

Learning the uncertainty set parameters requires more computation. For this reason, RAIS should not solve (PT) during every iteration. Our implementation solves (PT) asynchronously after every $\lceil |\mathcal{D}|/|\mathcal{M}| \rceil$ minibatches, with updates to $\mathbf{w}^{(t)}$ continuing during the process. Since the features $\mathbf{s}_{1:n}^{(t)}$ depend on past minibatch updates, we do not use RAIS for the first epoch of training—instead we sample examples sequentially. We also delay updates to $\mathbf{c}$ and $\mathbf{d}$ until $(n + |\mathcal{D}|)/|\mathcal{M}|$ iterations have completed (enough iterations to adequately initialize $\mathbf{s}_{1:n}^{(t)}$ as well as to collect $|\mathcal{D}|$ additional training points). We describe our algorithm for solving (PT) in Appendix A.2.2. Each iteration requires solving a nonnegative least squares problem with a $|\mathcal{D}| \times d_{\mathrm{R}}$ design matrix. We note that (PT) is convex, which guarantees $\mathbf{c}$ and $\mathbf{d}$ are globally optimal.

Compatibility with common tricks RAIS combines nicely with standard training tricks for deep learning. With no change, we find RAIS works well with momentum (Polyak, 1964; Sutskever et al., 2013). Incorporating data augmentation, dropout (Srivastava et al., 2014), or batch normalization (Ioffe and Szegedy, 2015) during training adds variance to the model's outputs and gradient norms. RAIS elegantly compensates for such inconsistency by learning a larger uncertainty set. Since the importance sampling distribution changes over time, we find it important to compute weighted batch statistics when using RAIS with batch normalization. That is, when computing normalization statistics during training, we weight contributions from each example by $(np_i^{(t)})^{-1}$.

Reducing the impact of outliers RAIS trains the uncertainty set so that the value of $\frac{1}{n}\sum_{i=1}^{n} Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, v_i^*)$ approximately equals one, where $v_i^* = \left\|\nabla f_i(\mathbf{w}^{(t)})\right\|$. In some cases—particularly problems for which the gain ratio is very large—we find that outliers dominate

this mean. That is, $Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, v_i^*) \ll 1$ for most examples, but the value is much larger for a small amount of outlier examples.

In general, we find that RAIS does not require special treatment of outliers in order to perform well. Outliers do have potential to create convergence issues, however, since the algorithm weights each $\nabla f_i(\mathbf{w}^{(t)})$ term by $(np_i^{(t)})^{-1}$ when computing $\mathbf{g}_{\mathrm{R}}^{(t)}$. If v_i^* is much larger than anticipated, the variance of $\mathbf{g}_{\mathrm{R}}^{(t)}$ could become large due to small $p_i^{(t)}$.

Gradient clipping is a natural solution to this potential issue, and the $\mathbf{c}$ and $\mathbf{d}$ parameters for RAIS provide a convenient way of specifying a gradient clipping threshold. In particular, we define an "outlier" as any example for which $Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, v_i^*)$ exceeds a threshold τ. For any outlier i, we temporarily scale f_i during iteration t (by a scalar < 1) so that $Q_{\mathbf{cd}}(\mathbf{s}_i^{(t)}, \left\|\nabla f_i(\mathbf{w}^{(t)})\right\|) = \tau$. This provides additional robustness by bounding the impact that a single example can have on the stochastic gradient. In practice, we use $\tau = 100$; we find that the number of outlier examples is often zero and rarely exceeds 0.1%.

Approximating per-example gradient norms RAIS computes $\|\nabla f_i(\mathbf{w}^{(t)})\|$ for each $i \in \mathcal{M}^{(t)}$ to train the uncertainty set. Unfortunately, existing software tools do not provide efficient access to gradient norms for individual training examples. Instead, libraries such as cuDNN are optimized for aggregating gradients over minibatches. Thus, to make RAIS practical, we must approximately compute $\|\nabla f_i(\mathbf{w}^{(t)})\|$ for the examples in each minibatch. To do so, we approximate $\|\nabla f_i(\mathbf{w}^{(t)})\|$ with the norm of only the loss layer's gradient (with respect to this layer's inputs). As shown in Figure 2.4, these values correlate strongly with the full gradient norms, since the loss layer begins the backward chain for computing $\nabla f_i(\mathbf{w}^{(t)})$.

While we find this approximation necessary to make RAIS practical, we are unaware of a fundamental factor that prohibits computing both aggregated minibatch gradients and per-example gradient norms efficiently. Perhaps future software tools will enable this flexibility.

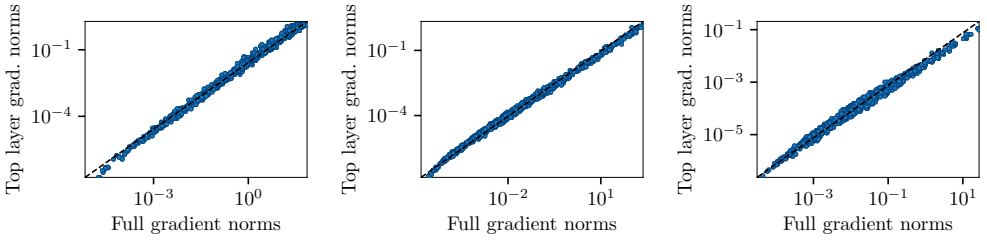


Figure 2.4: Loss layer gradient norms vs. full gradient norms. Plots compare loss layer gradient norms with $\left\|\nabla f_i(\mathbf{w}^{(t)})\right\|$ values for 1000 random training examples. From left to right, plots result from training a ResNet 18 model on the CIFAR-10 dataset for 30, 60, and 90 epochs. The top layer gradient norms correlate strongly with the full gradient norms. RAIS can use either set of values to train the uncertainty set.

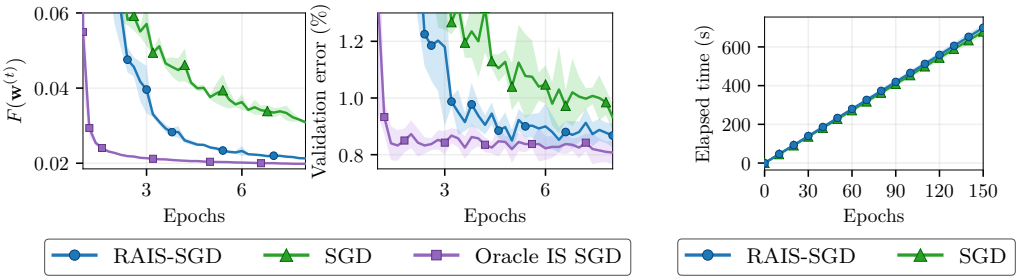


Figure 2.5: Supplemental plots. *Left:* Oracle importance sampling results for MNIST and LeNet model. *Right:* RAIS time overhead for rot-MNIST.

2.6 Empirical comparisons

In this section, we demonstrate how RAIS performs in practice. We consider the very popular task of training a convolutional neural network to classify images.

2.6.1 Small comparison with O-SGD

We first train a LeNet-5 model (Lecun et al., 1998) on the MNIST digits dataset. The model's small size makes it possible to compare with O-SGD. We use batch size 32, learning rate $\eta^{(t)} = 3.4/\sqrt{100+t}$, and L2 penalty $\lambda = 2.5 \times 10^{-4}$—the parameters are chosen so that SGD performs well. We do not use momentum or data augmentation. Figure 2.5(left) shows

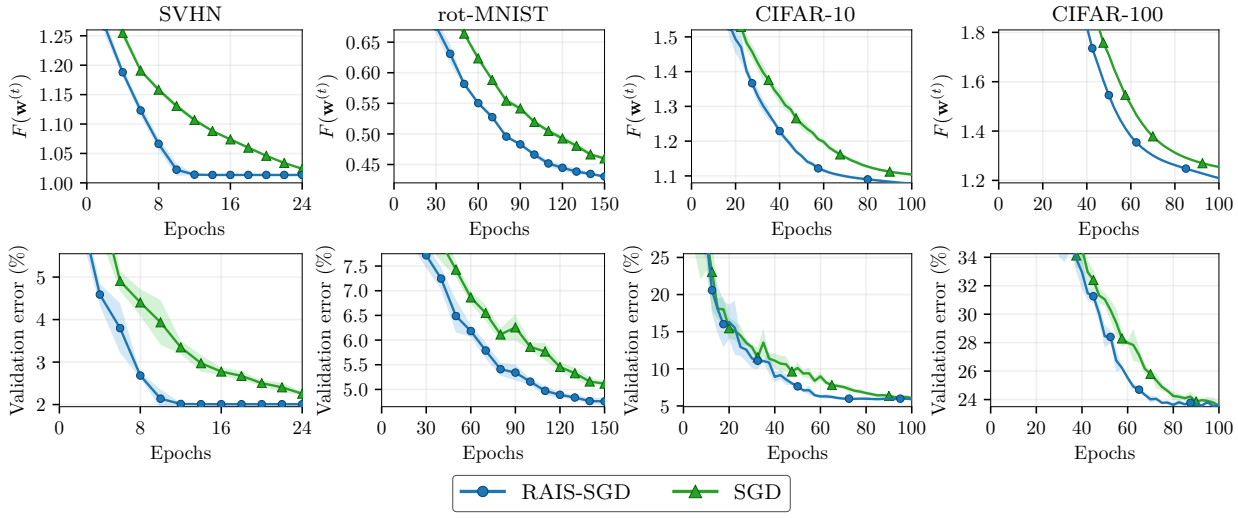


Figure 2.6: Learning curve comparison. RAIS consistently outperforms SGD with uniform sampling, both in terms of objective value and generalization performance. Curves show the mean of five trials with varying random seeds. Filled areas signify ± 1.96 times standard error of the mean.

the results of this experiment. Oracle sampling significantly outperforms RAIS, and RAIS significantly outperforms uniform sampling.

2.6.2 RAIS comparisons for larger problems

For our next comparisons, we consider street view house numbers (Netzer et al., 2011), rotated MNIST (Larochelle et al., 2007), and CIFAR tiny image (Krizhevsky, 2009) datasets. For rot-MNIST, we train a 7 layer CNN with 20 channels per layer—a strong baseline from (Cohen and Welling, 2016). Otherwise, we train an 18 layer ResNet preactivation model (He et al., 2016). CIFAR-100 contains 100 classes, while the other problems contain 10. The number of training examples is 6.0×10^5 for SVHN, 1.2×10^4 for rot-MNIST, and 5.0×10^4 for the CIFAR problems.

We follow standard training procedures to attain good generalization performance. We use batch normalization and standard momentum of 0.9. For rot-MNIST, we follow Cohen and Welling (2016), augmenting data with random rotations and training with dropout. For

the CIFAR problems, we augment the training set with random horizontal reflections and random crops (pad to 40x40 pixels; crop to 32x32).

We train the SVHN model with batch size 64 and the remaining models with $|\mathcal{M}| = 128$. For each problem, we approximately optimize λ and the learning rate schedule in order to achieve good validation performance with SGD at the end of training. The learning rate schedule decreases by a fixed fraction after each epoch ($n/|\mathcal{M}|$ iterations). This fraction is 0.8 for SVHN, 0.972 for rot-MNIST, 0.96 for CIFAR-10, and 0.96 for CIFAR-100. The initial learning rates are 0.15, 0.09, 0.08, and 0.1, respectively. We use $\lambda = 3 \times 10^{-3}$ for rot-MNIST and $\lambda = 5 \times 10^{-4}$ otherwise.

For RAIS-SGD, we use $|\mathcal{D}| = 2 \times 10^4$ training examples to learn $\mathbf{c}$ and $\mathbf{d}$ and $\alpha = 0.01$ to estimate $\hat{r}^{(t)}$. The performance of RAIS varies little with these parameters, since they only determine the number of minibatches to consider when training the uncertainty set and estimating the gain ratio. For the uncertainty set features, we use simple moving averages of the most recently computed gradient norms for each example. We use moving averages of different lengths—1, 2, 4, 8, and 16. For lengths of at least four, we also include the variance and standard deviation of these prior gradient norm values. We also incorporate a bias feature as well as the magnitude of the random crop offset.

We compare training curves of RAIS-SGD and SGD in Figure 2.6. Notice that RAIS-SGD consistently outperforms SGD. The relative speed-up ranges from approximately 20% for the CIFAR-100 problem to more than 2x for the SVHN problem. Due to varying machine loads, we plot results in terms of epochs (not wall time), but RAIS introduces very little time overhead. For example, Figure 2.5(right) includes time overhead results for the rot-MNIST comparison, which we ran on an isolated machine.

Figure 2.7 provides additional details of these results. In the figure's first row, we see the speed-up in terms of the gain ratio (the blue curve averages the value $(\hat{r}^{(t)} - 1) \cdot 100\%$ over consecutive epochs). The gain ratio tends to increase as training progresses, implying RAIS is most useful during later stages of training. We also plot the relative wall time overhead for RAIS, which again is very small.

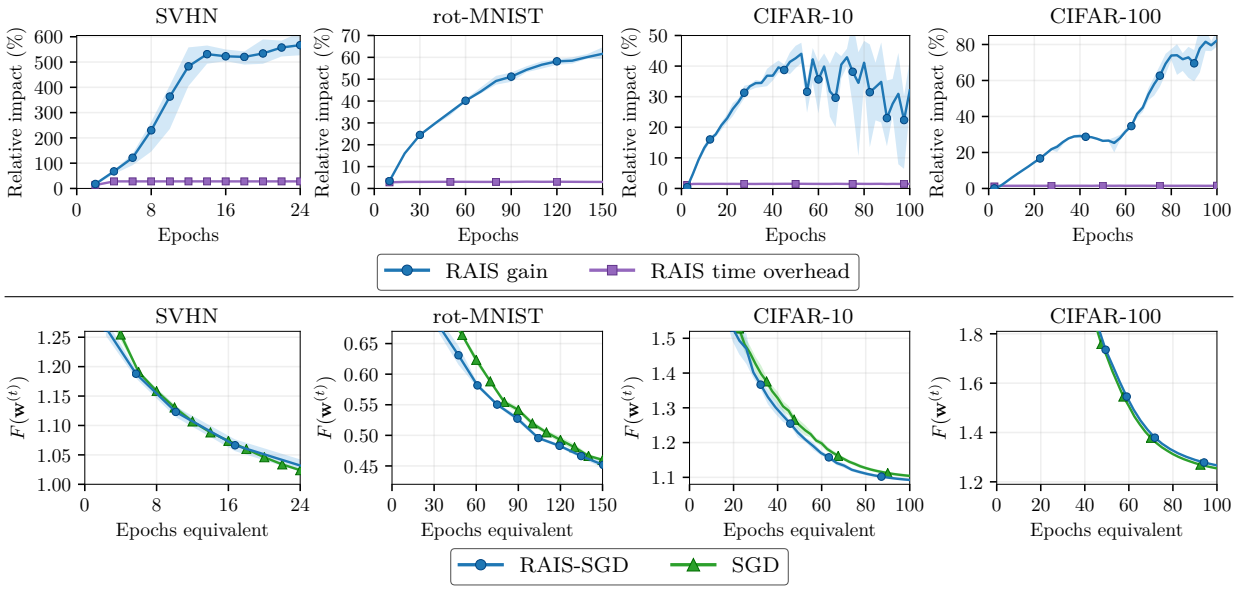


Figure 2.7: RAIS speed-up and alignment of epochs equivalent. *Above:* Blue shows increase in optimization speed due to RAIS, as measured by estimated gain ratio; purple indicates time overhead due to RAIS. Overhead is small compared to speed-up. *Below:* Objective value vs. epochs equivalent. For RAIS, epochs equivalent equals $\frac{|\mathcal{M}|}{n}\hat{t}^{(t)}$. The closely aligned curves suggest (i) RAIS-SGD is a suitable drop-in replacement for SGD, and (ii) the gain ratio correctly approximates speed-up.

In the second row of Figure 2.7, we compare RAIS-SGD and SGD in terms of *epochs equivalent*—the number of epochs measured in terms of effective iterations. Interestingly, the curves align closely. This alignment confirms that our learning rate adjustment is reasonable, as it results in a suitable drop-in replacement for SGD. This result contrasts starkly with (Alain et al., 2016), for example, in which case generalization performance differs significantly for the importance sampling and standard algorithms.

Table 2.1 concludes these comparisons with a summary of results:

Table 2.1: Quantities upon training completion.

Dataset	Algorithm	$F(\mathbf{w}^{(t)})$	Val. error	Val. loss	Epochs equivalent
SVHN	RAIS-SGD	**1.01**	**0.0201**	**0.121**	**114**
	SGD	1.02	0.0226	**0.121**	24.0
rot-MNIST	RAIS-SGD	**0.431**	**0.0476**	**0.149**	**214**
	SGD	0.460	0.0512	0.161	150.
CIFAR-10	RAIS-SGD	**1.08**	**0.0590**	**0.256**	**130.**
	SGD	1.10	0.0607	0.277	100.
CIFAR-100	RAIS-SGD	**1.21**	**0.236**	**0.962**	**138**
	SGD	1.25	**0.236**	0.989	100.

2.6.3 Importance of rescaling the learning rate

For our final comparison in this chapter, we demonstrate the importance of rescaling the learning rate when using RAIS. We train a LeNet-5 model (Lecun et al., 1998) on the MNIST digit recognition dataset. We use the learning rate schedule $\eta^{(t)} = \eta_0\sqrt{100/(100+t)}$ with regularization penalty $\lambda = 2.5 \times 10^{-4}$. We do not use momentum or data augmentation.

We compare RAIS-SGD with standard SGD as well as a control RAIS-SGD algorithm for which we do not adapt the learning rate (i.e., we define $\hat{r}^{(t)} = 1$ for all t). In the top row of Figure 2.8, we include results for $\eta_0 = 1.0$, which is a large choice of learning rate. (When setting $\eta_0 = 1.1$, for example, we found that SGD occasionally did not converge.) In the second row, we include results for $\eta_0 = 0.25$.

In the large learning rate scenario, we see that RAIS improves training times, regardless of

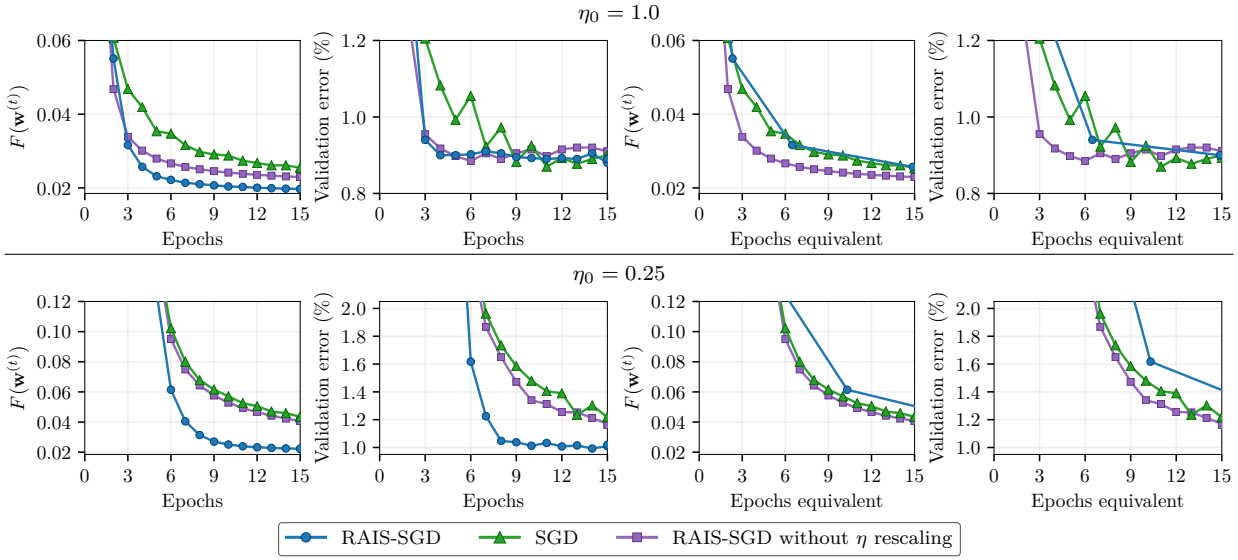


Figure 2.8: Effect of learning rate rescaling. When the original learning rate is small (below), we achieve significant improvement from importance sampling only after rescaling the learning rate—even though for this problem, the gain ratio exceeds 20 during training for both RAIS algorithms.

whether we rescale the learning rate. In the small learning rate scenario, however, adapting the learning rate is crucial for obtaining significant speed-ups.

2.7 Discussion

We proposed a relatively simple and very practical importance sampling procedure for training deep models more efficiently. By using robust optimization to define the sampling distribution, RAIS depends minimally on user-specified parameters. Additionally, RAIS introduces little computational overhead and combines nicely with standard training procedures. All together, RAIS is a promising approach with minimal downside and potential for large improvements in training speed.

Chapter 3

FAST, PRINCIPLED STRATEGIES FOR EXPLOITING PIECEWISE LINEAR STRUCTURE

In the previous chapter, we prioritized SGD by sampling challenging training examples frequently during training. We next consider a general "meta-algorithm" that, depending on the problem, prioritizes training examples (in the case of linear SVMs), parts of the model (in the case of sparse models), or constraints (in the case of constrained optimization). The concept that unifies these three cases is piecewise linear structure.

In this chapter, we propose a unique *working set algorithm* named BlitzWS. Working set algorithms reduce optimization to a sequence of smaller subproblems, which can be solved with any solver. Despite excellent performance in practice, working set implementations typically resort to heuristics to determine subproblem size, makeup, and stopping criteria.

Unlike prior working set algorithms, BlitzWS provides a useful theoretical guarantee for each subproblem. In particular, our theory relates subproblem size and stopping criteria to the amount of progress during each iteration. This result motivates strategies for optimizing algorithmic parameters and safely screening irrelevant components as BlitzWS progresses toward a solution.

Due to a novel piecewise problem formulation, BlitzWS applies to many convex problems, including training ℓ_1-regularized models and support vector machines. We showcase this versatility with empirical comparisons, which demonstrate BlitzWS is indeed a fast algorithm.

The work in this chapter is based on two of our earlier conference papers (Johnson and Guestrin, 2015, 2016). We have also incorporated many improvements into a longer paper, which is available as a preprint (Johnson and Guestrin, 2018a).

3.1 Introduction

Many optimization problems in machine learning have useful structure at their solutions. For sparse regression, the optimal model makes predictions using a fraction of available features. For support vector machines, easy-to-classify examples have no influence on the optimal model. In this chapter, we exploit such structure to train these models efficiently.

Working set algorithms prioritize optimization by reducing the problem to a sequence of simpler subproblems. Each subproblem considers only a priority subset of the problem's components—the features likely to have nonzero weight in sparse regression, for example, or training examples near the margin in SVMs. Likely the most prominent working set algorithms for machine learning are those of the LibLinear library (Fan et al., 2008), an efficient software package for training linear models. By using working set and related "shrinking" (Joachims, 1999) heuristics, LibLinear converges very quickly. Other successful applications of working sets include algorithms proposed by Osuna et al. (1997), Zanghirati and Zanni (2003), Tsochantaridis et al. (2005), Kim and Park (2008), Roth and Fischer (2008), Obozinski et al. (2009) and Friedman et al. (2010).

Despite the usefulness of working set algorithms, there is limited theoretical understanding of these methods. For LibLinear, except for guaranteed convergence to a solution, there are no guarantees with regard to working sets and shrinking. As a result, critical aspects of working set algorithms typically rely on heuristics rather than principled understanding.

We propose BlitzWS, a working set algorithm accompanied by useful theoretical analysis. Our theory explains how to prioritize components of the problem in order to guarantee a specified amount of progress during each iteration. To our knowledge, BlitzWS is the first working set algorithm with this type of guarantee. This result motivates a theoretically justified way to select each subproblem, making BlitzWS's choice of subproblem size, components, and stopping criteria more principled and robust than those of prior approaches.

BlitzWS solves instances of a novel problem formulation, which formalizes our notion of "exploiting structure" in problems such as sparse regression. Specifically, we define the

objective function as a sum of many piecewise terms. Each piecewise function is comprised of simpler subfunctions, some of which we assume to be linear. Exploiting structure amounts to selectively replacing piecewise terms in the objective with linear subfunctions. This results in a modified objective that can be much simpler to minimize. By solving a sequence of such subproblems, BlitzWS rapidly converges to the original problem's solution.

In addition to BlitzWS, we propose a closely related safe screening test called BlitzScreen. First proposed by El Ghaoui et al. (2012), safe screening identifies problem components that are guaranteed to be irrelevant to the solution. Compared to prior screening tests, BlitzScreen (i) applies to a larger class of problems, and (ii) simplifies the problem by a greater amount.

We include empirical evaluations to showcase the usefulness of BlitzWS and BlitzScreen. We find BlitzWS significantly outperforms LibLinear in many cases, especially for sparse logistic regression problems. Perhaps surprisingly, although our screening test improves on many prior tests, we find that screening often has *negligible* effect on overall convergence times. In contrast, BlitzWS improves convergence times significantly in nearly all cases.

We organize the remainder of this chapter as follows. In §3.3, we introduce BlitzWS for a simple constrained problem, emphasizing BlitzWS's main concepts. In §3.4, we introduce a piecewise problem formulation, which encompasses a larger set of problems than we consider in §3.3. In §3.5, we define BlitzWS for the general piecewise problem. In §3.6, we introduce BlitzScreen and explain its relation to BlitzWS. In §3.7, we demonstrate the usefulness of BlitzWS and BlitzScreen in practice. We discuss conclusions in §3.8.

3.2 Motivation: solving constrained problems with working sets

In this section, we describe the main idea of working set algorithms and some shortcomings of these algorithms. For now, we focus on solving a constrained optimization problem:

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^m}{\text{minimize}} \quad & \psi(\mathbf{x}) \\ \text{s.t.} \quad & \sigma_i(\mathbf{x}) \leq 0 \quad i = 1, \ldots, m\,. \end{aligned} \tag{PC}$$

We denote the solution to this problem by $\mathbf{x}^\star$, which we assume to be unique. A constraint

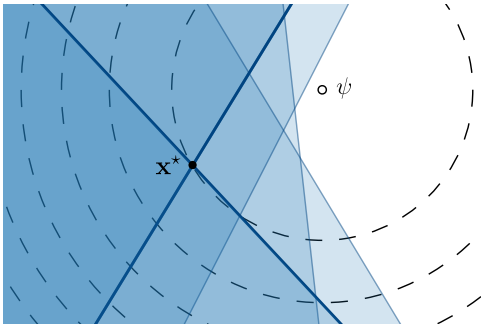


Figure 3.1: Structure in constrained optimization. This drawing depicts an instance of (PC) with solution $\mathbf{x}^\star$. Circles represent contours of the objective, ψ, while overlapping shaded areas represent the feasible regions of five linear constraints. If the problem contained only constraints that are active at $\mathbf{x}^\star$ (bold boundaries), $\mathbf{x}^\star$ would remain the solution.

i is *active* at $\mathbf{x}^\star$ if and only if $\sigma_i(\mathbf{x}^\star) = 0$. For many instances of (PC), a relatively small number of constraints are active at the solution, and the remaining constraints have no effect on $\mathbf{x}^\star$—that is, if inactive constraints were removed from (PC), the solution would not change. We illustrate this concept in Figure 3.1. By prioritizing constraints considered to be important, working set algorithms exploit this property to achieve fast convergence times.

3.2.1 A template working set algorithm for solving (PC)

Algorithm 3.1 defines a template working set algorithm for solving (PC). Each iteration t consists of two stages. First, the algorithm selects a set of priority constraints, $\mathcal{W}_t$, called the "working set." Second, the algorithm minimizes ψ subject only to constraints in $\mathcal{W}_t$, storing the result as $\mathbf{x}_t$. During iteration $t+1$, the algorithm uses $\mathbf{x}_t$ to update the working set. For an eventual iteration T, if $\mathbf{x}_T$ satisfies all m constraints, then $\mathbf{x}_T$ solves (PC).

With arbitrary choices of $\mathcal{W}_t$, Algorithm 3.1 does not necessarily converge. Even so, heuristic choices for $\mathcal{W}_t$ can both guarantee convergence to $\mathbf{x}^\star$ and result in fast convergence times in practice. Given a threshold $\tau_t \in \mathbb{R}$, a common definition of $\mathcal{W}_t$ is

$$\mathcal{W}_t = \{i \,:\, \sigma_i(\mathbf{x}_{t-1}) = 0\} \cup \{i \,:\, \sigma_i(\mathbf{x}_{t-1}) \geq \tau_t\}. \tag{3.1}$$

Algorithm 3.1 Template working set algorithm for solving (PC)

input initial $\mathbf{x}_0$ and rule for constructing working set
for $t = 1, 2, \ldots, T$ **until** converged **do**
  # Form subproblem:
  $\mathcal{W}_t \leftarrow \{i \in [m] : \texttt{include\_in\_working\_set?}(i, \mathbf{x}_{t-1})\}$
  # Solve subproblem:
  $\mathbf{x}_t \leftarrow \underset{\mathbf{x} \in \mathbb{R}^n}{\text{argmin}} \quad \psi(\mathbf{x})$
  s.t. $\quad \sigma_i(\mathbf{x}) \leq 0 \quad$ for all $i \in \mathcal{W}_t$
return $\mathbf{x}_T$

LibLinear uses a similar heuristic when solving solving sparse logistic regression problems (Yuan et al., 2012), and Bach et al. (2012) recommend a related $\mathcal{W}_t$ for sparse optimization.[1]

In (3.1), we can adjust τ_t to ensure that $\mathcal{W}_t$ contains a relatively small number of constraints. This heuristic makes some intuitive sense. During iteration t, $\mathcal{W}_t$ includes (i) all constraints that are active at the previous subproblem solution, $\mathbf{x}_{t-1}$, and (ii) the constraints most violated by $\mathbf{x}_{t-1}$. Here (i) ensures that the objective value $\psi(\mathbf{x}_t)$ is nondecreasing with t. By introducing constraints violated at $\mathbf{x}_{t-1}$ (assuming τ_t is sufficiently small), (ii) ensures that this objective value is not only nondecreasing but increasing.

By solving a sequence of smaller subproblems, Algorithm 3.1 often solves (PC) faster than algorithms that solve (PC) directly. This is especially true when m greatly exceeds the number of active constraints at $\mathbf{x}^\star$. We note Algorithm 3.1 may solve each subproblem using any solver. For this reason, working set algorithms are sometimes referred to as "meta-algorithms" (Bach et al., 2012).

3.2.2 Active set algorithms

Working set algorithms are similar to active set algorithms, and prior literature is at times inconsistent regarding the distinctions between these algorithm classes. Traditionally, active set algorithms reduce (PC) to a sequence of smaller *equality constrained* problems (Murty,

[1] To apply (3.1) to sparse problems, we can often transform the problems into (PC) instances via duality.

1988). In such algorithms, the "active set" is analogous to $\mathcal{W}_t$, but $\mathbf{x}_t$ is required to satisfy constraints in this set with equality. This equality requirement is computationally advantageous in some cases. For example, an equality constrained quadratic program can be reduced to solving a linear system (Nocedal and Wright, 1999, Chapter 16), whereas inequality constrained QPs are less straightforward to solve with precision.

For many optimization objectives, we see two primary reasons to favor working set algorithms over active set algorithms. First, with working set algorithms, we can use modern scalable algorithms, such as coordinate descent (Wright, 2015; Shi et al., 2016), to approximately solve each subproblem. Second, the quality of each subproblem solution is robust to including additional constraints in the working set. In Algorithm 3.1, including an additional constraint in $\mathcal{W}_t$ can only improve $\mathbf{x}_t$. In active set algorithms, requiring $\mathbf{x}_t$ to satisfy additional constraints with equality may instead worsen the subproblem solution's quality.

3.2.3 Shortcomings of existing working set algorithms

When implementing a working set algorithm, several design questions require attention. Heuristics like (3.1) make it difficult to answer these questions in a principled way. Developing theoretical understanding to help answer these questions is a primary goal of this chapter.

1. *What is a good way to prioritize constraints for inclusion in the working set?* With (3.1), if $\sigma_i(\mathbf{x}_{t-1}) > \sigma_j(\mathbf{x}_{t-1})$, then constraint i is prioritized over constraint j for inclusion in $\mathcal{W}_t$. While somewhat intuitive, this rule is also arbitrary. Consider, for example, the case that $\sigma_i(\mathbf{x}) = 100\sigma_j(\mathbf{x})$. Although these constraints have identical effect on (PC) ($\mathbf{x}$ satisfies constraint i iff $\mathbf{x}$ satisfies constraint j), one of the constraints receives priority over the other according to (3.1) (except if $\sigma_i(\mathbf{x}_{t-1}) = 0$). Prioritizing constraints in a principled way is a nontrivial task and the main focus of §3.3.

2. *How large should the working set be?* In Algorithm 3.1, the size of $\mathcal{W}_t$ can greatly affect convergence times. In the extreme case that $\mathcal{W}_t$ contains all m constraints, the working set algorithm provides no benefit—solving subproblem t is as difficult as solving (PC).

If $\mathcal{W}_t$ is very small, then solving subproblem t may not result in significant progress toward $\mathbf{x}^\star$. In §3.3, we quantify the trade-off between subproblem size and convergence progress by relating $|\mathcal{W}_t|$ to the amount of guaranteed progress during iteration t.

3. *How should the algorithm divide time between "form subproblem" and "solve subproblem" stages?* Since subproblem solvers are iterative, it is wasteful to compute $\mathbf{x}_t$ to machine precision in Algorithm 3.1. It would be useful if (i) convergence were guaranteed when each subproblem is solved approximately, and (ii) the algorithm could choose subproblem stopping criteria to balance the algorithm's time between "form subproblem" and "solve subproblem" stages. In §3.5, we accomplish (i) and (ii) by analyzing the effect of approximate subproblem solutions.

4. *If terminated prior to convergence, can the algorithm return a good approximate solution?* In Algorithm 3.1, each $\mathbf{x}_t$ minimizes ψ subject to a subset of constraints. Unless $\mathbf{x}_t$ solves (PC), then $\mathbf{x}_t$ violates at least one constraint in (PC). Therefore, if we terminate Algorithm 3.1 prior to convergence, returning the most recent $\mathbf{x}_t$ results in an infeasible result. In BlitzWS, there is a second iterate, $\mathbf{y}_t$, which remains feasible for all iterations and converges to the problem's solution.

3.3 BlitzWS for a simple constrained problem

In this section, we introduce BlitzWS for computing the minimum norm vector in a polytope. Given vectors $\mathbf{a}_i \in \mathbb{R}^n$ and scalars $b_i \in \mathbb{R}$ for $i = 1, 2, \ldots, m$, we solve

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^m}{\text{minimize}} \quad & \psi_{\text{MN}}(\mathbf{x}) := \tfrac{1}{2}\|\mathbf{x}\|^2 \\ \text{s.t.} \quad & \langle \mathbf{a}_i, \mathbf{x}\rangle \leq b_i \qquad\qquad i = 1, \ldots, m\,. \end{aligned} \tag{PMN}$$

We refer to the special case of BlitzWS that solves this min-norm problem as "BlitzMN." For now, we consider only this simple problem to emphasize the algorithm's concepts rather than its capability of solving a variety of problems.

(PMN) has important applications to machine learning. Most notably, ℓ_1-regularized least squares problems—the "lasso" (Tibshirani, 1996)—can be transformed into an instance of (PMN) using duality. We discuss using BlitzWS to solve the lasso more in §3.4.2.

3.3.1 Overview of BlitzMN

BlitzMN is an instance of Algorithm 3.1, our template working set algorithm. During iteration t, BlitzMN selects a working set of constraints, $\mathcal{W}_t$. BlitzMN then computes the minimizer of ψ_{MN} subject only to constraints in $\mathcal{W}_t$, which we refer to as "subproblem t":

$$\mathbf{x}_t \leftarrow \operatorname{argmin} \left\{ \tfrac{1}{2} \|\mathbf{x}\|^2 \;\middle|\; \langle \mathbf{a}_i, \mathbf{x} \rangle \leq b_i \text{ for all } i \in \mathcal{W}_t \right\} .$$

Typical working set algorithms select each $\mathcal{W}_t$ using heuristics. BlitzMN improves upon this with two main novelties. First, in addition to $\mathbf{x}_t$, BlitzMN introduces a second iterate, $\mathbf{y}_t$. This iterate is feasible (satisfies all constraints in (PMN)) for all t.

The iterate $\mathbf{y}_t$ is necessary for BlitzMN's second novelty, which is the principled choice of each working set. BlitzMN selects $\mathcal{W}_t$ in a way that guarantees quantified progress during iteration t. This amount of progress is determined by a progress parameter, $\xi_t \in (0, 1]$. For now we assume ξ_t is given, but in §3.5.9, we discuss a way to automatically select ξ_t.

If $\xi_t = 1$, then BlitzMN is guaranteed to return (PMN)'s solution upon completion of iteration t. As ξ_t decreases toward zero, BlitzMN guarantees less progress (which we quantify more precisely in §3.3.4). At a high level, BlitzMN combines two ideas to ensure this progress:

- *Including in $\mathcal{W}_t$ constraints that are active at the previous subproblem solution:* BlitzMN includes in $\mathcal{W}_t$ all i for which $\langle \mathbf{a}_i, \mathbf{x}_{t-1} \rangle = b_i$. This ensures $\psi_{\text{MN}}(\mathbf{x}_t) \geq \psi_{\text{MN}}(\mathbf{x}_{t-1})$.

- *Enforcing an equivalence region:* For subproblem t, BlitzMN defines an "equivalence region" $\mathcal{S}_\xi \subset \mathbb{R}^n$. BlitzMN selects $\mathcal{W}_t$ in a way that ensures subproblem t and (PMN) are identical within $\mathcal{S}_\xi$ (i.e., within $\mathcal{S}_\xi$, the feasible region is preserved).

  Establishing this equivalence region has two major implications. First, if $\mathbf{x}_t \in \mathcal{S}_\xi$,

then $\mathbf{x}_t$ solves (PMN). This is because subgradient values are preserved within the equivalence region. Second, if $\mathbf{x}_t$ does not equal the solution, then it must be the case that $\mathbf{x}_t \notin \mathcal{S}_\xi$. We design $\mathcal{S}_\xi$ to ensure "ξ_t progress" in this case.

3.3.2 Making working sets tractable with suboptimality gaps

We measure BlitzMN's progress during each iteration in terms of a *suboptimality gap*. Since $\mathbf{x}_t$ minimizes ψ_{MN} subject to a subset of constraints, it follows that $\psi_{\text{MN}}(\mathbf{x}_t) \leq \psi_{\text{MN}}(\mathbf{x}^\star)$, where $\mathbf{x}^\star$ solves (PMN). Given the feasible point $\mathbf{y}_t$, we can define the suboptimality gap

$$\Delta_t = \psi_{\text{MN}}(\mathbf{y}_t) - \psi_{\text{MN}}(\mathbf{x}_t) \geq \psi_{\text{MN}}(\mathbf{y}_t) - \psi_{\text{MN}}(\mathbf{x}^\star)\,.$$

Later, we analyze the improvement in Δ_t between iterations $t-1$ and t. Maximizing this improvement leads to a principled method for selecting each working set.

3.3.3 Converging from two directions: iterate $\mathbf{y}_t$ and line search

BlitzMN initializes $\mathbf{y}_0$ as a feasible point.[2] After the algorithm computes $\mathbf{x}_t$ during iteration t, BlitzMN performs a line search update to $\mathbf{y}_t$. Specifically, $\mathbf{y}_t$ is is the point on the segment $[\mathbf{y}_{t-1}, \mathbf{x}_t]$ that is closest to $\mathbf{x}_t$ while remaining feasible. Put differently, assuming that $\mathbf{x}_t$ violates at least one constraint, BlitzMN updates $\mathbf{y}_t$ so that (i) $\mathbf{y}_t$ lies on the segment $[\mathbf{y}_{t-1}, \mathbf{x}_t]$, (ii) $\mathbf{y}_t$ satisfies all m constraints, and (iii) unless $\mathbf{y}_t = \mathbf{x}_t$, there exists a "limiting constraint" i_{limit} for which $\langle \mathbf{a}_{i_{\text{limit}}}, \mathbf{x}_t \rangle > b_{i_{\text{limit}}}$ and $\langle \mathbf{a}_{i_{\text{limit}}} \mathbf{y}_t \rangle = b_{i_{\text{limit}}}$.

To perform this line search update, BlitzMN computes

$$\alpha_t = \min_{i\,:\,\langle \mathbf{a}_i, \mathbf{x}_t \rangle > b_i} \frac{b_i - \langle \mathbf{a}_i, \mathbf{y}_{t-1} \rangle}{\langle \mathbf{a}_i, \mathbf{x}_t \rangle - \langle \mathbf{a}_i, \mathbf{y}_{t-1} \rangle} \tag{3.2}$$

and subsequently defines $\mathbf{y}_t = \alpha_t \mathbf{x}_t + (1-\alpha_t)\mathbf{y}_{t-1}$. In the special case that $\langle \mathbf{a}_i, \mathbf{x}_t \rangle \leq b_i$ for all i, we define $\alpha_t = 1$ (and hence $\mathbf{y}_t = \mathbf{x}_t$). BlitzMN has converged in this case, since $\Delta_t = 0$.

---

[2] Computing a feasible $\mathbf{y}_0$ could be difficult in general, but this is not an issue for the applications we consider. When solving the lasso's dual (§3.4.2), for example, BlitzWS defines $\mathbf{y}_0 = \mathbf{0}$, which is feasible.

Because $\mathbf{x}_t$ minimizes ψ_{MN} subject to a subset of constraints, $\psi_{\text{MN}}(\mathbf{x}_t) \leq \psi_{\text{MN}}(\mathbf{y}_{t-1})$. By convexity of ψ_{MN}, this implies $\psi_{\text{MN}}(\mathbf{y}_t) \leq \psi_{\text{MN}}(\mathbf{y}_{t-1})$. Recall also from §3.3.1 that $\psi_{\text{MN}}(\mathbf{x}_t)$ is nondecreasing with t. Therefore, Δ_t is nonincreasing with t.

Next we will quantify *how much* Δ_t decreases between iterations.

3.3.4 Quantifying suboptimality gap progress during iteration t

In §3.3.1, we established that BlitzMN includes in $\mathcal{W}_t$ all constraints that are active at $\mathbf{x}_{t-1}$. We now add additional constraints to $\mathcal{W}_t$ to guarantee a specified amount of progress toward convergence. In particular, given a progress coefficient $\xi_t \in (0, 1]$, we design $\mathcal{W}_t$ such that

$$\Delta_t \leq (1 - \xi_t)\Delta_{t-1} \,. \tag{3.3}$$

Applying properties of convexity and the definition of $\mathbf{y}_t$, we derive the following bound:

Lemma 3.1. *Assume $\alpha_t > 0$, and define $\beta_t = \alpha_t(1 + \alpha_t)^{-1}$. Assume that $\mathcal{W}_t$ includes all constraints that are active at $\mathbf{x}_{t-1}$. Then we have*

$$\Delta_t \leq \tfrac{1-2\beta_t}{1-\beta_t} \left[\Delta_{t-1} - \tfrac{1-\beta_t}{\beta_t^2} \tfrac{1}{2} \left\| \mathbf{y}_t - \beta_t \mathbf{x}_{t-1} - (1 - \beta_t)\mathbf{y}_{t-1} \right\|^2 - \beta_t \tfrac{1}{2} \left\| \mathbf{x}_{t-1} - \mathbf{y}_{t-1} \right\|^2 \right] . \tag{3.4}$$

We prove this result in Appendix B.1. Since the special case that $\alpha_t = 0$ is unnecessary for understanding the algorithm's concepts, we ignore this case except in later proofs.

The working set affects only two variables in (3.4): β_t and $\mathbf{y}_t$. The other variables—$\mathbf{x}_{t-1}$, $\mathbf{y}_{t-1}$, and Δ_{t-1}—are given when selecting $\mathcal{W}_t$. By understanding how $\mathcal{W}_t$ affects β_t and $\mathbf{y}_t$, we can choose $\mathcal{W}_t$ so that the right side of (3.4) is upper bounded by $(1 - \xi_t)\Delta_{t-1}$.

Assume for a moment that β_t is given when BlitzMN defines $\mathcal{W}_t$. In this scenario, it is straightforward to select $\mathcal{W}_t$ in a way that guarantees (3.3). For the special case that $\beta_t = 1/2$, (3.4) simplifies to $\Delta_t \leq 0$—(3.3) holds, regardless of $\mathcal{W}_t$. If $\beta_t < 1/2$, we have $\alpha_t < 1$. Applying the definition of α_t in (3.2), there exists a constraint i_{limit} such that

$$\langle \mathbf{a}_{i_{\text{limit}}}, \mathbf{x}_t \rangle > b_{i_{\text{limit}}} \qquad \text{and} \qquad \langle \mathbf{a}_{i_{\text{limit}}}, \mathbf{y}_t \rangle = b_{i_{\text{limit}}} \,. \tag{3.5}$$

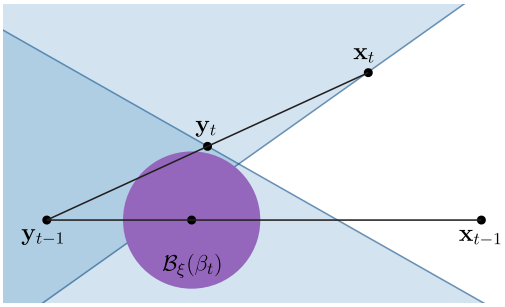


Figure 3.2: BlitzMN geometry assuming knowledge of β_t. Shaded areas represent the feasible regions of two linear constraints. Assume $\beta_t = \alpha_t(1+\alpha_t)^{-1}$ is known when choosing $\mathcal{W}_t$. Shown in purple, $\mathcal{B}_\xi(\beta_t)$ is a ball with center $\beta_t\mathbf{x}_{t-1} + (1-\beta_t)\mathbf{y}_{t-1}$ and radius $\tau_\xi(\beta_t)$. The bound (3.3) depends on $\mathbf{y}_t$'s distance from this ball's center. BlitzMN ensures this distance is large by choosing $\mathcal{W}_t$ so that if $\mathbf{y}_t \neq \mathbf{x}^\star$, then $\mathbf{y}_t \notin \mathcal{B}_\xi(\beta_t)$. In particular, BlitzMN selects $\mathcal{W}_t$ in a way that preverses (PMN)'s feasible region within $\mathcal{B}_\xi(\beta_t)$.

Since $\mathbf{x}_t$ violates constraint i_{limit}, we have $i_{\text{limit}} \notin \mathcal{W}_t$. To apply this fact, note $\mathbf{y}_t$ appears in Lemma 3.1 through the quantity $\|\mathbf{y}_t - \beta_t\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\|$. BlitzMN chooses $\mathcal{W}_t$ to ensure this norm equals a threshold $\tau_\xi(\beta_t)$ at minimum. To achieve this, we define the "equivalence ball"

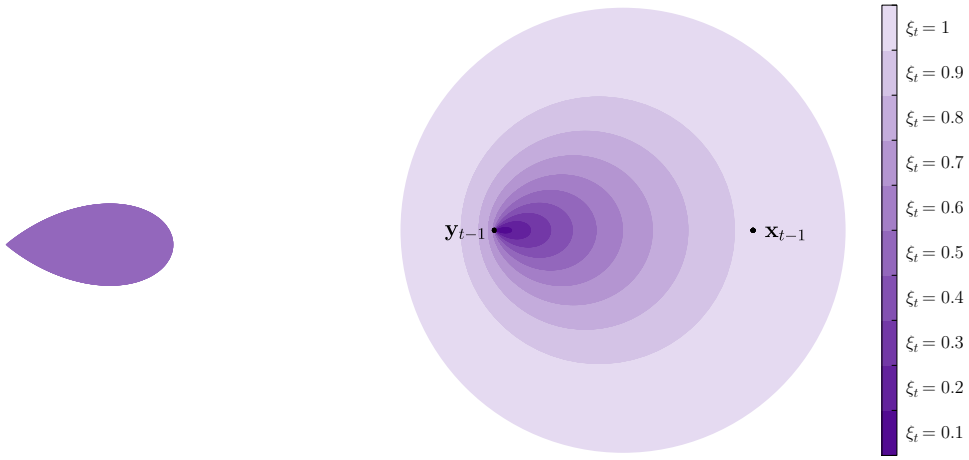
$$\mathcal{B}_\xi(\beta_t) = \{\mathbf{x} \mid \|\mathbf{x} - \beta_t\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\| < \tau_\xi(\beta_t)\}\,. \tag{3.6}$$

BlitzMN includes i in $\mathcal{W}_t$ if there exists an $\mathbf{x} \in \mathcal{B}_\xi(\beta_t)$ such that $\langle \mathbf{a}_i, \mathbf{x}\rangle \geq b_i$. This preserves (PMN)'s feasible region within $\mathcal{B}_\xi(\beta_t)$. Since $i_{\text{limit}} \notin \mathcal{W}_t$, this implies that no point on the boundary of constraint i_{limit}—$\mathbf{y}_t$ included, due to (3.5)—lies within $\mathcal{B}_\xi(\beta_t)$. By our definition of $\mathcal{B}_\xi(\beta_t)$ in (3.6), this guarantees that $\|\mathbf{y}_t - \beta_t\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\| \geq \tau_\xi(\beta_t)$. We illustrate this concept in Figure 3.2.

Having linked $\tau_\xi(\beta_t)$ to Lemma 3.1, we can define $\tau_\xi(\beta_t)$ to produce our desired bound, (3.3):

$$\tau_\xi(\beta_t) = \beta_t\sqrt{2\Delta_{t-1}}\left[1 + \tfrac{\beta_t}{1-\beta_t}\left(1 - \tfrac{\|\mathbf{x}_{t-1}-\mathbf{y}_{t-1}\|^2}{2\Delta_{t-1}}\right) - \tfrac{1-\xi_t}{1-2\beta_t}\right]_+^{1/2}. \tag{3.7}$$

This leads to the following result:



$\mathcal{S}_\xi$ when $\xi_t = 0.2$ (enlarged) $\mathcal{S}_\xi$ for many values of ξ_t

Figure 3.3: Geometry of equivalence regions. As ξ_t increases, the size of $\mathcal{S}_\xi$ increases, the number of constraints in $\mathcal{W}_t$ increases, and the amount of guaranteed progress increases. For small ξ_t, $\mathcal{S}_\xi$ has a "teardrop" shape. When $\xi_t = 1$, $\mathcal{S}_\xi$ is a ball with center $\frac{1}{2}(\mathbf{x}_{t-1} + \mathbf{y}_{t-1})$. To generate the figure, we let $\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\|^2 / \Delta_{t-1} = 1$.

Lemma 3.2. *Assume β_t is known when selecting $\mathcal{W}_t$, and assume $\beta_t > 0$. For all $i \in [m]$, let $\mathcal{W}_t$ include constraint i if either $\{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \geq b_i\} \cap \mathcal{B}_\xi(\beta_t) \neq \emptyset$ or $\langle \mathbf{a}_i, \mathbf{x}_{t-1} \rangle = b_i$. Then*

$$\Delta_t \leq (1 - \xi_t)\Delta_{t-1} \,.$$

We prove Lemma 3.2 in Appendix B.2. On its own, this lemma is impractical, as it assumes knowledge of β_t when choosing $\mathcal{W}_t$. Since β_t is unknown BlitzMN considers *all possible* β when selecting $\mathcal{W}_t$. To do so, we define the equivalence region

$$\mathcal{S}_\xi = \bigcup_{\beta \in (0, 1/2)} \mathcal{B}_\xi(\beta) \,.$$

This new equivalence region leads to the following result, which we prove in Appendix B.3:

Figure 3.4: Capsule approximation. To simplify computation, BlitzMN constructs $\mathcal{W}_t$ using $\mathcal{S}_\xi^{\text{cap}}$, which is a relaxation of $\mathcal{S}_\xi$. $\mathcal{S}_\xi$ is shaped like a teardrop when ξ_t is small, while $\mathcal{S}_\xi^{\text{cap}}$ is the smallest capsule (convex hull of two balls with equal radius) that contains $\mathcal{S}_\xi$. For the figure, $\xi_t = 0.4$, and $\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\|^2 / \Delta_{t-1} = 1$.

Theorem 3.3 (Guaranteed progress during iteration t of BlitzMN)**.** *During iteration t of BlitzMN, consider any progress coefficient $\xi_t \in (0, 1]$. For all $i \in [m]$, let $\mathcal{W}_t$ include constraint i if either $\mathcal{S}_\xi \cap \{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \geq b_i\} \neq \emptyset$ or $\langle \mathbf{a}_i, \mathbf{x}_{t-1} \rangle = b_i$. Then*

$$\Delta_t \leq (1 - \xi_t)\Delta_{t-1} \,.$$

3.3.5 Computing $\mathcal{W}_t$ efficiently with capsule approximations

Figure 3.3 contains renderings of $\mathcal{S}_\xi$. Note $\mathcal{S}_\xi$ grows in size as ξ_t increases, since $\tau_\xi(\beta)$ also increases with ξ_t due to (3.7). Unfortunately, using $\mathcal{S}_\xi$ to select $\mathcal{W}_t$ is problematic in practice, since testing if $\mathcal{S}_\xi \cap \{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \geq b_i\} \neq \emptyset$ is not simple. To reduce computation, BlitzMN constructs $\mathcal{W}_t$ using a relaxed region, $\mathcal{S}_\xi^{\text{cap}}$. This set is the convex hull of two balls:

$$\mathcal{S}_\xi^{\text{cap}} = \text{conv}\left(\mathcal{B}_1^{\text{cap}} \cup \mathcal{B}_2^{\text{cap}}\right) \,, \quad \text{where}$$

$$\mathcal{B}_1^{\text{cap}} = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{c}_1^{\text{cap}}\| < r^{\text{cap}}\} \,, \qquad \mathcal{B}_2^{\text{cap}} = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{c}_2^{\text{cap}}\| < r^{\text{cap}}\} \,,$$

$$\mathbf{c}_1^{\text{cap}} = \mathbf{y}_{t-1} + \tfrac{\mathbf{x}_{t-1} - \mathbf{y}_{t-1}}{\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\|}(d_{\text{min}}^{\text{cap}} + r^{\text{cap}}) \,, \qquad \mathbf{c}_2^{\text{cap}} = \mathbf{y}_{t-1} + \tfrac{\mathbf{x}_{t-1} - \mathbf{y}_{t-1}}{\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\|}(d_{\text{max}}^{\text{cap}} - r^{\text{cap}}) \,,$$

$$d_{\text{min}}^{\text{cap}} = \inf_{\beta \,:\, \tau_\xi(\beta) > 0} \beta \|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\| - \tau_\xi(\beta) \,, \qquad d_{\text{max}}^{\text{cap}} = \sup_{\beta \,:\, \tau_\xi(\beta) > 0} \beta \|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\| + \tau_\xi(\beta) \,,$$

$$\text{and} \quad r^{\text{cap}} = \sup_{\beta \,:\, \tau_\xi(\beta) > 0} \tau_\xi(\beta) \,.$$

In Appendix B.9.1, we prove that $\mathcal{S}_\xi \subseteq \mathcal{S}_\xi^{\text{cap}}$. Illustrated in Figure 3.4, $\mathcal{S}_\xi^{\text{cap}}$ is the smallest "capsule" (set of points within a fixed distance from a line segment) for which $\mathcal{S}_\xi \subseteq \mathcal{S}_\xi^{\text{cap}}$. We define the capsule using three scalars: $d_{\min}^{\text{cap}}$, $d_{\max}^{\text{cap}}$, and r^{cap} (in addition to the points $\mathbf{x}_{t-1}$ and $\mathbf{y}_{t-1}$). The radius of the capsule is r^{cap}, while $d_{\min}^{\text{cap}}$ and $d_{\max}^{\text{cap}}$ parameterize the capsule's endpoints. Determining these three parameters requires solving three 1-D optimization problems, which we can solve efficiently:

Theorem 3.4 (Computing capsule parameters is quasiconcave)**.** *For each $s \in \{-1, 0, +1\}$, the function $q_s(\beta) = s\beta \left\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\right\| + \tau_\xi(\beta)$ is quasiconcave over $\{\beta \mid q_s(\beta) > 0\}$. Thus, $d_{\min}^{\text{cap}}$, $d_{\max}^{\text{cap}}$, and r^{cap} are suprema of 1-D quasiconcave functions.*

We prove Theorem 3.4 in Appendix B.4. Making use of this result, BlitzML computes $d_{\min}^{\text{cap}}$, $d_{\max}^{\text{cap}}$, and r^{cap} using the bisection method. Empirically we find the computational cost of computing $\mathcal{S}_\xi^{\text{cap}}$ is negligible compared to the cost of solving each subproblem (unless (P) is very small).

After computing $\mathcal{S}_\xi^{\text{cap}}$, it is simple to test whether $\mathcal{S}_\xi^{\text{cap}} \cap \{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \geq b_i\} \neq \emptyset$. The condition is true iff $\mathcal{B}_1^{\text{cap}}$ or $\mathcal{B}_2^{\text{cap}}$ intersect $\{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \geq b_i\}$. It follows that $\mathcal{S}_\xi^{\text{cap}} \cap \{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \geq b_i\} \neq \emptyset$ iff

$$b_i - \max\left\{\langle \mathbf{a}_i, \mathbf{c}_1^{\text{cap}} \rangle, \langle \mathbf{a}_i, \mathbf{c}_2^{\text{cap}} \rangle\right\} < \|\mathbf{a}_i\| r^{\text{cap}}\,.$$

BlitzMN includes in the working set any i for which either the above inequality is satisfied or $\langle \mathbf{a}_i, \mathbf{x}_{t-1} \rangle = b_i$. Since $\mathcal{S}_\xi \subseteq \mathcal{S}_\xi^{\text{cap}}$, this $\mathcal{W}_t$ satisfies the conditions for Theorem 3.3, ensuring the suboptimality gap decreases by at least a $1 - \xi_t$ factor during iteration t.

3.3.6 BlitzMN definition and convergence guarantee

We formally define BlitzMN in Algorithm 3.2. BlitzMN assumes an initial feasible point $\mathbf{y}_0$ and initializes $\mathbf{x}_0$ as the zero vector ("subproblem 0" is implicitly defined as minimizing ψ_{MN} subject to no constraints). The suboptimality gap decreases with the following guarantee:

Algorithm 3.2 BlitzMN for solving (PMN)

input feasible point $\mathbf{y}_0$ and method for choosing ξ_t for all t
initialize $\mathbf{x}_0 \leftarrow \mathbf{0}$
for $t = 1, \ldots, T$ **until** $\mathbf{x}_T = \mathbf{y}_T$ **do**
  Choose progress coefficient $\xi_t \in (0, 1]$
  $\mathcal{S}_\xi^{\text{cap}} \leftarrow$ `compute_capsule_region`$(\xi_t, \mathbf{x}_{t-1}, \mathbf{y}_{t-1})$ *# see §3.3.5*
  $\mathcal{W}_t \leftarrow \{i \in [m] \mid$ `include_in_working_set`$?(i, \mathcal{S}_\xi^{\text{cap}}, \mathbf{x}_{t-1})\}$
  $\mathbf{x}_t \leftarrow \operatorname{argmin} \left\{ \frac{1}{2} \|\mathbf{x}\|^2 \;\middle|\; \langle \mathbf{a}_i, \mathbf{x} \rangle \leq b_i \text{ for all } i \in \mathcal{W}_t \right\}$
  $\mathbf{y}_t \leftarrow$ `compute_extreme_feasible_point`$(\mathbf{x}_t, \mathbf{y}_{t-1})$
return $\mathbf{y}_T$

function `include_in_working_set`$?(i, \mathcal{S}_\xi^{\text{cap}}, \mathbf{x}_{t-1})$
  $\mathbf{c}_1^{\text{cap}}, \mathbf{c}_2^{\text{cap}}, r^{\text{cap}} \leftarrow$ `get_capsule_centers_and_radius`$(\mathcal{S}_\xi^{\text{cap}})$ *# see §3.3.5*
  if $(b_i - \max\{\langle \mathbf{a}_i, \mathbf{c}_1^{\text{cap}} \rangle, \langle \mathbf{a}_i, \mathbf{c}_2^{\text{cap}} \rangle\} < \|\mathbf{a}_i\| \, r^{\text{cap}})$ **or** $(\langle \mathbf{a}_i, \mathbf{x}_{t-1} \rangle = b_i)$ **then**
    return true
  return false

function `compute_extreme_feasible_point`$(\mathbf{x}_t, \mathbf{y}_{t-1})$
  $\alpha_t \leftarrow 1$
  for $i \in [m]$ **do**
    if $(\langle \mathbf{a}_i, \mathbf{x}_t \rangle > b_i)$ **then**
      $\alpha_t \leftarrow \min \left\{ \alpha_t, \frac{b_i - \langle \mathbf{a}_i, \mathbf{y}_{t-1} \rangle}{\langle \mathbf{a}_i, \mathbf{x}_t \rangle - \langle \mathbf{a}_i, \mathbf{y}_{t-1} \rangle} \right\}$
  return $\alpha_t \mathbf{x}_t + (1 - \alpha_t) \mathbf{y}_{t-1}$

Theorem 3.5 (Convergence bound for BlitzMN)**.** *For any iteration T of Algorithm 3.2, define the suboptimality gap $\Delta_T = \psi_{\text{MN}}(\mathbf{y}_T) - \psi_{\text{MN}}(\mathbf{x}_T)$. For all $T > 0$, we have*

$$\Delta_T \leq \Delta_0 \prod_{t=1}^{T} (1 - \xi_t) \,.$$

We have yet to address several practical considerations for BlitzMN. This includes analysis of approximate subproblem solutions and a procedure for selecting ξ_t during each iteration. We address these details in §3.5 in the context of our more general working set algorithm, BlitzWS. Before that, we define a more general problem formulation.

3.4 Piecewise linear structure in convex problems

Rather than exploiting irrelevant constraints, BlitzWS exploits piecewise linear structure. We now reformulate the objective function to accommodate this more general concept.

3.4.1 Piecewise problem formulation

For the remainder of this chapter, we consider convex optimization problems of the form

$$\underset{\mathbf{x}\in\mathbb{R}^m}{\text{minimize}}\ f(\mathbf{x}) := \psi(\mathbf{x}) + \sum_{i=1}^{m} \phi_i(\mathbf{x})\,. \tag{P}$$

We assume each ϕ_i is *piecewise*. For each ϕ_i, we assume a domain-partitioning function $\pi_i : \mathbb{R}^m \to \{1, 2, \ldots, p_i\}$ and corresponding subfunctions $\phi_i^{(1)}, \phi_i^{(2)}, \ldots, \phi_i^{(p_i)}$ such that

$$\phi_i(\mathbf{x}) = \begin{cases} \phi_i^{(1)}(\mathbf{x}) & \text{if } \pi_i(\mathbf{x}) = 1, \\ \quad\vdots & \\ \phi_i^{(p_i)}(\mathbf{x}) & \text{if } \pi_i(\mathbf{x}) = p_i\,. \end{cases}$$

We assume ψ, ϕ_i, and $\phi_i^{(k)}$ for all i and k are convex lower semicontinuous. We also assume ψ is 1-strongly convex.[3] Let $\mathcal{X}_i^{(k)}$ denote the kth subdomain of ϕ_i: $\mathcal{X}_i^{(k)} = \{\mathbf{x} \mid \pi_i(\mathbf{x}) = k\}$. Denoting (P)'s solution by $\mathbf{x}^\star$, we use $\mathcal{X}_i^{(\star)}$ to denote the subdomain of ϕ_i that contains $\mathbf{x}^\star$.

We focus on instances of (P) for which the piecewise functions are the primary obstacle to efficient optimization (generally problems with large m). We also focus on problems for which many $\phi_i^{(k)}$ subfunctions are linear. We base our methods on the following proposition:

Proposition 3.6 (Exploiting piecewise structure at $\mathbf{x}^\star$)**.** *For each $i \in [m]$, assume knowledge of $\pi_i(\mathbf{x}^\star)$ and whether $\mathbf{x}^\star \in \mathrm{bd}(\mathcal{X}_i^{(\star)})$. Define $\phi_i^\star = \phi_i$ if $\mathbf{x}^\star \in \mathrm{bd}(\mathcal{X}_i^{(\star)})$ and $\phi_i^\star = \phi_i^{(\pi_i(\mathbf{x}^\star))}$ otherwise. Then $\mathbf{x}^\star$ is also the solution to*

$$\underset{\mathbf{x}\in\mathbb{R}^m}{\text{minimize}}\ f^\star(\mathbf{x}) := \psi(\mathbf{x}) + \textstyle\sum_{i=1}^{m} \phi_i^\star(\mathbf{x})\,. \tag{P*}$$

[3] We can adapt this formulation to the case that ψ is γ-strongly convex for any $\gamma > 0$ by scaling f by γ^{-1}.

Item	Description	Example	
ψ	Strongly convex term in objective	ℓ_2 regularization	$\frac{1}{2}\|\mathbf{x}\|^2$
ϕ_i	ith piecewise term in objective	Hinge loss	$(1 - b_i \langle \mathbf{a}_i, \mathbf{x} \rangle)_+$
$\phi_i^{(k)}$	Subfunction k of ith piecewise term	Constant term	0
$\mathcal{X}_i^{(k)}$	Subdomain k of ith piecewise term	Half-space	$\{\mathbf{x} : 1 - b_i \langle \mathbf{a}_i, \mathbf{x} \rangle \leq 0\}$
π_i	Domain-partitioning function for ϕ_i	$\pi_i(\mathbf{x}) = \begin{cases} 1 & \text{if } 1 - b_i \langle \mathbf{a}_i, \mathbf{x} \rangle \leq 0, \\ 2 & \text{otherwise} \end{cases}$	
$\mathcal{X}_i^{(\star)}$	Subdomain of ϕ_i that contains $\mathbf{x}^\star$	$\mathcal{X}_i^{(\star)} = \mathcal{X}_i^{(1)}$ if $\pi_i(\mathbf{x}^\star) = 1$	

Table 3.1: Summary of piecewise problem formulation. BlitzWS minimizes objectives of the form $f(\mathbf{x}) = \psi(\mathbf{x}) + \sum_{i=1}^m \phi_i(\mathbf{x})$.

Proposition 3.6 states that if f's minimizer does not lie on the boundary of $\mathcal{X}_i^{(\star)}$, then replacing ϕ_i with the subfunction $\phi_i^{(\pi_i(\mathbf{x}^\star))}$ in f does not change the objective's minimizer. We can verify this by observing that $f^\star$ preserves the subgradient of f at $\mathbf{x}^\star$, so $\mathbf{0} \in \partial f^\star(\mathbf{x}^\star)$.

Despite matching solutions, solving (P*) can require *much less* computation than solving (P). This is especially true when many $\phi_i^\star$ are linear subfunctions. In this case, the linear subfunctions collapse into a single linear term, making $f^\star$ simpler to minimize than f.

3.4.2 Piecewise linear structure in machine learning

We now describe several instances of (P) that are importance to machine learning.

Inactive constraints in constrained optimization

We first consider constrained optimization (for which (PMN) is a special case):

$$\begin{aligned} \underset{\mathbf{x} \in \mathbb{R}^m}{\text{minimize}} \quad & \psi(\mathbf{x}) \\ \text{s.t.} \quad & \sigma_i(\mathbf{x}) \leq 0 \quad i = 1, \ldots, m\,. \end{aligned} \tag{PC}$$

If each σ_i is convex and ψ is 1-strongly convex, this problem can be transformed into an instance of (P) using implicit constraints. For each $i \in [m]$, define ϕ_i as

$$\phi_i(\mathbf{x}) = \begin{cases} 0 & \text{if } \sigma_i(\mathbf{x}) \leq 0, \\ +\infty & \text{otherwise.} \end{cases}$$

There are two subdomains for each ϕ_i—the constraint's feasible and infeasible regions. Since $\mathbf{x}^\star$ must satisfy all constraints, note $\mathcal{X}_i^{(\star)}$ represents constraint i's feasible region.

Let us consider Proposition 3.6 in the context of (PC). Define $\mathcal{W}^\star = \{i \mid \sigma_i(\mathbf{x}^\star) = 0\}$, the set of constraints that are active at (PC)'s solution. The condition $\mathbf{x}^\star \in \operatorname{bd}(\mathcal{X}_i^{(\star)})$ implies $\sigma_i(\mathbf{x}^\star) = 0$ and $i \in \mathcal{W}^*$. To define each $\phi_i^\star$, we let $\phi_i^\star = \phi_i$ for all $i \in \mathcal{W}^\star$ and $\phi_i^\star = 0$ otherwise. Applying Proposition 3.6, we see that $\mathbf{x}^\star$ also solves

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^m}{\text{minimize}} & \psi(\mathbf{x}) \\ \text{s.t.} & \sigma_i(\mathbf{x}) \leq 0 \qquad \text{for } i \in \mathcal{W}^\star\,. \end{array} \tag{PC*}$$

That is, we have reduced (PC) to a problem with only $|\mathcal{W}^\star|$ constraints. Since often $|\mathcal{W}^\star| \ll m$, solving (PC*) can be significantly simpler than solving the original problem.

Zero-valued weights in sparse optimization

Optimization with sparsity-inducing penalties is popular in machine learning—see Bach et al. (2012) for a survey. Here we consider learning ℓ_1-regularized linear models.

Let $((\mathbf{a}_j, b_j))_{j=1}^n$ be a collection of n training examples where $\mathbf{a}_j \in \mathbb{R}^m$ is a feature vector and $b_j \in \mathcal{B}$ is a corresponding label. Typically $\mathcal{B} = \{-1, +1\}$ for classification problems, while $\mathcal{B} = \mathbb{R}$ for regression. We can fit parameters of a linear model to this data by solving

$$\underset{\boldsymbol{\omega}\in\mathbb{R}^m}{\text{minimize}} \ \sum_{j=1}^n L_j(\langle \mathbf{a}_j, \boldsymbol{\omega}\rangle) + \lambda \left\|\boldsymbol{\omega}\right\|_1 \,. \tag{PL1}$$

Above $\lambda > 0$ is a tuning parameter, and L_j is a loss function (parameterized by b_j). When λ is sufficiently large, a solution $\boldsymbol{\omega}^\star$ is *sparse*, meaning most entries of $\boldsymbol{\omega}^\star$ equal 0.

(PL1) is not directly an instance of (P), since this problem is not 1-strongly convex in

Loss	$L_j(\langle \mathbf{a}_j, \boldsymbol{\omega} \rangle)$	$L_j^*(x_j)$
Logistic	$4 \log(1 + \exp(-b_j \langle \mathbf{a}_j, \boldsymbol{\omega} \rangle))$	$-\frac{1}{4}\frac{x_j}{b_j}\log(-\frac{x_j}{b_j}) + \frac{1}{4}(1 + \frac{x_j}{b_j})\log(1 + \frac{x_j}{b_j})$
Squared hinge	$\begin{cases} \frac{1}{2}(1 - b_j \langle \mathbf{a}_j, \boldsymbol{\omega} \rangle)^2 & \text{if } b_j \langle \mathbf{a}_j, \boldsymbol{\omega} \rangle \leq 1, \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} \frac{1}{2}(\frac{x_j}{b_j} + 1)^2 - \frac{1}{2} & \text{if } \frac{x_j}{b_j} \leq 0, \\ +\infty & \text{otherwise} \end{cases}$
Squared	$\frac{1}{2}(\langle \mathbf{a}_j, \boldsymbol{\omega} \rangle - b_j)^2$	$\frac{1}{2}(x_j + b_j)^2 - \frac{1}{2}b_j^2$
Huber	$\begin{cases} \frac{1}{2}(\langle \mathbf{a}_j, \boldsymbol{\omega} \rangle - b_j)^2 & \text{if } \lvert\langle \mathbf{a}_j, \boldsymbol{\omega} \rangle - b_j\rvert \leq s, \\ s\lvert\langle \mathbf{a}_j, \boldsymbol{\omega} \rangle - b_j\rvert - \frac{1}{2}s^2 & \text{otherwise} \end{cases}$	$\begin{cases} \frac{1}{2}(x_j + b_j)^2 - \frac{1}{2}b_j^2 & \text{if } \lvert x_j\rvert \leq s, \\ +\infty & \text{otherwise} \end{cases}$

Table 3.2: Smooth loss examples. The dual of ℓ_1-regularized smooth loss minimization is a strongly convex constrained problem. Each feature corresponds to a dual constraint. We can use working sets to make convergence times less dependent on the number of features. In the table, L_j is the loss for training example j, $(\mathbf{a}_j, b_j)$ is the jth example, and s is a design parameter.

general. Assuming each L_j is 1-smooth, however, we can transform (PL1) into an instance of (PC) by considering the problem's dual (Borwein and Zhu, 2005, Chapter 4):

$$\begin{array}{ll} \underset{\mathbf{x} \in \mathbb{R}^m}{\text{minimize}} & \sum_{j=1}^n L_j^*(x_j) \\ \text{s.t.} & \lvert\langle \mathbf{A}_i, \mathbf{x} \rangle\rvert \leq \lambda \quad i = 1, \ldots, m\,. \end{array} \tag{PL1D}$$

By solving (PL1), we can efficiently recover (PL1D)'s solution and vice versa. In the dual problem, $\mathbf{A}_i \in \mathbb{R}^m$ refers to the ith column (feature) of the $n \times m$ design matrix $[\mathbf{a}_1, \ldots, \mathbf{a}_n]^T$. L_j^* denotes the convex conjugate of L_j. Since each L_j is 1-smooth, the L_j^* terms are 1-strongly convex (Rockafellar and Wets, 1997, Chapter 12). We include several examples of smooth loss functions and their convex conjugates in Table 3.2.

This dual transformation allows BlitzWS to exploit sparsity to solve (PL1) efficiently. Due to the correspondence between features and dual constraints, zero entries in $\boldsymbol{\omega}^\star$ correspond to constraints that are unnecessary for computing (PL1D)'s solution. That is, if we define $\mathcal{W}^\star = \{i \mid \omega_i^\star \neq 0\}$, then we can also compute $\mathbf{x}^\star$ by minimizing the dual objective subject only to constraints in $\mathcal{W}^\star$. Since $\boldsymbol{\omega}^\star$ is sparse, solving the problem with only $|\mathcal{W}^\star|$ constraints

Loss	Use	$L_i(\mathbf{x})$	
Hinge	Classification	$\begin{cases} 1 - b_i \langle \mathbf{a}_i, \mathbf{x} \rangle \\ 0 \end{cases}$	if $b_i \langle \mathbf{a}_i, \mathbf{x} \rangle \leq 1$, otherwise
Squared hinge	Classification	$\begin{cases} \frac{1}{2}(1 - b_i \langle \mathbf{a}_i, \mathbf{x} \rangle)^2 \\ 0 \end{cases}$	if $b_i \langle \mathbf{a}_i, \mathbf{x} \rangle \leq 1$, otherwise
Quantile	Regression	$\begin{cases} (1-s)(b_i - \langle \mathbf{a}_i, \mathbf{x} \rangle) \\ s(\langle \mathbf{a}_i, \mathbf{x} \rangle - b_i) \end{cases}$	if $\langle \mathbf{a}_i, \mathbf{x} \rangle \leq b_i$, otherwise

Table 3.3: Losses with piecewise linear subfunctions. For ℓ_2-regularized learning, we can leverage piecewise losses to reduce the training time's dependence on the number of observations. Above, L_i is the loss for example i, $(\mathbf{a}_i, b_i)$ is the ith training example, and s is a design parameter.

typically requires much less computation than solving (PL1D) directly.

Training examples in support vector machines

Our final example considers support vector machines and, more generally, ℓ_2-regularized loss minimization problems with piecewise loss functions. Given training examples $((\mathbf{a}_i, b_i))_{i=1}^m$ and a tuning parameter $C > 0$, we can learn a linear model by solving

$$\underset{\mathbf{x} \in \mathbb{R}^m}{\text{minimize}} \ \tfrac{1}{2} \|\mathbf{x}\|^2 + C \textstyle\sum_{i=1}^m L_i(\mathbf{x}) \,. \tag{PL2}$$

Each L_i is a loss function (parameterized by $\mathbf{a}_i$ and b_i). Often each L_i has piecewise linear components; we include some examples of such losses in Table 3.3.

When each L_i has piecewise linear components, we can solve the problem quickly by exploiting piecewise structure. Consider (PL2) instantiated with hinge loss. Given knowledge of $\pi_i(\mathbf{x}^\star)$ and whether $\mathbf{x}^\star \in \mathrm{bd}(\mathcal{X}_i^{(\star)})$ for each i (for this problem, this implies knowledge of the sign of $1 - b_i \langle \mathbf{a}_i, \mathbf{x}^\star \rangle$), we can define

$$L_i^\star(\mathbf{x}) = \begin{cases} 0 & \text{if } 1 - b_i \langle \mathbf{a}_i, \mathbf{x}^\star \rangle < 0 \,, \\ 1 - b_i \langle \mathbf{a}_i, \mathbf{x} \rangle & \text{if } 1 - b_i \langle \mathbf{a}_i, \mathbf{x}^\star \rangle > 0 \,, \\ L_i(\mathbf{x}) & \text{if } 1 - b_i \langle \mathbf{a}_i, \mathbf{x}^\star \rangle = 0 \,. \end{cases}$$

Applying Proposition 3.6, we can compute $\mathbf{x}^\star$ by solving

$$\underset{\mathbf{x}\in\mathbb{R}^m}{\text{minimize}}\ \tfrac{1}{2}\|\mathbf{x}\|^2 + C\sum_{i=1}^{n} L_i^\star(\mathbf{x})\,. \tag{PSVM*}$$

If we define $\mathcal{W}^\star = \{i\ :\ 1 - b_i\langle \mathbf{a}_i, \mathbf{x}^\star\rangle \neq 0\}$, the benefit becomes clear. (PSVM*) has the same solution as

$$\underset{\mathbf{x}\in\mathbb{R}^m}{\text{minimize}}\ \tfrac{1}{2}\|\mathbf{x}\|^2 + \langle \mathbf{a}^\star, \mathbf{x}\rangle + C\sum_{i\in\mathcal{W}^\star} L_i(\mathbf{x})\,, \tag{PSVM**}$$

where the vector $\mathbf{a}^\star$ is easily computable. We have reduced (PL2) from a problem with m training examples to a problem with $|\mathcal{W}^\star|$ training examples. Since often $|\mathcal{W}^\star| \ll m$, we can solve (PSVM*) much more efficiently.

3.5 BlitzWS working set algorithm

BlitzWS extends BlitzMN to problems of the form (P). In this section, we adapt main concepts from §3.3 to this piecewise formulation. We also address some practical considerations for BlitzWS.

3.5.1 Working set algorithms for piecewise objectives

To generalize working set algorithms to our piecewise problem formulation, we generalize the form of each subproblem. During each iteration t, BlitzWS minimizes a "relaxed objective,"

$$f_t(\mathbf{x}) = \psi(\mathbf{x}) + \sum_{i=1}^{m} \phi_{i,t}(\mathbf{x})\,, \tag{3.8}$$

where for $i \in [m]$, we have $\phi_{i,t} \in \{\phi_i\} \cup \{\phi_i^{(1)}, \phi_i^{(2)}, \ldots, \phi_i^{(p_i)}\}$. That is, BlitzWS defines each $\phi_{i,t}$ as either (i) the original piecewise function, ϕ_i, or (ii) one of ϕ_i's simpler subfunctions. The "working set" is the set of indices corresponding to piecewise functions in f_t.

Analogous to BlitzMN, BlitzWS computes $\mathbf{x}_t \leftarrow \operatorname{argmin} f_t(\mathbf{x})$ during iteration t. As long as $\phi_{i,t}$ is linear for most i, solving this subproblem generally requires much less time than solving (P).

3.5.2 Line search and $\mathbf{y}_t$ in BlitzWS

Like BlitzMN, BlitzWS maintains two iterates, $\mathbf{x}_t$ and $\mathbf{y}_t$. In BlitzMN, $\mathbf{y}_t$ is the feasible point on the segment $[\mathbf{y}_{t-1}, \mathbf{x}_t]$ with smallest objective value.

Extending this concept to the piecewise problem, BlitzWS initializes $\mathbf{y}_0$ such that $f(\mathbf{y}_0)$ is finite. After computing $\mathbf{x}_t$, BlitzWS updates $\mathbf{y}_t$ using the rule

$$\mathbf{y}_t \leftarrow \operatorname{argmin} \{f(\mathbf{x}) \mid \mathbf{x} \in [\mathbf{y}_{t-1}, \mathbf{x}_t]\} \ .$$

BlitzWS can compute this update using the bisection method; we elaborate on this in §3.5.8.

3.5.3 Equivalence regions in BlitzWS

BlitzWS's choice of each $\phi_{i,t}$ is a choice of *where* in ϕ_i's domain the algorithm ensures that $\phi_{i,t}$ and ϕ_i are equivalent. If $\phi_{i,t} = \phi_i^{(1)}$, then $\phi_{i,t}(\mathbf{x}) = \phi_i(\mathbf{x})$ is only guaranteed for $\mathbf{x} \in \mathcal{X}_i^{(1)}$.

Like BlitzMN, BlitzWS uses equivalence regions to ensure progress during each iteration. Given a progress parameter $\xi_t \in (0, 1]$, BlitzWS defines $\mathcal{S}_\xi^{\text{cap}}$ exactly as in §3.3.5:

$$\mathcal{S}_\xi^{\text{cap}} = \operatorname{conv} \left(\mathcal{B}_1^{\text{cap}} \cup \mathcal{B}_2^{\text{cap}}\right) \ .$$

Recall that $\mathcal{B}_1^{\text{cap}}$ and $\mathcal{B}_2^{\text{cap}}$ are balls with centers $\mathbf{c}_1^{\text{cap}}$ and $\mathbf{c}_2^{\text{cap}}$, respectively.

BlitzWS selects each $\phi_{i,t}$ so that for all $\mathbf{x} \in \mathcal{S}_\xi^{\text{cap}}$, we have $f_t(\mathbf{x}) = f(\mathbf{x})$. To establish this equivalence region, BlitzWS defines each $\phi_{i,t}$ so that $\phi_{i,t}(\mathbf{x}) = \phi_i(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{S}_\xi^{\text{cap}}$, $i \in [m]$. This property results from BlitzWS's first sufficient condition for including a particular $i \in [m]$ in the working set (there are three conditions total). For each $i \in [m]$, $k = \pi_i(\mathbf{c}_1^{\text{cap}})$, BlitzWS includes i in the working set—that is, BlitzWS defines $\phi_{i,t} = \phi_i$—if the following:

(C1) There exists an $\mathbf{x} \in \mathcal{S}_\xi^{\text{cap}}$ such that $\mathbf{x} \notin \mathcal{X}_i^{(k)}$, meaning *equivalence between ϕ_i and $\phi_i^{(k)}$ within $\mathcal{S}_\xi^{\text{cap}}$ is not guaranteed.*

We will define conditions (C2) and (C3) soon. If any of the conditions are true, BlitzWS includes i in the working set. Otherwise, BlitzWS defines $\phi_{i,t} = \phi_i^{(k)}$, where $k = \pi_i(\mathbf{c}_1^{\text{cap}})$.

3.5.4 Generalizing suboptimality gaps for BlitzWS

Condition (C2) for constructing the working set allows us to generalize the suboptimality gap to our piecewise formulation. Specifically, for each $i \in [m]$, $k = \pi_i(\mathbf{c}_1^{\text{cap}})$, BlitzWS defines $\phi_{i,t} = \phi_i$ if:

(C2) There exists an $\mathbf{x} \in \mathbb{R}^n$ such that $\phi_i^{(k)}(\mathbf{x}) > \phi_i(\mathbf{x})$—i.e., $\phi_i^{(k)}$ *does not lower bound* ϕ_i.

From (C2) and (3.8), it follows that $f_t(\mathbf{x}) \leq f(\mathbf{x})$ for all $\mathbf{x}$. We can use this property to define a suboptimality gap. Since $\mathbf{x}_t$ minimizes f_t, we have $f_t(\mathbf{x}_t) \leq f_t(\mathbf{x}^\star) \leq f(\mathbf{x}^\star)$. Thus, given any $\mathbf{y}_t$ such that $f(\mathbf{y}_t)$ is finite, we have the suboptimality gap

$$\Delta_t = f(\mathbf{y}_t) - f_t(\mathbf{x}_t) \geq f(\mathbf{y}_t) - f(\mathbf{x}^\star)\,.$$

We note (C2) does not generally affect the computational cost of minimizing f_t. The advantage of minimizing f_t instead of f results from the fact that linear $\phi_{i,t}$ functions collapse into a single linear term. In the case that $\phi_i^{(k)}$ is linear, then $\phi_i^{(k)}$ lower bounds ϕ_i as a result of convexity (assuming $\mathcal{X}_i^{(k)}$ has non-empty interior).

3.5.5 Ensuring $f_t(\mathbf{x}_t)$ is nondecreasing with t

BlitzMN defines each working set to ensure $\psi_{\text{MN}}(\mathbf{x}_t)$ is nondecreasing with t. Generalizing this idea, BlitzWS ensures $f_t(\mathbf{x}_t) \geq f_{t-1}(\mathbf{x}_{t-1})$ for all t. This property follows from condition (C3). For each $i \in [m]$, $k = \pi_i(\mathbf{c}_1^{\text{cap}})$, BlitzWS defines $\phi_{i,t} = \phi_i$ if:

(C3) $\phi_i^{(k)}$ *does not upper bound* $\phi_{i,t-1}$ *in a neighborhood of* $\mathbf{x}_{t-1}$—there exists a $\mathbf{g}_i \in \partial\phi_{i,t-1}(\mathbf{x}_{t-1})$ and $\mathbf{x} \in \mathbb{R}^n$ such that

$$\phi_i^{(k)}(\mathbf{x}) < \phi_{i,t-1}(\mathbf{x}_{t-1}) + \langle \mathbf{x} - \mathbf{x}_{t-1}, \mathbf{g}_i \rangle\ .$$

Because of (C3), we have $f_t(\mathbf{x}) \geq f_{t-1}(\mathbf{x}_{t-1}) + \langle \mathbf{x} - \mathbf{x}_{t-1}, \mathbf{g}_{t-1} \rangle$ for all $\mathbf{x}$ and $\mathbf{g}_{t-1} \in \partial f_{t-1}(\mathbf{x}_{t-1})$. Since $\mathbf{x}_{t-1}$ minimizes f_{t-1}, we have $\mathbf{0} \in \partial f_{t-1}(\mathbf{x}_{t-1})$. It follows that $f_t(\mathbf{x}) \geq f_{t-1}(\mathbf{x}_{t-1})$ for all $\mathbf{x}$, which implies that $f_t(\mathbf{x}_t) \geq f_{t-1}(\mathbf{x}_{t-1})$.

Algorithm 3.3 BlitzWS for solving (P)

input initial $\mathbf{y}_0$ such that $f(\mathbf{y}_0) < +\infty$, linear functions $(\phi_{i,0})_{i=1}^m$ for which $\phi_{i,0}(\mathbf{x}) \leq \phi_i(\mathbf{x})$ for all $\mathbf{x}$,
and method for choosing ξ_t
$\mathbf{x}_0 \leftarrow \operatorname{argmin} f_0(\mathbf{x}) := \psi(\mathbf{x}) + \sum_{i=1}^m \phi_{i,0}(\mathbf{x})$
for $t = 1, \ldots, T$ **until** $f_T(\mathbf{x}_T) = f(\mathbf{y}_T)$ **do**

 # Form subproblem:
Choose progress coefficient $\xi_t \in (0, 1]$
$\mathcal{S}_\xi^{\text{cap}} \leftarrow$ `compute_capsule_region`$(\xi_t, \mathbf{x}_{t-1}, \mathbf{y}_{t-1})$ *# see §3.3.5*
for $i = 1, \ldots, m$ **do**
if (C1) or (C2) or (C3) **then**
 # Include i in working set:
$\phi_{i,t} \leftarrow \phi_i$
else
$\phi_{i,t} \leftarrow \phi_i^{(k)}$ where k is the subdomain for which $\mathcal{S}_\xi^{\text{cap}} \subseteq \mathcal{X}_i^{(k)}$

 # Solve subproblem:
$\mathbf{x}_t \leftarrow \operatorname{argmin} f_t(\mathbf{x}) := \psi(\mathbf{x}) + \sum_{i=1}^m \phi_{i,t}(\mathbf{x})$

 # Perform line search update:
$\mathbf{y}_t \leftarrow \operatorname*{argmin}_{\mathbf{x} \in [\mathbf{y}_{t-1}, \mathbf{x}_t]} f(\mathbf{x})$

return $\mathbf{y}_T$

3.5.6 BlitzWS definition and convergence guarantee

We define BlitzWS in Algorithm 3.3. BlitzWS initializes $\mathbf{y}_0$ such that $f(\mathbf{y}_0)$ is finite, while $\mathbf{x}_0$ is the minimizer of a function f_0. BlitzWS defines f_0 so that $\mathbf{x}_0$ is easy to compute. For (PC), we might define $\phi_{i,0}(\mathbf{x}) = 0$ for all i, making $\mathbf{x}_0$ the unconstrained minimizer of ψ.

During each iteration t, BlitzWS chooses a progress coefficient ξ_t, which parameterizes the equivalence region $\mathcal{S}_\xi^{\text{cap}}$. BlitzWS defines f_t according to §3.5.3, §3.5.4, and §3.5.5. Given f_t, BlitzWS computes $\mathbf{x}_t \leftarrow \operatorname{argmin} f_t(\mathbf{x})$. At the end of iteration t, the algorithm updates $\mathbf{y}_t$ via line search.

Together, conditions (C1), (C2), and (C3) guarantee quantified progress toward convergence during iteration t. In particular, we have the following convergence result for BlitzWS:

Theorem 3.7 (Convergence bound for BlitzWS)**.** *For any iteration T of Algorithm 3.3, define the suboptimality gap $\Delta_T = f(\mathbf{y}_T) - f_T(\mathbf{x}_T)$. For all $T > 0$, we have*

$$\Delta_T \leq \Delta_0 \prod_{t=1}^{T} (1 - \xi_t) \,.$$

3.5.7 Accommodating approximate subproblem solutions

BlitzWS can minimize f_t using any subproblem solver. Since solvers are usually iterative, it is important to only compute $\mathbf{x}_t$ approximately. Computing $\mathbf{x}_t$ to high precision would require time that BlitzWS could instead use to solve subproblem $t + 1$.

To accommodate approximate solutions, we make several adjustments to BlitzWS. Most significantly, the subproblem solver returns three objects: (i) an approximate subproblem solution, $\mathbf{z}_t$, where $f_t(\mathbf{z}_t)$ is finite, (ii) a function f_t^{LB} that lower bounds f_t, and (iii) $\mathbf{x}_t = \operatorname{argmin} f_t^{\text{LB}}(\mathbf{x})$, which is a "dual" approximate minimizer of f_t. We assume f_t^{LB} takes the form

$$f_t^{\text{LB}}(\mathbf{x}) = \left[\psi(\mathbf{z}_t) + \left\langle \mathbf{g}_\psi^{\text{LB}}, \mathbf{x} - \mathbf{z}_t \right\rangle + \tfrac{1}{2} \left\| \mathbf{x} - \mathbf{z}_t \right\|^2\right] + \sum_{i=1}^{m} \left[\phi_{i,t}(\mathbf{z}_t) + \left\langle \mathbf{g}_i^{\text{LB}}, \mathbf{x} - \mathbf{z}_t \right\rangle\right] \,,$$

where $\mathbf{g}_\psi^{\text{LB}} \in \partial\psi(\mathbf{z}_t)$ and $\mathbf{g}_i^{\text{LB}} \in \partial\phi_i(\mathbf{z}_t)$ for each i. Since ψ is 1-strongly convex and each ϕ_i is convex, we have $f_t^{\text{LB}}(\mathbf{x}) \leq f_t(\mathbf{x})$ for all $\mathbf{x}$. Also, because f_t^{LB} is a simple quadratic function, it is straightforward to compute $\mathbf{x}_t$.

Together, $\mathbf{z}_t$, $\mathbf{x}_t$, and f_t^{LB} allow us to quantify the precision of the approximate subproblem solutions in terms of suboptimality gap. Since $\mathbf{x}_t$ minimizes f_t^{LB}, it follows that

$$f_t(\mathbf{z}_t) - f_t^{\text{LB}}(\mathbf{x}_t) \geq f_t(\mathbf{z}_t) - \min_{\mathbf{x}} f_t(\mathbf{x}) \,.$$

We note that when subproblem t is solved exactly, we can define f_t^{LB} such that this "subproblem suboptimality gap" is zero. To do so, we define $\mathbf{g}_\psi^{\text{LB}}$ and each $\mathbf{g}_i^{\text{LB}}$ such that $\mathbf{g}_\psi^{\text{LB}} + \sum_{i=1}^{m} \mathbf{g}_i^{\text{LB}} = \mathbf{0}$. In this case, $\mathbf{z}_t$ also minimizes f_t^{LB}, which implies that $\mathbf{x}_t = \mathbf{z}_t$ and $f_t(\mathbf{z}_t) - f_t^{\text{LB}}(\mathbf{x}_t) = 0$.

To bound the effect of approximate subproblem solutions, BlitzWS chooses a subproblem termination threshold $\epsilon_t \in [0, 1)$. We require that $\mathbf{z}_t$, $\mathbf{x}_t$, and f_t^{LB} satisfy two conditions:

$$f_t(\mathbf{z}_t) - f_t^{\text{LB}}(\mathbf{x}_t) \leq \epsilon_t \Delta_{t-1}\,,$$
$$\text{and} \qquad f_t^{\text{LB}}(\mathbf{x}_t) - f_{t-1}^{\text{LB}}(\mathbf{x}_{t-1}) \geq (1-\epsilon_t)\tfrac{1}{2}\left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2 .$$

The first condition bounds the subproblem suboptimality gap. The second condition lower bounds the algorithm's dual progress. The parameter ϵ_t trades off the precision of subproblem t's solution with the amount of time used to solve this subproblem—smaller ϵ_t values imply more precise subproblem solutions. While not obvious in this context, we note the second condition is always satisfied once the subproblem solution is sufficiently precise. We prove this fact in Appendix B.9.2.

In addition to the changes already discussed, we make three final modifications to BlitzWS to accommodate approximate subproblem solutions. First, we redefine BlitzWS's suboptimality gap to ensure $\Delta_t \geq f(\mathbf{y}_t) - f(\mathbf{x}^\star)$. Specifically, we define $\Delta_t = f(\mathbf{y}_t) - f_t^{\text{LB}}(\mathbf{x}_t)$.

The second final change is that instead of searching along the segment $[\mathbf{x}_t, \mathbf{y}_{t-1}]$, BlitzWS updates $\mathbf{y}_t$ by performing line search along $[\mathbf{z}_t, \mathbf{y}_{t-1}]$:

$$\mathbf{y}_t \leftarrow \operatorname*{argmin}_{\mathbf{x} \in [\mathbf{z}_t, \mathbf{y}_{t-1}]} f(\mathbf{x})\,.$$

The last change to BlitzWS adjusts condition (C3) from §3.5.5. Specifically, for each $i \in [m]$ and $k = \pi_i(\mathbf{c}_1^{\text{cap}})$, BlitzWS defines $\phi_{i,t} = \phi_i$ if:

(C3) $\phi_i^{(k)}$ *does not upper bound* $\phi_{i,t-1}^{\text{LB}}$—for some $\mathbf{x} \in \mathbb{R}^n$, we have $\phi_i^{(k)}(\mathbf{x}) < \phi_{i,t-1}^{\text{LB}}(\mathbf{x})$.

This change guarantees that f_t upper bounds f_{t-1}^{LB}. Compared to the original (C3) condition from §3.5.5, our new (C3) guarantees that $\phi_{i,t}$ upper bounds $\phi_{i,t-1}$ in a neighborhood of $\mathbf{z}_t$ as opposed to a neighborhood of $\mathbf{x}_t$.

Taking these changes into account, we have the following convergence result for BlitzWS with approximate subproblem solutions (proven in Appendix B.6):

Theorem 3.8 (Convergence bound for BlitzWS with approximate subproblem solutions)**.** *Consider BlitzWS with approximate subproblem solutions. For any iteration T, define the suboptimality gap $\Delta_T = f(\mathbf{y}_T) - f_T^{\mathrm{LB}}(\mathbf{x}_T)$. For all $T > 0$, we have*

$$\Delta_T \leq \Delta_0 \prod_{t=1}^{T} (1 - (1 - \epsilon_t)\xi_t) \,.$$

This result clearly describes the effect of approximate subproblem solutions. By solving subproblem t with tolerance $\epsilon_t \in [0, 1)$, it is guaranteed that BlitzWS makes $(1-\epsilon_t)\xi_t$ progress during iteration t. When $\epsilon_t = 0$, we recover our original convergence bound, Theorem 3.7.

The parameters ξ_t and ϵ_t allow BlitzWS to trade off between subproblem size, time spent solving subproblems, and progress toward convergence. We next explore these trade-offs in more detail.

3.5.8 Bottlenecks of BlitzWS

Each iteration t of BlitzWS has three stages: select subproblem t, solve subproblem t, and update $\mathbf{y}_t$. Here we discuss the amount of computation that each stage requires.

Time required to form each subproblem The time-consuming step for forming subproblem t is testing condition (C1). This step requires checking if $\mathcal{S}_\xi^{\mathrm{cap}} \subseteq \mathcal{X}_i^{(k)}$ for all $i \in [m]$.

Recall $\mathcal{S}_\xi^{\mathrm{cap}} = \mathrm{conv}\,(\mathcal{B}_1^{\mathrm{cap}} \cup \mathcal{B}_2^{\mathrm{cap}})$, where $\mathcal{B}_1^{\mathrm{cap}}$ and $\mathcal{B}_2^{\mathrm{cap}}$ are balls with centers $\mathbf{c}_1^{\mathrm{cap}}$ and $\mathbf{c}_2^{\mathrm{cap}}$ and radius r^{cap}. If $\mathcal{X}_i^{(k)}$ is convex, then $\mathcal{S}_\xi^{\mathrm{cap}} \subseteq \mathcal{X}_i^{(k)}$ iff $\mathcal{B}_1^{\mathrm{cap}} \subseteq \mathcal{X}_i^{(k)}$ and $\mathcal{B}_2^{\mathrm{cap}} \subseteq \mathcal{X}_i^{(k)}$. Unfortunately, when $\mathcal{X}_i^{(k)}$ is convex, testing whether $\mathcal{B}_1^{\mathrm{cap}} \subseteq \mathcal{X}_i^{(k)}$ is not convex in general. Even so, in the common scenarios that $\mathcal{X}_i^{(k)}$ is a half-space or ball, we can check if $\mathcal{B}_1^{\mathrm{cap}} \subseteq \mathcal{X}_i^{(k)}$ in $\mathcal{O}(n)$ time. In other cases, we can often approximately check this condition efficiently.

Let us first consider the case that $\mathcal{X}_i^{(k)}$ is a half-space. For some $\mathbf{a}_i \in \mathbb{R}^n$, $b_i \in \mathbb{R}$, we can write $\mathcal{X}_i^{(k)} = \{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \leq b_i\}$. Then $\mathcal{B}_1^{\mathrm{cap}} \subseteq \mathcal{X}_i^{(k)}$ iff

$$\langle \mathbf{a}_i, \mathbf{c}_1^{\mathrm{cap}} \rangle - b_i < \|\mathbf{a}_i\|\, r^{\mathrm{cap}} \,.$$

Alternatively, suppose that $\mathcal{X}_i^{(k)}$ is a ball: $\mathcal{X}_i^{(k)} = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{a}_i\| \leq b_i\}$. Then $\mathcal{B}_1^{\text{cap}} \subseteq \mathcal{X}_i^{(k)}$ iff

$$\|\mathbf{a}_i - \mathbf{c}_1^{\text{cap}}\| + r^{\text{cap}} < b_i \,.$$

When $\mathcal{X}_i^{(k)}$ is neither a half-space nor a ball, one option may be to approximate (C1) by defining a ball $\tilde{\mathcal{X}}_i^{(k)}$ such that $\tilde{\mathcal{X}}_i^{(k)} \subseteq \mathcal{X}_i^{(k)}$. If $\mathcal{B}_1^{\text{cap}} \subseteq \tilde{\mathcal{X}}_i^{(k)}$, then $\mathcal{B}_1^{\text{cap}} \subseteq \mathcal{X}_i^{(k)}$. If $\mathcal{B}_1^{\text{cap}} \subseteq \tilde{\mathcal{X}}_i^{(k)}$ and $\mathcal{B}_2^{\text{cap}} \subseteq \tilde{\mathcal{X}}_i^{(k)}$, then we must have $\mathcal{S}_\xi^{\text{cap}} \subseteq \mathcal{X}_i^{(k)}$. By approximating (C1) in this manner, f_t and f remain equivalent within $\mathcal{S}_\xi$, meaning Theorem 3.7 and Theorem 3.8 still apply.

Time required to solve subproblem t The time required to solve subproblem t depends mainly on three factors: the progress coefficient, the subproblem termination threshold, and the subproblem solver. Larger values of ξ_t result in a larger equivalence region, increasing the size of the working set due to (C1).

BlitzWS can use any solver to minimize f_t, but to be effective, the time required to solve each subproblem must increase with the working set size. This is usually the case but not always. For example, in the distributed setting, communication bottlenecks can affect convergence times greatly. Depending on the algorithm and implementation, some distributed solvers require $\mathcal{O}(n)$ communication per iteration, while communication for other solvers may scale with the working set size. The $\mathcal{O}(n)$ case is not desirable for BlitzWS, since time required to solve each subproblem depends less directly on the working set size.

Time required to update $\mathbf{y}_t$ Updating $\mathbf{y}_t$ requires minimizing f along the segment $[\mathbf{x}_t, \mathbf{y}_{t-1}]$. BlitzWS can perform this update using the bisection method, which requires evaluating f a logarithmic number of times. In this case, it is not necessary to compute $\mathbf{y}_t$ exactly. Our analysis requires only that $f(\mathbf{y}_t) \leq f(\mathbf{y}'_t)$, where $\mathbf{y}'_t$ is the point on the segment $[\mathbf{x}_t, \mathbf{y}_{t-1}]$ that is closest to $\mathbf{x}_t$ while remaining in the closure of $\mathcal{S}_\xi$.

In many cases it is also straightforward to compute $\mathbf{y}_t$ exactly. For constrained problems like (PMN), $\mathbf{y}_t$ is the extreme feasible point on the segment $[\mathbf{y}_{t-1}, \mathbf{x}_t]$. If the constraints are linear or quadratic, BlitzWS can compute $\mathbf{y}_t$ in closed form.

3.5.9 Choosing algorithmic parameters in BlitzWS

Each BlitzWS iteration uses a progress parameter, $\xi_t \in (0, 1]$, and termination threshold, $\epsilon_t \in [0, 1)$. We could assign ξ_t and ϵ_t constant values for all t. As we will see in §3.7, however, values of ξ_t and ϵ_t that work well for one problem may result in slow convergence times for other problems. For this reason, it is beneficial to choose these parameters in an adaptive manner.

To adapt the parameter choices to each problem, we model as functions of ξ_t and ϵ_t both (i) the time required to complete iteration t and (ii) BlitzWS's progress during this iteration. Using these models, BlitzWS selects ξ_t and ϵ_t by approximately optimizing the trade-offs between subproblem size, iteration duration, and convergence progress.

To model the time required for BlitzWS to complete iteration t, we define the function

$$\hat{T}_t(\xi, \epsilon) = \underbrace{C_t^{\text{setup}}}_{\substack{\text{Estimated time to} \\ \text{update } \mathbf{y}_t \text{ and define } f_t}} + \underbrace{C_t^{\text{solve}} \text{ProblemSize}(\xi)\epsilon^{-1}}_{\substack{\text{Estimated time to} \\ \text{solve subproblem } t}} . \tag{3.9}$$

Above, ProblemSize(ξ) measures the size of subproblem t as a function of ξ. For (PMN), we define

$$\text{ProblemSize}(\xi) = \textstyle\sum_{i \in \mathcal{W}_t(\xi)} \text{NNZ}(\mathbf{a}_i) \,,$$

where $\mathcal{W}_t(\xi)$ denotes the working set when $\xi_t = \xi$, and $\text{NNZ}(\mathbf{a}_i) = \|\mathbf{a}_i\|_0$.

BlitzWS adapts the scalars C_t^{setup} and C_t^{solve} from iteration to iteration. During each iteration t, BlitzWS measures the time required to solve subproblem t, denoted T_t^{solve}, as well as the time taken for all other steps of iteration t, denoted T_t^{setup}. Upon completion of iteration t, BlitzWS estimates C_t^{setup} and C_t^{solve} by solving for the appropriate value in the model:

$$\hat{C}_t^{\text{setup}} = T_t^{\text{setup}} \,, \qquad \text{and} \qquad \hat{C}_t^{\text{solve}} = \frac{T_t^{\text{solve}} \epsilon_t}{\text{ProblemSize}(\xi_t)} \,.$$

When selecting subproblem t, BlitzWS defines C_t^{setup} and C_t^{solve} by taking the median of

the five most recent estimates for these parameters. For example,

$$C_t^{\text{setup}} = \text{median}(\hat{C}_{t-1}^{\text{setup}}, \hat{C}_{t-2}^{\text{setup}}, \ldots, \hat{C}_{t-5}^{\text{setup}}) .$$

If $t \leq 5$, then the algorithm takes the median of only the past $t-1$ parameter estimates. Since this is not possible when $t = 1$ ($\hat{C}_0^{\text{setup}}$ does not exist), BlitzWS does not model the time required for iteration 1. Instead, during iteration 1, we define ξ_1 as the smallest value in $(0, 1]$ such that $f_t = f$, but we solve the subproblem crudely by terminating the subproblem solver after one iteration.

In addition to modeling the time for iteration t, BlitzWS applies Theorem 3.8 to model the suboptimality gap upon completion of this iteration:

$$\hat{\Delta}_t(\xi, \epsilon) = \max\left\{(1 - (1-\epsilon)\xi C_t^{\text{progress}})\Delta_{t-1}, \epsilon\Delta_{t-1}\right\} . \tag{3.10}$$

The parameter $C_t^{\text{progress}} \geq 1$ accounts for looseness in the theorem's bound, which guarantees that $\Delta_t \leq (1 - (1-\epsilon_t)\xi_t)\Delta_{t-1}$. The $\max\{\cdot\}$ in (3.10) results from the fact that we should not expect that $\Delta_t \leq \epsilon_t\Delta_{t-1}$, regardless of looseness in our bound. This is because as a termination condition for subproblem t, we only assume that the subproblem suboptimality gap does not exceed $\epsilon_t\Delta_{t-1}$.

BlitzWS estimates C_t^{progress} in the same way that the algorithm estimates C_t^{solve} and C_t^{setup}—by solving for the appropriate parameter after iteration t and taking the median over past estimates:

$$\hat{C}_t^{\text{progress}} = \frac{1}{(1-\hat{\epsilon}_t)\xi_t}\left[1 - \frac{\Delta_t}{\Delta_{t-1}}\right] , \qquad \hat{\epsilon}_t = \frac{f_t(\mathbf{z}_t) - f_t^{\text{LB}}(\mathbf{x}_t)}{\Delta_{t-1}} ,$$

$$\text{and} \quad C_t^{\text{progress}} = \max\left\{1, \text{median}(\hat{C}_{t-1}^{\text{progress}}, \hat{C}_{t-2}^{\text{progress}})\right\} .$$

Here the $\max\{1, \cdot\}$ guarantees that $C_t^{\text{progress}} \geq 1$—this ensures the value of $\hat{\Delta}(\xi, \epsilon)$ is at most the bound predicted by Theorem 3.8. In this case, we take the median of only the past two estimates for C_t^{progress}, allowing ξ_t to change significantly between iterations if necessary (unlike C_t^{setup}, for example, it is unclear whether we should expect C_t^{progress} to

be approximately constant for all t).

Having modeled both the time for iteration t and the progress during iteration t, BlitzWS combines $\hat{T}_t$ and $\hat{\Delta}_t$ to approximately optimize ξ_t and ϵ_t. Specifically, BlitzWS defines

$$\xi_t, \epsilon_t = \underset{\xi,\epsilon}{\operatorname{argmax}} \; -\frac{\log(\hat{\Delta}_t(\xi,\epsilon)/\Delta_{t-1})}{\hat{T}_t(\xi,\epsilon)}\,. \tag{3.11}$$

With (3.11), BlitzWS values time as if the algorithm converges linearly. That is, a subproblem that requires an additional second to solve should result in a Δ_t that is smaller by a multiplicative factor.

BlitzWS solves (3.11) approximately with grid search, considering 125 candidates for ξ_t and 10 candidates for ϵ_t. The candidates for ξ_t span between 10^{-6} and 1, while the candidates for ϵ_t span between 0.01 and 0.7. Later in §3.7, we examine some of these parameter values empirically.

We also enforce a time limit when solving each subproblem. In addition to the termination conditions described in §3.5.7, we also terminate subproblem t if the threshold ϵ_t is not reached before a specified amount of time elapses. We define the time limit as $C_t^{\text{solve}}\text{ProblemSize}(\xi_t){\epsilon_t}^{-1}$, which is the estimated time for solving the subproblem in (3.9).

3.5.10 Relation to prior algorithms

Many prior algorithms also exploit piecewise structure in convex problems. The classic simplex algorithm (Dantzig, 1965), for example, exploits redundant constraints in linear programs.

In the late 1990s and early 2000s, working set algorithms became important to machine learning for training support vector machines. Using working sets, Osuna et al. (1997) prioritized computation on training examples with suboptimal dual value. Joachims (1999) improved the choice of working sets based on a first-order "steepest feasible direction" strategy—an idea that Zoutendijk (1970) originally proposed for constrained optimization. To further reduce computation, Joachims developed a "shrinking" heuristic, which freezes values of spe-

cific dual variables that satisfy a condition during several consecutive iterations.

Later works refined and extended these working set ideas. Fan et al. (2005) as well as Glasmachers and Igel (2006) used second-order information to improve working sets for kernelized SVMs. Unlike BlitzWS, these approaches apply only to working sets of size two, which is limiting but nevertheless practical for kernelized SVMs. Zanghirati and Zanni (2003) and Zanni et al. (2006) considered larger working sets and parallel algorithms. Tsochantaridis et al. (2005) extended working set ideas to structured prediction problems. Hsieh et al. (2008) combined shrinking with dual coordinate ascent to train linear SVMs. This resulted in a very fast algorithm, and the popular LibLinear library (Fan et al., 2008) uses this approach to train linear SVMs today.

Similar coordinate descent strategies work well for training ℓ_1-regularized models. Friedman et al. (2010) proposed a fast algorithm that combines working sets with coordinate descent and a proximal Newton strategy. Similarly, Yuan et al. (2010) found that combining CD with shrinking heuristics leads to a fast algorithm for sparse logistic regression. Today LibLinear uses a refined version of this approach to train such models (Yuan et al., 2012), which applies working sets and shrinking in a two-layer prioritization scheme.

These are just two of many algorithms that incorporate working sets to speed up sparse optimization. For lasso-type problems, many additional studies combine working set (Scheinberg and Tang, 2016; Massias et al., 2017) or active set (Wen et al., 2012; Solntsev et al., 2015; Keskar et al., 2016) strategies with standard algorithms. Researchers have also applied working sets to many other sparse problems—see e.g. (Lee et al., 2007; Bach, 2008; Kim and Park, 2008; Roth and Fischer, 2008; Obozinski et al., 2009; Friedman et al., 2010; Schmidt and Murphy, 2010).

More generally, prioritizing components of the objective continues to be an important idea for scaling model training. Many works consider importance sampling to speed up stochastic optimization (Needell et al., 2014; Zhao and Zhang, 2015; Csiba et al., 2015; Vainsencher et al., 2015; Perekrestenko et al., 2017; Stich et al., 2017b). Harikandeh et al. (2015), Stich et al. (2017a), and Johnson and Guestrin (2017) use alternative strategies to

improve first-order algorithms.

To our knowledge, BlitzWS is the first working set algorithm that selects each working set in order to guarantee an arbitrarily large amount of progress during each iteration. Prior algorithms choose working sets in intuitive ways, but there is little understanding of the resulting progress. In contrast, our theory for BlitzWS provides justification for the algorithm, avoids possible pathological scenarios, and inspires new ideas, such as our approach to tuning algorithmic parameters.

We note that because BlitzWS can use any subproblem solver, BlitzWS could also use importance sampling, shrinking, or another strategy when solving each subproblem.

Two important features of BlitzWS are (i) the line search update to $\mathbf{y}_t$ and (ii) the fact that each subproblem objective lower bounds the original objective. We note that while combining these ideas within a working set algorithm is novel to our knowledge, combining these ideas is far from a new approach to convex optimization.

For example, the classic Frank-Wolfe algorithm (Frank and Wolfe, 1956) minimizes a constrained linear lower bound during each iteration before updating its iterate via line search. A more recent example is the "optimal quadratic averaging" algorithm proposed by D. Drusvyatskiy (2016). Other related algorithms include the bundle methods proposed by Teo et al. (2010), which also apply to training SVMs.

In each of these cases, a suboptimality gap results from minimizing a lower bound on the objective. This suboptimality gap is useful both for analysis and in practice.

3.6 BlitzScreen safe screening test

In this section, we introduce BlitzScreen, a safe screening test that relates closely to BlitzWS. Like BlitzWS, BlitzScreen involves minimizing a relaxed objective instead of the original objective, f. Unlike BlitzWS, BlitzScreen guarantees that the relaxed objective and f have the same minimizer.

3.6.1 BlitzScreen definition

BlitzScreen requires three ingredients:

1. An approximate minimizer of f, denoted $\mathbf{y}_0$, for which $f(\mathbf{y}_0)$ is finite.

2. A 1-strongly convex function, denoted f_0, that satisfies $f_0(\mathbf{x}) \leq f(\mathbf{x})$ for all $\mathbf{x}$.

3. The minimizer of f_0, denoted $\mathbf{x}_0$.

One way to construct such a f_0 uses a subgradient $\mathbf{g}_0 \in \partial f(\mathbf{y}_0)$. Given this $\mathbf{g}_0$, we can define

$$f_0(\mathbf{x}) = f(\mathbf{y}_0) + \langle \mathbf{g}_0, \mathbf{x} - \mathbf{y}_0 \rangle + \tfrac{1}{2} \|\mathbf{x} - \mathbf{y}_0\|^2 .$$

We can easily compute $\mathbf{x}_0 = \operatorname{argmin} f_0(\mathbf{x})$. Since f is 1-strongly convex, f_0 lower bounds f.

Regardless of how we define f_0, we have the following screening result.

Theorem 3.9 (BlitzScreen safe screening test)**.** *Let f_0 be any 1-strongly convex function that satisfies $f_0(\mathbf{x}) \leq f(\mathbf{x})$ for all $\mathbf{x}$, and let $\mathbf{x}_0 = \operatorname{argmin} f_0(\mathbf{x})$. Given any $\mathbf{y}_0 \neq \mathbf{x}^\star$, define the suboptimality gap $\Delta_0 = f(\mathbf{y}_0) - f_0(\mathbf{x}_0)$ as well as the "safe region"*

$$\mathcal{S}_1 = \left\{ \mathbf{x} \;\middle|\; \|\mathbf{x} - \tfrac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0)\| < \sqrt{\Delta_0 - \tfrac{1}{4}\|\mathbf{x}_0 - \mathbf{y}_0\|^2} \right\} .$$

For any $i \in [m]$, define k such that the subdomain $\mathcal{X}_i^{(k)}$ contains $\frac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0)$. Then if $\mathcal{S}_1 \subseteq \mathcal{X}_i^{(k)}$, we can safely replace ϕ_i with $\phi_i^{(k)}$ in f. That is, for all $i \in [m]$, if we let

$$\phi_{i,\mathrm{S}} = \begin{cases} \phi_i^{(k)} & \text{if } \mathcal{S}_1 \subseteq \mathcal{X}_i^{(k)} , \\ \phi_i & \text{otherwise,} \end{cases}$$

then the "screened objective" $f_{\mathrm{S}}(\mathbf{x}) := \psi(\mathbf{x}) + \sum_{i=1}^m \phi_{i,\mathrm{S}}(\mathbf{x})$ has the same minimizer as f.

We prove Theorem 3.9 in Appendix B.7. The proof relies on the equivalence of f_{S} and f within $\mathcal{S}_1$. As long as $f_{\mathrm{S}}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{S}_1$, these objectives have the same minimizer. Note the safe region size greatly depends on the approximate solutions. When Δ_0 is large,

$\mathcal{S}_1$ is large and $\phi_{i,\mathrm{S}} = \phi_i$ for many i. If Δ_0 is small, minimizing f_S can be significantly simpler than minimizing f.

Applying BlitzScreen requires checking whether $\mathcal{S}_1 \subseteq \mathcal{X}_i^{(k)}$ for each i. This condition is closely related to (C1) in BlitzWS, and our remarks in §3.5.8 about testing (C1) also apply to screening. In many scenarios, we can evaluate whether $\mathcal{S}_1 \subseteq \mathcal{X}_i^{(k)}$ in $\mathcal{O}(n)$ time.

3.6.2 Example: BlitzScreen for ℓ_1-regularized learning

As an example, we apply BlitzScreen to ℓ_1-regularized loss minimization:

$$\underset{\boldsymbol{\omega}\in\mathbb{R}^m}{\text{minimize}} \;\; g_{\mathrm{L1}}(\boldsymbol{\omega}) := \textstyle\sum_{j=1}^n L_j(\langle \mathbf{a}_j, \boldsymbol{\omega}\rangle) + \lambda \left\|\boldsymbol{\omega}\right\|_1 . \tag{PL1}$$

If each L_j is 1-smooth, we can transform the problem into its 1-strongly convex dual:

$$\underset{\mathbf{x}\in\mathbb{R}^m}{\text{minimize}} \;\; f_{\mathrm{L1D}}(\mathbf{x}) := \textstyle\sum_{j=1}^n L_j^*(x_j) + \sum_{i=1}^m \phi_i(\mathbf{x}) . \tag{PL1D}$$

Above, each implicit constraint defines $\phi_i(\mathbf{x}) = 0$ if $|\langle \mathbf{A}_i, \mathbf{x}\rangle| \le \lambda$ and $\phi_i(\mathbf{x}) = +\infty$ otherwise. Successfully screening a constraint in (PL1D) corresponds to eliminating a feature from (PL1) (the corresponding weight in $\boldsymbol{\omega}^\star$ takes value zero).

To apply BlitzScreen, we assume an approximate solution to (PL1), which we denote by $\boldsymbol{\omega}_0$. Letting $L_j'(\cdot)$ represent the derivative of $L_j(\cdot)$, we define

$$\mathbf{x}_0 = \left[L_1'(\langle \mathbf{a}_j, \boldsymbol{\omega}_0\rangle), \ldots, L_n'(\langle \mathbf{a}_j, \boldsymbol{\omega}_0\rangle)\right]^T , \quad \text{and} \quad f_0(\mathbf{x}) = \tfrac{1}{2}\left\|\mathbf{x} - \mathbf{x}_0\right\|^2 - g_{\mathrm{L1}}(\boldsymbol{\omega}_0) . \tag{3.12}$$

Using properties of duality, we show in Appendix B.9.3 that f_0 indeed lower bounds f_{L1D}.

BlitzScreen also requires a $\mathbf{y}_0 \in \mathbb{R}^m$ such that $f_{\mathrm{L1D}}(\mathbf{y}_0)$ is finite, meaning $\mathbf{y}_0$ must satisfy all constraints. We define $\mathbf{y}_0$ by scaling $\mathbf{x}_0$ toward $\mathbf{0}$ until this requirement is satisfied:

$$\mathbf{y}_0 = \frac{\lambda}{\max\limits_{i\in[m]} \; |\langle \mathbf{A}_i, \mathbf{x}_0\rangle|} \mathbf{x}_0 . \tag{3.13}$$

We note there exist more advanced strategies for defining $\mathbf{y}_0$ (Massias et al., 2018), but we do not consider such ideas. With (3.13), we have the following screening test for (PL1):

Corollary 3.10 (BlitzScreen for (PL1))**.** *Given any $\boldsymbol{\omega}_0$ that does not solve (PL1), define f_0, $\mathbf{x}_0$, and $\mathbf{y}_0$ as in (3.12) and (3.13). Define $\Delta_0 = f_{\mathrm{L1D}}(\mathbf{y}_0) + g_{\mathrm{L1}}(\boldsymbol{\omega}_0)$. For any $i \in [m]$, if*

$$\lambda - \left|\langle \mathbf{A}_i, \tfrac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0)\rangle\right| \geq \|\mathbf{A}_i\| \sqrt{\Delta_0 - \tfrac{1}{4}\|\mathbf{x}_0 - \mathbf{y}_0\|^2},$$

we can safely remove ϕ_i from (PL1D), which implies that $\omega_i^\star = 0$ for all $\boldsymbol{\omega}^\star$ that solve (PL1).

3.6.3 Relation to prior screening tests

BlitzScreen improves upon prior screening tests in a few ways, which we summarize as follows:

- *More broadly applicable:* Prior works have derived separate screening tests for different objectives, including sparse regression (El Ghaoui et al., 2012; Xiang and Ramadge, 2012; Tibshirani et al., 2012; Liu et al., 2014; Wang et al., 2015), sparse group lasso (Wang and Ye, 2014), as well as SVM and least absolute deviation problems (Wang et al., 2014). Extending screening to each new objective requires substantial new derivations. In contrast, BlitzScreen applies in a unified way to all instances of our piecewise problem formulation.

  Recently Raj et al. (2016) proposed a general recipe for deriving screening tests for different problems. Unlike this approach, BlitzScreen is an explicit screening test.

- *Adaptive:* Before recently, most safe screening tests relied on knowledge of an exact solution to a related problem. For example, El Ghaoui et al. (2012)'s test requires the solution to an identical problem but with greater regularization. This is disadvantageous for a few reasons, one of which is that screening only applies as a preprocessing step prior to optimization.

  Recent works have proposed *adaptive* (also called "dynamic") safe screening tests (Bonnefoy et al., 2014, 2015; Fercoq et al., 2015; Johnson and Guestrin, 2015; Ndiaye et al., 2015; Zimmert et al., 2015; Shibagaki et al., 2016; Raj et al., 2016; Ndiaye et al., 2016,

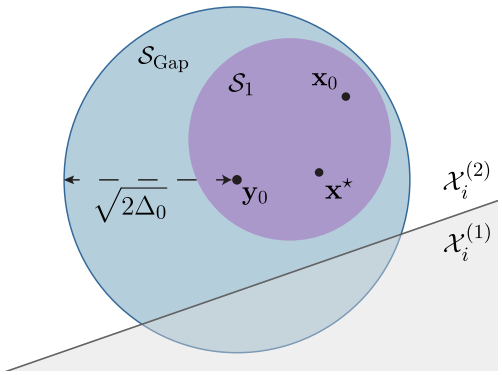


Figure 3.5: Relation to prior adaptive screening tests. BlitzScreen defines a region $\mathcal{S}_1$ that is a subset of the $\mathcal{S}_{\text{Gap}}$ region used by prior gap safe screening tests. As a result, BlitzScreen can simplify the objective more than these prior methods. In the illustration, we may safely replace ϕ_i with $\phi_i^{(2)}$ in f by using BlitzScreen but not by using prior adaptive tests.

2017). Adaptive screening tests increasingly simplify the objective as the quality of the approximate solution improves. BlitzScreen is an adaptive screening test.

- *More effective:* Prior to BlitzScreen, the "gap safe sphere" tests proposed by Fercoq et al. (2015) were state-of-the-art adaptive screening tests, as were the closely related tests proposed by Johnson and Guestrin (2015), Zimmert et al. (2015), Shibagaki et al. (2016), Raj et al. (2016) and Ndiaye et al. (2017). Each of these tests applies to a different class of objectives, but they relate to BlitzScreen in the same way. With the exception of Zimmert et al.'s result (which is a special case of BlitzScreen for SVM problems), we can recover these prior screening tests as special cases of BlitzScreen but only by replacing $\mathcal{S}_1$ with a larger set. Specifically, if we replace $\mathcal{S}_1$ in Theorem 3.9 with the larger ball

$$\mathcal{S}_{\text{Gap}} = \left\{ \mathbf{x} \;\middle|\; \|\mathbf{x} - \mathbf{y}_0\| \leq \sqrt{2\Delta_0} \right\}, \tag{3.14}$$

then the resulting theorem is a more general version of these existing tests. The main difference is that $\mathcal{S}_{\text{Gap}}$ is at least a factor $\sqrt{2}$ larger than the radius of $\mathcal{S}_1$. As a result, BlitzScreen is more effective at simplifying the objective.

3.6.4 Relation to BlitzWS

We can view safe screening as a working set algorithm that converges in one iteration. To solve "subproblem 1," we minimize the screened objective, f_S. The subproblem solution also solves (P).

Our next theorem shows that in the case of BlitzScreen and BlitzWS, this relation goes further:

Theorem 3.11 (Relation between equivalence regions in BlitzScreen and BlitzWS)**.** *Given points* $\mathbf{x}_0$ *and* $\mathbf{y}_0$*, function* f_0*, and suboptimality gap* Δ_0 *that satisfy the requirements for Theorem 3.9, define the ball* $\mathcal{S}_1$ *as in Theorem 3.9. In addition, consider the equivalence region* $\mathcal{S}_\xi$ *from §3.3 with parameter choices* $\xi_t = 1$*,* $\mathbf{x}_{t-1} = \mathbf{x}_0$*,* $\mathbf{y}_{t-1} = \mathbf{y}_0$*, and* $\Delta_{t-1} = \Delta_0$*. Then*

$$\mathcal{S}_1 = \mathcal{S}_\xi \,.$$

We prove Theorem 3.11 in Appendix B.8. When $\xi_1 = 1$, using BlitzWS is nearly equivalent to applying BlitzScreen. The only minor difference is that BlitzWS may not simplify the objective as much as BlitzScreen, since BlitzScreen does not consider conditions analogous to (C2) and (C3).

Importantly, it is usually *not* desirable for a working set algorithm to converge in one iteration. Since screening tests only make "safe" simplifications to the objective, screening tests often simplify the problem only a modest amount. In fact, unless a good approximate solution is already known, screening can fail to simplify the objective *at all*. We find it is usually better to simplify the objective aggressively, correcting erroneous choices later as needed. This is precisely the working set approach. As part of the next section, we support this observation with empirical results.

3.7 Empirical evaluation

This section demonstrates the performance of BlitzWS and BlitzScreen in practice.

3.7.1 Comparing the scalability of BlitzWS and BlitzScreen

We first consider a group lasso task and a linear SVM task. In each case, we examine how BlitzWS and BlitzScreen affect convergence times as the problem grows larger. To our knowledge, such scalability tests are a novel contribution to research on safe screening.

Scalability tests for group lasso application For our first experiment, we consider the group lasso objective (Yuan and Lin, 2006):

$$g_{\mathrm{GL}}(\boldsymbol{\omega}) := \tfrac{1}{2} \left\| \mathbf{A}\boldsymbol{\omega} - \mathbf{b} \right\|^2 + \lambda \sum_{i=1}^{m} \left\| \boldsymbol{\omega}_{\mathcal{G}_i} \right\| .$$

Here $\mathcal{G}_1, \ldots, \mathcal{G}_m$ are disjoint sets of feature indices such that $\cup_{i=1}^{m} \mathcal{G}_i = [q]$. Let $\boldsymbol{\omega}^\star \in \mathbb{R}^q$ denote a minimizer of g_{GL}. If $\lambda > 0$ is sufficiently large, then $\boldsymbol{\omega}^\star_{\mathcal{G}_i} = \mathbf{0}$ for many i.

We transform this problem into an instance of (P) by considering the dual problem:

$$\begin{array}{ll} \underset{\mathbf{x} \in \mathbb{R}^m}{\text{minimize}} & f_{\mathrm{GL}}(\mathbf{x}) := \tfrac{1}{2} \left\| \mathbf{x} + \mathbf{b} \right\|^2 - \tfrac{1}{2} \left\| \mathbf{b} \right\|^2 \\ \text{s.t.} & \left\| \mathbf{A}_{\mathcal{G}_i}^T \mathbf{x} \right\| \leq \lambda \qquad\qquad i = 1, \ldots, m \,. \end{array} \tag{PGD}$$

Each feature group corresponds to a constraint in the dual problem. Constraints that do not determine the dual solution correspond to zero-valued groups in the primal solution.

We apply group lasso to perform feature selection for a loan default prediction task. Using data available from Lending Club,[4] we train a boosted decision tree model to predict whether a loan will default during a given month. We apply group lasso to reduce the number of trees in the model. Features correspond to leaves in the tree model ($q \approx 3.0 \times 10^4$ features); groups correspond to trees ($m = 990$). We generate $n = 4.8 \times 10^5$ training examples by passing data through the tree model, using the model's prediction values (sum of appropriate leaf weights) as training labels. Since each tree maps each instance to one leaf, the feature matrices corresponding to each group are orthogonal.

There exist many algorithms for minimizing g_{GL} (Yuan and Lin, 2006; Liu et al., 2009;

[4] URL: `https://www.kaggle.com/wendykan/lending-club-loan-data`.

Kim et al., 2010). Our implementation uses the block coordinate descent approach of Qin et al. (2013). During an iteration, BCD updates weights in one group, keeping the remaining weights unchanged. Following Qin et al. (2013), our implementation computes an optimal update to $\boldsymbol{\omega}_{\mathcal{G}_i}$ for roughly the cost of multiplying a dual vector $\mathbf{x} \in \mathbb{R}^n$ by $\mathbf{A}_{\mathcal{G}_i}$. Each update requires solving a 1-D optimization problem, which we solve with the bisection method.

We implement BCD in C++. Using the same code base, we also implement the following:

- *BlitzWS:* To solve each subproblem, we use BCD.
- *BlitzWS + BlitzScreen:* After solving each BlitzWS subproblem, we apply BlitzScreen.
- *BCD + BlitzScreen:* After every five epochs, we apply BlitzScreen.
- *BCD + gap safe screening:* After every five passes over the groups, we apply gap safe screening (Ndiaye et al., 2015). This implementation is identical to BCD + BlitzScreen except we replace $\mathcal{S}_1$ in BlitzScreen with the $\mathcal{S}_{\text{Gap}}$ region defined in (3.14).

BlitzWS and the screening tests require checking if a region $\mathcal{S} = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{c}\| \leq r\}$ is a subset of a set $\mathcal{X}_i^{(1)} = \{\mathbf{x} \mid \left\|\mathbf{A}_{\mathcal{G}_i}^T \mathbf{x}\right\| \leq \lambda\}$. Computing this is nontrivial, so we apply relaxation ideas from §3.5.8 and §3.6.1. We define a set $\tilde{\mathcal{X}}_i^{(1)}$ such that $\tilde{\mathcal{X}}_i^{(1)} \subseteq \mathcal{X}_i^{(1)}$, and the algorithms test if $\mathcal{S} \subseteq \tilde{\mathcal{X}}_i^{(1)}$. For each $i \in [m]$, we let $L_i = \max_{k \in \mathcal{G}_i} \|\mathbf{A}_k\|$ and define $\tilde{\mathcal{X}}_i^{(1)} = \{\mathbf{x} \mid \left\|\mathbf{A}_{\mathcal{G}_i}^T \mathbf{c}\right\| + L_i \|\mathbf{x} - \mathbf{c}\| \leq \lambda\}$.

We perform data preprocessing to standardize groups in $\mathbf{A}$. For each i, we scale $\mathbf{A}_{\mathcal{G}_i}$ so the variances of each column sum to one. Our implementations include an unregularized bias variable. We can easily accommodate this bias term by adding the constraint $\langle \mathbf{x}, \mathbf{1} \rangle = 0$ to (PGD).

To test the scalability of BlitzWS and BlitzScreen, we create nine smaller problems from the original group lasso problem. We consider problems with $m = 990$, 330, and 110 groups by subsampling groups uniformly without replacement. We consider problems with $n = 480k$,

$160k$, and $53.3k$ training examples by subsampling examples. For each problem, we define λ so that exactly 10% of the groups have nonzero weight in the optimal model.

We evaluate performance using the relative suboptimality metric:

$$\text{Relative suboptimality} = \frac{g_{\text{GL}}(\boldsymbol{\omega}_T) - g_{\text{GL}}(\boldsymbol{\omega}^\star)}{g_{\text{GL}}(\boldsymbol{\omega}^\star)}.$$

Here $\boldsymbol{\omega}_T$ is the weight vector at time T. We take the optimal solution to be BlitzWS's solution after optimizing for twice the amount of time as displayed in each figure.

Figure 3.6 shows the results of these scalability tests. Our first takeaway is that for this problem, the number of training examples does not greatly affect the impact of BlitzWS and screening; as n increases, the relative performance of each algorithm is remarkably consistent. As the number of *groups* increases, we observe a different trend. When $m = 110$, the screening tests provide some speed-up compared to BCD with no screening, particularly once the relative suboptimality reaches 6×10^{-4}. When $m = 990$, however, screening provides much less benefit. In this case, despite being state-of-the-art for safe screening, BlitzScreen has no impact on convergence progress until relative suboptimality reaches 10^{-5}.

In contrast to safe screening, we find BlitzWS achieves significant speed-ups compared to BCD, regardless of m. We also note BlitzScreen provides no benefit when combined with BlitzWS. This is because BlitzWS already effectively prioritizes BCD updates.

Scalability tests for linear SVM application We perform similar scalability tests for a linear SVM problem ((PL2) with hinge loss). We consider a physics prediction task involving the Higgs boson (Adam-Bourdarios et al., 2014). We perform feature engineering using XGBoost (Chen and Guestrin, 2016), which achieves good accuracy for this problem (Chen and He, 2014). Using each leaf in the ensemble as a feature, the dataset contains $n = 8010$ features and $m = 10^7$ examples.

There exist many algorithms for solving this problem (Zhang, 2004; Joachims, 2006; Shalev-Shwartz et al., 2007; Teo et al., 2010). We use dual coordinate ascent, which is simple and fast (Hsieh et al., 2008). We implement DCA in C++. Like the group lasso

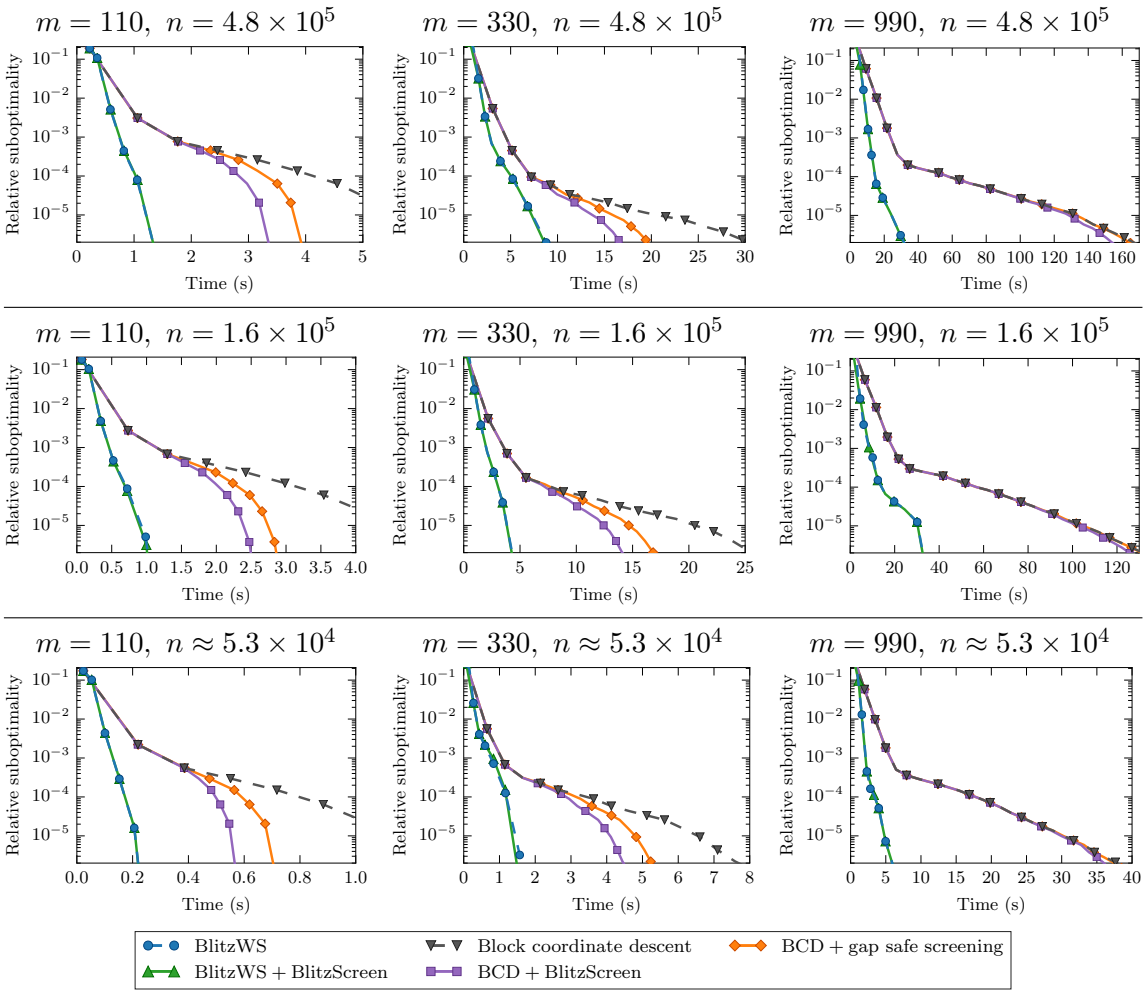


Figure 3.6: Scalability tests for group lasso. From left to right, the number of groups increases from 110 to 330 and finally to 990. From top to bottom, the number of examples decreases from 480k to 160k to 53.3k. The impact of screening degrades as the number of groups increases, but BlitzWS provides significant speed-ups in all cases. Each BlitzWS point represents 1 iteration; each BCD point represents 5 epochs.

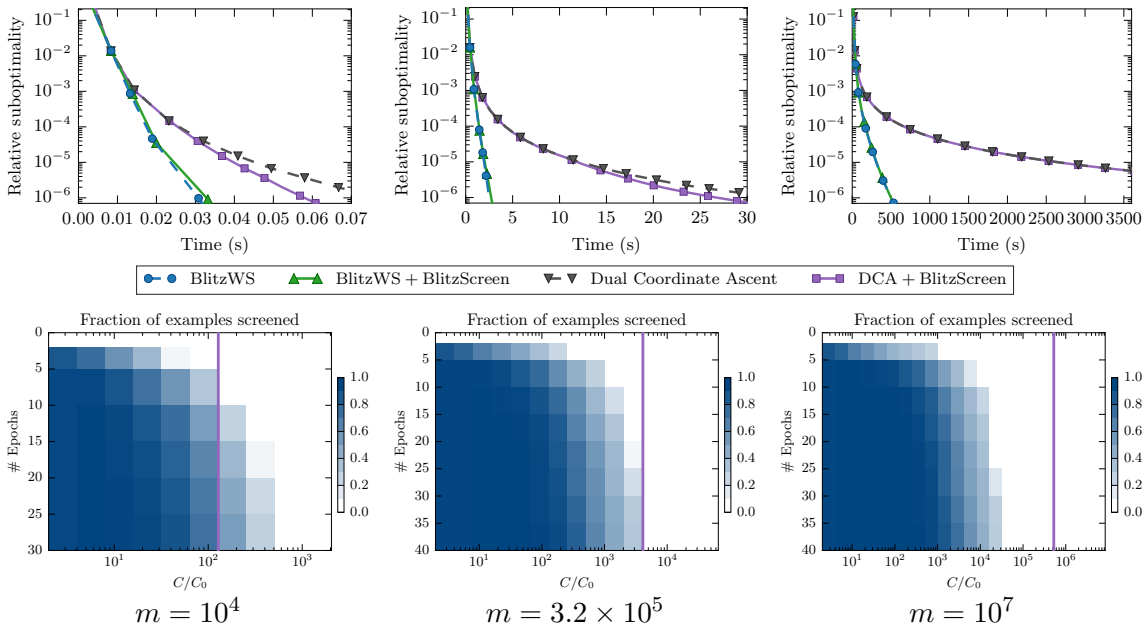


Figure 3.7: Scalability tests for linear SVM problem. From left to right, the number of training examples (m) increases for training a linear SVM. *Above:* Relative suboptimality vs. time. *Below:* Heat maps depicting the fraction of examples screened by BlitzScreen when used with dual coordinate ascent. The purple vertical line indicates the C chosen by five-fold cross validation. We also use this value of C for the above plots. As the number of examples increases, screening becomes less useful when training with desirable values of C.

comparisons, we implement BlitzWS using the same code base. For each algorithm, we also implement BlitzScreen.

By subsampling training examples without replacement, we test the scalability of BlitzWS and BlitzScreen using $m = 10^7$, 3.2×10^5, and 10^4 training examples. For each problem and each algorithm, we plot relative suboptimality vs. time—here we measure relative suboptimality using the dual objective. We choose C using five-fold cross validation.

We also test the performance of BlitzScreen using a range of C parameters. We show these results using heatmaps, where the y-axis indicates epochs completed by DCA, and the x-axis indicates C. The shading of the heat map depicts the fraction of training examples that BlitzScreen screens successfully at each point in the algorithm.

Figure 3.7 includes results from these comparisons. Similar to the group lasso case, we see BlitzScreen provides some speed-up when m is small. But as m increases, BlitzScreen has no impact on convergence times until the relative suboptimality is much smaller. In contrast, BlitzWS provides improvements that, relative to the DCA solver, do not degrade as m grows larger.

3.7.2 Comparing BlitzWS to LibLinear

LibLinear is one of the most popular and, to our knowledge, one of the fastest solvers for sparse logistic regression and linear SVM problems. Here we test how BlitzWS compares.

For sparse logistic regression, LibLinear uses working sets and shrinking to prioritize computation (Yuan et al., 2012). For linear SVM problems, LibLinear applies only shrinking (Joachims, 1999). We can view shrinking as a working set algorithm that initializes the working set with all components (i.e., $\mathcal{W}_t = [m]$ and $f_t = f$); then while solving the subproblem, shrinking progressively removes elements from $\mathcal{W}_t$ using a heuristic.

Sparse logistic regression comparisons Our LibLinear comparisons first consider sparse logistic regression ((PL1) with logistic loss). There are many efficient algorithms for solving this problem (Xiao, 2010; Shalev-Shwartz and Tewari, 2011; Bradley et al., 2011;

Defazio et al., 2014; Xiao and Zhang, 2014; Fercoq and Richtárik, 2015). To solve each subproblem, our BlitzWS implementation uses an inexact proximal Newton algorithm ("Prox-Newton"). We use coordinate descent to compute each proximal Newton step. LibLinear uses the same ProxNewton strategy (Yuan et al., 2012).

We compare BlitzWS with LibLinear version 2.11. We compile BlitzWS and LibLinear with GCC 4.8.4 and the `-O3` optimization flag. We compare with two baselines: our ProxNewton subproblem solver (no working sets) and ProxNewton combined with BlitzScreen. We perform screening as described in §3.6.2 after each ProxNewton iteration.

We compare the algorithms using data from the LIBSVM data repository.[5] Tasks include spam detection (Webb et al., 2006), malicious URL identification (Ma et al., 2009), text classification (Lewis et al., 2004), and educational performance prediction (Yu et al., 2010).

We perform conventional preprocessing on each dataset. We standardize all features to have unit variance. We remove features with fewer than ten nonzero entries. We include an unregularized bias term in the model. To accommodate this term, we add the constraint $\langle \mathbf{1}, \mathbf{x} \rangle = 0$ to (PL1D). Since LibLinear implements an ℓ_1-regularized bias term, we slightly modify LibLinear to (i) use regularization 0 for the bias variable, and (ii) always include the bias term in the working set.

We solve each problem using three λ values: $0.2\lambda_{\max}$, $0.02\lambda_{\max}$, and $0.002\lambda_{\max}$. Here $\lambda_{\max}$ is the smallest regularization value for which the problem's solution, $\boldsymbol{\omega}^\star$, equals $\mathbf{0}$. For each problem, we report the fraction of nonzero entries in $\boldsymbol{\omega}^\star$, which we denote by $s^\star$. We also report a weighted version of this quantity, which we define as $s^\star_{\mathrm{W}} = \frac{1}{\mathrm{NNZ}(\mathbf{A})} \sum_{i\,:\,\omega^\star_i \neq 0} \mathrm{NNZ}(\mathbf{A}_i)$. Here $\mathrm{NNZ}(\mathbf{A}_i)$ denotes the number of nonzero entries in column i of the design matrix.

With the exception of the spam detection problem, we solve each problem using a `m4.2xlarge` Amazon EC2 instance with 2.3 GHz Intel Xeon E5-2686 processors, 46 MB cache, and 32 GB memory. Due to memory requirements, we use a `r3.2xlarge` instance with 61 GB memory and Intel Xeon E5-2670 processors for the spam detection problem.

---

[5] URL: `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

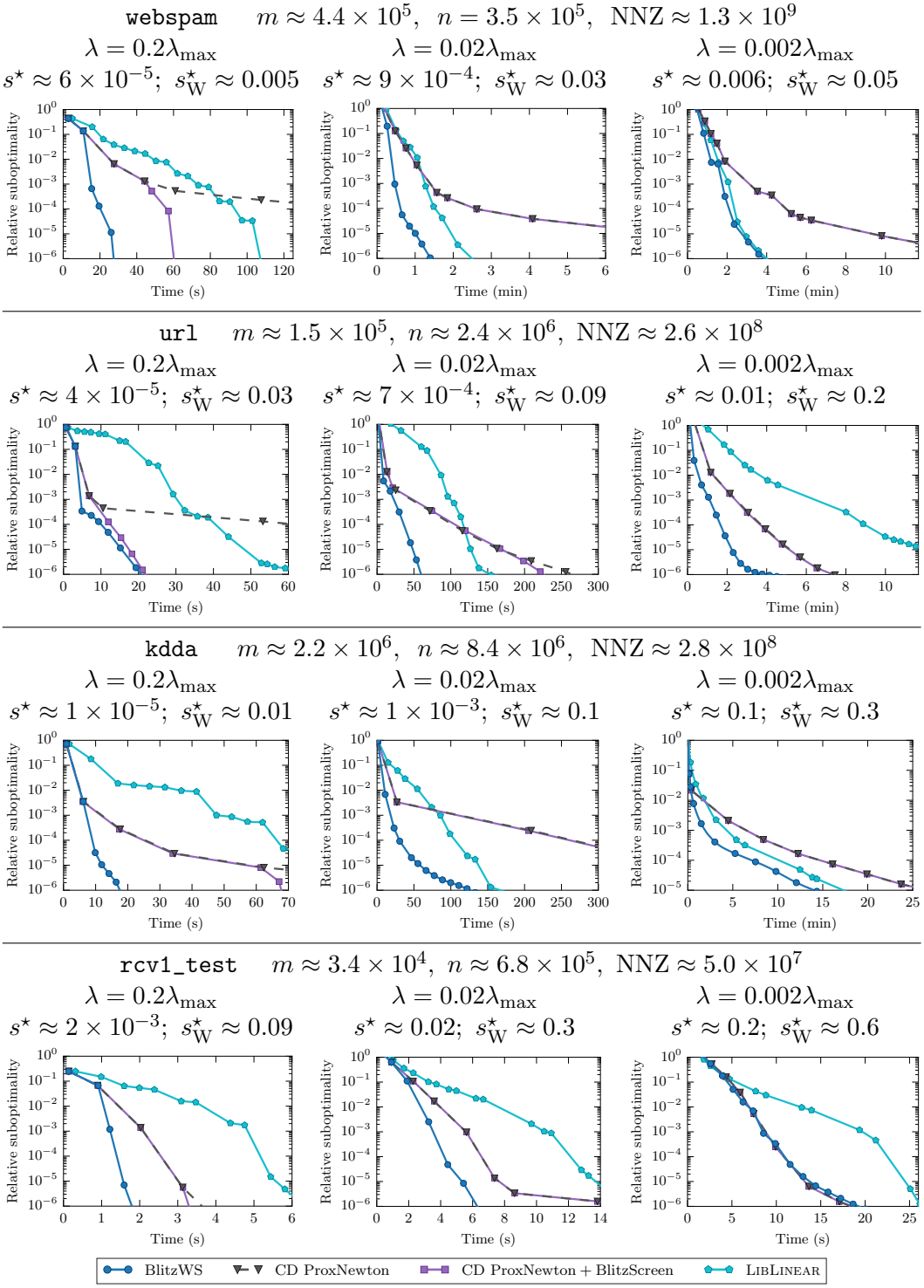


Figure 3.8: Convergence comparisons for sparse logistic regression. We compare BlitzWS to its subproblem solver and LIBLINEAR. BlitzWS provides consistent optimization speed-ups.

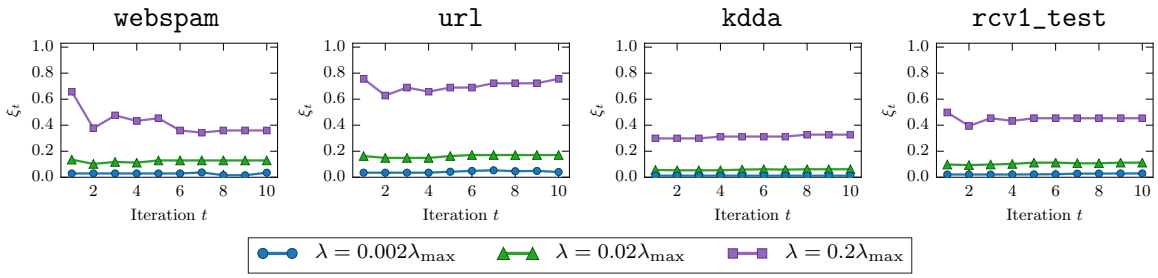


Figure 3.9: BlitzWS progress parameters for sparse logistic regression. Plots show ξ_t values that BlitzWS uses to produce the results in Figure 3.8. As regularization decreases, BlitzWS adapts by decreasing ξ_t.

Figure 3.8 contains the results of these comparisons. In many cases, we see that BlitzWS converges in much less time than LibLinear. Considering that LibLinear is an efficient, established library, these results show that BlitzWS is indeed a fast algorithm.

We also note that BlitzWS provides significant speed-ups compared to the non-working set approach. The amount of speed-up depends on the solution's sparsity, which is not surprising since we designed BlitzWS to exploit the solution's sparsity.

Adaptation to regularization strength We find BlitzWS outperforms BlitzScreen because BlitzWS adapts its ξ_t progress parameter to each problem. In contrast, ProxNewton + BlitzScreen is approximately equivalent to using BlitzWS with $\xi_t = 1$ for all iterations (as discussed in §3.6.4). Figure 3.9 contains plots of BlitzWS's chosen ξ_t parameters for each logistic regression problem. When $\lambda = 0.2\lambda_{\max}$, BlitzWS uses large ξ_t values, and screening (i.e., $\xi_t = 1$) also tends to perform well. As λ decreases, screening becomes ineffective, while BlitzWS adapts by choosing smaller ξ_t.

Impact of capsule approximation For the sparse logistic regression problems, we examine how BlitzWS's capsule approximation affects each working set. We log BlitzWS's state—$\mathbf{x}_{t-1}$, $\mathbf{y}_{t-1}$, and Δ_{t-1}—prior to selecting each working set. Then offline, we compute working sets for many values of ξ_t.

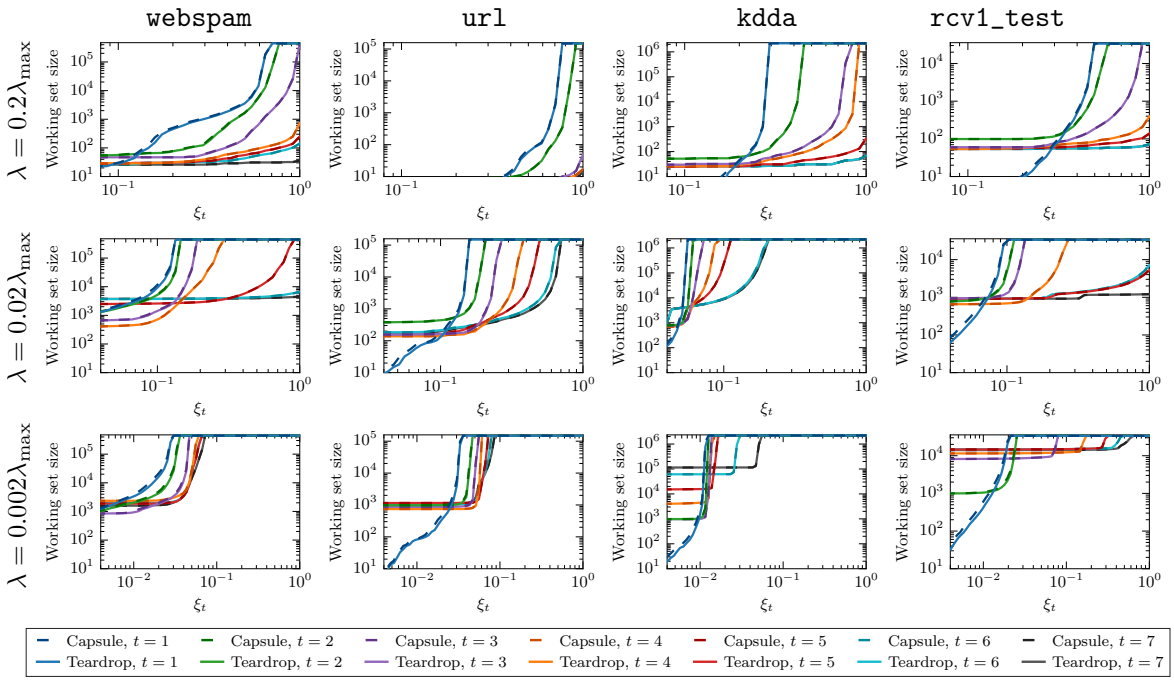


Figure 3.10: Impact of BlitzWS's capsule approximation. We plot the working set size vs. possible choices of the ξ_t progress parameter (dashed curves). Each of BlitzWS's first seven iterations corresponds to a different colored curve. We also plot the working set size when using the teardrop region, $\mathcal{S}_\xi$, to select each working set (solid curves). The close alignment of curves indicate the capsule approximation performs well.

We record working set sizes for each problem, iteration, and ξ_t value. In each case, we construct one working set using $\mathcal{S}_\xi$ and a second working set using the capsule approximation. To calculate the working set using $\mathcal{S}_\xi$, we this set using 200 β values.

We measure the capsule approximation's impact by comparing the sizes of $\mathcal{W}_\xi$ and $\mathcal{W}_\xi^{\text{cap}}$, where the teardrop determines $\mathcal{W}_\xi$ and the capsule determines $\mathcal{W}_\xi^{\text{cap}}$. If $|\mathcal{W}_\xi^{\text{cap}}| \approx |\mathcal{W}_\xi|$, then BlitzWS's capsule approximation has little impact on the makeup of each working set.

Figure 3.10 contains the results of this experiment. Observe that in all cases, $|\mathcal{W}_\xi^{\text{cap}}| \approx |\mathcal{W}_\xi|$. This suggests that $\mathcal{S}_\xi^{\text{cap}}$ is a very good approximation of $\mathcal{S}_\xi$.

Linear SVM comparisons We also compare with LibLinear for training linear SVMs. Like our BlitzWS implementation described in §3.7.1, LibLinear is also based on DCA.

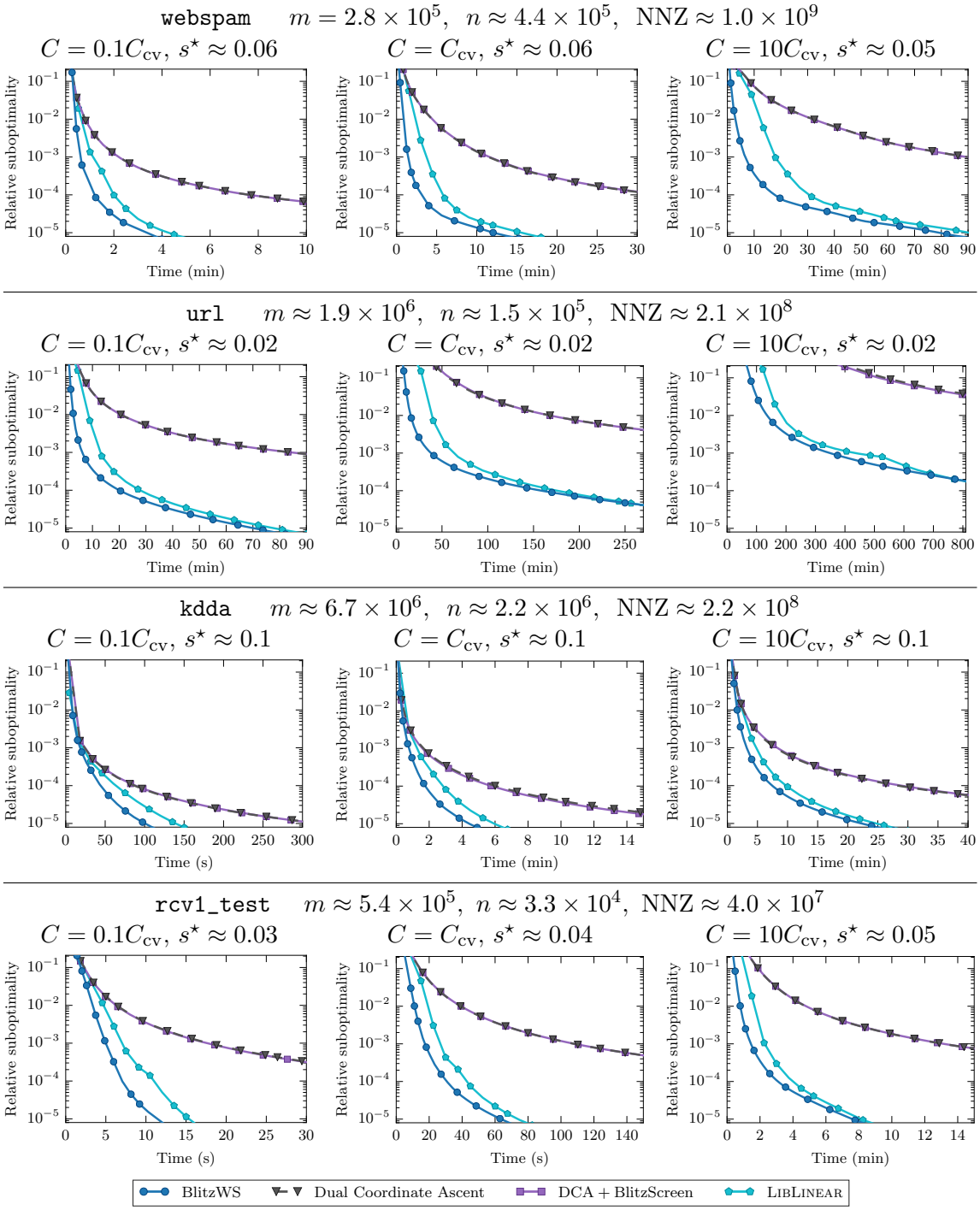


Figure 3.11: Convergence comparisons for linear SVMs. BlitzWS also leads to convergence time improvements when training linear SVMs. For more difficult problems, plot markers represent to multiple iterations.

For these comparisons, we use the same datasets, compilation settings, and hardware as we used in §3.7.2. For each dataset, we compute a practical value of C using five-fold cross validation, which we denote by C_{cv}. We compare using three values of C: $0.1C_{\text{cv}}$, C_{cv}, and $10C_{\text{cv}}$. We also report the solution's "sparsity," denoted $s^\star$, which we define as the fraction of training examples that are unbounded support vectors at the solution.

Figure 3.11 includes results from these comparisons. BlitzWS consistently provides speed-up compared to LibLinear, often during early iterations.

3.8 Discussion

We proposed BlitzWS, a principled yet practical working set algorithm. Unlike prior algorithms, BlitzWS selects subproblems in a way that maximizes guaranteed progress. We also analyzed the consequences of solving BlitzWS's subproblems approximately, and we applied this understanding to adapt algorithmic parameters as iterations progress.

In practice, BlitzWS is indeed a fast algorithm. Compared to the popular LibLinear library, BlitzWS achieves very competitive convergence times. Another appealing quality of BlitzWS is its capability of solving a variety of problems. This includes constrained problems, sparse problems, and piecewise loss problems. This flexibility results from §3.4's novel piecewise problem formulation. We find this formulation is a useful way of thinking about sparsity and related structure in optimization.

We also proposed a state-of-the-art safe screening test called BlitzScreen. Unlike prior screening tests, BlitzScreen applies to a large class of problems. Because of its relatively small safe region, BlitzScreen also simplifies the objective by a greater amount. Unfortunately, we found that in many practical scenarios, BlitzScreen had little impact on the algorithm's progress. While disappointing, we think this observation is an important contribution.

Exploiting piecewise structure can lead to large optimization speed-ups. Our analysis of BlitzWS and BlitzScreen provides a foundation for exploiting this structure in a principled way. We hope these contributions may serve as a starting point for future approaches to scalable optimization.

Chapter 4

PRIORITIZING COORDINATES WITH STINGY COORDINATE DESCENT

In this chapter, we propose an algorithm named "StingyCD." StingyCD shares many similarities with BlitzScreen from Chapter 3. Compared to screening, StingyCD leads to greater speed-ups, but StingyCD is also more specialized—it solves a much smaller class of problems, and it works only with coordinate descent.

Coordinate descent is a scalable and simple algorithm for solving several important optimization problems in machine learning. Despite this fact, CD can also be very computationally wasteful. Due to sparsity in sparse regression problems, for example, often the majority of CD updates result in no progress toward the solution.

To address this inefficiency, we propose StingyCD, which is a modified CD algorithm. By skipping over many updates that are guaranteed to not decrease the objective value, StingyCD significantly reduces convergence times. Since StingyCD only skips updates with this guarantee, however, StingyCD does not fully exploit the problem's sparsity. For this reason, we also propose StingyCD+, an algorithm that achieves further speed-ups by skipping updates more aggressively.

Since StingyCD and StingyCD+ rely on simple modifications to CD, it is straightforward to use these algorithms with other approaches to scaling optimization. In empirical comparisons, StingyCD and StingyCD+ improve convergence times considerably for ℓ_1-regularized learning problems.

The work in this chapter originally appeared at the 2017 International Conference on Machine Learning (Johnson and Guestrin, 2017).

4.1 Introduction

Known to be simple and fast, coordinate descent is a popular algorithm for training machine learning models. For ℓ_1-regularized loss minimization problems, such as the lasso (Tibshirani, 1996), CD updates just one weight variable at a time. As it turns out, these small yet inexpensive updates efficiently lead to the desired solution. Another attractive property of CD is its lack of parameters that require tuning, such as a learning rate.

Due to its appeal, researchers have analyzed CD extensively in recent years. This includes theoretical (Shalev-Shwartz and Tewari, 2011; Nesterov, 2012) and more applied (Fan et al., 2008; Friedman et al., 2010) contributions. Many works also consider scaling CD using parallelism (Bradley et al., 2011; Richtárik and Takáč, 2016). For surveys of research on CD, we refer the reader to (Wright, 2015) and (Shi et al., 2016).

Despite its popularity, CD has a significant drawback in some cases: for some applications, the majority of coordinate updates yield no progress toward convergence. In sparse regression, for example, most entries of the optimal weight vector equal zero. When CD updates these weights during optimization, the weights often equal zero both before and after the updates. This is very wasteful! Computing such "zero updates" requires time yet leaves the iterate unchanged.

In this chapter, we propose StingyCD, an improved CD algorithm for sparse optimization and linear SVM problems. With minimal added overhead, StingyCD identifies many coordinate updates that are guaranteed to result in zero improvement. By skipping over these updates instead of computing them directly, StingyCD can train models much faster.

StingyCD is related to safe screening tests (El Ghaoui et al., 2012), which for lasso problems, guarantee some weights equal zero *at the solution*. The algorithm can subsequently ignore screened weights for the remainder of optimization. As we have seen in Chapter 3, however, for screening to be effective, a good approximate solution must already be available. For this reason, screening often has little impact until convergence is near (Johnson and Guestrin, 2016, 2018a).

By identifying zero *updates* rather than zero-valued weights at the solution, StingyCD drastically improves convergence times compared to safe screening. At the same time, we find that skipping only updates that are guaranteed to be zero is limiting. For this reason, we also propose StingyCD+, an algorithm that estimates a probability that each update is zero. By also skipping updates that are likely zero, StingyCD+ achieves even greater speed-ups.

StingyCD and StingyCD+ require only simple changes to CD. Thus, we can combine these algorithms with other improvements to CD. In this chapter, we apply StingyCD+ to inexact proximal Newton and working set algorithms. In both cases, incorporating StingyCD+ leads to training time improvements, demonstrating that "stingy updates" can be a versatile strategy for speeding up CD algorithms.

4.2 StingyCD for nonnegative lasso

For simplicity, we introduce StingyCD for solving the problem

$$\begin{aligned} \underset{\boldsymbol{\omega}\in\mathbb{R}^m}{\text{minimize}} \quad & f(\boldsymbol{\omega}) := \tfrac{1}{2}\left\|\mathbf{A}\boldsymbol{\omega}-\mathbf{b}\right\|^2 + \lambda\left\langle \mathbf{1}, \boldsymbol{\omega}\right\rangle \\ \text{s.t.} \quad & \boldsymbol{\omega} \geq 0 \end{aligned} \qquad \text{(PNL)}$$

(PNL) is known as the "nonnegative lasso." Importantly, applications of StingyCD are not limited to (PNL). In §4.4, we explain how to apply StingyCD to general lasso and sparse logistic regression objectives as well as SVM problems.

In (PNL), $\mathbf{A}$ is an $n \times m$ design matrix, while $\mathbf{b} \in \mathbb{R}^n$ is a labels vector. Solving (PNL) results in a set of learned weights, which define a linear predictive model. The right term in the objective—commonly written as $\lambda \left\|\boldsymbol{\omega}\right\|_1$ for lasso problems without the nonnegativity constraint—is a regularization term that encourages the weights to have small value. The parameter $\lambda > 0$ controls the impact of the regularization term. Due to the nonnegativity constraint, a solution to (PNL) is *sparse* for sufficiently large λ. That is, the majority of entries in a solution to (PNL) have value zero.

Advantages of sparsity include reduced resources needed at test time, more interpretable models, and statistical efficiency (Wainwright, 2009). StingyCD is algorithm that exploits

Algorithm 4.1 Coordinate descent for solving (PNL)

initialize $\boldsymbol{\omega}^{(0)} \leftarrow \mathbf{0}^m$; $\mathbf{r}^{(0)} \leftarrow \mathbf{b}$
for $t = 1, 2, \ldots T$ **do**
  $i \leftarrow$ `get_next_coordinate()` *# choose sequentially or using uniform sampling*
  $\delta^{(t)} \leftarrow \max\left\{-\omega_i^{(t-1)}, \frac{\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle - \lambda}{\|\mathbf{A}_i\|^2}\right\}$
  $\boldsymbol{\omega}^{(t)} \leftarrow \boldsymbol{\omega}^{(t-1)} + \delta^{(t)} \mathbf{e}_i$
  $\mathbf{r}^{(t)} \leftarrow \mathbf{r}^{(t-1)} - \delta^{(t)} \mathbf{A}_i$
return $\mathbf{x}^{(T)}$

sparsity for efficient optimization.

4.2.1 Coordinate descent

Coordinate descent (CD) is a popular algorithm for solving (PNL). Algorithm 4.1 describes CD in detail. During iteration t, the algorithm chooses a coordinate $i \in \{1, \ldots, m\}$, usually at random or in round-robin fashion. CD updates entry i in $\boldsymbol{\omega}^{(t)}$ via $\omega_i^{(t)} \leftarrow \omega_i^{(t-1)} + \delta^{(t)}$, and remaining weights do not change. CD defines $\delta^{(t)}$ to maximally decrease the objective value subject to the nonnegativity constraint. Defining the residuals vector $\mathbf{r}^{(t-1)} = \mathbf{b} - \mathbf{A}\boldsymbol{\omega}^{(t-1)}$, we can write $\delta^{(t)}$ as

$$\delta^{(t)} = \max\left\{-\omega_i^{(t-1)}, \tfrac{1}{\|\mathbf{A}_i\|^2}\left(\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle - \lambda\right)\right\}. \tag{4.1}$$

Iteration t requires $\Theta(\mathrm{NNZ}(\mathbf{A}_i))$ time, where $\mathrm{NNZ}(\mathbf{A}_i)$ is the number of nonzero entries in column $\mathbf{A}_i$. Bottleneck operations are computing the dot product $\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle$ and updating $\mathbf{r}^{(t)}$ when $\delta^{(t)} \neq 0$. Efficient implementations cache $\|\mathbf{A}_i\|^2$ for later iterations.

4.2.2 Wasteful updates in coordinate descent

Because of the nonnegativity constraint and regularization penalty in (PNL), $\omega_i^{(t-1)} = 0$ during many iterations of Algorithm 4.1. When $\omega_i^{(t-1)} = 0$, if $\mathbf{r}^{(t-1)}$ lies outside of the "active update" region

$$\mathcal{A}_i = \{\mathbf{r} : \langle \mathbf{A}_i, \mathbf{r} \rangle - \lambda > 0\},$$

meaning $\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle - \lambda \leq 0$, then (4.1) implies $\delta^{(t)} = 0$. In this scenario, weight i equals zero at the beginning and end of iteration t. If solutions to (PNL) are sufficiently sparse, these "zero updates" account for the *majority* of iterations in naïve CD algorithms. Because each zero update requires $\Theta(\text{NNZ}(\mathbf{A}_i))$ time yet results in no progress toward convergence, computing these updates amounts to significant time wasted.

4.2.3 Stingy updates

Our proposed algorithm, StingyCD, improves convergence times for CD by "skipping over" many zero updates. Put differently, StingyCD computes some zero updates in constant time by guaranteeing $\delta^{(t)} = 0$ without computing this quantity directly via (4.1).

We saw in §4.2.2 that sufficient conditions for $\delta^{(t)} = 0$ are (i) $\omega_i^{(t-1)} = 0$ and (ii) $\mathbf{r}^{(t-1)} \notin \mathcal{A}_i$. Since directly testing the second condition requires $\Theta(\text{NNZ}(\mathbf{A}_i))$ time, simply checking these conditions does not lead to a useful method for quickly guaranteeing $\delta^{(t)} = 0$.

The insight that enables StingyCD is that we can relax the condition $\mathbf{r}^{(t-1)} \notin \mathcal{A}_i$ to form a condition that is testable in constant time. This relaxation depends on a region $\mathcal{S}^{(t)}$ for which $\mathbf{r}^{(t-1)} \in \mathcal{S}^{(t)}$. In particular, $\mathcal{S}^{(t)}$ is a ball:

$$\mathcal{S}^{(t)} = \left\{ \mathbf{r} \,:\, \|\mathbf{r} - \mathbf{rr}\|^2 \leq q^{(t-1)} \right\}, \text{ where } q^{(t-1)} = \left\|\mathbf{r}^{(t-1)} - \mathbf{rr}\right\|^2 .$$

Above, $\mathbf{rr}$ is a "reference residuals" vector—a copy of the residuals from a previous iteration. Formally, $\mathbf{rr} = \mathbf{r}^{(t-k)}$ for some $k \geq 1$ (to be defined more precisely later). Note that $\mathbf{r}^{(t-1)}$ lies on the boundary of $\mathcal{S}^{(t)}$.

During any iteration t for which $x_i^{(t-1)} = 0$, StingyCD considers whether $\mathcal{S}^{(t)} \cap \mathcal{A}_i = \emptyset$ before computing $\delta^{(t)}$. If $\mathcal{S}^{(t)} \cap \mathcal{A}_i = \emptyset$, then $\mathbf{r}^{(t-1)} \notin \mathcal{A}_i$, which guarantees $\delta^{(t)} = 0$. In this case, StingyCD continues to iteration $t+1$ without computing $\delta^{(t)}$ directly. We illustrate this concept in Figure 4.1. Defining $g_i = -\langle \mathbf{A}_i, \mathbf{rr} \rangle + \lambda$, we can simplify the condition

$\mathcal{S}^{(t)} \cap \mathcal{A}_i = \emptyset$ as follows:

$$\begin{aligned} \mathcal{S}^{(t)} \cap \mathcal{A}_i = \emptyset \quad &\Leftrightarrow \quad \max_{\mathbf{r} \in \mathcal{S}^{(t)}} \langle \mathbf{A}_i, \mathbf{r} \rangle - \lambda \leq 0 \\ &\Leftrightarrow \quad -g_i + \|\mathbf{A}_i\| \sqrt{q^{(t-1)}} \leq 0 \\ &\Leftrightarrow \quad q^{(t-1)} \leq \operatorname{sgn}(g_i) \tfrac{g_i^2}{\|\mathbf{A}_i\|^2} =: \tau_i \,. \end{aligned}$$

Thus, if $q^{(t-1)} \leq \tau_i$, then $\mathbf{r}^{(t-1)} \notin \mathcal{A}_i$ (implying $\delta^{(t)} = 0$ if also $\omega_i^{(t-1)} = 0$).

We note that τ_i can take positive or negative value, depending on if $\mathbf{rr} \in \mathcal{A}_i$. If $\mathbf{rr} \notin \mathcal{A}_i$, then $g_i \geq 0$, which implies $\tau_i \geq 0$. However, if $\mathbf{rr} \in \mathcal{A}_i$, then $\tau_i < 0$, and since $q^{(t-1)}$ is nonnegative, it cannot be true that $q^{(t-1)} \leq \tau_i$—StingyCD does not skip over coordinate i in this case. Therefore, when $\tau_i < 0$, the magnitude of τ_i is not significant to StingyCD, although this magnitude has greater importance for StingyCD+ in §4.3.

Importantly, we can test the condition $q^{(t-1)} \leq \tau_i$ with minimal overhead by (i) updating $\mathbf{rr}$ only occasionally, (ii) precomputing $\langle \mathbf{A}_i, \mathbf{rr} \rangle$ and τ_i for all i whenever StingyCD updates $\mathbf{rr}$, and (iii) tracking the value of $q^{(t-1)}$, which is updated appropriately after each nonzero coordinate update. We next describe these steps in more detail.

4.2.4 StingyCD definition and guarantees

We define StingyCD in Algorithm 4.2. StingyCD builds upon Algorithm 4.1 with three simple changes. First, during some iterations, StingyCD updates the reference residuals vector, $\mathbf{rr} \leftarrow \mathbf{r}^{(t-1)}$. When updating $\mathbf{rr}$, StingyCD also computes a thresholds vector, $\boldsymbol{\tau}$, which requires computing $\langle \mathbf{A}_i, \mathbf{rr} \rangle$ for all columns in $\mathbf{A}$. Updating $\mathbf{rr}$ is relatively costly, but frequent reference updates also lead to more skipped updates.

The second change to CD is that StingyCD tracks the quantity $q^{(t)} = \left\| \mathbf{r}^{(t)} - \mathbf{rr} \right\|^2$. After each update to $\mathbf{rr}$, StingyCD sets $q^{(t)}$ to 0. After each nonzero residuals update, $\mathbf{r}^{(t)} \leftarrow \mathbf{r}^{(t-1)} - \delta^{(t)} \mathbf{A}_i$, StingyCD makes a corresponding update to $q^{(t)}$. Importantly, we can perform this $q^{(t)}$ update in constant time by caching the quantities $\|\mathbf{A}_i\|^2$, $\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle$,

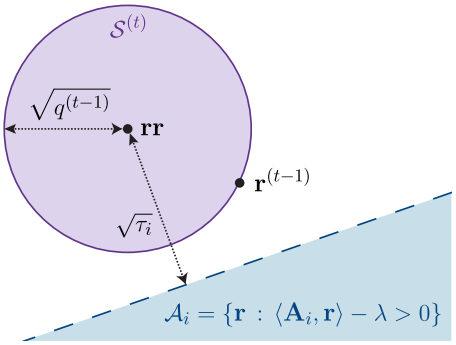


Figure 4.1: Geometry of StingyCD. During iteration t of CD, if $\omega_i^{(t-1)} = 0$ and $\mathbf{r}^{(t-1)} \notin \mathcal{A}_i$, then $\delta^{(t)} = 0$. In this case, computing $\delta^{(t)}$ is wasteful because the "update" makes no change to $\boldsymbol{\omega}^{(t-1)}$. StingyCD skips over many zero updates by establishing a region $\mathcal{S}^{(t)}$ for which $\mathbf{r}^{(t-1)} \in \mathcal{S}^{(t)}$. If $\mathcal{S}^{(t)} \cap \mathcal{A}_i = \emptyset$ and $\omega_i^{(t-1)} = 0$, it is guaranteed that $\delta^{(t)} = 0$, and StingyCD continues to iteration $t+1$ without computing $\delta^{(t)}$ directly. In the illustration, StingyCD successfully guarantees $\delta^{(t)} = 0$, since $q^{(t-1)} \leq \tau_i$. In contrast, StingyCD would compute $\delta^{(t)}$ directly if $q^{(t-1)} > \tau_i$. We note $\sqrt{\tau_i}$ is well-defined in the illustration—since $\mathbf{rr} \notin \mathcal{A}_i$, we have $\tau_i \geq 0$.

$\langle \mathbf{A}_i, \mathbf{rr} \rangle$, and $\delta^{(t)}$—all of which have been computed earlier.

The final modification to CD is StingyCD's use of stingy updates. Before computing $\delta^{(t)}$ during each iteration t, StingyCD checks whether $q^{(t-1)} \leq \tau_i$ and $x_i^{(t-1)} = 0$. If both are true, StingyCD continues to the next iteration without computing $\delta^{(t)}$. Note the threshold τ_i is computed after each update to $\mathbf{rr}$.

StingyCD's choice of $\boldsymbol{\tau}$ ensures that each skipped update is "safe":

Theorem 4.1 (Safeness of StingyCD)**.** *In Algorithm 4.2, every skipped update would, if computed, result in* $\delta^{(t)} = 0$*. That is, if* $q^{(t-1)} \leq \tau_i$ *and* $\omega_i^{(t-1)} = 0$*, then*

$$\max \left\{ -\omega_i^{(t-1)}, \frac{\left\langle \mathbf{A}_i, \mathbf{b} - \mathbf{A}\boldsymbol{\omega}^{(t-1)} \right\rangle - \lambda}{\|\mathbf{A}_i\|^2} \right\} = 0 \,.$$

We prove Theorem 4.1 in Appendix C.1. Theorem 4.1 is useful because it guarantees that although StingyCD may skip many updates, CD and StingyCD have identical weight vectors for all iterations (assuming each algorithm updates coordinates in the same order). Our next theorem formalizes the notion that these skipped updates come nearly "for free."

Algorithm 4.2 StingyCD for solving (PNL)

initialize $\boldsymbol{\omega}^{(0)} \leftarrow \mathbf{0}^m$; $\mathbf{r}^{(0)} \leftarrow \mathbf{b}$; $\mathbf{rr} \leftarrow \mathbf{r}^{(0)}$;
$q^{(0)} \leftarrow 0$; $\boldsymbol{\tau} \leftarrow$ `compute_thresholds`$(\boldsymbol{\omega}^{(0)})$
cache $\|\mathbf{A}_i\|^2$ for all $i \in \{1, \ldots, m\}$
for $t = 1, 2, \ldots T$ **do**
  # Update reference residuals on occasion:
  if `should_update_reference`() **then**
    $\mathbf{rr} \leftarrow \mathbf{r}^{(t-1)}$; $\boldsymbol{\tau} \leftarrow$ `compute_thresholds`$(\boldsymbol{\omega}^{(t-1)})$; $q^{(t-1)} \leftarrow 0$
  $i \leftarrow$ `get_next_coordinate`()
  if $q^{(t-1)} \leq \tau_i$ **and** $\omega_i^{(t-1)} = 0$ **then**
    # Skip update:
    $\boldsymbol{\omega}^{(t)} \leftarrow \boldsymbol{\omega}^{(t-1)}$; $\mathbf{r}^{(t)} \leftarrow \mathbf{r}^{(t-1)}$; $q^{(t)} \leftarrow q^{(t-1)}$
    continue
  $\delta^{(t)} \leftarrow \max\left\{-\omega_i^{(t-1)}, \frac{\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle - \lambda}{\|\mathbf{A}_i\|^2}\right\}$
  $\boldsymbol{\omega}^{(t)} \leftarrow \boldsymbol{\omega}^{(t-1)} + \delta^{(t)} \mathbf{e}_i$
  $\mathbf{r}^{(t)} \leftarrow \mathbf{r}^{(t-1)} - \delta^{(t)} \mathbf{A}_i$
  $q^{(t)} \leftarrow q^{(t-1)} - 2\delta^{(t)} \left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} - \mathbf{rr} \right\rangle + (\delta^{(t)})^2 \|\mathbf{A}_i\|^2$
return $\mathbf{x}^{(T)}$

function `compute_thresholds`$(\boldsymbol{\omega})$
  initialize $\boldsymbol{\tau} \leftarrow \mathbf{0}^m$
  for $i \in \{1, \ldots, m\}$ **do**
    $g_i \leftarrow \langle \mathbf{A}_i, \mathbf{A}\boldsymbol{\omega} - \mathbf{b} \rangle + \lambda$
    $\tau_i \leftarrow \operatorname{sgn}(g_i) \frac{g_i^2}{\|\mathbf{A}_i\|^2}$
  return $\boldsymbol{\tau}$

Theorem 4.2 (Per iteration time complexity of StingyCD—proven in Appendix C.2)**.** *Algorithm 4.2 can be implemented so that iteration t requires*

- *Less time than an identical iteration of Algorithm 4.1 if $q^{(t-1)} \leq \tau_i$ and $\omega^{(t-1)} = 0$ (the update is skipped) and* $\mathbf{rr}$ *is not updated. Specifically, StingyCD requires constant time, while CD requires $\Theta(\mathrm{NNZ}(\mathbf{A}_i))$ time.*
- *Approximately the same amount of time as a CD iteration if the update is not skipped and* $\mathbf{rr}$ *is not updated. (Both algorithms require the same number of $\Theta(\mathrm{NNZ}(\mathbf{A}_i))$ operations.)*
- *More time than a CD iteration if* $\mathbf{rr}$ *is updated. (StingyCD requires $\Theta(\mathrm{NNZ}(\mathbf{A}))$ time.)*

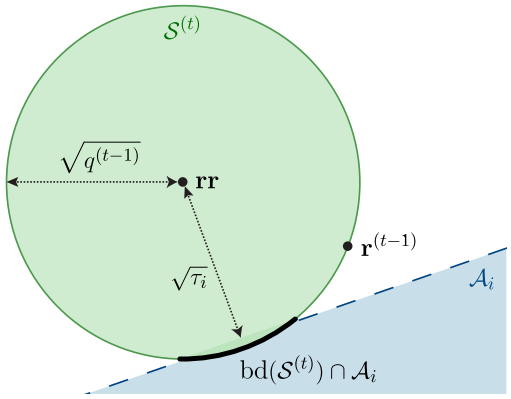


Figure 4.2: Probability of a useful update. StingyCD skips update t iff $q^{(t-1)} \leq \tau_i$ and $\omega_i^{(t-1)} = 0$, which guarantee $\delta^{(t)} = 0$. To skip more updates, StingyCD+ applies the intuition that if $q^{(t-1)}$ is only slightly larger than τ_i, it is unlikely that $\mathbf{r}^{(t-1)} \in \mathcal{A}_i$, implying that a nonzero update is also unlikely. To do so, StingyCD+ models the probability $P(\mathcal{U}^{(t)})$ that $\delta^{(t)} \neq 0$. Assuming $\omega_i^{(t-1)} = 0$ in the illustrated scenario, StingyCD+ computes $P(\mathcal{U}^{(t)})$ by dividing the length of the black arc $(\mathrm{bd}(\mathcal{S}^{(t)}) \cap \mathcal{A}_i)$ by the circumference of $\mathcal{S}^{(t)}$.

Note StingyCD requires no more computation than CD for nearly all iterations (and often much less). However, the cost of updating **rr** is not negligible. To ensure updates to **rr** do not overly impact convergence times, we schedule reference updates so that StingyCD invests less than 20% of its time in updating **rr**. Specifically, StingyCD first updates **rr** after the second epoch and records the time that this update requires. Later on, **rr** is updated each time an additional 5x of this amount of time has passed.

4.3 Skipping additional updates with StingyCD+

As we will see in §4.6, StingyCD can significantly improve training times. However, StingyCD is also limited by the requirement that only updates *guaranteed* to be zero are skipped. Intuitively, in cases where $q^{(t-1)}$ is only slightly greater than τ_i, the updates will likely be zero too. Perhaps StingyCD should skip these updates as well.

In this section, we propose StingyCD+, an algorithm that also skips many updates that are *not* guaranteed to be zero. To do so, StingyCD+ adds two components to StingyCD:

(i) a computationally inexpensive model of the probability that each update is zero, and (ii) a decision rule that applies this model to determine whether or not to skip each update.

4.3.1 Modeling the probability of nonzero updates

During iteration t of StingyCD, suppose $\omega_i^{(t-1)} = 0$ but $\tau_i < q^{(t-1)}$. StingyCD does not skip update t. For now, let us assume that $\tau_i \geq -q^{(t-1)}$ (otherwise we can guarantee $\mathbf{r}^{(t-1)} \in \mathcal{A}_i$, which implies $\delta^{(t)} \neq 0$). Let $\mathcal{U}^{(t)}$ be a variable that is true iff $\delta^{(t)} \neq 0$, i.e., iff update t is *useful*. From the algorithm's perspective, it is uncertain whether $\mathcal{U}^{(t)}$ is true or false when iteration t begins. Whether or not $\mathcal{U}^{(t)}$ is true depends on whether $\mathbf{r}^{(t-1)} \in \mathcal{A}_i$, which requires $\Theta(\text{NNZ}(\mathbf{A}_i))$ time to test. Such computation is wasteful when $\mathcal{U}^{(t)}$ is false.

StingyCD+ models the probability that $\mathcal{U}^{(t)}$ is true using information available to the algorithm. Specifically, the algorithm knows that $\mathbf{r}^{(t-1)}$ lies on the boundary of $\mathcal{S}^{(t)}$, which is a ball with center $\mathbf{rr}$ and radius $\sqrt{q^{(t-1)}}$. This leads to a simple modeling assumption:

Assumption 4.3 (Distribution of $\mathbf{r}^{(t-1)}$)**.** *To model the probability $P(\mathcal{U}^{(t)})$, StingyCD+ assumes $\mathbf{r}^{(t-1)}$ is uniformly distributed on the boundary of $\mathcal{S}^{(t)}$.*

By making this assumption, $P(\mathcal{U}^{(t)})$ is tractable. In particular, we have the following equation for $P(\mathcal{U}^{(t)})$:

Theorem 4.4 (Equation for $P(\mathcal{U}^{(t)})$)**.** *Assume $\omega_i^{(t-1)} = 0$ and $\tau_i \in [-q^{(t-1)}, q^{(t-1)})$. Then Assumption 4.3 implies*

$$P(\mathcal{U}^{(t)}) = \begin{cases} \frac{1}{2} I_{(1-\tau_i/q^{(t-1)})}(\frac{n-1}{2}, \frac{1}{2}) & \text{if } \tau_i \geq 0, \\ 1 - \frac{1}{2} I_{(1+\tau_i/q^{(t-1)})}(\frac{n-1}{2}, \frac{1}{2}) & \text{otherwise,} \end{cases}$$

where $I_x(a, b)$ is the regularized incomplete beta function.

Included in Appendix C.3, Theorem 4.4's proof calculates the probability that $\mathbf{r}^{(t-1)} \in \mathcal{A}_i$ by dividing the area of $\mathcal{A}_i \cap \text{bd}(\mathcal{S}^{(t)})$ by the area of $\text{bd}(\mathcal{S}^{(t)})$—we illustrate this in Figure 4.2. This fraction is a function of the incomplete beta function, since $\mathcal{A}_i \cap \text{bd}(\mathcal{S}^{(t)})$ is a hyperspherical cap (Li, 2011).

Using Theorem 4.4, StingyCD+ can approximately evaluate $P(\mathcal{U}^{(t)})$ efficiently using a lookup table. Specifically, for 128 values of $x \in (0,1)$, our implementation defines an approximate lookup table for $I_x(\frac{n-1}{2}, \frac{1}{2})$ prior to iteration 1. Before update t, StingyCD+ computes $\tau_i/q^{(t-1)}$ and then finds an appropriate estimate of $P(\mathcal{U}^{(t)})$ using the table.

So far, $P(\mathcal{U}^{(t)})$ models the probability that $\delta^{(t)} \neq 0$ when $\tau_i \in [-q^{(t-1)}, q^{(t-1)})$ and $\omega_i^{(t-1)} = 0$. We can also define $P(\mathcal{U}^{(t)})$ for other $\omega_i^{(t-1)}$ and τ_i. When $\omega_i^{(t-1)} \neq 0$, we define $P(\mathcal{U}^{(t)}) = 1$. If $\tau_i \geq q^{(t-1)}$ and $\omega_i^{(t-1)} = 0$ (the scenario in which StingyCD skips update t), we let $P(\mathcal{U}^{(t)}) = 0$. If $\tau_i < -q^{(t-1)}$ and $\omega_i^{(t-1)} = 0$, we define $P(\mathcal{U}^{(t)}) = 1$ (in this final case, we can show that $\mathcal{S}^{(t)} \subseteq \mathcal{A}_i$, which guarantees $\mathbf{r}^{(t-1)} \in \mathcal{A}_i$ and $\delta^{(t)} \neq 0$).

4.3.2 Decision rule for skipping updates

Given $P(\mathcal{U}^{(t)})$, consider the decision of whether to skip update t. Let t_i^{last} denote the most recent iteration during which StingyCD+ updated (did not skip) coordinate i. If this has not yet occurred, define $t_i^{\text{last}} = 0$. We define the "delay" $D_i^{(t)}$ as the number of updates that StingyCD+ did not skip between iterations t_i^{last} and $t-1$ inclusive.

Our intuition for StingyCD+ is that during iteration t, if $D_i^{(t)}$ is large and $\mathcal{U}^{(t)}$ is true, then StingyCD+ should not skip update t. However, if $D_i^{(t)}$ is small and $\mathcal{U}^{(t)}$ is true, the algorithm may want to skip the update in favor of updating a coordinate with larger delay. Finally, if $\mathcal{U}^{(t)}$ is false, StingyCD+ should skip the update, regardless of $D_i^{(t)}$.

Based on this intuition, StingyCD+ skips update t if the "expected relevant delay," defined as $\mathbb{E}[D_i^{(t)}\mathcal{U}^{(t)}]$, is small. That is, given a threshold $\xi^{(t)}$, StingyCD+ skips update t if

$$P(\mathcal{U}^{(t)})D_i^{(t)} < \xi^{(t)}\,. \tag{4.2}$$

Inserting (4.2) in place of StingyCD's condition for skipping updates is the only change from StingyCD to StingyCD+. In practice, we define $\xi^{(t)} = \text{NNZ}(\boldsymbol{\omega}^{(t-1)})$. Defining $\xi^{(t)}$ in this way leads to the following convergence guarantee:

Theorem 4.5 (Convergence of StingyCD+ to a solution)**.** *Assume* $\xi^{(t)} \leq \text{NNZ}(\boldsymbol{\omega}^{(t-1)})$ *for all*

$t > 0$ in StingyCD+. For each coordinate $i \in \{1, \ldots, m\}$, assume the number of consecutive iterations during which `get_next_coordinate()` *does not return i is bounded. Then*

$$\lim_{t \to \infty} f(\boldsymbol{\omega}^{(t)}) = f(\boldsymbol{\omega}^\star) \,.$$

Proven in Appendix C.4, Theorem 4.5 ensures StingyCD+ convergences to a solution as long as $\xi^{(t)}$ is sufficiently small. At a high level, as long as $\xi^{(t)}$ is not larger than $\mathrm{NNZ}(\boldsymbol{\omega}^{(t-1)})$, at least one coordinate—specifically a coordinate for which $\omega_i^{(t-1)} \neq 0$—will satisfy (4.2) during a future iteration. By defining $\xi^{(t)} = \mathrm{NNZ}(\boldsymbol{\omega}^{(t-1)})$ in practice, we find that StingyCD+ achieves fast training times.

4.4 Extending StingyCD to other problems

For simplicity, we introduced StingyCD for nonnegative lasso problems. In this section, we describe how to apply StingyCD to some other problems.

4.4.1 General (not nonnegative) lasso problems

It is simple to extend StingyCD to problems of the form

$$\underset{\boldsymbol{\omega} \in \mathbb{R}^m}{\text{minimize}} \; f_{\mathrm{L}}(\boldsymbol{\omega}) \,, \quad \text{where} \quad f_{\mathrm{L}}(\boldsymbol{\omega}) := \tfrac{1}{2} \left\| \mathbf{A}\boldsymbol{\omega} - \mathbf{b} \right\|^2 + \lambda \left\| \boldsymbol{\omega} \right\|_1 \,. \tag{PL}$$

We can transform (PL) into an instance of (PNL) by introducing two features (a positive and negative copy) for each column of $\mathbf{A}$. That is, we solve (PL) with design matrix $\mathbf{A}$ by solving (PNL) with design matrix $[\mathbf{A}, -\mathbf{A}]$. Importantly, we perform this duplication implicitly.

To incorporate duplicated features, we first adapt each update $\delta^{(t)}$ to the new objective:

$$\delta^{(t)} \leftarrow \underset{\delta}{\operatorname{argmin}} \; f_{\mathrm{L}}(\boldsymbol{\omega}^{(t-1)} + \delta \mathbf{e}_i) \,.$$

We also account for duplicated features in StingyCD's skip condition. Specifically, let

$$\tau_i^+ \leftarrow \operatorname{sgn}(\lambda - \langle \mathbf{A}_i, \mathbf{r}\mathbf{r} \rangle) \tfrac{(\lambda - \langle \mathbf{A}_i, \mathbf{r}\mathbf{r} \rangle)^2}{\|\mathbf{A}_i\|^2} \,, \quad \text{and} \quad \tau_i^- \leftarrow \operatorname{sgn}(\lambda + \langle \mathbf{A}_i, \mathbf{r}\mathbf{r} \rangle) \tfrac{(\lambda + \langle \mathbf{A}_i, \mathbf{r}\mathbf{r} \rangle)^2}{\|\mathbf{A}_i\|^2} \,.$$

StingyCD skips update t iff $\omega_i^{(t-1)} = 0$ and $q^{(t-1)} \leq \min\{\tau_i^+, \tau_i^-\}$. Modifying StingyCD+ to solve (PL) is similar. $P(\mathcal{U}^{(t)})$ becomes the sum of two probabilities corresponding to features $+\mathbf{A}_i$ and $-\mathbf{A}_i$. Specifically, $P(\mathcal{U}^{(t)}) = P(\mathcal{U}_+^{(t)}) + P(\mathcal{U}_-^{(t)})$. We define $P(\mathcal{U}_+^{(t)})$ and $P(\mathcal{U}_-^{(t)})$ the same way as we define $P(\mathcal{U}^{(t)})$ in §4.3.1, except we use τ_i^+ and τ_i^- in place of τ_i.

4.4.2 General ℓ_1-regularized smooth loss minimization

We can also use StingyCD to solve problems of the form

$$\underset{\boldsymbol{\omega}\in\mathbb{R}^m}{\text{minimize}} \ \sum_{j=1}^n L_j(\langle \mathbf{a}_j, \boldsymbol{\omega}\rangle) + \lambda \left\|\boldsymbol{\omega}\right\|_1, \tag{PL1}$$

where each L_j is smooth. To solve this problem, we redefine $\mathbf{r}^{(t-1)}$ as a vector of derivatives:

$$\mathbf{r}^{(t-1)} = [-L_1'(\langle \mathbf{a}_1, \boldsymbol{\omega}^{(t-1)}\rangle), \ldots, -L_n'(\langle \mathbf{a}_n, \boldsymbol{\omega}^{(t-1)}\rangle)]^T.$$

When updating coordinate i, it remains true that $\delta^{(t)} = 0$ if $\omega_i^{(t-1)} = 0$ and $\mathbf{r}^{(t-1)} \notin \mathcal{A}_i$. Unfortunately, updating $q^{(t-1)}$ no longer requires negligible computation. This is because in general, $\mathbf{r}^{(t)} \neq \mathbf{r}^{(t-1)} - \delta^{(t)}\mathbf{A}_i$. Thus, the update to $q^{(t)}$ in Algorithm 4.2 no longer applies, which means that $q^{(t)} = \left\|\mathbf{r}^{(t)} - \mathbf{rr}\right\|^2$ cannot be computed from $q^{(t-1)}$ using negligible time.

Nevertheless, we can use StingyCD to efficiently solve (PL1) by incorporating StingyCD into a proximal Newton algorithm. During each outer-iteration, we approximate the loss $\sum_j L_j(\langle \mathbf{a}_i, \boldsymbol{\omega}\rangle)$ with a second-order Taylor expansion. This results in a subproblem of the form (PL), which we solve using StingyCD. CD-based proximal Newton methods are known to be very fast for solving (PL1), especially if the loss is logistic loss (Yuan et al., 2012).

4.4.3 Linear support vector machines

We can also apply StingyCD to train linear SVMs. The following is the dual SVM problem:

$$\begin{aligned} \underset{\boldsymbol{\omega}\in\mathbb{R}^m}{\text{minimize}} \quad & \tfrac{1}{2}\left\|\mathbf{M}\boldsymbol{\omega}\right\|^2 - \langle \mathbf{1}, \boldsymbol{\omega}\rangle \\ \text{s.t.} \quad & \boldsymbol{\omega} \in [0, C]^m \end{aligned} \quad . \tag{PSVMD}$$

Note (PSVMD) is much like (PNL). If not for the constraint that $\omega_i \leq C$ for all i, in fact, (PSVMD) would be an instance of (PNL), and we could solve (PSVMD) with Algorithm 4.2 by defining $\mathbf{A} = \mathbf{M}$, $\mathbf{b} = \mathbf{0}$, and $\lambda = -1$.

For (PSVMD), we can use the exact approach from §4.2.3 to guarantee $\delta^{(t)} = 0$ in some cases when $\omega_i^{(t-1)} = 0$. By symmetry, we can develop similar conditions for when $\omega_i^{(t-1)} = C$.

4.5 Relation to prior work

StingyCD is related to safe screening tests as well as alternative strategies for prioritizing coordinate updates in CD.

4.5.1 Safe screening

We discussed safe screening in detail in Chapter 3. In §3.6 especially, we proposed a state-of-the-art safe screening test called BlitzScreen. For the problem (PNL), BlitzScreen relies on geometry similar to Figure 4.1. Specifically, the test defines a ball $\mathcal{S}_1$ that is known to contain the residual vector of a *solution* to (PNL). If $\mathcal{S}_1 \cap \mathcal{A}_i = \emptyset$, it is guaranteed that the ith weight in (PNL)'s solution has value 0.

The radius of $\mathcal{S}_1$ is typically *much* larger than that of $\mathcal{S}^{(t)}$ in StingyCD, however. Unlike $\mathcal{S}^{(t)}$, $\mathcal{S}_1$ must contain the optimal residual vector. Unless a good approximate solution is available already, $\mathcal{S}^1$ is large, and as we observed in Chapter 3, this leads to few screened features. By ensuring only that $\mathcal{S}^{(t)}$ contains the *current* residual vector and identifying zero-valued updates rather than zero-valued entries in the solution, StingyCD is drastically more effective for improving training times.

4.5.2 Other approaches to prioritizing CD updates

Similar to StingyCD, recent work by (Fujiwara et al., 2016) also uses a reference vector concept for prioritizing updates in CD. Unlike StingyCD, this work focuses on identifying nonzero-valued coordinates, which leads to an active set algorithm. The reference vector is also a primal weight vector as opposed to a residual vector.

Similarly, shrinking heuristics (Fan et al., 2008; Yuan et al., 2012) and working set algorithms are effective for prioritizing computation in CD algorithms. One interesting idea is to combine StingyCD with working set algorithms to further prioritize computation. In §4.6, we show that using StingyCD+ instead of CD for solving subproblems in BlitzWS can lead to further speed-ups.

4.6 Empirical comparisons

This section demonstrates the impact of StingyCD and StingyCD+ in practice. We first compare the algorithms to CD and CD + safe screening for lasso problems. Later, we show that StingyCD+ also leads to speed-ups when combined with working set and inexact proximal Newton algorithms. For each experiment, we note there exists additional results in the supplement of our StingyCD conference paper (Johnson and Guestrin, 2017).

4.6.1 Lasso problem comparisons

We implemented CD, CD with safe screening, StingyCD, and StingyCD+ to solve (PL). We update coordinates in round-robin fashion. We normalize columns of $\mathbf{A}$ and include an unregularized intercept term. We also remove features that have nonzero values in fewer than ten training examples. For CD with safe screening, we apply BlitzScreen from Chapter 3. Following (Fercoq et al., 2015), we perform screening after every ten epochs. Performing screening requires a full pass over the data, which adds a non-negligible amount of time.

We compare the algorithms using a financial document dataset (`finance`)[1] and an insurance claim prediction task (`allstate`)[2]. `finance` contains 1.6×10^4 examples, 5.5×10^5 features, and 8.8×10^7 nonzero values. The result included in this section uses regularization $\lambda = 0.05\lambda_{\max}$, where $\lambda_{\max}$ is the smallest λ value that results in an all-zero solution. The solution contains 1746 nonzero entries. The `allstate` dataset contains 2.5×10^5 examples,

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression.html#E2006-log1p`

[2] `https://www.kaggle.com/c/allstate-claims-severity`

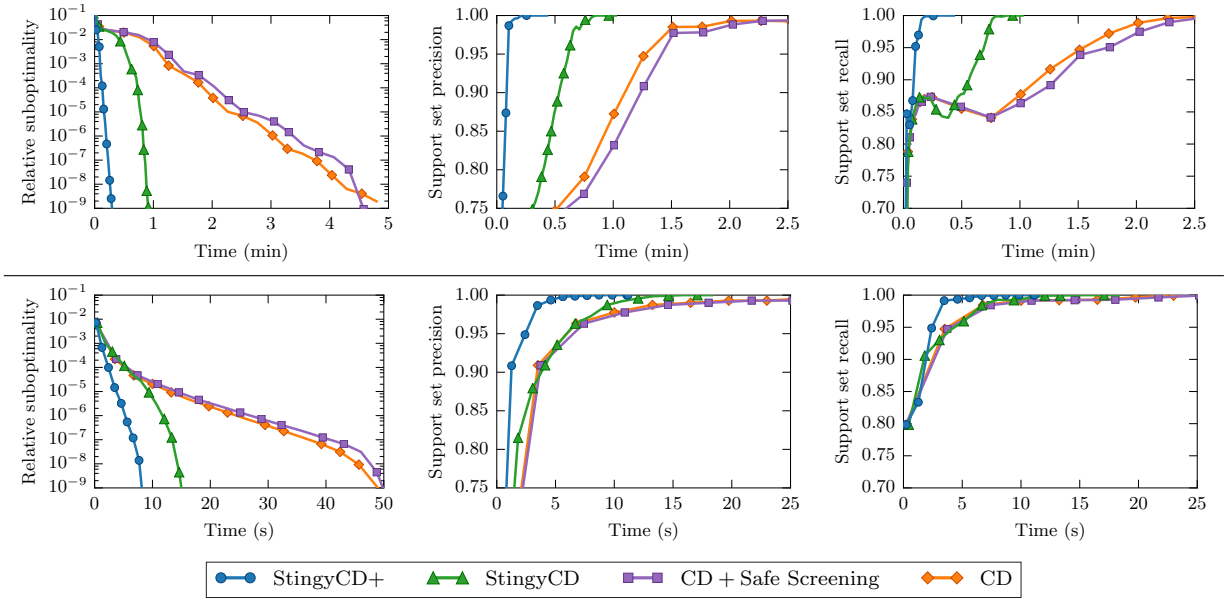


Figure 4.3: Lasso results. *Above:* `finance` dataset. *Below:* `allstate` dataset. Compared to naïve coordinate descent (with and without BlitzScreen), StingyCD trains sparse models much faster. Our StingyCD+ heuristic provides further improvement.

1.5×10^4 features, and 1.2×10^8 nonzero values. For this problem, we set $\lambda = 0.05\lambda_{\max}$, resulting in 1404 selected features.

We evaluate the algorithms with three metrics. The first metric is relative suboptimality:

$$\text{Relative suboptimality} = \frac{f(\boldsymbol{\omega}^{(t)}) - f(\boldsymbol{\omega}^\star)}{f(\boldsymbol{\omega}^\star)},$$

where $f(\boldsymbol{\omega}^{(t)})$ is the objective value at iteration t, and $\boldsymbol{\omega}^\star$ is the problem's solution. The other metrics are support set precision and recall. Let $\mathcal{F}^{(t)} = \{i : \omega_i^{(t)} \neq 0\}$, and let $\mathcal{F}^\star$ be the analogous set for the optimal weight vector. We define

$$\text{Precision} = \frac{\left|\mathcal{F}^{(t)} \cap \mathcal{F}^\star\right|}{\left|\mathcal{F}^{(t)}\right|}, \quad \text{Recall} = \frac{\left|\mathcal{F}^{(t)} \cap \mathcal{F}^\star\right|}{\left|\mathcal{F}^\star\right|}.$$

Precision and recall are arguably at least as important as suboptimality since (PL) is typically used for feature selection.

Results of these experiments are included in Figure 4.3. We see that StingyCD and

StingyCD+ both greatly improve convergence times. For reasons discussed in §4.5, BlitzScreen provides little improvement compared to CD in these cases—even when the relative suboptimality is plotted until 10^{-9}. StingyCD provides a "safeness" similar to safe screening yet with drastically greater impact.

4.6.2 Combining StingyCD+ with working sets

This section demonstrates that StingyCD+ can be useful when combined with other algorithms. We consider the problem of sparse logistic regression, an instance of (PL) in which each L_j term is a logistic loss function.

In this section, we use StingyCD+ as a subproblem solver for a proximal Newton algorithm and a working set algorithm. Specifically, we implement StingyCD+ within "Blitz" from (Johnson and Guestrin, 2015). At each iteration of Blitz, a subproblem is formed by selecting a set of priority features. Blitz then approximately minimizes the objective by updating weights corresponding only to elements in the working set. Blitz solves each subproblem approximately with a proximal Newton algorithm, and each proximal Newton subproblem is solved approximately with CD.

For these comparisons, we have replaced the aforementioned CD implementation with a StingyCD+ implementation. We demonstrate the effects of this change with and without working sets. For the case without working sets, we refer to the algorithm as "StingyCD+ ProxNewton" or "CD ProxNewton," depending on whether StingyCD+ is incorporated. We have not otherwise modified Blitz and the proximal Newton solver.

We compare the algorithms using an educational performance dataset (`kdda`)[3] and a loan default prediction task (`lending_club`)[4]. After removing features with fewer than ten nonzeros, `kdda`'s design matrix contains 8.4×10^6 examples, 2.2×10^6 features, and 2.8×10^8 nonzero values. We solve this problem with $\lambda = 0.005\lambda_{\max}$, which results in 692 nonzero weights at the problem's solution. The `lending_club` data contains 1.1×10^5 examples,

[3] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#kdd2010(algebra)`

[4] `https://www.kaggle.com/wendykan/lending-club-loan-data`

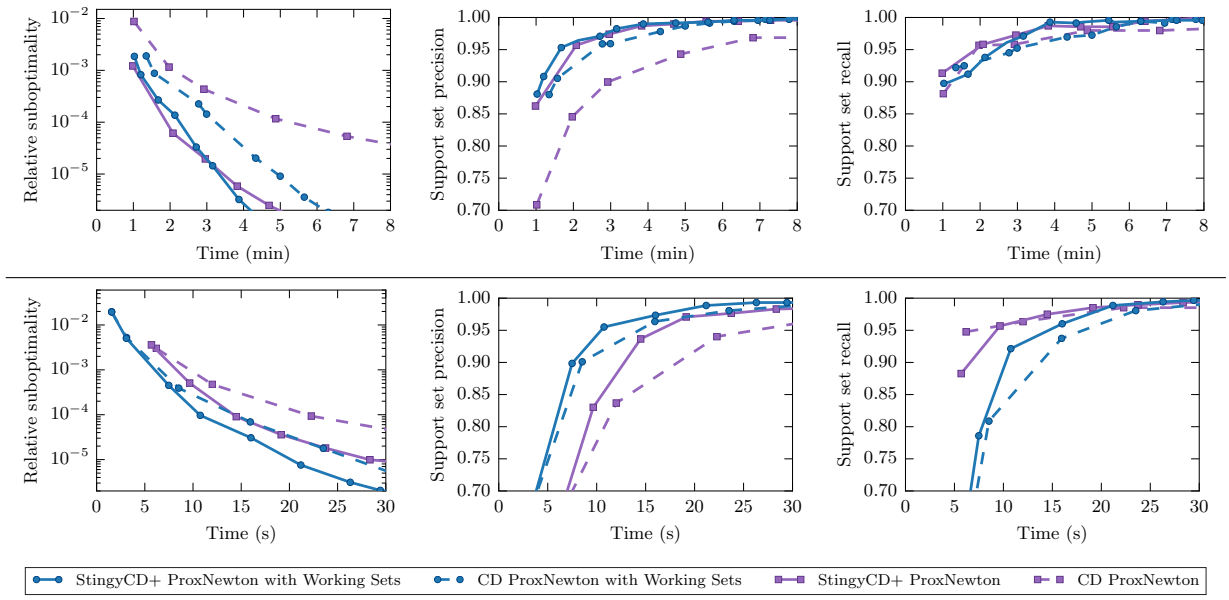


Figure 4.4: Combining StingyCD+ with other algorithms for sparse logistic regression. *Above:* `kdda` dataset. *Below:* `lending_club` dataset. Stingy updates can also improve training times when used as a subproblem solver in more elaborate algorithms. For both BlitzWS and inexact proximal Newton algorithms, StingyCD+ provides immediate advantages over standard CD.

3.1×10^4 features, and 1.0×10^8 nonzero values. We solve this problem with $\lambda = 0.02\lambda_{\max}$, resulting in 878 selected features.

Figure 4.4 displays the results of this experiment. We see that replacing CD with StingyCD+ in both Blitz and ProxNewton can result in immediate efficiency improvements. We note that the amount that StingyCD+ improved upon the working set approach depended significantly on λ, at least in the case of `lending_club`. For this dataset, when λ is relatively large (and thus the solution is very sparse), we observed little or no improvement due to StingyCD+. However, for smaller values of λ, StingyCD+ led to more significant gains. Moreover, StingyCD+ was the best performing algorithm in some cases (though in other cases, Blitz was much faster). This observation suggests that there likely exists a better approach to using working sets with StingyCD+—an ideal algorithm would obtain excellent performance across all relevant λ values.

4.7 Discussion

We proposed StingyCD, a coordinate descent algorithm that avoids large amounts of wasteful computation in applications such as sparse regression. StingyCD borrows geometric ideas from safe screening to guarantee many updates will result in no progress toward convergence. Compared to safe screening, StingyCD achieves considerably greater convergence time speed-ups. We also introduced StingyCD+, which uses a probabilistic assumption to further prioritize coordinate updates.

In general, we find the idea of "stingy updates" to be deserving of significantly more exploration. Currently this idea is limited to CD algorithms and, for the most part, objectives with quadratic losses. However, it seems likely that similar ideas would apply in many other contexts. For example, it could be useful to use stingy updates in distributed optimization algorithms to significantly reduce communication requirements.

Chapter 5

CONCLUSION

We proposed several strategies for training machine learning models faster via prioritized optimization. Figure 5.1 summarizes relations between these contributions. Most of our methods—specifically BlitzWS, BlitzScreen, StingyCD, and StingyCD+—exploit piecewise linear structure in convex problems. This amounts to prioritizing model components (in the case of sparse models) or prioritizing training examples (in the case of piecewise linear loss minimization). In addition, we also proposed RAIS, which prioritizes training examples when learning deep models.

In this section, we first discuss advantages and limitations of our proposed methods. To conclude, we suggest promising directions for future research on prioritized machine learning.

5.1 Achievements and limitations

Table 5.1 organizes the main advantages and disadvantages of our proposed strategies.

For convex problems, our contributions amount to significant new understanding of how to prioritize training. Most notably, we proposed a unique and fast working set algorithm, which prioritizes components in a principled way. Our work on safe screening also makes significant advances, including (i) large improvements on prior safe screening tests, (ii) an intuitive link between working set algorithms and screening tests, and (iii) StingyCD, which shares many similarities with safe screening yet leads to drastically greater speed-ups.

Our work on prioritizing training examples for deep learning also offers valuable contributions. The use of robust optimization to handle uncertainty in importance values seems especially useful.

Of course, our work also has limitations, which we next describe with two general themes.

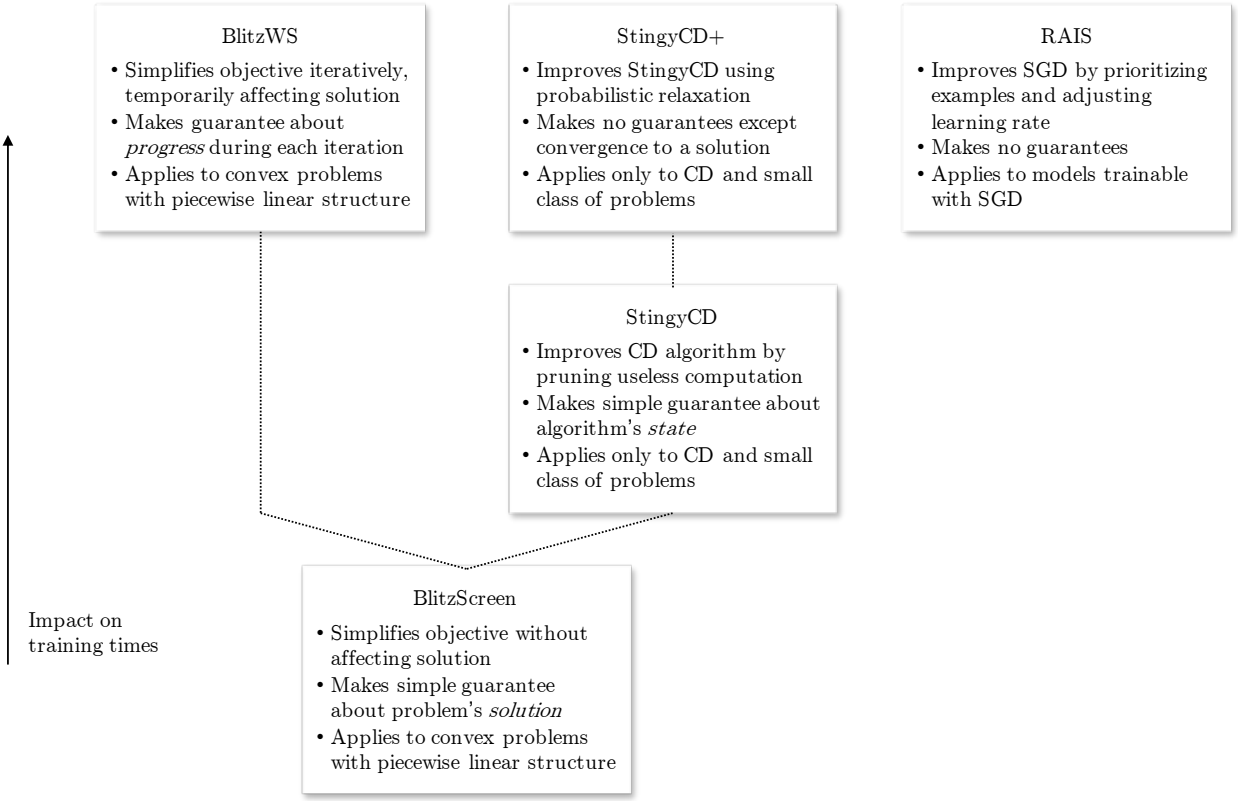


Figure 5.1: Relations between proposed strategies. BlitzWS, BlitzScreen, StingyCD, and StingyCD+ rely on similar structure in convex problems. RAIS makes few assumptions about the objective function and speeds up training in a different way. BlitzScreen and StingyCD have the simplest theoretical guarantees, but these strategies also have limited practical impact.

	RAIS	BlitzWS	BlitzScreen	StingyCD
Practical achievements	Useful importance sampling for deep learning (consistent gains, learning rate adjustment, no critical hyperparameters)	Fast in practice, reasonable choice of subproblems and parameters, compatible with any subproblem solver (even distributed)	Better at simplifying problem than prior screening tests, applicable to larger class of problems	Much more effective at prioritizing computation than BlitzScreen, minimal downside, further improvement with StingyCD+
Practical limitations	More difficult to implement than basic SGD, gains limited when gradient norms unpredictable	Capsule approximation tricky to implement, dependent on piecewise structure being bottleneck, less useful when only few passes over data desired	Not effective until convergence is near, poor scaling with problem size	Limited to smaller class of problems and specific algorithm, StingyCD not as impactful as StingyCD+ or BlitzWS
Theoretical achievements	Less arbitrary than prior approaches (uses robust optimization instead of arbitrary smoothing)	Unique per-iteration progress guarantee for working sets, elegant link to safe screening, some theoretical basis for adapting algorithmic parameters	Domination over prior safe screening tests, applicable to many problems	Simple guarantee for skipping useless computation, sequence of iterates unaffected
Theoretical limitations	No theoretical guarantees (not even convergence)	No guarantee of speed-ups, no overall convergence rate	Dependent on good approximate solution	Applies only to quadratic objectives and CD, few guarantees for StingyCD+, no theory for scheduling reference updates

Table 5.1: Summary of achievements and limitations.

5.1.1 The challenge of identifying what to prioritize

For each problem we considered, we proposed an "ideal" method for prioritizing optimization. For deep learning, our ideal is the oracle algorithm from §2.4. For problems with convex piecewise structure, we ideally would prioritize according to Proposition 3.6.

These ideal methods are impractical, however, as they rely on information that the algorithm cannot access in practice. Instead, we must approximately identify important components.

A recurring theme is that such approximations tend to be difficult initially and easier during later stages of training. During later stages, the model changes more gradually, which makes the recent state of the algorithm more predictive of each component's importance. As a result, our strategies tend to work best during "fine-tuning" stages of training.

Since learning is usually a process with diminishing returns, training algorithms typically spend a disproportionate amount of time fine-tuning. For this reason, strategies such as RAIS and BlitzWS can be quite helpful, as they can drastically accelerate fine-tuning.

At the same time, our strategies must avoid taking too much training time to become useful, as machine learning applications typically require only approximate solutions. Unfortunately, we found that BlitzScreen—and safe screening in general—rarely satisfies this requirement, especially for large training tasks. In fact, we found that screening often has *negligble* impact on training until a very good approximate solution is available. Fortunately, we also found that StingyCD performs much better in comparison, while StingyCD+ and BlitzWS provide even greater impact.

5.1.2 Trade-offs between theoretical understanding and practical impact

To achieve significant speed-ups, BlitzWS, StingyCD+, and RAIS prioritize optimization aggressively. In contrast, BlitzScreen and StingyCD prioritize optimization conservatively, pruning only computation that is *guaranteed* to be unnecessary.

Perhaps not surprisingly, aggressive prioritization seemingly comes at the cost of weaker

theoretical understanding. For example, StingyCD has clear and simple guarantees—every skipped update is a zero update. In contrast, StingyCD+ is a heuristic with only a convergence guarantee. Similarly, theoretical analysis of BlitzScreen is much simpler than analysis of BlitzWS. Moreover, there are some theoretical results missing for BlitzWS that could be useful, such as an overall convergence rate or speed-up guarantees.

It is helpful in research, of course, to have multiple strategies with varying amounts of practicality and principled understanding. Perhaps future work will build upon our contributions, both in terms of practical impact and theoretical guarantees.

5.2 Directions for future research

We conclude with suggestions for future research on prioritized machine learning.

5.2.1 Solutions to overcome limits of variance reduction

Our RAIS procedure is fundamentally a variance reduction technique for SGD. RAIS decreases the variance of the stochastic gradient, which leads to training time improvements.

A current trend, especially in industry, also speeds up training by reducing the stochastic gradient's variance. The mechanism for doing so is not importance sampling, however, but rather parallelism. For example, Goyal et al. (2017) train an image classification model on 256 GPUs with a minibatch size of 8192. Since the stochastic gradient's variance scales inversely with minibatch size, such training setups have a variance reduction effect similar to that of RAIS.

Even with unlimited computation, however, there are limits to the impact of variance reduction. One fundamental limitation is the "generalization gap," which describes the phenomenon of lower test accuracy due to training deep networks with less stochastic gradient variance (LeCun et al., 2012; Keskar et al., 2017). This makes intuitive sense, as deep model training is highly nonconvex, and including noise in iterative updates could be helpful for avoiding bad local minima.

With this generalization gap in mind, it is important to research ideas for scalable training other than straightforward parallelism and variance reduction. Such ideas would likely still rely on SGD and backpropagation. Even so, there could be many creative ways to train in parallel, in a prioritized way, or both. To start, one might examine the generalization gap more methodically in order to better understand the effect of the gradient's variance and SGD's learning rate schedule.

5.2.2 Simultaneous prioritization of training examples and model components

In this work, we considered strategies for prioritizing training examples or model components, but not strategies for prioritizing both simultaneously. Since long training times result from both large datasets and models, perhaps simultaneously prioritizing training examples and model weights could lead to important results.

From the safe screening literature, Shibagaki et al. (2016) proposed a screening test that simultaneously identifies irrelevant features and training examples. We are unaware of other works that fit into this category.

Given the current relevance of deep learning, it is especially interesting to consider strategies for prioritizing deep model components. Even without prioritizing training examples, it is unclear what strategies would work well. To improve training times significantly, is it possible to train specific portions of the model via a specialized procedure?

Previously, one idea we considered was to assign more priority to training the network's top layer. We did not pursue this idea enough to reach a conclusion, however.

5.2.3 Holistic approaches to prioritized machine learning

We limited the scope of this work in a few ways. In all chapters, we formulate our problem as an optimization task. We assume an objective function for supervised learning, and our goal is to compute the objective's minimizer as efficiently as possible. By incorporating additional practical elements into our problem statement, we could build upon our work in many directions.

Beyond supervised learning In machine learning, optimization is relevant for more than supervised learning. We could apply similar ideas to unsupervised problems, reinforcement learning, and other types of problems.

Interestingly, an idea similar to StingyCD works well for pruning computation in Lloyd's algorithm for k-means clustering (Elkan, 2003). Perhaps simple refinements to other common algorithms could result in computational savings.

Prioritized training could also speed up reinforcement learning in multiple ways. Schaul et al. (2016) and Horgan et al. (2018) have already shown the usefulness of prioritized experience replay when training deep RL models. In this case, the algorithm maintains a buffer of past experiences, and the goal is to continue learning from these past experiences in a prioritized way. Since these approaches require arbitrary smoothing parameters, a robust optimization strategy could result in improved algorithms.

Another promising direction is to use prioritization with imitation learning. In this case, the agent learns to imitate an expert agent. Procedures like RAIS could help prioritize training on important cases where the model's policy differs most from the expert's. In a similar way, one might also prioritize the collection of new expert demonstrations.

Prioritized objective functions Our RAIS procedure prioritizes training examples in order to efficiently minimize the average loss over the training set. This objective—i.e., minimizing average loss—scores models in a way that assigns equal priority to all training examples.

In some cases, it may be beneficial to prioritize learning in the objective function itself. As an extreme example, Shalev-Shwartz and Wexler (2016) argue that minimizing the *max* loss over the data can be more effective than minimizing the average loss. When the dataset contains many easy examples, Shalev-Shwartz and Wexler show that optimizing the max loss can be particularly effective.

Minimizing the max loss is challenging, however, as we can no longer use SGD with uniform sampling. Max loss also exacerbates the impact of outlier training examples, especially

mislabeled data. As a result, it seems worthwhile to research new ways of learning with prioritized objectives. Katharopoulos and Fleuret (2017) propose one interesting approach, which uses a hybrid of max and average loss. For optimization, Katharopoulos and Fleuret use SGD with approximate importance sampling, and so our robust optimization ideas from Chapter 2 may again be helpful.

Prioritizing with nearly infinite data RAIS requires multiple epochs of SGD in order to train the uncertainty set. Consider the case, however, that our dataset contains enough training examples that learning requires just one epoch (or even less). In this case, RAIS is not useful.

It is interesting to consider whether we can still prioritize training examples in such cases. In particular, it may be worthwhile to revisit challenging examples, even before considering new examples that the model has yet to train on. This idea is similar to the concept of prioritized experience replay for reinforcement learning (Schaul et al., 2016).

Prioritizing with parallelism Our last direction for future work is to design prioritized algorithms specifically with parallelism in mind. We can use most strategies in this work with or without parallelism, although parallelism was not a primary design consideration.

With multiple processing elements, it becomes unclear how best to prioritize optimization. Alain et al. (2016), for example, use a cluster of GPUs to compute importance sampling values, and a single GPU uses these values to train a model with SGD and importance sampling. Alternatively, we could use all GPUs in a distributed SGD algorithm without prioritizing examples at all. Finally, we could interpolate between these algorithms or divide work a different way entirely.

The question of how best to use parallel computing resources for training is interesting and open-ended. Given the success of prioritized optimization in this work, it seems likely that prioritization could play an important role in answering this question as well.

Appendix A

SUPPLEMENTAL MATERIAL FOR CHAPTER 2

A.1 Proof of Proposition 2.1

This appendix shows work for deriving (2.4) and the oracle importance sampling distribution.

A.1.1 Equation for gradient second moment

To derive (2.4), first define

$$\nabla f_{\mathcal{M}^{(t)}}(\mathbf{w}^{(t)}) = \mathbf{g}_{\mathrm{O}}^{(t)} - \lambda \mathbf{w}^{(t)} \,.$$

We have

$$\begin{aligned}
\mathbb{E}\left[\|\mathbf{g}_{\mathrm{O}}^{(t)}\|^2\right] &= \mathbb{E}\left[\|\mathbf{g}_{\mathrm{O}}^{(t)} - \nabla \bar{f}(\mathbf{w}^{(t)}) + \nabla \bar{f}(\mathbf{w}^{(t)})\|^2\right] \\
&= \mathbb{E}\left[\|\nabla f_{\mathcal{M}^{(t)}}(\mathbf{w}^{(t)}) - \bar{f}(\mathbf{w}^{(t)}) + \nabla F(\mathbf{w}^{(t)})\|^2\right] \\
&= \mathbb{E}\left[\|\nabla f_{\mathcal{M}^{(t)}}(\mathbf{w}^{(t)}) - \bar{f}(\mathbf{w}^{(t)})\|^2\right] + \|\nabla F(\mathbf{w}^{(t)})\|^2 \,. \qquad \text{(A.1)}
\end{aligned}$$

Above, we used the fact that $\mathbb{E}\left[\nabla f_{\mathcal{M}^{(t)}}(\mathbf{w}^{(t)})\right] = \bar{f}(\mathbf{w}^{(t)})$.

Continuing, we have

$$\begin{aligned}
\mathbb{E}\left[\|\nabla f_{\mathcal{M}^{(t)}}(\mathbf{w}^{(t)}) - \bar{f}(\mathbf{w}^{(t)})\|^2\right] &= \mathbb{E}\left[\left\|\tfrac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}^{(t)}} \tfrac{1}{np_i^{(t)}}\nabla f_i(\mathbf{w}^{(t)}) - \bar{f}(\mathbf{w}^{(t)})\right\|^2\right] \\
&= \tfrac{1}{|\mathcal{M}|^2}\mathbb{E}\left[\left\|\sum_{i\in\mathcal{M}^{(t)}}\left(\tfrac{1}{np_i^{(t)}}\nabla f_i(\mathbf{w}^{(t)}) - \bar{f}(\mathbf{w}^{(t)})\right)\right\|^2\right] \\
&= \tfrac{1}{|\mathcal{M}|}\mathbb{E}\left[\left\|\tfrac{1}{np_i^{(t)}}\nabla f_i(\mathbf{w}^{(t)}) - \bar{f}(\mathbf{w}^{(t)})\right\|^2\right]
\end{aligned}$$

$$= \frac{1}{|\mathcal{M}|}\left(\mathbb{E}\left[\left\|\frac{1}{np_i^{(t)}}\nabla f_i(\mathbf{w}^{(t)})\right\|^2\right] - \left\|\bar{f}(\mathbf{w}^{(t)})\right\|^2\right).$$

Combining with (A.1) leads to the result:

$$\mathbb{E}\left[\|\mathbf{g}_{\mathrm{O}}^{(t)}\|^2\right] = \frac{1}{|\mathcal{M}|n^2}\sum_{i=1}^{n}\frac{1}{p_i^{(t)}}\left\|\nabla f_i(\mathbf{w}^{(t)})\right\|^2 - \frac{1}{|\mathcal{M}|}\left\|\bar{f}(\mathbf{w}^{(t)})^2\right\| + \left\|\nabla F(\mathbf{w}^{(t)})\right\|^2.$$

A.1.2 Finding the optimal sampling distribution

We want to find the distribution $\mathbf{p}$ that minimizes $\sum_{i=1}^{n} p_i^{-1}(v_i^*)^2$, where $v_i^* = \left\|\nabla f_i(\mathbf{w}^{(t)})\right\|$. From Jensen's inequality, it follows that

$$\sum_{i=1}^{n} p_i^{-1}(v_i^*)^2 = \sum_{i=1}^{n} p_i\left(\frac{v_i^*}{p_i}\right)^2 \geq \left(\sum_{i=1}^{n} v_i^*\right)^2.$$

When $p_i = v_i^*/\sum_j v_j^*$, the inequality is satisfied with equality. Thus, this choice for the sampling distribution is optimal.

A.2 Details of solving (PRC) and (PT)

This appendix provides details of solving the optimization problems in RIAS—specifically solving the robust counterpart and training the uncertainty set.

A.2.1 Justification that the minimax theorem applies to (PRC)

We use the change of variables $u_i = v_i^2$. Define

$$\tilde{\mathcal{U}}^{(t)} = \left\{\mathbf{u} \in \mathbb{R}^n_{\geq 0} \mid \frac{1}{n}\sum_{i=1}^{n}\frac{(\langle \mathbf{c}, \mathbf{s}_i^{(t)}\rangle - \sqrt{u_i})^2}{\langle \mathbf{d}, \mathbf{s}_i^{(t)}\rangle} \leq 1\right\}.$$

Note (i) $\mathbf{u} \in \tilde{\mathcal{U}}^{(t)} \Leftrightarrow \mathbf{v} \in \mathcal{U}^{(t)}$, and (ii) $\tilde{\mathcal{U}}^{(t)}$ is compact and convex, since $\langle \mathbf{c}, \mathbf{s}_i^{(t)}\rangle \geq 0$ and $\langle \mathbf{d}, \mathbf{s}_i^{(t)}\rangle > 0$ for all $i \in [n]$. Denoting the set $\mathcal{P} = \{\mathbf{p} \in \mathbb{R}^n_{>0} \mid \sum_{i=1}^{n} p_i = 1\}$, the robust

counterpart is

$$\inf_{\mathbf{p}\in\mathcal{P}} \max_{\mathbf{u}\in\tilde{\mathcal{U}}^{(t)}} \sum_{i=1}^{n} \tfrac{1}{np_i} u_i \,.$$

This objective is separately concave in $\mathbf{u}$ and convex in $\mathbf{p}$. Thus, we have satisfied the conditions for Sion's minimax theorem.

A.2.2 Approach to solving (PT)

We need to solve

$$\begin{array}{ll} \underset{\mathbf{c},\mathbf{d}\geq 0}{\text{minimize}} & \sum_{i=1}^{n} \langle \mathbf{d}, \mathbf{s}_i^{(t)} \rangle \\ \text{s.t.} & \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \tilde{w}_i \frac{(\langle \mathbf{c}, \tilde{\mathbf{s}}_i \rangle - \tilde{v}_i)^2}{\langle \mathbf{d}, \tilde{\mathbf{s}}_i \rangle} \leq 1 \,. \end{array}$$

We reduce the problem to the unconstrained problem

$$\underset{\mathbf{c},\mathbf{d}\geq 0}{\text{minimize}} \; \frac{1}{n} \sum_{i=1}^{n} \langle \mathbf{d}, \mathbf{s}_i^{(t)} \rangle + \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \tilde{w}_i \frac{(\langle \mathbf{c}, \tilde{\mathbf{s}}_i \rangle - \tilde{v}_i)^2}{\langle \mathbf{d}, \tilde{\mathbf{s}}_i \rangle} \,.$$

We minimize this objective using alternating minimization. Each separate update to $\mathbf{c}$ and $\mathbf{d}$ is a Newton step that we compute with a nonnegative least squares solver.

Appendix B

PROOFS FOR CHAPTER 3

B.1 Proof of Lemma 3.1

Lemma 3.1. *Assume $\alpha_t > 0$, and define $\beta_t = \alpha_t(1+\alpha_t)^{-1}$. Assume that $\mathcal{W}_t$ includes all constraints that are active at $\mathbf{x}_{t-1}$. Then we have*

$$\Delta_t \leq \tfrac{1-2\beta_t}{1-\beta_t}\left[\Delta_{t-1} - \tfrac{1-\beta_t}{\beta_t^2}\tfrac{1}{2}\left\|\mathbf{y}_t - \beta_t\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\right\|^2 - \beta_t\tfrac{1}{2}\left\|\mathbf{x}_{t-1}-\mathbf{y}_{t-1}\right\|^2\right]. \tag{3.4}$$

Proof. We start with the definition of $\mathbf{y}_t$:

$$\begin{aligned}\Delta_t &= \psi_{\mathrm{MN}}(\mathbf{y}_t) - \psi_{\mathrm{MN}}(\mathbf{x}_t)\\ &= \psi_{\mathrm{MN}}(\alpha_t\mathbf{x}_t + (1-\alpha_t)\mathbf{y}_{t-1}) - \psi_{\mathrm{MN}}(\mathbf{x}_t)\\ &= (1-\alpha_t)\left[\Delta_{t-1} - \tfrac{1}{2}\alpha_t\left\|\mathbf{x}_t-\mathbf{y}_{t-1}\right\|^2 - \left[\psi_{\mathrm{MN}}(\mathbf{x}_t) - \psi_{\mathrm{MN}}(\mathbf{x}_{t-1})\right]\right].\end{aligned} \tag{B.1}$$

Since ψ_{MN} is 1-strongly convex,

$$\begin{aligned}&\psi_{\mathrm{MN}}(\mathbf{x}_t) \geq \psi_{\mathrm{MN}}(\mathbf{x}_{t-1}) + \langle\nabla\psi_{\mathrm{MN}}(\mathbf{x}_{t-1}), \mathbf{x}_t - \mathbf{x}_{t-1}\rangle + \tfrac{1}{2}\left\|\mathbf{x}_t-\mathbf{x}_{t-1}\right\|^2\\ \Rightarrow\ &\psi_{\mathrm{MN}}(\mathbf{x}_t) - \psi_{\mathrm{MN}}(\mathbf{x}_{t-1}) \geq \tfrac{1}{2}\left\|\mathbf{x}_t-\mathbf{x}_{t-1}\right\|^2.\end{aligned} \tag{B.2}$$

Above, we have used the fact that $\langle\nabla\psi_{\mathrm{MN}}(\mathbf{x}_{t-1}), \mathbf{x}_t - \mathbf{x}_{t-1}\rangle \geq 0$, which must be true because $\psi_{\mathrm{MN}}(\mathbf{x}_{t-1}) \leq \psi_{\mathrm{MN}}(\mathbf{x}_t)$. Combining (B.2) with (B.1), we have

$$\Delta_t \leq (1-\alpha_t)\left[\Delta_{t-1} - \tfrac{1}{2}\alpha_t\left\|\mathbf{x}_t-\mathbf{y}_{t-1}\right\|^2 - \tfrac{1}{2}\left\|\mathbf{x}_t-\mathbf{x}_{t-1}\right\|^2\right]. \tag{B.3}$$

Next, we use the algebraic fact

$$\alpha_t\left\|\mathbf{x}_t-\mathbf{y}_{t-1}\right\|^2 + \left\|\mathbf{x}_t-\mathbf{x}_{t-1}\right\|^2 = (1+\alpha_t)\left\|\mathbf{x}_t - \tfrac{\mathbf{x}_{t-1}+\alpha_t\mathbf{y}_{t-1}}{1+\alpha_t}\right\|^2 + \tfrac{\alpha_t}{1+\alpha_t}\left\|\mathbf{x}_{t-1}-\mathbf{y}_{t-1}\right\|^2. \tag{B.4}$$

To simplify notation, we define $d_{t-1} = \|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\|$. Applying the assumption that $\alpha_t > 0$, we can write $\mathbf{x}_t = \frac{\mathbf{y}_t - (1-\alpha_t)\mathbf{y}_{t-1}}{\alpha_t}$. Substituting this equality into (B.4), we have

$$\begin{aligned}\alpha_t \|\mathbf{x}_t - \mathbf{y}_{t-1}\|^2 + \|\mathbf{x}_t - \mathbf{x}_{t-1}\|^2 &= (1+\alpha_t)\left\|\tfrac{\mathbf{y}_t-(1-\alpha_t)\mathbf{y}_{t-1}}{\alpha_t} - \tfrac{\mathbf{x}_{t-1}+\alpha_t\mathbf{y}_{t-1}}{1+\alpha_t}\right\|^2 + \tfrac{\alpha_t}{1+\alpha_t}d_{t-1}^2 \\ &= \tfrac{1+\alpha_t}{\alpha_t^2}\left\|\mathbf{y}_t - \tfrac{\alpha_t\mathbf{x}_{t-1}+\mathbf{y}_{t-1}}{1+\alpha_t}\right\|^2 + \tfrac{\alpha_t}{1+\alpha_t}d_{t-1}^2\,. \qquad \text{(B.5)}\end{aligned}$$

Inserting (B.5) into (B.3), we see that

$$\Delta_t \le (1-\alpha_t)\left[\Delta_{t-1} - \tfrac{1+\alpha_t}{\alpha_t^2}\tfrac{1}{2}\left\|\mathbf{y}_t - \tfrac{\alpha_t\mathbf{x}_{t-1}+\mathbf{y}_{t-1}}{1+\alpha_t}\right\|^2 - \tfrac{\alpha_t}{1+\alpha_t}\tfrac{1}{2}d_{t-1}^2\right].$$

Using the fact $\beta_t = \alpha_t(1+\alpha_t)^{-1}$, we can plug in $\alpha_t = \beta_t(1-\beta_t)^{-1}$ to complete the proof. □

B.2 Proof of Lemma 3.2

Lemma 3.2. *Assume β_t is known when selecting $\mathcal{W}_t$, and assume $\beta_t > 0$. For all $i \in [m]$, let $\mathcal{W}_t$ include constraint i if either $\{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x}\rangle \ge b_i\} \cap \mathcal{B}_\xi(\beta_t) \ne \emptyset$ or $\langle \mathbf{a}_i, \mathbf{x}_{t-1}\rangle = b_i$. Then*

$$\Delta_t \le (1-\xi_t)\Delta_{t-1}\,.$$

Proof. If $\beta_t = \frac{1}{2}$, we have $\Delta_t = 0$ by Lemma 3.1. The bound holds in this case because $\Delta_{t-1}(1-\xi_t) \ge 0$. For the remainder of the proof, we assume that $\beta_t < \frac{1}{2}$, which implies that $\alpha_t < 1$.

Since $\alpha_t < 1$, there exists a constraint $i_{\text{limit}} \notin \mathcal{W}_t$ for which $\langle \mathbf{a}_{i_{\text{limit}}}, \mathbf{y}_t\rangle = b_i$. Since $i_{\text{limit}} \notin \mathcal{W}_t$, we must have $\mathcal{B}_\xi(\beta_t) \cap \{\mathbf{x} : \langle \mathbf{a}_{i_{\text{limit}}}, \mathbf{x}\rangle \ge b_i\} = \emptyset$. Thus, $\mathbf{y}_t \notin \mathcal{B}_\xi(\beta_t)$. Since $\mathcal{B}_\xi(\beta_t)$ is a ball with center $\beta_t\mathbf{x}_{t-1} + (1-\beta_t)\mathbf{y}_{t-1}$ and radius $\tau_\xi(\beta_t)$, this implies that

$$\|\mathbf{y}_t - \beta_t\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\| \ge \tau_\xi(\beta_t)\,.$$

To simplify notation, we define $d_{t-1} = \|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\|$. Combining with Lemma 3.1 and

plugging in the definition of $\tau_\xi(\beta_t)$, we have

$$\begin{aligned}\Delta_t &\le \tfrac{1-2\beta_t}{1-\beta_t}\left[\Delta_{t-1} - \tfrac{1-\beta_t}{{\beta_t}^2}\tfrac{1}{2}\tau_\xi(\beta_t)^2 - \beta_t\tfrac{1}{2}d_{t-1}^2\right] \\ &= \tfrac{1-2\beta_t}{1-\beta_t}\left[\Delta_{t-1} - (1-\beta_t)\Delta_{t-1}\left[1 + \tfrac{\beta_t}{1-\beta_t}\left(1 - \tfrac{d_{t-1}^2}{2\Delta_{t-1}}\right) - \tfrac{1-\xi_t}{1-2\beta_t}\right]_+ - \beta_t\tfrac{1}{2}d_{t-1}^2\right] \\ &\le \tfrac{1-2\beta_t}{1-\beta_t}\left[\Delta_{t-1}\tfrac{(1-\xi_t)(1-\beta_t)}{1-2\beta_t}\right] \\ &= (1-\xi_t)\Delta_{t-1}\,.\end{aligned} \tag{B.6}$$

□

B.3 Proof of Theorem 3.3

Theorem 3.3 (Guaranteed progress during iteration t of BlitzMN)**.** *During iteration t of BlitzMN, consider any progress coefficient $\xi_t \in (0, 1]$. For all $i \in [m]$, let $\mathcal{W}_t$ include constraint i if either $\mathcal{S}_\xi \cap \{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x}\rangle \ge b_i\} \neq \emptyset$ or $\langle \mathbf{a}_i, \mathbf{x}_{t-1}\rangle = b_i$. Then*

$$\Delta_t \le (1-\xi_t)\Delta_{t-1}\,.$$

The proof can be divided into three cases: $\beta_t = 1/2$, $\beta_t \in (0, 1/2)$, and $\beta_t = 0$. Here we present the proof of Theorem 3.3 for only the main case that $\beta_t \in (0, 1/2)$, and we rely on the proof in Appendix B.6 (a more general proof) for the edge cases.

Partial proof. Assuming $\beta_t < 1/2$ implies $\alpha_t < 1$. This implies there exists a $i_{\text{limit}} \notin \mathcal{W}_t$ such that $\langle \mathbf{a}_{i_{\text{limit}}}, \mathbf{y}_t\rangle = b_i$. Since $i_{\text{limit}} \notin \mathcal{W}_t$, we must also have $\mathcal{S}_\xi \cap \{\mathbf{x} : \langle \mathbf{a}_{i_{\text{limit}}}, \mathbf{x}\rangle \ge b_i\} = \emptyset$, which implies $\mathbf{y}_t \notin \mathcal{S}_\xi$. Since $\mathbf{y}_t \notin \mathcal{S}_\xi$, then for all $\beta \in (0, 1/2)$, we have $\mathbf{y}_t \notin \mathcal{B}_\xi(\beta)$. Applying the definition of $\mathcal{B}_\xi(\beta)$, we have $\|\mathbf{y}_t - \beta\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\| \ge \tau_\xi(\beta)$ for all $\beta \in (0, 1/2)$. Thus,

$$\|\mathbf{y}_t - \beta_t\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\| \ge \tau_\xi(\beta_t)\,. \tag{B.7}$$

At this point, we can combine (B.7) with Lemma 3.1 to achieve the desired bound. The result follows from the same steps as Lemma 3.2's proof, starting at (B.6). □

B.4 Proof of Theorem 3.4

Theorem 3.4 (Computing capsule parameters is quasiconcave)**.** *For each $s \in \{-1, 0, +1\}$, the function $q_s(\beta) = s\beta \|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\| + \tau_\xi(\beta)$ is quasiconcave over $\{\beta \mid q_s(\beta) > 0\}$. Thus, $d_{\min}^{\text{cap}}$, $d_{\max}^{\text{cap}}$, and r^{cap} are suprema of 1-D quasiconcave functions.*

Proof. Plugging in definitions of q_s and τ_β, we have

$$q_s(\beta) = s\beta \|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\| + \beta\sqrt{2\Delta_{t-1}} \left[1 + \tfrac{\beta}{1-\beta}\left(1 - \tfrac{\|\mathbf{x}_{t-1}-\mathbf{y}_{t-1}\|^2}{2\Delta_{t-1}}\right) - \tfrac{1-\xi_t}{1-2\beta}\right]_+^{1/2} .$$

To simplify notation, define $d_{t-1} = \|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\|$. Taking the log and substituting $\beta = \theta(1+\theta)^{-1}$, we have

$$\tilde{q}_s(\theta) = \log\left(\tfrac{\theta}{1+\theta}\right) + \log\left(sd_{t-1} + \sqrt{2\Delta_{t-1}}\left[1 + \theta\left(1 - \tfrac{d_{t-1}^2}{2\Delta_{t-1}}\right) - (1-\xi)\tfrac{1+\theta}{1-\theta}\right]_+^{1/2}\right) .$$

Since $\frac{\theta}{1+\theta}$ is concave, θ is concave, $-\frac{1+\theta}{1-\theta}$ is concave, $[\cdot]^{1/2}$ is concave and nondecreasing, and $\log(\cdot)$ is concave and nondecreasing, we see that $\tilde{q}_s(\theta)$ is log-concave on $\{\theta : \tilde{q}_s(\theta) > 0\}$. (Here we have also used the facts that $1 - \frac{d_{t-1}^2}{2\Delta_{t-1}} \geq 0$ and $(1-\xi) \geq 0$.) Since all log-concave functions are quasiconcave and quasiconcavity is preserved under composition with the increasing function $\theta = \beta(1-\beta)^{-1}$ (on the domain $0 < \beta \leq 1/2$), it must be the case that $q_s(\beta)$ is quasiconcave.

□

B.5 Proof of Theorem 3.5

Theorem 3.5 (Convergence bound for BlitzMN)**.** *For any iteration T of Algorithm 3.2, define the suboptimality gap $\Delta_T = \psi_{\text{MN}}(\mathbf{y}_T) - \psi_{\text{MN}}(\mathbf{x}_T)$. For all $T > 0$, we have*

$$\Delta_T \leq \Delta_0 \prod_{t=1}^{T} (1 - \xi_t) .$$

Proof. In §B.9.1, we prove that $\mathcal{S}_\xi \subseteq \mathcal{S}_\xi^{\text{cap}}$. Since $\mathcal{S}_\xi \subseteq \mathcal{S}_\xi^{\text{cap}}$, we have

$$\mathcal{S}_\xi \cap \{\mathbf{x} : |\langle \mathbf{A}_i, \mathbf{x}\rangle| \geq \lambda\} \neq \emptyset \;\Rightarrow\; \mathcal{S}_\xi^{\text{cap}} \cap \{\mathbf{x} : |\langle \mathbf{A}_i, \mathbf{x}\rangle| \geq \lambda\} \neq \emptyset .$$

Thus, during iteration t of Algorithm 3.2, condition (i) for Theorem 3.3 is satisfied. That is, for any $i \in [m]$, if $\mathcal{S}_\xi \cap \{\mathbf{x} : |\langle \mathbf{A}_i, \mathbf{x}\rangle| \geq \lambda\} \neq \emptyset$, then $i \in \mathcal{W}_t$. Since condition (ii) of the theorem is satisfied by our definition of Algorithm 3.2, we have by Theorem 3.3, $\Delta_t \leq (1-\xi_t)\Delta_{t-1}$ for all $t \geq 1$. The theorem then follows from induction. □

B.6 Proof of Theorem 3.7 and Theorem 3.8

Since Theorem 3.7 is a special case of Theorem 3.8, we prove both theorems by proving Theorem 3.8. To recover Theorem 3.7, we define $\epsilon_t = 0$, $f_t^{\text{LB}} = f_t$, and $\mathbf{x}_t = \mathbf{z}_t = \operatorname{argmin}_{\mathbf{x}} f_t(\mathbf{x})$.

Theorem 3.8 (Convergence bound for BlitzWS with approximate subproblem solutions)**.** *Consider BlitzWS with approximate subproblem solutions. For any iteration T, define the suboptimality gap $\Delta_T = f(\mathbf{y}_T) - f_T^{\text{LB}}(\mathbf{x}_T)$. For all $T > 0$, we have*

$$\Delta_T \leq \Delta_0 \prod_{t=1}^{T} (1 - (1-\epsilon_t)\xi_t)\,.$$

Proof. We will prove that for all $t > 0$, we have

$$\Delta_t \leq (1 - (1-\epsilon_t)\xi_t)\Delta_{t-1}\,. \tag{B.8}$$

To prove (B.8) for any $t > 0$, let us define the scalar

$$\theta_t = \max\{\theta \in [0,1] : \theta\mathbf{z}_t + (1-\theta)\mathbf{y}_{t-1} \in \text{cl}(\mathcal{S}_\xi^{\text{cap}})\}$$

and point $\mathbf{y}'_t = \theta_t\mathbf{z}_t + (1-\theta_t)\mathbf{y}_{t-1}$. Above, $\text{cl}(\cdot)$ denotes the closure of a set. Note $\mathbf{y}_{t-1} \in \text{cl}(\mathcal{S}_\xi^{\text{cap}})$.

Since $\mathbf{y}_t$ minimizes f along $[\mathbf{y}_{t-1}, \mathbf{z}_t]$, it follows that $f(\mathbf{y}_t) \leq f(\mathbf{y}'_t)$. Due to (C1), we have that $f_t(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{S}_\xi^{\text{cap}}$. Since $\mathbf{y}'_t \in \text{cl}(\mathcal{S}_\xi^{\text{cap}})$ and f is convex lower semicontinuous, it follows that $f_t(\mathbf{y}'_t) = f(\mathbf{y}'_t)$. Beginning with the definition of Δ_t, we can write

$$\Delta_t = f(\mathbf{y}_t) - f_t^{\text{LB}}(\mathbf{x}_t) \leq f(\mathbf{y}'_t) - f_t^{\text{LB}}(\mathbf{x}_t) = f_t(\mathbf{y}'_t) - f_t^{\text{LB}}(\mathbf{x}_t)\,.$$

We divide the remainder of the proof into three cases.

Case 1: $\theta_t = 1$ In this case, $\mathbf{y}'_t = \mathbf{z}_t$, and it follows that

$$\Delta_t \leq f_t(\mathbf{y}'_t) - f_t^{\mathrm{LB}}(\mathbf{x}_t) = f_t(\mathbf{z}_t) - f_t^{\mathrm{LB}}(\mathbf{x}_t) \leq \epsilon_t \Delta_{t-1} \leq (1 - (1-\epsilon_t)\xi_t)\Delta_{t-1}\,.$$

Above, the second-to-last step results from termination conditions for subproblem t, while the final step is true because $\xi_t \in (0, 1]$.

Case 2: $\theta_t \in (0, 1)$ Applying the definition of $\mathbf{y}'_t$, the fact that f_t is 1-strongly convex, the fact that $f_t(\mathbf{x}) \leq f(\mathbf{x})$ for all $\mathbf{x}$, and the definition of Δ_{t-1}, we have

$$\begin{aligned}
\Delta_t &\leq f_t(\mathbf{y}'_t) - f_t^{\mathrm{LB}}(\mathbf{x}_t) \\
&= f_t(\theta_t \mathbf{z}_t + (1-\theta_t)\mathbf{y}_{t-1}) - f_t^{\mathrm{LB}}(\mathbf{x}_t) \\
&\leq \theta_t f_t(\mathbf{z}_t) + (1-\theta_t) f_t(\mathbf{y}_{t-1}) - \tfrac{1}{2}(1-\theta_t)\theta_t \left\|\mathbf{z}_t - \mathbf{y}_{t-1}\right\|^2 - f_t^{\mathrm{LB}}(\mathbf{x}_t) \\
&\leq \theta_t f_t(\mathbf{z}_t) + (1-\theta_t) f(\mathbf{y}_{t-1}) - \tfrac{1}{2}(1-\theta_t)\theta_t \left\|\mathbf{z}_t - \mathbf{y}_{t-1}\right\|^2 - f_t^{\mathrm{LB}}(\mathbf{x}_t) \\
&= (1-\theta_t)\Delta_{t-1} - (1-\theta_t)[f_t^{\mathrm{LB}}(\mathbf{x}_t) - f_{t-1}^{\mathrm{LB}}(\mathbf{x}_{t-1})] + \\
&\qquad\qquad \theta_t[f_t(\mathbf{z}_t) - f_t^{\mathrm{LB}}(\mathbf{x}_t)] - \tfrac{1}{2}(1-\theta_t)\theta_t \left\|\mathbf{z}_t - \mathbf{y}_{t-1}\right\|^2 .
\end{aligned}$$

From termination conditions for subproblem t, we have that $f_t(\mathbf{z}_t) - f_t^{\mathrm{LB}}(\mathbf{x}_t) \leq \epsilon_t \Delta_{t-1}$ and also that $f_t^{\mathrm{LB}}(\mathbf{x}_t) - f_{t-1}^{\mathrm{LB}}(\mathbf{x}_{t-1}) \geq (1-\epsilon_t)\frac{1}{2}\left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2$. Thus,

$$\begin{aligned}
\Delta_t &\leq (1-\theta_t)\Delta_{t-1} - (1-\theta_t)(1-\epsilon_t)\tfrac{1}{2}\left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2 + \theta_t\epsilon_t\Delta_{t-1} - \tfrac{1}{2}(1-\theta_t)\theta_t\left\|\mathbf{z}_t - \mathbf{y}_{t-1}\right\|^2 \\
&\leq \Delta_{t-1} - (1-\epsilon_t)\left[\theta_t\Delta_{t-1} + \tfrac{1}{2}(1-\theta_t)\left(\theta_t\left\|\mathbf{z}_t - \mathbf{y}_{t-1}\right\|^2 + \left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2\right)\right]\,. \qquad \text{(B.9)}
\end{aligned}$$

We next use the fact

$$\theta_t\left\|\mathbf{z}_t - \mathbf{y}_{t-1}\right\|^2 + \left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2 = (1+\theta_t)\left\|\mathbf{z}_t - \tfrac{\mathbf{x}_{t-1} + \theta_t\mathbf{y}_{t-1}}{1+\theta_t}\right\|^2 + \tfrac{\theta_t}{1+\theta_t}\left\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\right\|^2\,. \quad \text{(B.10)}$$

To simplify notation slightly, we define $d_{t-1} = \left\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\right\|$. Applying the assumption that

$\theta_t > 0$, we can write $\mathbf{z}_t = \frac{\mathbf{y}'_t - (1-\theta_t)\mathbf{y}_{t-1}}{\theta_t}$. Substituting this equality into (B.10), we have

$$\begin{aligned}\theta_t \left\|\mathbf{z}_t - \mathbf{y}_{t-1}\right\|^2 + \left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2 &= (1+\theta_t)\left\|\tfrac{\mathbf{y}'_t-(1-\theta_t)\mathbf{y}_{t-1}}{\theta_t} - \tfrac{\mathbf{x}_{t-1}+\theta_t\mathbf{y}_{t-1}}{1+\theta_t}\right\|^2 + \tfrac{\theta_t}{1+\theta_t} d_{t-1}^2 \\ &= \tfrac{1+\theta_t}{\theta_t^2}\left\|\mathbf{y}'_t - \tfrac{\theta_t\mathbf{x}_{t-1}+\mathbf{y}_{t-1}}{1+\theta_t}\right\|^2 + \tfrac{\theta_t}{1+\theta_t} d_{t-1}^2 \,. \end{aligned} \tag{B.11}$$

Inserting (B.11) into (B.9), it follows that

$$\Delta_t \leq \Delta_{t-1} - (1-\epsilon_t)\left[\theta_t\Delta_{t-1} + \tfrac{1-\theta_t^2}{\theta_t^2}\tfrac{1}{2}\left\|\mathbf{y}'_t - \tfrac{\theta_t\mathbf{x}_{t-1}+\mathbf{y}_{t-1}}{1+\theta_t}\right\|^2 + \tfrac{\theta_t(1-\theta_t)}{1+\theta_t}\tfrac{1}{2}d_{t-1}^2\right]. \tag{B.12}$$

Let us denote the quantity within the brackets above by P (for "progress" toward convergence). Also, let us define $\beta_t = \theta_t(1+\theta_t)^{-1}$, which implies $\theta_t = \beta_t(1-\beta_t)^{-1}$. We see that

$$\begin{aligned} P &= \theta_t\Delta_{t-1} + \tfrac{1-\theta_t^2}{\theta_t^2}\tfrac{1}{2}\left\|\mathbf{y}'_t - \tfrac{\theta_t\mathbf{x}_{t-1}+\mathbf{y}_{t-1}}{1+\theta_t}\right\|^2 + \tfrac{\theta_t(1-\theta_t)}{1+\theta_t}\tfrac{1}{2}d_{t-1}^2 \\ &= \tfrac{\beta_t}{1-\beta_t}\Delta_{t-1} + \tfrac{1-2\beta_t}{{\beta_t}^2}\tfrac{1}{2}\left\|\mathbf{y}'_t - \beta_t\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\right\|^2 + \tfrac{\beta_t(1-2\beta_t)}{1-\beta_t}\tfrac{1}{2}d_{t-1}^2 \\ &= \tfrac{1-2\beta_t}{1-\beta_t}\left[\tfrac{\beta_t}{1-2\beta_t}\Delta_{t-1} + \tfrac{1-\beta_t}{{\beta_t}^2}\tfrac{1}{2}\left\|\mathbf{y}'_t - \beta_t\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\right\|^2 + \beta_t\tfrac{1}{2}d_{t-1}^2\right]. \end{aligned} \tag{B.13}$$

Since $\theta_t < 1$, by definition of θ_t and $\mathbf{y}'_t$, we must have $\mathbf{y}'_t \in \mathrm{bd}(\mathcal{S}_\xi^{\mathrm{cap}})$. Since $\mathcal{S}_\xi^{\mathrm{cap}}$ is an open set and $\mathbf{y}'_t \in \mathrm{bd}(\mathcal{S}_\xi^{\mathrm{cap}})$, we have $\mathbf{y}'_t \notin \mathcal{S}_\xi^{\mathrm{cap}}$. Furthermore, since $\mathcal{S}_\xi^{\mathrm{cap}} \supseteq \mathcal{S}_\xi \supseteq \mathcal{B}_\xi(\beta_t)$, it follows that $\mathbf{y}'_t \notin \mathcal{B}_\xi(\beta_t)$. By definition of $\mathcal{B}_\xi(\beta_t)$, we have

$$\left\|\mathbf{y}'_t - \beta_t\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\right\| \geq \tau_\xi(\beta_t)\,.$$

Plugging in the definition of $\tau_\xi(\beta_t)$, it follows that

$$\begin{aligned}\tfrac{1-\beta_t}{{\beta_t}^2}\tfrac{1}{2}\left\|\mathbf{y}'_t - \beta_t\mathbf{x}_{t-1} - (1-\beta_t)\mathbf{y}_{t-1}\right\|^2 &\geq (1-\beta_t)\Delta_{t-1}\left[1 + \tfrac{\beta_t}{1-\beta_t}\left(1 - \tfrac{d_{t-1}^2}{2\Delta_{t-1}}\right) - \tfrac{1-\xi_t}{1-2\beta_t}\right]_+ \\ &= \left[\Delta_{t-1} - \beta_t\tfrac{1}{2}d_{t-1}^2 - \tfrac{1-\beta_t}{1-2\beta_t}(1-\xi_t)\Delta_{t-1}\right]_+ \\ &= \left[\tfrac{1-\beta_t}{1-2\beta_t}\xi_t\Delta_{t-1} - \tfrac{\beta_t}{1-2\beta_t}\Delta_{t-1} - \beta_t\tfrac{1}{2}d_{t-1}^2\right]_+ \end{aligned}$$

Plugging this result into (B.13), we have

$$\begin{aligned} P &\geq \tfrac{1-2\beta_t}{1-\beta_t}\left[\tfrac{\beta_t}{1-2\beta_t}\Delta_{t-1} + \left[\tfrac{1-\beta_t}{1-2\beta_t}\xi_t\Delta_{t-1} - \tfrac{\beta_t}{1-2\beta_t}\Delta_{t-1} - \tfrac{\beta_t}{2}d_{t-1}^2\right]_+ + \tfrac{\beta_t}{2}d_{t-1}^2\right] \\ &\geq \tfrac{1-2\beta_t}{1-\beta_t}\left[\tfrac{1-\beta_t}{1-2\beta_t}\xi_t\Delta_{t-1}\right] \\ &= \xi_t\Delta_{t-1}\,. \end{aligned} \tag{B.14}$$

By combining (B.14) with (B.12), we obtain the desired bound.

Case 3: $\theta_t = 0$ Using the definition of $\mathbf{y}'_t$, we have

$$\begin{aligned} \Delta_t &\leq f_t(\mathbf{y}'_t) - f_t^{\mathrm{LB}}(\mathbf{x}_t) \\ &= f_t(\mathbf{y}_{t-1}) - f_t^{\mathrm{LB}}(\mathbf{x}_t) \\ &\leq f(\mathbf{y}_{t-1}) - f_t^{\mathrm{LB}}(\mathbf{x}_t) \\ &= \Delta_{t-1} - \left[f_t^{\mathrm{LB}}(\mathbf{x}_t) - f_{t-1}^{\mathrm{LB}}(\mathbf{x}_{t-1})\right] \\ &\leq \Delta_{t-1} - (1-\epsilon_t)\tfrac{1}{2}\left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2\,. \end{aligned} \tag{B.15}$$

Above, the last step follows from termination conditions for subproblem t.

Since $\theta_t = 0$, it follows from the definition of θ_t that $\beta_t\mathbf{z}_t + (1-\beta_t)\mathbf{y}_{t-1} \notin \mathrm{cl}(\mathcal{S}_\xi^{\mathrm{cap}})$ for all $\beta \in (0, 1/2)$. Since $\mathcal{S}_\xi^{\mathrm{cap}} \supseteq \mathcal{S}_\xi \supseteq \mathcal{B}_\xi(\beta)$ for all $\beta \in (0, 1/2)$, then $\beta\mathbf{z}_t + (1-\beta)\mathbf{y}_{t-1} \notin \mathrm{cl}(B_\xi(\beta))$ for all $\beta \in (0, 1/2)$. By definition of $\mathcal{B}_\xi(\beta)$, this means that

$$\begin{aligned} &\|\beta\mathbf{z}_t + (1-\beta)\mathbf{y}_{t-1} - \beta\mathbf{x}_{t-1} - (1-\beta)\mathbf{y}_{t-1}\| > \tau_\xi(\beta) \\ \Rightarrow \quad &\|\mathbf{z}_t - \mathbf{x}_{t-1}\| > \frac{\tau_\xi(\beta)}{\beta}\,. \end{aligned}$$

This implies that

$$\|\mathbf{z}_t - \mathbf{x}_{t-1}\| \geq \lim_{\beta\to 0^+}\frac{\tau_\xi(\beta)}{\beta} = \tfrac{d}{d\beta}\tau_\xi(\beta)\Big|_{\beta=0} = \sqrt{2\Delta_{t-1}\xi_t}\,. \tag{B.16}$$

By combining (B.16) with (B.15), we obtain the result.

□

B.7 Proof of Theorem 3.9

Theorem 3.9 (BlitzScreen safe screening test)**.** *Let f_0 be any 1-strongly convex function that satisfies $f_0(\mathbf{x}) \leq f(\mathbf{x})$ for all $\mathbf{x}$, and let $\mathbf{x}_0 = \operatorname{argmin} f_0(\mathbf{x})$. Given any $\mathbf{y}_0 \neq \mathbf{x}^\star$, define the suboptimality gap $\Delta_0 = f(\mathbf{y}_0) - f_0(\mathbf{x}_0)$ as well as the "safe region"*

$$\mathcal{S}_1 = \left\{ \mathbf{x} \;\middle|\; \|\mathbf{x} - \tfrac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0)\| < \sqrt{\Delta_0 - \tfrac{1}{4}\|\mathbf{x}_0 - \mathbf{y}_0\|^2} \right\}.$$

For any $i \in [m]$, define k such that the subdomain $\mathcal{X}_i^{(k)}$ contains $\frac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0)$. Then if $\mathcal{S}_1 \subseteq \mathcal{X}_i^{(k)}$, we can safely replace ϕ_i with $\phi_i^{(k)}$ in f. That is, for all $i \in [m]$, if we let

$$\phi_{i,\mathrm{S}} = \begin{cases} \phi_i^{(k)} & \text{if } \mathcal{S}_1 \subseteq \mathcal{X}_i^{(k)}, \\ \phi_i & \text{otherwise,} \end{cases}$$

then the "screened objective" $f_\mathrm{S}(\mathbf{x}) := \psi(\mathbf{x}) + \sum_{i=1}^m \phi_{i,\mathrm{S}}(\mathbf{x})$ has the same minimizer as f.

Proof. We need to show that $\mathbf{x}^\star = \operatorname{argmin}_\mathbf{x} f(\mathbf{x}) = \operatorname{argmin}_\mathbf{x} f_\mathrm{S}(\mathbf{x}) = \mathbf{x}_\mathrm{S}$. First note that since f_0 is a 1-strongly convex lower bound on f, and $\mathbf{x}_0$ minimizes f_0, it follows that

$$f(\mathbf{x}^\star) \geq f_0(\mathbf{x}_0) + \tfrac{1}{2}\|\mathbf{x}^\star - \mathbf{x}_0\|^2 . \tag{B.17}$$

Since f is 1-strongly convex, and $\mathbf{x}^\star$ minimizes f, we have

$$f(\mathbf{y}_0) \geq f(\mathbf{x}^\star) + \tfrac{1}{2}\|\mathbf{y}_0 - \mathbf{x}^\star\|^2 . \tag{B.18}$$

Combining (B.18) with (B.17), we have

$$\begin{aligned} f(\mathbf{x}^\star) + f(\mathbf{y}_0) &\geq f_0(\mathbf{x}_0) + \tfrac{1}{2}\|\mathbf{x}^\star - \mathbf{x}_0\|^2 + f(\mathbf{x}^\star) + \tfrac{1}{2}\|\mathbf{y}_0 - \mathbf{x}^\star\|^2 \\ \Rightarrow\quad \Delta_0 &\geq \left\|\mathbf{x}^\star - \tfrac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0)\right\|^2 + \tfrac{1}{4}\|\mathbf{x}_0 - \mathbf{y}_0\|^2 \\ \Rightarrow\quad \mathbf{x}^\star &\in \mathrm{cl}(\mathcal{S}_1) . \end{aligned} \tag{B.19}$$

By construction, $f_\mathrm{S}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{S}_1$. Since $\mathcal{S}_1$ is an open set, if $\mathbf{x}^\star \in \mathcal{S}_1$, then

$$\partial f_\mathrm{S}(\mathbf{x}^\star) = \partial f(\mathbf{x}^\star) \;\Rightarrow\; \mathbf{0} \in \partial f_\mathrm{S}(\mathbf{x}^\star) \;\Rightarrow\; \mathbf{x}^\star = \mathbf{x}_\mathrm{S} .$$

For the remainder of the proof, we consider the case that $\mathbf{x}^\star \in \mathrm{bd}(\mathcal{S}_1)$. In this case, (B.18) holds with equality (since (B.19) holds with equality), meaning

$$f(\mathbf{y}_0) = f(\mathbf{x}^\star) + \tfrac{1}{2}\left\|\mathbf{y}_0 - \mathbf{x}^\star\right\|^2 . \tag{B.20}$$

Define $\mathbf{z} = \frac{1}{2}(\mathbf{y}_0 + \mathbf{x}^\star)$. Note $\mathbf{z} \in \mathcal{S}_1$, since $\mathbf{x}^\star \in \mathrm{bd}(\mathcal{S}_1)$, $\mathbf{y}_0 \in \mathrm{cl}(\mathcal{S}_1)$, $\mathbf{x}^\star \neq \mathbf{y}_0$, and $\mathcal{S}_1$ is an open ball. Also, since $\mathbf{z}$ lies on the segment $[\mathbf{x}^\star, \mathbf{y}_0]$, and f is 1-strongly convex, (B.20) implies that

$$f(\mathbf{z}) = f(\mathbf{x}^\star) + \tfrac{1}{2}\left\|\mathbf{z} - \mathbf{x}^\star\right\|^2 .$$

This implies that $\mathbf{z} - \mathbf{x}^\star \in \partial f(\mathbf{z})$, since $f(\mathbf{x}^\star) + \frac{1}{2}\left\|\mathbf{x} - \mathbf{x}^\star\right\|^2 \leq f(\mathbf{x})$ for all $\mathbf{x}$. Because $\mathbf{z} \in \mathcal{S}_1$, it follows that $\mathbf{z} - \mathbf{x}^\star \in \partial f_{\mathrm{S}}(\mathbf{z})$. Since f_{S} is 1-strongly convex, then for all $\mathbf{x}$, we have

$$\begin{aligned} f_{\mathrm{S}}(\mathbf{x}) &\geq f(\mathbf{z}) + \langle \mathbf{z} - \mathbf{x}^\star, \mathbf{x} - \mathbf{z} \rangle + \tfrac{1}{2}\left\|\mathbf{x} - \mathbf{z}\right\|^2 \\ &= f(\mathbf{x}^\star) + \tfrac{1}{2}\left\|\mathbf{z} - \mathbf{x}^\star\right\|^2 + \langle \mathbf{z} - \mathbf{x}^\star, \mathbf{x} - \mathbf{z} \rangle + \tfrac{1}{2}\left\|\mathbf{x} - \mathbf{z}\right\|^2 \\ &\geq f(\mathbf{x}^\star) . \end{aligned} \tag{B.21}$$

At the same time, since $f_{\mathrm{S}}(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{S}_1$, and f_{S} is lower semicontinuous, we must have $f_{\mathrm{S}}(\mathbf{x}^\star) = f(\mathbf{x}^\star)$. Combined with (B.21), it follows that $\mathbf{x}^\star$ minimizes f_{S}.

□

B.8 Proof of Theorem 3.11

Theorem 3.11 (Relation between equivalence regions in BlitzScreen and BlitzWS)**.** *Given points $\mathbf{x}_0$ and $\mathbf{y}_0$, function f_0, and suboptimality gap Δ_0 that satisfy the requirements for Theorem 3.9, define the ball $\mathcal{S}_1$ as in Theorem 3.9. In addition, consider the equivalence region $\mathcal{S}_\xi$ from §3.3 with parameter choices $\xi_t = 1$, $\mathbf{x}_{t-1} = \mathbf{x}_0$, $\mathbf{y}_{t-1} = \mathbf{y}_0$, and $\Delta_{t-1} = \Delta_0$. Then*

$$\mathcal{S}_1 = \mathcal{S}_\xi .$$

Proof. Consider any $\mathbf{x}_{\mathcal{S}_1} \in \mathcal{S}_1$. For some $\delta > 0$, we have

$$\left\|\mathbf{x}_{\mathcal{S}_1} - \tfrac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0)\right\| = \sqrt{\Delta_0 - \tfrac{1}{4}\left\|\mathbf{x}_0 - \mathbf{y}_0\right\|^2} - \delta$$

From the definition of $\mathcal{S}_\xi$, we know $\mathcal{S}_\xi \supseteq \mathcal{B}_\xi(\beta)$ for all $\beta \in (0, 1/2)$. Recall that $\mathcal{B}_\xi(\beta)$ is a ball with center $\beta\mathbf{x}_0 + (1-\beta)\mathbf{y}_0$ and radius $\tau_\xi(\beta)$. We have

$$\begin{aligned}\lim_{\beta \to 1/2^-} \left[\tau_\xi(\beta) - \left\|\mathbf{x}_{\mathcal{S}_1} - \beta\mathbf{x}_0 - (1-\beta)\mathbf{y}_0\right\|\right] &= \sqrt{\Delta_0 - \tfrac{1}{4}\left\|\mathbf{x}_0 - \mathbf{y}_0\right\|^2} - \left\|\mathbf{x}_{\mathcal{S}_1} - \tfrac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0)\right\| \\ &= \delta > 0\,.\end{aligned}$$

Thus, for some $\beta \in (0, 1/2)$, we have $\mathbf{x}_{\mathcal{S}_1} \in \mathcal{B}_\xi(\beta)$, implying $\mathbf{x}_{\mathcal{S}_1} \in \mathcal{S}_\xi$. We have shown $\mathcal{S}_1 \subseteq \mathcal{S}_\xi$.

To show that $\mathcal{S}_\xi \subseteq \mathcal{S}_1$, consider any $\mathbf{x}_{\mathcal{S}_\xi} \in \mathcal{S}_\xi$. Since $\xi_t = 1$, for all $\beta \in (0, 1/2)$ we have

$$\tau_\xi(\beta) = \beta\sqrt{2\Delta_0\left[1 + \tfrac{\beta}{1-\beta}\left(1 - \tfrac{\|\mathbf{x}_0 - \mathbf{y}_0\|^2}{2\Delta_0}\right)\right]} = 2\beta\sqrt{\Delta_0 - \tfrac{1}{4}\left\|\mathbf{x}_0 - \mathbf{y}_0\right\|^2}\,.$$

Above we used the fact $\frac{\|\mathbf{x}_0 - \mathbf{y}_0\|^2}{2\Delta_0} \leq 1$, which follows from $f(\mathbf{y}_0) \geq f(\mathbf{x}_0) + \frac{1}{2}\left\|\mathbf{y}_0 - \mathbf{x}_0\right\|^2$.

From the definition of $\mathcal{S}_\xi$, there exists a $\beta \in (0, 1/2)$ such that $\mathbf{x}_{\mathcal{S}_\xi} \in \mathcal{B}_\xi(\beta)$. From this, we see

$$\begin{aligned}&\left\|\mathbf{x}_{\mathcal{S}_\xi} - \beta\mathbf{x}_0 - (1-\beta)\mathbf{y}_0\right\| < \tau_\xi(\beta) \\ \Rightarrow &\left\|\mathbf{x}_{\mathcal{S}_\xi} - \beta\mathbf{x}_0 - (1-\beta)\mathbf{y}_0\right\| < 2\beta\sqrt{\Delta_0 - \tfrac{1}{4}\left\|\mathbf{x}_0 - \mathbf{y}_0\right\|^2} \\ \Rightarrow &\left\|\mathbf{x}_{\mathcal{S}_\xi} - \tfrac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0) + (\tfrac{1}{2} - \beta)(\mathbf{x}_0 - \mathbf{y}_0)\right\| < 2\beta\sqrt{\Delta_0 - \tfrac{1}{4}\left\|\mathbf{x}_0 - \mathbf{y}_0\right\|^2} \\ \Rightarrow &\left\|\mathbf{x}_{\mathcal{S}_\xi} - \tfrac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0)\right\| < (1-2\beta)\tfrac{1}{2}\left\|\mathbf{x}_0 - \mathbf{y}_0\right\| + 2\beta\sqrt{\Delta_0 - \tfrac{1}{4}\left\|\mathbf{x}_0 - \mathbf{y}_0\right\|^2} \\ \Rightarrow &\left\|\mathbf{x}_{\mathcal{S}_\xi} - \tfrac{1}{2}(\mathbf{x}_0 + \mathbf{y}_0)\right\| < \sqrt{\Delta_0 - \tfrac{1}{4}\left\|\mathbf{x}_0 - \mathbf{y}_0\right\|^2} \\ \Rightarrow &\ \mathbf{x}_{\mathcal{S}_\xi} \in \mathcal{S}_1\,.\end{aligned}$$

Note we again used the fact $\frac{\|\mathbf{x}_0 - \mathbf{y}_0\|^2}{2\Delta_0} \leq 1$, which implies $\frac{1}{2}\left\|\mathbf{x}_0 - \mathbf{y}_0\right\| \leq \sqrt{\Delta_0 - \frac{1}{4}\left\|\mathbf{x}_0 - \mathbf{y}_0\right\|^2}$.

□

B.9 Miscellaneous proofs

B.9.1 Proof that teardrop equivalence region is a subset of capsule equivalence region

Theorem B.1. *Define $\mathcal{S}_\xi$ and $\mathcal{S}_\xi^{\text{cap}}$ as in §3.3.4 and §3.3.5. Then $\mathcal{S}_\xi \subseteq \mathcal{S}_\xi^{\text{cap}}$.*

Proof. Recall $\mathcal{S}_\xi^{\text{cap}}$ is the set of points within a distance r^{cap} from the segment $[\mathbf{c}_1^{\text{cap}}, \mathbf{c}_2^{\text{cap}}]$, where

$$\mathbf{c}_1^{\text{cap}} = \beta_1 \mathbf{x}_{t-1} + (1-\beta_1)\mathbf{y}_{t-1}\,, \qquad \mathbf{c}_2^{\text{cap}} = \beta_2 \mathbf{x}_{t-1} + (1-\beta_2)\mathbf{y}_{t-1}\,,$$

$$\beta_1 = \tfrac{d_{\min}^{\text{cap}} + r^{\text{cap}}}{\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\|}\,, \qquad \beta_2 = \tfrac{d_{\max}^{\text{cap}} - r^{\text{cap}}}{\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\|}\,.$$

Consider any $\mathbf{x}' \in \mathcal{S}_\xi$. By definition of $\mathcal{S}_\xi$, there exists a scalar $\beta' \in (0, 1/2)$ such that

$$\|\mathbf{x}' - (\beta' \mathbf{x}_{t-1} + (1-\beta')\mathbf{y}_{t-1})\| < \tau_\xi(\beta')\,.$$

In the case that $\beta_1 \le \beta' \le \beta_2$, then $\beta' \mathbf{x}_{t-1} + (1-\beta')\mathbf{y}_{t-1}$ falls on the segment $[\mathbf{c}_1^{\text{cap}}, \mathbf{c}_2^{\text{cap}}]$. This implies that $\mathbf{x}' \in \mathcal{S}_\xi^{\text{Cap}}(\mathcal{S}_\xi)$, since

$$\|\mathbf{x}' - (\beta' \mathbf{x}_{t-1} + (1-\beta')\mathbf{y}_{t-1})\| < \tau_\xi(\beta') \le r^{\text{cap}}\,.$$

In the case that $\beta' \le \beta_1$, we have

$$\begin{aligned}
\|\mathbf{x}' - \mathbf{c}_1^{\text{cap}}\| &\le \|\mathbf{x}' - (\beta' \mathbf{x}_{t-1} + (1-\beta')\mathbf{y}_{t-1})\| + \|\beta' \mathbf{x}_{t-1} + (1-\beta')\mathbf{y}_{t-1} - \mathbf{c}_1^{\text{cap}}\| \\
&< \tau_\xi(\beta') + (\beta_1 - \beta')\,\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\| \\
&= [\tau_\xi(\beta') - \beta'\,\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\|] + \beta_1\,\|\mathbf{x}_{t-1} - \mathbf{y}_{t-1}\| \\
&\le -d_{\min}^{\text{cap}} + [d_{\min}^{\text{cap}} + r^{\text{cap}}] \\
&= r^{\text{cap}}\,.
\end{aligned}$$

Thus, $\mathbf{x}' \in \mathcal{S}_\xi^{\text{cap}}$ if $\beta' \le \beta_1$. A similar argument implies $\mathbf{x}' \in \mathcal{S}_\xi^{\text{cap}}$ when $\beta' \ge \beta_2$.

Thus, for all β', we have $\mathbf{x}' \in \mathcal{S}_\xi$, which implies $\mathcal{S}_\xi \subseteq \mathcal{S}_\xi^{\text{cap}}$. □

B.9.2 Proof that dual progress termination condition can be satisfied

Theorem B.2. *For the BlitzWS algorithm with approximate subproblem solutions described in §3.5.7, if subproblem t is solved exactly, then it is always the case that*

$$f_t^{\mathrm{LB}}(\mathbf{x}_t) - f_{t-1}^{\mathrm{LB}}(\mathbf{x}_{t-1}) \geq (1-\epsilon_t)\tfrac{1}{2}\left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2 .$$

Proof. If subproblem t is solved exactly, then $f_t(\mathbf{z}_t) = f_t^{\mathrm{LB}}(\mathbf{x}_t)$, since $\mathbf{x}_t = \mathbf{z}_t$. Due to condition (C3) in §3.5.7, we have $f_t(\mathbf{x}) \geq f_{t-1}^{\mathrm{LB}}(\mathbf{x})$ for all $\mathbf{x}$. Thus,

$$\begin{aligned}
& f_t(\mathbf{x}) \geq f_{t-1}^{\mathrm{LB}}(\mathbf{x}_{t-1}) + \tfrac{1}{2}\left\|\mathbf{x} - \mathbf{x}_{t-1}\right\|^2 \\
\Rightarrow\quad & f_t(\mathbf{z}_t) \geq f_{t-1}^{\mathrm{LB}}(\mathbf{x}_{t-1}) + \tfrac{1}{2}\left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2 \\
\Rightarrow\quad & f_t^{\mathrm{LB}}(\mathbf{x}_t) - f_{t-1}^{\mathrm{LB}}(\mathbf{x}_{t-1}) \geq \tfrac{1}{2}\left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2 \\
\Rightarrow\quad & f_t^{\mathrm{LB}}(\mathbf{x}_t) - f_{t-1}^{\mathrm{LB}}(\mathbf{x}_{t-1}) \geq (1-\epsilon_t)\tfrac{1}{2}\left\|\mathbf{z}_t - \mathbf{x}_{t-1}\right\|^2 .
\end{aligned}$$

□

B.9.3 Proof that f_0 lower bounds f_{L1D} in §3.6.2

Theorem B.3. *For any $\boldsymbol{\omega}_0 \in \mathbb{R}^m$, define f_0, f_{L1D}, and $\mathbf{x}_0$ as in §3.6.2. Then $f_0(\mathbf{x}) \leq f_{\mathrm{L1D}}(\mathbf{x})$ for all $\mathbf{x}$.*

Proof. Let $[\mathbf{x}_0]_j$ denote the $j$the entry of $\mathbf{x}_0$. For all x_j, the Fenchel-Young inequality implies

$$L_j^*(x_j) - x_j \left\langle \mathbf{a}_j, \boldsymbol{\omega}_0 \right\rangle \geq -L_j(\left\langle \mathbf{a}_j, \boldsymbol{\omega}_0 \right\rangle) .$$

When $x_j = L_j'(\left\langle \mathbf{a}_j, \boldsymbol{\omega}_0 \right\rangle)$, this inequality holds with equality, implying $L_j^*(x_j) - x_j \left\langle \mathbf{a}_j, \boldsymbol{\omega}_0 \right\rangle$ is minimized when $x_j = [\mathbf{x}_0]_j$. By assuming that L_j is 1-smooth, L_j^* is 1-strongly convex. Thus,

$$\begin{aligned}
\textstyle\sum_{j=1}^n \left[L_j^*(x_j) - x_j \left\langle \mathbf{a}_j, \boldsymbol{\omega}_0 \right\rangle\right] &\geq \tfrac{1}{2}\left\|\mathbf{x} - \mathbf{x}_0\right\|^2 + \textstyle\sum_{j=1}^n \left[L_j^*([\mathbf{x}_0]_j) - [\mathbf{x}_0]_j \left\langle \mathbf{a}_j, \boldsymbol{\omega}_0 \right\rangle\right] \\
&= \tfrac{1}{2}\left\|\mathbf{x} - \mathbf{x}_0\right\|^2 - \textstyle\sum_{j=1}^n L_j(\left\langle \mathbf{a}_j, \boldsymbol{\omega}_0 \right\rangle) .
\end{aligned}$$

Applying this result, we have

$$\begin{aligned} f_0(\mathbf{x}) \leq f_{\mathrm{L1D}}(\mathbf{x}) \;&\Leftrightarrow\; \tfrac{1}{2}\left\|\mathbf{x}-\mathbf{x}_0\right\|^2 - g_{\mathrm{L1}}(\boldsymbol{\omega}_0) \leq \textstyle\sum_{j=1}^n L_j^*(x_j) + \sum_{i=1}^m \phi_i(\mathbf{x}) \\ &\Leftrightarrow\; -\textstyle\sum_{j=1}^n x_j \left\langle \mathbf{a}_j, \boldsymbol{\omega}_0 \right\rangle - \lambda \left\|\boldsymbol{\omega}_0\right\|_1 \leq \textstyle\sum_{i=1}^m \phi_i(\mathbf{x}) \\ &\Leftrightarrow\; -\left\langle \mathbf{A}\boldsymbol{\omega}_0, \mathbf{x} \right\rangle - \lambda \left\|\boldsymbol{\omega}_0\right\|_1 \leq \textstyle\sum_{i=1}^m \phi_i(\mathbf{x})\,. \end{aligned} \tag{B.22}$$

Thus, it remains to prove (B.22). For each i, note $\phi_i(\mathbf{x}) = +\infty$ if $|\langle \mathbf{A}_i, \mathbf{x}\rangle| > \lambda$. Thus, we must only consider the case $|\langle \mathbf{A}_i, \mathbf{x}\rangle| \leq \lambda$, which implies $\phi_i(\mathbf{x}) = 0$. Assuming $|\langle \mathbf{A}_i, \mathbf{x}\rangle| \leq \lambda$, we have

$$\begin{aligned} -[\boldsymbol{\omega}_0]_i \left\langle \mathbf{A}_i, \mathbf{x} \right\rangle - \lambda \left|[\boldsymbol{\omega}_0]_i\right| &\leq \left|[\boldsymbol{\omega}_0]_i\right| \left|\left\langle \mathbf{A}_i, \mathbf{x} \right\rangle\right| - \lambda \left|[\boldsymbol{\omega}_0]_i\right| \\ &= \left|[\boldsymbol{\omega}_0]_i\right| \left(\left|\left\langle \mathbf{A}_i, \mathbf{x} \right\rangle\right| - \lambda\right) \\ &\leq 0 \\ &= \phi_i(\mathbf{x})\,. \end{aligned}$$

Summing over $i \in [m]$ proves (B.22). □

Appendix C

PROOFS FOR CHAPTER 4

C.1 Proof of Theorem 4.1

Theorem 4.1 (Safeness of StingyCD)**.** *In Algorithm 4.2, every skipped update would, if computed, result in* $\delta^{(t)} = 0$*. That is, if* $q^{(t-1)} \leq \tau_i$ *and* $\omega_i^{(t-1)} = 0$*, then*

$$\max\left\{-\omega_i^{(t-1)}, \frac{\left\langle \mathbf{A}_i, \mathbf{b} - \mathbf{A}\boldsymbol{\omega}^{(t-1)}\right\rangle - \lambda}{\left\|\mathbf{A}_i\right\|^2}\right\} = 0\,.$$

Proof. Since $\omega_i^{(t-1)} = 0$ when an update is skipped, we need to prove that if $q^{(t-1)} \leq \tau_i$, then

$$\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)}\right\rangle - \lambda \leq 0,$$

where we have used the definition $\mathbf{r}^{(t-1)} = \mathbf{b} - \mathbf{A}\boldsymbol{\omega}^{(t-1)}$.

We show by induction that $q^{(t)} = \left\|\mathbf{rr} - \mathbf{r}^{(t)}\right\|^2$. The base case is that $q^{(t-1)} = 0$ whenever StingyCD performs the update $\mathbf{rr} \leftarrow \mathbf{r}^{(t-1)}$. The inductive step is that

$$\begin{aligned}
q^{(t)} &= q^{(t-1)} - 2\delta^{(t)}\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} - \mathbf{rr}\right\rangle + (\delta^{(t)})^2\left\|\mathbf{A}_i\right\|^2 \\
&= \left\|\mathbf{r}^{(t-1)} - \mathbf{rr}\right\|^2 - 2\delta^{(t)}\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} - \mathbf{rr}\right\rangle + (\delta^{(t)})^2\left\|\mathbf{A}_i\right\|^2 \\
&= \left\|\mathbf{r}^{(t-1)} - \delta^{(t)}\mathbf{A}_i - \mathbf{rr}\right\|^2 \\
&= \left\|\mathbf{r}^{(t)} - \mathbf{rr}\right\|^2 .
\end{aligned}$$

Recall the definition $\tau_i = \text{sgn}(g_i)\frac{g_i^2}{\|\mathbf{A}_i\|^2}$, where $g_i = -\langle \mathbf{A}_i, \mathbf{rr}\rangle + \lambda$. It follows that

$$\begin{aligned}
q^{(t-1)} \leq \tau_i \quad &\Rightarrow \quad \left\|\mathbf{r}^{(t-1)} - \mathbf{rr}\right\|^2 \leq \text{sgn}(g_i)\frac{g_i^2}{\|\mathbf{A}_i\|^2} \\
&\Rightarrow \quad \left\|\mathbf{r}^{(t-1)} - \mathbf{rr}\right\| \leq \frac{g_i}{\|\mathbf{A}_i\|} \\
&\Rightarrow \quad \|\mathbf{A}_i\| \left\|\mathbf{r}^{(t-1)} - \mathbf{rr}\right\| \leq -\langle \mathbf{A}_i, \mathbf{rr}\rangle + \lambda \\
&\Rightarrow \quad \langle \mathbf{A}_i, \mathbf{rr}\rangle + \|\mathbf{A}_i\| \left\|\mathbf{r}^{(t-1)} - \mathbf{rr}\right\| - \lambda \leq 0 \\
&\Rightarrow \quad \left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)}\right\rangle - \lambda \leq 0\,.
\end{aligned}$$

□

C.2 Proof of Theorem 4.2

Theorem 4.2 (Per iteration time complexity of StingyCD—proven in Appendix C.2)**.** *Algorithm 4.2 can be implemented so that iteration t requires*

- *Less time than an identical iteration of Algorithm 4.1 if* $q^{(t-1)} \leq \tau_i$ *and* $\omega^{(t-1)} = 0$ *(the update is skipped) and* $\mathbf{rr}$ *is not updated. Specifically, StingyCD requires constant time, while CD requires* $\Theta(\text{NNZ}(\mathbf{A}_i))$ *time.*
- *Approximately the same amount of time as a CD iteration if the update is not skipped and* $\mathbf{rr}$ *is not updated. (Both algorithms require the same number of* $\Theta(\text{NNZ}(\mathbf{A}_i))$ *operations.)*
- *More time than a CD iteration if* $\mathbf{rr}$ *is updated. (StingyCD requires* $\Theta(\text{NNZ}(\mathbf{A}))$ *time.)*

Proof. Note that at each iteration, CD computes a dot product of length $\text{NNZ}\mathbf{A}_i$ to compute $\delta^{(t)}$. If $\delta^{(t)} \neq 0$, an additional $\mathcal{O}(\text{NNZ}\mathbf{A}_i)$ operation is required to update $\mathbf{r}^{(t)}$.

Case 1: the update is skipped and rr is not updated In this case, the only computation StingyCD performs is (i.) deciding not to update the reference vector, (ii.) choosing a coordinate to update, and (iii.) checking whether $q^{(t-1)} \leq \tau_i$ and $\omega_i^{(t-1)} = 0$. We can trivially define (i.) and (ii.) so that these steps require constant time. Checking the conditions for (iii.) also requires constant time.

Case 2: the update is not skipped and rr is not updated In this case, the only additional operation that we have not already considered is the update to $q^{(t)}$. This update can be performed in constant time by caching previous computations of $\langle \mathbf{A}_i, \mathbf{rr} \rangle$, $\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle$, and $\|\mathbf{A}_i\|^2$. The value of $\langle \mathbf{A}_i, \mathbf{rr} \rangle$ was computed when computing the threshold τ_i, and $\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle$ and $\|\mathbf{A}_i\|^2$ are necessary to compute $\delta^{(t)}$.

Case 3: rr is updated In this case, computing τ_i for all i requires computing $\langle \mathbf{A}_i, \mathbf{rr} \rangle$ for all columns in $\mathbf{A}$. This is a matrix-vector multiply that requires $\Theta(\text{NNZ}(\mathbf{A}_i))$ operations. □

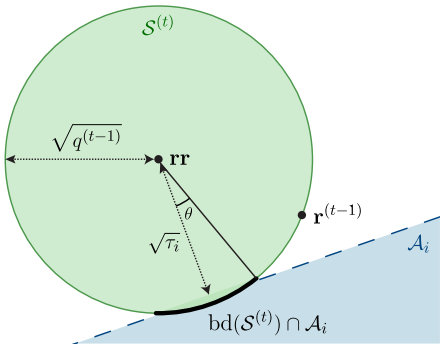
C.3 Proof of Theorem 4.4

Theorem 4.4 (Equation for $P(\mathcal{U}^{(t)})$)**.** *Assume* $\omega_i^{(t-1)} = 0$ *and* $\tau_i \in [-q^{(t-1)}, q^{(t-1)})$*. Then Assumption 4.3 implies*

$$P(\mathcal{U}^{(t)}) = \begin{cases} \frac{1}{2} I_{(1-\tau_i/q^{(t-1)})}(\frac{n-1}{2}, \frac{1}{2}) & \text{if } \tau_i \geq 0, \\ 1 - \frac{1}{2} I_{(1+\tau_i/q^{(t-1)})}(\frac{n-1}{2}, \frac{1}{2}) & \text{otherwise,} \end{cases}$$

where $I_x(a, b)$ *is the regularized incomplete beta function.*

Proof. Recall the illustration form Figure 4.2:



Because we assume $\mathbf{r}^{(t-1)}$ is uniformly distributed on the boundary of $\mathcal{S}^{(t)}$, the probability that $\mathbf{r}^{(t-1)} \in \mathcal{A}_i$ is given by dividing the area of $\mathcal{A}_i \cap \text{bd}(\mathcal{S}^{(t)})$ by the area of $\text{bd}(\mathcal{S}^{(t)})$. The region $\mathcal{A}_i \cap \text{bd}(\mathcal{S}^{(t)})$ is a hyperspherical cap. In the case that $\mathbf{rr} \notin \mathcal{A}_i$, we know from (Li,

2011) that the area of $\mathcal{A}_i \cap \mathrm{bd}(\mathcal{S}^{(t)})$ is given by

$$\tfrac{1}{2}\mathrm{area}(\mathcal{S}^{(t)}) I_{\sin(\theta)^2}(\tfrac{n-1}{2}, \tfrac{1}{2}) \,,$$

where $\mathrm{area}(\mathcal{S}^{(t)})$ is the surface area of $\mathcal{S}^{(t)}$ and θ is the angle indicated in the diagram.

When $\tau_i \geq 0$, note that by definition of τ_i, we have $\mathbf{rr} \notin \mathcal{A}_i$. It follows then that when $\tau_i \geq 0$, we have

$$\begin{aligned} P(U_t) &= \frac{\frac{1}{2}\mathrm{area}(\mathcal{S}^{(t)}) I_{\sin(\theta)^2}(\frac{n-1}{2}, \frac{1}{2})}{\mathrm{area}(\mathcal{S}^{(t)})} \\ &= \tfrac{1}{2} I_{(1-\cos(\theta)^2)}(\tfrac{n-1}{2}, \tfrac{1}{2}) \\ &= \tfrac{1}{2} I_{(1-\tau_i/q^{(t-1)})}(\tfrac{n-1}{2}, \tfrac{1}{2}) \,. \end{aligned}$$

In the case that $\tau_i < 0$, we have $\mathbf{rr} \in \mathcal{A}_i$, and we can use symmetry to see that

$$P(U_t) = 1 - \tfrac{1}{2} I_{(1+\tau_i/q^{(t-1)})}(\tfrac{n-1}{2}, \tfrac{1}{2}) \,.$$

□

C.4 Proof of Theorem 4.5

Theorem 4.5 (Convergence of StingyCD+ to a solution)**.** *Assume $\xi^{(t)} \leq \mathrm{NNZ}(\boldsymbol{\omega}^{(t-1)})$ for all $t > 0$ in StingyCD+. For each coordinate $i \in \{1, \ldots, m\}$, assume the number of consecutive iterations during which* `get_next_coordinate`() *does not return i is bounded. Then*

$$\lim_{t \to \infty} f(\boldsymbol{\omega}^{(t)}) = f(\boldsymbol{\omega}^\star) \,.$$

Before proving the theorem, we introduce and prove a few lemmas.

Lemma C.1. *Given the assumptions of Theorem 4.5, let M be a number larger than the maximum number of consecutive iterations* `get_next_coordinate`() *does not return coordinate i for all $i \in [m]$. Consider any iteration $t > 0$ of StingyCD+ and assume an $i \in [m]$ such that $\omega_i^{(t-1)} \neq 0$. Then there exists an iteration $t' \geq t$ during which StingyCD+ computes an update to coordinate i. Furthermore, we have $t' \leq t + mM$.*

Proof. Define $\mathcal{C}^{(t-1)}$ as the set of coordinates that correspond to nonzero entries in $\boldsymbol{\omega}^{(t-1)}$:

$$\mathcal{C}^{(t-1)} = \{i : \omega_i^{(t-1)} \neq 0\}.$$

Let i_{delayed} denote the unique coordinate in $\mathcal{C}^{(t-1)}$ such that the delay $D_i^{(t)}$ is largest:

$$i_{\text{delayed}} = \operatorname*{argmax}_{i \in \mathcal{C}^{(t-1)}} D_i^{(t)}. \tag{C.1}$$

This coordinate is unique because t_i^{last} differs for all $i \in \mathcal{C}^{(t-1)}$—StingyCD+ updates at most one coordinate during each iteration.

We must have $D_{i_{\text{delayed}}}^{(t)} \geq \text{NNZ}(\boldsymbol{\omega}^{(t-1)})$, since the $\text{NNZ}(\boldsymbol{\omega}^{(t-1)}) - 1$ coordinates in $\mathcal{C}^{(t-1)}$ not equal to i_{delayed} were updated before the most recent update to i_{delayed} (otherwise (C.1) would not hold).

Now let $k \geq 0$ be the smallest such k for which `get_next_coordinate()` returns i_{delayed} during iteration $t + k$. Note that $k < M$ by the theorem's assumption. We must have $D_{i_{\text{delayed}}}^{(t+k)} \geq \text{NNZ}(\boldsymbol{\omega}^{(t+k-1)})$, since (i) until an update for coordinate i is computed, $D_i^{(t)}$ is nondecreasing with t for all i, (ii) we have $D_{i_{\text{delayed}}}^{(t)} \geq \text{NNZ}(\boldsymbol{\omega}^{(t-1)})$, and (iii) whenever $\text{NNZ}(\boldsymbol{\omega}^{(t')}) = \text{NNZ}(\boldsymbol{\omega}^{(t'-1)}) + 1$ for $t' \in \{t, t+1, \ldots, t+k-1\}$, we must also have $D_{i_{\text{delayed}}}^{(t'+1)} = D_{i_{\text{delayed}}}^{(t')} + 1$.

In addition, since $i_{\text{delayed}} \in \mathcal{C}^{(t-1)}$ and i_{delayed} has not been updated since before iteration t, we must have $\omega_{i_{\text{delayed}}}^{(t+k-1)} \neq 0$. Thus, by definition of $P(U^{(t+k)})$, we must have $P(U^{(t+k)}) = 1$. Applying the assumption that $\xi^{(t+k)} \leq \text{NNZ}(\boldsymbol{\omega}^{(t+k-1)})$, it follows that

$$P(U^{(t+k)}) D_{i_{\text{delayed}}}^{(t+k)} = D_{i_{\text{delayed}}}^{(t+k)} \geq \text{NNZ}(\boldsymbol{\omega}^{(t+k)-1}) \geq \xi^{(t+k)}. \tag{C.2}$$

Thus, the condition for skipping update $t + k$ in StingyCD+ is *not* satisfied. That is, during iteration $t + k$, StingyCD+ computes an update to coordinate i_{delayed}. It follows that $D_{i_{\text{delayed}}}^{(t+k+1)} = 1$. That is, i_{delayed} now corresponds to the weight with *smallest* delay.

Now consider any i such that $\omega_i^{(t-1)} \neq 0$. This coordinate was last updated during iteration t_i^{last}. It follows that if coordinate i is not updated by iteration $t_i^{\text{last}} + (m-1)M$,

then i corresponds to the weight with largest delay among nonzero weights. This is because we have shown that the nonzero weight with maximum delay is updated within M iterations, after which it is the weight with smallest delay. Thus, before coordinate i is updated again, at most $(m-1)$ other coordinates correspond to the nonzero weight with largest delay, each of which requires at most M iterations to update. It follows that after an additional M iterations—that is, by iteration $t_i^{\text{last}} + mM$—coordinate i must be updated. □

Lemma C.2. *Given the assumptions of Theorem 4.5, then for some set $\mathcal{F}$, StingyCD+ converges to a solution of the problem*

$$\begin{aligned} \underset{\boldsymbol{\omega}\in\mathbb{R}^m}{\text{minimize}} \quad & f(\boldsymbol{\omega}) := \tfrac{1}{2}\left\|\mathbf{A}\boldsymbol{\omega}-\mathbf{b}\right\|^2 + \lambda\left\langle \mathbf{1}, \boldsymbol{\omega}\right\rangle \\ \text{s.t.} \quad & \boldsymbol{\omega} \geq 0 \\ & \omega_i = 0 \quad \textit{for all } i \in \mathcal{F} \end{aligned} \qquad \text{(P')}$$

Proof. Let us assume that $\boldsymbol{\omega}^{(t)}$ does not converge to a solution of (P') for all such $\mathcal{F}$. Then there exists a $\nu > 0$ for which the following holds: for all $t' > 0$, there exists a $t > t'$ and i satisfying $x_i^{(t-1)} \neq 0$, such that

$$\left|\delta^{(t)}\right| = \left|\max\left\{-\omega_i^{(t-1)}, \frac{\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)}\right\rangle - \lambda}{\left\|\mathbf{A}_i\right\|^2}\right\}\right| \geq \nu\,. \tag{C.3}$$

In other words, for an eventual t, coordinate $\omega_i^{(t-1)}$ is nonzero and suboptimal by at least ν.

Next, due to any update $\delta^{(t)}$ to a coordinate i during iteration t of StingyCD+, we have

$$\begin{aligned} f(\boldsymbol{\omega}^{(t)}) - f(\boldsymbol{\omega}^{(t-1)}) &= \left(\lambda - \left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)}\right\rangle\right)\delta^{(t)} + \tfrac{1}{2}\left\|\mathbf{A}_i\right\|^2(\delta^{(t)})^2 \\ &\leq -\tfrac{1}{2}\left\|\mathbf{A}_i\right\|^2(\delta^{(t)})^2\,. \end{aligned} \tag{C.4}$$

Now define $\hat{f} = \lim_{t\to\infty} f(\boldsymbol{\omega}^{(t)})$, which exists since $f(\boldsymbol{\omega}^{(t)})$ is a bounded monotone sequence. Consider an iteration t' such that $f(\boldsymbol{\omega}^{(t')}) \leq \hat{f} + \epsilon$, where we define $\epsilon > 0$ later. According to (C.3), there exists a $t > t'$ and i for which $\omega_i^{(t-1)} > 0$, where

$$\left|\max\left\{-\omega_i^{(t-1)}, \frac{\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)}\right\rangle - \lambda}{\left\|\mathbf{A}_i\right\|^2}\right\}\right| \geq \nu\,.$$

According to Lemma C.1, StingyCD+ will compute at least one update to coordinate i between iterations t and $t + mM$. During each of the iterations between iteration t and $t + mM$, suppose that coordinate i' is updated by an amount δ'. Then

$$\delta' \leq \frac{\sqrt{2\epsilon}}{\|\mathbf{A}_{i'}\|} \,. \tag{C.5}$$

Otherwise the fact that $\hat{f} = \lim_{t\to\infty} f(\boldsymbol{\omega}^{(t)})$ would be violated due to (C.4).

Now let T denote the iteration during which coordinate i is next updated. From the triangle inequality and (C.5), it follows that

$$\left\|\mathbf{r}^{(t-1)} - \mathbf{r}^{(T-1)}\right\| \leq mM\sqrt{2\epsilon} \,.$$

This implies that

$$\frac{\left\langle \mathbf{A}_i, \mathbf{r}^{(T-1)} \right\rangle}{\|\mathbf{A}_i\|^2} - \frac{\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle}{\|\mathbf{A}_i\|^2} \in \left[-\tfrac{mM\sqrt{2\epsilon}}{\|\mathbf{A}_i\|}, +\tfrac{mM\sqrt{2\epsilon}}{\|\mathbf{A}_i\|} \right] .$$

Now let δ be the update to coordinate i during iteration T. It follows that

$$\begin{aligned}
|\delta| &= \left| \max\left\{ \omega_i^{(T-1)}, \frac{\left\langle \mathbf{A}_i, \mathbf{r}^{(T-1)} \right\rangle - \lambda}{\|\mathbf{A}_i\|^2} \right\} \right| \\
&\geq \left| \max\left\{ \omega_i^{(t-1)}, \frac{\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle - \lambda}{\|\mathbf{A}_i\|^2} \right\} \right| - \frac{mM\sqrt{2\epsilon}}{\|\mathbf{A}_i\|} \\
&\geq \nu - \frac{mM\sqrt{2\epsilon}}{\|\mathbf{A}_i\|} \,.
\end{aligned}$$

Now let us define $s = \min_{i' \,:\, \|\mathbf{A}_{i'}\| > 0} \|\mathbf{A}_{i'}\|$ and $\epsilon = \frac{1}{8}(\frac{\nu s}{mM})^2$. Then it follows that

$$|\delta| > \tfrac{1}{2}\nu \,.$$

From (C.4), it follows that

$$f(\boldsymbol{\omega}^{(T)}) \leq f(\boldsymbol{\omega}^{(T-1)}) - \tfrac{1}{2}\|\mathbf{A}_i\|^2 \delta^2 \leq \hat{f} + \epsilon - \tfrac{1}{2}s^2\nu^2 < \hat{f} \,,$$

which violates the assumption that $\lim_{t\to\infty} f(\boldsymbol{\omega}^{(t)}) = \hat{f}$, proving the lemma by contradiction. □

Proof of Theorem 4.5. Suppose that StingyCD+ does not converge to a solution to (PNL). Now define $\hat{f} = \lim_{t\to\infty} f(\boldsymbol{\omega}^{(t)})$. Also define $\hat{\mathbf{r}} = \lim_{t\to\infty} \mathbf{r}^{(t)}$ and $\hat{\boldsymbol{\omega}} = \lim_{t\to\infty} \boldsymbol{\omega}^{(t)}$.

Lemma C.2 guarantees that the algorithm at least converges to a solution of (P') for some set $\mathcal{F}$. Using this assumption, if StingyCD+ does not converge to (PNL)'s solution then there exists a $\nu > 0$ and some i for which $\hat{\omega}_i = 0$, such that

$$\langle \mathbf{A}_i, \hat{\mathbf{r}} \rangle - \lambda \geq \nu \,.$$

Consider an iteration t' such that $f(\boldsymbol{\omega}^{(t'-1)}) \leq \hat{f} + \epsilon$, where we define $\epsilon > 0$ later. By Taylor expansion, we have for any $t \geq t'$,

$$\begin{aligned} f(\boldsymbol{\omega}^{(t)}) &= f(\hat{\boldsymbol{\omega}}) + \left\langle \nabla f(\hat{\boldsymbol{\omega}}), \boldsymbol{\omega}^{(t)} - \hat{\boldsymbol{\omega}} \right\rangle + \tfrac{1}{2} \left\| \mathbf{A}\boldsymbol{\omega}^{(t)} - \mathbf{A}\hat{\boldsymbol{\omega}} \right\|^2 \\ &\geq \hat{f} + \tfrac{1}{2} \left\| \hat{\mathbf{r}} - \mathbf{r}^{(t-1)} \right\|^2 . \end{aligned}$$

This implies that for any $t \geq t'$, we have

$$\left\| \hat{\mathbf{r}} - \mathbf{r}^{(t-1)} \right\| \leq \sqrt{2\epsilon} \,. \tag{C.6}$$

Define $\epsilon = \min_{i' \,:\, \|\mathbf{A}_{i'}\| \neq 0} \frac{\nu^2}{8\|\mathbf{A}_{i'}\|^2}$. It follows then that for all $t \geq t'$,

$$\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle - \lambda \geq \langle \mathbf{A}_i, \hat{\mathbf{r}} \rangle - \|\mathbf{A}_i\| \sqrt{2\epsilon} - \lambda \geq \nu - \|\mathbf{A}_i\| \sqrt{2\epsilon} \geq \tfrac{1}{2}\nu \,. \tag{C.7}$$

Also, if we assume $-\langle \mathbf{A}_i, \mathbf{r}\mathbf{r} \rangle + \lambda > 0$, we must have

$$\begin{aligned} \tau_i &= \frac{\left(-\langle \mathbf{A}_i, \mathbf{r}\mathbf{r} \rangle + \lambda\right)^2}{\|\mathbf{A}_i\|^2} \\ &\leq \frac{\left(-\left\langle \mathbf{A}_i, \mathbf{r}^{(t-1)} \right\rangle + \lambda + \|\mathbf{A}_i\| \left\| \mathbf{r}^{(t-1)} - \mathbf{r}\mathbf{r} \right\| \right)^2}{\|\mathbf{A}_i\|^2} \\ &\leq (q^{(t-1)} - \tfrac{1}{2}\nu)^2 \\ &< q^{(t-1)} \,. \end{aligned}$$

Otherwise, we must have $-\langle \mathbf{A}_i, \mathbf{r}\mathbf{r} \rangle + \lambda < 0$, which ensures $\tau_i \leq 0 \leq q^{(t-1)}$. In addition, $q^{(t-1)}$ is bounded as $t \to \infty$ due to (C.6). As a result, whenever i is returned by

`get_next_coordinate`() during an iteration $t > t'$, then $P(\mathcal{U}^{(t)})$ is bounded away from zero. As $t \to \infty$, the delay $D_i^{(t)}$ increases as, at a minimum, nonzero-valued coorinates are updated. Thus, for an eventual iteration T, we have

$$P(\mathcal{U}^{(t)})D_i^{(t)} \geq \xi^{(t)} .$$

At this point, an update to coordinate i is computed. From (C.7), it follows that

$$\delta \geq \tfrac{1}{2} \frac{\nu}{\|\mathbf{A}_i\|^2} ,$$

which ensures that

$$\begin{aligned} f(\boldsymbol{\omega}^{(T)}) &\leq f(\boldsymbol{\omega}^{(T-1)}) - \tfrac{1}{2} \|\mathbf{A}_i\|^2 \delta^2 \\ &\leq f(\hat{\boldsymbol{\omega}}) + \epsilon - \tfrac{1}{2} \frac{\nu^2}{\|\mathbf{A}_i\|^2} \\ &\leq f(\hat{\boldsymbol{\omega}}) - \tfrac{3}{8} \frac{\nu^2}{\|\mathbf{A}_i\|^2} . \end{aligned}$$

This contradicts the definition of $\hat{\boldsymbol{\omega}}$. Thus, our assumption that $\boldsymbol{\omega}^{(t)}$ does not converge to a solution of (PNL) is false. □

BIBLIOGRAPHY

C. Adam-Bourdarios, G. Cowan, C. Germain, I. Guyon, B. Kégl, and D. Rousseau. Learning to discover: The Higgs boson machine learning challenge. Technical report, 2014. URL `https://higgsml.lal.in2p3.fr/files/2014/04/documentation_v1.8.pdf`.

G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio. Variance reduction in SGD by distributed importance sampling. In *4th International Conference on Learning Representations Workshop*, 2016.

F. Bach. Exploring large feature spaces with hierarchical multiple kernel learning. In *Advances in Neural Information Processing Systems 21*, 2008.

F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2012.

A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, 2009.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

A. Bonnefoy, V. Emiya, L. Ralaivola, and R. Gribonval. A dynamic screening principle for the lasso. In *22nd European Signal Processing Conference*, 2014.

A. Bonnefoy, V. Emiya, L. Ralaivola, and R. Gribonval. Dynamic screening: Accelerating first-order algorithms for the lasso and group-lasso. *IEEE Transactions on Signal Processing*, 63(19):5121–5132, 2015.

Z. Borsos, A. Krause, and K. Y. Levy. Online variance reduction for stochastic optimization. arXiv:1802.04715, 2018.

J. M. Borwein and Q. J. Zhu. *Techniques of Variational Analysis*. Springer, 2005.

C. Boutsidis, M. Mahoney, and P. Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the Symposium on Discrete Algorithms*, 2009.

J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for L_1-regularized loss minimization. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD Interational Conference on Knowledge Discovery and Data Mining*, 2016.

T. Chen and T. He. Higgs boson discovery with boosted trees. In *NIPS Workshop on HEPML*, 2014.

T. S. Cohen and M. Welling. Group equivariant convolutional networks. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

D. Csiba, Z. Qu, and P. Richtárik. Stochastic dual coordinate ascent with adaptive probabilities. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

S. Roy D. Drusvyatskiy, M. Fazel. An optimal first order method based on optimal quadratic averaging. arXiv:1703.02518, 2016.

G. B. Dantzig. *Linear programming and extensions*. Princeton University Press, 1965.

J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25*, 2012.

A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems 27*, 2014.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

L. El Ghaoui, V. Viallon, and T. Rabbani. Safe feature elimination for the lasso and sparse supervised learning problems. *Pacific Journal of Optimization*, 8(4):667–698, 2012.

C. Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.

R.-E. Fan, P.-H. Chen, and C.-J Lin. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Fercoq and P. Richtárik. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

O. Fercoq, A. Gramfort, and J. Salmon. Mind the duality gap: Safer rules for the lasso. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

Y. Fujiwara, Y. Ida, H. Shiokawa, and S. Iwamura. Fast lasso algorithm via selective coordinate descent. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

T. Glasmachers and C. Igel. Maximum-gain working set selection for SVMs. *Journal of Machine Learning Research*, 7:1437–1466, 2006.

S. Gopal. Adaptive sampling for SGD by exploiting side information. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. arXiv:1706.02677, 2017.

A. McCallum H.-S. Chang, E. Learned-Miller. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems 30*, 2017.

R. Harikandeh, M. O. Ahmed, A. Virani, M. Schmidt, J. Konečný, and S. Sallinen. Stop wasting my gradients: Practical SVRG. In *Advances in Neural Information Processing Systems 28*, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Identity mapping in deep residual networks. In *European Conference on Computer Vision*, 2016.

D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver. Distributed prioritized experience replay. In *6th International Conference on Learning Representations*, 2018.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *32nd International Conference on Machine Learning*, 2015.

T. Joachims. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD Interational Conference on Knowledge Discovery and Data Mining*, 2006.

T. B. Johnson and C. Guestrin. Blitz: A principled meta-algorithm for scaling sparse optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

T. B. Johnson and C. Guestrin. Unified methods for exploiting piecewise linear structure in convex optimization. In *Advances in Neural Information Processing Systems 29*, 2016.

T. B. Johnson and C. Guestrin. StingyCD: Safely avoiding wasteful updates in coordinate descent. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

T. B. Johnson and C. Guestrin. A fast, principled working set algorithm for exploiting piecewise linear structure in convex problems. arXiv:1807.08046, 2018a.

Tyler B. Johnson and Carlos Guestrin. Training deep models faster with robust, approximate importance sampling. In *Advances in Neural Information Processing Systems 31*, 2018b.

A. Katharopoulos and F. Fleuret. Biased importance sampling for deep neural network training. arXiv:1706.00043, 2017.

A. Katharopoulos and F. Fleuret. Not all samples are created equal: Deep learning with importance sampling, 2018.

N. Keskar, J. Nocedal, F. Öztoprak, and A. Wächter. A second-order method for convex ℓ_1-regularized optimization with active-set prediction. *Optimization Methods and Software*, 31(3):605–621, 2016.

N. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations*, 2017.

D. Kim, S. Sra, and I. Dhillon. A scalable trust-region algorithm with application to mixed-norm regression. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

H. Kim and H. Park. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM Journal on Matrix Analysis and Applications*, 30(2):713–730, 2008.

D. P. Kingma and J. L. Ba. ADAM: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, 2012.

H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, 2007.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998.

Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. *Neural networks: Tricks of the trade*, chapter Efficient BackProp, pages 9–48. Springer, 2012.

H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 20*, 2007.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

S. Li. Concise formulas for the area and volume of a hyperspherical cap. *Asian Journal of Mathematics and Statistic*, 4(1):66–70, 2011.

J. Liu, S. Ji, and J. Ye. *SLEP: Sparse Learning with Efficient Projections*. Arizona State University, 2009. URL `http://www.public.asu.edu/~jye02/Software/SLEP`.

J. Liu, Z. Zhao, J. Wang, and J. Ye. Safe screening with variational inequalities and its application to lasso. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8), 2012.

J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious URLs: An application of large-scale online learning. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

P. Ma, B. Yu, , and M. Mahoney. A statistical perspective on algorithmic leveraging. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

M. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine learning*, 3(2), 2011.

M. Massias, A. Gramfort, and J. Salmon. From safe screening rules to working sets for faster lasso-type solvers. In *10th NIPS Workshop on Optimization for Machine Learning*, 2017.

M. Massias, A. Gramfort, and J. Salmon. Celer: A fast solver for the lasso with dual extrapolation. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013.

K. G. Murty. *Linear Complementarity, Linear and Nonlinear Programming*. Heldermann-Verlag, Berlin, 1988.

E. Ndiaye, O. Fercoq, A. Gramfort, and J. Salmon. GAP safe screening rules for sparse multi-task and multi-class models. In *Advances in Neural Information Processing Systems 28*, 2015.

E. Ndiaye, O. Fercoq, A. Gramfort, and J. Salmon. GAP safe screening rules for sparse-group lasso. In *Advances in Neural Information Processing Systems 29*, 2016.

E. Ndiaye, O. Fercoq, A. Gramfort, and J. Salmon. Gap safe screening rules for sparsity enforcing penalties. *Journal of Machine Learning Research*, 18:1–33, 2017.

D. Needell, R. Ward, and N. Srebro. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. In *Advances in Neural Information Processing Systems 27*, 2014.

Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341âĂŞ–362, 2012.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

F. Niu, B. Recht, C. Re, and S. J. Wright. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24*, 2011.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.

G. Obozinski, B. Taskar, and M. I. Jordan. Joint covariate selection and joint subspace selection for multiple classification problems. *Statistics and Computing*, 20(2):231–252, 2009.

E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *Neural Networks for Signal Processing VII: Proceedings of the 1997 IEEE Signal Processing Society Workshop*, 1997.

D. Perekrestenko, V. Cevher, and M. Jaggi. Faster coordinate descent via adaptive importance sampling. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Z. Qin, K. Scheinberg, and D. Goldfarb. Efficient block-coordinate descent algorithms for the group lasso. *Mathematical Programming Computation*, 5:143–169, 2013.

A. Raj, J. Olbrich, B. Gärtner, B. Schölkopf, and M. Jaggi. Screening rules for convex problems. arXiv:1609.07478, 2016.

P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1):433–484, 2016.

R. T. Rockafellar and R. J.-B. Wets. *Variational Analysis*. Springer-Verlag, 1997.

V. Roth and B. Fischer. The group-lasso for generalized linear models: uniqueness of solutions and efficient algorithms. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *6th International Conference on Learning Representations*, 2016.

K. Scheinberg and X. Tang. Practical inexact proximal quasi-Newton method with global complexity analysis. *Mathematical Programming*, 160:495–529, 2016.

M. Schmidt and K. Murphy. Convex structure learning in log-linear models: Beyond pairwise potentials. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010.

S. Shalev-Shwartz and A. Tewari. Stochastic methods for l1-regularized loss minimization. *Journal of Machine Learning Research*, 12(June):1865–1892, 2011.

S. Shalev-Shwartz and Y. Wexler. Minimizing the maximal loss: How and why. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proceedings of the 24th International Conference on Machine Learning*, 2007.

H.-J. M. Shi, S. Tu, Y. Xu, and W. Yin. A primer on coordinate descent algorithms. arXiv:1610.00040, 2016.

A. Shibagaki, M. Karasuyama, K. Hatano, and I. Takeuchi. Simultaneous safe screening of features and samples in doubly sparse modeling. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

A. Shrivastava, A. Gupta, and R. Girshick. Training region-based object detectors with

online hard example mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

S. Solntsev, J. Nocedal, and R. H. Byrd. An algorithm for quadratic ℓ_1-regularized optimization with a flexible active-set strategy. *Optimization Methods and Software*, 30(6): 1213–1237, 2015.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

S. U. Stich, A. Raj, and M. Jaggi. Approximate steepest coordinate descent. In *Proceedings of the 34th International Conference on Machine Learning*, 2017a.

S. U. Stich, A. Raj, and M. Jaggi. Safe adaptive importance sampling. In *Advances in Neural Information Processing Systems 30*, 2017b.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, 2014.

C. H. Teo, S.V.N. Vishwanathan, A. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11:311–365, 2010.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society, Series B*, 74(2):245–266, 2012.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

D. Vainsencher, H. Liu, and T. Zhang. Local smoothness in variance reduced optimization. In *Advances in Neural Information Processing Systems 28*, 2015.

M. J. Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1-constrained quadratic programming (lasso). *IEEE Transactions on Information Theory*, 55(5):2183–2202, 2009.

J. Wang and J. Ye. Two-layer feature reduction for sparse-group lasso via decomposition of convex sets. In *Advances in Neural Information Processing Systems 27*, 2014.

J. Wang, P. Wonka, and J. Ye. Scaling SVM and least absolute deviations via exact data reduction. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

J. Wang, P. Wonka, and J. Ye. Lasso screening rules via dual polytope projection. *Journal of Machine Learning Research*, 16:1063–1101, 2015.

S. Webb, J. Caverlee, and C. Pu. Introducing the webb spam corpus: Using email spam to identify web spam automatically. In *Proceedings of the Third Conference on Email and Anti-Spam*, 2006.

Z. Wen, W. Yin, H. Zhang, and D. Goldfarb. On convergence of an active set method for ℓ_1 minimization. *Optimization Methods and Software*, 27(6):1127–1146, 2012.

S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

Z. J. Xiang and P. J. Ramadge. Fast lasso screening tests based on correlations. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2012.

L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2543–2596, 2010.

L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

H.-F. Yu, H.-Y. Lo, H.-P. Hsieh, J.-K. Lou, T.G. McKenzie, J.-W. Chou, P.-H. Chung, C.-H. Ho, C.-F. Chang, Y.-H. Wei, J.-Y. Weng, E.-S. Yan, C.-W. Chang, T.-T. Kuo, Y.-C. Lo, P.-T. Chang, C. Po, C.-Y. Wang, Y.-H. Huang, C.-W. Hung, Y.-X. Ruan, Y.-S. Lin, S.-D. Lin, H.-T. Lin, and C.-J. Lin. Feature engineering and classifier ensemble for KDD Cup 2010. In *Proceedings of the KDD Cup 2010 Workshop*, 2010.

G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A comparison of optimization methods and software for large-scale L1-regularized linear classification. *Journal of Machine Learning Research*, 11:3183–3234, 2010.

G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved GLMNET for L1-regularized logistic regression. *Journal of Machine Learning Research*, 13:1999–2030, 2012.

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B*, 68:49–67, 2006.

G. Zanghirati and L. Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel Computing*, 29:535–551, 2003.

L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7: 1467–1492, 2006.

C. Zhang, H. Kjellström, and S. Mandt. Determinantal point processes for mini-batch diversification. Conference in Uncertainty in Artificial Intelligence, 2017.

C. Zhang, C. Öztireli, S. Mandt, and G. Salvi. Active mini-batch sampling using repulsive point processes. arXiv:1804.02772, 2018.

T. Zhang. Solving large sale linear predition problems using stohastic gradient desent algorithms. In *Proeedings of the 21st International Conferene on Mahine Learning*, 2004.

P. Zhao and T. Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

J. Zimmert, C. S. de Witt, G. Kerg, and M. Kloft. Safe screening for support vector machines. In *NIPS Workshop on Optimization for Machine Learning*, 2015.

M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.

G. Zoutendijk. *Methods of Feasible Directions: A Study in Linear and Non-linear Programming.* Elsevier, 1970.