Sampling-based Algorithms for Fast and Deployable AI

by

Cenk Baykal

B.S., University of North Carolina at Chapel Hill (2015)
S.M., Massachusetts Institute of Technology (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

Author . . . .
Department of Electrical Engineering and Computer Science
August 26, 2021

Certified by. . . . .
Daniela Rus
Andrew (1956) and Erna Viterbi Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Sampling-based Algorithms for Fast and Deployable AI

by

Cenk Baykal

Submitted to the Department of Electrical Engineering and Computer Science
on August 26, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

We present sampling-based algorithms with provable guarantees to alleviate the increasingly prohibitive costs of training and deploying modern AI systems. At the core of this thesis lies importance sampling, which we use to construct representative subsets of inputs and compress machine learning models to enable fast and deployable systems. We provide theoretical guarantees on the representativeness of the generated subsamples for a variety of objectives, ranging from eliminating data redundancy for efficient training of ML models to compressing large neural networks for real-time inference. In contrast to prior work that has predominantly focused on heuristics, the algorithms presented in this thesis can be widely applied to varying scenarios to obtain provably competitive results. We conduct empirical evaluations on real-world scenarios and data sets that demonstrate the practicality and effectiveness of the presented work.

Thesis Supervisor: Daniela Rus
Title: Andrew (1956) and Erna Viterbi Professor of Electrical Engineering and Computer Science

This doctoral thesis has been examined by a Committee of the Department of Electrical Engineering and Computer Science:

Daniela Rus . . . .

Chairperson, Thesis Supervisor
Professor of Electrical Engineering and Computer Science

Piotr Indyk . . . .

Member, Thesis Committee
Professor of Electrical Engineering and Computer Science

Aleksander Mądry . . . .

Member, Thesis Committee
Professor of Electrical Engineering and Computer Science

Acknowledgments

I am deeply grateful for all of the people who have supported me in every step of the way during my graduate studies. They have helped me push through thick-and-thin and have truly made this work possible.

First and foremost, I would like to thank my advisor, Prof. Daniela Rus. Daniela has always supported me and fostered my interests – even those that were outside of robotics. She has given me invaluable guidance in every challenge and the freedom to explore topics that interested me over the years. I always think about how I came to MIT fully intending to be a hands-on roboticist; it was only thanks to the generous encouragement and technical guidance of Daniela that I was able to discover and nurture my passion for algorithms and scalable AI. I am grateful for Daniela's vast expertise and ability to maneuver through difficult research obstacles. Her dexterity in focusing on the big picture and guidance in conducting integral research were crucially helpful for me. She has taught me a lot over the years and helped me grow both as a researcher and person. I will miss having her as an advisor and being a part of the Distributed Robotics Lab.

I am likewise grateful for my committee members, Prof. Piotr Indyk, and Prof. Aleksander Mądry, for their generous help and feedback on this thesis. Many of the ideas in this thesis built on their courses and work. This includes Sketching Algorithms for Big Data, where Piotr introduced me to sketching algorithms and helped me gain a deeper understanding of coresets, and the Science of Deep Learning, where Aleksander covered modern problems and approaches to ML, which ultimately motivated our work on network compression. I am additionally thankful to Piotr for providing helpful and comforting guidance over the years as my academic advisor. Thank you, Piotr and Aleksander!

I am indebted to my undergraduate advisors at the University of North Carolina at Chapel Hill, Prof. Ron Alterovitz, Prof. Gary Bishop, and Prof. Ming Lin, who provided nurturing opportunities and encouraged my pursuit of a PhD in the first place. I would not have even considered graduate school as an option without their support and the rewarding research experiences they enabled. Thank you for taking a chance on me when I was only a sophomore without any true understanding of what research entailed and for helping me grow as a researcher.

I feel especially lucky to have been part of the Distributed Robotics Lab during my time at MIT. I will miss the camaraderie and the lab members who helped make my time at MIT so memorable: Aaron Ray, Alexander Amini, Alyssa Pierson, Andy Spielberg, Annan Zhang, Brandon Araki, Daniel Wrafter, Hunter Hansen, Igor Gilitschenski, James Bern, Jeff Lipton, John Romanishin, Johnson Wang, Joseph DelPreto, Joshie Hughes, Lillian Chin, Lucas Liebenwein, Murad Abu-Khalaf, Noam Buckman, Paul Tylkin, Ramin Hasani, Ryan Truby, Teddy Ort, Tim Seyde, Veevee Cai, Wilko Schwarting, Xiao Li, and Yutong Ban. Thank you for your warm friendship and all that you have taught me over the years. I would also like to thank Mieke Moran for ensuring that we always had whatever resources we needed and for her warm assistance.

I will particularly miss working alongside my dear friend and long-time collaborator, Lucas Liebenwein. He has been the best PhD companion I could have asked for. Many of the ideas in this thesis would not have been possible without his vast capabilities spanning both theory and practice. Lucas was always there for me both in and out of the lab, and would generously stop whatever he would be doing to help me the best he could. I will miss his warm personality and ability to work hard and play hard at the right moments, the many interesting courses we took together, the exciting and animated conversations we had about all aspects of research and life, our brainstorming sessions over ping-pong, and many more.

I am likewise grateful for my close friends Brandon Araki, Igor Gilitschenski, and Wilko Schwarting. Brandon, you were my first friend at MIT and I feel very fortunate to have met you. You helped me get through many rough patches over the years with your consistent support, level-headed advice, and witty humor — your in-depth knowledge about ancient civilizations will be missed. Igor, you were my mentor and big brother figure for many years and I have learned so much from you. I will miss our deep conversations, your "humbling" in ping-pong, and above all, your great company. Wilko, thank you for your warm friendship and all that you have done for me over the years. I will cherish the good times we had taking courses, frequent dinners, and the candid conversations we had in and out of work; you have been deeply missed around the lab since you graduated last year.

Beyond MIT, I would like to thank Murad Tukan, Dan Feldman, and Stephanie Gil. Murad, I will miss our banter and deep conversations as we worked on proofs. Danny, thank you for introducing me to coresets. Stephanie, thank you for the fruitful collaboration and helpful advice you provided when I first stepped foot into the lab as a PhD student.

Outside of academia, I am deeply grateful for my partner Reem, for her love and encouragement over the years. She has helped me find balance in life and has taught me that there is a beauty to life outside of research as well. I feel the luckiest to have met you and cannot wait to create more memories with you.

Lastly, I am very grateful for my loving parents and brother. Thank you to my mom Bengi, my dad Cengiz, and my brother Cem for encouraging me in every endeavour and for your unending support. As someone who was born and raised in a foreign country, I realize that this may be cliche to say, but it is the truth: I would not be where I am without the sacrifices that my parents have made for me. It is solely thanks to my parents' selflessness and unconditional love that I have had the privilege and the means to pursue a PhD in the first place. Thank you for being the best parents I could have asked for and all that you have done for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern Artificial Intelligence (AI) systems, such as deep neural networks, have been unprecedentedly successful in a wide variety of high-impact applications such as Autonomous Driving [SAMR18, LSV$^+$17], Computer Vision [FJY$^+$19], health care [BM20], and Natural Language Processing [BMR$^+$20]. Some are even capable of superhuman performance in voice and image recognition [Sch15], language translation [BMR$^+$20], and games beyond Chess [SAH$^+$20] such as Poker [MSB$^+$17], Go [SHM$^+$16], and Starcarft [VBC$^+$19]. Moreover, state-of-the-art generative models can produce photorealistic images of faces, melodic songs that include singing, and creative pieces of fiction [BMR$^+$20, DJP$^+$20, KALL17]. Often times, these artificial generations are so impressively realistic that humans have a very difficult time telling them apart from their real-world or human-generated counterparts.

In parallel to their expanding capabilities, these intelligent systems have become increasingly ingrained in our everyday lives. Examples range from the AI we use in our smartphones for, e.g., voice assistants (e.g., Siri, Alexa), facial recognition, and high-quality image capture, to the AI that curates our personalized news, playlist (e.g., Spotify), and social media feed (e.g., Instagram). The fact that many modern smartphones and small embedded devices now come equipped with specialized hardware tailored for AI can be seen as a testament to the pervasiveness of these systems in devices of all shapes and sizes. Overall, it is difficult to overstate the ubiquity of AI in many of the everyday activities and processes that we partake in, whether they be at work or at home.

1.1 Motivation

Worryingly, however, AI's successes have come on the back of increasingly large and complex models trained on massive sets of labeled data. In fact, many contemporary advances in AI are achieved through sheer scale [PGL+21, BGMMS21]. For example, GPT-3, a recent language model that has demonstrated unprecedented capabilities, contains 175 *billion* parameters and was trained on hundreds of billions of words taken from sources containing up to petabytes of data such as Common Crawl, WebText2, and Wikipedia [BMR+20]. For context, training GPT-3 with a single[1] Tesla V100, one of the fastest GPUs on the market as of this writing, would take 355 years and cost $\$4,600,000$ even with the lowest-priced GPU cloud [Li20]. Even OpenAI, the well-resourced pioneer of GPT-3, concedes that the overall cost of training the model was so high that, despite a bug found in data filtering, they could not retrain the model [BMR+20].

The story of GPT-3 is sadly not an isolated one. More generally, the amount of compute used for the largest AI has doubled every 3.4 *months* and grown by more than $300,000$ times since 2012 [Ope19]. For comparison, Moore's Law has a 2 *year* doubling period [M+65]. At the same time, the success of large AI models has led many to espouse the *Bigger is better* paradigm [NKB+20] and, for the most part, write off the consequences as a price to pay for performance [PGL+21, BGMMS21]. However, we are rapidly approaching a point of indisputable [PGL+21] marginal returns of sheer size, and in turn, experiencing exponential costs in computation time, energy consumption, data acquisition, and deployment to resource-constrained

[1]In practice this is not currently possible and a distributed system with many GPUs is required; the single GPU statistic is based on a hypothetical calculation that abstracts out some of the practical requirements, e.g., memory constraints [Li20].

platforms [TGLM20, GYK+21, BGMMS21].

AI's voracious appetite for computation also has ramifications for global climate change and the environment at large. For instance, training GPT-3 produced the equivalent of 552 metric tons of CO_2 – equivalent to the amount corresponding to driving 120 passenger cars for a year [PGL+21]. Even training BERT, an older model that is 500x smaller than GPT-3 (345 million parameters), requires as much energy as a trans-atlantic flight [BGMMS21]. Worse yet, these figures do not account for the compute and effort required for other components of training, such as (often brute-force) hyperparameter tuning and data cleansing.

The issues surrounding AI like GPT-3 with billions of parameters do not end at the training step. The sheer size of the models also leads to computationally-expensive inference and infeasibility in deployment to resource-constrained platforms, such as mobile phones, embedded devices, or small-scale robotic platforms [BLG+19a]. [Li20] reports that just loading GPT-3 into memory would require 350 GB of VRAM, equivalent to at least 11 32-GB Tesla V100 GPUs. Beyond memory concerns, the inference-time costs of models this size further hamstring their deployment to a variety of high-impact applications. For example, end-to-end autonomous driving applications [APB+18] require an efficient and compact network in order to make real-time (i.e., fast) decisions. Given the current trend, deploying models of the size of GPT-3 to low-resource platforms seems out of reach, and even if they could be deployed, inference would most likely be exceedingly slow for the application at hand.

As these models have grown exponentially in size over the last decade, so have the size of the labeled data sets used for training. A pertinent saying goes *"If there is one thing statisticians and deep learning practitioners agree on, it is that more data is always better"* [NKB+20]. While labeled data may be relatively straightforward to come by or annotate when e.g., junk email or images of cats and dogs are in question, acquiring large sets of labeled data may be extremely costly or infeasible altogether in other domains like health care. For instance, applying deep networks to the task of cancer detection may require medical images that can only be labeled with the expertise of health care professionals, and a single accurate annotation may come at the cost of a biopsy on a patient [SMR+19]. These concerns motivate the question *Can we can train powerful AI using significantly smaller sets of labeled data?*

In sum, progress in developing more efficient **exact** algorithms is far outpaced by the ever-growing size of state-of-the-art machine learning models and labeled data sets. Given these diminishing returns — as evidenced by other areas like computational linear algebra, where the growing size of matrices has even exceeded the capabilities of algorithms studied for centuries [Mus18] — it is highly unlikely that significantly faster *exact* algorithms for training and inference exist. Thus, in this thesis we focus on efficient **approximation** algorithms based on importance sampling of the input. To this end, we develop novel methods to generate compact and representative subsets on which existing algorithms can be efficiently applied to obtain provably competitive solutions.

1.2 Challenges & Vision

In this thesis we introduce novel sampling-based algorithms that can render existing methods significantly more efficient. The key insight is to create subsamples of the input and model parameters to enable fast and approximate computation. To clearly depict the challenges involved, we can consider as an example the initial attempt of using uniform sampling to reduce the number of training data points from potentially billions to thousands prior to training the model for computational efficiency. More specifically, given a large data set $\mathcal{P}$, we first construct a smaller (weighted) subset $\mathcal{C} \subset \mathcal{P}$ by sampling elements from $\mathcal{P}$ uniformly at random. Subsequently, we (efficiently) train the machine learning model on this much smaller data set $\mathcal{C}$, with the hope that it is sufficiently representative of the original set $\mathcal{P}$ so that we obtain a highly-accurate model. Although this may sound promising, the downside of this approach is that there is no guarantee that uniform sampling will lead to a sufficiently rich training subsample, and we may miss crucial training points by sampling in this way.

This shortcoming is especially conspicuous in real-world scenarios that tend to have inputs that are unbalanced in terms of their labels or outliers that are crucial for the task at hand. For instance, in the case of an unbalanced data set of 10 images of cats and 990 images of dogs, the probability that we do not sample a single cat image among a uniform subsample of 70 points is roughly 0.5[2]. Therefore, if we sample uniformly at random, approximately half the time the subsample will not be sufficiently rich to contain even a single image of a cat, which would lead to training an uninformed model unaware of the existence of cats. Ideally, we would sample non-uniformly so we obtain at least one image of a cat in the subsample with exceedingly high probability, and reweigh the selected points accordingly to reflect their representation. It turns out that this toy example can be generalized, and it has been shown that uniform sampling can lead to arbitrarily bad performance both in practice and in theory [MSSW18, MS18, BFL16, BLK17, BTFR20].

Besides uniform sampling, similar heuristic approaches have also been investigated to reduce other costs as well. This includes existing *network pruning* algorithms tailored to remove redundant parameters of neural networks to speed up inference, alleviate energy consumption, and enable their deployment to resource-constrained platforms such as mobile phones [LKD+16, HMD15]. On the whole, however, a common shortcoming of prior attempts is their lack of theoretical guarantees on the generated compact model, which poses serious challenges for their widespread applicability to real-world machine learning tasks – especially to those that are safety-critical, e.g., autonomous driving.

Vision Motivated by the issues of contemporary AI and the research gap described above, the vision of this thesis is to enable efficient and deployable AI systems through *importance sampling* with theoretical guarantees. Rather than attempting to develop a faster or more efficient AI algorithm for each application, we envision using importance sampling to construct sufficiently representative data and model summaries, among others. This has the potential to enable *efficient* applications of existing AI methods on the compact representation, with provably competitive results relative to using the

[2]Assuming sampling with replacement, as is common in literature: $(1 - {}^{10}/_{1000})^{70} \approx 0.5$.

full representation.

1.3 Thesis Overview & Contributions

In this thesis, we develop and analyze a variety of sampling-based algorithms to enable fast and deployable AI. At the core of this thesis lies importance sampling, which we use to subsample important elements of a larger collection. Although this collection may refer to the training set as in the example in the previous section, we will also consider collections that represent other objects like the parameters of a large neural network. The premise of our approach is that if we can efficiently identify and sample important elements from this collection as a pre-processing step, then running existing algorithms on this subsample will be (i) computationally efficient and (ii) provably-competitive with the results on the full collection.

More specifically, the overview and contributions of this thesis are outlined as follows.

1.3.1 Reachability Analysis

In Chapter 3, we begin with a motivating and introductory application of importance sampling for the problem of *reachability analysis*. Importance sampling is a foundational concept in this thesis and we will use it extensively to accelerate existing methods or enable new functionalities, such as real-time performance. Here, we introduce importance sampling in the context of reachability analysis because it is easy to visualize and conceptualize. At a high level, given a set of initial states that is infinite in size, we use importance sampling to construct a *finite*, representative sample of initial states – among a set of uncountably infinite ones – to efficiently obtain a provably approximate set of reachable states.

This work is motivated by the fact that highly automated, real-world systems inherently depend on effectively incorporating rigorous guarantees on the performance and safety through formal verification and validation methods. For instance, in order to ensure collision-free paths, advanced driver-assistance systems need to be capable of anticipating all potential actions of the driver without overly conservative assumptions. This requires performing on-line reachability analysis, i.e., computation of states that these vehicles can reach within a given time interval. Applications of reachability analysis include safety, correctness, and controller synthesis problems involving intricate specifications or robotic systems such as autonomous aircraft and cars, medical robots, and personal-assistance robots.

Typically the state of a system is not fully observable, e.g., a car might not have precise knowledge about its position. Thus, conducting accurate reachability analysis by definition requires reasoning about *all* possible trajectories from *every* possible state. Reasoning about all possible behaviors of a system renders reachability analysis computationally intractable in practice [AD14]. This computational challenge is further compounded by the generally large size and high complexity of the system in consideration, and the practical need to obtain verification results in a reasonably short time (i.e., seconds or minutes, not days) for the sake of, for example, real-time motion planning. These computational challenges in conjunction with the need to obtain provably valid results motivate the development of approximation schemes *with provable guarantees* for reachability analysis.

Roughly speaking, given a set of initial states $\mathcal{X}$ for e.g., an autonomous agent, the objective of reachability analysis is to compute the set of *all* possible reachable states $F(\mathcal{X};T)$ after T time steps, where F is a highly computationally expensive evaluation function. Since the set of initial states is uncountably infinite, we cannot use the reachability function F on every initial state $x \in \mathcal{X}$ to compute the entirety of the reachable region. Due to this difficulty and the practical need to perform verification in a short amount of time (e.g., real-time motion planning), previous work has proposed methods that impose restrictive assumptions on the geometry of the set of initial states $\mathcal{X}$ and the reachability function F. Moreover, prior methods have also predominantly focused on *over*-approximations of the reachable set, which may lead to false-positives and is consequently not applicable to e.g., checking the feasibility of prospective motion plans.

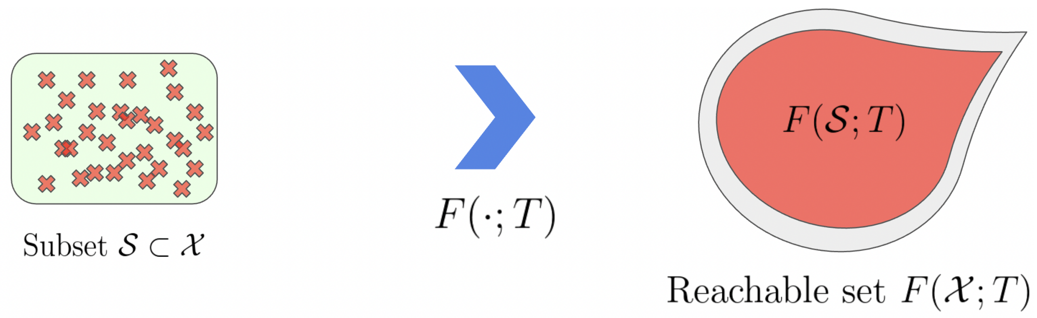

Subset $\mathcal{S} \subset \mathcal{X}$
$F(\cdot; T)$
$F(\mathcal{S}; T)$
Reachable set $F(\mathcal{X}; T)$

Figure 1-1: An example sampled set of initial states $\mathcal{S}$ (red) from $\mathcal{X}$ (green) on the left and the corresponding reachable set $F(\mathcal{S}; T)$ (red region) relative to the full reachable set, $F(\mathcal{X}; T)$ (gray) on the right.

In Chapter 3, we propose a sample-based approach to reachability analysis that imposes minimal assumptions on the initial set of states $\mathcal{X}$ and only a mild Lipschitz assumption on the reachability function F. Relative to prior work, our approach generates under-approximations of the reachable set by only evaluating the reachability of a small subset of initial states $\mathcal{S} \subset \mathcal{X}$ (see Fig. 1-1 for an example). We provide analytical sample complexity bounds that enable the practitioner to trade-off the approximation accuracy (i.e., volume of coverage of $F(\mathcal{X}; T)$) and computation time *prior to deployment and prior to even running our algorithm*. This property is crucial in safety-critical tasks such as autonomous driving, where provably accurate verification of reachability is necessary to plan collision-free paths. We also provide an anytime, asymptotically-optimal extension of our algorithm that generates increasingly better solutions as more time is allotted.

1.3.2 Streaming Coresets for Support Vector Machines

Next, in Chapter 4, we consider the problem of downsizing the set of training points required to train an accurate Support Vector Machine (SVM) model. Unlike the problem of reachability analysis from the previous subsection involving an uncountably infinite input space to sample from, here we are given a *finite* set of training data points $\mathcal{P}$; our goal is to generate a small, weighted subset $\mathcal{S}$ so that the model found by training on $\mathcal{S}$ is provably competitive with the model found by training on $\mathcal{P}$. Fig. 1-2 depicts a toy scenario where a similar separator to the near-optimal one shown could be obtained by only training on a subset of the points near the decision boundary. To achieve this, we build on prior work in *coresets* [BHPI02, HPM04, LS10, FL11, MIGR19, Fel19], which are small summaries that contain the *core* components of a larger *set*.

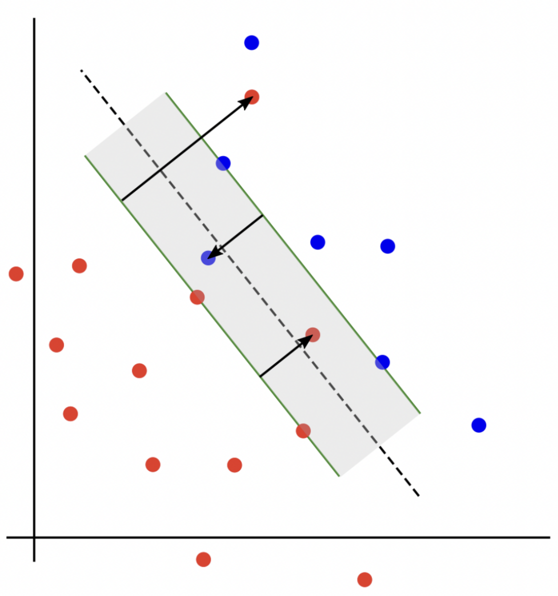


Figure 1-2: An example classification scenario with an SVM where the task is to compute the maximum-margin separator (hyperpane) that correctly separates the red input points from the blue ones. Here the points with the arrows denote those that were on the wrong side of the separation. In the context of the example scenario depicted above, Chapter 4 considers the question of whether we can compute a similar separator to the one shown by training on a smaller subset of the displayed data points, such as only those near the decision boundary.

Unlike prior work in accelerating SVM training, including related coresets work like Core Vector Machines [TKC05], our approach builds on the importance sampling framework of [BFL16, LS10] that constructs coresets by sampling data points according to their *sensitivities*. The sensitivity of a data point is deemed to be the maximum relative impact of that point on the training loss across all possible queries $w \in \mathcal{W}$. For example, in the case of SVMs, the queries $\mathcal{W}$ are the possible margins w and biases b that describe the set of all possible separating hyperplanes.

We first prove a negative result showing that no coreset of size sublinear in the number of data points n exists for pathological choices of the regularization parameter λ and data sets. Notwithstanding, we prove a sharp upper bound on the sensitivity of each point by bridging the SVM problem with k-means clustering. This leads to analytical sufficient conditions for the existence of small coresets, which turn out to be very mild in practice. By sampling points in proportion to our sensitivity upper bounds, we obtain an efficient coreset construction algorithm for SVMs with a sampling complexity that is sub-linear for a wide range of real-world hyperparameter choices and data sets. We present empirical results demonstrating our coresets' effectiveness in both offline and streaming data settings, where the training points arrive in a stream.

1.3.3 Pruning Neural Networks for Fast and Deployable AI

In the last subsection, we discussed our work on importance sampling of training points for efficient training of SVMs that appears in Chapter 4. In Chapters 5 and 6, we extend this work to sampling essential *parameters* of large neural networks to construct compact and deployable models. The main idea is once again to use importance sampling, but this time, to select and keep the important parameters of neural networks and prune the unsampled ones. We show that pruning networks through our approach generates compact models that are efficient and easy to deploy, while remaining competitively accurate as the original network. Fig. 1-3 depicts an example visualization of this process. Network pruning can also be used to reduce the burden of manually designing a small network by automatically inferring efficient architectures from larger networks. Principled pruning approaches also have potential to enable insights into the theoretical and practical properties of neural networks, including overparameterization and generalization [AGNZ18, LBL$^+$20].

As in our work on SVMs outlined in the previous subsection, we use the notion of coresets; but, here the challenge is to treat the model parameters as the input space that we construct a coreset for, rather than the input training points. Hence, our main approach is to extend importance sampling of input data based on the sensitivity framework to parameter sampling. One caveat here is that the roles of the queries (parameters of a model) and the data points swap relative to those in the sensitivity framework [BFL16, LS10] that we used for SVM coresets in chapter 4. That is, we are now interested in sampling from *weights* of a neural network $w \in \mathcal{W}$

and considering the maximum impact that the weight has on the output across all input points. It turns out that if we apply the traditional notion of sensitivity here, we end up with uniform sensitivities across all parameters, which consequently leads to uniform sampling. By digging deeper, we observe that this is due to the high capacity of neural networks and the fact that the definition of sensitivity does not consider the randomness in the queries, i.e., the input data points in the case of network pruning.

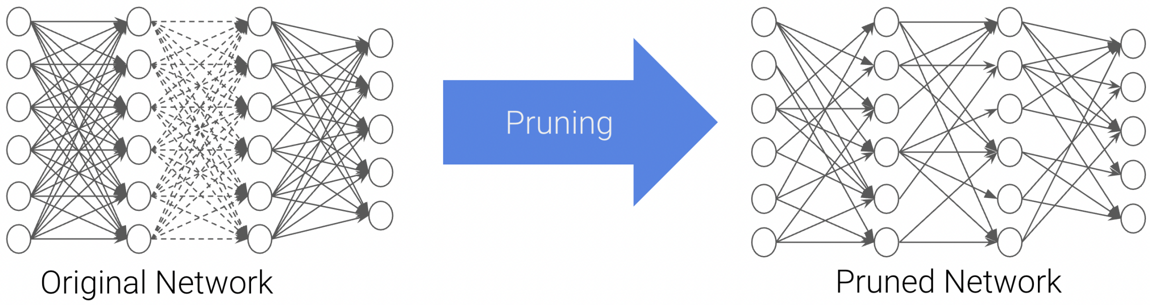


Figure 1-3: An example of network pruning. The parameters (edges) of the graph on the left is removed to generate a compact network (right). The goal of network pruning is to generate a sparse network whose predictive capability (e.g., test accuracy) is comparable with that of the original network.

To resolve this shortcoming of the existing coresets framework, we introduce the notion of *empirical sensitivity* for pruning the parameters of a machine learning model, such as those of a deep neural network. The empirical sensitivity differs from the traditional notion of sensitivity because it explicitly captures the probabilistic nature of the input training points, and leads to more informed sensitivity (i.e., importance) assignments to each of the model's weights. To the best of our knowledge, our work [BLG+19a, LBL+20, BLG+19b] is the first to extend the sensitivity coreset framework to pruning the parameters of machine learning models.

We use the empirical sensitivity framework for *provably* pruning the weights of a network model (Chapter 5) as well as neurons and filters (Chapter 6). We present theoretical guarantees on the trade-off between the size of the pruned network and its predictive accuracy relative to the original model. We also present empirical results demonstrating the effectiveness of our approach relative to existing pruning strategies. Relative to existing approaches, the theoretical guarantees of our pruning methods hold regardless of the specific state of the network — i.e., regardless of whether the given network is untrained, trained, or partially trained — and hence tends to perform consistently well across diverse pruning pipelines that incorporate varying amounts of retraining. Our analysis also provides an analytical compression bound for neural networks with respect to the desired approximation accuracy of the compressed network, which may have applications to deriving novel generalization bounds [BLG+19a, AGNZ18, AZLL19, ADH+19, NLB+19, ZVA+18].

We conclude our discussion of network pruning at the end of Chapter 5, where we discuss the practical ramifications of pruning neural networks beyond test accuracy. Namely, our recent results [LBC+21] empirically demonstrate that pruned networks are more brittle to noisy, out-of-distribution, and adversarial inputs [MMS+17, TCBM20, IST+19, CAP+19]. This calls for pruning algorithms that can consider multiple objectives rather than solely the test accuracy. We are hopeful that future work in this realm can build on the framework and techniques described in this thesis to make this possible.

1.3.4 Active Learning for Label-Efficient Deep Learning

In the last subsection, we discussed pruning large AI models using a sampling-based approach. For the last chapter (Chapter 7), we consider the use of importance sampling for label-efficient training of large-scale AI models. This setting resembles that of data reduction for efficient training of SVMs from Chapter 4, however, the main difference here is that the set of training points is assumed to be *unlabeled*. It turns out that this aspect of the problem precludes the application of the techniques from the previous chapter. Intuitively, this is because without the inputs' labels, it is not possible to accurately capture the relative importance of a point to the neural network training process.

Motivated by applications such as health care, where large sets of labeled data are difficult to obtain, we investigate whether we can train highly accurate models by only requesting the labels of a subset of points that are deemed to be highly informative.

This is called *Active Learning* and it is often conducted in an iterative fashion where the labels of small batches of inputs are sequentially requested. The objective of an active learning algorithm is to decide the best (i.e., most informative) batch of unlabeled data points to label at each time step.

Unsurprisingly, a vast amount of prior work has focused on devising proxies that can capture the informativeness of each data point without knowledge of its label. Examples include proxies based on model uncertainty [RXC+20], clustering on the network embedding [SS17a, AZK+19], and margin proximity [DP18]. Despite their effectiveness in certain scenarios, virtually all of the existing active learning methods are based on greedy selection of the points that are ranked as most informative with respect to the proxy measure. Despite the intuitiveness of this approach, it is known to be highly sensitive to outliers and to occasionally perform significantly worse than uniform sampling on even simple scenarios involving the MNIST dataset [EGSD20].

In our work, we deviate from the greedy paradigm and instead propose a low-regret active learning framework that can be applied with any user-specified notion of informativeness. We view the problem of active learning as one of prediction with (sleeping) expert advice, and develop and analyze a regret minimization algorithm, AdaProd$^+$, tailored to the active learning setting. Our analysis and evaluations also show that AdaProd$^+$ can be applied off-the-shelf with existing informativeness measures to robustify and improve upon greedy selection.

1.4 Thesis Contributions

This thesis makes the following contributions:

- A novel sampling algorithm for provable reachability analysis based on a geometric packing of the input space. We provide bounds on the relative volume of the ground-truth reachable set we can cover using the finite subsample generated by our approach. To the best of our knowledge, this work was the first to present a general approach to reachability analysis with explicit theoretical bounds on the resulting approximation as a function of the sample size. Unlike prior work, this work was the first to enable the practitioner to control the computation time vs. accuracy trade-off analytically and above all, *prior to deployment* to safety-critical tasks like autonomous driving.

- The first importance sampling approach with an explicit sampling complexity for constructing coresets for accelerated SVM training in offline, streaming, and dynamic data settings, where points are frequently inserted and deleted.

- A novel theoretical framework for parameter pruning of machine learning models, Sensitivity-informed Provable Pruning (SiPP), and a family of neural network pruning algorithms that are instantiations. Namely, we introduce a family of deterministic and randomized sampling algorithms for unstructured (weight) pruning of large-scale neural networks and provide bounds on the performance of the generated pruned models. We also bridge network pruning and generalization bounds for neural networks. To the best of our knowledge, this work was the first to provably generate pruned networks with guarantees on their accuracy for unforeseen inputs at the time of publication.

- The first algorithm for structured (i.e., neuron and filter) pruning with guarantees on the performance of the generated model as a function of the model's size. Additionally, a fully-automated sample allocation procedure based on our error bounds to prioritize sampling and retaining components of important layers.

- A novel, low-regret active learning approach for label-efficient training of modern, large-scale neural network models. Unlike prior work, our method is applicable with any notion of informativeness and is the first to provide bounds on the regret of data acquisition for efficient training of deep neural networks.

The main contribution of this thesis relative to prior work is the development of widely-applicable, practical algorithms with provable guarantees for model pruning and input (data) compression. This stands in contrast to related work, which has primarily focused on developing heuristics and presented results that were mostly empirical. For example, as detailed in Chapter 2, virtually all of prior edge pruning methods are heuristics that do not provide any bounds on the performance of the generated pruned network. Our approach in Chapter 5 on the other hand, explicitly provides guarantees on the size and capability of the pruned model. The same can be said for filter and neuron pruning, where prior approaches consider the norm of the filter or neuron weights to decide what to prune with the hope that it captures their importance, but this leads to unreliably pruned networks. Our work in Chapter 6, on the other hand, introduces an approach that provably ensures that neurons or filters that are crucial are approximately preserved during pruning. More generally, the theoretical guarantees of the algorithms presented in this thesis enable the efficient training and deployment of reliable and capable AI systems for a wide range of applications, including safety-critical tasks such as autonomous driving where assessing potential risks prior to deployment is crucial.

1.5 Thesis Outline

The outline of this thesis as follows:

Chapter II: Background and Related Work We cover background material for the topics covered in this thesis and provide an overview of prior work.

Chapter III: Reachability Analysis We enable real-time and approximately-optimal reachability analysis by providing a sampling approach to construct and evaluate a *finite* and representative subset of initial states among an infinitely large set.

Chapter IV: Streaming Coresets for Support Vector Machines We present an efficient sampling approach of large data sets to accelerate SVM training in offline streaming, and dynamic data settings, and present bounds on the performance of the model trained on the subsample.

Chapter V: Provable Weight Pruning of Neural Networks We introduce a theoretical framework and a family of importance sampling algorithms to prune individual parameters of large-scale models and provide bounds on the resulting compact model's performance.

Chapter VI: Structured Pruning and The Next Pruning Frontier We extend the importance sampling framework from the previous chapter to sample and prune neurons and filters, i.e., structured pruning, to obtain networks capable of fast and real-time inference. We discuss the next pruning frontier in context of the limitations of state-of-the-art pruning approaches and present avenues for future development.

Chapter VII: Active Learning for Label-Efficient Deep Learning We present a novel, low-regret active learning algorithm for label-efficient deep learning that alleviates the brittleness of prior greedy approaches and leads to reliable and uniformly superior performance in practice.

Chapter VIII: Conclusion We summarize the contributions and implications of the work presented in this thesis, discuss the lessons learned along the way, and outline avenues for future work.

1.6 Relevant Publications

The core chapters of this thesis are based on the following publications. Here, the superscript * denotes shared first-authorship (equal contribution).

Chapter III: Reachability Analysis

1. Lucas Liebenwein*, Cenk Baykal*, Igor Gilitschenski, Sertac Karaman, and Daniela Rus, "Sampling-Based Approximation Algorithms for Reachability Analysis with Provable Guarantees," in Robotics: Science and Systems (RSS), 2018 [LBG$^+$18].

Chapter IV: Streaming Coresets for Support Vector Machines

1. Cenk Baykal*, Murad Tukan*, Dan Feldman, and Daniela Rus, "Coresets for Support Vector Machines," as special issue invite to Theoretical Computer Science (TCS), 2021 [BTFR21].

2. Murad Tukan*, Cenk Baykal*, Dan Feldman, and Daniela Rus, "On Coresets for Support Vector Machines," in Theory and Applications of Models of Computation (TAMC), 2020 [TBFR20].

Chapter V: Provable Weight Pruning of Neural Networks

1. Cenk Baykal*, Lucas Liebenwein*, Igor Gilitschenski, Dan Feldman, Daniela Rus, "Data-Dependent Coresets for Compressing Neural Networks with Applications to Generalization Bounds," in International Conference on Machine Learning (ICLR), May 2019 [BLG+18].

2. Cenk Baykal*, Lucas Liebenwein*, Igor Gilitschenski, Dan Feldman, and Daniela Rus, "SiPPing Neural Networks: Sensitivity-informed Provable Pruning of Neural Networks," 2021, submitted to SIAM Journal on Mathematics of Data Science (SIMODS), available on arXiv [BLG+21].

Chapter VI: Structured Pruning and The Next Pruning Frontier

1. Lucas Liebenwein*, Cenk Baykal*, Harry Lang, Dan Feldman, and Daniela Rus, "Provable Filter Pruning for Efficient Neural Networks," in International Conference on Machine Learning (ICLR), 2020 [LBL+20].

2. Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus, "Lost in Pruning: The Effects of Pruning Neural Networks beyond Test Accuracy," in Conference on Machine Learning and Systems (MLSys), 2021 [LBC+21].

Chapter VII: Active Learning for Label-Efficient Deep Learning

1. Cenk Baykal, Lucas Liebenwein, Dan Feldman, Daniela Rus, "Low-Regret Active Learning," submitted to NeurIPS 2021, available on arXiv [BLFR21].

1.7 Other Relevant Results

This thesis presents a subset of the results obtained as part of the student's PhD work for sake of brevity and cohesion. Additional results along the same lines as the ones presented in this thesis include:

1. **Coresets for Sparse PageRank** (Harry Lang*, Cenk Baykal*, Najib Abu Samra, Tony Tannous, Dan Feldman, and Daniela Rus, "Deterministic Coresets for Stochastic Matrices with Applications to Scalable Sparse PageRank," in TAMC, 2019) [LBS+19].

2. **Algorithms for Resilient Multi-Agent Consensus** (Stephanie Gil, Cenk Baykal, and Daniela Rus, "Resilient Multi-Agent Consensus using Wi-Fi Signals," in IEEE Control Systems Letters, 2019) [GBR18].

Chapter 2

Background and Related Work

Our work builds on prior work in importance sampling, reachability analysis, coresets, network pruning, and active learning. In this chapter, we cover background and prior work pertinent to the material presented in this thesis. In Sec. 2.1, we cover prior work in importance sampling for approximate machine learning that is related to, but different than the flavor of sampling we consider in our work. Next, in Sec. 2.2, we overview prior approaches to approximate reachability analysis. In Sec. 2.3, we describe and outline relevant work in coresets, a foundational idea that will be commonly referenced throughout this thesis. Relatedly, we discuss acceleration methods for Support Vector Machine training in Sec. 2.4, including prior coreset-based methods. We overview the state-of-the-art in neural network pruning in Sec. 2.5. We conclude the chapter with a coverage of related approaches and their limitations for active learning in Sec. 2.6.

2.1 Importance Sampling for Approximate Queries

Here, we cover related work where the objective is to accelerate the core training or evaluation procedure on the fly by embedding importance sampling into it, rather than use sampling as a pre-processing step to generate compact representations as we do in this thesis. Stochastic Gradient Descent (SGD) [RM51] is one such example: it samples a mini batch of data points at each iteration to approximate the gradient over the entire data set *during training*. This is different – but complementary – to the flavor of sampling we cover in this thesis, which aims to generate compact representations prior to applying existing (including sampling-based) algorithms such as SGD.

Approaches in this realm include Locality Sensitive Hashing (LSH)-based [GIM+99, IN07] samplers for efficient and approximate ML applications [SS17c, SS17b]. In [SS17c], for example, the authors speed up training of neural networks by LSH-based sampling of nodes with probabilities proportional to their activations [SS17c] *during training*, and performing forward and backward propagation on only the sampled nodes. This is similar to the work on (Adaptive) Dropout [MF14, SHK+14], where the forward and backpropagation steps are done on adaptively selected subsets of nodes with high activations. The power of LSH-based sampling lies in its ability to generate (desirably) correlated samples in sublinear time [SS17c]; hence, it has been successfully applied to efficient training of large language models [SS17b], importance sampling for SGD and ADAM [CXS19], and mutual information estimation [SS20]. LSH-based sampling can be synergistically combined with the approaches outlined in this thesis by, for example, combining the LSH-based sampler for efficient training with the network pruning algorithms presented in Chapters 5 and 6.

A similar line of work involves the use of importance sampling for approximate tensor operations in neural network training and inference [ALHS18, PBB+19]. The main idea is to replace the computationally-expensive tensor operations, e.g., matrix multiplication and tensor convolutions, during the forward (and backward, if training) passes with faster operations involving tensors made up of sampled rows and columns [ALHS18]. Related methods include channel gating [HZDS+18], which attempts to learn a gate (sampling) function at run-time to identify important regions in the input features, and skip the computation on the unimportant parts; sparsified backpropagation where only a small subset of the full gradient is used to update the model parameters [SRMW17]; and Sub-LInear Deep Learning Engine (SLIDE) [CMF$^+$19], an approach that blends various randomized (sampling-based) algorithms to speed up neural network training and inference.

Unlike the work in this thesis, these approaches aim to reduce the runtime by integrating the sampling procedure directly into the original algorithm at run-time, as opposed to sparsifying the data set or the machine learning model prior to execution. Their shortcoming lies in the fact that they cannot alleviate the difficulty of deploying over-parameterized models to resource-constrained platforms or handling dynamic streams of data in a memory-efficient way. However, our work can be synergistically combined with these related sampling methods to simultaneously reap the benefits of both using a compact representation and a sampling approach at run-time for training and deploying efficient machine learning models.

2.2 Reachability Analysis

A large body of literature has been devoted to formal analysis of reachability for finite [CGL93], continuous [BF04, CK08], and hybrid systems [ACH$^+$95, MBT05, ADI06, CSM$^+$15] with applications ranging from ensuring the safety of mobile robots in human environments to flight maneuver verification [LRH$^+$17, BGM$^+$14, GHH$^+$11, PLA$^+$15, XD10, Ser95, Imm15, EWR$^+$15, GO05, MBT05]. Accurate reachability analysis necessitates the computation of the reachable set for every single state in an uncountable state space, which is computationally intractable in practice [Tab09]. Therefore, a vast collection of prior work has focused on developing approximation algorithms for the computation of approximate reachable sets.

To this end, an approach using zonotopes is presented in [Alt15] and implemented as the CORA toolbox. Taylor flow tubes were used in Flow* tool [CÁS13]. Other tools such as HyTech [HHWT97] and [CK99] consider only linear dynamics. In [FCTS15, MBT05] reachability is cast as Partial Differential Equations (PDEs) and standard tools for solving PDEs are used. However, virtually all of these tools compute over-approximations and cast the generally (highly) non-linear system dynamics as polynomials or even linear functions, which results in potentially unbounded error terms. Moreover, they are highly sensitive to the dimensionality of the input space and suffer to a great extent from the curse of dimensionality [MBT05].

To overcome the computational tractability issues, the simplified version of the problem has been addressed in the context of safety, namely *falsification* [PKV09, CK08, BF04]. In this case, an invariant set is fixed and the procedure generates some trajectory that exists in the set. This approach culminated in the development of frameworks such as counter-example guided abstraction refinement methods [CGJ+00, KDSA14] for safety verification and synthesis. Another falsification method for continuous and hybrid systems based on the Rapidly-exploring Random Tree (RRT) algorithm (and its variants) was proposed in [BF04]. Other approaches to overcome the inherit tractability issues of verification include decoupling the dynamics of the system [CT15], which, however, poses a strong assumption on the types of systems that can be considered.

Previous work has also investigated verification for autonomous cars and other agents. In [AD14], planned driving maneuvers are verified before execution via zonotope-based approximations of the reachable set. Similarly, [EFG16] considered safe envelopes for shared steering of a vehicle, however, the approach does not consider vehicle dynamics, but its performance heavily depends on the geometry of the environment. The work in [LSV+17] introduces a compositional verification framework for a large array of driving scenarios to verify planner constraints on a city-level scale. In [AGJT14], the coupled dynamics of vehicle platooning is investigated and verified offline.

In contrast to prior work, our work in Chapter 3 addresses the problem of generating accurate under-approximations of reachable sets and closes the research gap in approximate reachability analysis, which has predominantly focused on computing over-approximations. Unlike prior approaches that lack theoretical guarantees on performance or impose strong assumptions on the problem, our sampling-based algorithm is simple-to-implement, imposes minimal assumptions, and is provably-optimal up to any desired approximation accuracy. This enables the practitioner to explicitly make the trade-off between the computational complexity and approximation accuracy prior to deployment to safety-critical tasks like autonomous driving.

2.3 Coresets

In this section, we cover prior work in coresets, a fundamental idea that we will revisit frequently throughout our work. As discussed in Chapter 1, popular machine learning algorithms are computationally expensive, or worse yet, intractable to train on massive data sets where the input data set is so large that it may not be possible to process all the data at one time. A natural approach to achieve scalability when faced with Big Data is to first conduct a preprocessing step to summarize the input data points by a significantly smaller, representative set. Off-the-shelf training algorithms can then be run efficiently on this compressed set of data points. The premise of this two-step learning procedure is that the model trained on the compressed set will be provably competitive with the model trained on the original set – as long as the data summary, i.e., the *coreset*, can be generated efficiently and is sufficiently representative.

Coresets are small weighted subsets of the training points such that models trained on the coreset are approximately as good as the ones trained on the original (massive) data set. Coreset constructions were originally introduced for k-means and k-median clustering [HPM04] and subsequently generalized for applications to other problems via an importance sampling-based, *sensitivity* framework [LS10, BFL16, BLK17]. Coresets have been used successfully to accelerate various machine learning algorithms such as k-means clustering [FL11, BFL16], graphical model training [MMK18], and logistic regression [HCB16] (see the surveys of [BLK17] and [MS18] for a complete list).

The sensitivity framework provides a popular coreset construction technique, and we will frequently reference and leverage it in this thesis. The main idea is to perform importance sampling with respect to the points' sensitivities, where the sensitivity of each point is defined to be the maximum relative impact of the data point on the objective (loss) function across *all queries*. Points with high sensitivities have a large impact on the objective value and are sampled with correspondingly high probability, and vice-versa. *The main challenge in generating small coresets often lies in evaluating the importance of each point in a sufficiently accurate and computationally efficient way.*

2.4 Accelerating Support Vector Machines

Training SVMs requires $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space in the offline setting where n is the number of training points. Towards the goal of accelerating SVM training in the offline setting, [TKK07, TKC05] introduced the Core Vector Machine (CVM) and Ball Vector Machine (BVM) algorithms, which are based on reformulating the SVM problem as the Minimum Enclosing Ball (MEB) problem and Enclosing Ball (EB) problem, respectively, and by leveraging existing coreset constructions for each; see [BC03]. However, CVM's accuracy and convergence properties have been noted to be at times inferior relative to those of existing SVM implementations [LC07]. Additionally, unlike the coreset construction we introduce in Chapter 4, neither the CVM, nor the BVM algorithm extends naturally to streaming or dynamic settings where data points are continuously inserted or deleted.

Similar geometric approaches, including extensions of the MEB formulation, those based on convex hulls and extreme points, among others, were investigated by [AS10, GJ09, HPRZ07, Joa06, NKT14, RDIV09] Another class of related work includes the use of canonical optimization algorithms such as the Frank-Wolfe algorithm [Cla10], Gilbert's algorithm [Cla10, CHW12], and a primal-dual approach combined with Stochastic Gradient Descent (SGD) [HKS11]. SGD-based approaches, such as Pegasos [SSSSC11], have been a popular tool of choice in approximately-optimal SVM training. Pegasos is a stochastic sub-gradient algorithm for obtaining a $(1+\varepsilon)$-approximate solution to the SVM problem in $\widetilde{\mathcal{O}}(dn\lambda/\varepsilon)$ time for a linear kernel, where λ is the regularization parameter and d is the dimensionality of the input data points. In contrast to the approach presented in Chapter 4, these approaches and their corresponding theoretical guarantees do not readily extend to dynamic data sets and/or streaming settings.

There has been prior work in streaming algorithms for SVMs, such as those of [AS10, HPRZ07, NR14, RDIV09]. However, they generally suffer from poor practical performance in comparison to that of approximately optimal SVM algorithms in the offline (batch) setting, high difficulty of implementation and application to practical settings, or lack theoretical guarantees.

2.5 Pruning Neural Networks for Fast and Scalable AI

Prior pruning work has considered varying techniques ranging from those based on Singular Value Decomposition (SVD) to regularized (sparsity-aware) training [DSD$^+$13, DZB$^+$14, JVZ14, KPY$^+$15, TXZ$^+$15, IRS$^+$15, AS17, YLWT17]. Other general methodologies include those that exploit the structure of the weight tensors to induce sparsity [ZLW$^+$17, SSK15, CYF$^+$15, CCB$^+$16, WWW$^+$16], Bayesian approaches [ZZWT19, LUW17], and those based on dynamic reparameterization where certain weights are pruned and others grown back iteratively throughout the training process [MW19, CPI18, BKML17, GYC16]; see [GEH19, BGOFG20, YLLW18] for a more extensive overview. In contrast to these approaches, we consider pruning a given network independent of how it was trained, and we specifically take into the account the input data distribution in order to make more informed pruning decisions.

Existing network pruning algorithms are predominantly based on data oblivious [RFC20, HMD15] or data-informed [GKDP20, LGGT19, MMT$^+$19, YLC$^+$18, LAT18] heuristics that work well in practice as part of a pruning pipeline that incorporates retraining [LBC$^+$21]. However, they generally lack provable guarantees and thus provide little insight into the mechanics of the pruning algorithms and consequently into the pruned network.

Weight pruning Weight pruning [LDS90] hinges on the idea that only a few of dominant weights within a layer, or more generally, the entire network, are required to approximately preserve the output. Approaches of this flavor were investigated by [LL16, DCP17], e.g., by embedding sparsity as a constraint [IHM$^+$16, AANR17, LRLZ17]. A popular weight-based pruning method is that of [HMD15, RFC20], where weights with absolute values below a threshold are removed. A recent approach of [LAT18] prunes the parameters of the network by using a mini-batch of data points to approximate the influence of each parameter on the loss function of a randomly initialized network. Other data-informed techniques include [GKDP20, LSB$^+$20, LGGT19, MTK$^+$16, MMT$^+$19, YLC$^+$18, LBL$^+$20]. Despite their favorable empirical performance, these approaches generally lack rigorous theoretical analysis of the effect that the discarded weights can have on the model's performance. Approaches based on Alternating Direction Method of Multipliers (ADMM) have also been investigated for weight [MCL$^+$19, ZYZ$^+$18] and filter [LJL$^+$19] pruning. However, these methods tend to be more computationally-intensive as they require running an iterative algorithm (ADMM) for the pruning step itself, and do not provide an explicit theoretical guarantee for or trade-off between the performance and size of the compressed network.

Provable pruning In the recent years, [AGNZ18] introduced a compression method based on random projections and proved norm-based bounds on the compressed network – however, this bound only applies to points from the *training set only*. In contrast, our work provides approximation guarantees on the network's output that hold even for points outside the training set. The work of [AAR20] formulates the pruning problem as a convex optimization problem at each layer and provides bounds depending on the convex program. However, this approach requires solving an optimization program iteratively (via ADMM) to prune each layer and does not provide an explicit control or theoretical trade-off between the size and accuracy of the compressed network.

We close this research gap in Chapters 5 and 6 by introducing sensitivity-informed pruning, a family of network pruning algorithms that provably prunes parameters and filters of a given network in a data-informed manner. Building and improving on state-of-the-art pruning methods, the presented approaches are simultaneously provably accurate, data-informed, and applicable to various architectures including fully-connected (FNNs), convolutional (CNNs), and recurrent neural networks (RNNs).

2.6 Active Learning for Label-Efficient Deep Learning

As we saw in Chapter 1, the successes of large-scale AI have come on the back of training large models on massive labeled data sets, which may be costly or even infeasible to obtain in other applications like Healthcare [BM20]. *Active learning* focuses on alleviating the high label-cost of learning by only querying the labels of points that are deemed to be the most informative. The notion of informativeness is not concrete and may be defined in a task-specific way. Unsurprisingly, prior work in active learning has primarily focused on devising proxy metrics to appropriately quantify the informativeness of each data point in a tractable way. Examples include proxies based on model uncertainty [GIG17], clustering [SS17a, AZK+19], and margin proximity [DP18] (see [RXC+20] for a detailed survey).

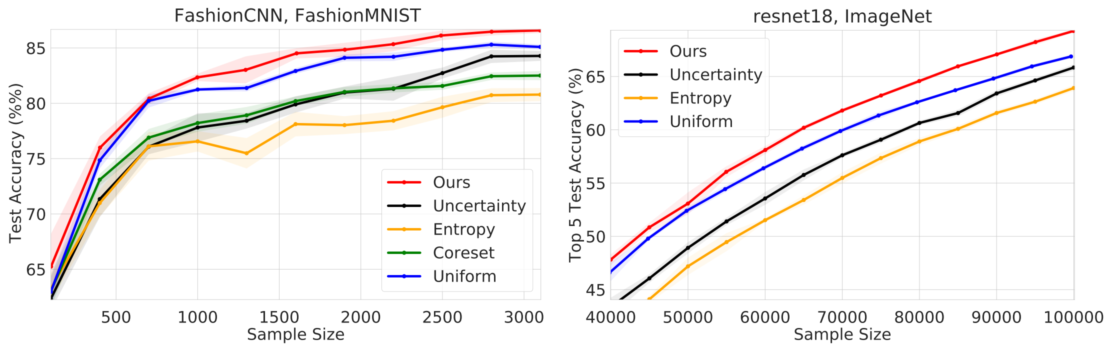


Figure 2-1: Evaluations on FashionMNIST and ImageNet with benchmark active learning algorithms. Existing approaches based on greedy selection are not robust and may perform significantly worse than uniform sampling.

An overwhelming majority of existing methods are based on greedy selection of the points that are ranked as most informative with respect to the proxy criterion. Despite the intuitiveness of this approach, it is known to be highly sensitive to outliers and to occasionally perform significantly worse than uniform sampling on certain tasks [EGSD20] – as Fig. 2-1 also depicts. In fact, this shortcoming manifests itself even on reportedly redundant data sets, such as MNIST, where existing approaches can lead to models with up to 15% (absolute terms) higher test error [Mut19] than those obtained with uniform sampling. In sum, the general lack of robustness and reliability guarantees of prior (greedy) approaches impedes their widespread applicability to high-impact deep learning tasks.

In Chapter 7, we propose a low-regret active learning framework that can be applied with any user-specified notion of informativeness. Our approach deviates from the standard greedy paradigm and instead formulates the active learning problem as that of learning with expert advice in an adversarial environment. We develop and analyze a regret minimization algorithm tailored to the active learning setting. In this regard, our work aims to advance the development of effective and robust active learning strategies that can be widely applied to modern deep learning tasks.

Chapter 3

Importance Sampling in the Context of Reachability Analysis

3.1 Overview

We begin our exposition on sampling-based algorithms with a motivating application to the problem of reachability analysis, i.e., the computation of states that can be reached by a safe trajectory from any of the specified initial states. Given that the set of states we have to consider is infinite in size, we have to settle for approximations. Here, we propose a principled approach for generating such an approximation based on *set packing* from computational geometry. By constructing an appropriate packing for the set of initial states, we enable computational efficiency and real-time performance by only requiring the evaluation of a *finite* set of states, at the cost of small, bounded approximation error.

Above all, this application highlights the effectiveness of sampling-based approaches in AI; with sampling, we solve the exact same problem as before except on a finite set of states whose reachability can be computed efficiently. Moreover, unlike prior work that is based on complex geometrical assumptions and methods, sampling serves as a highly general approach that imposes minimal assumptions and is capable of handling

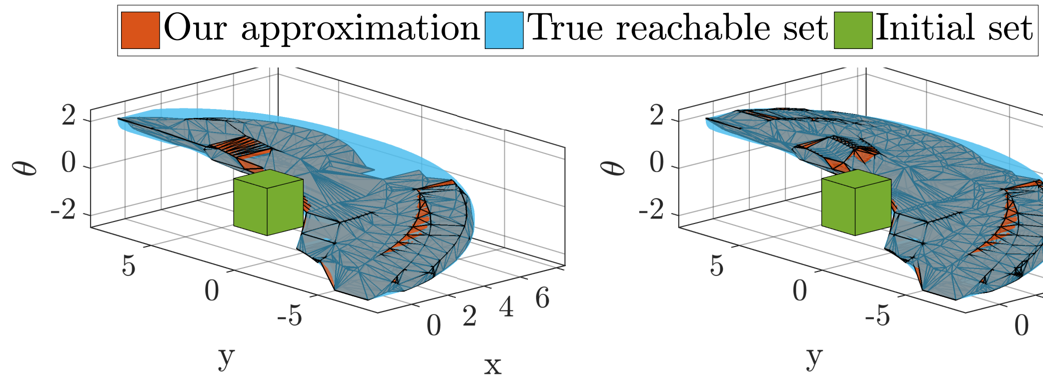


Figure 3-1: $(1-\varepsilon) = 0.2$

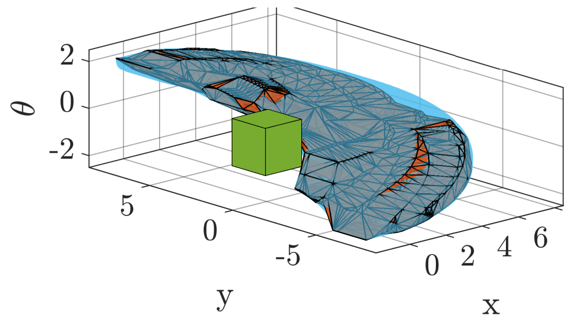


Figure 3-2: $(1-\varepsilon) = 0.4$

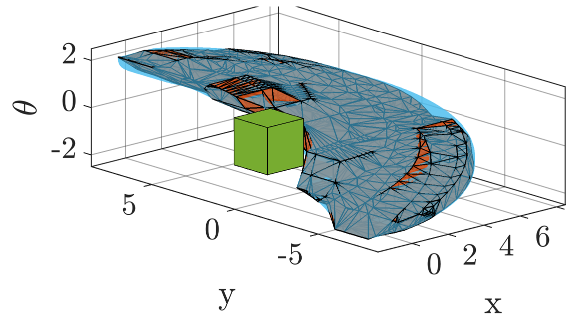


Figure 3-3: $(1-\varepsilon) = 0.6$

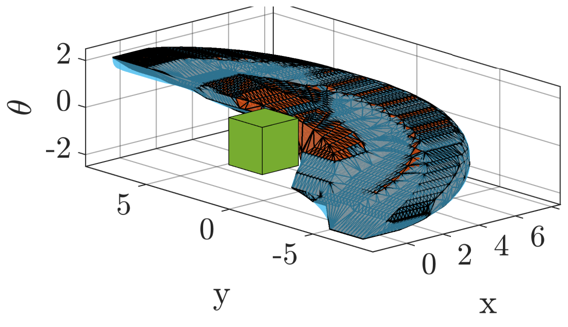


Figure 3-4: $(1-\varepsilon) = 0.8$

Figure 3-5: (a)-(d): A 0.2, 0.4, 0.6, and 0.8-approximation respectively, of the reachable set of a unicycle car. The set of initial conditions is taken to be the unit cube around the origin.

nonlinear dynamics as well as arbitrarily non-convex regions of states. This work is based on [LBG+18] and contributes the following:

1. A unified problem formulation that imposes minimal system-specific assumptions, for the provable *under-approximation* of the reachable set of a continuous set,

2. A simple-to-implement, sampling-based algorithm to sample sufficiently diverse initial states in order to generate a provably-accurate approximation of the target reachable set, up to any desired accuracy. Additionally, an anytime variant of our approximation algorithm that is asymptotically optimal,

3. An analysis of the proposed algorithms and their theoretical properties, including approximation accuracy and computational complexity, as a function of the desired approximation error ε and system-specific variables,

4. Empirical results demonstrating the broad applicability and practical effectiveness of our algorithm on a set of real-world inspired, simulated scenarios.

3.2 The Reachability Problem

Consider a robot described by the dynamic system: $\dot{x} = h(x, u)$, where $x \in \mathbb{R}^d$ is the state, $u \in \mathcal{U}$ is the control signal, the set of controls $\mathcal{U} \subset \mathbb{R}^m$ is a compact set, and h is a continuously differentiable function. Let $\mathbf{x}(x_0, t, u(\cdot))$ denote the robot's state at time t starting from the initial state x_0 and evolving under input control signal $u(t)$. Denote the set $H(x_0, T) = \{\mathbf{x}(x_0, T, u(\cdot)) \mid u(t) \in \mathcal{U}, \forall t \in [0, T]\}$.

Let $\mathcal{X} \subset \mathbb{R}^d$ denote the d-dimensional compact set of initial states and let $\mathcal{Y}$ denote the d-dimensional compact set of all reachable states. The *reachability function* $f : \mathbb{R}^d \to 2^{\mathcal{Y}}$ maps each state $x \in \mathcal{X}$ to a compact set of reachable states, $f(x) \subseteq 2^{\mathcal{Y}}$. Let $T > 0$ be the terminal time, the reachability function is $f(x) = H(x, T)$. The domain of f is defined to be the entire d-dimensional space for convenience in our analysis, however, without loss of generality we assume that $f(z) = \emptyset \, \forall z \notin \mathcal{X}$. For any subset $\mathcal{X}' \subseteq \mathcal{X}$, define the function that represents the union of all reachable sets in $\mathcal{X}'$, $F(\mathcal{X}') = \cup_{x \in \mathcal{X}'} f(x)$ for notational brevity. Note that F is monotonous, i.e., for any subset $\mathcal{X}' \subseteq \mathcal{X}$, $F(\mathcal{X}') \subseteq F(\mathcal{X})$ and that the *ground truth reachable set* is $F(\mathcal{X})$. We assume that both the state space, $\mathcal{X}$, and the ground truth reachable set, $F(\mathcal{X})$, are compact.

Our objective is to generate an approximation to $F(\mathcal{X})$ via the union of the reachable sets of a finite set $\mathcal{S} \subset \mathcal{X}$ such that $|\mathcal{S}| = n \in \mathbb{N}_+$. That is, our goal is to judiciously construct a finite set $\mathcal{S} \subset \mathcal{X}$ such that $F(\mathcal{S}) \approx F(\mathcal{X})$. We will quantify the accuracy of our approximation by comparing the volume of $F(\mathcal{S})$ to that of $F(\mathcal{X})$. More formally, let $\mu(\cdot)$ denote the Lebesgue measure, i.e., volume, of any measurable set and let $\mu(F(\mathcal{X}))$ denote the volume of the ground truth reachable set.

We formalize the reachability problem as follows.

Problem 1 (Approximate Reachability Problem). *For any given $\varepsilon \in (0,1)$, generate a finite subset $\mathcal{S} \subset \mathcal{X}$ such that*

$$(1-\varepsilon)\mu(F(\mathcal{X})) \leq \mu(F(\mathcal{S})) \leq \mu(F(\mathcal{X})). \tag{3.1}$$

3.3 Method

In this section, we present our algorithm for generating reachable sets that are provably competitive with the ground-truth reachable set to any desired accuracy. We show that our main method (Alg. 2) can easily be used as a sub-procedure to obtain an anytime, asymptotically-optimal algorithm (Alg. 3) for reachability analysis.

3.3.1 Overview

Accurate construction of the ground-truth reachable set $F(\mathcal{X})$ requires the evaluation of the reachable set $f(x)$ for all initial states $x \in \mathcal{X}$ in the worst case. However, the set of initial states $\mathcal{X}$ is uncountably infinite, which renders straightforward evaluation of $F(\mathcal{X})$ computationally intractable. To address this challenge, we take a sampling-based approach to reachability analysis.

Our method is based on the premise that evaluating the reachability of a carefully constructed finite subset $\mathcal{S} \subset \mathcal{X}$ of the initial states can serve as an accurate approximation of the ground-truth reachable set. The crux of our approach lies in generating a set $\mathcal{S}$ containing points that are sufficiently diverse, i.e., far-apart from one another, to ensure that that the union of the reachable sets $F(\mathcal{S})$ covers as much of $F(\mathcal{X})$ as possible. To this end, we use the GreedyPack [Wu16] algorithm (Alg. 1) to construct a δ-packing for $\mathcal{X}$, i.e., a subset $\mathcal{S} \subset \mathcal{X}$ such that the minimum pairwise distances between the points in $\mathcal{S}$ is greater than δ (see Sec. 3.4), for an appropriate $\delta > 0$.

Algorithm 1 GreedyPack

```
Input: $\mathcal{X} \subset \mathbb{R}^d$: $d$-dimensional set of input states,
$\delta \in \mathbb{R}_+$: packing precision
Output: $\mathcal{S}$: a $\delta$-packing for $\mathcal{X}$
1: $\mathcal{S} \leftarrow$ Random point chosen from $\mathcal{X}$;
2: while $\exists x \in \mathcal{X} : \forall y \in \mathcal{S}, \|x - y\| \geq \delta$ do
3:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{x\}$;
4: return $\mathcal{S}$;
```

3.3.2 Approximately-optimal Algorithm

Our algorithm for approximately-optimal reachability analysis is shown as ApproximateReachability (Alg. 2). We give an overview of our method, which follows directly from the constructive proofs presented in Sec. 3.4. In particular, for any desired approximation accuracy $\varepsilon \in (0, 1)$, our analysis establishes an appropriate value of δ to be used in constructing the δ-packing for $\mathcal{X}$. Lines 1-4 of Alg. 2 generate upper bounds on the system-specific constraints, which are then used, along with ε, to set the appropriate δ parameter for the packing (Line 6). The δ-packing, $\mathcal{S}$, is then constructed (Line 7) and the reachability of $\mathcal{S}$ is computed and returned (Lines 8-12).

3.3.3 Anytime, Asymptotically-optimal Algorithm

Our anytime, asymptotically-optimal algorithm is shown as AnytimeApproximateReachability (Alg. 3). The main idea behind our algorithm is that if Alg. 2 is iteratively invoked with increasingly small values of ε as input, then the generated reachable sets will converge to the ground-truth reachable set as the number of iterations i tends to infinity.

3.4 Theoretical Guarantees

We prove under mild assumptions that for any specified error $\varepsilon \in (0, 1)$, Alg. 2 generates an approximately optimal reachable set by computing the reachable sets of only finitely many initial states. As a corollary, we prove that the anytime variant of our approximation algorithm, Alg. 3, is asymptotically optimal. For brevity, some of

Algorithm 2 ApproximateReachability

```
Input: $\mathcal{X} \subset \mathbb{R}^d$: a $d$-dimensional set of input states,
$\varepsilon \in (0,1)$: desired approximation accuracy
Output: $\hat{F}_{\mathcal{S}}$: approximate reachable set such that $\mu(\hat{F}_{\mathcal{S}}) \geq (1-\varepsilon)\mu(F(\mathcal{X}))$
1: $\alpha \leftarrow$ UpperSurfAreaToVolume($\mathcal{X}$);
2: $K \leftarrow$ UpperLipschitzConstant($\mathcal{X}$);
3:                                   ▷ Approximate the universal constant from Lemma 3
4: $c \leftarrow$ UpperUniversalConstant($\mathcal{X}$);
5:                                   ▷ Set packing precision as established in Theorem 7
6: $\delta \leftarrow d\left((1-\varepsilon)^{-1/d} - 1\right)/(\alpha K c)$;
7: $\mathcal{S} \leftarrow$ GreedyPack($\mathcal{X}, \delta$);          ▷ Generate a $\delta$-packing for $\mathcal{X}$
8: $\hat{F}_{\mathcal{S}} \leftarrow \emptyset$;
9: for $x \in \mathcal{S}$ do          ▷ Evaluate the reachable set for each $x \in \mathcal{S}$
10:     $\hat{f}(x) \leftarrow$ EvaluateReachability($x$);
11:     $\hat{F}_{\mathcal{S}} \leftarrow \hat{F}_{\mathcal{S}} \cup \hat{f}(x)$;
12: return $\hat{F}_{\mathcal{S}}$;
```

Algorithm 3 AnytimeApproximateReachability

```
Input: $\mathcal{X} \subset \mathbb{R}^d$: $d$-dimensional set of input states
Output: $\hat{F}_{\mathcal{S}}$: asymptotically-optimal reachable set
1: $\varepsilon \leftarrow 1/2$;   $\hat{F}_{\mathcal{S}} \leftarrow \emptyset$;
2: while allotted time remains do
3:     $\hat{F}_{\mathcal{S}} \leftarrow$ ApproximateReachability($\mathcal{X}, \varepsilon$);
4:     $\varepsilon \leftarrow \varepsilon/2$;
5: return $\hat{F}_{\mathcal{S}}$;
```

the proofs have been omitted from this manuscript.

The intuition behind our analysis is as follows. Assuming that the reachability function is Lipschitz continuous, we expect similar states to map to similar reachable sets. Therefore, to establish a bound on the quality of our finite set generated by Alg. 1, we show that the total overlap between the reachable sets of $x \in \mathcal{S}$ and the neighboring states of x is high (Lemmas 3, 4). This implies that $f(x)$ serves as a good approximation of the reachable sets of all neighboring states. By generalizing and applying this argument to all points in $\mathcal{S}$, we establish that $F(\mathcal{S})$ serves as a good approximation for the entire reachable set, given that the points are sampled sufficiently far apart from one another (Lemmas 5, 6). We conclude by establishing sufficient conditions on the constructed set $\mathcal{S}$ to ensure a $(1-\varepsilon)$-approximation of the reachable set and analyzing the computational complexity of our algorithm (Theorems 7, 10). The proofs of the technical lemmas are deferred to Sec. 3.6 for clarity of exposition.

3.4.1 Preliminaries

For any measurable two sets A, B, let $d_{\mathrm{H}}(A, B)$ denote the Hausdorff distance [Mun14] between A and B, i.e.,

$$d_{\mathrm{H}}(A, B) = \max\left\{ \sup_{a \in A} \inf_{b \in B} \|a - b\|\,,\ \sup_{b \in B} \inf_{a \in A} \|a - b\| \right\},$$

where $\|\cdot\|$ denotes the Euclidean norm. Intuitively, the Hausdorff distance is the maximum length of the path from a point in one of the sets to the closest point belonging to the other set. An equivalent way to define the Hausdorff distance between compact sets A, B is via a δ-fattening:

$$d_{\mathrm{H}}(A, B) = \inf\{\delta \geq 0 : A \subseteq B_\delta \text{ and } B \subseteq A_\delta\},$$

where $A_\delta \subseteq \mathbb{R}^d$ denotes the δ-fattening of the set A: $A_\delta = \cup_{a \in A} \mathcal{B}_\delta(a)$, where $\mathcal{B}_\delta(a)$ denotes the closed ball of radius δ centered at a [Mun14].

Assumption 1 (Measure Properties of f)**.** *For all states $x \in \mathcal{X}$, $f(x)$ is measurable and has non-zero measure, i.e., $\forall x \in \mathcal{X}$ $\mu(f(x)) > 0$.*

Assumption 2 (Lipschitz Continuity of f)**.** *There exists a Lipschitz constant $K > 0$ such that for any two states $x, y \in \mathcal{X}$, $d_H(f(x), f(y)) \leq K \|x - y\|$.*

A reachable set $A \subseteq \mathcal{Y}$ is said to be *m-rectifiable* if there exists a Lipschitz map $g : \mathbb{R}^m \to \mathcal{Y}$ onto $\mathcal{Y}$ [Fed69, Definition 3.2.14]. Less formally, rectifiability of A implies that A enjoys many of the properties shared by smooth manifolds and that A is in a sense a piece-wise smooth set.

Assumption 3 (Properties of F)**.** *For all subsets $\mathcal{X}' \subseteq \mathcal{X} \subseteq \mathbb{R}^d$, $F(\mathcal{X}')$ is compact and for any $\delta \geq 0$, the δ-fattening of $F(\mathcal{X}')$, $F(\mathcal{X}')_\delta$, is a $(d-1)$-rectifiable set.*

Assumption 1 ensures that f maps to reachable sets that have strictly positive volume. Assumption 2 guarantees that similar initial states have similar reachable sets. Finally, Assumption 3 rules out pathological problem instances, where the reachable sets may have arbitrarily large surface areas, e.g., fractals. We note that these assumptions generally hold in practical settings. In particular, dynamical systems in real-world scenarios often exhibit these kind of properties.

We will leverage properties of coverings and packings with respect to the Euclidean norm in our analysis. For $\delta > 0$, a d-dimensional space A defining the metric space $(A, \|\cdot\|)$, a set $C = \{c_1, \ldots, c_N\} \subset B$ is said to be a δ-covering of $B \subset A$ if for all $b \in B$, there exists $c \in C$ such that $\|b - c\| \leq \delta$. The *covering number* of B, $N(B, \delta)$, is defined as the minimum cardinality of a δ-covering of B [Wu16]. Under the same setting as before, $D = \{d_1, \ldots, d_M\} \subset B$ is a δ-packing for B if $\min_{i,j \in [M]: i \neq j} \|d_i - d_j\| > \delta$. The *packing number* of B, $M(B, \delta)$ is defined as the maximum cardinality of a δ-packing of B [Wu16]. We present the following standard results in covering and packing numbers for completeness.

Theorem 1 ([Wu16, Theorem 14.2])**.** *For $\delta > 0$, $\mathcal{X} \subset \mathbb{R}^d$, and $C = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$ the following holds:*

$$\left(\frac{1}{\delta}\right)^d \frac{\mu(\mathcal{X})}{C} \leq N(\mathcal{X}, \delta) \leq M(\mathcal{X}, \delta) \leq \left(\frac{2\Delta(\mathcal{X})}{\delta} + 1\right)^d,$$

where $\Delta(\mathcal{X}) = \max_{x,y \in \mathcal{X}} \|x - y\|$ and $\Gamma(\cdot)$ is the Euler gamma function [Dav59].

Note that in order for a δ-packing for the set $\mathcal{X} \subseteq \mathbb{R}^d$ to be non-trivial, i.e., contain at least 2 points, it must be the case that $\delta \leq \Delta(\mathcal{X})$. Since, if $\delta > \Delta(\mathcal{X})$, there cannot exist more than one point in the packing by definition of $\Delta(\mathcal{X}) = \max_{x,y \in \mathcal{X}} \|x - y\|$. Therefore, we henceforth will assume that we are interested in generating non-trivial packings with parameter $\delta \in (0, \Delta(\mathcal{X})]$. Our first lemma bounds the size of the points generated by GreedyPack (Alg. 1).

Lemma 2. *Given a compact set $\mathcal{X} \subset \mathbb{R}^d$ and $\delta \in (0, \Delta(\mathcal{X})]$,* GreedyPack *(Alg. 1) generates a packing for $\mathcal{X}$, $\mathcal{S}$, such that*

$$\left(\frac{1}{\delta}\right)^d \frac{\mu(\mathcal{X})}{C} \leq |\mathcal{S}| \leq \left(\frac{3\Delta(\mathcal{X})}{\delta}\right)^d,$$

where $\Delta(\mathcal{X})$ and C are defined as in Theorem 1.

Proof. By the termination condition of GreedyPack, we have the negation of the following statement $\exists x \in \mathcal{X}\ \forall y \in \mathcal{S}\ \ \|x - y\| > \delta$, which is $\forall x \in \mathcal{X}\ \exists y \in \mathcal{S}\ \ \|x - y\| \leq \delta$, which implies that upon termination of the algorithm, $\mathcal{S}$ is an δ-covering of $\mathcal{X}$ and thus $|\mathcal{S}| \geq N(\mathcal{X}, \delta)$. Invoking Theorem 1, we have

$$|\mathcal{S}| \geq N(\mathcal{X}, \delta) \geq \left(\frac{1}{\delta}\right)^d \frac{\mu(\mathcal{X})}{C}.$$

The upper bound follows by the upper bound on $M(\mathcal{X}, \delta)$ from Theorem 1 and the inequality $\delta \leq \Delta(\mathcal{X})$. $\square$

3.4.2 Analysis of Algorithms 2 and 3

For a non-empty, compact set $A \subseteq \mathbb{R}^d$, the Minkowski Content of A, denoted by $\lambda(\partial A)$, is defined by the Minkowski-Steiner formula [Fed69]:

$$\lambda(\partial A) = \liminf_{\delta \to 0} \frac{\mu(A_\delta) - \mu(A)}{\delta}. \tag{3.2}$$

We note that for sufficiently regular sets A, $\lambda(\partial A)$ corresponds to the surface area of A [Fed69]. In the subsequent lemma, we establish a technical inequality that will later be used to establish the relationship between the volume of δ-fattenings.

Lemma 3. *Let $A \subset \mathbb{R}^d$ be a non-empty compact set with finite diameter and let $\delta \in (0, \Delta(\mathcal{X})]$. If $\mu(A) > 0$ and the δ-fattening of A, A_δ, is $(d-1)$-rectifiable for all*

$\delta \in (0, \Delta(\mathcal{X})]$, *then there exists a finite universal constant* $c \geq 1$*:*

$$c = \max\{M/(d\mu(\mathcal{B}_1(\cdot))^{1/d}), 1\} < \infty,$$

where $M > 0$ *is a finite constant independent of* δ *and* A*, such that*

$$\frac{\lambda(\partial A_\delta)}{\mu(A_\delta)^{(d-1)/d}} \leq \frac{c\,\lambda(\partial A)}{\mu(A)^{(d-1)/d}},$$

where c *is independent of* δ *and* A*, and* $\lambda(\partial A)$ *is the Minkowski Content as defined in* (3.2).

The following technical Lemma quantifies the relationship between $\mu(A)$ and $\mu(A_\delta)$ and will be used later in our main result.

Lemma 4. *Consider any finite, strictly positive* δ *and a non-empty, compact set* $A \subset \mathbb{R}^d$ *such that* $\mu(A) > 0$ *and its* δ*-fattening,* A_δ*, is a* $(d-1)$*-rectifiable set for all* $\delta \geq 0$*. Then,*

$$\mu(A_\delta) \leq \left(1 + \frac{c\,\delta\lambda(\partial A)}{\mu(A)d}\right)^d \mu(A), \tag{3.3}$$

where $c \geq 1$ *is the universal constant from Lemma 3, and* $\lambda(\partial A)$ *is the Minkowski Content as defined in* (3.2).

Proof. Define the function $g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ such that $g(x) = (\mu(A_x)/\mu(A))^{1/d}$, and let $h(\delta) : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be the function defining the Minkowski Content $h(\delta) = \frac{\mu(A_\delta)-\mu(A)}{\delta}$. Observe that since A_δ is a $(d-1)$-rectifiable set we have that the limit inferior of the expression in (3.2) is equivalent to its limit superior [Fed69, Theorem 3.2.39], and thus the traditional limit exists:

$$\begin{aligned}\lambda(\partial A) &= \liminf_{\delta \to 0} \frac{\mu(A_\delta) - \mu(A)}{\delta} = \liminf_{\delta \to 0}\, h(\delta) \\ &= \limsup_{\delta \to 0}\, h(\delta) = \lim_{\delta \to 0}\, h(\delta).\end{aligned}$$

Let $\varepsilon > 0$ and define $\lambda' = \lambda(\partial A) + \varepsilon > \lambda(\partial A)$. By definition of $\lim_{\delta\to 0} h(\delta) = \lambda(\partial A)$, there exists an open interval defined by a constant (as a function of ε), $\xi(\varepsilon) > 0$ such that for all $\delta' \in (0, \xi(\varepsilon))$, $|h(\delta') - \lambda(\partial A)| < \varepsilon$. This implies that $h(\delta') < \lambda(\partial A) + \varepsilon$ and thus by definition of $h(\delta')$, we have for all $\delta' \in (0, \xi(\varepsilon))$

$$\mu(A_{\delta'}) < \mu(A) + \delta' \left(\lambda(\partial A) + \varepsilon\right) = \mu(A) + \delta' \lambda'.$$

Thus, for all $\delta' \in (0, \xi(\varepsilon))$, we have

$$\begin{aligned} g(\delta') = \left(\frac{\mu(A_{\delta'})}{\mu(A)}\right)^{1/d} < \left(1 + \frac{\delta'\lambda'}{\mu(A)}\right)^{1/d} \leq 1 + \frac{\delta'\lambda'}{\mu(A)d} \\ \leq 1 + \frac{c\delta'\lambda'}{\mu(A)d}, \end{aligned} \tag{3.4}$$

where the second to last inequality follows by Bernoulli's inequality and the last inequality follows by the fact that $c \geq 1$. The inequality (3.4) implies that if $\delta \in (0, \xi(\varepsilon))$, then the inequality trivially holds and we are done. Therefore, we next consider the case where $\delta \in [\xi(\varepsilon), \Delta(\mathcal{X})]$.

Differentiating $g(\delta')$ with respect to δ' yields:

$$\begin{aligned} \frac{\mathrm{d}\, g(\delta')}{\mathrm{d}\delta'} &= \frac{1}{\mu(A)^{1/d}} \cdot \frac{\mathrm{d}\mu(A_{\delta'})^{1/d}}{\mathrm{d}\delta'} \\ &= \frac{\lambda(\partial A_{\delta'})}{\mu(A)^{1/d}\, \mu(A_{\delta'})^{(d-1)/d}\, d}, \end{aligned}$$

where we used the $(d-1)$-rectifiability of $A_{\delta'}$ to replace the limit with the Minkowski Content, since the limit is ensured to exist. Moreover, note that

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\delta'}\left(1 + \frac{c\delta'\lambda'}{\mu(A)d}\right) &= \frac{c\lambda'}{\mu(A)d} > \frac{c\lambda(\partial A)}{\mu(A)d} \\ &\geq \frac{1}{d\mu(A)^{1/d}} \cdot \frac{\lambda(\partial A_{\delta'})}{\mu(A_{\delta'})^{(d-1)/d}} = \frac{\mathrm{d}\, g(\delta')}{\mathrm{d}\delta'}, \end{aligned}$$

where the inequality follows from Lemma 3.

Thus, we have that for all $\delta' \in (0, \Delta(\mathcal{X})]$, $g(\delta')$ grows not faster than does the expression on the right-hand side of the inequality in (3.4). This observation combined with the fact that the inequality holds for all values of $\delta' \in (0, \xi(\varepsilon))$ implies that for all $\delta' \in (0, \Delta(\mathcal{X})]$, inequality (3.4) holds and thus we have also have for the originally specified δ that $g(\delta) \leq 1 + \frac{c\delta\lambda'}{\mu(A)d}$. Finally, taking the limit of both sides of this inequality yields

$$\begin{aligned} g(\delta) = \lim_{\lambda' \to \lambda(\partial A)} g(\delta) &\leq \lim_{\lambda' \to \lambda(\partial A)} \left(1 + \frac{c\,\lambda(\partial A)\delta}{\mu(A)d}\right) \\ &= 1 + \frac{c\,\delta\lambda(\partial A)}{\mu(A)d}, \end{aligned}$$

and the lemma follows by definition of $g(\delta)$. □

For our subsequent results, we assume that a δ-packing for $\mathcal{X}$, $\mathcal{S} \subset \mathcal{X}$, is generated by GreedyPack (Alg. 1) for a predefined constant $\delta \in (0, \Delta(\mathcal{X})]$. We now employ Lemma 4 to establish the amount of overlap.

Lemma 5. *For all $x \in \mathcal{S}$, it follows that*

$$f(x) \subseteq F(\mathcal{B}_\delta(x)) \subseteq f(x)_{\delta K}, \tag{3.5}$$

where $f(x)_{\delta K}$ is the (δK)-fattening of $f(x)$, $\delta > 0$ is the constant used to construct $\mathcal{S}$, and K is the Lipschitz constant from Assumption 2.

Proof. By definition of $\mathcal{B}_\delta(x)$ and by Assumption 2, it follows that for all $y \in \mathcal{B}_\delta(x)$,

$$d_{\mathrm{H}}(f(x), f(y)) \leq K\left\|x - y\right\| \leq K\delta.$$

We have that $x \in \mathcal{B}_\delta(x)$, $f(x) \subseteq F(\mathcal{B}_\delta(x))$, and $f(x)$ is compact. Moreover for all $f(y)$, $y \in \mathcal{B}_\delta(x)$, $f(y)$ is also compact. Thus, it follows by definition of Hausdorff distance that the (δK)-fattening of $f(x)$, $f(x)_{\delta K}$ fully contains $f(y)$, i.e., $f(y) \subseteq f(x)_{\delta K}$. Since this holds for all $y \in \mathcal{B}_\delta(x)$, we have by definition of Hausdorff distance

$$d_{\mathrm{H}}(f(x), \cup_{y \in \mathcal{B}_\delta(x)} f(y)) = d_{\mathrm{H}}(f(x), F(\mathcal{B}_\delta(x))) \leq \delta K.$$

The lemma then follows by definition of Hausdorff distance in terms of (δK)-fattenings.

□

Lemma 6. *Suppose that the set $\mathcal{S} \subset \mathcal{X} \subseteq \mathbb{R}^d$ is constructed as previously described. Then, it follows that for all $x \in \mathcal{S}$*

$$F(\mathcal{X}) = \cup_{x \in \mathcal{S}} F(\mathcal{B}_\delta(x)) \subseteq \cup_{x \in \mathcal{S}} f(x)_{\delta K} = F(\mathcal{S})_{\delta K}, \tag{3.6}$$

where $f(x)_{\delta K}$ is the (δK)-fattening of $f(x)$, $F(\mathcal{S})_{\delta K}$ is the (δK)-fattening of $F(\mathcal{S})$, $\delta > 0$ is the constant used to construct $\mathcal{S}$, and K is the Lipschitz constant from Assumption 2.

Proof. Observe that since $\mathcal{S}$ is a δ-covering of $\mathcal{X}$, it follows that union of $|\mathcal{S}|$ balls of radius δ centered at each of points of $x \in \mathcal{S}$ forms a superset of $\mathcal{X}$. Recall by definition of the reachability function, $\forall x \notin \mathcal{X}, f(x) = \emptyset$. This enables us to conveniently deal with evaluation of points outside the domain of initial states, $\mathcal{X}$.

Now, consider $F(\mathcal{X})$ and note that by definition of f on input outside the domain of $\mathcal{X}$, we have

$$F(\mathcal{X}) = F\left(\cup_{x \in \mathcal{S}} \cup_{y \in \mathcal{B}_\delta(x)} y\right) = \cup_{x \in \mathcal{S}} F(\mathcal{B}_\delta(x)).$$

By Lemma 5, it follows that for any arbitrary $x \in \mathcal{S}$, $F(\mathcal{B}_\delta(x)) \subseteq f(x)_{\delta K}$. Taking the union over all $x \in \mathcal{S}$ on both sides, we obtain $\cup_{x \in \mathcal{S}} F(\mathcal{B}_\delta(x)) = \cup_{x \in \mathcal{S}} f(x)_{\delta K}$, and the lemma follows from the fact that $\cup_{x \in \mathcal{S}} f(x)_{\delta K} = F(\mathcal{S})_{\delta K}$. □

Theorem 7. *Given any $\varepsilon \in (0, 1)$ consider the δ-packing of $\mathcal{X}$, $\mathcal{S} \subset \mathcal{X} \subseteq \mathbb{R}^d$, generated by* GreedyPack *(Alg. 1) with parameter $\delta > 0$ satisfying*

$$\delta \leq \frac{d((1-\varepsilon)^{-1/d} - 1)}{\alpha K c},$$

where $\alpha = \sup_{\mathcal{X}' \subseteq \mathcal{X}} \frac{\lambda(\partial F(\mathcal{X}'))}{\mu(F(\mathcal{X}'))} < \infty$, and c is the universal constant from Lemma 3:

$c = \max\{M/(d\mu(\mathcal{B}_1(\cdot))^{1/d}), 1\}$. *Then, $\mathcal{S}$ solves the approximate reachability problem defined by* (3.1)*, i.e.,* $(1-\varepsilon)\mu(F(\mathcal{X})) \leq \mu(F(\mathcal{S})) \leq \mu(F(\mathcal{X}))$.

The following corollary follows immediately from the theorem established above.

Corollary 8. *Given a set of initial states $\mathcal{X} \subseteq \mathbb{R}^d$ and $\varepsilon \in (0,1)$,* ApproximateReachability *(Alg. 2) generates a reachable set $\hat{F}_{\mathcal{S}}$ such that $\mu(\hat{F}_{\mathcal{S}}) \geq (1-\varepsilon)\mu(F(\mathcal{X}))$.*

Corollary 9. *For all $\varepsilon \in (0,1)$, the reachability problem defined by* (3.1) *can be solved by a δ-packing, $\mathcal{S} \subset \mathcal{X}$ of size at most $|\mathcal{S}| \leq (3\alpha K \Delta(\mathcal{X})c/\varepsilon)^d$.*

Let $\mathcal{C}_\alpha$, $\mathcal{C}_K$, and $\mathcal{C}_c$ denote the computational complexity of approximating the system-specific constants α (ratio of surface area to volume), K (Lipschitz constant), and c (universal constant from Lemma 3) respectively. Also let $\mathcal{C}_{\mathcal{S}}$ and $\mathcal{C}_f$ be upper bounds on the computational complexity of generating the δ-packing $\mathcal{S}$ and evaluating $f(x)$ for any $x \in \mathcal{X}$, respectively. Application of Corollary 9 yields the following theorem.

Theorem 10. *Given a set of initial states $\mathcal{X} \subseteq \mathbb{R}^d$ and $\varepsilon \in (0,1)$,* ApproximateReachability *(Alg. 2) generates a reachable set $\hat{F}_{\mathcal{S}}$ such that $\mu(\hat{F}_{\mathcal{S}}) \geq (1-\varepsilon)\mu(F(\mathcal{S}))$, in $\mathcal{O}\left(\mathcal{C}_\alpha + \mathcal{C}_K + \mathcal{C}_c + \mathcal{C}_{\mathcal{S}} + (\alpha K \Delta(\mathcal{X})c/\varepsilon)^d \mathcal{C}_f\right)$ time.*

Define the sequence $(\mathcal{F}_i)_{i\in\mathbb{N}_+}$ such that for each $i \in \mathbb{N}_+$, $\mathcal{F}_i$ denotes the approximate reachable set $\hat{F}_{\mathcal{S}_i}$ generated at iteration i of Alg. 3, i.e., $\mathcal{F}_i = \hat{F}_{\mathcal{S}_i}$, where, $\mathcal{S}_i \subset \mathcal{X}$ is the packing generated at iteration i.

Proposition 11. *The* AnytimeApproximateReachability *algorithm (Alg. 3) is asymptotically-optimal, i.e.,* $\lim_{i\to\infty} \mu(\mathcal{F}_i) = \lim_{i\to\infty} \mu(\hat{F}_{\mathcal{S}_i}) = \mu(F(\mathcal{X}))$.

3.4.3 Simultaneous under and over approximations

Finally, we show that a (δK)-fattening of the reachable set generated by our algorithm simultaneously produces a bounded over-approximation, and we conclude with a corollary that combines both aspects of our approach.

Theorem 12 (Over-approximation)**.** *Given any $\varepsilon \in (0, 1/2)$ consider the δ-packing of $\mathcal{X}$, $\mathcal{S} \subset \mathcal{X} \subseteq \mathbb{R}^d$, generated by* GreedyPack *(Alg. 1) with parameter $\delta > 0$ satisfying*

$$\delta \leq \frac{d((1+\varepsilon)^{1/d} - 1)}{\alpha K c},$$

then, the (δK)-fattening of $F(\mathcal{S})$, denoted by $F(\mathcal{S})_{\delta K}$ satisfies

$$F(\mathcal{X}) \subseteq F(\mathcal{S})_{\delta K},$$

and

$$\mu(F(\mathcal{X})) \leq \mu(F(\mathcal{S})_{\delta k}) \leq (1+\varepsilon)\mu(F(\mathcal{X})).$$

Proof. Since $\mathcal{S}$ is a δ-covering of $\mathcal{X}$, by Lemma 6, we have $F(\mathcal{X}) \subseteq F(\mathcal{S})_{\delta K}$. Thus, by monotonicity of measure and by application of Lemma 4 to the (δK)-fattening of $F(\mathcal{S})$, we have

$$\begin{aligned}
\mu(F(\mathcal{X})) &\leq \mu(F(\mathcal{S})_{\delta K}) \\
&\leq \left(1 + \frac{c\,\delta K \lambda(\partial F(\mathcal{S}))}{\mu(F(\mathcal{S}))d}\right)^d \mu(F(\mathcal{S})) \\
&\leq \left(1 + \frac{c\,\delta K \alpha}{d}\right)^d \mu(F(\mathcal{S})) \\
&\leq (1+\varepsilon)\mu(F(\mathcal{S})) \\
&\leq (1+\varepsilon)\mu(F(\mathcal{X})),
\end{aligned}$$

where the second-to-last inequality follows by our choice of δ and the last inequality follows by monotonicity of measure. □

Corollary 13 (Simultaneous under and over approximations)**.** *Given any $\varepsilon \in (0, 1/2)$ consider the δ-packing of $\mathcal{X}$, $\mathcal{S} \subset \mathcal{X} \subseteq \mathbb{R}^d$, generated by* GreedyPack *(Alg. 1) with parameter $\delta > 0$ satisfying*

$$\delta \leq \frac{d((1+\varepsilon)^{1/d} - 1)}{\alpha K c},$$

then, $F(\mathcal{X}) \subseteq F(\mathcal{S})_{\delta K}$ *and*

$$(1-\varepsilon)\mu(F(\mathcal{X})) \leq \mu(F(\mathcal{S})) \leq \mu(F(\mathcal{S})_{\delta K}) \leq (1+\varepsilon)\mu(F(\mathcal{X})).$$

Proof. The proof follows from the fact that for $\varepsilon \in (0, 1/2)$, we have

$$\delta \leq \frac{d((1+\varepsilon)^{1/d}-1)}{\alpha K c} \leq \frac{d((1-\varepsilon)^{-1/d}-1)}{\alpha K c},$$

which means that our choice of δ is simultaneously less than the value required to obtain both an under and over approximation. □

3.5 Results

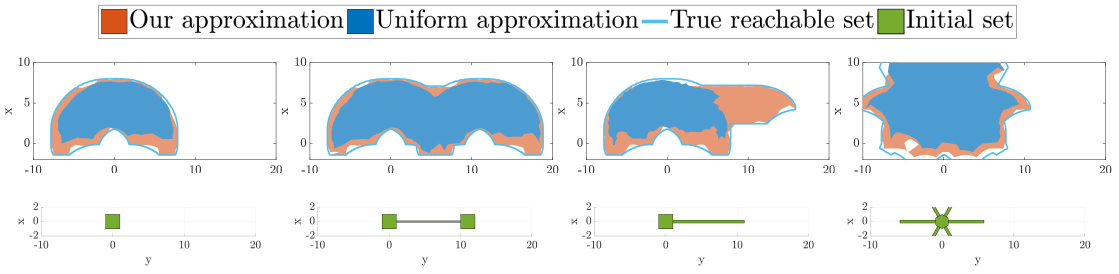


Figure 3-6: Clockwise: Set of initial conditions and the resulting reachable set for the unit cube, dumbbell, lollipop, and hedgehog scenarios. We compare the reachable sets of uniform sampling, our algorithm, and the ground truth. The visualizations show that uniform sampling initial states performs poorly when the set of initial states has an uneven distribution of volume.

We apply our approximation algorithm to simulated scenarios with a diverse set of initial states (see Fig. 3-6), where the objective is to generate the reachable set of a unicycle model. We evaluate the performance of our algorithm and compare the generated reachable sets to the ground truth reachable set, which can be readily computed for a unicycle model [Dub57]. We show that the theoretical guarantees hold and compare the performance of our algorithm with that of uniform sampling. We implemented our reachability algorithm in MATLAB. The simulations were conducted on a PC with a 2.60 GHz Intel i9-7980XE processor (single core) and 128 GB RAM.

3.5.1 Experimental Setup

We consider the reachable set $F(\cdot)$ of a mobile robot described by the unicycle dynamics:

$$\frac{\mathrm{d}}{\mathrm{dt}}\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} u_v \cos\theta \\ u_v \sin\theta \\ u_\omega \end{pmatrix}, \tag{3.7}$$

where $x, y \in \mathbb{R}$ denote the position of the robot and $\theta \in \mathbb{R}$ the orientation, and the control inputs $(u_v, u_\omega) \in \mathcal{U} \subset \mathbb{R}^2$ of the system are given by the speed and angular velocity, respectively. We are interested in the reachable set $F(\mathcal{X})$, where $\mathcal{X} \in \mathbb{R}^3$ denotes the set of initial conditions for which we want to approximate the reachable set at a given time T.

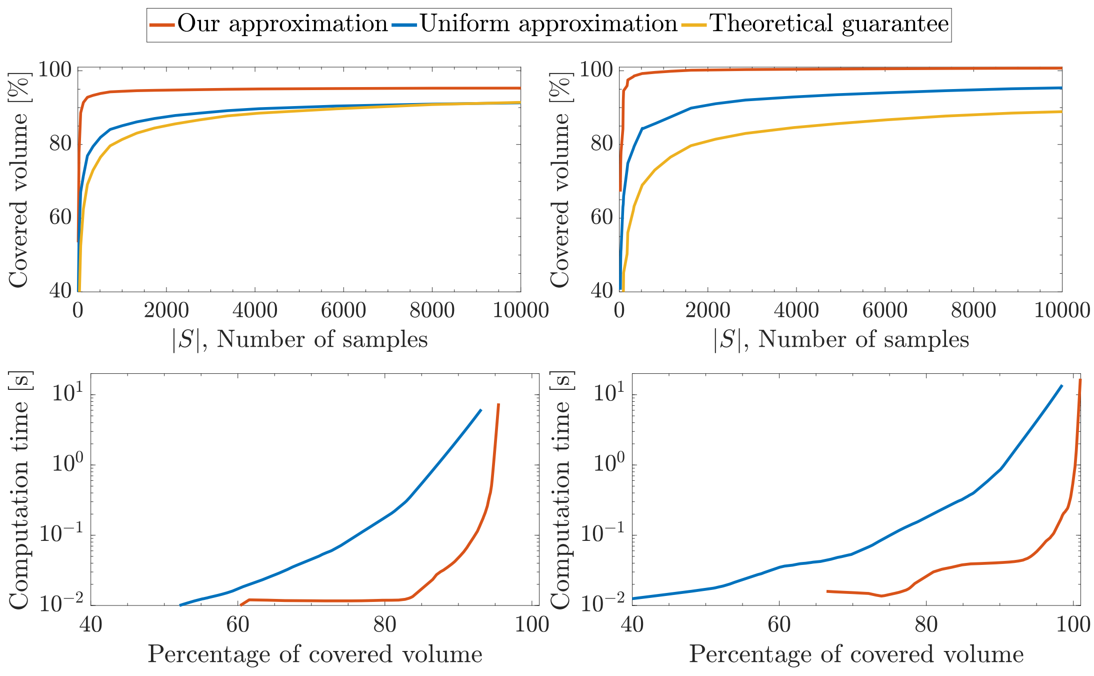


Figure 3-7: Comparisons of the performance of our algorithm with that of uniform sampling for the unit cube scenario (first column) and the dumbbell scenario (second column). The corresponding scenarios are depicted in the first and second column of Fig. 3-6, respectively.

We note that the reachable set for a unicycle model with minimal turning radius ρ and velocity u_v is known [Dub57, PPF03] and that the boundary of the set can be described by a set of curves consisting of straight segments (S) as well as left turns (L) and right turns (R) at the maximum turning radius. In particular, the reachable

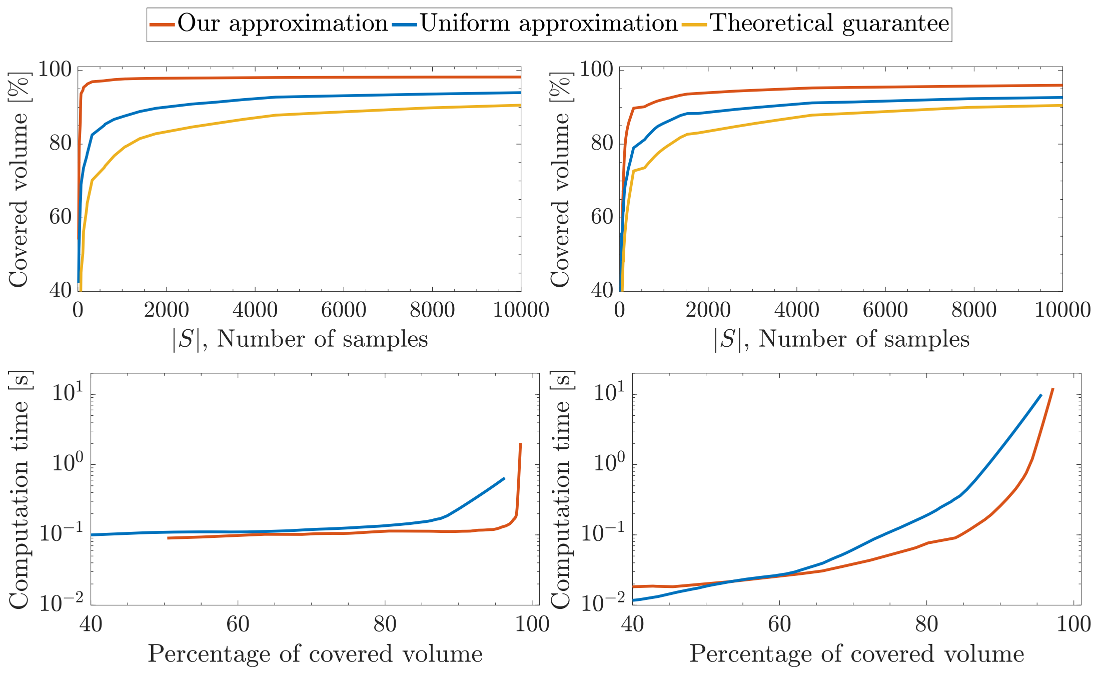


Figure 3-8: The performance of the evaluated reachability analysis methods for the lollipop scenario (first column) and the hedgehog scenario (second column). The evaluated scenarios are shown in the third and fourth column of Fig. 3-6, respectively.

set consists of the curves RLR, LRL, RSR, LSL, RSL, and LSR. As an exemplary parametrization for these curves, we give the parametrization for the curve RSL:

$$\begin{pmatrix} x_{RSL} \\ y_{RSL} \\ \theta_{RSL} \end{pmatrix} = \rho \begin{pmatrix} 2\sin(\theta_1) + \theta_2\cos(\theta_1) - \sin(\theta_1 - \theta_3) \\ -1 + 2\cos(\theta_1) - \theta_2\sin(\theta_1) - \cos(\theta_1 - \theta_3) \\ \theta_1/\rho - \theta_2/\rho \end{pmatrix},$$

where $\theta_i = \frac{u_v t_i}{\rho}, \forall i \in \{1,2,3\}$ and t_1 denotes the time in segment R, t_2 in segment S, and t_3 in segment L, respectively. Note that $t_1 + t_2 + t_3 \leq T$, where T is the given time as before, and this solution can be shown to be extended for a range of velocities $u_v \in [v_{min}, v_{max}]$, [PPF03].

Using the unicycle model allows us to compare the algorithm to the ground truth reachable set in an exact manner.

3.5.2 Evaluation of Computed Reachable Sets

We evaluated the performance of our algorithm and uniform sampling against a set of diverse initial states: (i) unit cube, (ii) dumbbell, (iii) lollipop, and (iv) hedgehog. To increase the efficiency of our implementation, we replaced the random construction of a δ-covering by a grid construction. The terminal time T was taken to be 1 second.

Fig. 3-6 depicts the projections onto (x, y) of the four sets of considered initial conditions, the respective reachable sets computed by uniform sampling and our algorithm, and the ground truth sets. The visualizations of the computed reachable sets show that uniform sampling may be a reasonable approximation for convex sets such as the unit cube, but its performance suffers significantly when non-convex sets, with non-uniformly distributed volumes are considered, such as the dumbbell or lollipop. Unlike uniform sampling, our algorithm still generates highly accurate reachable sets when evaluated against scenarios with non-uniform and/or non-convex initial states, which underlines the significance of judiciously generating a structured set of points as is done by our algorithm. Similar scenarios might arise in real-world situations, where dynamic obstacles are present that constrain the reachable space to non-convex, irregular regions.

To quantify the quality of the generated reachable sets, we ran our algorithm and uniform sampling against each scenario and averaged the results over 10 trials. Fig. 3-7 depicts the performance of uniform sampling and our algorithm in computing approximate reachable sets for the unit cube and dumbbell scenarios. Our results indicate that our algorithm is capable of generating higher quality approximations of the reachable set with a fewer amount of samples when compared to uniform sampling. Fig. 3-8 shows the results of evaluation against the lollipop and hedgehog scenarios with respect to volumetric coverage of the reachable set and the computation time. Since the sets of initial states exhibit highly non-uniform distribution of volume and are non-convex, we once again observe the significant gap in performance of uniform sampling when compared to the quality of approximations generated by our algorithm.

We note that across all experiments, our theoretical bounds of volumetric coverage hold and that the computation time required by our algorithm is significantly less than that required by uniform sampling for the same approximation accuracy. In particular, we note that the computation time required to generate the reachable set of the sampled subset (δ-packing) is near real-time. Our algorithm's favorable performance with respect to both approximation quality and computational efficiency on a wide variety of scenarios and non-convex initial states highlights its applicability to real-world motion planning and decision-making problems of autonomous systems.

3.6 Proofs

3.6.1 Proof of Lemma 3

Lemma 3. *Let $A \subset \mathbb{R}^d$ be a non-empty compact set with finite diameter and let $\delta \in (0, \Delta(\mathcal{X})]$. If $\mu(A) > 0$ and the δ-fattening of A, A_δ, is $(d-1)$-rectifiable for all $\delta \in (0, \Delta(\mathcal{X})]$, then there exists a finite universal constant $c \geq 1$:*

$$c = \max\{M/(d\mu(\mathcal{B}_1(\cdot))^{1/d}), 1\},$$

where $M > 0$ is independent of δ and A, such that

$$\frac{\lambda(\partial A_\delta)}{\mu(A_\delta)^{(d-1)/d}} \leq \frac{c\,\lambda(\partial A)}{\mu(A)^{(d-1)/d}},$$

where c is independent of δ and A, and $\lambda(\partial A)$ is the Minkowski Content as defined in (3.2).

Proof. Fix the finite universal constant $c > 0$, to be determined later. By the Isoperimetric inequality [Fed69, Theorem 3.2.43], the right hand side of the inequality is bounded below by a universal constant independent of δ and A:

$$\frac{c\,\lambda(\partial A)}{\mu(A)^{(d-1)/d}} \geq c\,d\mu(\mathcal{B}_1(\cdot))^{1/d}.$$

Moreover, since A_δ is compact, $(d-1)$-rectifiable, and $\mu(A_\delta) > 0$, for all $\delta \in (0, \Delta(\mathcal{X})]$, we have that $\lambda(\partial A_\delta)$ is equivalent to the $d-1$ Hausdorff measure of A_δ [Fed69, Theorem 3.2.39] which is finite by properties of A_δ [Mor16]. Thus, the left hand side is bounded above by a universal finite value, i.e., there exists a universal upper bound $M > 0$, independent of A and δ, for the left hand side of the inequality such that

$$\frac{\lambda(\partial A_\delta)}{\mu(A_\delta)^{(d-1)/d}} \leq M.$$

Thus, define the universal constant c to be

$$c = \max\{M/(d\mu(\mathcal{B}_1(\cdot))^{1/d}), 1\},$$

and note that this choice of c yields

$$\frac{c\,\lambda(\partial A)}{\mu(A)^{(d-1)/d}} \geq c\,d\mu(\mathcal{B}_1(\cdot))^{1/d} \geq M \geq \frac{\lambda(\partial A_\delta)}{\mu(A_\delta)^{(d-1)/d}},$$

as desired. □

3.6.2 Proof of Theorem 7

Theorem 7. *Given any $\varepsilon \in (0,1)$ consider the δ-packing of $\mathcal{X}$, $\mathcal{S} \subset \mathcal{X} \subseteq \mathbb{R}^d$, generated by* GreedyPack *(Alg. 1) with parameter $\delta > 0$ satisfying*

$$\delta \leq \frac{d((1-\varepsilon)^{-1/d} - 1)}{\alpha K c},$$

where

$$\alpha = \sup_{\mathcal{X}' \subseteq \mathcal{X}} \frac{\lambda(\partial F(\mathcal{X}'))}{\mu(F(\mathcal{X}'))} < \infty,$$

and c is the universal constant from Lemma 3, $c = \max\{M/(d\mu(\mathcal{B}_1(\cdot))^{1/d}), 1\}$. Then, $\mathcal{S}$ solves the approximate reachability problem defined by (3.1), i.e., $(1-\varepsilon)\mu(F(\mathcal{X})) \leq \mu(F(\mathcal{S})) \leq \mu(F(\mathcal{X}))$.

Proof. Fix $\delta > 0$, to be determined later. Combining Lemmas 4 and 6, it follows that

$$\mu(F(\mathcal{X})) \leq \mu(F(\mathcal{S})_{\delta K}) \leq \left(1 + \frac{\delta K \alpha c}{d}\right)^d \mu(F(\mathcal{S})).$$

The inequality above yields

$$\mu(F(\mathcal{S})) \geq \left(1 + \frac{\delta K \alpha c}{d}\right)^{-d} \mu(F(\mathcal{X})).$$

Thus, to ensure that $\mu(F(\mathcal{S}))$ is a $(1-\varepsilon)$ approximation to $\mu(F(\mathcal{X}))$, we seek to set δ such that the following inequality holds:

$$\begin{aligned} &\left(1 + \frac{\delta K \alpha c}{d}\right)^{-d} \mu(F(\mathcal{X})) \geq (1-\varepsilon)\mu(F(\mathcal{X})) \\ \Leftrightarrow &\left(1 + \frac{\delta K \alpha c}{d}\right)^{-d} \geq 1 - \varepsilon. \end{aligned}$$

Solving for δ satisfying the inequality above yields

$$\delta \leq \frac{d((1-\varepsilon)^{-1/d} - 1)}{\alpha K c}.$$

The theorem follows from the fact that $\alpha < \infty$ since $F(\mathcal{X}')$ is compact and rectifiable for all $\mathcal{X}' \subseteq \mathcal{X}$ [Mor16], $d \geq 1$, $\varepsilon \in (0,1)$, and $\alpha K c > 0$, thus, the expression on the right hand-side is strictly positive, which completes the proof. □

3.6.3 Proof of Corollary 9

Corollary 9. *For all $\varepsilon \in (0,1)$, the reachability problem defined by (3.1) can be solved by a δ-packing, $\mathcal{S} \subset \mathcal{X}$ of size at most*

$$|\mathcal{S}| \leq (3\alpha K \Delta(\mathcal{X}) c/\varepsilon)^d = \mathcal{O}_{\alpha,K,\Delta(\mathcal{X}),c}(\varepsilon^{-d}),$$

where $\mathcal{O}_{\alpha,K,\Delta(\mathcal{X}),c}(\cdot)$ notation suppresses α, K, c, and $\Delta(\mathcal{X})$ factors.

Proof. Plugging in the expression for δ from Theorem 7 with equality, i.e., $\delta = \frac{d(1-\varepsilon)^{-1/d}-d}{\alpha K c}$, into the expression from Lemma 2 yields

$$|\mathcal{S}| \leq \left(\frac{3\alpha K \Delta(\mathcal{X}) c}{d((1-\varepsilon)^{-1/d}-1)} \right)^d.$$

Now, since $\varepsilon \in (0,1)$ and $-1/d < 0$, it follows by Bernoulli's inequality that $(1-\varepsilon)^{-1/d} \geq 1 + \varepsilon/d$. Plugging this lower bound for the denominator in the expression above establishes the result. $\square$

3.6.4 Proof of Proposition 11

Proposition 11. *The* AnytimeApproximateReachability *algorithm (Alg. 3) is asymptotically-optimal, i.e.,*

$$\lim_{i\to\infty} \mu(\mathcal{F}_i) = \lim_{i\to\infty} \mu(\hat{F}_{\mathcal{S}_i}) = \mu(F(\mathcal{X})).$$

Proof. Let ε_i denote the value of ε at the beginning of the ith iteration of the algorithm. Since the value of ε is halved after each iteration, it follows immediately that $\varepsilon_i = 2^{-i}$ and therefore $\lim_{i\to\infty} \varepsilon_i = 0$. Thus,

$$\lim_{i\to\infty} \mu(\mathcal{F}_i) \geq \lim_{i\to\infty} (1-\varepsilon_i)\mu(F(\mathcal{X})) = \mu(F(\mathcal{X})).$$

Moreover, by monotonicity of measure $\mu(\mathcal{F}_i) \leq \mu(F(\mathcal{X}))$ for all i, thus it must be the case that $\lim_{i\to\infty} \mu(\mathcal{F}_i) = \mu(F(\mathcal{X}))$. $\square$

3.7 Discussion and Future Work

In this chapter, we presented a sampling-based approach to reachability analysis that imposes minimal assumptions and can be applied to a wide variety of systems. Our algorithm enables computational efficiency by computing the reachable set of a carefully constructed finite subset of initial states that provides a covering of the entire state space. We proved that our algorithm generates an approximation to the ground-truth reachable set that is approximately optimal up to any desired approximation accuracy.

Our favorable results in real-world inspired scenarios validate the favorable theoretical properties of our algorithm and demonstrate its applicability to a diverse set of reachability problems. We envision that our method can be used to conduct reachability analysis to facilitate decision-making and trajectory planning for autonomous agents in a wide variety of application including autonomous driving, parallel autonomy, and supervision of deep learning-based planning systems. In future work, we plan to extend our algorithm and analysis to obtain both under- and over-approximations of reachable sets with provable guarantees.

Acknowledgments

This research was supported in part by the Toyota Research Institute (TRI) and National Science Foundation award IIS-1723943. This chapter solely reflects the opinions and conclusions of its authors, and not TRI or any other Toyota entity.

Chapter 4

Streaming Coresets for Support Vector Machines

4.1 Overview

In the previous chapter, we sampled a *finite*, representative set from an *infinite* input space of reachable states for approximate reachability analysis. In this chapter, we focus on subsampling data points from a finite set of input training points for efficient machine learning. In particular, we introduce an efficient coreset construction algorithm to generate compact representations of large data sets to accelerate SVM training. Unlike our set-coverage-based approach from the previous chapter, here we leverage the notion of coresets and use importance sampling of data points according to their sensitivities [BFL16]. In contrast to prior approaches in efficient SVM training, our approach is both (i) provably efficient and (ii) naturally extends to streaming or dynamic data settings. Above all, it can be used *to enable the applicability of any off-the-shelf SVM solver* – including gradient-based and/or approximate ones, e.g., Pegasos [SSSSC11] – to streaming and distributed data settings by exploiting the *composibility* and *reducibility* properties of coresets [Fel19].

This work is based on [BTFR21, TBFR20] and contributes the following:

1. A coreset construction algorithm for accelerating SVM training based on an efficient importance sampling scheme.

2. An analysis proving lower bounds on the number of samples required by any coreset construction algorithm to approximate the input data set.

3. A corollary to our analysis that provides justification for the widely reported empirical success of using k-means clustering as a way to generate data summaries for large-scale SVM training.

4. Theoretical guarantees on the efficiency and accuracy of our coreset construction algorithm.

5. Evaluations on synthetic and real-world data sets that demonstrate the effectiveness of our algorithm in both streaming and offline settings.

4.2 Setting & Objective

4.2.1 Setting

Let $P = \big\{(x, y) : x \in \mathbb{R}^d \times 1, y \in \{\pm 1\}\big\}$ denote a set of n input points. Note that for each point $p = (x, y) \in P$, the last entry $x_{d+1} = 1$ of x accounts for the bias term embedding into the feature space[1]. To present our results with full generality, we consider the setting where the input points P may have weights associated with them. Hence, given P and a weight function $u : P \to \mathbb{R}_{\geq 0}$, we let $\mathcal{P} = (P, u)$ denote the weighted set with respect to P and u. The canonical unweighted case can be represented by the weight function that assigns a uniform weight of 1 to each point, i.e., $u(p) = 1$ for every point $p \in P$. For every $T \subseteq P$, let $U(T) = \sum_{p \in T} u(p)$. We consider the scenario where n is much larger than the dimension of the data points, i.e., $n \gg d$.

For a normal to a separating hyperplane $w \in \mathbb{R}^{d+1}$, let $w_{1:d}$ denote vector which

[1]We perform this embedding for ease of presentation later on in our analysis.

contains the first d entries of w. The last entry of w (w_{d+1}) encodes the bias term $b \in \mathbb{R}$. Under this setting, the hinge loss of any point $p = (x, y) \in P$ with respect to a normal to a separating hyperplane, $w \in \mathbb{R}^{d+1}$, is defined as $h(p, w) = [1 - y\langle x, w\rangle]_+$, where $[\cdot]_+ = \max\{0, \cdot\}$. As a prelude to our subsequent analysis of sensitivity-based sampling, we quantify the contribution of each point $p = (x, y) \in P$ to the SVM objective function as

$$f_\lambda(p, w) = \frac{1}{2U(P)} \|w_{1:d}\|_2^2 + \lambda h(p, w), \tag{4.1}$$

where $\lambda \in [0, 1]$ is the SVM regularization parameter, and $h(p, w) = [1 - y\langle x, w\rangle]_+$ is the hinge loss with respect to the query $w \in \mathbb{R}^{d+1}$ and point $p = (x, y)$. Putting it all together, we formalize the λ-regularized SVM problem as follows.

Definition 12 (λ-regularized SVM Problem)**.** *For a given weighted set of points $\mathcal{P} = (P, u)$ and a regularization parameter $\lambda \in [0, 1]$, the λ-regularized SVM problem with respect to $\mathcal{P}$ is given by*

$$\min_{w \in \mathbb{R}^{d+1}} F_\lambda(\mathcal{P}, w),$$

where

$$F_\lambda(\mathcal{P}, w) = \sum_{p \in \mathcal{P}} u(p) f_\lambda(p, w). \tag{4.2}$$

We let w^* denote the optimal solution to the SVM problem with respect to $\mathcal{P}$, i.e., $w^* \in \operatorname{argmin}_{w \in \mathbb{R}^{d+1}} F_\lambda(\mathcal{P}, w)$. A solution $\hat{w} \in \mathbb{R}^{d+1}$ is an ξ-approximation to the SVM problem if $F_\lambda(\mathcal{P}, \hat{w}) \leq F_\lambda(\mathcal{P}, w^*) + \xi$. Next, we formalize the *coreset guarantee* that we will strive for when constructing our data summaries.

4.2.2 Coresets

Here, we formalize the notion of coresets presented in the previous chapters by defining it specifically for the SVM problem. Recall that a coreset is a compact representation of the full data set that provably approximates the SVM cost function (4.2) for *every query* $w \in \mathbb{R}^{d+1}$ – including that of the optimal solution w^*. This is fleshed out in the definition below for the SVM problem with objective function $F_\lambda(\cdot)$ as in (4.2).

Definition 13 (ε-coreset)**.** *Let $\varepsilon \in (0,1)$ and let $\mathcal{P} = (P, u)$ be the weighted set of training points as before. A weighted subset $\mathcal{S} = (S, v)$, where $S \subset P$ and $v : S \to \mathbb{R}_{\geq 0}$ is an ε-coreset for $\mathcal{P}$ if*

$$\forall w \in \mathbb{R}^{d+1} \quad \left|F_\lambda\left(\mathcal{P}, w\right) - F_\lambda\left(\mathcal{S}, w\right)\right| \leq \varepsilon F_\lambda\left(\mathcal{P}, w\right). \tag{4.3}$$

This strong guarantee implies that the models trained on the coreset $\mathcal{S}$ with *any* off-the-shelf SVM solver will be approximately (and provably) as good as the optimal solution w^* obtained by training on the entire data set $\mathcal{P}$. We formalize this in the lemma below.

Lemma 14. *Let $\mathcal{P}$ be a weighted set, $\varepsilon \in \left(0, \frac{1}{2}\right)$ and let $\mathcal{S}$ be an ε-coreset with respect to $\mathcal{P}$. Let*

$$w^*_{\mathcal{S}} = \operatorname*{argmin}_{w \in \mathbb{R}^{d+1}} F_\lambda\left(\mathcal{S}, w\right)$$

be the optimal solution on the set $\mathcal{S}$ and let

$$w^*_{\mathcal{P}} = \operatorname*{argmin}_{w \in \mathbb{R}^{d+1}} F_\lambda\left(\mathcal{P}, w\right)$$

be defined similarly as the optimal solution with respect to points $\mathcal{P}$. Then, the performance of the solution $w^*_{\mathcal{S}}$ on the entire set $\mathcal{P}$ is $(1+4\varepsilon)$-competitive with that of the ground-truth optimal solution $w^*_{\mathcal{P}}$, i.e.,

$$F_\lambda(\mathcal{P}, w^*_{\mathcal{S}}) \leq (1+4\varepsilon) F_\lambda(\mathcal{P}, w^*_{\mathcal{P}}).$$

Proof. We have

$$\begin{aligned} F_\lambda(\mathcal{P}, w^*_{\mathcal{S}}) &\leq \frac{F_\lambda(\mathcal{S}, w^*_{\mathcal{S}})}{1-\varepsilon} \leq \frac{F_\lambda(\mathcal{S}, w^*_{\mathcal{P}})}{1-\varepsilon} \\ &\leq \frac{1+\varepsilon}{1-\varepsilon} F_\lambda(\mathcal{P}, w^*_{\mathcal{P}}) \leq (1+4\varepsilon) F_\lambda(\mathcal{P}, w^*_{\mathcal{P}}), \end{aligned}$$

where the first and third inequalities follow from (S, v) being an ε-coreset (see Definition 13), the second inequality holds by definition of $w^*_{\mathcal{S}}$, and the last inequality follows from the assumption that $\varepsilon \in \left(0, \frac{1}{2}\right)$. $\square$

The lemma above implies that, if the size of the coreset $\mathcal{S}$ is provably small relative to $\mathcal{P}$, e.g., logartihmic in n (see Sec. 4.4), then an approximately optimal solution can be obtained much more quickly by training on $\mathcal{S}$ rather than $\mathcal{P}$, leading to computational gains in practice for both offline and streaming data settings (see Sec. 4.6).

The difficulty in constructing coresets lies in constructing them (i) *efficiently*, so that the preprocess-then-train pipeline takes less time than training on the full data set and (ii) *accurately*, so that important data points – i.e., those that are imperative to obtaining accurate models – are not left out of the coreset, and redundant points are eliminated so that the coreset size is small. In the following sections, we introduce and analyze our coreset algorithm for the SVM problem.

4.3 Our Approach

Our coreset construction scheme is based on the unified framework of [FL11, LS10] and is shown in Alg. 4. The crux of our algorithm lies in generating the importance sampling distribution via efficiently computable upper bounds (proved in Sec. 4.4) on the importance of each point (Lines 1–10). Sufficiently many points are then sampled from this distribution and each point is given a weight that is inversely proportional to its sample probability (Lines 11–12). The number of points required to generate an ε-coreset with probability at least $1 - \delta$ is a function of the desired accuracy ε, failure probability δ, and complexity of the data set (t from Theorem 19). Under mild assumptions on the problem at hand (see Sec. 4.8.4), the required sample size is polylogarithmic in n.

Our algorithm is an importance sampling procedure that first generates a judicious sampling distribution based on the structure of the input points and samples sufficiently

Algorithm 4 $\textsc{Coreset}(P, u, \lambda, \xi, k, m)$

```
Inputs: $(P, u, \lambda, \xi, k, m)$ – A set of training points $P \subseteq \mathbb{R}^{d+1} \times \{-1, 1\}$ containing
    $n$ points, weight function $u : P \to \mathbb{R}_{\geq 0}$, a regularization parameter $\lambda \in [0, 1]$, an
    approximation factor $\xi > 0$, a positive integer $k$, a sample size $m$
Output: A weighted set $(S, v)$ which satisfies Theorem 19
1: $\tilde{w} \leftarrow$ An $\xi$-approximation for the optimal SVM of $(P, u)$;
2: $\widetilde{opt}_\xi \leftarrow F_\lambda(\mathcal{P}, \tilde{w}) - \xi$;
3: for $y \in \{-, +\}$ do
4:     $P_y \leftarrow$ all the points in $P$ that are associated with the label $y$;
5:     $\left(c_y^{(i)}, P_y^{(i)}\right)_{i=1}^{k} \leftarrow$ K-MEANS++$(\mathcal{P}_y,\ k)$; where $c_y^{(i)}$ is the centroid of the $i$th
   cluster;
6:     for every $i \in [k]$ do
7:         $\alpha_y^{(i)} \leftarrow \frac{U\left(P \setminus P_y^{(i)}\right)}{2\lambda U(P) U\left(P_y^{(i)}\right)}$;
8:         for every $p = (x, y) \in P_y^{(i)}$ do
9:             $p_\Delta \leftarrow y(c_y^{(i)} - x)$;
10:            $\gamma(p) \leftarrow \frac{u(p)}{U\left(P_y^{(i)}\right)} + \frac{9\lambda u(p)}{2} \max\left\{\frac{4\alpha_y^{(i)}}{9}, \sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\|p_\Delta\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)}\right\}$;
11: $t \leftarrow \sum_{p \in P} \gamma(p)$;
12: $(S, v) \leftarrow m$ weighted samples from $\mathcal{P} = (P, u)$ where each point $p \in P$ is sampled
    with probability $q(p) = \frac{\gamma(p)}{t}$ and, if sampled, has weight $v(p) = \frac{u(p)}{mq(p)}$;
13: return $(S, v)$;
```

many points from the original data set. The resulting weighted set of points $\mathcal{S} = (S, v)$, serves as an unbiased estimator for $F_\lambda(\mathcal{P}, w)$ for any query $w \in \mathbb{R}^{d+1}$, i.e., $\mathbb{E}\left[F_\lambda\left(\mathcal{S}, w\right)\right] = F_\lambda(\mathcal{P}, w)$. Although sampling points uniformly with appropriate weights can also generate such an unbiased estimator, it turns out that the variance of this estimation is minimized if the points are sampled according to the distribution defined by the ratio between each point's sensitivity and the sum of sensitivities, i.e., $\gamma(p)/t$ on Line 12 [BLK17].

4.3.1 Computational Complexity

Coresets are intended to provide efficient and provable approximations to the optimal SVM solution. However, the very first line of our algorithm entails computing an (approximately) optimal solution to the SVM problem. This seemingly eerie phenomenon is explained by the merge-and-reduce technique [HPM04] that ensures that our coreset algorithm is only run against small partitions of the original data set [BFL16, HPM04, LFKF17]. The merge-and-reduce approach leverages the fact that coresets are composable and reduces the coreset construction problem for a (large) set of n points into the problem of computing coresets for $\frac{n}{2|S|}$ points, where $2|S|$ is the minimum size of input set that can be reduced to half using Algorithm 4 [BFL16]. Assuming that the sufficient conditions for obtaining polylogarithmic size coresets implied by Theorem 19 hold, the overall time required is approximately linear in n.

4.4 Analysis

In this section, we analyze the sample-efficiency and computational complexity of our algorithm. The outline of this section is as follows: we first formalize the importance (i.e., *sensitivity*) of each point and summarize the necessary conditions for the existence of small coresets. We then present the negative result that, in general, sublinear coresets do not exist for *every* data set (Lem. 16). Despite this, we show that we can obtain accurate approximations for the sensitivity of each point via an approximate k-means clustering (Lems. 17 and 18), and present non-vacuous, data-dependent bounds on the sample complexity (Thm. 19). We defer all proofs to Sec. 4.8 for clarity of exposition.

4.4.1 Preliminaries

We will henceforth state all of our results with respect to the weighted set of training points $\mathcal{P} = (P, u)$, $\lambda \in [0, 1]$, and SVM cost function F_λ (as in Sec. 4.2). The definition below rigorously quantifies the *relative contribution* of each point.

Definition 15 (Sensitivity [BFL16])**.** *The sensitivity of each point $p \in P$ is given by*

$$s(p) = \sup_w \frac{u(p) f_\lambda(p, w)}{F_\lambda(\mathcal{P}, w)}. \tag{4.4}$$

Note that in practice, exact computation of the sensitivity is intractable, so we usually settle for (sharp) upper bounds on the sensitivity $\gamma(p) \geq s(p)$ (e.g., as in Alg. 4). Sensitivity-based importance sampling then boils down to normalizing the sensitivities by the normalization constant – to obtain an importance sampling distribution – which in this case is the *sum of sensitivities* $t = \sum_{p \in P} s(p)$. It turns out that the required size of the coreset is at least linear in t [BFL16], which implies that one immediate necessary condition for sublinear coresets is $t \in o(n)$.

4.4.2 Lower bound for Sensitivity

The next lemma shows that a sublinear-sized coreset cannot be constructed for *every* SVM problem instance. The proof of this result is based on demonstrating a hard point set for which the sum of sensitivities is $\Omega(n\lambda)$, ignoring d factors, which implies that sensitivity-based importance sampling roughly boils down to uniform sampling for this data set. This in turn implies that if the regularization parameter is too large, e.g., $\lambda = \theta(1)$, and if $d \ll n$ (as in Big Data applications) then the required number of samples for property (4.3) to hold is $\Omega(n)$.

Lemma 16. *For an even integer $d \geq 2$, there exists a set of weighted points $\mathcal{P} = (P, u)$ such that*

$$s(p) \geq \frac{n\lambda + d^2}{n\,(\lambda + d^2)} \qquad \forall p \in P \qquad \textit{and} \qquad \sum_{p \in P} s(p) \geq \frac{n\lambda + d^2}{(\lambda + d^2)}.$$

Next, we provide upper bounds on the sensitivity of each data point with respect to the complexity of the input data. Despite the non-existence results established above, our upper bounds shed light into the class of problems for which small-sized coresets are ensured to exist.

4.4.3 Sensitivity Upper Bound

In this subsection we present sharp, data-dependent upper bounds on the sensitivity of each point. Our approach is based on an approximate solution to the k-means clustering problem and to the SVM problem itself (as in Alg. 4). To this end, we will henceforth let k be a positive integer, $\xi \in [0, F_\lambda(\mathcal{P}, w^*)]$ be the error of the (coarse) SVM approximation, and let $(c_y^{(i)}, P_y^{(i)})$, $\alpha_y^{(i)}$ and p_Δ for every $y \in \{+,-\}$, $i \in [k]$ and $p \in P$ as in Lines 4–9 of Algorithm 4.

The next lemma leverages the clusters found by an approximate k-median solution and an approximate SVM solution to bound the relative importance, i.e., sensitivity, of each data point.

Lemma 17. *Let k be a positive integer, $\xi \in [0, F_\lambda(\mathcal{P}, w^*)]$, and let $\mathcal{P} = (P, u)$ be a weighted set. Then for every $i \in [k]$, $y \in \{+,-\}$ and $p \in P_y^{(i)}$,*

$$s(p) \leq \frac{u(p)}{U\left(P_y^{(i)}\right)} + \frac{9\lambda u(p)}{2} \max\left\{\frac{4\alpha_y^{(i)}}{9}, \sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\left\|p_\Delta\right\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)}\right\} = \gamma(p).$$

Next, we bound the *sum of sensitivities*, which is a quantity that appears in the application of concentration inequalities to our importance sampling scheme and dictates the sampling complexity of the problem.

Lemma 18. *In the context of Lemma 17, the sum of sensitivities is bounded by*

$$\sum_{p \in P} s(p) \leq t = 4k + \sum_{i=1}^{k} \frac{3\lambda\, Var_+^{(i)}}{\sqrt{2\widetilde{opt}_\xi}} + \frac{3\lambda\, Var_-^{(i)}}{\sqrt{2\widetilde{opt}_\xi}},$$

where $Var_y^{(i)} = \sum_{p \in P_y^{(i)}} u(p) \left\|p_\Delta\right\|_2$ for all $i \in [k]$ and $y \in \{+,-\}$.

Having bounded the sum of sensitivities from above, we now apply a seminal result of [BFL16] to establish the number of samples necessary under our algorithm to generate an ε-coreset. In the theorem below, we can see that the coreset size scales linearly in the sum of sensitivities t and explains the rationale for our effort in obtaining sharp sensitivity bounds.

Theorem 19. *For any $\varepsilon \in (0, 1/2), \delta \in (0, 1)$, let m be an integer satisfying*

$$m \in \Theta \left(\frac{t}{\varepsilon^2} \left(d \log t + \log(1/\delta) \right) \right),$$

where t is as in Lem. 18. Invoking Coreset *with the inputs defined in this context yields a ε-coreset $\mathcal{S} = (S, v)$ with probability at least $1 - \delta$ in $\mathcal{O}(nd + T)$ time, where T represents the computational complexity of obtaining an ξ-approximated solution to SVM and applying k-means++ on P_+ and P_-.*

Sufficient conditions and the effect of k-means on our sensitivity Theorem 19 immediately implies that, for reasonable ε and δ, coresets of poly-logarithmic (in n) size can be obtained if $d = \mathcal{O}(\text{polylog}(n))$, which is usually the case in our target Big Data applications, and if $\sum_{i=1}^{k} \frac{3\lambda \text{Var}_+^{(i)}}{\sqrt{2\widetilde{opt}_\xi}} + \frac{3\lambda \text{Var}_-^{(i)}}{\sqrt{2\widetilde{opt}_\xi}} = \mathcal{O}(\text{polylog}(n))$.

Despite the fact that any k-partitioning of the data can be applied instead of k-means for achieving an upper bound on the sensitivities of the points, it's important to note that k-means actually acts as a trade-off mechanism between the *raw contribution* and the *actual contribution* (the weight term and the max term from Lemma 17, respectively). Choosing the best k can be done via binary search over the values of k that minimize the sensitivity.

4.5 Intuition for k-means Clustering

Recall the bound from Lemma 17, and note that it was achieved by using k-means++ clustering as depicted at Algorithm 4. Following our analysis from Sec. 4.8.2, we observe that we can simply use any k-partitioning of the dataset, instead of applying k-means clustering. Moreover, we can also simply choose $k = 1$ which translates to simply taking the mean of the labeled points. Despite all of above, we did choose to use a clustering algorithm as well as having larger values of k, and such decisions were inspired by the following observations:

(i) k-means clustering aims to optimize the sum of squared distances between the points and their corresponding center, which on some level, helps in lowering the distance between points and their corresponding centers. Such observation leads to having tighter bounds for the sensitivity of each point, consequently leading to lower coreset sizes; See Thm. 19.

(ii) Having larger k also helps lowering the distance between a point and its corresponding center.

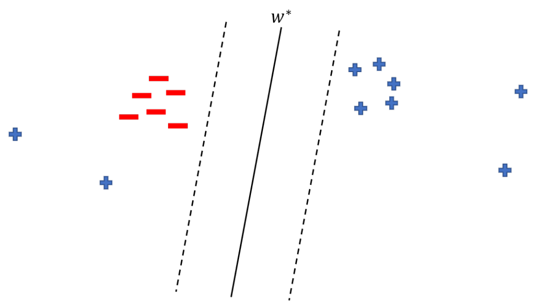
(iii) k-means clustering acts as a trade-off mechanism between

- the *raw contribution* of each point, which is translated into the weight of the point divided by the sum of the weights with respect to the cluster that each point is assigned to,

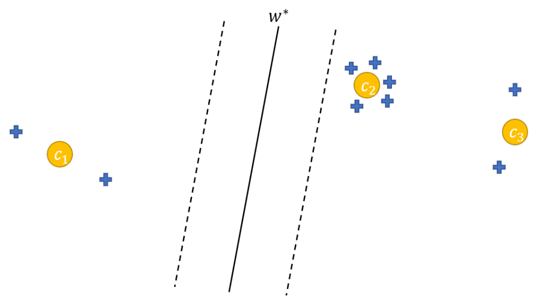
- and the *actual contribution* of each point which is translated to the distance between each point and its corresponding center.

In light of the above, we observe that as k goes larger, the sensitivity of each point gets closer and closer to being simply the *raw contribution*, which in case of unweighted data set, is simply applying uniform sampling. Thus, for each weighted (P, u), we simply chose $k = \log n$ where n denotes the total number of points in a P.

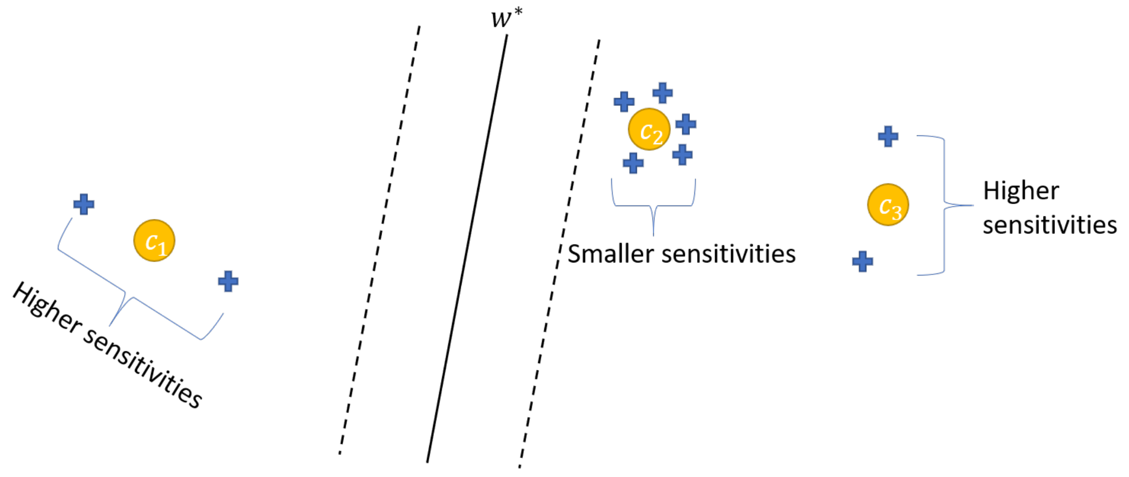
This observation helps in understanding how the sensitivities that we are providing for outliers and misclassified points actually quantifies the importance of such points. Specifically speaking, as k goes larger the outliers would mostly be assigned to the same cluster and since in general there aren't much of these points, we end up giving higher sensitivities for such points (than others) due to the fact that their *raw contribution* increases as the size of their corresponding cluster decreases. When k isn't large enough to separate these points from the rest of the data points, the *actual contribution* is used, which then results in shifting the mean of the cluster towards the "middle" between the outliers and the rest of the points in that cluster. This boosts the *actual contribution* of the rest of points inside the same cluster; See Fig. 4-1.



(a) At first, an optimal solution for the SVM problem is found.



(b) We focus on the the positive labeled points (blue points) and we find a k-means clustering using k-means++, where we set $k = 3$. The yellow points are the centers found by the k-means++ algorithm



(c) Points in small clusters have higher sensitivities, which quantifies the importance of outliers and misclassified points as they will be mostly in small clusters as k goes larger.

Figure 4-1: Understanding the effect of k-means on the sensitivities of the points.

4.5.1 Automating the Search for the (Approximately) Optimal k

To find the approximately optimal value of k, observe that the sum of sensitivities (see Lemma 18) contains two distinct expressions in terms of the number of clusters. By Lemma 18, they are (roughly) (1) k and (2) the sum of distances to the nearest cluster. Note that term (1) is increasing with k while (2) is decreasing with k. This means that we can perform a binary search over $[1, n]$ to find the best k.

However, a challenge here is that the clusterings are generated by k-means++ in practice, which is only $\mathcal{O}(\log k)$-competitive in expectation. Nevertheless, at the additional multiplicative cost of $\mathcal{O}(\log 1/\delta + \log\log n)$ time, we can amplify the success probability by running k-means++ for a clustering size of k $\Theta(\log((\log n)/\delta))$ times and picking the best k-clustering to obtain a $\mathcal{O}(\log k)$-competitive solution with probability at least $1 - \delta/\mathcal{O}(\log n)$ (by combining Markov's inequality and amplification). Doing this for every k we pick as we binary search, and accounting for the total $\log n$ iterations of binary search leads to a multiplicative factor increase of $\mathcal{O}\left(\log n \left(\log 1/\delta + \log\log n\right)\right)$ in the computation time. Using the $\mathcal{O}(\log k)$-competitiveness of *all* of the k-means instances implies that the binary search[2] generates a $\hat{k}$-clustering such that the sum of sensitivities is within $\Theta(\log n)$ factor of the minimum (over all $k \in [1, n]$) sum of sensitivities with respect to the best clustering, with probability[3] at least $1 - \delta/2$.

The additional concern here is the run-time of k-means++, which is $\mathcal{O}(ndk)$ for a clustering of size k [AV07]. To alleviate the dependence on d in the case of a high dimensional data set, we can use the (fast) Johnson-Lindenstrauss transform [CEM+15, IN07] on the input data points as a pre-processing step to reduce the dimensionality d to $\Theta(\log n/\varepsilon^2)$ at the cost of ε distortion in the pairwise distances between the points, while retaining the $\mathcal{O}(\log k)$-competitiveness of the k-means++ solution [MMR19]. This leads to at most $\mathcal{O}(nk\log n)$ time per k-means++ invocation. To further reduce the runtime in the case of large values of k, we can restrict the end point of our binary search to e.g., $k \in [1, \sqrt{n}]$ (or even $k \in [1, \mathcal{O}(\log n)]$), since any $k \geq \sqrt{n}$ would yield a sum of sensitivities that is $\geq \sqrt{n}$, and in turn, a sampling complexity of at least $\sqrt{n}$ – which would imply a higher-than-ideal coreset size in the first place. With these changes in place, the runtime per invocation of k-means++ reduces to $\mathcal{O}(nb\log n)$, where b ($\leq \mathcal{O}(\log n)$ for instance) is the end-point of the binary search over $k \in [1, b]$, without compromising its $\mathcal{O}(\log k)$-competitiveness.

[2] And keeping track of the best k found thus far, including the end points of the binary search range.

[3] For an appropriate choice of constants within the asymptotic notation.

4.6 Empirical Evaluations

In this section, we present experimental results that demonstrate and compare the effectiveness of our algorithm on a variety of synthetic and real-world data sets in offline and streaming data settings [Lic13]. Our empirical evaluations demonstrate the practicality and wide-spread effectiveness of our approach: our algorithm consistently generated more compact and representative data summaries, and yet incurred a negligible increase in computational complexity when compared to uniform sampling.

Measurements \ Dataset	HTRU	Credit	Pathol.	Skin	Cod	W1
Number of data-points (n)	$17,898$	$30,000$	$1,000$	$245,057$	$488,565$	$49,749$
Sum of Sensitivities (t)	475.8	$1,013.0$	77.6	271.5	$2,889.2$	$24,231.6$
t/n (Percentage)	2.7%	3.4%	7.7%	0.1%	0.6%	51.3%

Table 4.1: The number of input points and measurements of the total sensitivity computed empirically for each data set in the offline setting. The sum of sensitivities is significantly less than n for virtually all of the data sets, which, by Thm. 19, ensures the sample-efficiency of our approach on the evaluated scenarios.

Evaluation We considered 6 real-world data sets of varying size and complexity as depicted in Table 4.1. For each data set of size n, we selected a set of $M = 15$ geometrically-spaced subsample sizes $m_1, \ldots, m_{\mathrm{M}} \subset [\log n, n^{4/5}]$. For each sample size m, we ran each algorithm (Alg. 4 or uniform sampling) to construct a subset $\mathcal{S} = (S, v)$ of size m. We then trained the SVM model as per usual on this subset to obtain an optimal solution with respect to the coreset $\mathcal{S}$, i.e., $w_{\mathcal{S}}^* = \mathrm{argmin}_w \, F_\lambda(\mathcal{S}, w)$. We then computed the relative error incurred by the solution computed on the coreset ($w_{\mathcal{S}}^*$) with respect to the ground-truth optimal solution computed on the entire data set (w^*): ${|F_\lambda(P,w_{\mathcal{S}}^*) - F_\lambda(P,w^*)|}/{F_\lambda(P,w^*)}$. The results were averaged across 100 trials.

Figures 4-2 and 4-3 depict the results of our comparisons against uniform sampling in the offline setting. In Fig. 4-2, we see that the coresets generated by our algorithm are much more representative and compact than the ones constructed by uniform sampling: across all data sets and sample sizes, training on our coreset yields significantly better solutions to SVM problem when compared to those generated by training on a uniform

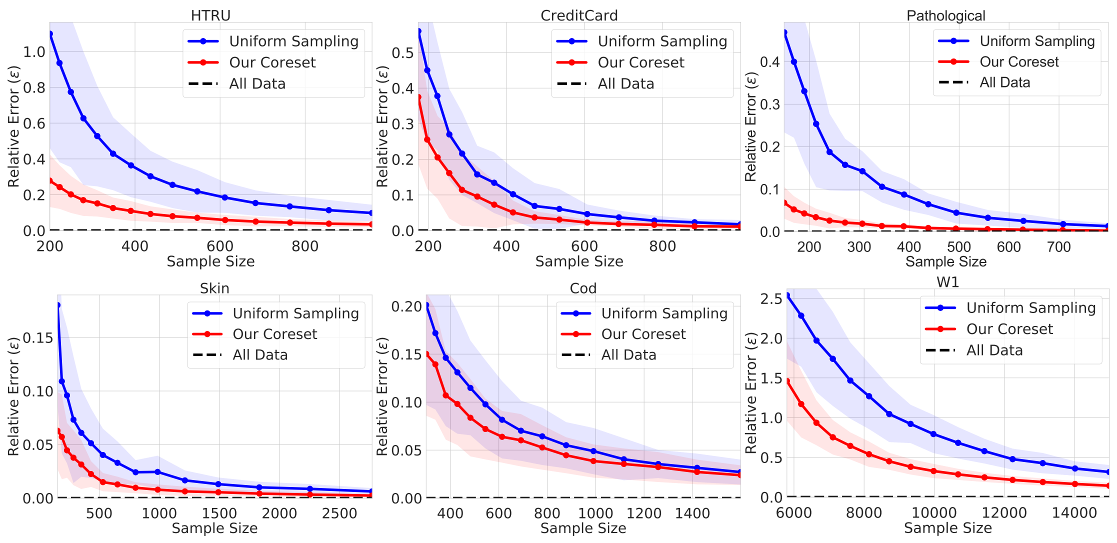


Figure 4-2: The relative error of query evaluations with respect uniform and coreset subsamples for the 6 data sets in the offline setting. Shaded region corresponds to values within one standard deviation of the mean.

sample. For certain data sets, such as HTRU, Pathological, and W1, this relative improvement over uniform sampling is at least an order of magnitude better, especially for small sample sizes. Fig. 4-2 also shows that, as a consequence of a more informed sampling scheme, the variance of each model's performance trained on our coreset is much lower than that of uniform sampling for all data sets.

Fig. 4-3 shows the total computational time required for constructing the sub-sample (i.e., coreset) $\mathcal{S}$ and training the SVM on the subset $\mathcal{S}$ to obtain $w^*_{\mathcal{S}}$. We observe that our approach takes significantly less time than training on the original model when considering non-trivial data sets (i.e., $n \geq 18,000$), and underscores the efficiency of our method: we incur a negligible cost in the overall SVM training time due to a more involved coreset construction procedure, but benefit heavily in terms of the accuracy of the models generated (Fig. 4-2).

Next, we evaluate our approach in the streaming setting, where data points arrive one-by-one and the entire data set cannot be kept in memory, for the same 6 data sets. The results of the streaming setting are shown in Fig. 4-4. Figs. 4-4 portray a similar trend as the one we observed in our offline evaluations: our approach significantly

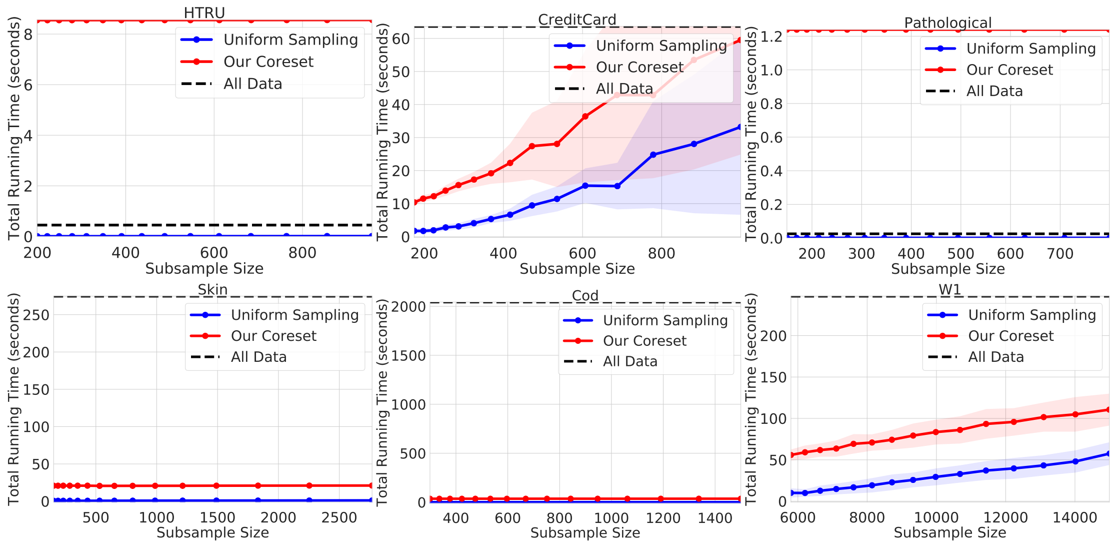


Figure 4-3: The *total* computational cost of constructing a coreset and training the SVM model on the coreset, plotted as a function of the size of the coreset.

outperforms uniform sampling for all of the evaluated data sets and sample sizes, with negligible computational overhead.

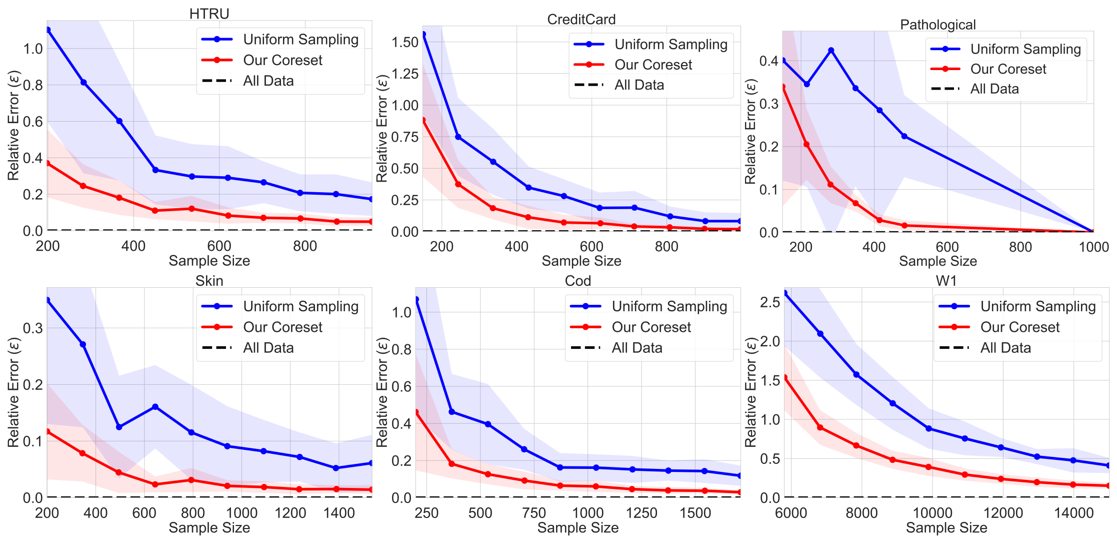


Figure 4-4: The relative error of query evaluations with respect uniform and coreset subsamples for the 6 data sets in the streaming setting. The figure shows that our method tends to fare even better in the streaming setting (cf. Fig. 4-2).

In sum, our empirical evaluations demonstrate the practical efficiency of our algorithm and reaffirm the favorable theoretical guarantees of our approach: the additional computational complexity of constructing the coreset is negligible relative to that of uniform sampling, and the entire preprocess-then-train pipeline is significantly more efficient than training on the original massive data set.

4.7 Extension to Streaming Settings

As a corollary to our main method, Alg. 5 extends the capabilities of any SVM solver, exact or approximate, to the streaming setting, where data points arrive one-by-one. Alg. 5 is inspired by [FL11, LS10] and constructs a binary tree, termed the *merge-and-reduce tree*, starting from the leaves which represent chunks of the data stream points. For each stream of l points, we construct an ε-coreset using Algorithm 4, and then we add each resulted tuple to B_1, a bucket which is responsible for storing each of the parent nodes of the leaves (Lines 1-6).

Note that for every $i > 1$, the bucket B_i will hold every node which is a root of a subtree of height i. Then, for every two successive items in each bucket B_i, for every $i \geq 1$, a parent node is generated by computing a coreset on the union of the coresets, which is then, added to the bucket B_{i+1} (Lines 7-11). This process is done till all the buckets are emptied other than B_h, that will contain only one tuple (S, v) which is set to be the root of the merge-and-reduce tree.

In sum, we obtain a binary tree of height $h = \Theta(\log(n))$ for a stream of n data points. Thus, at Lines 3 and 9, we have used error parameter $\varepsilon' = \varepsilon/(2\log(n))$ and failure parameter $\delta' = \delta/(2\log(n))$ in order to obtain ε-coreset with probability at least $1 - \delta$.

Coreset construction of size poly-logarithmic in n In case of the total sensitivity being sub-linear in n where n denotes the number of points in P, which is obtained by Lemma 17, we provide the following theorem which constructs a $(1+\varepsilon)$-coreset of size poly-logarithmic in n.

Lemma 20. *Let* $\varepsilon \in \left[\frac{1}{\log n}, \frac{1}{2}\right]$, $\delta \in \left[\frac{1}{\log n}, 1\right)$, $\lambda \in (0, 1]$, *a weighted set* (P, u), $\xi \in [0, F_\lambda(\mathcal{P}, w^*)]$ *where* $w^* \in \operatorname{argmin}_{w \in \mathbb{R}^{d+1}} F_\lambda(\mathcal{P}, w)$. *Let* t *denote the total sensitivity from Lemma 17 and suppose that there exists* $\beta \in (0.1, 0.8)$ *such that* $t \in \Theta(n^\beta)$. *Let*

Algorithm 5 STREAMING-CORESET$(P, u, \ell, \lambda, \xi, k)$

```
Inputs: $(P, u, \ell, \lambda, \xi, k)$ – An input stream $P$ in $\mathbb{R}^{d+1} \times \{-1, 1\}$ of $n$ points, a leaf
    size $\ell > 0$, a weight function $u : P \to \mathbb{R}_{\geq 0}$, a regularization parameter $\lambda \in [0, 1]$,
    a positive integer $k$, and an approximation factor $\xi > 0$.
Output: A weighted set $(\mathcal{S}, v)$.
1: $B_i \leftarrow \emptyset$ for every $1 \leq i \leq \infty$; $h \leftarrow 1$;
2: for each set $Q$ of consecutive $2\ell$ points from $P$ do
3:     $(T, v) \leftarrow$ CORESET$(Q, u, \lambda, \xi, k, \ell)$;    $j \leftarrow 1$; $B_j \leftarrow B_j \cup (T, v)$;
4:     for each $j \leq h$ do
5:         while $|B_j| \geq 2$ do
6:             $(T_1, u_1), (T_2, u_2) \leftarrow$ top two items in $B_j$;
7:             Set $\tilde{u} : T_1 \cup T_2 \to [0, \infty)$ such that for every $p \in T_1 \cup T_2$,
8:             $\tilde{u}(p) = \begin{cases} u_1(p) & p \in T_1, \\ u_2(p) & \text{otherwise} \end{cases}$;
9:             $(T, v) \leftarrow$ CORESET$(T_1 \cup T_2, \tilde{u}, \lambda, \xi, k, \ell)$ ;
10:            $B_{j+1} \leftarrow B_{j+1} \cup (T, v)$; $h \leftarrow \max\{h, j + 1\}$;
11: Set $(S, v)$ to be the only item in $B_h$ return $(S, v)$
```

$\ell \geq \max\left\{2^{\frac{\beta}{1-\beta}}, c\left(\frac{t \log^2(n)}{\varepsilon^2}\left(d \log t + \log(\log n/\delta)\right)\right)\right\}$ and let (S, v) be the output of a call to STREAMING-CORESET$(P, u, \ell, \lambda, \xi,)$. *Then (S, v) is an ε-coreset of size*

$$|S| \in (\log n)^{\mathcal{O}(1)}.$$

Proof. First we note that using Theorem 19 on each node in the merge-and-reduce tree, would attain that the root of the tree, i.e., (S, v) attains that for every w

$$(1 - \varepsilon)^{\log n} F_\lambda(\mathcal{P}, w) \leq F_\lambda\left((S, v), w\right) \leq (1 + \varepsilon)^{\log n} F_\lambda(\mathcal{P}, w),$$

with probability at least $(1 - \delta)^{\log n}$.

We observe by the properties of the natural number e,

$$(1 + \varepsilon)^{\log n} = \left(1 + \frac{\varepsilon \log n}{\log n}\right)^{\log n} \leq e^{\varepsilon \log n},$$

which when replacing ε with $\varepsilon' = \frac{\varepsilon}{2 \log n}$ in the above inequality as done at Lines 3 and 9 of Algorithm 5, we obtain that

$$(1+\varepsilon')^{\log n} \leq e^{\frac{\varepsilon}{2}} \leq 1+\varepsilon, \tag{4.5}$$

where the inequality holds since $\varepsilon \in [\frac{1}{\log n}, \frac{1}{2}]$.

As for the lower bound, observe that

$$(1-\varepsilon)^{\log n} \geq 1 - \varepsilon \log n,$$

where the inequality holds since $\varepsilon \in [\frac{1}{\log n}, \frac{1}{2}]$.

Hence,

$$(1-\varepsilon')^{\log n} \geq 1 - \varepsilon' \log n = 1 - \frac{\varepsilon}{2} \geq 1 - \varepsilon.$$

Similar arguments holds also for the failure probability δ. What is left for us to do is setting the leaf size which will attain us an ε-coreset of size poly-logarithmic in n (the number of points in P).

Let $\ell \in (0, \infty)$ be the size of a leaf in the merge-and-reduce tree. We observe that a coreset of size poly-logarithmic in n, can be achieved by solving the inequality

$$\frac{2\ell}{2} \geq (2\ell)^{\beta},$$

which is invoked when ascending from any two leafs and their parent node at the merge-and-reduce tree.

Rearranging the inequality, we yield that

$$\ell^{1-\beta} \geq 2^{\beta}.$$

Since $\ell \in (0, \infty)$, any $\ell \geq \sqrt[1-\beta]{2^{\beta}}$ would be sufficient for the inequality to hold. What is left for us to do, is to show that when ascending through the merge-and-reduce tree from the leaves towards the root, each parent node can't be more than half of the merge of it's children (recall that the merge-and-reduce tree is built in a binary tree fashion, as depicted at Algorithm 5).

Thus, we need to show that,

$$2^{\sum\limits_{j=1}^{i} \beta^j} \cdot \ell^{\beta^i} \leq \frac{2^{\sum\limits_{k=0}^{i-1} \beta^k} \cdot \ell^{\beta^{i-1}}}{2} = 2^{\sum\limits_{k=1}^{i-1} \beta^k} \cdot \ell^{\beta^{i-1}},$$

holds, for any $i \in [\lceil \log n \rceil]$ where $\log n$ is the height of the tree. Note that the left most term is the parent node's size and the right most term represents half the size of both parent's children nodes.

In addition, for $i = 1$, the inequality above represents each node which is a parent of leaves. Thus, we observe that for every $i \geq 1$, the inequality represents ascending from node which is a root of a sub-tree of height $i-1$ to it's parent in the merge-and-reduce tree.

By simplifying the inequality, we obtain the same inequality which only addressed the leaves. Hence, by using any $\ell \geq 2^{\frac{\beta}{1-\beta}}$ as a leaf size in the merge and reduce tree, we obtain an ε-coreset of size poly-logarithmic in n. $\square$

4.8 Proofs of the Analytical Results in Section 4.4

This section includes the full proofs of the technical results given in Sec. 4.4.

4.8.1 Proof of Lemma 16

Lemma 16. *For an even integer $d \geq 2$, there exists a set of weighted points $\mathcal{P} = (P, u)$ such that*

$$s(p) \geq \frac{n\lambda + d^2}{n\left(\lambda + d^2\right)} \qquad \forall p \in P \qquad \textit{and} \qquad \sum_{p \in P} s(p) \geq \frac{n\lambda + d^2}{\left(\lambda + d^2\right)}.$$

Proof. Following [YCRM17], let $n = \binom{d}{d/2}$ and let $\mathcal{P} = (P, u)$, where $P \subseteq \mathbb{R}^{d+1} \times \{\pm 1\}$ be set of n labeled points, and $u : P \to 1$. For every $p = (x, y) \in P$, where $x \in \mathbb{R}^d \times \{1\}$ and $y \in \{\pm 1\}$, among the first d entries of x, exactly $\frac{d}{2}$ entries are equivalent to

$$y\sqrt{\frac{2}{d}},$$

where the remaining $\frac{d}{2}$ entries among the first d are set to 0. Hence, for our proof to hold, we assume that P contains all such combinations and at least one point of each label. For every $p = (x, y) \in P$, define the set of non-zero entries of p as the set

$$B_p = \{i \in [d+1] : x_i \neq 0\}.$$

Put $p \in P$ and note that for bounding the sensitivity of point p, consider w with entries defined as

$$\forall i \in [d+1] \;\; w_i = \begin{cases} 0 \;\text{ if } \; i \in B_p, \\ \frac{1}{\sqrt{\frac{2}{d}}} \;\; \text{otherwise.} \end{cases}$$

Note that $\|w\|_2^2 = \frac{d}{2}\left(\frac{1}{\sqrt{\frac{2}{d}}}\right)^2 = \frac{d^2}{4}$. We also have that $h(p, w) = 1$ since $y\langle x, w\rangle = \sum_{i \in B_p} y x_i w_i = \frac{d}{2} 0 = 0$. To bound the sum of hinge losses contributed by other points $q \in P \setminus \{p\}$, note that $B_q \setminus B_p \neq \emptyset$. Then for every $q = (x', y') \neq p$,

$$y'\langle x', w\rangle = \sum_{i \in B_q \setminus B_p} y' x'_i w_i \geq \frac{1}{\sqrt{\frac{2}{d}}}\sqrt{\frac{2}{d}} = 1,$$

which implies that $h(q, w) = 0$. Thus,

$$\sum_{q \in P} h(q, w) = 1.$$

Putting it all together,

$$s(p) = \sup_{\substack{w' \in \mathbb{R}^{d+1} \\ F_\lambda(\mathcal{P}, w') \neq 0}} \frac{f_\theta[w']}{F_\lambda(\mathcal{P}, w')} \geq \frac{\frac{d^2}{8n} + \lambda\, h(p, w)}{\frac{\|w\|_2^2}{2} + \lambda} = \frac{\frac{d^2}{8n} + \lambda}{\frac{d^2}{8} + \lambda}.$$

Since the above holds for every $p \in P$, summing the above inequality over every $p \in P$, yields that

$$\sum_{p \in P} s(p) \geq \frac{\frac{d^2}{8} + n\lambda}{\frac{d^2}{8} + \lambda} \in \Omega\left(\frac{d^2 + n\lambda}{d^2 + \lambda}\right).$$

□

4.8.2 Proof of Lemma 17

In what follows, for simplicity of presentation, we present some of the definitions of the terms that are used throughout the proof of Lemma 17.

Term	Definition
$u(p)$	The weight of $p \in P$
$\mathcal{P}$	The tuple (P, u) where u is weight function
P_y	All points in the set P that are associated with the label $y \in \{+, -\}$
$c_y^{(i)}$	$\frac{1}{U\left(P_y^{(i)}\right)} \sum_{q=(x_q, y_q) \in P_y^{(i)}} u(q) x_q$ (i.e., the centroid of cluster i)
$U(P_y)$	The sum of weights of the points in P_y
$\alpha_y^{(i)}$	$\frac{U\left(P \setminus P_y^{(i)}\right)}{2\lambda U(P) U\left(P_y^{(i)}\right)}$
p_Δ	$y(c_y^{(i)} - x)$ for every $p = (x, y) \in P_y$, $i \in [k]$, and $y \in \{+, -\}$

Table 4.2: Definition of terms that are stated in Algorithm 4

Lemma 17. *Let k be a positive integer, $\xi \in [0, F_\lambda(\mathcal{P}, w^*)]$, and let $\mathcal{P} = (P, u)$ be a weighted set. Then for every $i \in [k]$, $y \in \{+, -\}$ and $p \in P_y^{(i)}$,*

$$s(p) \leq \frac{u(p)}{U\left(P_y^{(i)}\right)} + \frac{9\lambda u(p)}{2} \max\left\{\frac{4\alpha_y^{(i)}}{9}, \sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\left\|p_\Delta\right\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)}\right\} = \gamma(p).$$

Proof. Let $P_y \subseteq P$ denote the set of points with the same label as p as in Line 4 of Algorithm 4. Consider a clustering of the points in P_y into k clusters with centroids $\mathcal{C}^y = \{c_y^{(1)}, \ldots, c_y^{(k)}\} \subseteq \mathbb{R}^{d+1}$ being their mean as in Line 5, and let $\alpha_y^{(i)}$ be as defined in Line 7 for every $i \in [k]$ and $y \in \{+,-\}$. In addition, let $\mathcal{P} \setminus \mathcal{P}_y^{(i)}$ denote the weighted set $\left(P \setminus P_y^{(i)}, u\right)$ for every $i \in [k]$ and $y \in \{+,-\}$.

We begin with the following property about the ramp function $x \mapsto \max\{0, x\}$:

$$\max\{0, a\} \leq \max\{0, b\} + \max\{0, a - b\}. \tag{4.6}$$

To see this, note that the inequality is trivial if $a \leq b$. On the other hand, for $a > b$, we can apply the fact that $\max\{0, x\}$ is 1-Lipschitz, which gives

$$|\max\{0, a\} - \max\{0, b\}| \leq |a - b| = \max\{0, a - b\},$$

where the last equality follows by the fact that $a > b$.

Next, put $p = (x, y) \in P$ and let $i \in [k]$ be the index of the cluster which p belongs to, i.e., $p \in P_y^{(i)}$. Recall that the hinge loss of p is defined as

$$h(p, w) = \max\{0, \underbrace{1 - y\langle x, w\rangle}_{=a}\}.$$

For another point $q = (z, y)$ of the same label y as p, the hinge loss is similarly

$$h(q, w) = \max\{0, \underbrace{1 - y\langle z, w\rangle}_{=b}\}.$$

Applying (4.6) with the underlined $a = 1 - y\langle x, w\rangle$ and $b = 1 - y\langle z, w\rangle$ implies that

$$\begin{aligned} h(p, w) &= \max\{0, a\} && (4.7)\\ &\leq \max\{0, b\} + \max\{0, a - b\} && (4.8)\\ &= h(q, w) + \max\{0, y\langle z - x, w\rangle\}. && (4.9) \end{aligned}$$

Now let $c_y^{(i)}$ denote the centroid (mean) of the *inputs* in cluster $P_y^{(i)}$, i.e.,

$$c_y^{(i)} = \frac{1}{U\left(P_y^{(i)}\right)} \sum_{q=(x_q,y_q)\in P_y^{(i)}} u(q)x_q,$$

where $U\left(P_y^{(i)}\right)$ is the total weight of the points in $P_y^{(i)}$

$$U\big(P_y^{(i)}\big) = \sum_{q\in P_y^{(i)}} u(q).$$

Note that $c_y^{(i)}$ itself is not a point since it is a mean over inputs and consequently does not have the label information embedded in it. By embedding the label information, however, we do obtain a point (i.e., $(c_y^{(i)}, y)$) that we can use as input to functions we have previously defined.

To this end, using the upper bound of Eq. (4.9) with $q = (c_y^{(i)}, y)$ yields

$$\begin{aligned} h(p,w) &\le h\left((c_y^{(i)}, y), w\right) + \max\{0, y\langle c_y^{(i)} - x, w\rangle\} \\ &= h\left((c_y^{(i)}, y), w\right) + \max\{0, \langle p_\Delta, w\rangle\}, \end{aligned}$$

where

$$p_\Delta = y\big(c_y^{(i)} - x\big).$$

Observing that the hinge loss is convex in the point, we invoke Jensen's inequality to obtain for the point $(c_y^{(i)}, y)$

$$f_\lambda((c_y^{(i)}, y), w) \le \frac{1}{U\left(P_y^{(i)}\right)} \sum_{q\in P_y^{(i)}} u(q)f(q,w) = \frac{F_\lambda(\mathcal{P}, w) - F_\lambda(\mathcal{P}\setminus \mathcal{P}_y^{(i)}, w)}{U\left(P_y^{(i)}\right)}.$$

Applying the two inequalities established above to $s(p)/u(p)$ yields that

$$\frac{s(p)}{u(p)} = \sup_w \frac{f_\lambda(p,w)}{F_\lambda(\mathcal{P},w)} \tag{4.10}$$

$$\leq \sup_w \frac{f_\lambda((c_y^{(i)}, y), w) + \lambda \left[\langle w, p_\Delta \rangle\right]_+}{F_\lambda(\mathcal{P}, w)} \tag{4.11}$$

$$\leq \sup_w \frac{\sum_{q \in P_y^{(i)}} u(q) f_\lambda(q, w)}{U\left(P_y^{(i)}\right) F_\lambda(\mathcal{P}, w)} + \frac{\lambda \left[\langle w, p_\Delta \rangle\right]_+}{F_\lambda(\mathcal{P}, w)} \tag{4.12}$$

$$= \sup_w \frac{F_\lambda(\mathcal{P}, w) - F_\lambda(\mathcal{P} \setminus \mathcal{P}_y^{(i)}, w)}{U\left(P_y^{(i)}\right) F_\lambda(\mathcal{P}, w)} + \frac{\lambda \left[\langle w, p_\Delta \rangle\right]_+}{F_\lambda(\mathcal{P}, w)} \tag{4.13}$$

$$= \frac{1}{U\left(P_y^{(i)}\right)} + \sup_w \frac{\lambda \left[\langle w, p_\Delta \rangle\right]_+ - F_\lambda(P \setminus P_y^{(i)}, w) / U\left(P_y^{(i)}\right)}{F_\lambda(\mathcal{P}, w)} \tag{4.14}$$

By definition of $F_\lambda(P \setminus P_y^{(i)}, w)$, we have

$$F_\lambda(P \setminus P_y^{(i)}, w) \geq \frac{\|w_{1:d}\|_2^2 \, U\left(P \setminus P_y^{(i)}\right)}{2U(P)}.$$

Continuing from above and dividing both sides by λ yields

$$\begin{aligned} \frac{s(p)}{\lambda u(p)} &\leq \frac{1}{\lambda U\left(P_y^{(i)}\right)} + \sup_w \frac{\left[\langle w, p_\Delta \rangle\right]_+ - \frac{\|w_{1:d}\|^2 \, U\left(P \setminus P_y^{(i)}\right)}{2\lambda U(P) U\left(P_y^{(i)}\right)}}{F_\lambda(\mathcal{P}, w)} \\ &\leq \frac{1}{\lambda u(\mathcal{C}_p)} + \sup_w \frac{\left[\langle w, p_\Delta \rangle\right]_+ - \alpha_y^{(i)} \|w_{1:d}\|_2^2}{F_\lambda(\mathcal{P}, w)}, \end{aligned}$$

where

$$\alpha_y^{(i)} = \frac{U\left(P \setminus P_y^{(i)}\right)}{2\lambda U(P) U\left(P_y^{(i)}\right)}. \tag{4.15}$$

Let

$$g(w) = \frac{\left[\langle w, p_\Delta \rangle\right]_+ - \alpha_y^{(i)} \|w_{1:d}\|_2^2}{F_\lambda(\mathcal{P}, w)}$$

be the expression on the right hand side of the sensitivity inequality above, and let $\hat{w} \in \operatorname{argmax}_w g(w)$. The rest of the proof will focus on bounding $g(\hat{w})$, since an upper bound on the sensitivity of a point as a whole would follow directly from an upper bound on $g(\hat{w})$.

Note that by definition of p_Δ and the embedding of 1 to the $(d+1)^{\text{th}}$ entry of the original d-dimensional point (with respect to p),

$$\langle \hat{w}, p_\Delta \rangle = \langle \hat{w}_{1:d}, (p_\Delta)_{1:d} \rangle,$$

where the equality holds since the $(d+1)$th entry of p_Δ is zero.

We know that $\langle \hat{w}, p_\Delta \rangle \geq \alpha_y^{(i)} \left\| \hat{w}_{1:d} \right\|_2^2 \geq 0$, since otherwise $g(\hat{w}) < 0$, which contradicts the fact that $\hat{w}$ is the maximizer of $g(w)$. This implies that for each entry $j \in [d]$ of the sub-gradient of $g(\cdot)$ evaluated at $\hat{w}$, denoted by $\nabla g(\hat{w})$, is given by

$$\nabla g(\hat{w})_j = \frac{\left((p_\Delta)_j - 2\alpha_y^{(i)} \hat{w}_j \right) F_\lambda(\mathcal{P}, \hat{w}) - \nabla F_\lambda(\mathcal{P}, \hat{w})_j \left(\langle w, p_\Delta \rangle - \alpha_y^{(i)} \left\| w_{1:d} \right\|_2^2 \right)}{F_\lambda(\mathcal{P}, \hat{w})^2}. \tag{4.16}$$

Note that since $\hat{w}$ is the maximizer of $g\left(\cdot\right)$, the gradient of g at $\hat{w}$ is zero. Hence, we obtain that for $\nabla g(\hat{w})_{d+1} = 0$, i.e., either the numerator of g is zero or the $\nabla F_\lambda(\mathcal{P}, \hat{w})_{d+1} = 0$. If the numerator is zero for $\hat{x}$ then the sensitivity of p is $\frac{1}{\lambda u(\mathcal{C}_p)}$. However, since we aim to bound the term from above, we assume that without loss of generality the numerator of g is not zero. This means that $\nabla F_\lambda(\mathcal{P}, \hat{w})_{d+1} = 0$.

Letting $\gamma = \langle \hat{w}, p_\Delta \rangle - \alpha_y^{(i)} \left\| \hat{w}_{1:d} \right\|_2^2$ and setting each entry of the gradient $\nabla g(\hat{w})$ to 0, we solve for p_Δ to obtain

$$(p_\Delta)_{1:d} = \frac{\gamma \, \nabla F_\lambda(\mathcal{P}, \hat{w})_{1:d}}{F_\lambda(\mathcal{P}, \hat{w})} + 2\alpha_y^{(i)} \hat{w}_{1:d}.$$

This implies that

$$\langle \hat{w}, p_\Delta \rangle = \frac{\gamma \left\langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \right\rangle}{F_\lambda(\mathcal{P}, \hat{w})} + 2\alpha_y^{(i)} \left\| \hat{w}_{1:d} \right\|_2^2$$

Rearranging and using the definition of γ, we obtain

$$\gamma = \frac{\gamma \langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})} + \alpha_y^{(i)} \left\| \hat{w}_{1:d} \right\|_2^2 , \tag{4.17}$$

where Lemma 17 holds by taking $\frac{9}{2}$ outside the max term.

By using the same equivalency for p_Δ from above, we also obtain that

$$\begin{aligned} \left\| p_\Delta \right\|_2^2 = \langle p_\Delta, p_\Delta \rangle &= \left\| \frac{\gamma \nabla F_\lambda(\mathcal{P}, \hat{w})_{1:d}}{F_\lambda(\mathcal{P}, \hat{w})} + 2\alpha_y^{(i)} \hat{w} \right\|^2 \\ &= \frac{\gamma^2}{F_\lambda(\mathcal{P}, \hat{w})^2} \left\| \nabla F_\lambda(\mathcal{P}, \hat{w}) \right\|_2^2 + 4 \left(\alpha_y^{(i)} \right)^2 \left\| \hat{w}_{1:d} \right\|_2^2 + 4\alpha_y^{(i)} \frac{\gamma \langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})}, \end{aligned}$$

but $\frac{\gamma \langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})} = \gamma - \alpha_y^{(i)} \left\| \hat{w}_{1:d} \right\|_2^2$, and so continuing from above, we have

$$\begin{aligned} \left\| p_\Delta \right\|_2^2 &= \frac{\gamma^2}{F_\lambda(\mathcal{P}, \hat{w})^2} \left\| \nabla F_\lambda(\mathcal{P}, \hat{w}) \right\|_2^2 + 4 \left(\alpha_y^{(i)} \right)^2 \left\| \hat{w}_{1:d} \right\|_2^2 + 4\alpha_y^{(i)} (\gamma - \alpha_y^{(i)} \left\| \hat{w}_{1:d} \right\|_2^2) \\ &= \frac{\gamma^2}{F_\lambda(\mathcal{P}, \hat{w})^2} \left\| \nabla F_\lambda(\mathcal{P}, \hat{w})_{1:d} \right\|_2^2 + 4\alpha_y^{(i)} \gamma \\ &= \gamma^2 \tilde{x} + 4\alpha_y^{(i)} \gamma, \end{aligned}$$

where $\tilde{x} = \frac{\left\| \nabla F_\lambda(\mathcal{P}, \hat{w}) \right\|_2^2}{F_\lambda(\mathcal{P}, \hat{w})^2}$. Solving for γ from the above equation yields for $\tilde{x} > 0$

$$\gamma = \frac{\sqrt{4 \left(\alpha_y^{(i)} \right)^2 + \left\| p_\Delta \right\|^2 \tilde{x}} - 2\alpha_y^{(i)}}{\tilde{x}}. \tag{4.18}$$

Now we subdivide the rest of the proof into two cases. The first is the trivial case in which the sensitivity of the point is sufficiently small enough to be negligible, and the second case is the involved case in which the point has a high influence on the SVM cost function and its contribution cannot be captured by the optimal solution w^* or something close to it.

Case $g(\hat{w}) \leq 3\alpha_y^{(i)}$ the bound on the sensitivity follows trivially from the analysis above.

Case $g(\hat{w}) > 3\alpha_y^{(i)}$ note that the assumption of this case implies that w^* cannot be the maximizer of $g(\cdot)$, i.e., $\hat{w} \neq w^*$. This follows by the convexity of the SVM loss function which implies that the norm of the gradient evaluated at w^* is 0. Thus by (4.17):

$$\gamma = \alpha_y^{(i)} \left\| {w^*}_{1:d} \right\|_2^2 .$$

Since $F_\lambda(\mathcal{P}, w^*) \geq \left\| {w^*}_{1:d} \right\|_2^2 / 2$, we obtain

$$s(p) \leq \frac{\alpha_y^{(i)} \left\| {w^*}_{1:d} \right\|_2^2}{F_\lambda(\mathcal{P}, w^*)} \leq 2\alpha_y^{(i)}.$$

Hence, we know that for this case we have $\left\| \nabla F_\lambda(\mathcal{P}, \hat{w}) \right\|_2 > 0$, $F_\lambda(\mathcal{P}, \hat{w}) > F_\lambda(\mathcal{P}, w^*) \geq 0$, and so we obtain $\tilde{x} > 0$.

This implies that we can use Eq.(4.18) to upper bound the numerator γ of the sensitivity. Note that γ from (4.18) is decreasing as a function of $\tilde{x}$, and so it suffices to obtain a lower bound on $\tilde{x}$. To do so, lets focus on Eq.(4.17) and let divide both sides of it by γ, to obtain that

$$1 = \frac{\langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})} + \frac{\alpha_y^{(i)}}{\gamma} \left\| w_{1:d} \right\|_2^2 .$$

By rearranging the above equality, we have that

$$\frac{\langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle}{F_\lambda(\mathcal{P}, \hat{w})} = 1 - \frac{\alpha_y^{(i)} \left\| w_{1:d} \right\|_2^2}{\gamma}. \tag{4.19}$$

Recall that since the last entry of p_Δ is 0 then it follows from Eq.(4.16) that $\nabla F_\lambda(\mathcal{P}, \hat{w})_{d+1}$ is also zero, which implies that

$$\begin{aligned} \langle \hat{w}, \nabla F_\lambda(\mathcal{P}, \hat{w}) \rangle &= \langle \hat{w}_{1:d}, \nabla F_\lambda(\mathcal{P}, \hat{w})_{1:d} \rangle \\ &\leq \left\| \hat{w}_{1:d} \right\|_2 \left\| \nabla F_\lambda(\mathcal{P}, \hat{w})_{1:d} \right\|_2 \\ &= \left\| \hat{w}_{1:d} \right\|_2 \left\| \nabla F_\lambda(\mathcal{P}, \hat{w}) \right\|_2 \end{aligned} \tag{4.20}$$

where the inequality is by Cauchy-Schwarz.

Combining Eq.(4.19) with Eq. (4.20) yields

$$\begin{aligned}\frac{\|\hat{w}_{1:d}\|_2 \|\nabla F_\lambda(\mathcal{P},\hat{w})\|_2}{F_\lambda(\mathcal{P},\hat{w})} &\geq 1 - \frac{\alpha_y^{(i)} \|w_{1:d}\|_2^2}{\gamma} \\ &\geq 1 - \frac{\alpha_y^{(i)} \|\hat{w}_{1:d}\|_2^2}{3\alpha_y^{(i)} F_\lambda(\mathcal{P},\hat{w})} \\ &\geq 1 - \frac{\alpha_y^{(i)} 2F_\lambda(\mathcal{P},\hat{w})}{3\alpha_y^{(i)} F_\lambda(\mathcal{P},\hat{w})} \\ &= \frac{1}{3},\end{aligned}$$

where the second inequality holds by the assumption of the case, the third inequality follows from the fact that $\|\hat{w}_{1:d}\|_2^2 \leq 2F_\lambda(\mathcal{P},\hat{w})$.

This implies that

$$\frac{\|\nabla F_\lambda(\mathcal{P},\hat{w})\|_2}{F_\lambda(\mathcal{P},\hat{w})} \geq \frac{1}{3\|w_{1:d}\|_2} \geq \frac{\sqrt{2}}{3\sqrt{F_\lambda(\mathcal{P},\hat{w})}}.$$

Hence by definition of $\tilde{x}$, we have that

$$\tilde{x} \geq \frac{2}{9F_\lambda(\mathcal{P},\hat{w})} \tag{4.21}$$

Plugging Eq.(4.21) into Eq.(4.18), we obtain that

$$\frac{\gamma}{F_\lambda(\mathcal{P},\hat{w})} \leq \frac{9}{2}\left(\sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\|p_\Delta\|_2^2}{9F_\lambda(\mathcal{P},\hat{w})}} - 2\alpha_y^{(i)}\right).$$

Recall that

$$F_\lambda(\mathcal{P},\hat{w}) \geq F_\lambda(\mathcal{P},w^*) \geq F_\lambda(\mathcal{P},\tilde{w}) - \xi,$$

which implies that

$$\frac{\gamma}{F_\lambda(\mathcal{P}, \hat{w})} \leq \frac{9}{2}\left(\sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\left\|p_\Delta\right\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)}\right), \tag{4.22}$$

where $\widetilde{opt}_\xi = F_\lambda(\mathcal{P}, \tilde{w}) - \xi$.

Combining both cases, yields that

$$s(p) \leq \frac{u(p)}{U\left(P_y^{(i)}\right)} + u(p)\lambda \max\left\{2\alpha_y^{(i)}, \frac{9}{2}\left(\sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\left\|p_\Delta\right\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)}\right)\right\}, \tag{4.23}$$

where Lemma 17 holds by rearranging Eq. 4.23. □

4.8.3 Proof of Lemma 18

Lemma 18. *In the context of Lemma 17, the sum of sensitivities is bounded by*

$$\sum_{p \in P} s(p) \leq t = 4k + \sum_{i=1}^{k} \frac{3\lambda Var_+^{(i)}}{\sqrt{2\widetilde{opt}_\xi}} + \frac{3\lambda Var_-^{(i)}}{\sqrt{2\widetilde{opt}_\xi}},$$

where $Var_y^{(i)} = \sum_{p \in P_y^{(i)}} u(p)\left\|p_\Delta\right\|_2$ *for all* $i \in [k]$ *and* $y \in \{+,-\}$.

Proof. We first observe that that

$$\sum_{p \in P} s(p) = \sum_{i \in [k]}\left(\sum_{p \in P_+^{(i)}} s(p) + \sum_{p \in P_-^{(i)}} s(p)\right).$$

Thus we will focus on the summing the sensitivity of the all the points whose label is positive. We note that

$$\sum_{i \in [k]} \sum_{p \in P_+^{(i)}} \frac{u(p)}{U\left(P_+^{(i)}\right)} = \sum_{i=1}^{k} 1 = k. \tag{4.24}$$

In addition, we observe that $\max\{a,b\} \le a+b$ for every $a,b \ge 0$, which implies that for every $i \in [k]$ and $p \in P_{+}^{(i)}$,

$$\begin{aligned}
&\max\left\{2\alpha_y^{(i)}, \frac{9}{2}\left(\sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\left\|p_\Delta\right\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)}\right)\right\} \\
&\le 2\alpha_y^{(i)} + \frac{9}{2}\left(\sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\left\|p_\Delta\right\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)}\right).
\end{aligned} \tag{4.25}$$

Since $\sqrt{a+b} \le \sqrt{a} + \sqrt{b}$ for every $a,b \ge 0$, we have for every $i \in [k]$ and $p \in P_{-}^{(i)}$,

$$\begin{aligned}
&\sqrt{4\left(\alpha_y^{(i)}\right)^2 + \frac{2\left\|p_\Delta\right\|_2^2}{9\widetilde{opt}_\xi}} - 2\alpha_y^{(i)} \\
&\le 2\alpha_y^{(i)} + \frac{\sqrt{2}\left\|p_\Delta\right\|_2}{3\sqrt{\widetilde{opt}_\xi}} - 2\alpha_y^{(i)} \\
&= \frac{\sqrt{2}\left\|p_\Delta\right\|_2}{3\sqrt{\widetilde{opt}_\xi}}.
\end{aligned} \tag{4.26}$$

Hence by combining Eq.(4.24), Eq.(4.25), and Eq.(4.26), we yield that

$$\begin{aligned}
\sum_{i\in[k]}\sum_{p\in P_{+}^{(i)}} s(p) &\le k + \sum_{i\in[k]}\sum_{p\in P_{+}^{(i)}} 2\lambda\alpha_{+}^{(i)} + \frac{9}{2}\lambda u(p)\frac{\sqrt{2}\left\|p_\Delta\right\|_2}{3\sqrt{\widetilde{opt}_\xi}} \\
&= k + \sum_{i\in[k]}\sum_{p\in P_{+}^{(i)}} 2\lambda\alpha_{+}^{(i)} + \sum_{i\in[k]} \lambda\frac{3\mathrm{Var}_{+}^{(i)}}{\sqrt{2\widetilde{opt}_\xi}} \\
&\le 2k + \sum_{i\in[k]} \lambda\frac{3\mathrm{Var}_{+}^{(i)}}{\sqrt{2\widetilde{opt}_\xi}},
\end{aligned}$$

where the inequality follows from definition of $\alpha_y^{(i)}$ for every $i \in [k]$ and $y \in \{+,-\}$ as defined in Eq.(4.15).

Since all of the previous arguments hold similarly for P_-, we obtain that

$$\sum_{p \in P} s(p) \leq 4k + \sum_{i \in [k]} \frac{3\mathrm{Var}_+^{(i)}}{\sqrt{2\widetilde{opt}_\xi}} + \frac{3\mathrm{Var}_-^{(i)}}{\sqrt{2\widetilde{opt}_\xi}}.$$

□

4.8.4 Proof of Theorem 19

Theorem 19. *For any $\varepsilon \in (0, 1/2), \delta \in (0,1)$, let m be an integer satisfying*

$$m \in \Theta\left(\frac{t}{\varepsilon^2}\left(d \log t + \log(1/\delta)\right)\right),$$

where t is as in Lem. 18. Invoking Coreset *with the inputs defined in this context yields a ε-coreset $\mathcal{S} = (S, v)$ with probability at least $1 - \delta$ in $\mathcal{O}(nd + T)$ time, where T represents the computational complexity of obtaining an ξ-approximated solution to SVM and applying k-means++ on P_+ and P_-.*

Proof. By Lemma 17 and Theorem 5.5 of [BFL16] we have that the coreset constructed by our algorithm is an ε-coreset with probability at least $1 - \delta$ if

$$m \geq c\left(\frac{t}{\varepsilon^2}\left(d \log t + \log(1/\delta)\right)\right),$$

where we used the fact that the VC dimension of a SVMs in the case of a linear kernel is bounded $\dim(\mathcal{F}) \leq d + 1 = \mathcal{O}(d)$ [VV98], and c is a sufficiently large constant which can be determined using similar techniques to that of [LLS01]. Moreover, note that the computation time of our algorithm is dominated by going over the whole weighted set $\mathcal{P}$ which takes $\mathcal{O}(n)$ and attaining an ξ-approximation to the SVM problem at Line 1 followed by applying k-means clustering as shown in Algorithm 4 which takes $\mathcal{O}(T)$ time. This implies that the overall time is $\mathcal{O}(nd + T)$. □

Sufficient Conditions Theorem 19 immediately implies that, for reasonable ε and δ, coresets of poly-logarithmic (in n) size can be obtained if $d = \mathcal{O}(\text{polylog}(n))$, which is usually the case in our target Big Data applications, and if

$$\sum_{i=1}^{k} \frac{3\lambda \text{Var}_{+}^{(i)}}{\sqrt{2\widetilde{opt}_{\xi}}} + \frac{3\lambda \text{Var}_{-}^{(i)}}{\sqrt{2\widetilde{opt}_{\xi}}} = \mathcal{O}(\text{polylog}(n)).$$

For example, a value of $\lambda \leq \frac{\log n}{n}$ for the regularization parameter λ satisfies the sufficient condition for all data sets with points normalized such that they are contained within the unit ball. Note that the total sensitivity, which dictates how many samples are necessary to obtain an ε-coreset with probability at least $1 - \delta$ and in a sense measures the difficulty of the problem, increases monotonically with the sum of distances of the points from their label-specific means.

4.9 Experimental Details

Our experiments were implemented in Python and performed on a 3.2GHz i7-6900K (8 cores total) machine with 64GB RAM. We considered the following datasets in our evaluations.

1. *HTRU* — $17,898$ radio emissions, each with 9 features, of the Pulsar star.

2. *CreditCard* — $30,000$ client entries each consisting of 24 features that include education, age, and gender among other factors.

3. *Pathological* — $1,000$ points in two dimensional space describing two clusters distant from each other of different labels, as well as two points of different labels which are close to each other.[4]

4. *Skin* — $245,057$ random samples of B,G,R from face images consisting of 4 dimensions.

[4]We note that uniform sampling performs particularly poorly against this data set due to the presence of outliers.

5. *Cod(-rna)* — 488565 RNA records consisting each of 8 features.[5]

6. *W1* — $49,749$ records of web pages consisting each of 300 features.

Preprocessing step Each data set has gone through a standardization process which aims to rescale the features so that they will have zero mean and unit standard deviation. As for the case where a data set is unweighted, we simply give each data point a weight of 1, i.e., $u : P \to 1$ where P denotes the data set, and the regularization parameter λ was set to be 1 throughout all of our experiments.

k-means clustering In our experiments, we set $k = \log n$ where n is the number of points in the dataset (each datasets has different k value). As for the clustering itself, we have applied k-means++ [AV07] on each of $\mathcal{P}_+$ and $\mathcal{P}_-$ as stated in our analysis; see Sec. 4.4.

Evaluation under streaming setting Under streaming setting, the range for sample sizes is the same as for running under offline settings (See Figures 4-2 and 4-3). What differs is the quality of the solver itself, which we use to show the effectiveness of our coreset compared to uniform sampling, i.e., we have chosen to make the solver (SVC of Sklearn) more accurate by lowering its optimal tolerance.

4.10 Discussion

We presented an efficient coreset construction algorithm for generating compact representations of the input data points that are provably competitive with the original data set in training Support Vector Machine models. Unlike prior approaches, our method and its theoretical guarantees naturally extend to streaming settings and scenarios involving dynamic data sets, where points are continuously inserted and deleted. We established instance-dependent bounds on the number of samples required to obtain accurate approximations to the SVM problem as a function of input

[5]This data set was attained by merging the training, validation and testing sets.

data complexity and established dataset dependent conditions for the existence of compact representations. Our experimental results on real-world data sets validate our theoretical results and demonstrate the practical efficacy of our approach in speeding up SVM training. We conjecture that our coreset construction can be extended to accelerate SVM training for other classes of kernels and can be applied to a variety of Big Data scenarios.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation (NSF) under Awards 1723943 and 1526815, Office of Naval Research (ONR) Grant N00014-18-1-2830, Microsoft, and JP Morgan Chase.

Chapter 5

Provable Weight Pruning of Neural Networks

5.1 Overview

In the previous chapter, we focused on subsampling the input data for faster SVM training. Here, we instead focus on using importance sampling to prune neural network models for efficiency and deployability, as well as to alleviate their exponentially increasing ramifications for the environment. It turns out that the sensitivity framework from Chapter 4 leads to uniform sampling due to the the high capacity of neural networks and the pessimistic nature of the sensitivity definition that does not consider the randomness over the queries. To overcome this limitation, we extend the sensitivity framework and introduce *empirical sensitivity*, a data-informed notion that is effective in practice and exhibits favorable theoretical properties. We then apply this framework to prune the weights of large neural networks and provide bounds on its performance. Our approach can be applicabled to various architectures including fully-connected (FNNs), convolutional (CNNs), and recurrent neural networks (RNNs), and the complete codebase for the presented methods is publicly available online [Lie21].

The work presented in this chapter is based on [BLG+21, BLG+18] and contributes the following:

1. A versatile family of pruning algorithms, SiPP, that combines novel sample size allocation and adaptive sparsification procedures to prune network parameters without any assumptions on the specific state of the network — i.e., regardless of whether the given network is untrained, trained, or partially trained.

2. An analysis of the resulting size and accuracy of the compressed network generated by SiPP that establishes novel compression bounds for a large class of neural networks.

3. Empirical evaluations on fully-connected and convolutional networks with comparisons to baseline pruning approaches that demonstrate the practical effectiveness of SiPP across a set of diverse, real-world pruning tasks with varying amounts of retraining.

5.2 Background

The set of parameters θ of a neural network with L layers is a tuple of multi-dimensional weight tensors corresponding to each layer, i.e., $\theta = (W^1, \ldots, W^L)$. The set of parameters θ defines the mapping $f_\theta : \mathcal{X} \to \mathcal{Y}$ from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. We consider the setting where we have access to independent and identically distributed (i.i.d.) samples (x, y) from a joint distribution defined on $\mathcal{X} \times \mathcal{Y}$ from which we can gather a training, test, and validation data set. To this end, we let $\mathcal{D}$ denote the marginal distribution over the input space $\mathcal{X}$.

5.2.1 Network Notation

Layers For a given input $x \sim \mathcal{D}$, we denote the pre-activation and activation of layer ℓ by $Z^\ell(x)$ and $A^\ell(x)$, respectively. Note that

$$f_\theta(x) = A^L(x), \qquad A^0(x) = x, \qquad \text{and} \qquad A^\ell(x) = \phi^\ell(Z^\ell(x)),$$

where $\phi^\ell(\cdot)$ denotes the activation function for layer ℓ. We consider any multi-dimensional layer that can be described by a linear map with *parameter sharing*, e.g. fully-connected layers, convolutional layers, or LSTM cells. Specifically, for a layer ℓ the pre-activation $Z^\ell(x)$ of layer ℓ is described by the linear mapping of the activation $A^{\ell-1}(x)$ with W^ℓ, i.e.,

$$Z^\ell(x) = W^\ell * A^{\ell-1}(x),$$

where $*$ denotes the operator of the linear map, e.g., the convolutional operator.

Parameter groups We denote by c^ℓ the number of parameter groups within a layer ℓ that do not interact with each other, e.g., individual filters in convolutional layers. Then, let

$$Z_i^\ell(x) = W_i^\ell * A^{\ell-1}(x), \quad i \in [c^\ell],$$

denote the i^{th} pre-activation channel of layer ℓ produced by parameter group W_i^ℓ. Then the entire pre-activations $Z^\ell(x)$ of a layer ℓ is constructed by appropriately concatenating the individual pre-activations from individual parameter groups, i.e.,

$$Z^\ell(x) = \left[Z_1^\ell(x), \ldots, Z_{c^\ell}^\ell(x)\right].$$

Moreover, we let $\eta^\ell \in \mathbb{N}$ denote the number of scalar values of $Z^\ell(\cdot)$ and let $\eta = \sum_{\ell=1}^L \eta^\ell$. Finally, let ρ denote the maximum number of parameters within a parameter group, i.e., $\rho = \max_{i,\ell} \|W_i^\ell\|_0$.

Patches Within a parameter group, parameters may be used multiple times, c.f. *parameter sharing*, in order to produce the output $Z_i^\ell(x) = W_i^\ell * A^{\ell-1}(x)$. For example, in case of a convolutional layer the filter W_i^ℓ gets "slid" across the input $A^{\ell-1}(x)$ of the layer in order to produce one output pixel after another. Hereby, the filter acts on a distinct *patch* of the layer input $A^{\ell-1}(x)$ in order to produce a specific output pixel $z(x) \in Z_i^\ell(x)$, where with slight abuse of notation $z(x)$ denotes a scalar entry of $Z_i^\ell(x)$. To precisely specify the associated operation that produces the output $z(x)$ we define by $\mathcal{A}_i^\ell$ the *set of patches* of the layer input $A^{\ell-1}(x)$ that are required to produce the output $Z_i^\ell(x)$. Specifically, let $a(\cdot) \in \mathcal{A}_i^\ell$ denote some patch of $\mathcal{A}_i^\ell$. Then, $a(\cdot) \subseteq A^{\ell-1}(\cdot)$ is defined such that a dot product between the parameter group W_i^ℓ and the patch $a(\cdot)$ produces the associated output scalar $z(x)$, i.e.,

$$z(x) = \langle W_i^\ell, a(x) \rangle = \sum_{k \in \mathcal{I}_i^\ell} w_k a_k(x),$$

where $\mathcal{I}_i^\ell$ denotes the index set of weights for the parameter group W_i^ℓ and $w_k, a_k(x)$ denote a scalar entry of the parameter group and patch for some input $x \sim \mathcal{D}$, respectively. Note that $\eta^\ell = \sum_{i \in [c^\ell]} \left|\mathcal{A}_i^\ell\right|$.

The notation of patch maps $\mathcal{A}$ lets us conveniently abstract away some of the implementation details of the linear map $*$ without restricting ourselves to a particular type of linear map $*$. For example in the context of convolutional layers, the actual linear map $*$ can significantly vary depending on the parameter settings such as stride length, padding, and so forth. It also enables us to consider other layers, such as recurrent layers, at the same time. In this case, $\mathcal{A}$ can be generated by considering each recursive input to the layer as a separate patch.

In the case of two-dimensional convolutions (i.e. for images), we note that our notion of patch maps corresponds to the Unfold operation in PyTorch [PyT20b], which we find to be a helpful reference to further contextualize the concept of patch maps.

5.2.2 Problem Definition

We now proceed to formally state the problem definition that motivates the use of SiPP and subsequent analysis. To this end, let the size of the parameter tuple θ, $\|\theta\|_0$, to be the number of all non-zero entries in the weight tensors $W^1, \ldots, W^L$.

Problem 2. *For given $\varepsilon, \delta \in (0,1)$, our overarching goal is to use a pruning algorithm to generate a sparse reparameterization $\hat{\theta}$ of θ such that $\|\hat{\theta}\|_0 \ll \|\theta\|_0$ and for $x \sim \mathcal{D}$ the ℓ_2-norm of the reference network output $f_\theta(x)$ can be approximated by $f_{\hat{\theta}}(x)$ up to*

$1 \pm \varepsilon$ multiplicative error with probability greater than $1-\delta$, i.e.,

$$\mathbb{P}_{\hat{\theta},x}\left(\|f_{\hat{\theta}}(x) - f_\theta(x)\| \le \varepsilon \|f_\theta(x)\|\right) \ge 1-\delta,$$

where $\|\cdot\| = \|\cdot\|_2$ and $\mathbb{P}_{\hat{\theta},x}$ considers the randomness over both the pruning algorithm and the network's input.

5.3 SiPP

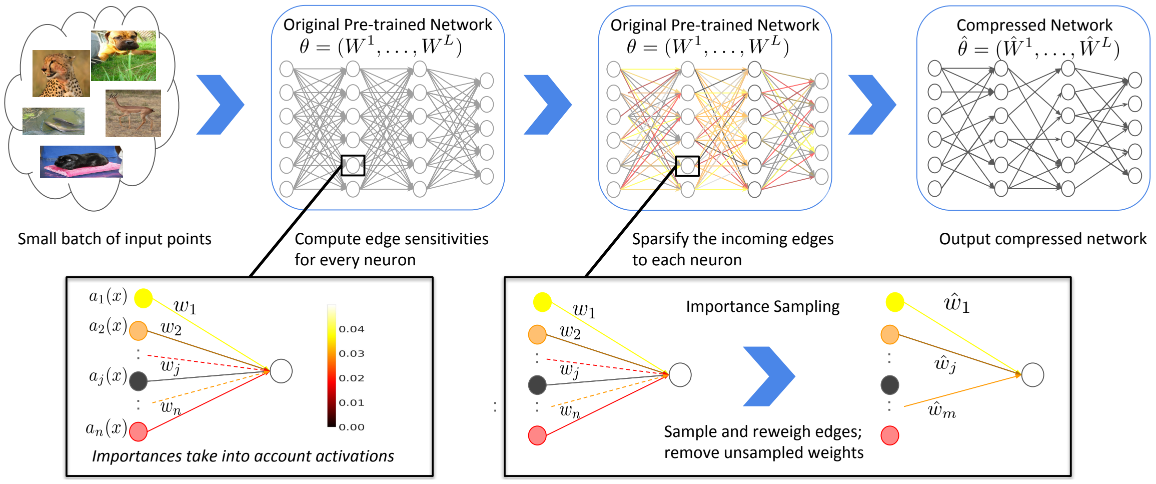

Original Pre-trained Network
$\theta = (W^1, \dots, W^L)$
Original Pre-trained Network
$\theta = (W^1, \dots, W^L)$
Compressed Network
$\hat{\theta} = (\hat{W}^1, \dots, \hat{W}^L)$
Small batch of input points
Compute edge sensitivities for every neuron
Sparsify the incoming edges to each neuron
Output compressed network
$a_1(x)$ w_1
$a_2(x)$ w_2
$a_j(x)$ w_j
$a_n(x)$ w_n
0.04
0.03
0.02
0.01
0.00
Importances take into account activations
Importance Sampling
w_1 w_2 w_j w_n
$\hat{w}_1$ $\hat{w}_j$ $\hat{w}_m$
Sample and reweigh edges; remove unsampled weights

Figure 5-1: The overview of our randomized method consisting of 4 parts. We use a small batch of input points to quantify the relative contribution (importance) of each edge to the output of each neuron. We then construct an importance sampling distribution over the incoming edges and sample a small set of weights for each neuron. The unsampled parameters are then discarded to obtain the resulting compressed network with fewer edges.

In this section, we present an overview for our family of pruning algorithms, SiPP: Sensitivity-informed Provable Pruning (see Figure 5-1 and Algorithm 6). In its core, SiPP proceeds as follows: (1) optimally allocate a given budget across layers and parameter groups to minimize the theoretical error bounds resulting from our analysis (OptAlloc, Line 1); (2) compute the relative importance of individual weights within each parameter group (EmpiricalSensitivity, Line 5); (3) prune weights within each parameter group using the desired variant of SiPP according to their relative importance (Sparsify, Line 6); (4) repeat steps (2) and (3) for each parameter group and each layer.

5.3.1 OptAlloc

In the course of our analysis (see Section 5.4) we establish relative error bounds for the approximation $\hat{Z}_i^\ell(x) = \hat{W}_i^\ell * A^{\ell-1}(x)$ of the form $\hat{Z}_i^\ell(x) \in \left(1 \pm \varepsilon_i^\ell(m_i^\ell)\right) Z_i^\ell(x)$ for individual parameter groups. Roughly speaking, the associated relative error $\varepsilon_i^\ell(m_i^\ell)$ is a (convex) function of the parameter group, the input, and the allocated budget m_i^ℓ. Thus in order to optimally utilize a desired budget $\mathcal{B}$ we aim to minimize the following objective during the allocation procedure:

$$\min_{m_i^\ell \in \mathbb{N}\ \forall i \in [c^\ell],\ \forall \ell \in [L]} \quad \textstyle\sum_{\ell \in [L],\ i \in [c^\ell]} \varepsilon_i^\ell(m_i^\ell) \quad \text{s. t.} \quad \textstyle\sum_{\ell \in [L],\ i \in [c^\ell]} m_i^\ell \leq \mathcal{B}. \tag{5.1}$$

We note that the integral constraint $m_i^\ell \in \mathbb{N}$ prevents us from efficiently finding a solution, so we build on techniques from Randomized Rounding [Sri99] to relax the constraint to $m_i^\ell \in \mathbb{R}$ to find the optimal fractional solution. We can then use the established rounding technique in [Sri99] to find an approximately optimal integral solution. Depending on the variant of SiPP, however, this step is not necessary.

Algorithm 6 SiPP $(\theta, \mathcal{B}, \mathcal{S})$

Input: $\theta = (W^1, \ldots, W^L)$: weights of the uncompressed neural network; $\mathcal{B} \in \mathbb{N}$: sampling budget; $\mathcal{S}$: a set of n i.i.d. validation points drawn from $\mathcal{D}$
Output: sparse weights $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$

```
1: $m_i^\ell \leftarrow$ OptAlloc$(\theta, \mathcal{B}, \mathcal{S})$ $\forall i \in [c^\ell]$, $\forall \ell \in [L]$    ▷ optimally allocate budget $\mathcal{B}$ across parameter groups and layers
2: for $\ell \in [L]$ do
3:     $\hat{W}^\ell \leftarrow \mathbf{0}$;    ▷ Initialize a null tensor
4:     for $i \in [c^\ell]$ do
5:         $s_j \leftarrow$ EmpiricalSensitivity$(\theta, \mathcal{S}, i, \ell)$ $\forall w_j \in W_i^\ell$    ▷ Compute parameter importance for each weight $w_j$ in the parameter group
6:         $\hat{W}_i^\ell \leftarrow$ Sparsify$(W_i^\ell, m_i^\ell, \{s_j\}_j)$    ▷ prune weights according to SiPPDet, SiPPRand, or SiPPHybrid such that only $m_i^\ell$ weights remain
7: return $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$;
```

5.3.2 EmpiricalSensitivity

To estimate the relative importance of weight w_j within a parameter group W_i^ℓ, we use and extend the notion of *empirical sensitivity* (ES) first introduced in [BLG⁺19a] for fully-connected layers only. In its essence, ES quantifies the maximum relative contribution of a weight parameter w_j to the output (pre-activation) of the layer compared to other weights in the parameter group. More formally, let the ES s_j of w_j in parameter group W_i^ℓ be defined as

$$s_j = \max_{x \in \mathcal{S}} \max_{a(\cdot) \in \mathcal{A}} \frac{w_j a_j(x)}{\sum_k w_k a_k(x)}, \tag{5.2}$$

where we assume $W_i^\ell \geq 0$, $A^{\ell-1}(x) \geq 0$ for ease of exposition (see Section 5.7 for the generalization to all weights and activations). We note that the required size of $\mathcal{S}$ is given explicitly by our analysis and is a function of the desired failure probability δ. More specifically, in Lemma 26 of Sec. 5.7, we show that the required size of $\mathcal{S}$ is roughly $\mathcal{O}(\log(\eta/\delta))$, where $\eta = \sum_{\ell \in [L]} c^\ell$ is the total number of parameters in the original network θ. Taking a larger $\mathcal{S}$ would push this failure probability further down, however, there is a trade-off: a larger $\mathcal{S}$ leads to a larger parameter sample-complexity (Theorem 37). This is because the sample-complexity of each parameter group is roughly linear in the sum of sensitivities S, and S is non-decreasing with the size of $\mathcal{S}$ by definition of ES (5.2). Thus, we take the smallest-sized $\mathcal{S}$ that ensures the desired failure probability, as explicitly provided in Theorem 37.

We note that the definition of s_j entails two maxima. The maximum over data points $\max_{x \in \mathcal{S}}$ ensures that ES captures the relative importance of w_j sufficiently well for any i.i.d. data point $x \sim \mathcal{D}$[1]. The maximum over patches $\mathcal{A}$, which are generated from $A^{\ell-1}(x)$, ensures that ES approximates the relative importance of w_j sufficiently well for *all scalars in the output* $Z_i^\ell(x)$ that require w_j (cf. parameter sharing). To further contextualize the purpose of patches $\mathcal{A}$, consider a single parameter group within a convolutional layer, i.e., a filter. The filter is slid across the input image to

[1]The maximum is necessary to obtain the high-probability bounds we seek to establish.

generate the output image by repeatedly applying the same weights (i.e., convolution operator). Thus in order to quantify the importance of some weight w_j we need to consider its relative importance across all sliding windows, hence culminating in the $\max_{a(\cdot)\in\mathcal{A}}$ operation.

5.3.3 Sparsify

Equipped with the allocated budget (OptAlloc) and a notion of parameter importance (EmpiricalSensitivity), we introduce the three variants of SiPP, whose analysis can be found in Section 5.4, to prune weights from a parameter group:

1. SiPPDet: we deterministically pick the m_i^ℓ weights with largest sensitivity and zero out the rest of the weights to construct $\hat{W}_i^\ell$.

2. SiPPRand: we construct an importance sampling distribution over weights w_j using their associated sensitivities s_j, then sample with replacement until we obtain a set of m_i^ℓ unique weights to construct $\hat{W}_i^\ell$.

3. SiPPHybrid: we evaluate the theoretical error guarantees (see Section 5.4) associated with the two other methods, and prune using the method that incurs the lower relative error.

We note that while SiPPDet is particularly simple to implement, SiPPHybrid provides the biggest amount of flexibility and consistently good prune results since it can adaptively choose for each parameter group whether to prune using SiPPDet or SiPPRand.

5.4 Analysis

In this section, we outline the theoretical guarantees for SiPP. We start out by establishing the core lemmas that constitute the relative error guarantees for both SiPPDet and SiPPRand for the case where $W_i^\ell \geq 0$, $A^{\ell-1}(x) \geq 0$ for ease of exposition. Specifically, we establish relative error guarantees for each individual output patch that is associated with a parameter group. We then outline the steps that are required to generalize the analysis to all weights and activations. Finally, we show — by means of composing together the error guarantees from individual output patches, parameters groups, and layers — how to derive the analytical compression bounds for the entire network. For clarity of presentation, we omit technical proofs from this section and provide the full proofs in Sec. 5.7.

5.4.1 Empirical sensitivity

In the previous section we introduce the notion of ES, see equation 5.2, as a means to quantify the importance of weight w_j relative to the other weights within a parameter group W_i^ℓ. Using ES we establish a key inequality that upper bounds the contribution of $w_j a_j(x)$ to its associated output patch $z(x) = \sum_k w_k a_k(x)$ for any $x \sim \mathcal{D}$ with high probability (w.h.p.) under a mild regularity assumption on the input distribution to the layer.

Lemma 19 (Informal ES inequality)**.** *For weights w_j from parameter group W_i^ℓ and an arbitrary input patch $a(\cdot)$ we have w.h.p. for any $x \sim \mathcal{D}$ that $w_j a_j(x) \leq C s_j z(x)$, where $z(x)$ denotes the associated output patch and $C = \mathcal{O}(1)$.*

The ES inequality is a key ingredient in bounding the error of SiPPDet and SiPPRand in terms of sensitivity. Specifically, Lemma 19 puts the individual contribution of a weight to the output patch in terms of its sensitivity and the output patch itself. The inequality hereby holds *w.h.p.* for any data point $x \sim \mathcal{D}$ which enables us to bound the quality of the approximation even for previously unseen data points. We leverage Lemma 19 in the subsequent analysis to quantify the approximation error of an output patch when the output patch was only approximately computed using a subset of weights, i.e., with the weights that remain after pruning.

5.4.2 Error guarantees for SiPPDet

Recall that SiPPDet prunes weights by keeping the m_i^ℓ weights of parameter group W_i^ℓ with largest ES. Now let $\mathcal{I}$ denote the index set of all weights in W_i^ℓ and $\mathcal{I}_{det}$ the index set of weights with largest sensitivity that are kept after pruning such that $|\mathcal{I}_{det}| = m_i^\ell$. We bound the incurred error of the approximation by considering the difference between the output patch and the approximated output patch, i.e., the difference between $z(x) = \sum_{j \in \mathcal{I}} w_j a_j(x)$ and $\hat{z}_{det}(x) = \sum_{j \in \mathcal{I}_{det}} w_j a_j(x)$.

Lemma 20 (Informal SiPPDet error bound)**.** *For weights w_j from parameter group W_i^ℓ, an arbitrary associated input patch $a(\cdot) \in \mathcal{A}$, and corresponding output patch $z(\cdot)$* SiPPDet *generates an index set $\mathcal{I}_{det}$ of pruned weights such that for any $x \sim \mathcal{D}$ w.h.p. $|\hat{z}_{det}(x) - z(x)| \leq \varepsilon_{det} z(x)$, where $\varepsilon_{det} = C \sum_{j \in \mathcal{I} \setminus \mathcal{I}_{det}} s_j$.*

The proof of Lemma 20 follows from the fact that the difference between the approximate output patch $\hat{z}_{det}(x)$ and the unpruned output patch $z(x)$ is exactly the sum over the contributions from weights that are *not* in the pruned subset of weights $\mathcal{I}_{det}$. Using Lemma 19 we then bound the error in terms of the sensitivity of the *pruned* weights. Intuitively, ES of an individual weight precisely quantifies the relative error incurred when that weight is pruned. The resulting relative error can thus be described by the cumulative ES of pruned weights.

5.4.3 Error guarantees for SiPPRand

Here we prune weights from a parameter group by constructing an importance sampling distribution from the associated ESs. Specifically, some weight w_j is sampled with probability $q_j = {s_j}/{\sum_{k \in \mathcal{I}} s_k}$ and we repeatedly sample with replacement until the corresponding set of sampled weights contains m_i^ℓ unique weights. Each sampled weight is then reweighed by the number of times it was sampled divided by the total number of samples and its sample probability to construct the approximate output patch, i.e.,

$$\hat{z}_{rand}(x) = \sum_{j \in \mathcal{I}_{rand}} \hat{w}_j a_j(x) = \sum_{j \in \mathcal{I}_{rand}} \frac{n_j}{N q_j} w_j a_j(x),$$

where $\mathcal{I}_{rand}$ denotes the index set of weights that were sampled at least once, n_j denotes the number of times weight w_j was sampled, and $N = \sum_{j \in \mathcal{I}_{rand}} n_j$ denotes the total number of samples. We then bound the incurred error by analyzing the random difference between the approximated output patch and the original output patch, i.e., $|\hat{z}_{rand}(x) - z(x)|$, establishing the following error guarantee.

Lemma 21 (Informal SiPPRand error bound)**.** *For weights w_j from parameter group W_i^ℓ, an arbitrary associated input patch $a(\cdot) \in \mathcal{A}$, and corresponding output patch $z(\cdot)$ SiPPRand generates a set of pruned weights such that for any $x \sim \mathcal{D}$ w.h.p. $|\hat{z}_{rand}(x) - z(x)| \leq \varepsilon_{rand} z(x)$, where $\varepsilon_{rand} = \mathcal{O}(\sqrt{S/N})$ and $S = \sum_{k \in \mathcal{I}} s_k$ denote the relative error and sum of ESs, respectively.*

The proof proceeds in two steps. First, we show that the (random) approximation is an unbiased estimator of the original parameter group, i.e., $\mathbb{E}\left[\hat{z}_{rand}(x)\right] = z(x)$, which follows from the reweighing term of $\hat{w}_j$. Second, we show that using Bernstein's concentration inequality [Ver16] the sampling distribution exhibits strong subGaussian [Ver16] concentration around the mean, i.e., the approximate output patch is ε-close to the original, unpruned output patch w.h.p. Specifically, we leverage Lemma 19 to bound the variance of the approximate output patch using the cumulative ES of the parameter group $S = \sum_{k \in \mathcal{I}} s_k$.

5.4.4 Discussion of error bounds and SiPPHybrid

Most notably, SiPPRand is an unbiased estimator regardless of the budget, while SiPPDet is not. On the other hand, if the parameter group is dominated by a few weights, SiPPDet can directly capture these weights with a small number of samples whereas the randomness and reweighting of SiPPRand may introduce additional sources of failure in the approximation. Combining the strengths of both, we introduce SiPPHybrid, which compares the theoretical error guarantees of each variant before pruning a parameter group to adaptively choose the better prune strategy.

5.4.5 Generalization to all weights

Previously, we have assumed that both the parameter group and input activations are strictly non-negative, i.e., $W_i^\ell \geq 0$ and $A^{\ell-1}(x) \geq 0$. To handle the general case, we split the parameter group and input activations each into a positive and negative part representing the four quadrants such that each quadrant is now strictly non-negative. We can then incorporate each quadrant into our pruning procedure to ensure that the error guarantees hold simultaneously for all quadrants. To obtain error bounds for the actual pre-activation we introduce Δ^ℓ, which quantifies the "sign complexity" of the overall approximation for a particular layer to quantify the additional complexity from considering the alternating signs of each quadrant. For ease and clarity of exposition, we defer the full technical details and complete proofs to Sec. 5.7.

5.4.6 Network compression bounds

In the previous section we sketched the error guarantees for individual output patches. Naturally, since the guarantees hold for all patches within a parameter group and individual parameter groups within a layer are independent from each other, we can simultaneously establish norm-based error guarantees for the entire pre-activation of a layer, i.e., $\left\|\hat{Z}^\ell(x) - Z^\ell(x)\right\| \leq \varepsilon \left\|Z^\ell(x)\right\|$ w.h.p. Moreover, assuming the activation function is entry-wise and 1-Lipschitz[2], the same relative error guarantees hold for the activation of layer. Note that any common activation function satisfies the above assumption, including and all others listed in PyTorch's documentation [PyT20a]. Finally, we consider the effect of pruning *multiple layers* simultaneously and the implications of the layer-wise error on the final output $f_\theta(x) = A^L(x)$ of the network. Informally speaking, we incur two sources of error from each layer: (1) the error associated from pruning within layers and (2) the error associated with propagated the incurred error throughout the network to the output layer. We quantify the error within layers using our patch-wise guarantees and the sign complexity Δ^ℓ of the layers. We quantify the propagated error across layers by upper bounding the layer condition

[2]Can be generalized to L-Lipschitz functions as well, with the error bound depending on L.

number, κ^ℓ which quantifies the relative error incurred in the output for some relative error incurred within the layer. Intuitively, the concept of the layer condition number is closely related to the Lipschitz constant between some layer and the output of the network. Below, we informally state the compression bound when pruning the entire network with SiPPRand.

Theorem 22 (Informal compression bound)**.** *For given $\delta \in (0,1)$ and budget $\mathcal{B}$ SiPP (Algorithm 6) generates a set of pruned parameters $\hat{\theta}$ such that $\|\hat{\theta}\|_0 \leq \mathcal{B}$,*

$$\mathbb{P}_{\hat{\theta},x}\left(\|f_{\hat{\theta}}(x) - f_\theta(x)\| \leq \varepsilon \,\|f_\theta(x)\|\right) \geq 1 - \delta,$$

and $\varepsilon = \mathcal{O}(\sum_{\ell=1}^{L} \kappa^\ell \Delta^\ell \max_{i \in [c^\ell]}(S_i^\ell - S_i^\ell(N_i^\ell)))$, where S_i^ℓ and $S_i^\ell(N_i^\ell)$ is the sum over all and the largest N_i^ℓ ESs, respectively, and N_i^ℓ is the budget allocated for parameter group W_i^ℓ.

We note that the compression bound is proportional to the sum of cumulative ESs for each parameter group, a term which arises in numerous applications of coresets [FL11]. Moreover, we see the layer condition number κ^ℓ and sign complexity Δ^ℓ of each layer appear in the final bound. Both terms are related to how injecting error simultaneously in each layer (by pruning the network) affects the overall output of the network and are related to concepts such as the Lipschitz constant of the network and/or interlayer cushion as introduced in related work that establishes generalization bounds for neural networks [AGNZ18, NBS18]. Like other recent work in the field [AGNZ18, SAN20] our work highlights the intrinsic connection between the compression ability and generalization ability of neural networks.

5.4.7 Computation time

Note that for all the three variants, we pass $|\mathcal{S}|$ data points as input to the original network θ in order to compute the activations in all layers corresponding to the validation set $\mathcal{S}$ (to compute ESs). If we let $\mathcal{T}$ denote the time required to compute the output of a single point ($f_\theta(x)$) on the original network parameterized by θ, then the total time required by the preprocessing step to compute the empirical sensitivities is $\mathcal{O}(|\mathcal{S}|\mathcal{T})$. For the RAND variant, sampling and reweighting weights takes asymptotically linear time in the number of parameters for each parameter group, giving a total time of $\mathcal{O}(\eta)$ where $\eta = \sum_{\ell \in [L]} c^\ell$ is the total number of network parameters. For the hybrid and deterministic approaches, the computational complexity is dominated by a sort of all of the empirical sensitivities, which takes $\mathcal{O}(\eta \log \eta)$ time. Thus, the sparsification itself for any variant takes at most $\mathcal{O}(\eta \log \eta)$ time. Finally, the budget allocation problem in (5.1) is a special type of convex optimization problem known as the *water-filling problem* [BV04, Example 5.2] and can be solved in $\mathcal{O}(\eta \log \eta)$ time by a clever sorting procedure [PF05]. Putting it all together, the total computation time required of pruning a network with any SIPP variant is at most

$$\mathcal{O}\left(\max\{|\mathcal{S}|\mathcal{T}, \eta \log \eta\}\right) \tag{5.3}$$

time, where $|\mathcal{S}|$ is roughly logarithmic in $1/\delta$ (see Lemma 26 in Sec. 5.7 for details) and, as before, $\eta = \sum_{\ell \in [L]} c^\ell$ and $\mathcal{T}$ is the computation time required to compute $f_\theta(x)$ for a single input x.

5.4.8 Classification Error

As a corollary to the norm-based bounds established in Theorem 37, we can also obtain margin-based bounds for the relative classification error of the pruned network. Without loss of generality, assume for the time being that the output softmax probabilities $\sigma(f_\theta(x)) \in [0,1]^K$ of an output $f_\theta(x)$ are in descending order so that $\sigma(f_\theta(x))_0 \geq \sigma(f_\theta(x))_1 \geq \cdots \geq \sigma(f_\theta(x))_K$[3]. For a given network θ, we can define the *margin* by considering the worst-case margin across the entire support of the distribution $\mathcal{D}$, $\gamma_\theta = \inf_{x \in \mathrm{supp}(\mathcal{D})}(\sigma(f_\theta(x))_0 - \sigma(f_\theta(x))_1)$, i.e., the minimum difference between the softmax output of the predicted label and the second highest probability label.

[3]This is only to make the definition of the margin more clear.

However, in the spirit of obtaining better than worst-case guarantees (with sufficiently high probability), we can instead define the less stringent margin that takes into account the randomness (inspired by [PBD18]) in the input $x \sim \mathcal{D}$ with respect to failure probability $\delta \in (0, 1)$ and network θ:

$$\gamma_\theta(\delta) = \inf \left\{ \gamma > 0 : \mathop{\mathbb{P}}_{x \sim \mathcal{D}} \left(\sigma(f_\theta(x))_0 - \sigma(f_\theta(x))_1 > \gamma \right) \geq 1 - \delta \right\}.$$

In words, this definition means that the minimum margin between the top-2 predicted labels is at least $\gamma_\theta(\delta)$ for $(1 - \delta)$ fraction (more rigorously, relative volume) of the points in $\operatorname{supp}(\mathcal{D})$. Given this margin and an original network θ with test accuracy A_θ, the corollary below provides a sufficient size for the pruned network so that the test accuracy of the pruned network is at least $(1 - \delta)\mathrm{A}_\theta$ with high probability.

Corollary 23 (Classification Error Bound). *For given $\delta \in (0, 1)$, original network θ, and a margin $\gamma_\theta(\delta/2)$ as defined above, define the budget $\mathcal{B}$ so that the absolute incurred error (as in Thm. 37) is less than $\gamma_\theta(\delta/2)/2$, i.e.,*

$$\mathcal{B} = \operatorname{argmin} \left\{ \mathcal{B}' \in \mathbb{N} : C \sum_{\ell=1}^{L} \kappa^\ell \Delta^\ell \max_{i \in [c^\ell]} (S_i^\ell - S_i^\ell(N_i^\ell(\mathcal{B}'))) < \frac{\gamma_\theta(\delta/2)}{2 \left\| f_\theta(x) \right\|_2} \right\}$$

where $N_i^\ell(\mathcal{B}')$ is the budget allocated for parameter group W_i^ℓ with respect to budget $\mathcal{B}'$, and $C > 0$ is a universal constant. Then, invoking algorithm SiPP *(Algorithm 6) with budget $\mathcal{B}$ and failure probability $\delta' = \delta/2$ generates a set of pruned parameters $\hat{\theta}$ such that $\|\hat{\theta}\|_0 \leq \mathcal{B}$ and*

$$\mathop{\mathbb{P}}_{\hat{\theta}, x} \left(\operatorname*{argmax}_{i \in [k]} f_{\hat{\theta}}(x)_i = \operatorname*{argmax}_{i \in [k]} f_\theta(x)_i \right) \geq 1 - \delta.$$

Proof. Without loss of generality, assume the descending order

$$\sigma(f_\theta(x))_0 \geq \sigma(f_\theta(x))_1 \geq \cdots \geq \sigma(f_\theta(x))_K,$$

note that by Thm. 37 with

$$\varepsilon(\mathcal{B}') < \gamma_\theta(\delta/2)/(2\left\|f_\theta(x)\right\|_2),$$

we have that with probability at least $1-\delta/2$,

$$\left\|f_{\hat{\theta}}(x) - f_\theta(x)\right\|_2 \leq \gamma_\theta(\delta/2)/2.$$

Now, since the softmax function $\sigma(\cdot)$ defined by $x \mapsto \Big(\exp(x_i)/\sum_{j\in[k]}\exp(x_j)\Big)_{i\in[k]}$ is 1-Lipschitz [GP17], we have with probability at least $1-\delta/2$

$$\begin{aligned}\left\|\sigma(f_{\hat{\theta}}(x)) - \sigma(f_\theta(x))\right\|_2 &\leq \left\|f_{\hat{\theta}}(x) - f_\theta(x)\right\|_2 \\ &< \frac{\gamma_\theta(\delta/2)}{2}.\end{aligned}$$

Using the inequality for norms $\|x\|_2 \geq \|x\|_\infty$ yields

$$\left\|\sigma(f_{\hat{\theta}}(x)) - \sigma(f_\theta(x))\right\|_\infty < \frac{\gamma_\theta(\delta/2)}{2}. \tag{5.4}$$

This error bound implies that for the top-2 highest probability labels $i, j \in [k]$ with respect to output $\sigma(f_\theta(x))$, (corresponding to indices $i=0$ and $j=1$ if we assume σ is ordered), we have

$$\begin{aligned}\sigma(f_{\hat{\theta}}(x))_i - \sigma(f_{\hat{\theta}}(x))_j &> (\sigma(f_\theta(x))_i - \gamma_\theta(\delta/2)/2) - (\sigma(f_\theta(x))_j + \gamma_\theta(\delta/2)/2) \\ &= \sigma(f_\theta(x))_i - \sigma(f_\theta(x))_j - \gamma_\theta(\delta/2) \\ &> 0,\end{aligned}$$

where the first inequality follows by the error bound (5.4) applied to each term and the last inequality by the definition of the margin $\gamma_\theta(\delta/2)$. The strict inequality above means that the relative ranking between the top-2 softmax outputs i and j does not differ for the pruned network $\hat{\theta}$, and so i remains the predicted label, i.e., $\operatorname{argmax}_{c\in[k]} f_{\hat{\theta}}(x)_c = \operatorname{argmax}_{c\in[k]} f_\theta(x)_c$. Applying the union bound over the two failure events that each occur with probability at most $\delta/2$ yields the corollary. □

5.5 Experiments

In this section, we evaluate and compare the performance of our algorithm, SiPP, on pruning fully-connected, convolutional, and residual networks. We embed our pruning algorithm into pruning pipelines including retraining to empirically test its performance and test it for scenarios involving significant amounts of (re)-training as well as a prune pipeline that utilizes no more training epochs than regular training. We consider standard retraining pipelines inspired by [LAT18, RFC20] that are network-agnostic and thus easily adaptable to various benchmarks. Specifically, we consider two scenarios – iterative prune + retrain and random-init + prune + train – as described below.

5.5.1 Setup

Architectures and data sets We train and prune networks on MNIST [LBBH98], CIFAR10 [TFF08], and ImageNet [RDS+15]. We consider a custom fully-connected architecture according to [AAR20] and LeNet300-100 [LBBH98] on MNIST; ResNets20/56/110 [HZRS16], WideResNet16-8/28-2 [ZK16], Densenet22 [HLVDMW17], VGG16 [SZ14a], and CNN5 [NKB+20] on CIFAR10; and ResNet18 and ResNet101 [HZRS16] for ImageNet.

Training For both training and retraining we deploy the standard sets of hyperparameters as described in the respective papers. All hyperparameters are listed in Sec. 5.8.

Variants of SiPP We consider the following variants of SiPP for our experiments:

- SiPPDet. We prune the entire network deterministically. Note that in this case (due to the sample size allocation procedure) SiPPDet corresponds to global thresholding of sensitivity (reminiscent of weight thresholding).

- SiPPRand. We prune the entire network using importance sampling.
- SiPPHybrid. We use our combined pruning approach as outlined in Algorithm 6.

Comparison methods We compare SiPP to a diverse set of pruning methods. Specifically, we consider Weight Thresholding (WT) [HMD15, RFC20] and SNIP [LAT18] for our baseline experiments in our standardized retraining pipeline. Additionally, we compare to Net-Trim [AAR20, AANR17], Bayesian Compression (BC) [LUW17], Dynamic Network Surgery (DNS) [GYC16], Dynamic Sparse Reparameterization (DSR) [MW19], Deep Rewiring (DeepR) [BKML17], Sparse Evolutionary Training (SET) [MMS+18], "To prune, or not to prune" (TPNTP) [ZG17], Alternating Direction Method of Multipliers (ADMM) [ZYZ+18], Unified Approximation Framework (UAF) [MCL+19], and Learning Compression (LC) [CPI18].

5.5.2 Experiments with baseline comparisons

We provide comparisons to two frequently used pruning heuristics, namely WT [HMD15, RFC20], which is weight-based, and SNIP [LAT18], which is gradient-based. To highlight the versatility of SiPP we consider two pruning pipelines that differ in the total amount of retraining required (iterative retraining and no retraining).

Iterative prune + retrain

Methodology We deploy a network-agnostic iterative prune + retrain scheme inspired by [RFC20, LBL+20] that proceeds as follows:

1. train network to completion;
2. prune a fixed ratio of parameters from the network;
3. retrain using the same hyperparameters as during training;
4. iteratively repeat steps 2., 3. to obtain smaller prune ratios.

We consider SiPPDet as comparison method for WT and SNIP in this setup due to its simplicity.

Results Figure 5-2 summarizes the results of the iterative prune + retrain procedure for various CIFAR10 networks. The results were averaged across 3 trained networks. Our empirical evaluation shows that our algorithm consistently performs comparably to WT with learning rate rewinding [RFC20]. We note that SNIP's performance is much lower is these scenarios. We suspect this is due to the gradients being close to zero for a fully-trained network (the pruning step is performed after training in this scenario). In Figure 5-3 we show results for ResNet18 and ResNet101 trained, pruned, and retrained on ImageNet. As in the case of CIFAR10 networks we observe that SiPP performs en par with WT. We excluded SNIP from the experiments given the expensive nature of ImageNet experiments and since WT is the clearly stronger baseline for this scenario.

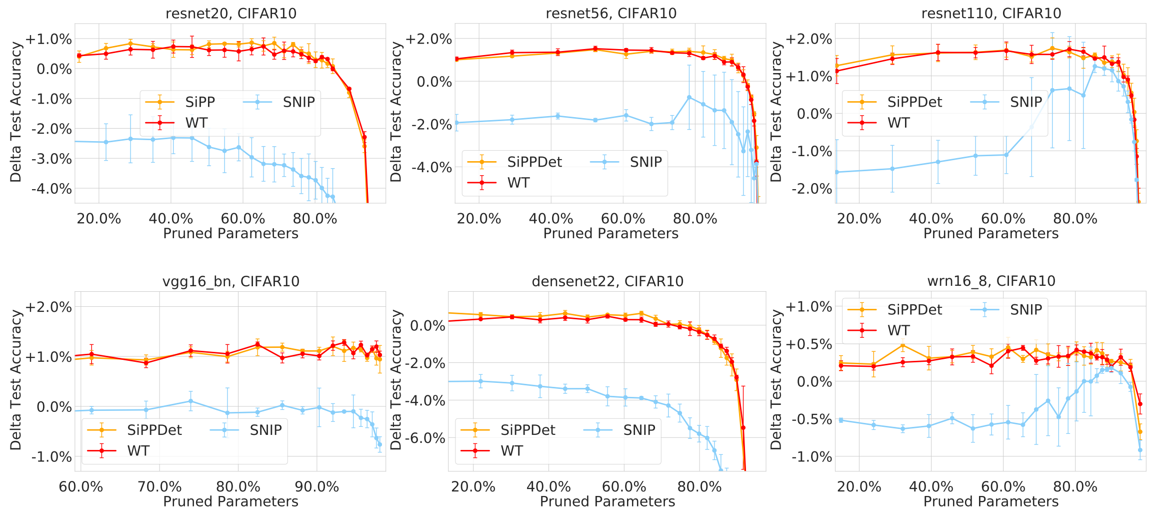
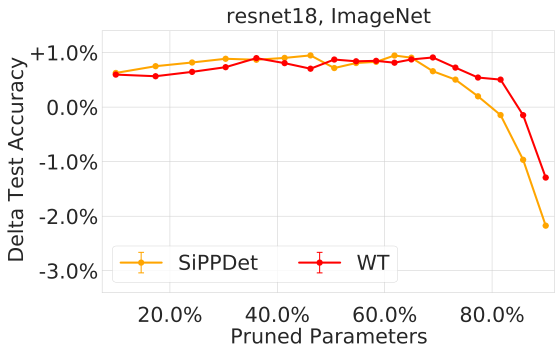


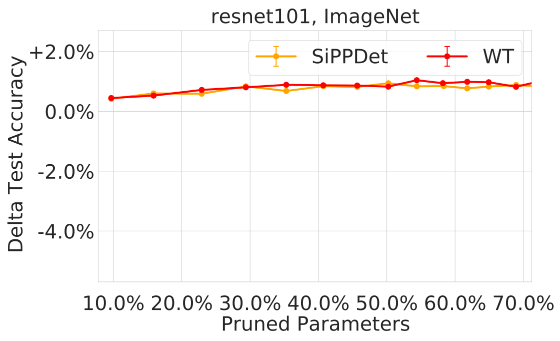
Figure 5-2: The delta in test accuracy to the uncompressed network for the generated pruned models trained on CIFAR10 for various target prune ratios. The networks were pruned using the **iterative prune+retrain** pipeline.

Random-init + prune + train

Methodology On the other "extreme" of possible pruning pipeline, we consider the following scenario as described in [LAT18]:



(a) ResNet18, Top 1



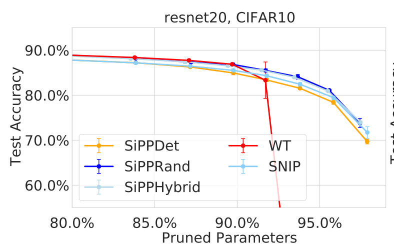
(b) ResNet101, Top 1

Figure 5-3: The accuracy of the generated pruned **ResNet18** and **ResNet101** models trained on ImageNet for the evaluated pruning schemes for various target prune ratios.

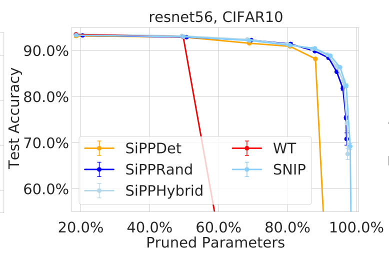
1. randomly initialize the network;
2. prune the network to the desired prune ratio;
3. train the network using the regular hyperparameters.

While (due to the limited amount of training) this pipeline does not achieve as high prune ratios as the above scenario, it is simple and requires much less training epochs overall. It also serves as a useful experimental platform to understand if pruning methods are able to unearth important connections inherent in the network.

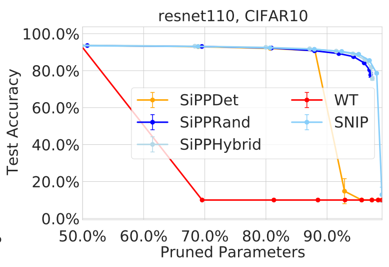
Results In Figure 5-4 the prune results for various CIFAR10 networks are shown. We note that for low prune ratios all pruning methods perform uniformly well, which most likely can be attributed to the overall overparameterization of the tested networks. For higher prune ratios, we observe vastly different performance. Specifically, WT's performance drops to 10% test accuracy (uniformly at random for CIFAR10) for prune ratios beyond 90%. We suspect that weights do not contain sufficient information about the importance of the connection before training and thus WT fails. On the other hand, SNIP performs consistently well due to the consideration of data and the gradients of weights. We note that SIPPHYBRID specifically, which adaptively mixes SIPP and SIPPRAND according to the theoretical bounds, performs well across all tested networks and achieves the same prune performance as SNIP. For deeper



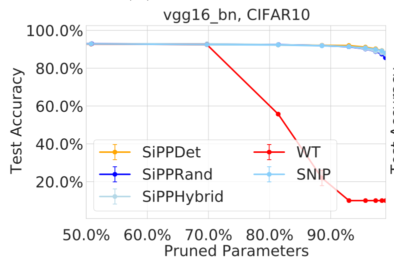
(a) Resnet20



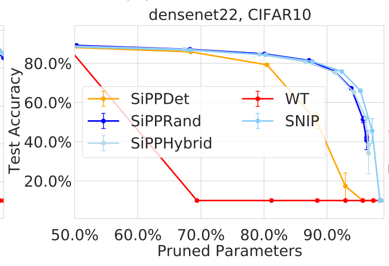
(b) Resnet56



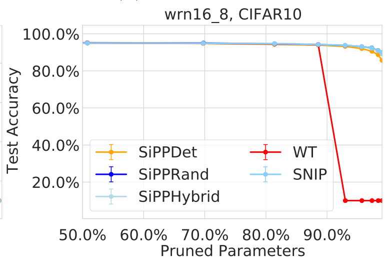
(c) Resnet110



(d) VGG16



(e) DenseNet22



(f) WRN16-8

Figure 5-4: The delta in test accuracy to the uncompressed network for the generated pruned models trained on CIFAR10 for various target prune ratios. The networks were pruned using the **random-init+ prune+train** pipeline.

networks (ResNet20 and ResNet56) in particular, we observe all SiPP variations performing well or even outperforming Snip.

5.5.3 Benchmark comparisons

Next, we test our method on additional benchmarks using the prune pipeline outlined in Section 5.5.2. For each benchmark we compare to available results in the literature as reported in the respective papers.

Fully-connected network on MNIST

We consider the performance of SiPP on the fully-connected architecture of [AAR20] (denoted by "FC network" in [AAR20]). Specifically, we compare SiPP to the results presented in Figure 9a of [AAR20] (Net-Trim) which additionally includes BC and DNS as comparison methods. Our results are presented in Figure 5-5. We observe that SiPP can prune more weights with higher accuracy compared to other prune methods (over 95% pruning with more than 99% accuracy). Similar to Section 5.5.2, we note that SiPPDet and SiPPHybrid work particularly well when considering

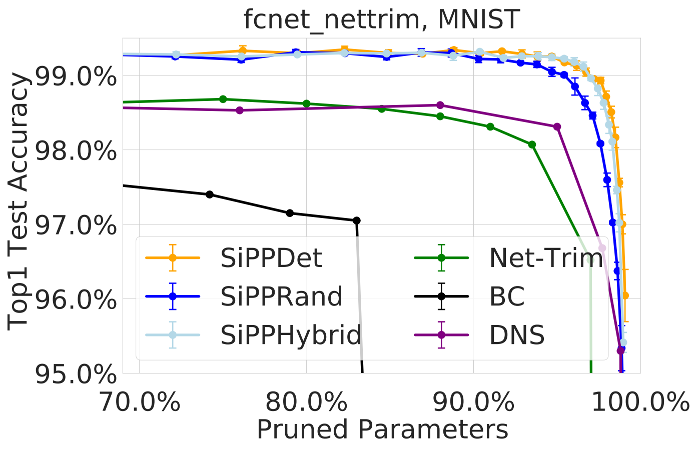


Figure 5-5: The performance of the SiPP variants and the competing baseline approaches in pruning FC-net (MNIST).

more retraining.

LeNet300-100 on MNIST

Next, we compared the performance of the SiPP variants to the performance of ADMM and LC algorithms as reported in the respective paper. Our results for pruning the LeNet300-100 network trained on MNIST are given in the upper part of Table 5.1. From the results, we can see that all SiPP variants achieve a test accuracy above 98.57% with a prune ratio of at least 94.90%, outperforming both ADMM and LC overall. For instance, SiPPHybrid achieves roughly the same sparsity as does LC (slightly over 96%), but yet the pruned network generated by SiPPHybrid is nearly 1% better in absolute test accuracy. Interestingly, even our worst-performing variant (SiPPRand) achieves a test accuracy higher than 98.65% (+0.25% w.r.t. ADMM and approximately +0.8% w.r.t. LC), and yet is only slightly worse than ADMM in terms of the prune ratio (last column of Table 5.1).

VGG16 on CIFAR10

In the previous subsection we compared to the baselines on LeNet300-100, a fully-connected network with 267,000 parameters, trained on MNIST. Here, we consider a more challenging pruning scenario where we prune VGG16, a CNN with roughly 138

Table 5.1: Results of our evaluations on LeNet300-100 trained on MNIST (upper part) and VGG16 trained on CIFAR10 (lower part). The value of the best-performing method that achieves commensurate (within 0.5%) test accuracy with respect to each objective is shown in **bold**. Overall, the table shows that the SiPP variants – notably SiPPDet and SiPPHybrid – outperform the competing baselines in obtaining a compact network with commensurate accuracy on both of the evaluated scenarios.

	Method	Accuracy (%)	Prune Ratio (%)
LeNet300-100 (MNIST) Acc: 98.75%	SiPPDet	98.57	**96.19**
	SiPPRand	**98.65**	94.90
	SiPPHybrid	**98.65**	96.16
	ADMM [ZYZ+18]	98.40	95.63
	LC [CPI18]	97.79	96.5[4]
VGG16 (CIFAR10) Acc: 92.78%	SiPPDet	**93.73**	**98.00**
	UAF [MCL+19]	91.65	77.5

million parameters, on CIFAR10 and compare the performance of SiPPDet to that of UAF. The results are shown in the lower part of Table 5.1. From the reported test accuracy and prune ratio, we can see that the network generated by SiPP is not only over 2% (absolute terms) better in test accuracy, but it also 20% (absolute terms) more compact than the pruned network generated by UAF.

In sum, the evaluations reported in Table 5.1 suggest that the SiPP variants, and notably SiPPHybrid and SiPPDet, outperform the competing methods in generating sparse networks while retaining (or improving) the predictive power (i.e., test accuracy) of the original network.

WRN28-2 on CIFAR10

In the previous comparisons, we saw that SiPP outperformed baseline approaches across both fully-connected and convolutional architectures. To conclude our comparisons, we consider benchmarking the performance of SiPP variants on WideResNet (WRN28-2) trained on CIFAR10. In particular, we compare the effectiveness of SiPP with that of the DSR, DeepR, SET, and TPNTP pruning algorithms as reported in Figure 1a of [MW19] (DSR). Fig. 5-6 depicts the test accuracies obtained for various prune ratios. We can see from the figure that *all* SiPP variants outperform all of the

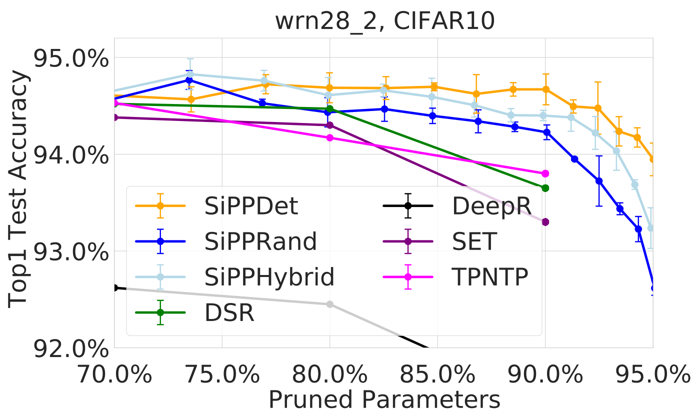


Figure 5-6: The performance on the WRN28-2 architecture trained on CIFAR10. The figure shows that all SiPP variants outperform the competing methods at all intermediate prune ratios.

competing approaches across all levels of sparsity. Notably, some of the SiPP variants such as SiPPDet and SiPPHybrid, outperform the best-performing baseline by over 0.5% (see 90% parameters retained)[5]. Taken in whole with the results reported in Table 5.1, we can see that SiPP remains uniformly more effective than the compared baselines across all evaluated architectures and data sets.

5.5.4 Empirical computation time of SiPP

For small networks and data sets (MNIST), our algorithm took at most 5 seconds total to sparsify the entire network. For ResNet18 trained on CIFAR10, this computation was at most 18 seconds. For the largest network and data set (e.g., ImageNet results), our algorithm took at most 10 minutes total to prune the given network θ. These timings suggest that the computation time required by SiPP is on-par with or significantly less than the computation time reported for the same pruning scenarios in previous work (e.g., [AAR20]). In context of the theoretical computation complexity (see (5.3)), we also observed that $|\mathcal{S}|\mathcal{T} > \eta \log \eta$ in practice for the evaluated scenarios, implying that the asymptotic computational complexity of our algorithm was at most $\mathcal{O}(|\mathcal{S}|\mathcal{T})$. Intuitively, since the size of $\mathcal{S}$ is significantly less than the size of the training

[5]Values for the baselines above 90% pruned parameters are not shown because they were not reported in the respective publications of the compared baselines. We chose to show above 90% sparsity for SiPP in Fig. 5-6 to highlight the relatively favorable curvature (i.e., gradual degradation of test accuracy) of the SiPP variants even at extreme sparsities.

set in practice (by Lemma 26), *the computation complexity of any* SiPP *variant from start to finish was at least an order of magnitude less than that of executing a single training epoch.*

5.5.5 Discussion

Our results with SiPP on a diverse set of pruning scenarios in terms of the amount of (re-)training epochs highlight the versatility and robustness of SiPP in performing well across varying tasks. While traditional pruning methods, such as WT and SNIP, perform unreliably when used in the context of alternative pruning pipelines we observe that SiPP serves as a consistent plug-and-play solution to the core pruning method of a pruning pipeline. Among the SiPP variants, we see that SiPPDet tends to perform particularly well for small prune ratios (such as in the case of iterative prune+retrain) while SiPPRand performs the best for extreme prune ratio (such as in the case of random-init + prune + train). SiPPHybrid usually finds a close-to-optimal mixture of strategies and thus provides the most versatility among the SiPP variants, which comes at the cost of increased implementation effort. Notably, unlike other provably pruning methods, our method readily scales to large-scale networks and datasets such as ImageNet without further modifications. Our comparisons to a diverse set of recently-proposed pruning methods on varying pruning scenarios shows that SiPP can outperform baseline approaches across a wide variety of benchmark data sets and architectures.

5.6 Method Pseudocode

In this section, we provide additional details for SiPP as introduced in Sec. 5.3.

5.6.1 Overview

Algorithm 7 provides an extended over view of SiPP. Moreover, in Algorithm 8 we present Sparsify, which is the sub-routine to adaptively prune weights from a

Algorithm 7 SiPP $(\theta, \mathcal{B}, \delta)$

Input: $\theta = (W^1, \ldots, W^L)$: weights of the uncompressed neural network; $\mathcal{B} \in \mathbb{N}$: sampling budget; $\delta \in (0, 1)$: failure probability;
Output: $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$: sparse weights

```
1: $\mathcal{S} \leftarrow$ Uniform sample (without replacement) of $K \log(8\,\eta\,\rho/\delta)$ points from validation set
2: $\{m_i^\ell\}_{i,\ell} \leftarrow$ OptAlloc$(\theta, \mathcal{B}, \mathcal{S})$ $\forall i \in [c^\ell]$, $\forall \ell \in [L]$    ▷ optimally allocate budget $\mathcal{B}$ applying Lemma 28 to evaluate the resulting relative error guarantees
3: for $\ell \in [L]$ do
4:     $\hat{W}^\ell \leftarrow \mathbf{0}$;    ▷ Initialize a null tensor
5:     for $i \in [c^\ell]$ do
6:         $\{s_j\}_j \leftarrow$ EmpiricalSensitivity$(\theta, \mathcal{S}, i, \ell)$ $\forall w_j \in W_i^\ell$    ▷ Compute parameter importance for each weight $w_j$ in the parameter group according to Definitions 25 and 30
7:         $\hat{W}_i^\ell \leftarrow$ Sparsify$(W_i^\ell, m_i^\ell, \{s_j\}_j)$    ▷ prune weights according to SiPPDet, SiPPRand, or SiPPHybrid such that only $m_i^\ell$ weights remain in the parameter group
8: return $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$;
```

parameter group according to either SiPPDet, SiPPRand, or SiPPHybrid.

5.6.2 Details regarding OptAlloc

As mentioned in Section 5.3, OptAlloc proceeds by minimizing the sum of relative error guarantees associated with each parameter group for a given overall weight budget $\mathcal{B}$, i.e.,

$$\min_{m_i^\ell \in \mathbb{N}\ \forall i \in [c^\ell],\ \forall \ell \in [L]} \quad \textstyle\sum_{\ell \in [L],\, i \in [c^\ell]} \varepsilon_i^\ell(m_i^\ell) \quad \text{s. t.} \quad \textstyle\sum_{\ell \in [L],\, i \in [c^\ell]} m_i^\ell \leq \mathcal{B}. \tag{5.5}$$

Here, m_i^ℓ and $\varepsilon_i^\ell(m_i^\ell)$ denote the desired number of weights and the associated theoretical error in parameter group W_i^ℓ after pruning, respectively. We evaluate the theoretical error according to Lemmas 27 and Lemma 28 when pruning with SiPPDet and SiPPHybrid or SiPPRand, respectively. Note that in order to evaluate Lemma 28 we have to first convert m_i^ℓ to the expected number of required samples in order to obtain m_i^ℓ unique samples, which is also shown in Line 4 of Algorithm 8.

Algorithm 8 SPARSIFY$(W_i^\ell, m_i^\ell, \{s_j\}_j)$

Input: W_i^ℓ: parameter group to be pruned; m_i^ℓ: assigned budget; $\{s_j\}_j$: sensitivities associated with weights in parameter group
Output: $\hat{W}_i^\ell$: sparse parameter group

```
 1: $S \leftarrow \sum_{j \in \mathcal{I}_i^\ell} s_j$    ▷ Compute sum of sensitivities
 2: $\tilde{S} \leftarrow \frac{SC}{3} \log(16\eta/\delta)$
 3: $\{q_j\}_j \leftarrow \frac{s_j}{S}$    ▷ Compute sample probabilities for SiPPRand
 4: $N \leftarrow N(m_i^\ell, \{q_j\}_j)$    ▷ Get expected number of required samples to obtain $m_i^\ell$ unique samples
 5: $\mathcal{I}_{det} \leftarrow$ subset of indices from $\mathcal{I}_i^\ell$ corresponding to the largest $m_i^\ell$ sensitivities (c.f. Lemma 27)
 6: $\varepsilon_{rand} \leftarrow \frac{\tilde{S}+\sqrt{\tilde{S}(\tilde{S}+6N)}}{N}$    ▷ c.f. Lemma 28
 7: $\varepsilon_{det} \leftarrow C \sum_{j \in (\mathcal{I} \setminus \mathcal{I}_{det})} s_j$    ▷ c.f. Lemma 27
 8: if ($\varepsilon_{rand} > \varepsilon_{det}$ or always SiPPDet) and not always SiPPRand then
 9:     $\hat{W}_i^\ell \leftarrow$ prune weights from $W_i^\ell$, i.e set to 0, that are not in $\mathcal{I}_{det}$ and keep the rest
10: else
11:     $\{n_j\}_j \sim$ MULTINOMIAL$(\{q_j\}_j, N)$    ▷ Sample $N$ times and return the counts
12:     $\hat{W}_i^\ell \leftarrow$ prune weights such that $\hat{w}_j = \frac{n_j}{N q_j} w_j$ for each weight $\hat{w}_j$, $j \in \mathcal{I}_i^\ell$ in the parameter group
13: return $\hat{W}_i^\ell$
```

5.6.3 Details regarding EMPIRICALSENSITIVITY

We note that the empirical sensitivity (ES) s_j of a weight w_j in the parameter group W_i^ℓ is given by Definition 25, where we define ES as the maximum of the relative parameter importance $g_j(x)$ over a set $\mathcal{S}$ of i.i.d. data points. To account for both negative weights and activations, we utilize the generalized parameter importance as defined in Definition 30 to compute $g_j(x)$ for a particular input x. To ensure that ES holds with probability at least $1 - \delta$ for all patches and parameters simultaneously we have to appropriately choose the size of $\mathcal{S}$, c.f. Line 1 of Algorithm 7 and Section 5.7.4.

5.6.4 Details regarding SPARSIFY

In Algorithm 8 we present the pruning strategy for both SiPPDet and SiPPRand as shown in Line 9 and Lines 11, 12, respectively. Recall that SiPPHybrid adaptively chooses between both strategies according to the associated error guarantees, which get computed in Lines 7 and 6 for SiPPDet and SiPPRand, respectively. We then choose the better strategy accordingly, see Line 8. We can also choose to always prune using SiPPRand or SiPPDet as indicated in Line 8.

5.6.5 Simple SiPP

We can greatly simplify our pruning algorithm if we prune all parameter groups using SiPPDet. To see this consider the solution to OptAlloc when evaluating the relative error according to Lemma 27. Since the relative error for a particular parameter group in this case is the sum over sensitivities that were not included and the objective of OptAlloc is to minimize the sum over all relative errors the optimal solution is to *globally* keep the weights with largest sensitivity. In other words, pruning with SiPPDet only results in global thresholding of weights according to their sensitivity. The resulting procedure is shown in Algorithm 9. Note that this procedure is very reminiscent of simple, global weight thresholding [HMD15, RFC20] but using sensitivity instead of the magnitude of the weights as prune criterion. In contrast to weight thresholding, however, SiPPSimple still exhibits the same theoretical error guarantees as SiPP.

Algorithm 9 SiPPSimple $(\theta, \mathcal{B}, \delta)$

Input: $\theta = (W^1, \dots, W^L)$: weights of the uncompressed neural network; $\mathcal{B} \in \mathbb{N}$: sampling budget; $\delta \in (0,1)$: failure probability;
Output: $\hat{\theta} = (\hat{W}^1, \dots, \hat{W}^L)$: sparse weights

```
1: $\mathcal{S} \leftarrow$ Uniform sample of $K \log(16\,\eta\,\rho/\delta)$ points from validation set
2: Compute sensitivity for all weights in the network using $\mathcal{S}$
3: Prune weights globally by keeping the $\mathcal{B}$ weights with largest sensitivity
4: Return $\hat{\theta} = (\hat{W}^1, \dots, \hat{W}^L)$
```

5.7 Proofs and Technical Details

In this section, we establish the theoretical guarantees of SiPP as presented in Algorithm 7 and state our main compression theorem.

5.7.1 Outline

We begin by considering the sparsification of an arbitrary output patch in an arbitrary parameter group and layer assuming that both the input to the layer and the weights are non-negative. To this end, we first establish the empirical sensitivity (ES) inequality that quantifies the contribution of an individual scalar weight to an output patch (Section 5.7.2 and informal Lemma 19). We then establish the relative error guarantees for each variation of SiPP for an arbitrary output patch (Section 5.7.3 and informal Lemmas 20, 21). Next, we formally generalize the approximation scheme to arbitrary weights and input activations (Section 5.7.4). Finally, we provide our formal network compression bounds by composing together the error guarantees from individual layers and parameter groups. (Section 5.7.5).

5.7.2 Empirical Sensitivity

Recall that an arbitrary parameter group indexed by $i \in [c^\ell]$ in an arbitrary layer $\ell \in [L]$, is denoted by W_i^ℓ and $\mathcal{I}$ denotes its parameter index set. Moreover, let w_j denote some scalar entry of W_i^ℓ for some $j \in \mathcal{I}$. Also as before, $A^{\ell-1}(x)$ denotes the input activation to layer ℓ and $Z_i^\ell(x) = W_i^\ell * A^{\ell-1}(x)$ denotes the output pre-activation of parameter group W_i^ℓ. Finally, recall that $a(\cdot) \in \mathcal{A}_i^\ell$ denotes some patch of $\mathcal{A}_i^\ell$ and that the patch $a(\cdot)$ produces the associated output scalar $z(x)$, i.e., $z(x) = \langle W_i^\ell, a(x) \rangle = \sum_{k \in \mathcal{I}_i^\ell} w_k a_k(x)$, We now proceed with the formal definition of relative parameter importance and empirically sensitivity, which is defined as the maximum relative parameter importance over multiple data points.

Definition 24 (Relative parameter importance). *For a scalar parameter w_j, $j \in \mathcal{I}_i^\ell$,* *of parameter group W_i^ℓ in layer ℓ, its relative importance $g_j(x)$ is given by*

$$g_j(x) = \max_{a(\cdot) \in \mathcal{A}_i^\ell} \frac{w_j \, a_j(x)}{\sum_{k \in \mathcal{I}_i^\ell} w_k \, a_k(x)},$$

where $\mathcal{A}_i^\ell$ denotes the set of patches for parameter group W_i^ℓ.

Definition 25 (Empirical sensitivity)**.** *Let $\mathcal{S}$ be a set of i.i.d. samples from the validation data set. Then, the empirical sensitivity $s_j(x)$ of a scalar parameter w_j, $j \in \mathcal{I}_i^\ell$, of parameter group W_i^ℓ in layer ℓ is given by*

$$s_j(x) = \max_{x \in \mathcal{S}} g_j(x).$$

We note that, for ease of notation, we do not explicitly enumerate ES over i and ℓ for parameter groups and layers, respectively.

To ensure that a *small* batch of points $\mathcal{S}$ suffices for an accurate approximation of parameter importance, we impose the following mild regularity assumption on the Cumulative Distribution Function (CDF) of $g_j(x)$ similar to the assumption of [BLG+19a].

Assumption 4 (Regularity assumption)**.** *There exist universal constants $C, K > 0$ such that for all $j \in \mathcal{I}_i^\ell$, the CDF of the random variable $g_j(x) \in [0, 1]$ for $x \sim \mathcal{D}$, denoted by $F_j(\cdot)$, satisfies*

$$F_j\left({}^1\!/\!_C\right) \leq \exp\left(-{}^1\!/\!_K\right).$$

Traditional distributions such as the Gaussian, Uniform, and Exponential, among others, supported on the interval $[0, 1]$ satisfy Assumption 4 with sufficiently small values of K and K'. In other words, Assumption 4 ensures that there are no outliers of $g_j(x)$ with non-negligible probability that are *not* within a constant multiplicative factor of most other values of $g_j(x)$. Capturing outliers that are within a constant multiplicative factor, on the other hand, can be captured by considering an appropriate scaling factor of ES in the ES inequality (see Lemma 26 below). However, we cannot capture these non-negligible outliers (unless we significantly increase the cardinality of $\mathcal{S}$) when they are not within a constant multiplicative factor.

We now proceed to state the ES inequality as informally stated in Lemma 19. We note that, intuitively, the ES inequality enables us to quantify, i.e., upper-bound, the contribution $w_j a_j(x)$ coming from an individual weight w.h.p. in terms of the output patch $z(x)$ and the sensitivity s_j of the weight.

Lemma 26 (ES inequality)**.** *For $\delta \in (0,1)$, the ES s_j of the scalar parameter w_j, $j \in \mathcal{I}_i^\ell$, of parameter group W_i^ℓ computed with a set $\mathcal{S}$ of i.i.d. data points, $|\mathcal{S}| = K \log(\rho/\delta)$, satisfies*

$$\mathbb{P}_x (w_j a_j(x) \leq C s_j z(x)) \geq 1 - \delta \quad \forall j \in \mathcal{I}_i^\ell,$$

for some input $x \sim \mathcal{D}$ and some fixed input patch $a(\cdot) \in \mathcal{A}_i^\ell$, where C, K are the universal constants of Assumption 4 and $z(x) = \sum_{k \in \mathcal{I}_i^\ell} w_k a_k(x)$.

Proof. We consider a fixed weight w_j from the parameter group and a fixed input patch $a(\cdot)$. Note that

$$\frac{w_j a_j(x)}{z(x)} \leq g_j(x) \tag{5.6}$$

by definition of $g_j(x)$ since the relative parameter importance is the maximum over patches for a specific input x. We now consider the probability that $s_j(\mathcal{S}) = \max_{x' \in \mathcal{S}} g_j(x')$ is not an upper bound for $g_j(x)$ when appropriately scaled for random draws over $\mathcal{S}$, where we explicitly denote the dependency of $s_j(\mathcal{S})$ on $\mathcal{S}$ for the purpose of this proof. By showing this occurs with low probability we can then conclude that s_j is indeed an upper bound for $g_j(x)$ most of the time. Specifically,

$$\begin{aligned}
\mathbb{P}_{\mathcal{S}} (C s_j(\mathcal{S}) \leq g_j(x)) &= \mathbb{P}_{\mathcal{S}} (s_j(\mathcal{S}) \leq {}^{g_j(x)}/_{C}) \\
&\leq \mathbb{P}_{\mathcal{S}} (s_j(\mathcal{S}) \leq {}^{1}/_{C}) && \text{since } g_j(x) \leq 1
\end{aligned}$$

$$
\begin{aligned}
&= \mathbb{P}_{\mathcal{S}}\left(\max_{x' \in \mathcal{S}} g_j(x') \leq {}^1\!/\!_C\right) && \text{by definition of } s_j(\mathcal{S}) \\
&= \left(\mathbb{P}_{x'}\left(g_j(x') \leq {}^1\!/\!_C\right)\right)^{|\mathcal{S}|} && \text{since } \mathcal{S} \text{ is } |\mathcal{S}| \text{ i.i.d. draws from } \mathcal{D} \\
&= F_j\left({}^1\!/\!_C\right)^{|\mathcal{S}|} && \text{since } F_j(\cdot) \text{ is the CDF of } g_j(x') \\
&\leq \exp\left(-{}^{|\mathcal{S}|}\!/\!_K\right) && \text{by Assumption 4} \\
&= \frac{\delta}{\rho} && \text{since } |\mathcal{S}| = K \log\left(\rho/\delta\right) \text{ by definition.}
\end{aligned}
$$

Thus, we can conclude that for a fixed weight w_j and some input $x \sim \mathcal{D}$ its relative contribution $g_j(x)$ is upper bound by its sensitivity s_j. Moreover, the inequality also holds for any weight w_j by the union bound, i.e.,

$$
\begin{aligned}
\mathbb{P}_{\mathcal{S}}\left(\exists j \in \mathcal{I}_i^\ell | C s_j(\mathcal{S}) \leq g_j(x)\right) &\leq \left|\mathcal{I}_i^\ell\right| \mathbb{P}_{\mathcal{S}}\left(C s_j(\mathcal{S}) \leq g_j(x)\right) && \text{by the union bound} \\
&\leq \left|\mathcal{I}_i^\ell\right| \frac{\delta}{\rho} && \text{by the analysis above} \\
&\leq \rho \frac{\delta}{\rho} && \text{since } \rho = \max_{i,\ell} \left|\mathcal{I}_i^\ell\right| \\
&= \delta
\end{aligned}
$$

We thus have with probability at least $1-\delta$ over the construction of s_j that

$$
C s_j \geq g_j(x) \quad \forall j \in \mathcal{I}_i^\ell
$$

and by (5.6) that

$$
\frac{w_j a_j(x)}{z(x)} \leq g_j(x) \leq C s_j,
$$

which concludes the proof since the above inequality holds for any $x \sim \mathcal{D}$. □

5.7.3 Error Guarantees for positive weights and activations

Equipped with Lemma 26 we now proceed to establish the relative error guarantees for the three variants of SiPP. As before, we consider a fixed output patch for a fixed parameter group and we assume that both input activations and the weights are non-negative.

Error Guarantee for SiPPDet

We note that SiPPDet prunes weights from a parameter group by keeping only the weights with largest sensitivity. Let the index set of weights kept be denoted by $\mathcal{I}_{det}$. Below we state the formal error guarantee for a fixed output patch of a parameter group when we only keep the weights indexed by $\mathcal{I}_{det}$. Note that the below error guarantee holds for any index set $\mathcal{I}_{det}$ that we decide to keep. Naturally, however, it makes sense to keep the weights with largest sensitivity as this minimizes the associated relative error.

Lemma 27 (SiPPDet error bound)**.** *For $\delta \in (0,1)$, pruning parameter group W_i^ℓ by keeping only the weights indexed by $\mathcal{I}_{det} \subseteq \mathcal{I}_i^\ell$ generates a pruned parameter group $\hat{W}_i^\ell$ such that for a fixed input patch $a(\cdot) \in \mathcal{A}_i^\ell$ and $x \sim \mathcal{D}$*

$$\mathbb{P}\left(|\hat{z}_{det}(x) - z(x)| \geq \varepsilon_{det} z(x)\right) \leq \delta \quad \textit{with} \quad \varepsilon_{det} = C \textstyle\sum_{j \in \mathcal{I} \setminus \mathcal{I}_{det}} s_j \in (0,1),$$

where

$$z(x) = \langle W_i^\ell, a(x) \rangle = \textstyle\sum_{j \in \mathcal{I}_i^\ell} w_j a_j(x) \quad \textit{and} \quad \hat{z}_{det}(x) = \langle \hat{W}_i^\ell, a(x) \rangle = \textstyle\sum_{j \in \mathcal{I}_{det}} w_j a_j(x)$$

denote the unpruned and approximate output patch, respectively, associated with the input patch $a(\cdot)$. The sensitivities $\{s_j\}_{j \in \mathcal{I}_i^\ell}$ are hereby computed over a set $\mathcal{S}$ of $K \log(\rho/\delta)$ i.i.d. data points drawn from $\mathcal{D}$.

Proof. We proceed by considering the absolute difference $|z(x) - \hat{z}_{det}(x)|$ and note that

$$\begin{aligned}
|z(x) - \hat{z}_{det}(x)| &= \left| \langle W_i^\ell, a(x) \rangle - \langle \hat{W}_i^\ell, a(x) \rangle \right| \\
&= \left| \sum_{j \in \mathcal{I}_i^\ell} w_j a_j(x) - \sum_{j \in \mathcal{I}_{det}} w_j a_j(x) \right|
\end{aligned}$$

$$= \sum_{j \in \mathcal{I}_i^\ell \setminus \mathcal{I}_{det}} w_j a_j(x)$$

Invoking Lemma 26 we know that with probability at least $1-\delta$ each individual weight term in the above sum is upper bound by its sensitivity, i.e.,

$$w_j a_j(x) \leq C s_j z(x) \quad \forall j \in \mathcal{I}_i^\ell.$$

We now bound the error in terms of sensitivity as

$$\begin{aligned} |z(x) - \hat{z}_{det}(x)| &= \sum_{j \in \mathcal{I}_i^\ell \setminus \mathcal{I}_{det}} w_j a_j(x) \\ &\leq \sum_{j \in \mathcal{I}_i^\ell \setminus \mathcal{I}_{det}} C s_j z(x) && \text{using the above inequality} \\ &= \varepsilon_{det} z(x) && \text{by definition of } \varepsilon_{det}. \end{aligned}$$

We conclude by mentioning that above error bound holds with probability at least $1-\delta$ since the associated ES inequalities hold with probability at least $1-\delta$. □

Error Guarantee for SiPPRand

As before, we consider a fixed parameter group W_i^ℓ, which has been assigned a budget of m_i^ℓ unique weights to be kept. Recall that SiPPRand is a sampling procedure that proceeds as follows:

1. Assign probabilities $q_j = {s_j}/{\sum_{k \in \mathcal{I}_i^\ell} s_k}$ for all $j \in \mathcal{I}_i^\ell$.

2. Compute the expected number of samples, N, to obtain m_i^ℓ unique weights from the sampling procedure.

3. Sample weights N times with replacement from W_i^ℓ according to q_j.

4. Reweigh the weights, $\hat{w}_j$, to obtain the approximate weights such that $\hat{w}_j = \frac{n_j}{N q_j} w_j$, where n_j denotes the number of times w_j was sampled.

We note that if a weight has not been sampled, i.e. $n_j = 0$, we can drop it since the resulting weight is 0. We now consider the resulting error bound when sampling N times with replacement.

Lemma 28 (SiPPRand error bound)**.** *For $\delta \in (0, 1)$, pruning parameter group W_i^ℓ by sampling weights $N = N(m_i^\ell)$ times with replacement, such that weight w_j is sampled with probability $q_j = {}^{s_j}/\sum_{k \in \mathcal{I}_i^\ell} s_k$, generates a pruned parameter group $\hat{W}_i^\ell$ such that for a fixed input patch $a(\cdot) \in \mathcal{A}_i^\ell$ and $x \sim \mathcal{D}$*

$$\mathbb{P}\left(|\hat{z}_{rand}(x) - z(x)| \geq \varepsilon_{rand} z(x)\right) \leq \delta \text{ and } \varepsilon_{rand} = \left(\sqrt{\frac{\tilde{S}}{N}\left(\frac{\tilde{S}}{N} + 6\right)} + \frac{\tilde{S}}{N}\right) \in (0, 1),$$

where $\hat{z}_{rand}(x) = \langle \hat{W}_i^\ell, a(x) \rangle$ and $z(x) = \langle W_i^\ell, a(x) \rangle$ are with respect to patch map $a(\cdot)$ as before, $\tilde{S} = \frac{SC}{3} \log(4/\delta)$, and $S = \sum_{j \in \mathcal{I}_i^\ell} s_j$. The sensitivities $\{s_j\}_{j \in \mathcal{I}_i^\ell}$ are hereby computed over a set $\mathcal{S}$ of $K \log(2\,\rho/\delta)$ i.i.d. data points drawn from $\mathcal{D}$.

Proof. Our proof closely follows the proof of Lemma 1 of [BLG+19a]. The sampling procedure of sampling N with replacement is equivalent to sequentially constructing a *multiset* consisting of N samples from $\mathcal{I}_i^\ell$ where each $j \in \mathcal{I}_i^\ell$ is sampled with probability q_j. Now, let $\mathcal{C} = \{c_1, \ldots, c_N\}$ be that multiset of weight indices $\mathcal{I}_i^\ell$ used to construct $\hat{W}_i^\ell$. Let $a(\cdot) \in \mathcal{A}_i^\ell$ be arbitrary and fixed, let $x \sim \mathcal{D}$ be an i.i.d. sample from $\mathcal{D}$, and let

$$\hat{z}_{rand}(x) = \langle \hat{W}_i^\ell, a(x) \rangle = \sum_{j \in \mathcal{C}} \frac{w_j}{N q_j} a_j(x)$$

be the approximate intermediate value corresponding to the sparsified tensor $\hat{W}_i^\ell$ and let

$$z(x) = \sum_{j \in \mathcal{I}_i^\ell} w_j a_j(x)$$

as before. Define N random variables $T_{c_1}, \ldots, T_{c_N}$ such that for all $j \in \mathcal{C}$

$$T_j = \frac{w_j a_j(x)}{N q_j} = \frac{S w_j a_j(x)}{N s_j}. \tag{5.7}$$

For any $j \in \mathcal{C}$, we have for the expectation of T_j:

$$\mathbb{E}\left[T_j\right] = \sum_{k \in \mathcal{I}_i^\ell} \frac{w_k a_k(x)}{N q_k} q_k = \frac{z(x)}{N}.$$

Let $T = \sum_{j \in \mathcal{C}} T_j = \hat{z}_{rand}(x)$ denote our approximation and note that by linearity of expectation,

$$\mathbb{E}\left[T\right] = \sum_{j \in \mathcal{C}} \mathbb{E}\left[T_j\right] = z(x).$$

Thus, $\hat{z}_{rand}(x) = T$ is an unbiased estimator of $z(x)$ for any $x \sim \mathcal{D}$.

For the remainder of the proof we will assume that $z(x) > 0$, since otherwise, $z(x) = 0$ if and only if $T_j = 0$ for all $j \in \mathcal{C}$ almost surely, in which case the lemma follows trivially. We now proceed with the case where $z(x) > 0$ and invoke Lemma 26 (ES inequality) with $\mathcal{S}$ consisting of $K \log(2\,\rho/\delta)$ i.i.d. data points, which implies that

$$w_j a_j(x) \leq C s_j z(x) \quad \forall j \in \mathcal{I}_i^\ell \qquad \text{with probability at least} \qquad 1 - \frac{\delta}{2}. \tag{5.8}$$

Consequently, we can bound the variance of each T_j, $j \in \mathcal{C}$ with probability at least $1 - \delta/2$ as follows

$$\begin{aligned}
\mathrm{Var}(T_j) &\leq \mathbb{E}\left[T_j^2\right] \\
&= \sum_{k \in \mathcal{I}_i^\ell} \frac{(w_k a_k(x))^2}{(N q_k)^2} q_k \\
&= \frac{S}{N^2} \sum_{k \in \mathcal{I}_i^\ell} \frac{w_k a_k(x)}{s_j} w_k a_k(x) \\
&\leq \frac{S}{N^2} C z(x) \sum_{k \in \mathcal{I}_i^\ell} w_k a_k(x) \qquad \text{by the ES inequality as stated in (5.8)} \\
&= \frac{S C z(x)^2}{N^2}.
\end{aligned}$$

Since T is a sum of independent random variables, we obtain

$$\operatorname{Var}(T) = N \operatorname{Var}(T_j) \leq \frac{SCz(x)^2}{N} \tag{5.9}$$

for the overall variance.

Now, for each $j \in \mathcal{C}$ let

$$\tilde{T}_j = T_j - \mathbb{E}\left[T_j\right] = T_j - z(x),$$

and let $\tilde{T} = \sum_{j \in \mathcal{C}} \tilde{T}_j$. Note that by the definition of T_j and the ES inequality (5.8) we have that

$$T_j = \frac{Sw_j a_j(x)}{Ns_j} \leq \frac{SCz(x)}{N}$$

and consequently for the centered random variable $\tilde{T}_j$ that

$$\left|\tilde{T}_j\right| = \left|T_j - \frac{z(x)}{N}\right| \leq \frac{SCz(x)}{N} =: M, \tag{5.10}$$

which holds with probability at least $1 - \delta/2$ for any $x \sim \mathcal{D}$. Also note that $\operatorname{Var}(\tilde{T}) = \operatorname{Var}(T)$.

Now conditioned on the ES inequality (5.8) holding, applying Bernstein's inequality to both $\tilde{T}$ and $-\tilde{T}$ we have by symmetry and the union bound,

$$\begin{aligned}
\mathbb{P}\left(\left|\tilde{T}\right| \geq \varepsilon_{rand} z(x)\right) &= \mathbb{P}\left(|T - z(x)| \geq \varepsilon_{rand} z(x)\right) \\
&\leq 2\exp\left(-\frac{\varepsilon_{rand}^2 z(x)^2}{2\operatorname{Var}(T) + \frac{2\varepsilon_{rand} z(x) M}{3}}\right) && \text{by Bernstein's inequality} \\
&\leq 2\exp\left(-\frac{\varepsilon_{rand}^2 z(x)^2}{\frac{2SCz(x)^2}{N} + \frac{2SC\varepsilon_{rand} z(x)^2}{3N}}\right) && \text{by (5.9) and (5.10)} \\
&= 2\exp\left(-\frac{3\varepsilon_{rand}^2 N}{SC(6 + 2\varepsilon_{rand})}\right) \\
&\leq \frac{\delta}{2} && \text{by our choice of } \varepsilon_{rand}
\end{aligned}$$

Note that the undesired event $|T - z(x)| \geq \varepsilon_{rand} z(x) = |\hat{z}_{rand}(x) - z(x)| \geq \varepsilon_{rand} z(x)$

occurs with probability at most $\delta/2$, which was conditioned on the ES inequality holding, which occurs with probability at least $1 - \delta/2$. Thus by the union bound, the overall failure probability is at most δ, which concludes the proof. $\square$

Error Guarantee for SiPPHybrid

We note that the error guarantee for SiPPHybrid follow straightforward from the error guarantees for SiPPDet and SiPPRand as stated in Lemma 27 and 28, respectively, since SiPPHybrid chooses the strategy among those two for which the associated error guarantee is lower. We can therefore state the error guarantee as follows.

Lemma 29 (SiPPHybrid error bound)**.** *In the context of Lemmas 27 and 28, for $\delta \in (0, 1)$ SiPPHybrid generates a pruned parameter group $\hat{W}_i^\ell$ such that for a fixed input patch $a(\cdot) \in \mathcal{A}_i^\ell$, output patch $z(x)$, and $x \sim D$*

$$\mathbb{P}\left(|\hat{z}_{hybrid}(x) - z(x)| \geq \varepsilon_{hybrid} z(x)\right) \leq \delta \quad \textit{with} \quad \varepsilon_{hybrid} = \min\left\{\varepsilon_{det}, \varepsilon_{rand}\right\} \in (0, 1),$$

where $z_{hybrid}(x)$ is the associated approximate output patch.

5.7.4 Generalization to all weights and activations

In this section, we generalize our analysis from the previous section to include all weights and activations. We also adapt the resulting error guarantees to simultaneously hold for all patches of all parameter groups within a layer instead of a fixed patch.

We handle the general case by splitting both the input activations and the weights into their respective positive and negative parts representing the four quadrants, i.e.,

$$\begin{aligned} z^{++}(x) &= \langle W_i^{\ell,+}, a(x)^+ \rangle \qquad & z^{+-}(x) &= \langle W_i^{\ell,+}, a(x)^- \rangle \\ z^{-+}(x) &= \langle W_i^{\ell,-}, a(x)^+ \rangle \qquad & z^{--}(x) &= \langle W_i^{\ell,+}, a(x)^- \rangle, \end{aligned}$$

where

$$W_i^\ell = W_i^{\ell,+} - W_i^{\ell,-}, \quad W_i^{\ell,+}, W_i^{\ell,-} \geq 0,$$
$$a(x) = a(x)^+ - a(x)^-, \quad a(x)^+, a(x)^- \geq 0.$$

First, consider negative activations for a non-negative parameter group. Specifically, when computing sensitivities over some set $\mathcal{S}$ we split the input activations into their respective positive and negative part, and take an additional maximum over both parts. Henceforth the ES inequality 26 can be applied to the positive and negative part of $a(x)$ at the same time. Similarly, we can split the parameter group into its positive and negative part when computing sensitivity such that the ES inequality 26 holds for both parts of the parameter group as well.

More formally, the generalized relative parameter importance $g_j(x)$ for some parameter w_j of parameter group W_i^ℓ can be defined as follows.

Definition 30 (Generalized relative parameter importance)**.** *For a scalar parameter $w_j = w_j^+ - w_j^-$, w_j^+, $w_j^- \geq 0$, $j \in \mathcal{I}_i^\ell$, of parameter group W_i^ℓ in layer ℓ, its generalized relative importance $g_j(x)$ is given by the maximum over its quadrant-wise relative importances, i.e.,*

$$g_j(x) = \max\{g_j^{++}(x), g_j^{+-}(x), g_j^{-+}(x), g_j^{--}(x)\},$$

where
$$g_j^{++}(x) = \max_{a(\cdot)\in\mathcal{A}_i^\ell} \frac{w_j^+ a_j^+(x)}{\sum_{k\in\mathcal{I}_i^\ell} w_k^+ a_k^+(x)}, \text{ and so forth,}$$

and where $\mathcal{A}_i^\ell$ denotes the set of patches for parameter group W_i^ℓ and $\mathcal{A}_i^\ell \ni a(\cdot) = a^+(\cdot) - a^-(\cdot)$, $a^+(\cdot)$, $a^-(\cdot) \geq 0$.

The definition of generalized ES does not change compared to Definition 25 and henceforth we do not re-state it explicitly. We proceed by re-deriving the ES inequality for the generalized parameter importance and any patch of the parameter group.

Lemma 31 (Generalized ES inequality)**.** *For $\delta \in (0,1)$, the ES s_j of the scalar parameter w_j, $j \in \mathcal{I}_i^\ell$, of parameter group W_i^ℓ computed with a set $\mathcal{S}$ of i.i.d. data points, $|\mathcal{S}| = K \log(\rho/\delta)$, satisfies for each quadrant*

$$\mathbb{P}_x\left(w_j^+ a_j^+(x) \leq C s_j z^{++}(x)\right) \geq 1 - \delta \quad \forall j \in \mathcal{I}_i^\ell, \text{ and so forth,}$$

for some input $x \sim \mathcal{D}$ and some fixed input patch $a(\cdot) \in \mathcal{A}_i^\ell$, where C, K are the universal constants of Assumption 4 and $z^{++}(x) = \sum_{k \in \mathcal{I}_i^\ell} w_k^+ a_k^+(x)$, and so forth, denotes the quadrant-wise output patch.

Proof. The proof follows the steps of Lemma 26 with the exception of Equation (5.6). To adapt it to the general case note that

$$\frac{w_j^+ a_j^+(x)}{z^{++}(x)} \leq g_j^{++}(x) \leq g_j(x),$$

and so forth, for each quadrant. □

Consequently, we can re-derive Lemmas 27-29 such that they hold for each quadrant of a fixed patch. The derivations are analogues to the derivations in Section 5.7.3. Finally, we adapt our guarantees to hold quadrant-wise for all patches of all parameter groups and layers simultaneously. We note that we can achieve this by appropriately adjusting the failure probability for Lemmas 27-29 such that, by the union bound, the overall failure probability is bounded δ. Specifically, we can invoke Lemmas 27-29 with $\delta' = {}^{\delta}/_{4\eta}$ such that

$$\tilde{S} = \frac{SC}{3}\log(16\eta/\delta) \qquad \text{and} \qquad |\mathcal{S}| = K \log(8\eta\,\rho/\delta),$$

where η denotes the number of total patches across all layers and parameter groups. The rest of the Lemmas remains unchanged. Therefore, we have that for all quadrant-wise patches our error guarantees hold. We utilize our patch-wise bounds as outlined in Section 5.6 to optimally allocate our budget across layers to minimize the relative error within each quadrant of the parameter groups and prune each parameter group according to the budget and the desired variant of SiPP.

5.7.5 Network compression bounds

Up to this point we have established patch-wise and quadrant-wise error guarantees for the network, which suffices to prune the network according to Algorithm 7. However, we can also leverage our theoretical guarantees to establish network-wide compression bounds of the form

$$\mathbb{P}_{\hat{\theta},x}\left(\|f_{\hat{\theta}}(x) - f_\theta(x)\| \leq \varepsilon \, \|f_\theta(x)\|\right) \geq 1 - \delta,$$

for given $\varepsilon, \delta \in (0,1)$ as described in Problem 2.

We will restrict ourselves to analyzing the general case for SiPPDet but we note that each step can be applied analogously for SiPPRand and SiPPHybrid. We begin by generalizing Lemma 27 to establish norm-based bounds for each quadrant of the pre-activation. To this end, let

$$Z^{\ell++}(x) = W^{\ell+} * A^{\ell-1,+}(x), \quad \hat{Z}^{\ell++}(x) = \hat{W}^{\ell+} * A^{\ell-1,+}(x), \quad \text{and so forth}$$

denote the unpruned and approximate pre-activation quadrants, respectively. Moreover, let S_i^ℓ denote the sum of ES for parameter group W_i^ℓ as before and let $S_i^\ell(N_i^\ell)$ denote the sum over the N_i^ℓ largest ES for parameter group W_i^ℓ.

Corollary 32. *For $\delta \in (0,1)$, pruning layer ℓ according to* SiPPDet *generates a pruned weight tensor $\hat{W}^\ell$ such that for a fixed quadrant and $x \sim \mathcal{D}$*

$$\mathbb{P}\left(\left\|\hat{Z}^{\ell++}(x) - \hat{Z}^{\ell++}(x)\right\| \geq \varepsilon^\ell \left\|Z^{\ell++}(x)\right\|\right) \leq \delta \quad \text{with} \quad \varepsilon^\ell = \max_{i \in c^\ell} C\left(S_i^\ell - S_i^\ell(N_i^\ell)\right),$$

where N_i^ℓ denotes the number of samples allocated to parameter group W_i^ℓ. The ESs are hereby computed over a set $\mathcal{S}$ of $K \log\left(\eta^\ell \rho/\delta\right)$ i.i.d. data points drawn from $\mathcal{D}$.

Proof. Let $Z_i^{\ell++}(x)$ denote the pre-activation quadrant associated with parameter group W_i^ℓ. Invoking Lemma 27 with a set $\mathcal{S}$ of $K \log\left(\eta^\ell \rho/\delta\right)$ i.i.d. data points drawn from $\mathcal{D}$ and N_i^ℓ samples for the respective parameter group implies that any associated patch, i.e. entry, of $Z_i^{\ell++}(x)$ is approximated with relative error at most $\varepsilon_i^\ell = C\left(S_i^\ell - S_i^\ell(N_i^\ell)\right)$ with probability at least $1 - {}^{\delta}/_{\eta^\ell}$. Consequently, $\left\|Z_i^{\ell++}(x)\right\|$ is also preserved with relative error ε_i^ℓ. Thus we have w.h.p. that

$$\begin{aligned}
\left\|Z^{\ell++}(x) - \hat{Z}^{\ell++}(x)\right\|^2 &= \sum_{i\in[c^\ell]} \left\|Z_i^{\ell++}(x) - \hat{Z}_i^{\ell++}(x)\right\|^2 \\
&\leq \sum_{i\in[c^\ell]} (\varepsilon_i^\ell)^2 \left\|Z_i^{\ell++}(x)\right\|^2 \\
&\leq (\varepsilon^\ell)^2 \sum_{i\in[c^\ell]} \left\|Z_i^{\ell++}(x)\right\|^2 && \text{by definition of } \varepsilon^\ell \\
&= (\varepsilon^\ell)^2 \left\|Z^{\ell++}(x)\right\|^2
\end{aligned}$$

Taking a union bound over all η^ℓ patches in the pre-activation $Z^\ell(x)$ concludes the proof. □

We note that Corollary 32 is stated for $Z^{\ell++}(x)$ but naturally extends to the other quadrants as well.

As a next step, we establish guarantees to approximate $Z^\ell(x)$ by leveraging the guarantees for each quadrant. To this end, note that $Z^\ell(x) = Z^{\ell++}(x) - Z^{\ell+-}(x) - Z^{\ell-+}(x) + Z^{\ell--}(x)$. Further, let Δ^ℓ denote the "sign complexity" of approximating the overall pre-activation, which is defined as

Definition 33 (Sign complexity)**.** *For layer ℓ, its sign complexity Δ^ℓ is given by*

$$\Delta^\ell = \max_{x\in\mathcal{S}} \frac{\left\|Z^{\ell++}(x)\right\| + \left\|Z^{\ell+-}(x)\right\| + \left\|Z^{\ell-+}(x)\right\| + \left\|Z^{\ell--}(x)\right\|}{\left\|Z^\ell(x)\right\|}$$

where $\mathcal{S}$ denotes a set of i.i.d. data points drawn from $\mathcal{D}$.

Intuitively, Δ^ℓ captures the additional complexity of approximating the layer when considering the actual signs of the quadrants as opposed to treating them separately. We can now state the error guarantees for SiPP in context of Corollary 32 for the overall pre-activation.

Lemma 34 (Layer error bound)**.** *For given $\delta \in (0,1)$ and sample budget N_i^ℓ for each parameter group, invoking* SiPPDet *to prune W^ℓ generates a pruned weight tensor $\hat{W}^\ell$ such that for $x \sim \mathcal{D}$*

$$\mathbb{P}\left(\left\|\hat{Z}^\ell(x) - \hat{Z}^\ell(x)\right\| \geq \varepsilon^\ell \Delta^\ell \left\|Z^\ell(x)\right\|\right) \leq \delta \quad \textit{with} \quad \varepsilon^\ell = \max_{i \in c^\ell} C\left(S_i^\ell - S_i^\ell(N_i^\ell)\right),$$

where N_i^ℓ denotes the number of samples allocated to parameter group W_i^ℓ. The ESs are hereby computed over a set $\mathcal{S}$ of $K \log\left(5\eta^\ell \rho/\delta\right)$ i.i.d. data points drawn from $\mathcal{D}$.

Proof. Consider invoking Corollary 32 with a set $\mathcal{S}$ of $K \log\left(5\eta^\ell \rho/\delta\right)$ i.i.d. data points drawn from $\mathcal{D}$. Then for each quadrant we have w.h.p. that

$$\left\|\hat{Z}^{\ell++}(x) - \hat{Z}^{\ell++}(x)\right\| \leq \varepsilon^\ell \left\|\hat{Z}^{\ell++}(x)\right\|, \text{ and so forth,}$$

for an appropriate notion of high probability specified subsequently. Note that

$$Z^\ell(x) = Z^{\ell++}(x) - Z^{\ell+-}(x) - Z^{\ell-+}(x) + Z^{\ell--}(x)$$

and so w.h.p. we have that

$$\begin{aligned}
\left\|\hat{Z}^\ell(x) - \hat{Z}^\ell(x)\right\| &= \Big\|\hat{Z}^{\ell++}(x) - \hat{Z}^{\ell++}(x) - \hat{Z}^{\ell+-}(x) + \hat{Z}^{\ell+-}(x) \\
&\qquad - \hat{Z}^{\ell-+}(x) + \hat{Z}^{\ell-+}(x) + \hat{Z}^{\ell--}(x) - \hat{Z}^{\ell--}(x)\Big\| \\
&\leq \left\|\hat{Z}^{\ell++}(x) - \hat{Z}^{\ell++}(x)\right\| + \left\|\hat{Z}^{\ell+-}(x) - \hat{Z}^{\ell+-}(x)\right\| \\
&\qquad + \left\|\hat{Z}^{\ell-+}(x) - \hat{Z}^{\ell-+}(x)\right\| + \left\|\hat{Z}^{\ell--}(x) - \hat{Z}^{\ell--}(x)\right\| \\
&\leq \varepsilon^\ell \left(\left\|\hat{Z}^{\ell++}(x)\right\| + \left\|\hat{Z}^{\ell+-}(x)\right\| + \left\|\hat{Z}^{\ell-+}(x)\right\| + \left\|\hat{Z}^{\ell--}(x)\right\|\right) \\
&= \varepsilon^\ell \frac{\left\|Z^{\ell++}(x)\right\| + \left\|Z^{\ell+-}(x)\right\| + \left\|Z^{\ell-+}(x)\right\| + \left\|Z^{\ell--}(x)\right\|}{\left\|Z^\ell(x)\right\|} \left\|Z^\ell(x)\right\|
\end{aligned}$$

$$\leq \varepsilon^\ell \Delta^\ell \left\| Z^\ell(x) \right\|,$$

where the last step followed from our definition of Δ^ℓ. By imposing a regularity assumption on Δ^ℓ similar to that of ES, we can show that Δ^ℓ is an upper bound for any $x \sim \mathcal{D}$ w.h.p. following the proof of the ES inequality (Lemma 26).

To specify the appropriate notion of high probability, we consider the individual failure cases and apply the union bound. In particular, for our choice of for the size of $\mathcal{S}$, we have that for a particular quadrant the approximation fails with probability at most $\delta/5$. Thus across all quadrants we have a overall failure probability of at most $4\delta/5$. Finally, we consider the event that Δ^ℓ does not upper bound the hardness for some input $x \sim \mathcal{D}$, which occurs with probability at most $\delta/5$ by our choice for the size of $\mathcal{S}$. Henceforth, our overall failure probability is at most δ, again by the union bound, which concludes the proof. $\square$

We now consider the effect of pruning multiple layers at the same time and analyze the final resulting error in the output. To this end, consider the activation $\phi^\ell(\cdot)$ for which we assume the following.

Assumption 5. *For layer $\ell \in [L]$, the activation function, denoted by $\phi^\ell(\cdot)$, is Lipschitz continuous with Lipschitz constant K^ℓ.*

Without loss of generality, we will further assume that the activation function is 1-Lipschitz, which is the case, e.g., for ReLU and Softmax, to avoid introducing additional notation. We now state a lemma pertaining to the error resulting from pruning multiple layers simultaneously, which will provide the basis for establishing error bounds across the entire network.

Lemma 35 (Error propagation). *Let $\hat{A}^\ell(x)$, $\ell \leq L$, denote the activation of layer ℓ when we have pruned layers $1, \dots, \ell$ according to Lemma 34. Then the overall* *approximation in layer ℓ is bounded by*

$$\left\|\hat{A}^\ell(x) - A^\ell(x)\right\| \leq \sum_{k=1}^{\ell} \left(\prod_{k'=k+1}^{\ell} \left\|W^{k'}\right\|_F \right) \varepsilon^k \Delta^k \left\|Z^k(x)\right\|$$

with probability at least $1-\delta$. The ESs are hereby computed over a set $\mathcal{S}$ of $K \log\left(5 \sum_{k \in [\ell]} \eta^k \rho / \delta\right)$ i.i.d. data points drawn from $\mathcal{D}$.

Proof. We prove the above statement by induction. For layer $\ell = 1$, we have that

$$\begin{aligned}
\left\|\hat{A}^1(x) - A^1(x)\right\| &= \left\|\phi^1(\hat{W}^1 * \hat{A}^0(x)) - \phi^1(\hat{W}^1 * A^0(x))\right\| \\
&\leq \left\|\hat{W}^1 * \hat{A}^0(x) - \hat{W}^1 * A^0(x)\right\| && \text{since the } \phi^1(\cdot) \text{ is 1-Lipschitz} \\
&= \left\|\hat{W}^1 * A^0(x) - \hat{W}^1 * A^0(x)\right\| && \text{since } \hat{A}^0(x) = A^0(x) = x \\
&= \left\|\hat{Z}^1(x) - Z^1(x)\right\| && \text{by definition of } \hat{Z}^1(x),\, Z^1(x) \\
&\leq \varepsilon^1 \Delta^1 \left\|Z^1(x)\right\| && \text{by Lemma 34,}
\end{aligned}$$

which proves that the base case holds.

We now proceed with the inductive step. Assuming the inequality is true for layer ℓ, we have for layer $\ell + 1$ that

$$\begin{aligned}
\left\|\hat{A}^{\ell+1}(x) - A^{\ell+1}(x)\right\| &= \\
&= \left\|\phi^{\ell+1}(\hat{W}^{\ell+1} * \hat{A}^\ell(x)) - \phi^{\ell+1}(W^{\ell+1} * A^\ell(x))\right\| \\
&\leq \left\|\hat{W}^{\ell+1} * \hat{A}^\ell(x) - W^{\ell+1} * A^\ell(x)\right\| \quad \text{since } \phi^{\ell+1}(\cdot) \text{ is 1-Lipschitz} \\
&= \left\|\hat{W}^{\ell+1} * \hat{A}^\ell(x) - \hat{W}^{\ell+1} * A^\ell(x) + \hat{W}^{\ell+1} * A^\ell(x) - W^{\ell+1} * A^\ell(x)\right\| \\
&\leq \left\|\hat{W}^{\ell+1} * (\hat{A}^\ell(x) - A^\ell(x))\right\| + \left\|(\hat{W}^{\ell+1} - W^{\ell+1}) * A^\ell(x)\right\|
\end{aligned}$$

Note that we can bound the first term by

$$\left\|\hat{W}^{\ell+1} * (\hat{A}^\ell(x) - A^\ell(x))\right\| \leq \left\|\hat{W}^{\ell+1}\right\|_{op} \left\|\hat{A}^\ell(x) - A^\ell(x)\right\|$$

$$\leq \left\| \hat{W}^{\ell+1} \right\|_F \left\| \hat{A}^\ell(x) - A^\ell(x) \right\|$$
$$\leq \left\| W^{\ell+1} \right\|_F \left\| \hat{A}^\ell(x) - A^\ell(x) \right\|$$

where the last inequality followed from the fact that $\hat{W}^{\ell+1}$ is a subset of $W^{\ell+1}$ and $\|\cdot\|_{op}$ and $\|\cdot\|_F$ denote the ℓ_2-induced operator norm and Frobenius norm, respectively. The second term is bounded by Lemma 34, i.e.,

$$\left\| (\hat{W}^{\ell+1} - W^{\ell+1}) * A^\ell(x) \right\| = \left\| \hat{Z}^{\ell+1}(x) - Z^{\ell+1}(x) \right\| \leq \varepsilon^{\ell+1} \Delta^{\ell+1} \left\| Z^{\ell+1}(x) \right\|.$$

Putting both terms back together we have that

$$\begin{aligned}
&\left\| \hat{A}^{\ell+1}(x) - A^{\ell+1}(x) \right\| \\
&\quad \leq \left\| W^{\ell+1} \right\|_F \left\| \hat{A}^\ell(x) - A^\ell(x) \right\| + \varepsilon^{\ell+1} \Delta^{\ell+1} \left\| Z^{\ell+1}(x) \right\| \\
&\quad \leq \left\| W^{\ell+1} \right\|_F \left(\sum_{k=1}^{\ell} \left(\prod_{k'=k+1}^{\ell} \left\| W^{k'} \right\|_F \right) \varepsilon^k \Delta^k \left\| Z^k(x) \right\| \right) + \varepsilon^{\ell+1} \Delta^{\ell+1} \left\| Z^{\ell+1}(x) \right\| \\
&\quad = \sum_{k=1}^{\ell} \left(\prod_{k'=k+1}^{\ell+1} \left\| W^{k'} \right\|_F \right) \varepsilon^k \Delta^k \left\| Z^k(x) \right\| + \varepsilon^{\ell+1} \Delta^{\ell+1} \left\| Z^{\ell+1}(x) \right\| \\
&\quad = \sum_{k=1}^{\ell+1} \left(\prod_{k'=k+1}^{\ell+1} \left\| W^{k'} \right\|_F \right) \varepsilon^k \Delta^k \left\| Z^k(x) \right\|,
\end{aligned}$$

where the second inequality followed from our induction hypothesis. Finally, we note that, by our choice for the size of $\mathcal{S}$ and the union bound, the overall failure probability is bounded above by δ.

$\square$

From the analysis the term $\prod_{k'=k+1}^{\ell} \left\| W^{k'} \right\|_F$ arises, which is an upper bound for the Lipschitz constant of the network starting from layer $k+1$. Moreover, the coefficient of the propagated error is closely related to the condition number between layer ℓ and the network's output. To this end, consider the following upper bound on the condition number.

Definition 36 (Layer condition number)**.** *For layer ℓ, the condition number from the pre-activation of layer ℓ to the output of the network (activation of layer L) is given by*

$$\kappa^{\ell} = \max_{x \in S} \left(\prod_{k=\ell+1}^{L} \left\| W^{\ell} \right\|_F \right) \frac{\left\| Z^{\ell}(x) \right\|}{\left\| A^{L}(x) \right\|},$$

where $\mathcal{S}$ denotes a set of i.i.d. data points drawn from $\mathcal{D}$.

Interestingly, a variant of the layer condition number (roughly, the product of layer norms, i.e., $\prod_{k=1}^{L} \left\| W^{\ell} \right\|_F$) appears in recent work on generalization bounds for neural networks [JNM+19]. Moreover, under the link-normalized assumption or the commonly imposed assumption that the norm of the weights satisfy $\left\| W^{\ell} \right\|_F \leq 1$ [AAR20, AANR17], the condition number is non-increasing with the number of layers L. As mentioned in the concluding remarks (Sec. 5.9), obtaining tighter bounds that reflect the fact that the injected noise attenuates (see Fig. 1 of [AGNZ18]) over the network's layers is an avenue for future work.

To see that κ^{ℓ} is indeed an upper bound on the condition number we note that the condition number is defined as the maximum relative change in the output over the maximum relative change in the input, i.e.,

$$\max_{x,x' \in \mathcal{S}} \frac{\frac{\left\| A^{L}(x) - A^{L}(x') \right\|}{\left\| A^{L}(x) \right\|}}{\frac{\left\| Z^{\ell}(x) - Z^{\ell}(x') \right\|}{\left\| Z^{\ell}(x) \right\|}} = \max_{x,x' \in \mathcal{S}} \frac{\left\| A^{L}(x) - A^{L}(x') \right\|}{\left\| Z^{\ell}(x) - Z^{\ell}(x') \right\|} \frac{\left\| Z^{\ell}(x) \right\|}{\left\| A^{L}(x) \right\|}.$$

The first term can be upper bounded as

$$\begin{aligned} \frac{\left\| A^{L}(x) - A^{L}(x') \right\|}{\left\| Z^{\ell}(x) - Z^{\ell}(x') \right\|} &\leq \frac{\left\| Z^{L}(x) - Z^{L}(x') \right\|}{\left\| Z^{\ell}(x) - Z^{\ell}(x') \right\|} \\ &= \frac{\left\| W^{L} * (A^{L-1}(x) - A^{L-1}(x')) \right\|}{\left\| Z^{\ell}(x) - Z^{\ell}(x') \right\|} \\ &\leq \left\| W^{L} \right\|_F \frac{\left\| (A^{L-1}(x) - A^{L-1}(x')) \right\|}{\left\| Z^{\ell}(x) - Z^{\ell}(x') \right\|} \\ &\leq \dots \end{aligned}$$

$$
\begin{aligned}
&\leq \prod_{k=\ell+1}^{L} \left\|W^k\right\|_F \frac{\left\|A^\ell(x) - A^\ell(x')\right\|}{\left\|Z^\ell(x) - Z^\ell(x')\right\|} \\
&\leq \prod_{k=\ell+1}^{L} \left\|W^k\right\|_F ,
\end{aligned}
$$

which plugged back in above yields the definition of the layer condition number κ^ℓ.

Equipped with Lemma 35 and Definition 36 we are now ready to state our main compression bound over the entire network.

Theorem 37 (Network compression bound)**.** *For given $\delta \in (0,1)$, a set of parameters $\theta = (W^1, \ldots, W^L)$, and a sample budget $\mathcal{B}$* SiPP *(Algorithm 7) generates a set of compressed parameters $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$ such that $\|\hat{\theta}\|_0 \leq \mathcal{B}$, $\|W_i^\ell\|_0 \leq N_i^\ell$, $\forall i \in [c^\ell]$, $\ell \in [L]$,*

$$
\mathbb{P}_{\hat{\theta},x}\left(\|f_{\hat{\theta}}(x) - f_\theta(x)\| \leq \varepsilon \|f_\theta(x)\|\right) \geq 1 - \delta \quad \textit{and} \quad \varepsilon = C \sum_{\ell=1}^{L} \kappa^\ell \Delta^\ell \max_{i \in [c^\ell]} \left(S_i^\ell - S_i^\ell(N_i^\ell)\right),
$$

where S_i^ℓ is the sum of sensitivities for parameter group W_i^ℓ computed over a set $\mathcal{S}$ of $K \log(6\eta\,\rho/\delta)$ i.i.d. data points.

Proof. Invoking Lemma 35 for $\ell = L$ implies with high probability that

$$
\begin{aligned}
\left\|\hat{A}^L(x) - A^L(x)\right\| &\leq \sum_{\ell=1}^{L} \left(\prod_{k=\ell+1}^{L} \left\|W^k\right\|_F\right) \varepsilon^\ell \Delta^\ell \left\|Z^\ell(x)\right\| \\
&= \sum_{\ell=1}^{L} \left(\prod_{k=\ell+1}^{L} \left\|W^k\right\|_F\right) \frac{\left\|Z^\ell(x)\right\|}{\left\|A^L(x)\right\|} \Delta^\ell \varepsilon^\ell \left\|A^L(x)\right\| \\
&\leq \sum_{\ell=1}^{L} \kappa^\ell \Delta^\ell \varepsilon^\ell \left\|A^L(x)\right\| \\
&= \varepsilon \left\|A^L(x)\right\|,
\end{aligned}
$$

where the last inequality followed from our definition of the layer condition number κ^ℓ. Moreover, following the analysis of Lemma 26 we can establish that κ^ℓ is an upper bound for any $x \sim \mathcal{D}$ with high probability. Finally, we note that the overall failure probability is bounded by δ by our choice for the size of $\mathcal{S}$ and by a union bound over the failure probabilities of Lemma 35 and of κ^ℓ not being an upper bound for some $x \sim \mathcal{D}$. $\square$

5.8 Experimental details

5.8.1 Setup and Hyperparameters

All hyperparameters for training, retraining, and pruning are outlined in Table 5.2. For training CIFAR10 networks we used the training hyperparameters outlined in the respective original papers, i.e., as described by [HZRS16], [SZ14b], [HLVDMW17], and [ZK16] for ResNets, VGGs, DenseNets, and WideResNets, respectively. For retraining, we did not change the hyperparameters and repurposed the training hyperparameters. We added a warmup period in the beginning where we linearly scale up the learning rate from 0 to the nominal learning rate. Iterative pruning is conducted by repeatedly removing the same ratio of parameters (denoted by α in Table 5.2). The prune parameter δ describes the failure probability of SiPP, which also determines the required size of $\mathcal{S}$ (usually around a few hundred). We note no other additional hyperparameters are required to run SiPP.

For ImageNet, we show experimental results for a ResNet18 and a ResNet101. As in the case of the CIFAR10 networks, we re-purpose the same training hyperparameters as indicated in the original papers of the competing methods. We also use the same hyperparameters for retraining. The hyperparameters are summarized in Table 5.3.

5.8.2 Additional results for CIFAR10 (iterative prune+retrain)

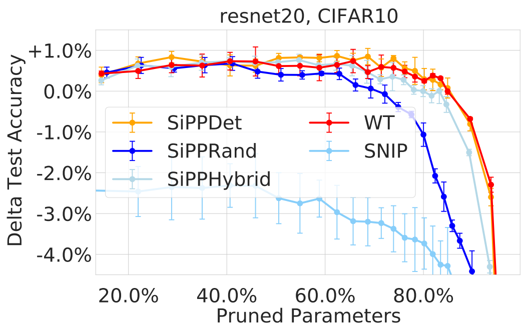
In Figure 5-7, we compare the performance of the three variations of our algorithm when using iterative prune+retrain as outlined in Section 5.5.2. Note that the performance for all of them is very similar, henceforth we choose SiPPDet for its simplicity when comparing to other methods for this expensive iterative prune+retrain pipeline.

		VGG16	Resnet20/56/110	DenseNet22	WRN-16-8
Train	test error	7.19	8.6/7.19/6.43	10.10	4.81
	loss	cross-entropy	cross-entropy	cross-entropy	cross-entropy
	optimizer	SGD	SGD	SGD	SGD
	epochs	300	182	300	200
	warm-up	5	5	5	5
	batch size	256	128	64	128
	LR	0.05	0.1	0.1	0.1
	LR decay	0.5@{30, ...}	0.1@{91, 136}	0.1@{150, 225}	0.2@{60, ...}
	momentum	0.9	0.9	0.9	0.9
	Nesterov	No	No	Yes	Yes
	weight decay	5.0e-4	1.0e-4	1.0e-4	5.0e-4
Prune	δ	1.0e-16	1.0e-16	1.0e-16	1.0e-16
	α	0.85	0.85	0.85	0.85

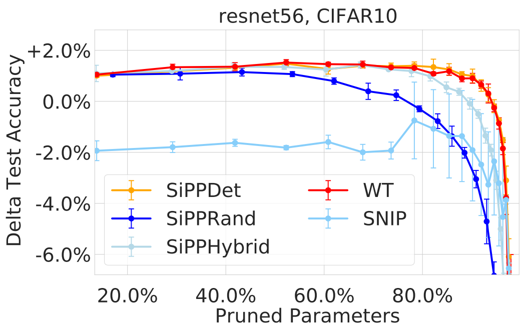
Table 5.2: We report the hyperparameters used during training, pruning, and retraining for various convolutional architectures on CIFAR-10. LR hereby denotes the learning rate and LR decay denotes the learning rate decay. During retraining we used the same hyperparameters. $\{30, \ldots\}$ denotes that the learning rate is decayed every 30 epochs.

5.8.3 Additional results for ImageNet (iterative prune+retrain)

Finally, we show additional results for a ResNet18 and ResNet101 trained on ImageNet, see Figure 5-8 for Top-5 accuracy after retraining. From the results, we can conclude that SiPP scales well to larger architectures and datasets, such as ImageNet, and can perform en par with existing state-of-the-art methods.



(a) Resnet20



(b) Resnet56

Figure 5-7: The delta in test accuracy to the uncompressed network for the generated pruned models trained on CIFAR10 for various target prune ratios. The networks were pruned using the **iterative prune+retrain** pipeline.

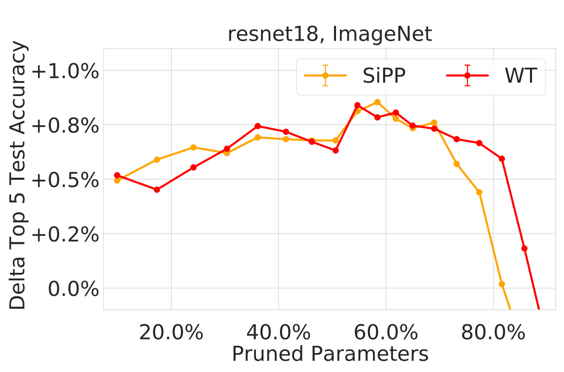
		ResNet18/101
Train	top-1 test error	30.26/22.63
	top-5 test error	10.93/6.45
	loss	cross-entropy
	optimizer	SGD
	epochs	90
	warm-up	5
	batch size	256
	LR	0.1
	LR decay	0.1@{30, 60, 80}
	momentum	0.9
	Nesterov	No
	weight decay	1.0e-4
Prune	δ	1.0e-16
	α	0.90

Table 5.3: We report the hyperparameters used during training, pruning, and retraining for various convolutional architectures on ImageNet. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs.

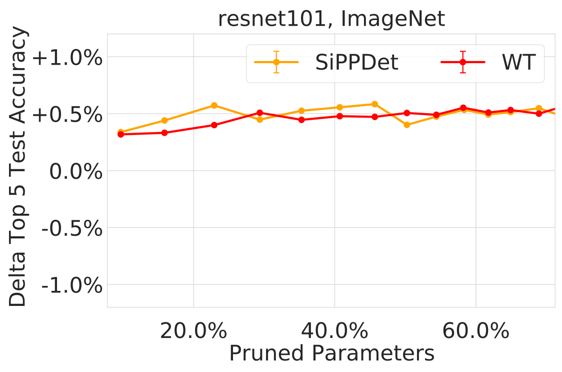
5.8.4 Sensitivity to the validation set size

In this subsection, we evaluate the sensitivity of our algorithms to the size of the validation set $\mathcal{S}$. For this experiment, we considered various sizes of $\mathcal{S}$ for pruning the LeNet300-100 network trained on MNIST. Figure 5-9 depicts the highest prune ratios obtained with varying size of $\mathcal{S}$ in the prune-only and fine-tune scenarios subject to the constraint of being within 4% and 1% (absolute terms) of the original network's test accuracy, respectively. One aspect to note is that the largest size of $\mathcal{S}$ plotted (268) corresponds to the required size of $\mathcal{S}$ for an input failure probability $\delta = 1 \times 10^{-32}$ (roughly machine precision) in the context of our theory (Theorem 37). Hence, 268 can be considered an upper bound on the size of $\mathcal{S}$ required by our theory on any practical application on this network.

As we can see from Fig. 5-9a, there is a sweet spot for the size of $\mathcal{S}$ (roughly around 175-200 samples) that achieves the highest performance of the SiPP variants. After this global max, the performance of the algorithms actually decreases as we increase the size of $\mathcal{S}$ (see plots after 210 samples in Fig. 5-9a). This trade-off in the size of $\mathcal{S}$ is predicted by our theory and explicitly stated in the discussion of $\mathcal{S}$ in Section 5.3.2.

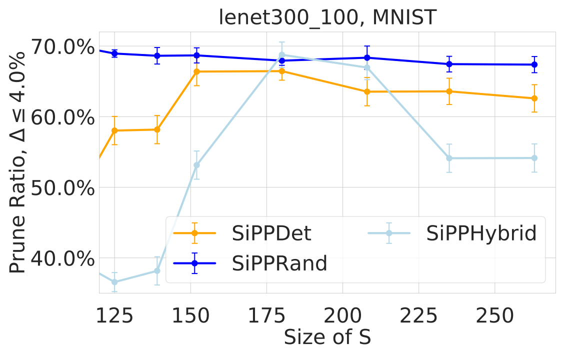


(a) ResNet18, Top 5

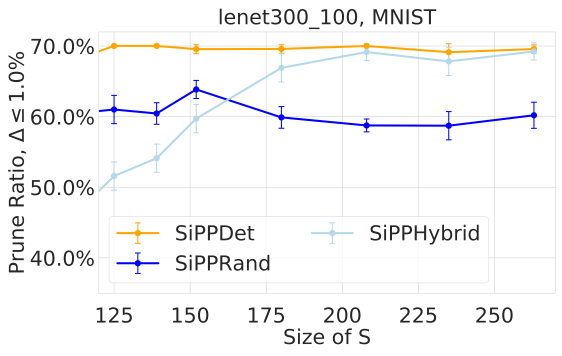


(b) ResNet101, Top 5

Figure 5-8: The Top-5 accuracy of the generated pruned **ResNet18** and **ResNet101** models trained on ImageNet for the evaluated pruning schemes for various target prune ratios. The Top-1 accuracy is shown in Figure 5-3.



(a) Prune-only



(b) Fine-tuning for 1 epoch

Figure 5-9: Evaluations of the sensitivity of SiPP variants to the size of $\mathcal{S}$ for the task of pruning LeNet300-100 trained on MNIST. The plots show the maximal prune ratio obtained subject to the constraint of being within $\Delta \leq 4\%$ and $\Delta \leq 1\%$ (absolute terms) test accuracy of the original network's accuracy, respectively. Overall, we can see that after even a single retraining step, SiPP remains robust to varying size of $\mathcal{S}$.

Fig. 5-9b similarly shows the sensitivity to the size of $\mathcal{S}$ when a single epoch of retraining is executed after the pruning step. Here, we can see a similar trend to Fig. 5-9a, where initially the performance of the variants increases as the size of $\mathcal{S}$ increases, but then tends to plateau (or decay) after a certain size. Interestingly, the decay in performance after a certain size of $\mathcal{S}$ is not as apparent in the fine-tune scenario as it is in the prune-only setting (Fig. 5-9a). We conjecture that this could be due to the fact that a single step of fine-tuning after pruning may serve as a recovery step that alleviates the negative impact of a sub-optimal choice for the size of $\mathcal{S}$.

5.9 Discussion and Future Work

In this work, we presented a simultaneously provably and practical family of network pruning methods, SiPP, that is grounded in a data-informed measure of sensitivity. Our analysis establishes provable guarantees that quantify the trade-off between the desired model sparsity and resulting accuracy of the pruned model establishing novel analytical compression bounds for a large class of neural networks. SiPP's versatility in providing strong prune results across a variety of tasks suggests that our method inherently considers the crucial pathways through the network, and does not merely operate by considering the properties, e.g., values, of the network parameters alone.

We envision that SiPP can spur further research into provable network pruning and that future work can build on the work presented here to develop reliable and effective pruning algorithms. For instance, the analysis provided here aims to minimize the loss in accuracy due to pruning, which intuitively and empirically [LBC+21, LAT18] correlates with a better model after retraining – since the initial solution of the optimization problem is better off. One avenue for future work is to establish rigorous guarantees on the network's performance *after retraining*, by potentially building on existing tools, e.g., those in [AAR20]. The explicit consideration of retraining may also lead to more-informed algorithms that optimize the corresponding retrained error, leading to even more compact networks in practice.

A related research direction of high impact would be to investigate tighter and non worst-case error propagation bounds that mathematically capture the fact that the injected noise (layer-wise error) actually *attenuates* — rather than increases — over the network's layers (see [AGNZ18, Fig. 1]). To the best of our knowledge, there does not exist any prior work that can capture this phenomenon in a mathematically rigorous way without resorting to additional assumptions and task-dependent quantities [AGNZ18] whose values are not known prior to pruning. Even with these assumptions and task-dependent (and a priori unknown) quantities in place, existing bounds are vacuous in that they only hold for the performance of the compressed network over the training set [AGNZ18], which implies that they do not apply to inputs from e.g., the test set, and unforeseen data points in practice.

Finally, in future work we plan to improve the empirical sensitivity framework itself by considering an approach that does not rely on an expression of the form $\max_{x \in \mathcal{S}}$ compute the optimal sampling distribution per layer. For example, one idea is to numerically compute the optimal distribution that minimizes the empirical Bernstein error layer-wise – or more simply, the empirical sampling variance – over the subset of points $\mathcal{S}$ (rather than taking a max over $\mathcal{S}$). This problem is convex in the sampling distribution p and an approximately-optimal solution can be found via gradient-based approaches. The general idea of the proof would then be to use generalization bounds on the empirical Bernstein error minimization problem to quantify the (expected) layer-wise error of the empirically computed sampling distribution on unforeseen data points $x \sim \mathcal{D}$. Our ongoing work suggests that this approach may produce improved theoretical bounds for the *expected* error compared to those that we investigated in this chapter, which hold with probability at least $1 - \delta$. This avenue may also lead to better-performing algorithms in practice.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation (NSF) under Awards 1723943 and 1526815, Office of Naval Research (ONR) Grant N00014-18-1-2830, Microsoft, and JP Morgan Chase.

Chapter 6

Structured Pruning & The Next Pruning Frontier

6.1 Overview

In the previous chapter, we introduced SiPP and its variants for pruning the weights of neural networks. Here, we build on the Empirical Sensitivity (ES) framework to prune neurons and filters from modern networks rather than individual weights. This form of *structured pruning* leads to slimmer networks with reduced-dimensionality weight tensors, enabling inference-time speedups with any off-the-shelf hardware and deep learning library. The flexibility of the framework from the previous chapter leads to a straightforward extension to neuron and filter pruning with minimal modification to the method and analysis. In the interest of clarity and conciseness, we will focus on the main ideas and results of this extension, while making explicit mentions of the modifications (if any) in the techniques and proofs relative to those of SiPP (Chapter 5). The full extent of our work is based on [BLG+18, LBL+20].

We conclude the chapter by discussing the next frontier for pruning in the context of its ramifications. Namely, we investigate other objectives for pruning beyond test accuracy, such as robustness to adversarial or noisy data. Our recently published results [LBC+21] suggest that future pruning algorithms would benefit from incorporating varying objectives such as resilience to out-of-distribution data, rather than just focusing on test accuracy. These observations motivate the development of multi-objective pruning strategies, and in conjunction with the ease of flexibility to filter pruning demonstrated in this chapter, the extension of the ES framework for this purpose as an avenue for future work. The full extent of this work is based on [LBC+21].

6.2 Structured Pruning

As we saw in the previous chapter, we can prune a significant portion of the weights of various benchmark networks (FNNs, CNNs, Resnets, WideResnets) trained on real-world data sets (e.g., CIFAR, Imagenet) without sacrificing the predictive power of the neural network. This leads to easier deployment of these models to resource-constrained platforms and alleviated memory and storage requirements. At the same time, however, weight pruning leads to irregular sparsity patterns. This means that if computational speedup is the practitioner's primary objective, then specialized libraries pertaining to sparse linear algebra algorithms and specialized hardware are required to fully leverage the resulting sparsity.

Structured pruning has potential to alleviate these practical concerns. The main idea is to constrain the pruning procedure so that rather than pruning individuals weights, we remove entire neurons (and/or filters) and all of the corresponding incoming and outgoing edges. The outcome of this procedure is a *slimmer* network where the weight tensors have lower dimensionality. This is due to the fact that, unlike weight pruning, neuron pruning corresponds to removing an entire column[1] worth of weights. The same applies to convolutional layers, where removing a filter reduces the dimension of the weight tensor. Fig. 6-1 depicts an example of the neuron pruning process.

[1]Or a column (WLOG) depending on the definition of the weight matrices and whether a transpose is applied.

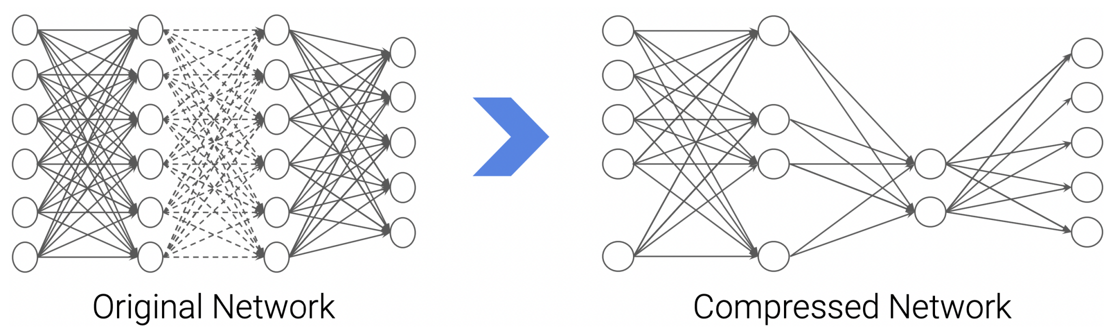


Figure 6-1: An example application of neuron pruning to the original network shown on the left. The generated pruned network on the right is slimmer, i.e., with lower dimensional weight matrices, because entire neurons have been removed from layers.

Prior work in filter pruning has focused on extensions of the popular Weight Thresholding (WT) approach for edge pruning as described in the previous chapter. The premise of these approaches is that filters that are important will have a large weight norm. To this end, prior methods for filter (and neuron) pruning have considered taking various ℓ_p (e.g., $p = 1, 2$) norms of the filters [HKD+18, LKD+16] or the path-norms of the outgoing weights of a neuron [YLC+17]. However, the drawback of these strategies is that they heavily rely on the *Smaller-norm-less-informative* assumption [YLLW18], an assumption that has been challenged in recent work [YLLW18].

Our work, which we term *Provable Filter Pruning (PFP)* on the other hand, builds on the theoretical guarantees and the data-informed approach of SiPP. We present a subset of empirical results in this chapter in the interest of brevity. The full set of empirical results and theoretical analysis, including additional variants of PFP that combine randomized and deterministic approaches, can be found in our published work [LBL+20].

6.3 Provable Filter Pruning

6.3.1 Extending SiPP

PFP builds on the empirical sensitivity framework introduced in the previous chapter and used by SiPP, and is almost identical algorithmically as shown in Fig. 6-2. The

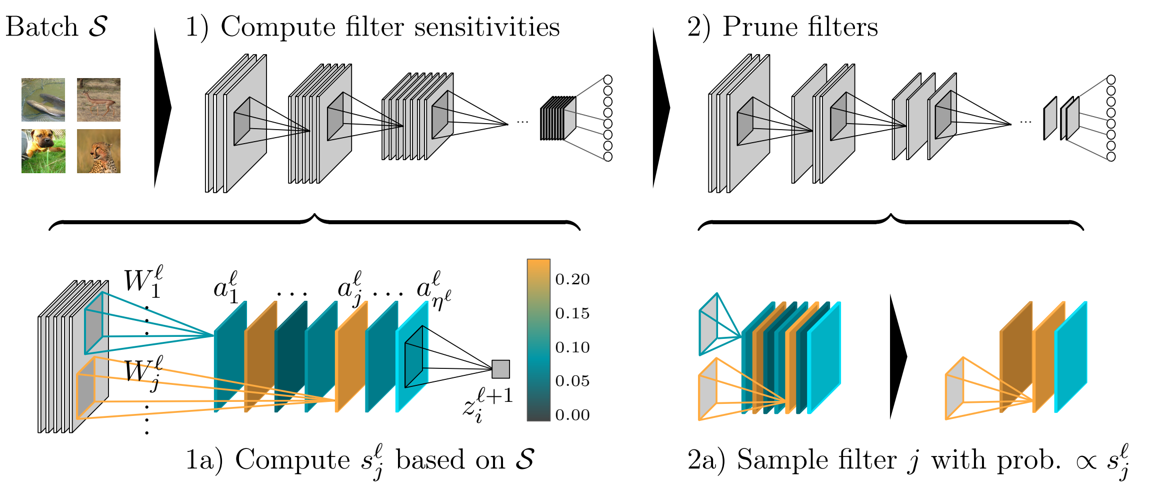


Figure 6-2: Overview of the filter pruning method. The sequence of operations is almost identical to the one presented in Chapter 5 (see Fig. 5-1). The only exception is that the definition of sensitivity for a filter accounts for the maximum impact that it has on any of the outputs in the next layer. This figure originally appeared in Lucas Liebenwein's Ph.D. thesis and is taken from our published work [LBL+20].

only difference is that (i) filters (or neurons) instead of individual weights are sampled and all of the weights of a sampled filter are reweighted and (ii) the definition of empirical sensitivity is slightly different, which we focus on describing next. For ease of exposition we will focus on pruning neurons in a fully-connected network. This can be extended to CNNs and other architectures by using the same technique as SiPP of accounting for all the possible patches (see Chapter 5). The full extension and proofs are in our published work [LBL+20].

Recall that for SiPP in the setting of a fully-connected network, the importance of a weight was defined as its maximum relative contribution to the pre-activation output of the layer (over the points in the batch $\mathcal{S}$). This definition was sufficient to capture the parameter's importance since a weight in a fully-connected network only affects the output of a single neuron in the next layer. However, as we see from Fig. 6-1, for example, this same logic does exactly apply to neuron pruning because removing a neuron in layer ℓ affects *every output in the next layer* $\ell + 1$ since all of the incoming and outgoing edges of that neuron have to be removed. This motivates a definition of sensitivity that also considers the maximum relative impact that removing a neuron can have across all of the outputs in the next layer.

6.3.2 Method and Analysis Overview

In light of the discussion above, we build on the definition of empirical sensitivity from SiPP and define the empirical sensitivity of a neuron in a fully-connected network[2] as follows.

Definition 38 (Empirical neuron sensitivity). *The empirical sensitivity for a neuron $j \in [\eta^\ell]$ is given by*

$$s_j^\ell = \max_{x \in \mathcal{S}} \max_{i \in [\eta^{\ell+1}]} \frac{w_{ij}^{\ell+1} a_j^\ell(x)}{\sum_{k \in [\eta^\ell]} w_{ik}^{\ell+1} a_k^\ell(x)}, \tag{6.1}$$

where $\mathcal{S}$ is a batch of i.i.d. random points from the input data distribution $\mathcal{D}$.

This definition formalizes the intuition above. Here, the numerator $w_{ij}^{\ell+1} a_j^\ell(x)$ denotes the contribution of neuron $j \in [\eta^\ell]$ to the neuron $i \in [\eta^{\ell+1}]$ in the next layer $\ell + 1$, and to account for the *relative* contribution, the denominator is the pre-activation output of neuron i, i.e., the dot product $\sum_{k \in [\eta^\ell]} w_{ik}^{\ell+1} a_k^\ell(x)$. Now, the additional $\max_{i \in [\eta^{\ell+1}]}$ relative to SiPP ensures that we consider the maximum impact of neuron j over all outputs in the next layer as discussed, and the maximum over $\mathcal{S}$ is the same as before in SiPP. This concludes the extent of the sensitivity modification relative to edge pruning.

It turns out that by building on the empirical sensitivity framework and the proofs in the previous chapter, the analysis of PFP is relatively straightforward and requires minimal modification to the proof techniques. We provide an overview of the main ideas here and defer to the published work for the details [LBL+20, BLG+18].

As before, the main idea is to sample and reweigh the neurons (and/or filters) in order to have an unbiased estimate *with low variance and magnitude* for the pre-activation outputs. This is requirement is formalized by Bernstein's concentration inequality below, which bounds the sample complexity of tight concentration around the expectation of a sequence of samples in terms of their variance and maximum

[2]The extension to CNNs and other architectures employs the same strategy as before: additionally reason about all of the patches (sliding windows of filters) by taking an additional $\max_{a \in \mathcal{A}}$ as in SiPP.

magnitude.

Theorem 39 (Bernstein's inequality [Ver16])**.** *Consider a sequence of m i.i.d. random variables $Z_1, \dots, Z_m$ satisfying $\max_{k \in [m]} |Z_k - \mathbb{E}[Z_k]| \leq R$, and let $\bar{Z} = {}^1\!/_m \sum_{k=1}^m Z_k$ be their mean. Then $\forall \varepsilon > 0$, $\delta \in (0,1)$, and*

$$m \geq \frac{\log(2/\delta)}{(\varepsilon E[\bar{Z}])^2} \left(Var(Z_1) + \frac{2}{3} \varepsilon \, \mathbb{E}[\bar{Z}] R \right),$$

we have

$$\mathbb{P}\left(\left| \bar{Z} - \mathbb{E}[\bar{Z}] \right| \geq \varepsilon \, \mathbb{E}[\bar{Z}] \right) \leq \delta.$$

Bernstein's inequality suggests that we should minimize $\mathrm{Var}(Z_k)$ and the maximum (centered) magnitude R in order to have the smallest sampling complexity possible. For SiPP, our empirical sensitivity definition ensured that the variance and the magnitude R were each (roughly) bounded by the sum of sensitivities S and the expectation squared $\mathbb{E}[\bar{Z}]^2$. It turns out that almost an identical bound holds for neuron (and filter) pruning as well under the extended definition of (38). Thus, we arrive at an almost identical result to the one in SiPP for the case of pruning neurons with non-negative weights:

Theorem 40. *Let $\varepsilon, \delta \in (0,1), \ell \in [L]$, and let $\mathcal{S}$ be a set of $\Theta(\log(\eta_* \, \rho/\delta))$ i.i.d. samples drawn from $\mathcal{D}$. Then, for a sample of*

$$\mathcal{O}(S^\ell \, \log(\eta_*/\delta)\varepsilon^{-2}))$$

filters from W^ℓ, with probability[3] *at least $1 - \delta$, we have the following entry-wise guarantee for the output of layer ℓ with respect to input $x \sim \mathcal{D}$*

$$\hat{z}^\ell(x) \in (1 \pm \varepsilon) z^\ell(x),$$

[3]As in SiPP, the randomness here is over both the randomness in the importance sampling and the randomness in the input $x \sim \mathcal{D}$.

where $z^\ell(x)$ is the ground-truth layer output with respect to input x and $\hat{z}^\ell(x)$ is the approximate layer output after pruning the filters in layer ℓ only, and $\eta_* = \max_{\ell \in [L]} \eta^\ell$.

As in Chapter 5, we build on and generalize the bound above to hold for all weights and for all outputs of a single layer ℓ. Subsequently, we conduct an error propagation analysis across all of the pruned layers analogous to that in Sec. 5.7.5, which culminates in our final compression theorem before. To demonstrate the flexibility of our proof techniques, we present the result based on the error propagation analysis of our prior published work [BLG+18], which provides an *entry-wise* approximation guarantee on the outputs of the pruned network, in contrast to the norm-based bound in the previous chapter[4].

Theorem 41. *Let $\varepsilon, \delta \in (0,1)$ be arbitrary, let $\mathcal{S} \subset \mathcal{X}$ denote a set of $K \log(4\eta\,\rho/\delta)$ i.i.d. points drawn from the input data distribution $\mathcal{D}$, and suppose we are given a network with parameters $\theta = (W^1, \ldots, W^L)$. Consider the set of parameters $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$ generated by pruning channels of θ by sampling and reweighting filters in each layer according to the (normalized) empirical sensitivities as in (6.1) for each $\ell \in [L]$. Then, $\hat{\theta}$ satisfies*

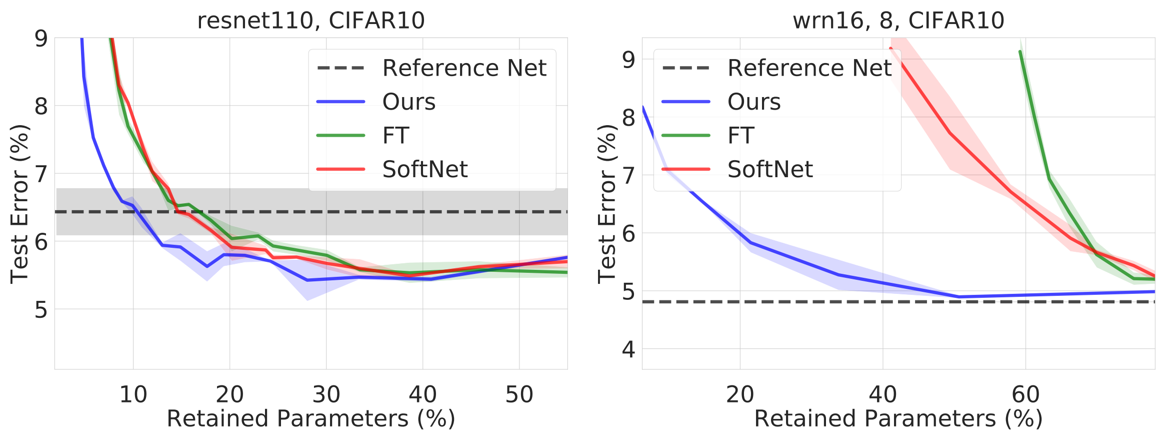
$$\mathop{\mathbb{P}}_{\hat{\theta},\, x \sim \mathcal{D}} \left(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon) f_\theta(x) \right) \geq 1 - \delta,$$

and the number of filters in $\hat{\theta}$ is bounded by

$$\mathcal{O}\left(\sum_{\ell=1}^{L} \frac{L^2\, (\Delta^{\ell \rightarrow})^2\, S^\ell\, C\, \log(\eta/\delta)}{\varepsilon^2} \right),$$

where S^ℓ is the sum of sensitivities over the filters in layer ℓ, and Δ^ℓ is the entry-wise sign complexity [BLG+18, LBL+20] analogous to (33).

[4] Although a similar norm-based bound can be proven for PFP, and likewise, an entrywise bound on the output can be proven for SiPP by applying the error propagation proof in [BLG+18].



(a) ResNet110 (b) WRN16-8

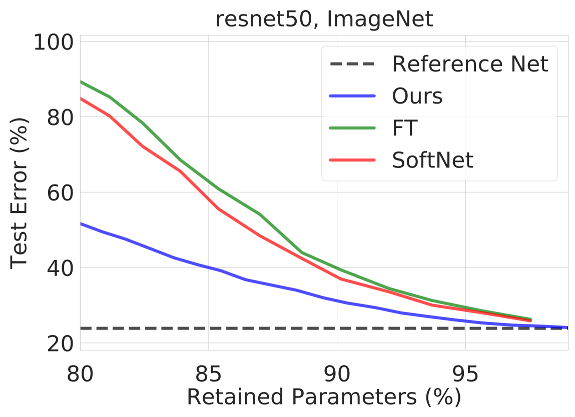
Figure 6-3: The performance of pruned models generated by our algorithm and those of benchmark approaches on the CIAR10 data set. The figures plot the test accuracy as a function of the percentage of the retained parameters (i.e., **lower is better**) for the resnet110 (left) and WRN16-8 (right) models. Shaded regions correspond to values within one standard deviation of the mean. Our results show that our approach generates more compact and accurate networks than competing approaches.

6.4 Experiments

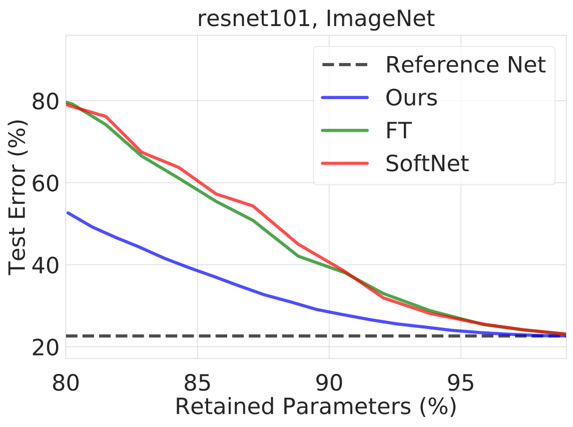
We conclude our exposition of PFP by presenting a small subset of the results published in [LBL+20]. Overall, our evaluations on benchmark data sets and models, and comparisons to state-of-the-art filter pruning approaches demonstrate the improved effectiveness of PFP relative to competing methods in practice.

Fig. 6-3 depicts some of the results on a resnet and wide residual network trained on the CIFAR10 data set. Here, we can see that PFP can generate much more compact models with minimal degradation in accuracy. In fact, for resnet110, it can prune over 90% of the network's filters while achieving commensurate test accuracy with the original network. The results are even more encouraging for wide residual networks (WRN16-8 in Fig. 6-3) where PFP truly shines, which we conjecture is due to the increased number of filters to sample from and the corresponding exponentially-decaying failure probability (by Bernstein's inequality).

To demonstrate the scalability of our approach, we also provide pruning results on the ImageNet data set trained on relatively large resnets (resnet50 and resnet101) in Fig. 6-4. Here, we plot *prune-only* results without a fine-tuning step following it unlike the previous results. Nevertheless, the trend remains the same: PFP generates more compact networks relative to the compared approaches, and arguably, its relative improvement over the competing methods is even more pronounced with the increasing scale of the models and data sets used for evaluation.



(a) ResNet50



(b) ResNet101

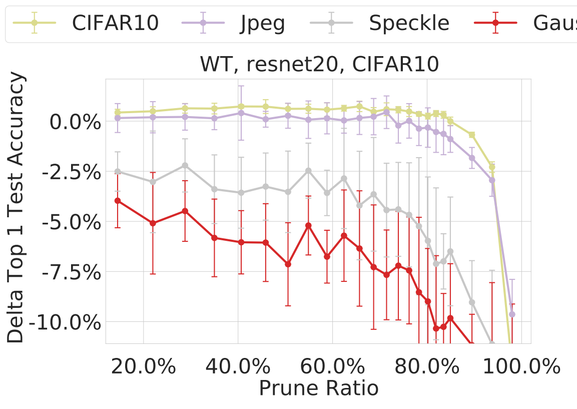
Figure 6-4: The results of our evaluations of the algorithms in the *prune-only* scenario, where the network is iteratively pruned down to a specified target prune ratio and the fine-tuning step is omitted. The figures plot the test accuracy as a function of the percentage of the retained parameters (i.e., lower is better) for resnet50 and resnet101 trained on the ImageNet dataset.

6.5 The Next Pruning Frontier

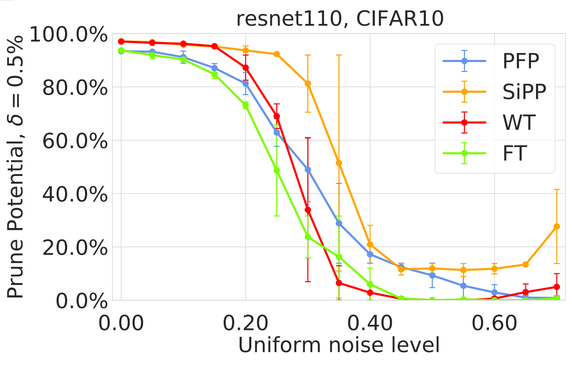
We conclude this chapter and our story on pruning that we started in Chapter 5 by discussing the limitations of network pruning and the next frontier for (provable) network pruning algorithms. The full set of results presented in this section can be found in the published work [LBC+21].

6.5.1 What is Lost in Pruning?

In Chapters 5 and 6, we introduced methods for pruning edges, neurons, and filters from various network architectures. The main objective of our pruning strategy and of virtually every network pruning algorithm proposed to date has been to mitigate drops in test accuracy of the pruned network. However, there may be other objectives beyond test accuracy that might be more important to the application at hand. For example, are pruned networks with commensurate test accuracy as robust to adversarial or noisy data as the original network as well? Relatedly, answering this question would give us insights on whether focusing on test accuracy alone in the design of pruning algorithms is appropriate.



(a) Prune-test curve (WT)



(b) ResNet110

Figure 6-5: Left: a prune-accuracy curve for the Weight Thresholding (WT) pruning algorithm under various types of injected noise. Right: the prune-potential of various pruning algorithms for $\delta = 0.5\%$, i.e., the maximum sparsity achievable subject to the constraint that the test accuracy is within 0.5% of the original network, plotted for various pruning algorithms (including SiPP and PFP) as a function of the injected noise. For both figures, **higher is better** in terms of the pruned network's performance under noise.

In Fig. 6-5 we present a small subset of our results published in [LBC+21] that seeks to answer questions of this nature. In Fig. 6-5a, we show the performance of the highly-popular weight-pruning algorithm, Weight Thresholding (WT), under various types of noise injected into the input (CIFAR10) relative to the noise-free input. Here, we can see that although WT performs well on the original (noiseless) CIFAR10 data set, injecting Gaussian noise (red curve) leads to an accuracy drop of more than 10% relative to the test accuracy of the original network *evaluated on the very same set with Gaussian noise*[5]. We observe similar drops in accuracy for the Speckle noise as well. In sum, Fig. 6-5a suggests the susceptibility of other popularly employed network

[5]Since the test accuracy would degrade with noisy inputs as it is, our plot shows the test accuracy of the pruned network on the noisy data *relative* to the test accuracy of the original network on the same noisy data.

pruning algorithms to noisy, out-of-distribution, and adversarial inputs.

Next, we consider the impact of the *magnitude* of the injected noise and compare the performance of various pruning algorithms, including SiPP and PFP. Fig. 6-5b depicts the pruning results of our evaluations with varying noise levels injected into the CIFAR10 data sets. Here, we plot the relationship between the noise level and the *prune potential* with respect to $\delta = 0.5\%$, i.e., the maximum sparsity achieved subject to the constraint that the test accuracy of the pruned network is within 0.5% (absolute terms) of the original network. Here we see an intuitive trend: increasing the magnitude of the injected noise leads to decreasing performance across all algorithms. Interestingly, our proposed data-informed algorithms (SiPP and PFP) outperform weight-based edge and filter-pruning approaches, WT and FT, respectively.

6.5.2 Discussion and Future Work

The empirical results presented in this chapter demonstrate the potential vulnerability of existing and widely used pruning algorithms to various types and levels of noise in the input. This is especially worrisome for the deployment of pruned networks to safety-critical applications such as autonomous driving, where out-of-distribution data and noisy inputs are commonly encountered in practice [ASSR19].

These observations motivate the development of pruning algorithms that can consider objectives, for example, robustness to adversarial input, beyond test accuracy alone. We envision that future work in network pruning can build on the empirical sensitivity framework that we introduced in this thesis in order to develop pruning algorithms with this capability. Given the ease of extension of our framework from edge to filter pruning as demonstrated in this chapter, we are hopeful that similar advances can be made possible by appropriately building on the notion of empirical sensitivity, as we did here.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation (NSF) under Awards 1723943 and 1526815, Office of Naval Research (ONR) Grant N00014-18-1-2830, Microsoft, and JP Morgan Chase.

Chapter 7

Active Learning for Deep Learning

7.1 Overview

In the previous two chapters (Chapters 5 and 6), we focused on provably pruning large neural network models to obtain compact, efficient AI. For the last chapter, we consider alleviating the other prominent challenge in scalable AI: the need for massive sets of labeled training data. In particular, we revisit the problem of compressing the input data set as in Chapter 4 (for SVMs), but this time with a focus on reducing the label-cost of large-scale neural network training. Due to lack of knowledge about the labels and the theoretical complexity of neural network training, we cannot readily apply standard techniques such as the sensitivity framework from the previous chapters. Hence, we instead formulate the active learning problem as the prediction with *sleeping* expert advice problem and introduce a novel low-regret algorithm. Our approach seeks to remedy the shortcomings of prior work, which has predominantly focused on greedy heuristics that simply pick the points that are ranked as most informative according to a proxy measure as discussed in Chapter 2.

The work presented in this chapter is based on [BLFR21] and contributes the following:

1. Formulation of active learning as a prediction with sleeping experts problem and

the development of an efficient, predictive algorithm AdaProd$^+$ for low-regret active learning,

2. Analysis of its regret bounds and demonstration of its ability to broadly improve and robustify existing greedy approaches off-the-shelf,

3. Empirical evaluations that demonstrate the effectiveness of the presented method on a diverse set of benchmarks and its uniformly superior performance over competitors across scenarios involving real-world data sets and modern architectures.

7.2 Active Learning

We consider the setting where we are given a set of n unlabeled data points $\mathcal{P} \subset \mathcal{X}^n$ from the input space $\mathcal{X} \subset \mathbb{R}^d$. We assume that there is an oracle Oracle that maps each point $x \in \mathcal{P}$ to one of k categories. Given a network architecture and sampling budget $b \in \mathbb{N}_+$, our goal is to generate a subset of points $\mathcal{S} \subset \mathcal{P}$ with $|\mathcal{S}| = b$ such that training on $\{(x, \textsc{Oracle}(x))_{x \in \mathcal{S}}\}$ leads to the most accurate model θ among all other choices for a subset $\mathcal{S} \subset \mathcal{P}$ of size b.

The iterative variant of acquisition procedure is shown as Alg. 10, where Acquire is an active learning algorithm that identifies (by using θ_{t-1}) b_t unlabeled points to label at each iteration $t \in [T]$ and Train trains a model initialized with θ_{t-1} using the labeled set of points. We emphasize that prior work has overwhelmingly used the Scratch option (Line 6, Alg. 10), which entails discarding the model information θ_{t-1} from the previous iteration and training a randomly initialized model from scratch on the set of labeled points acquired thus far, $\mathcal{S}$.

7.2.1 Background & Greedy Selection

Consider an *informativeness function* $g : \mathcal{X} \times \Theta \to [0, 1]$ that quantifies the informativeness of each point $x \in \mathcal{X}$ with respect to the model $\theta \in \Theta$, where Θ is the

Algorithm 10 ActiveLearning

```
Input: Set of points $\mathcal{P} \subseteq \mathbb{R}^{d \times n}$, Acquire: an active learning algorithm for selecting labeled points
1: $\mathcal{S} \leftarrow \emptyset$;   $\theta_0 \leftarrow$ Randomly initialized network model;
2: for $t \in [T] = \{1, \ldots, T\}$ do
3:     $\mathcal{C}_t \leftarrow$ Acquire$(\mathcal{P} \setminus \mathcal{S}, b_t, \theta_{t-1})$    ▷ Get new batch of $b_t \in \mathbb{N}_+$ points to label using algorithm Acquire
4:     $\mathcal{S} \leftarrow \mathcal{S} \cup C_t$    ▷ Add new points
5:     (if Scratch option) $\theta_{t-1} \leftarrow$ Randomly initialized network
6:     $\theta_t \leftarrow$ Train$(\theta_{t-1}, \{(x, \text{Oracle}(x))_{x \in \mathcal{S}}\})$    ▷ Train network on the labeled samples thus far
7: return $\theta_T$
```

set of all possible parameters for the given architecture. An example of the gain function is the maximum variation ratio (also called the *uncertainty* metric) defined as $g(x, \theta) = 1 - \max_{i \in [k]} f_\theta(x)_i$, where $f_\theta(x) \in \mathbb{R}^k$ is the softmax output of the model θ given input x. As examples, the *gain* $g(x, \theta)$ of point x is 0 if the network is absolutely certain about the label of x and $1 - 1/k$ when the network's prediction is uniform. In the context of Alg. 10, prior work on active learning [Mut19, GEY17, GIG17, SS17a] has generally focused on greedy acquisition strategies (Acquire in Alg. 10) that rank the remaining unlabeled points by their informativeness $g(x, \theta_{t-1})$ *as a function of the model* θ_{t-1}, and pick the top b_t points to label.

Why greedy can fail Greedy approaches to data acquisition have shown promise in certain active learning applications and tasks [GIG17, SS17a]), however, as noted in Sec. 2.6 – and further evidenced by our empirical results in Sec. 7.5 – these approaches are highly sensitive to outliers and at times perform significantly worse than naive uniform sampling. In fact, Fig. 2-1 depicts a scenario where various popular active learning approaches perform significantly worse than uniform sampling.

To understand why this could be happening, note that at iteration $t \in [T]$ the greedy approach makes a judgment about the informativeness of each point using only the model θ_{t-1} (Line 4 of Alg. 10). However, in the deep learning setting where stochastic elements such as random initialization, stochastic optimization, (randomized) data augmentation, and dropout are commonly present, θ_{t-1} is itself a random variable with non-negligible variance. This means that, for example, we could get unlucky with our training and obtain a deceptive model θ_{t-1} (e.g., training diverged) that assigns high gains (informativeness) to points that may not truly be helpful towards training a better model. Nevertheless, GREEDY would still base the entirety of the decision making solely on θ_{t-1} (not on the entire history $\theta_1, \ldots \theta_{t-2}$) and blindly pick the top-b_t points ranked using θ_{t-1}, leading to a misguided selection. This example also applies to greedy clustering, e.g., CORESET [SS17a], BADGE [AZK+19].

7.2.2 Active Learning as Prediction with Expert Advice

Rather than attempting to model this randomness in the gains observed, we will assume that the gains can be generated by a *non-oblivious adversary* that has knowledge of our actions in the preceding rounds. This formulation leads us to the well-studied learning with experts problem as we outline in this subsection. We note that at the expense of formulating a seemingly more difficult problem involving an adversary, we obtain the benefit of generality and widespread applicability without imposing any assumptions on how the gains (i.e., informativeness) are defined or generated.

Setting We let $g_{t,i}$ denote the gain $g(x_i, \theta_{t-1})$ (see Alg. 10) in round $t \in [T]$ where x_i is the i^{th} point in $\mathcal{P}$. For ease of presentation[1], assume for the time being that $b_t = 1$, i.e., we select a single new point at each iteration. Rather than picking this point deterministically, consider selecting sampling this point with respect to a probability distribution $p \in \Delta$ where $\Delta = \{p \in [0,1]^n : \sum_{j=1}^n p_j = 1\}$ is the probability simplex. We note that there have been prior attempts to make greedy approaches robust by sampling in this very same way (rather than deterministic selection), however, it was observed that this led to *worse* performance in practice [GSS19].

To map this problem to the canonical learning with experts problem, we consider the problem of minimizing the sum of losses $\ell_{t,i} = 1 - g_{t,i} \in [0,1]$ over the T active learning

[1]See Sec. 7.3.3 and Sec. 7.7 for the extension to the batch setting.

iterations. Under this setting a natural first attempt at a formulation of expected regret is to define the regret as $\mathrm{Regret}(p_1, \ldots, p_T) = \sum_{t=1}^{T} \langle p_t, l_t \rangle - \min_{p \in \Delta} \sum_{t=1}^{T} \langle p, l_t \rangle$, where $\langle p_t, l_t \rangle$ is the expected loss of our random choice $i \sim p_t$ conditioned on our past history. However, the previous formulation is ill-equipped for active learning, since (i) we should not be picking points that have already been labeled in prior active learning iterations and (ii) it does not make sense to compete with a fixed distribution when the set of actions, i.e., pool of remaining unlabeled data, is shrinking over time.

Sleeping Experts and Dynamic Regret To resolve these challenges and ensure that we only sample from the pool of unlabeled data points, we generalize the prior formulation to one with *sleeping experts* [SGV20, LS15, GSVE14, KNMS10]. More concretely, let $\mathcal{I}_{t,i} \in \{0, 1\}$ denote whether expert $i \in [n]$ is sleeping in round t. The sleeping expert problem imposes the constraint that $\mathcal{I}_{t,i} = 0 \Rightarrow p_{t,i} = 0$. For the data acquisition setting, we define for each $i \in [n]$ $\mathcal{I}_{t,i} = \mathbb{1}\{x_i \text{ not picked in any of the preceding rounds}\}$, so that we do not sample already-labeled points, and formulate the dynamic active learning regret as

$$\mathrm{Regret}(\mathbf{p}) = \sum_{t=1}^{T} \langle p_t, l_t \odot \mathcal{I}_t \rangle - \sum_{t=1}^{T} \min_{p \in \mathcal{A}_t} \langle p, l_t \odot \mathcal{I}_t \rangle \tag{7.1}$$

where $\mathbf{p} = (p_1, \ldots, p_T)$ is the sequence of sampling distributions over T and $\mathcal{A}_t = \{p \in \Delta : \forall i \in [n]\ p_i = 0 \text{ if } \mathcal{I}_{t,i} = 0\}$ is the constrained probability simplex with respect to $\mathcal{I}_t \in \{0, 1\}^n$.

7.3 A Low-Regret Approach

In this section we motivate and present Alg. 11, an efficient online learning algorithm with instance-dependent guarantees that performs well on predictable sequences while remaining resilient to adversarial ones. Additional implementation details are deferred to Sec. 7.8 for ease of exposition.

7.3.1 Background

Algorithms for the prediction with sleeping experts problem have been extensively studied in literature [GSVE14, LS15, SGV20, KNMS10, SAM19, KVE15]. These algorithms enjoy strong guarantees in the adversarial setting; however, they suffer from (i) sub-optimal regret bounds in predictable settings and/or (ii) high computational complexity. Our approach hinges on the observation that the active learning setting may not always be adversarial in practice, and if this is the case, we should be competitive with greedy approaches. For example, we may expect the informativeness of the points to resemble a predictable sequence plus random noise which models the random components of the training (see Sec. 7.2) at each time step. This (potential) predictability in the corresponding losses motivates an algorithm that can leverage predictions about the loss for the next time step to achieve lower regret by being more aggressive – akin to Greedy – when the losses do not vary significantly over time.

7.3.2 AdaProd$^+$

To this end, we extend the Optimistic Adapt-ML-Prod algorithm [WHL17] (henceforth, OAMLProd) to the active learning setting with batched plays where the set of experts (unlabeled data points) is changing and/or unknown in advance. Optimistic online learning algorithms are capable of incorporating predictions $\hat{\ell}_{t+1}$ for the loss in the next round ℓ_{t+1} and guaranteeing regret as a function of the predictions' accuracy, i.e., as a function of $\sum_{t=1}^{T} ||\ell_t - \hat{\ell}_t||_\infty^2$. Although we could have attempted to extend other optimistic approaches [SL14, Ora19, MY15, RS13], the work of [WHL17] ensures – to the best of our knowledge – the smallest regret in predictable environments when compared to related approaches.

Our algorithm AdaProd$^+$ is shown as Alg. 11. Besides its improved computational efficiency relative to OAMLProd in the active learning setting, AdaProd$^+$ is also the result of a tightened analysis that leads to significant practical improvements over OAMLProd as shown in Fig. 7-4 of Sec. 7.5.5. Our insight is that our predictions can be leveraged to improve practical performance by allowing larger learning rates

Algorithm 11 AdaProd$^+$

```
1: For all $i \in [n]$, initialize $R_{1,i} \leftarrow 0$;   $C_{1,i} \leftarrow 0$;   $\eta_{0,(1,i)} \leftarrow \sqrt{\log n}$;  $w_{0,(1,i)} = 1$;
   $\hat{r}_{1,i} = 0$;
2: for each round $t \in [T]$ do
3:     $\mathcal{A}_t \leftarrow \{i \in [n] : \mathcal{I}_{t,i} = 1\}$;    ▷ Set of awake experts, i.e., set of unlabeled data
   points
4:     $p_{t,i} \leftarrow \sum_{s\in[t]} \eta_{t-1,(s,i)} w_{t-1,(s,i)} \exp(\eta_{t-1,(s,i)}\, \hat{r}_{t,i})$    for each $i \in \mathcal{A}_t$
5:     $p_{t,i} \leftarrow {}^{p_{t,i}}/_{\sum_{j\in\mathcal{A}_t} p_{t,j}}$ for each $i \in \mathcal{A}_t$    ▷ Normalize
6:     Adversary reveals $\ell_t$ and we suffer loss $\tilde{\ell}_t = \langle \ell_t, p_t \rangle$
7:     For all $i \in \mathcal{A}_t$, $r_{t,i} \leftarrow \tilde{\ell}_t - \ell_{t,i}$ and $C_{t,i} \leftarrow 0$
8:     For all $i \in \mathcal{A}_t$ and $s \in [t]$, set $C_{s,i} \leftarrow C_{s,i} + (\hat{r}_{t,i} - r_{t,i})^2$
9:     Get prediction $\hat{r}_{t+1} \in [-1, 1]^n$ for next round (see Sec. 7.3.2)
10:    For all $i \in \mathcal{A}_t$, set $w_{t-1,(t,i)} \leftarrow 1$, $\eta_{t-1,(t,i)} \leftarrow \sqrt{\log n}$, and for all $s \in [t]$, set

       $\eta_{t,(s,i)} \leftarrow \min\left\{\eta_{t-1,(s,i)}, \frac{2}{3(1+\hat{r}_{t+1,i})}, \sqrt{\frac{2\log(n)}{C_{s,i}}}\right\}$   and
       $w_{t,(s,i)} \leftarrow \left(w_{t-1,(s,i)} \exp\left(\eta_{t-1,(s,i)}\, r_{t,i} - \eta_{t-1,(s,i)}^2 (r_{t,i} - \hat{r}_{t,i})^2\right)\right)^{\eta_{t,(s,i)}/\eta_{t-1,(s,i)}}$
```

to be used without sacrificing theoretical guarantees (Line 10 of Alg. 11). Empirical comparisons with Adapt-ML-Prod and other state-of-the-art algorithms can be found in Sec. 7.5.5.

Generating Predictions Our approach can be used with general predictors $\hat{\ell}_t$ for the true loss ℓ_t at round t, however, to obtain bounds in terms of the temporal variation in the losses, we use the most recently observed loss as our prediction for the next round, i.e., $\hat{\ell}_t = \ell_{t-1}$. A subtle issue is that our algorithm requires a prediction $\hat{r}_t \in [-1, 1]^n$ for the instantaneous regret at round t, i.e., $r_t = \langle p_t, \ell_t \rangle - \ell_t$, which is not available since p_t is a function of r_t. To achieve this, we follow [WHL17] and define the mapping $\hat{r}_t : \alpha \mapsto (\alpha - \ell_t) \in [-1, 1]^n$ and perform a binary search over the update rule in Lines 4-5 of Alg. 11 so that $\alpha \in [0, 1]$ is such that $\alpha = \langle p_t(\hat{r}_t(\alpha)), \hat{\ell}_t \rangle$, where $p_t(\hat{r}_t(\alpha))$ is the distribution obtained when $\hat{r}_t(\alpha)$ is used as the optimistic prediction in Lines 4-5. The existence of such an α follows by applying the intermediate value theorem to the continuous update.

7.3.3 Back to Active Learning

To unify AdaProd$^+$ with Alg. 10, observe that we can define the Acquire function to be a procedure that at time step t first samples a point by sampling with respect to probabilities p_t, obtains the (user-specified) losses ℓ_t with respect to the model θ_{t-1}, and passes them to Alg. 11 as if they were obtained from the adversary (Line 6). This generates an updated probability distribution p_{t+1} and we iterate.

To generalize this approach to sampling a batch of b_t points, we build on ideas from [UNK10]. Here, we provide an outline of this procedure; the full details are provided (Sec. 7.7). At time t, we apply a capping algorithm [UNK10] to the probability p_t generated by AdaProd$^+$ – which takes $\mathcal{O}(n_t \log n_t)$ time, where $n_t \leq n$ is the number of remaining unlabeled points at iteration t – to obtain a modified distribution $\tilde{p}_t$ satisfying $\max_i \tilde{p}_{t,i} \leq {}^1/_{b_t}$. This projection to the capped simplex ensures that the scaled version of $\tilde{p}_t$, $\hat{p}_{t,i} = b_t \tilde{p}_{t,i}$, satisfies $\hat{p}_{t,i} \in [0,1]$ and $\sum_j \hat{p}_{t,j} = b_t$. Now the challenge is to sample exactly b_t distinct points according to probability $\hat{p}_t$. To achieve this, we use a dependent randomized rounding scheme [GKPS06] (Alg. 13 in 7.7) that runs in $\mathcal{O}(n_t)$ time. The overall computational overhead of batch sampling is $\mathcal{O}(n_t \log n_t)$.

7.3.4 Flexibility via Proprietary Loss

We end this section by underscoring the generality of our approach, which can be applied *off-the-shelf with any definition of informativeness measure that defines the loss* $\ell \in [0,1]^n$, i.e., 1 - informativeness. For example, our framework can be applied with the *uncertainty metric* as defined in Sec. 7.2 by defining the losses to be $\ell_{t,i} = \max_{j \in [k]} f_{\theta_{t-1}}(x_i)_j$. As we show in Sec. 7.5.4, we can also use other popular notions of informativeness such as Entropy [RXC$^+$20] and the BALD metrics [GIG17] to obtain improved results *relative to greedy selection.* This flexibility means that our approach can always be instantiated with any state-of-the-art notion of informativeness, and consequently, can scale with future advances in appropriate notions of informativeness widely studied in literature.

7.4 Regret Guarantees

In this section, we present the theoretical guarantees of our algorithm in the learning with sleeping experts setting. Our main result is an instance-dependent bound on the dynamic regret of our approach in the active learning setting. Here, we focus on the key technical results for clarity of exposition and refer the interested reader to Sec. 7.6 for the full proofs and technical details.

The main idea of our analysis is to show that AdaProd$^+$ (Alg. 11), which builds on Optimistic Adapt-ML-Prod [WHL17], retains the adaptive regret guarantees of the time-varying variant of their algorithm without having to know the number of experts a priori [WHL17]. Inspired by AdaNormalHedge [LS15], we show that our algorithm can efficiently ensure adaptive regret by keeping track of $\sum_{t\in 1}^{T} n_t \leq nt$ experts at time step t, where n_t denotes the number of unlabeled points remaining, $n_t = \sum_{i=1}^{n} \mathcal{I}_{t,i}$, rather than nt experts as in prior work. This leads to efficient updates and applicability to the active learning setting where the set of unlabeled points remaining (experts) significantly shrinks over time.

Our second contribution is an improved learning rate schedule (Line 10 of Alg. 11) that arises from a tightened analysis that enables us to get away with strictly larger learning rates without sacrificing any of the theoretical guarantees. For comparison, the learning rate schedule of [WHL17] would be $\eta_{t,(s,i)} = \min\{1/4, \sqrt{2\log(n)/(1 + C_{s,i})}\}$ in the context of Alg. 11. It turns out that the dampening factor of 1 from the denominator can be removed, and the upper bound of 1/4 is overly-conservative and can instead be replaced by $\min\{\eta_{t-1,(s,i)}, 2/\left(3(1 + \hat{r}_{t+1,i})\right)\}$. This means that we can leverage the predictions at round t to set the threshold in a more informed way. Although this change does not improve (or change) the worst-case regret bound asymptotically, our results in Sec. 7.5 (see Fig. 7-4) show that it translates to significant practical improvements in the active learning setting.

In light of the above, we summarize the two main results here in the interest of clarity, and refer to Sec. 7.6 for the full technical details and complete proofs. The lemma below bounds the *adaptive regret* of AdaProd$^+$, which concerns the cumulative regret over a time interval $[t_1, t_2]$, with respect to $C_{t_2,(t_1,i)} = \sum_{t=t_1}^{t_2} (r_{t,i} - \hat{r}_{t,i})^2$.

Lemma 42 (Adaptive Regret of AdaProd$^+$)**.** *For any $t_1 \leq t_2$ and $i \in [n]$, the regret of Alg. 11 with respect to time interval $[t_1, t_2]$ is bounded by*

$$\sum_{t=t_1}^{t_2} r_{t,i} \leq \hat{\mathcal{O}}\left(\log(n) + \sqrt{C_{t_2,(t_1,i)} \log(n)}\right),$$

where $\hat{\mathcal{O}}$ suppresses $\log T$ factors, $r_{t,i} = \langle \ell_t, p_t \rangle - \ell_{t,i}$ is the instantaneous regret of $i \in [n]$ at time t.

It turns out that there is a deep connection between dynamic (see (7.1)) and adaptive regret, and that an adaptive regret bound implies a dynamic regret bound [LS15]. Hence, we combine the adaptive regret bound from above with Theorem 4 of [LS15] to conclude the main regret guarantee of our algorithm. We note that the theorem below is specialized to our setting where we predict the next round's loss to be the most recently observed loss, i.e., $\hat{\ell}_t = \ell_{t-1}$. More generally, the bound would roughly scale with

$$\sqrt{\sum_{t\in[T]} ||(\ell_t - \hat{\ell}_t) \odot \mathcal{I}_t||_\infty^2},$$

where $\odot$ is the Hadamard (entry-wise) product, with other choices for $\hat{\ell}_t$, such as the mean over the losses seen thus far.

Theorem 43 (Dynamic Regret)**.** AdaProd$^+$ *takes at most $\tilde{\mathcal{O}}(tn_t)$ [2] time for the t^{th} update, and its dynamic regret (Eq. 7.1) with batch size $b_t = 1$ over T steps is bounded by*

$$\mathrm{Regret}(p_1, \ldots, p_t) \leq \tilde{\mathcal{O}}\left(\sqrt{\left(1 + \sum_{t\in[T]} ||(\ell_t - \ell_{t-1}) \odot \mathcal{I}_t||_\infty^2\right) \mathcal{V}_+(u^*_{1:T})}\right),$$

where $\odot$ is the Hadamard (entry-wise) product, $u^*_{1:T} = (u^*_1, \ldots, u^*_T)$ is such that for

[2]We use $\tilde{\mathcal{O}}(\cdot)$ to suppress $\log T$ and $\log n$ factors.

each t,

$$u_t^* = \operatorname*{argmin}_{u \in \mathcal{A}_t} \langle u, \ell_t \rangle,$$

and

$$\mathcal{V}_+(u_{1:T}^*) = \sum_{t=1}^{T} \sum_{i=1}^{n} \max\{0, u_{t,i}^* - u_{t-1,i}^*\}$$

is a measure of the variation in the time-varying optimal distribution.

7.5 Experiments

In this section, we present evaluations of our algorithm and compare the performance of its variants on common vision tasks. Additional results can be found in 7.8. Our evaluations across a diverse set of configurations and benchmarks demonstrate the practical effectiveness and reliability of our method. In particular, they show that our approach (i) is the only one to significantly improve on the performance of uniform sampling across all scenarios, (ii) reliably outperforms competing approaches even with the intuitive UNCERTAINTY metric (Fig. 7-1,7-2), (iii) when instantiated with other metrics, leads to strict improvements over greedy selection (Fig. 7-3), and (iv) outperforms modern algorithms for learning with expert advice (Fig. 7-4).

7.5.1 Setup

We compare our active learning algorithm Alg. 11 (labeled OURS) with the *uncertainty* loss described in Sec. 7.2; UNCERTAINTY: greedy variant of our algorithm with the same measure of informativeness; ENTROPY: greedy approach that defines informativeness by the entropy of the network's softmax output; CORESET: clustering-based active learning algorithm of [SS17a, GEY17]; BATCHBALD: approach based on the mutual information of points and model parameters [KVAG19]; and UNIFORM sampling. We implemented the algorithms in Python and used the PyTorch [PGM+19] library for deep learning.

We consider the following popular vision data sets trained on modern convolutional networks:

1. **FashionMNIST**[XRV17]: $60,000$ grayscale images of size 28×28

2. **CIFAR10** [KH+09]: $50,000$ color images (32×32) each belonging to one of 10 classes

3. **SVHN** [NWC+11]: $73,257$ real-world images (32×32) of digits taken from Google Street View

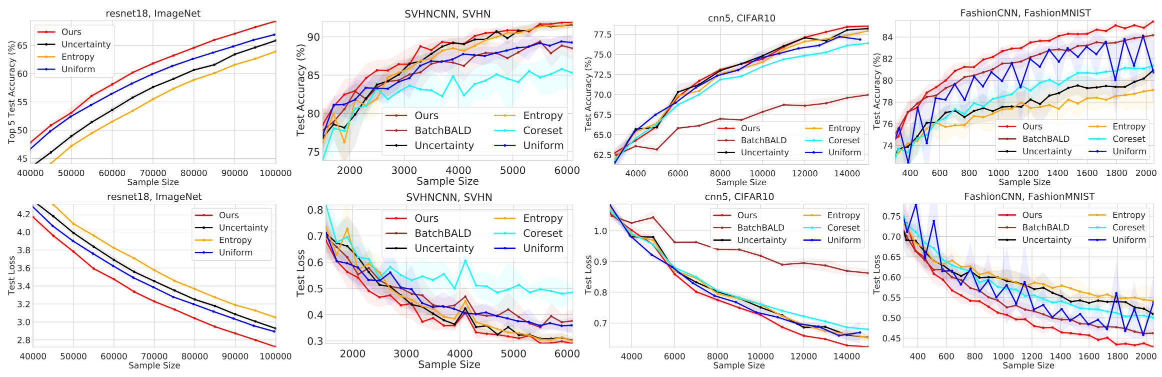
4. **ImageNet** [DDS+09]: more than 1.2 million images spanning 1000 classes

We used standard convolutional networks for training FashionMNIST [XRV17] and SVHN [Che20], the CNN5 architecture [NKB+19] and residual networks (resnets) [HZRS16] for our evaluations on CIFAR10 and ImageNet. The networks were trained with optimized hyper-parameters from the corresponding reference. All results were averaged over 10 trials unless otherwise stated. The full set of hyper-parameters and details of each experimental setup are deferred to Sec. 7.8 for clarity of exposition.

Computation Time Across all data sets, our algorithm took at most 3 minutes per update step. This was comparable (within a factor of 2) to that required by UNCERTAINTY and ENTROPY. However, relative to more sophisticated approaches, OURS was up to $\approx$ 12.3x faster than CORESET, due to expensive pairwise distance computations involved in clustering, and up to $\approx$ 11x faster than BATCHBALD, due to multiple (≥ 10) forward passes over the entire data on a network with dropout required for its Bayesian approximation [KVAG19].

7.5.2 Evaluations on Vision Tasks

As our initial experiment, we evaluate and compare the performance of our approach on benchmark computer vision applications. Fig. 7-1 depicts the results of our experiments on the data sets evaluated with respect to test accuracy and test loss of the obtained network. For these experiments, we used the standard methodology [RXC+20, GIG17]



(a) ImageNet (b) SVHN (c) CIFAR10 (d) FashionMNIST

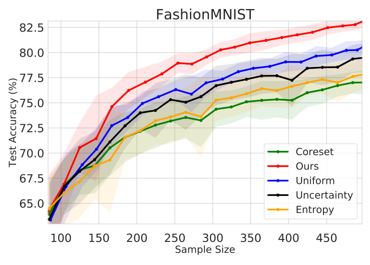
Figure 7-1: Evaluations on popular computer vision benchmarks trained on convolutional neural networks. Our algorithm consistently achieves higher performance than uniform sampling and outperforms or matches competitors on all scenarios. This is in contrast to the highly varying performance of competing methods. *Shaded regions correspond to values within one standard deviation of the mean.*

of retraining the network from scratch as the option in Alg. 10.

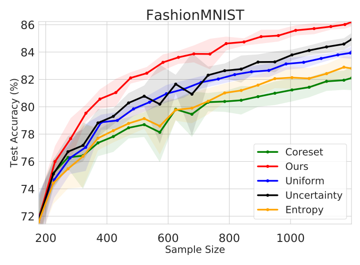
Note that for all data sets, our algorithm (shown in red) consistently outperforms uniform sampling, and in fact, also leads to reliable and strict improvements over existing approaches for all data sets. On ImageNet, we consistently perform better than competitors when it comes to test accuracy and loss. This difference is especially notable when we compare to greedy approaches that are outpaced by UNIFORM by up to $\approx 5\%$ test accuracy. Our results support the widespread reliability and scalability of ADAPROD$^+$, and show promise for its effectiveness on even larger models and data sets.

7.5.3 Robustness Evaluations

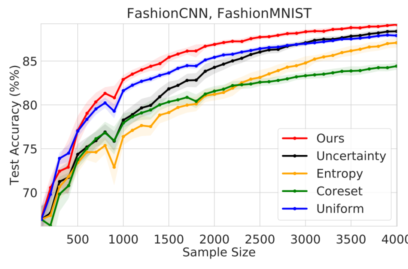
Next, we investigate the robustness of the considered approaches across varying data acquisition configurations evaluated on *a fixed data set*. To this end, we define a data acquisition configuration as the tuple $(\text{OPTION}, n_{\text{start}}, b, n_{\text{end}})$ where OPTION is either SCRATCH or INCR in the context of Alg. 10, n_{start} is the number of initial points at the first step of the active learning iteration, b is the fixed label budget per iteration, and n_{end} is the number of points at which the active learning process stops. Intuitively, we expect robust active learning algorithms to be resilient to changes in the data acquisition configuration and to outperform uniform sampling in a configuration-agnostic way.



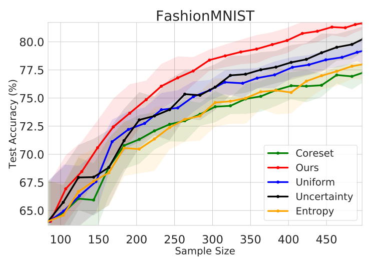
(a) $(\textsc{Scratch}, 50, 15, 500)$



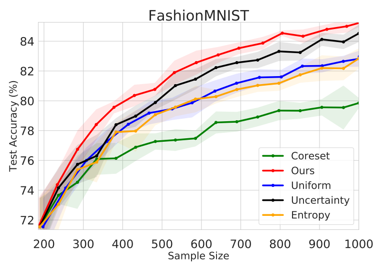
(b) $(\textsc{Scratch}, 150, 50, 1200)$



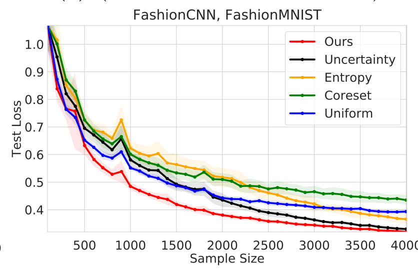
(c) $(\textsc{Scratch}, 50, 100, 4000)$



(d) $(\textsc{Incr}, 50, 15, 500)$



(e) $(\textsc{Incr}, 150, 64, 1000)$

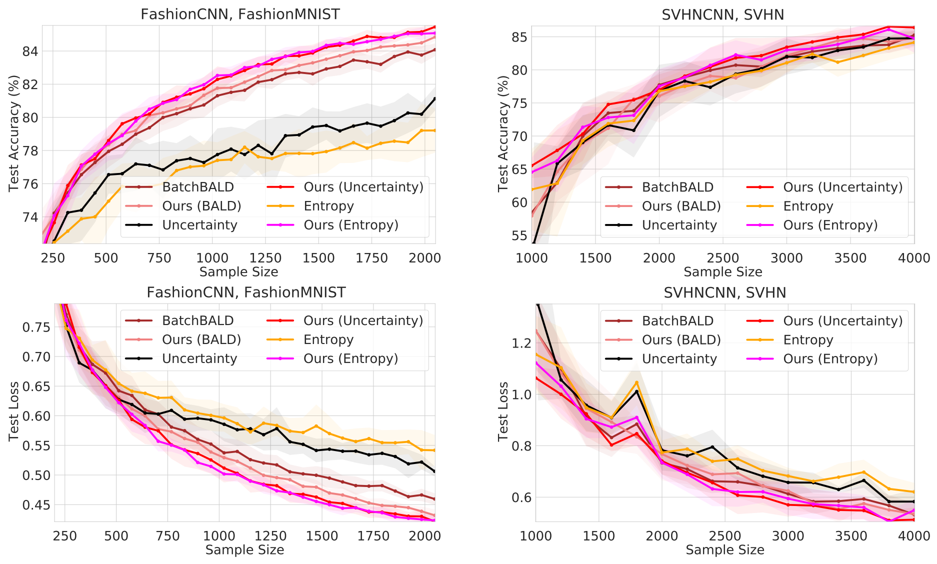


(f) $(\textsc{Incr}, 50, 100, 4000)$

Figure 7-2: Our evaluations on the FashionMNIST data set with varying data acquisition configurations and Incr and Scratch – $(\textsc{Option}, n_{\text{start}}, b, n_{\text{end}})$. All figures except for (f) depict the test accuracy. The performance of competing methods varies greatly across configurations even when the data set is fixed.

Fig. 7-2 shows the results of our experiments on FashionMNIST. From the figures, we can see that our approach performs significantly better than the compared approaches in terms of both test accuracy and loss in all evaluated configurations. In fact, the compared methods' performance fluctuates wildly, supporting our premise about greedy acquisition. For instance, we can see that the uncertainty metric in Fig. 7-2 fares worse than naive uniform sampling in (a), but outperforms Uniform in settings (d) and (e); curiously, in (c), it is only better after an interesting cross-over point towards the end.

This inconsistency and sub-uniform performance is even more pronounced for the Entropy and Coreset algorithms that tend to perform significantly worse – up to -7% and -4% (see (a) and (e) in Fig. 7-2) absolute test accuracy when compared to that of our method and uniform sampling, respectively. We postulate that the poor performance of these competing approaches predominantly stems from their inherently greedy acquisition of data points in a setting with significant randomness as a result of stochastic training and data augmentation, among other elements. In contrast, our approach has provably low-regret with respect to the data acquisition objective, and we conjecture that this property translates to consistent performance across varying configurations and data sets.



(a) FashionCNN (b) SVHN

Figure 7-3: The performance of our algorithm when instantiated with informativeness metrics from prior work compared to that of existing greedy approaches. Using AdaProd^{+} off-the-shelf with the corresponding metrics took only a few lines of code and lead to strict gains in performance on all evaluated benchmark data sets.

7.5.4 Boosting Prior Approaches

Despite the favorable results presented in the previous subsections, a lingering question still remains: to what extent is our choice of the loss as the uncertainty metric responsible for the effectiveness of our approach? More generally, can we expect our algorithm to perform well off-the-shelf – and even lead to improvements over greedy acquisition – with other choices for the loss? To investigate, we implement three variants of our approach, Ours (Uncertainty), Ours (Entropy), and Ours (BALD) that are instantiated with losses defined in terms of uncertainty, entropy, BALD metrics respectively, and compare to their corresponding greedy variants on SVHN and FashionMNIST. We note that the uncertainty loss corresponds to $\ell_{t,i} = \max_j f_t(x_i)_j \in [0, 1]$ and readily fits in our framework. For the Entropy and

BALD loss, the application is only slightly more nuanced in that we have to be careful that losses are bounded in the interval $[0, 1]$. This can be done by scaling the losses appropriately, e.g., by normalizing the losses for each round to be in $[0, 1]$ or scaling using a priori knowledge, e.g., the maximum entropy is $\log(k)$ for a classification task with k classes.

The performance of the compared algorithms are shown in Fig. 7-3. Note that for all evaluated data sets and metrics, our approach fares significantly better than its greedy counterpart. In other words, applying AdaProd$^+$ off-the-shelf with existing informativeness measures leads to strict improvements compared to their greedy variants. As seen from Fig. 7-3, our approach has potential to yield up to a 5% increase in test accuracy, and in all cases, achieves significantly lower test loss.

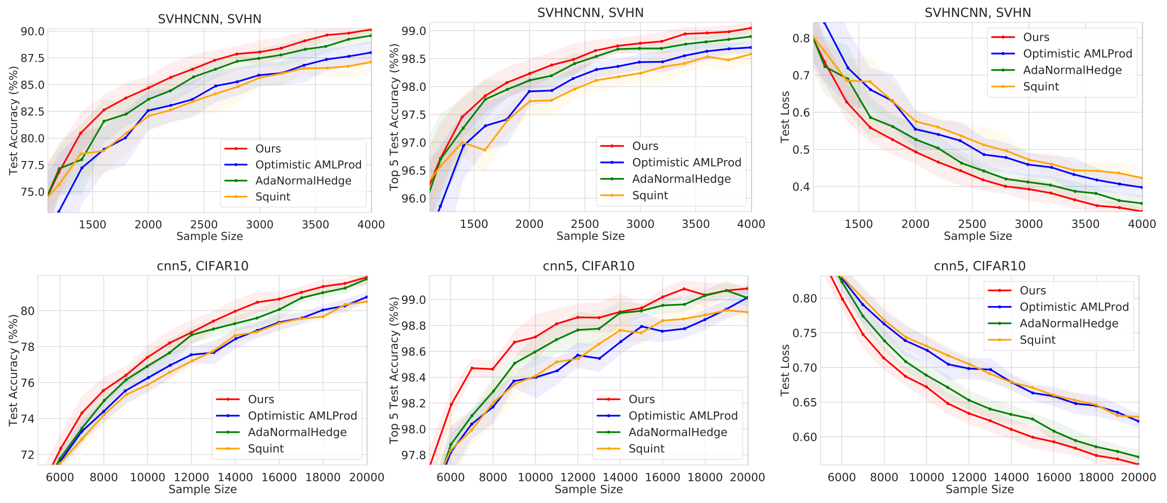


Figure 7-4: Comparisons with competing algorithms for learning with prediction advice on the SVHN (first row) and CIFAR10 (second row) data sets. In both scenarios, AdaProd* outperforms the compared algorithms, and significantly improves on its predecessor, Optimistic AMLProd, on both data sets and all evaluated metrics.

7.5.5 Comparison to Existing Expert Algorithms

In this section, we consider the performance of AdaProd$^+$ relative to that of state-of-the-art algorithms for learning with prediction advice. In particular, compare our approach to Optimistic AMLProd [WHL17], AdaNormalHedge(.TV) [LS15], and Squint(.TV) [KVE15] on the SVHN and CIFAR10 data sets. Fig. 7-4 depicts the results of our evaluations. As the figures show, our approach outperforms the compared approaches across both data sets in terms of all of the metrics considered. AdaNormalHedge comes closest to our method in terms of performance. Notably, the improved learning rate schedule (see Sec. 7.4) of AdaProd$^+$ compared to that of Optimistic AMLProd enables up to 3% improvements on test error on, e.g., SVHN and 2% on CIFAR10.

7.6 Proofs and Full Analytical Details

In this section, we present the full proofs and technical details of the claims made in Sec. 7.4. The outline of our analysis as follows. We first consider the *base* AdaProd$^+$ algorithm (shown as Alg. 12), which is nearly the same algorithm as AdaProd$^+$, with the exception that it is meant to be a general purpose algorithm for a setting with K experts (K is not necessarily equal to the number of points n). We show that this algorithm retains the original regret guarantees with respect to a stationary competitor of Adapt-ML-Prod.

We then consider the thought experiment where we use this standard version of our algorithm with the $K = nT$ sleeping experts reduction shown in [WHL17, GSVE14] to obtain guarantees for adaptive regret. This leads us to the insight (as in [LS15, KVE15]) that we do not need to keep track of the full set of K experts, and can instead keep track of a much smaller (but growing) set of experts in an efficient way without compromising the theoretical guarantees.

7.6.1 Recovering Optimistic Adapt-ML-Prod Guarantees for Alg. 12

We begin by observing that Alg. 12 builds on the standard Optimistic Adapt-ML-Prod algorithm [WHL17] by using a different initialization of the variables (Line 1) and upper bound imposed on the learning rates (as in Alg. 11, and analogously, in Line 10 of Alg. 12). Hence, the proof is has the same structure as [WHL17, GSVE14], and we prove all of the relevant claims (at times, in slightly different ways) below for

Algorithm 12 BASE ADAPROD$^+$

```
1: For all $i \in [K]$, $C_{i,0} \leftarrow 0$;  $\eta_{0,i} \leftarrow \sqrt{\log(K)/2}$; $w_{0,i} = 1$;  $\hat{r}_{1,i} = 0$;
2: for each round $t \in [T]$ do
3:     $p_{t,i} \leftarrow \eta_{t-1,i} w_{t-1,i} \exp(\eta_{t-1,i} \hat{r}_{t,i})$ for each $i \in [K]$
4:     $p_{t,i} \leftarrow p_{t,i} / \sum_{j \in [K]} p_{t,j}$ for each $i \in [K]$            ▷ Normalize
5:     Adversary reveals $\ell_t$ and we suffer loss $\tilde{\ell}_t = \langle \ell_t, p_t \rangle$
6:
7:     For all $i \in [K]$, set $r_{t,i} \leftarrow \tilde{\ell}_t - \ell_{t,i}$
8:     For all $i \in [K]$, set $C_{t,i} \leftarrow C_{t-1,i} + (\hat{r}_{t,i} - r_{t,i})^2$
9:     Get prediction $\hat{r}_{t+1} \in [-1,1]^K$ for next round (see Sec. 7.3.2)
10:    For all $i \in [K]$, update the learning rate
           $$\eta_{t,i} \leftarrow \min\left\{\eta_{t-1,i}, \frac{2}{3(1+\hat{r}_{t+1,i})}, \sqrt{\frac{\log(K)}{C_{t,i}}}\right\}$$
11:    For all $i \in [K]$, update the weights
           $$w_{t,i} \leftarrow \left(w_{t-1,i} \exp\left(\eta_{t-1,i} r_{t,i} - \eta_{t-1,i}^2 (r_{t,i} - \hat{r}_{t,i})^2\right)\right)^{\eta_{t,i}/\eta_{t-1,i}}$$
```

clarity and completeness. We proceed with our key lemma about the properties of the learning rates.

Lemma 44 (Properties of Learning Rates)**.** *Assume that the losses are bounded $\ell_t \in [0,1]^K$ and that the learning rates $\eta_{t,i}$ are set according to Line 10 of Alg. 12 for all $t \in [T]$ and $i \in [K]$, i.e.,*

$$\eta_{t,i} \leftarrow \min\left\{\eta_{t-1,i}, \frac{2}{3(1+\hat{r}_{t+1,i})}, \sqrt{\frac{\log(K)}{C_{t,i}}}\right\}.$$

Then, all of the following hold for all $t \in [T]$ and $i \in [K]$:

1. $\eta_{t,i}(r_{t+1,i} - \hat{r}_{t+1,i}) - \eta_{t,i}^2(r_{t+1,i} - \hat{r}_{t+1,i})^2 \le \log\left(1 + \eta_{t,i}(r_{t+1,i} - \hat{r}_{t+1,i})\right),$

2. $x \le x^{\eta_{t,i}/\eta_{t+1,i}} + 1 - \frac{\eta_{t+1,i}}{\eta_{t,i}} \quad \forall x \ge 0,$

3. $\frac{\eta_{t,i} - \eta_{t+1,i}}{\eta_{t,i}} \le \log(\eta_{t,i}/\eta_{t+1,i})$.

Proof. For the first claim, observe that the range of admissible values in the original Prod inequality [CBL06]

$$\forall x \geq -1/2 \quad x - x^2 \leq \log(1+x)$$

can be improved[3] to $\forall x \geq -2/3$. Now let $x = \eta_{t,i}(r_{t+1,i} - \hat{r}_{t+1,i})$, and observe that since $\ell_t \in [0,1]^K$, we have $r_{t+1,i} = \langle p_{t+1}, \ell_{t+1} \rangle - \ell_{t+1} \in [-1,1]$, and so

$$\begin{aligned} x &\geq \eta_{t,i}(-1 - \hat{r}_{t+1,i}) = -\eta_{t,i}(1 + \hat{r}_{t+1,i}) \\ &\geq -2/3, \end{aligned}$$

where in the last inequality we used the upper bound on $\eta_{t,i} \leq 2/(3(1+\hat{r}_{t+1,i})$ which holds by definition of the learning rates.

For the second claim, recall Young's inequality[4] which states that for non-negative a, b, and $p \geq 1$,

$$ab \leq a^p/p + b^{p/(p-1)}(1 - 1/p).$$

For our application, we set $a = x$, $b = 1$, and $p = \eta_{t,i}/\eta_{t+1,i}$. Observe that p is indeed greater than 1 since the learning rates are non-increasing over time (i.e., $\eta_{t+1,i} \leq \eta_{t,i}$ for all t and i) by definition. Applying Young's inequality, we obtain

$$x \leq x^{\eta_{t,i}/\eta_{t+1,i}}(\eta_{t+1,i}/\eta_{t,i}) + \frac{\eta_{t,i} - \eta_{t+1,i}}{\eta_{t,i}},$$

and the claim follows by the fact that the learning rates are non-increasing.

For the final claim, observe that the derivative of $\log(x)$ is $1/x$, and so by the mean value theorem we know that there exists $c \in [\eta_{t+1,i}, \eta_{t,i}]$ such that

$$\frac{\log(\eta_{t,i}) - \log(\eta_{t+1,i})}{\eta_{t,i} - \eta_{t+1,i}} = \frac{1}{c}.$$

[3]By inspection of the root of the function $g(x) = \log(1+x) - x + x^2$ closest to $x = -1/2$, which we know exists since $g(-1/2) > 0$ while $g(-1) < 0$.

[4]This follows by taking logarithms and using the concavity of the logarithm function.

Rearranging and using $c \leq \max\{\eta_{t,i}, \eta_{t+1,i}\} = \eta_{t,i}$, we obtain

$$\log(\eta_{t,i}/\eta_{t+1,i}) = \frac{\eta_{t,i} - \eta_{t+1,i}}{c} \geq \frac{\eta_{t,i} - \eta_{t+1,i}}{\eta_{t,i}}.$$

□

Having established our helper lemma, we now proceed to bound the regret with respect to a single expert as in [WHL17, GSVE14]. The main statement is given by the lemma below.

Lemma 45 (BASE ADAPROD$^+$ Static Regret Bound)**.** *The static regret of Alg. 12 with respect to any expert $i \in [K]$, $\sum_t r_{t,i}$, is bounded by*

$$\mathcal{O}\left(\log K + \log\log T + (\sqrt{\log K} + \log\log T)\sqrt{C_{T,i}}\right),$$

where $C_{T,i} = \sum_{t \in [T]} (r_{t,i} - \hat{r}_{t,i})^2$.

Proof. Consider $W_t = \sum_{i \in [K]} w_{t,i}$ to be the sum of *potentials* at round t. We will first show an upper bound on the potentials and then show that this sum is an upper bound on the regret of any expert (plus some additional terms). Combining the upper and lower bounds will lead to the statement of the lemma. To this end, we first show that the sum of potentials does not increase too much from round t to $t+1$. To do so, we apply (2) from Lemma 44 with $x = w_{t+1,i}$ to obtain for each $w_{t+1,i}$

$$w_{t+1,i} \leq w_{t+1,i}^{\eta_{t,i}/\eta_{t+1,i}} + \frac{\eta_{t,i} - \eta_{t+1,i}}{\eta_{t,i}}.$$

Now consider the first term on the right hand side above and note that

$$\begin{aligned}
w_{t+1,i}^{\eta_{t,i}/\eta_{t+1,i}} &= w_{t,i} \exp\left(\eta_{t,i} r_{t+1,i} - \eta_{t,i}^2 (r_{t+1,i} - \hat{r}_{t+1,i})^2\right) \\
&= w_{t,i} \exp(\eta_{t,i}\hat{r}_{t+1,i}) \exp\left(\eta_{t,i}(r_{t+1,i} - \hat{r}_{t+1,i}) - \eta_{t,i}^2 (r_{t+1,i} - \hat{r}_{t+1,i})^2\right) \\
&\leq w_{t,i} \exp(\eta_{t,i}\hat{r}_{t+1,i}) \left(1 + \eta_{t,i}(r_{t+1,i} - \hat{r}_{t+1,i})\right)
\end{aligned}$$

$$= w_{t,i}\eta_{t,i}\exp(\eta_{t,i}\hat{r}_{t+1,i})r_{t+1,i} + w_{t,i}\exp(\eta_{t,i}\hat{r}_{t+1,i})\underbrace{(1-\eta_{t,i}\hat{r}_{t+1,i})}_{\leq \exp(-\eta_{t,i}\hat{r}_{t+1,i})}$$

$$\leq \underbrace{w_{t,i}\eta_{t,i}\exp(\eta_{t,i}\hat{r}_{t+1,i})}_{\propto p_{t+1,i}} r_{t+1,i} + w_{t,i},$$

where we applied (1) of Lemma 44. As the brace above shows, the first part of the first expression on the right hand side is proportional to $p_{t+1,i}$ by construction (see Line 3 in Alg. 12). Recalling that $r_{t+1,i} = \langle p_{t+1}, \ell_{t+1}\rangle - \ell_{t+1,i}$, we have by dividing and multiplying by the normalization constant,

$$\sum_{i\in[K]} w_{t,i}\eta_{t,i}\exp(\eta_{t,i}\hat{r}_{t+1,i})r_{t+1,i} = \left(\sum_{i\in[K]} w_{t,i}\eta_{t,i}\exp(\eta_{t,i}\hat{r}_{t+1,i})\right)\sum_{i\in[K]} p_{t+1,i}r_{t+1,i} = 0,$$

since $\sum_{i\in[K]} p_{t+1,i}r_{t+1,i} = 0$. This shows that $\sum_{i\in[K]} w_{t+1,i}^{\eta_{t,i}/\eta_{t+1,i}} \leq \sum_{i\in[K]} w_{t,i} = W_t$.

Putting it all together and applying (3) from Lemma 44 to bound $\frac{\eta_{t,i}-\eta_{t+1,i}}{\eta_{t,i}}$, we obtain for the sum of potentials for $t \in [T]$:

$$W_{t+1} \leq W_t + \sum_{i\in[K]} \log(\eta_{t,i}/\eta_{t+1,i}).$$

A subtle issue is that for $t = 0$, we have $\eta_{0,i} = \sqrt{\log(K)/2}$ for all $i \in [n]$, which means that we cannot apply (1) of Lemma 44. So, we have to bound the change in potentials between W_1 and W_0. Fortunately, since this only occurs at the start, we can use the rough upper bound $\exp(x - x^2) = \exp(x(1-x)) \leq \exp(1/4) \leq 1.285$, which holds for all $x \in \mathbb{R}$, to obtain for $t = 0$

$$\begin{aligned}
w_{t+1,i}^{\eta_{t,i}/\eta_{t+1,i}} &\leq w_{t,i}\exp(\eta_{t,i}\hat{r}_{t+1,i})\exp(1/4) \\
&= w_{t,i}(1-\eta_{t,i}\hat{r}_{t+1,i} + \eta_{t,i}\hat{r}_{t+1,i})\exp(\eta_{t,i}\hat{r}_{t+1,i})\exp(1/4) \\
&\leq w_{t,i}(\exp(-\eta_{t,i}\hat{r}_{t+1,i}) + \eta_{t,i}\hat{r}_{t+1,i})\exp(\eta_{t,i}\hat{r}_{t+1,i})\exp(1/4) \\
&= \exp(1/4)w_{t,i} + \hat{r}_{t+1,i}\underbrace{\exp(1/4)w_{t,i}\eta_{t,i}\exp(\eta_{t,i}\hat{r}_{t+1,i})}_{\propto p_{t+1,i}},
\end{aligned}$$

where we used $1 - x \leq \exp(-x)$. Summing the last expression we obtained across all $i \in [K]$, we have for $t = 0$

$$\sum_{i \in [K]} w_{t+1,i}^{\eta_{t,i}/\eta_{t+1,i}} \leq \sum_{i \in [K]} \exp(1/4) w_{t,i},$$

where we used the fact that $\sum_{i \in [K]} \hat{r}_{t+1,i} w_{t,i} \eta_{t,i} \exp(\eta_{t,i} \hat{r}_{t+1,i}) = 0$ by definition of our predictions. Putting it all together, we obtain $W_1 \leq \exp(1/4) W_0 = \exp(1/4) K$ given that $W_0 = K$.

We can now unroll the recursion in light of the above to obtain

$$\begin{aligned}
W_T &\leq K \exp(1/4) + \sum_{t \in [T]} \sum_{i \in [K]} \log(\eta_{t,i}/\eta_{t+1,i}) \\
&= K \exp(1/4) + \sum_{i \in [K]} \sum_{t \in [T]} \log(\eta_{t,i}/\eta_{t+1,i}) \\
&= K \exp(1/4) + \sum_{i \in [K]} \log\left(\prod_{t+1 \in [T]} \eta_{t,i}/\eta_{t+1,i} \right) \\
&= K \exp(1/4) + \sum_{i \in [K]} \log(\eta_{0,i}/\eta_{T,i}) \\
&\leq K \left(\exp(1/4) + \log\left(\max_{i \in [K]} \sqrt{C_{T,i}} \right) \right) \\
&\leq K \left(\exp(1/4) + \log(4T)/2 \right).
\end{aligned}$$

Now, we establish a lower bound for W_t in terms of the regret with respect to any expert $i \in [K]$. Taking the logarithm and using the fact that the potentials are always non-negative, we can show via a straightforward induction (as in [GSVE14]) that

$$\log(W_T) \geq \log(w_{T,i}) \geq \eta_{T,i} \sum_{t \in [T]} (r_{t,i} - \eta_{t-1,i}(r_{t,i} - \hat{r}_{t,i})^2).$$

Rearranging, and using the upper bound on W_T from above, we obtain

$$\sum_{t\in[T]} r_{t,i} \le \eta_{T,i}^{-1} \log\left(K(1+\log(\max_{i\in[K]}\sqrt{1+C_{T,i}})\right) + \sum_{t\in[T]} \eta_{t-1,i}(r_{t,i}-\hat{r}_{t,i})^2. \tag{7.2}$$

For the first term in (7.2), consider the definition of $\eta_{T,i}$ and note that $\eta_{T,i} \ge \min\{1/3, \eta_{T-1,i}, \sqrt{\log(K)/(C_{T,i})}\}$ since $\hat{r}_{T+1,i} \le 1$. Now to lower bound $\eta_{T,i}$, consider the claim that $\eta_{t,i} \ge \min\{1/3, \sqrt{\log(K)/(C_{T,i})}\}$. Note that this claim holds trivially for the base cases where $t=0$ and $t=1$ since the learning rates are initialized to 1 and our optimistic predictions can be at most 1. By induction, we see that if this claim holds at time step t, we have for time step $t+1$

$$\begin{aligned}\eta_{t+1,i} &\ge \min\{1/3, \eta_{t,i}, \sqrt{\log(K)/(C_{t+1,i})}\} \ge \min\{1/3, \eta_{t,i}, \sqrt{\log(K)/(C_{T,i})}\}\\ &= \min\{\eta_{t,i}, \min\{1/3, \sqrt{\log(K)/(C_{T,i})}\}\}\\ &\ge \min\left\{\min\{1/3, \sqrt{\log(K)/(C_{T,i})}\}, \min\{1/3, \sqrt{\log(K)/(C_{T,i})}\}\right\}\\ &= \min\{1/3, \sqrt{\log(K)/(C_{T,i})}\}.\end{aligned}$$

Hence, we obtain $\eta_{T,i} \ge \min\{1/3, C_{T,i}\}$, and this implies that (by the same reasoning as in [GSVE14]) that

$$\eta_{T,i}^{-1} \log\left(K(1+\log(\max_{i\in[K]}\sqrt{C_{T,i}})\right) \le \mathcal{O}\left((\sqrt{\log K} + \log\log T)\sqrt{C_{T,i}} + \log K\right).$$

Now to bound the second term in (7.2), $\sum_{t\in[T]} \eta_{t-1,i}(r_{t,i}-\hat{r}_{t,i})^2$, we deviate from the analysis in [WHL17] in order to show that the improved learning schedule without the dampening term in the denominator suffices. To this end, we first upper bound $\eta_{t-1,i}$ as follows

$$\begin{aligned}\eta_{t-1,i} &\le \min\left\{\eta_{0,i}, \frac{2}{3(1+\hat{r}_{t,i})}, \sqrt{\frac{\log(K)}{C_{t-1,i}}}\right\}\\ &\le \min\left\{\eta_{0,i}, \sqrt{\frac{\log(K)}{C_{t-1,i}}}\right\}\end{aligned}$$

$$= \min\left\{\eta_{0,i}, \eta_{0,i}\sqrt{\frac{2}{C_{t-1,i}}}\right\}$$

where first inequality follows from the fact that the learning rates are monotonically decreasing, the second inequality from the definition of min, the last equality by definition $\eta_{0,i} = \sqrt{\log(K)/2}$.

By the fact that the minimum of two positive numbers is less than its harmonic mean[5], we have

$$\eta_{t-1,i} \leq \frac{2\sqrt{2}\eta_{0,i}}{\sqrt{2} + \sqrt{C_{t-1,i}}},$$

and so

$$\begin{aligned}
(r_{t,i} - \hat{r}_{t,i})^2 \eta_{t-1,i} &\leq c_{t,i}\frac{2\sqrt{2}\eta_{0,i}}{\sqrt{2} + \sqrt{C_{t-1,i}}} \\
&= 8\sqrt{2}\eta_{0,i}\frac{(c_{t,i}/4)}{\sqrt{2} + 2\sqrt{C_{t-1,i}/4}} \\
&\leq 4\sqrt{2}\eta_{0,i}\frac{(c_{t,i}/4)}{\sqrt{1/2 + C_{t-1,i}/4}},
\end{aligned}$$

where we used the subadditivity of the square root function in the last step.

Summing over all $t \in [T]$ and applying Lemma 14 of [GSVE14] on the scaled variables $c_{t,i}/4 \in [0,1]$, we obtain

$$\begin{aligned}
\sum_{t\in[T]} \eta_{t-1,i}(r_{t,i} - \hat{r}_{t,i})^2 &\leq 4\sqrt{2}\eta_{0,i}\sum_{t\in[T]}\sqrt{C_{T,i}} \\
&= 4\sqrt{C_{T,i}\log K},
\end{aligned}$$

where in the last equality we used the definition of $\eta_{i,0}$ and $C_{T,i} = \sum_{t\in[T]}(r_{t,i} - \hat{r}_{t,i})^2$ as before, and this completes the proof.

□

[5] $\min\{a,b\} = \min\{a^{-1}, b^{-1}\}^{-1} \leq (\frac{1}{2}(a^{-1} + b^{-1}))^{-1} = 2/(1/a + 1/b)$

7.6.2 Adaptive Regret

We now turn to establishing adaptive regret bounds via the sleeping experts reduction as in [WHL17, LS15] using the reduction of [GSVE14]. The overarching goal is to establish an adaptive bound for the regret of *every* time interval $[t_1, t_2], t_1, t_2 \in [T]$, which is a generalization of the static regret which corresponds to the regret over the interval $[1, T]$. To do so, in the setting of n experts as in the main document, the main idea is to run the base algorithm (Alg. 12) on $K = nT$ *sleeping* experts instead[6]. These experts will be indexed by (t, i) with $t \in [T]$ and $i \in [n]$. Moreover, at time step t, each expert (s, i) is defined to be awake if $s \leq t, i \in [n]$ **and** $\mathcal{I}_{t,i} = 1$ (the point has not yet been sampled, see Sec. 7.2), and the remaining experts will be considered sleeping. This will generate a probability distribution $\bar{p}_{t,(s,i)}$ over the awake experts. Using this distribution, at round t we play

$$p_{t,i} = \mathcal{I}_{t,i} \sum_{s \in [t]} \bar{p}_{t,(s,i)} / Z_t,$$

where $Z_t = \sum_{j \in [K]} \mathcal{I}_{t,j} \sum_{s' \in [t]} \bar{p}_{t,(s',j)}$.

The main idea is to construct losses to give to the base algorithm so that that at any point $t \in [T]$, each expert (s, i) suffers the interval regret from s to t (which is defined to be 0 if $s > t$), i.e., $\sum_{\tau=1}^{t} r_{\tau,(s,i)} = \sum_{\tau=s}^{t} r_{\tau,i}$. To do so, we build on the reduction of [WHL17] to keep track of both the sleeping experts from the sense of achieving adaptive regret and also the traditional sleeping experts regret with respect to only those points that are not yet labeled (as in Sec. 7.2). The idea is to apply the base algorithm (Alg. 12) with the modified loss vectors $\bar{\ell}_{t,(s,i)}$ for expert (s, i) as the original loss if the expert is awake, i.e., $\bar{\ell}_{t,(s,i)} = \ell_{t,i}$ **if** $s \leq t$ (original reduction in [WHL17]) **and** $\mathcal{I}_{t,i} = 1$ (the point has not yet been sampled), and $\bar{\ell}_{t,(s,i)} = \langle p_{t,i}, \ell_t \rangle$ otherwise. The prediction vector is defined similarly: $\bar{r}_{t,(s,i)} = \hat{r}_{t,i}$ if $s \leq t$, and 0 otherwise.

Note that this construction implies that the regret of the base algorithm with respect

[6]Note that this notion of sleeping experts is the same as the one we used for dealing with constructing a distribution over only the unlabeled data points remaining.

to the modified losses and predictions, i.e., $\bar{r}_{\tau,(s,i)} = \langle \bar{p}_{\tau,(s,i)}, \bar{\ell}_{\tau,(s,i)} \rangle$ is equivalent to $r_{\tau,i}$ for rounds $\tau > s$ where the expert is awake, and 0 otherwise. Thus,

$$\sum_{\tau \in [t]} \bar{r}_{\tau,(s,i)} = \sum_{\tau=s}^{t} r_{\tau,i},$$

which means that the regret of expert (s, i) with respect to the base algorithm is precisely regret of the interval $[s, t]$. Applying Lemma 45 to this reduction above (with $K = nT$) immediately recovers the adaptive regret guarantee of Optimisic Adapt-ML-Prod.

Lemma 46 (Adaptive Regret of Base AdaProd$^+$)**.** *For any $t_1 \leq t_2$ and $i \in [n]$, invoking Alg. 12 with the sleeping experts reduction described above ensures that*

$$\sum_{t=t_1}^{t_2} r_{t,i} \leq \hat{\mathcal{O}}\left(\log(K) + \sqrt{C_{t_2,(t_1,i)} \log(K)}\right),$$

where $C_{t_2,(t_1,i)} = \sum_{t=t_1}^{t_2} (r_{t,i} - \hat{r}_{t,i})^2$ and $\hat{\mathcal{O}}$ suppresses $\log T$ factors.

7.6.3 AdaProd$^+$ and Dynamic Regret

To put it all together, we relax to requirement of having to update and keep track of $K = NT$ experts and having to know T. To do so, observe that $\log(K) \leq \log(nT) \leq 2\log(n)$ since $T \leq n / \min_{t \in [T]} b_t \leq n$, where $b_t \geq 1$ is the number of new points to label at active learning iteration t. This removes the requirement of having to know T or the future batch sizes beforehand, meaning that we can set the numerator of $\eta_{t,(s,i)}$ to be $\sqrt{2\log(n)}$ instead of $\sqrt{\log(K)}$ (as in 11 in Sec. 7.3). Next, observe that in the sleeping experts reduction above, we have

$$p_{t,i} = \mathcal{I}_{t,i} \sum_{s \in [t]} \bar{p}_{t,(s,i)} / Z_t,$$

where $Z_t = \sum_{j \in [K]} \mathcal{I}_{t,j} \sum_{s' \in [t]} \bar{p}_{t,(s',j)}$. But for $s \leq t$ and $j \in [n]$ satisfying $\mathcal{I}_{t,j} = 1$, by definition of $\bar{p}_{t,(s,j)}$ and the fact that expert (s, j) is awake, we have $\bar{p}_{t,(s,j)} \propto$

$\eta_{t-1,(s,j)} w_{t-1,(s,j)} \exp(\eta_{t-1,(s,j)} \hat{r}_{t,j})$, and so the normalization constant cancels from the numerator (from $\bar{p}_{t,(s,j)}$) and the denominator (from the $\bar{p}_{t,(s',j)}$ in $Z_t = \sum_{j \in [K]} \mathcal{I}_{t,j} \sum_{s' \in [t]} \bar{p}_{t,(s',j)}$), leaving us with

$$p_{t,i} = \sum_{s \in [t]} \frac{\eta_{t-1,(s',j)} w_{t-1,(s',j)} \exp(\eta_{t-1,(s',j)} \hat{r}_{t,j})}{\gamma_t},$$

where $\gamma_t = \sum_{j \in [K]} \sum_{s' \in [t]} \eta_{t-1,(s',j)} w_{t-1,(s',j)} \exp(\eta_{t-1,(s',j)} \hat{r}_{t,j})$. Note that this corresponds precisely to the probability distribution played by AdaProd$^+$. Further, since AdaProd$^+$ does not explicitly keep track of the experts that are asleep, and only updates the potentials $W_{t,(s,i)}$ of those experts that are awake, AdaProd$^+$ mimics the updates of the reduction described above[7] involving passing of the modified losses to the base algorithm. Thus, we can conclude that AdaProd$^+$ leads to the same updates and generated probability distributions as the base algorithm for adaptive regret. This discussion immediately leads to the following lemma for the adaptive regret of our algorithm, very similar to the one established above except for $\log n$ replacing $\log T$ terms.

Lemma 47 (Adaptive Regret of AdaProd$^+$). *For any $t_1 \leq t_2$ and $i \in [n]$, Alg. 11 ensures that*

$$\sum_{t=t_1}^{t_2} r_{t,i} \leq \mathcal{O}\left(\log n + \log\log n + (\sqrt{\log n} + \log\log n)\sqrt{C_{t_2,(t_1,i)}}\right),$$

where $C_{t_2,(t_1,i)} = \sum_{t=t_1}^{t_2} (r_{t,i} - \hat{r}_{t,i})^2$.

Finally, applying the reduction from adaptive to dynamic regret (Theorem 4 of [LS15]) and bounding

$$(r_{t,i} - \hat{r}_{t,i})^2 \leq 2(\langle p_{t,i}, \ell_{t,i}\rangle - \langle p_{t,i}, \ell_{t-1,i}\rangle)^2 + 2(\ell_{t,i} - \ell_{t-1,i})^2 \leq 4\left\|\ell_t - \ell_{t-1}\right\|_\infty^2$$

by applying Hölder's inequality twice, we obtain Theorem 43 from Sec. 7.4.

[7]The only minor change is in the constant in the learning rate schedule of AdaProd$^+$ which has a $\sqrt{2\log(n)}$ term instead of $\sqrt{\log(nT)} \leq \sqrt{2\log(n)}$. This only affects the regret bounds by at most a factor of $\sqrt{2}$, and the reduction remains valid – it would be analogous to running Alg. 12 on a set of $n^2 \geq nT$ experts instead.

7.7 Implementation Details and Batch Sampling

To sample a batch of b points at step t, we first apply a capping algorithm to the probability distribution p_t computed by AdaProd$^+$ to generate $\tilde{p}_t$. This capping is more rigorously the information projection (i.e., projection with respect to the KL Divergence) to the capped simplex constraining the probabilities to be at most $\frac{1}{b}$ where b is the batch size [WK08]. This projection can be performed by a sort and iterative capping algorithm as described in [WK08, UNK10] ($\mathcal{O}(n \log n)$ time total). After obtaining $\tilde{p}_t$, we compute the scaled probability $\hat{p}_t = b\tilde{p}_t$ satisfying $\hat{p}_t \in [0,1]^n$ and $\sum_j \hat{p}_{t,j} = b$. Now, to sample b points according to $\hat{p}_t$, we use use the DepRound algorithm [UNK10] shown as Alg. 13, which takes $\mathcal{O}(n)$ time.

Algorithm 13 DepRound

```
Inputs: Subset size $m$, probabilities $p \in [0,1]^n$ such that $\sum_i p_i = m$
Output: set of indices $\mathcal{C} \subset [n]$ of size $m$
1: while $\exists i \in [n]$ such that $0 < p_i < 1$ do
2:     Pick $i, j \in [n]$ satisfying $i \neq j$, $0 < p_i < 1$, and $0 < p_j < 1$
3:     Set $\alpha = \min(1 - p_i, p_j)$ and $\beta = \min(p_i, 1 - p_j)$
4:     Update $p_i$ and $p_j$
5:         $(p_i, p_j) = \begin{cases} (p_i + \alpha, p_j - \alpha) & \text{with probability } \frac{\beta}{\alpha+\beta}, \\ (p_i - \beta, p_j + \beta) & \text{with probability } 1 - \frac{\beta}{\alpha+\beta}. \end{cases}$
6: $\mathcal{C} \leftarrow \{i \in [n] : p_i = 1\}$
7: return $\mathcal{C}$
```

In all of our empirical evaluations, the original probabilities generated by AdaProd$^+$ were already less than $1/b$, so the capping procedure did not get invoked. We conjecture that this may be a natural consequence of the active learning setting, where we are attempting to incrementally build up a small set of labeled data among a very large pool of unlabeled ones, i.e., $b \ll n$. This description also aligns with the relatively small batch sizes widely used in active learning literature as benchmarks [GSS19, AZK$^+$19, RXC$^+$20, SS17a, Mut19].

The focus of our work is not on the full extension of Adapt-ML-Prod [WHL17] to the batch setting, however, we summarize some of our ongoing and future work here for the interested reader. If we assume that the probabilities generated by AdaProd$^+$

satisfy $p_{t,i} \leq 1/b$, which is a mild assumption in the active learning setting as evidenced by our evaluations, our analysis suggests that the regret bounds we derived in the previous section scale by a factor of $\tilde{\mathcal{O}}(\sqrt{b})$ by Cauchy-Schwarz for the regret defined with respect to sampling a batch of b points at each time step. Our ongoing work includes relaxing the assumption $p_{t,i} \leq 1/b$ by building on techniques from prior work, such as by exploiting the inequalities associated with the Information (KL divergence) Projection as in [WK08] or capping the weight potential $w_{i,t}$ as in [UNK10] as soon as weights get too large (rather than modifying the probabilities). Based on our preliminary analysis, we conjecture that the same regret bound as the $\tilde{\mathcal{O}}(\sqrt{b})$-scaled bound holds asymptotically (ignoring $\log n$ and $\log T$ factors) for AdaProd$^+$ in the batch setting without this assumption. Formalizing this extension rigorously is an avenue for future work.

7.8 Experimental Setup & Additional Evaluations

In this section we (i) describe the experimental setup and detail hyper-parameters used for our experiments and (ii) provide additional evaluations and comparisons to supplement the results presented.

7.8.1 Setup

Table 7.1 depicts the hyperparameters used for training the network architectures used in our experiments. Given an active learning configuration (Option, $n_{\text{start}}, b, n_{\text{end}}$), these parameters describe the training process for each choice of Option as follows:

Incremental : we start the active learning process by acquiring and labeling n_{start} points chosen uniformly at random from the n unlabeled data points, and we train with the corresponding number of epochs and learning rate schedule listed in Table 7.1 under rows *epochs* and *lr decay*, respectively, to obtain θ_1. We then proceed as in Alg. 10 to iteratively acquire b new labeled points based on the Acquire function and *incrementally train* a model starting from the model from the previous iteration,

	FashionCNN	SVHNCNN	Resnet18	CNN5 (width=128)
loss	cross-entropy	cross-entropy	cross-entropy	cross-entropy
optimizer	Adam	Adam	SGD	Adam
epochs	60	60	80	60
epochs incremental	15	15	N/A	15
batch size	128	128	256	128
learning rate (lr)	0.001	0.001	0.1	0.001
lr decay	0.1@(50)	0.1@(50)	0.1@(30, 60)	0.1@(50)
lr decay incremental	0.1@(10)	0.1@(10)	N/A	0.1@(10)
momentum	N/A	N/A	0.9	N/A
Nesterov	N/A	N/A	No	N/A
weight decay	0	0	1.0e-4	0

Table 7.1: We report the hyperparameters used during training the convolutional architectures listed above corresponding to our evaluations on FashionMNIST, SVHN, CIFAR10, and ImageNet. except for the ones indicated in the lower part of the table. The notation $\gamma@(n_1, n_2, \ldots)$ denotes the learning rate schedule where the learning rate is multiplied by the factor γ at epochs $n_1, n_2, \ldots$ (this corresponds to MultiStepLR in PyTorch).

θ_{t-1}. This training is done with respect to the number of corresponding epochs and learning rate schedule shown in Table 7.1 under *epochs incremental* and *lr decay incremental*, respectively.

Scratch : the only difference relative to the Incremental setting is that rather than training the model starting from θ_{t-1}, we train a model from a randomly initialized network at each active learning iteration with respect to the training parameters under *epochs* and *lr decay* in Table 7.1.

Architectures We used the following convolutional networks on the specified data sets.

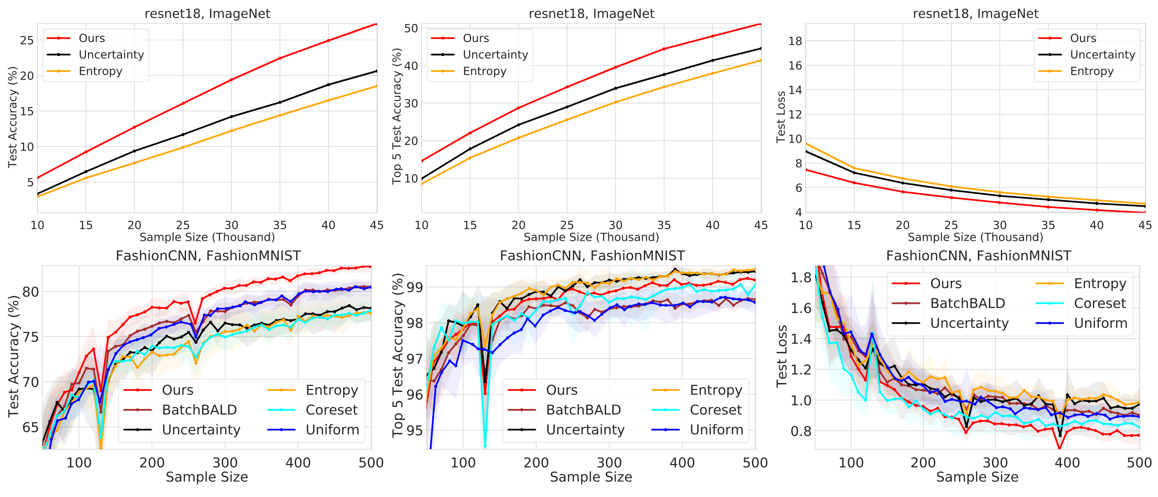
1. *FashionCNN [Pan18]* (for FashionMNIST): a network with 2 convolutional layers with batch normalization and max pooling, 3 fully connected layers, and one dropout layer with $p = 0.25$ in [Pan18]. This architecture achieves over 93% accuracy when trained with the whole data set.

2. *SVHNCNN [Che20]* (for SVHN): a small scale convolutional model very similar

to FashionCNN except there is no dropout layer.

3. *Resnet18 [HZRS16]* (for ImageNet): an 18 layer residual network with batch normalization.

4. *CNN5 [NKB+19]* (for CIFAR10): a 5-layer convolutional neural network with 4 convolutional layers with batch normalization. We used the width=128 setting in the context of [NKB+19].



(a) Top-1 test accuracy (b) Top-5 test accuracy (c) Top-5 test accuracy

Figure 7-5: Results for the data-starved configuration $(\textsc{Scratch}, 5k, 5k, 45k)$ on ImageNet (first row) and $(\textsc{Scratch}, 50, 10, 500)$ on FashionMNIST (second row). Shown from left to right are the results with respect to test accuracy, top-5 test accuracy, and test loss. Shaded region corresponds to values within one standard deviation of the mean.

Settings for experiments in Sec. 7.5 Prior to presenting additional results and evaluations in the next subsections, we specify the experiment configurations used for the experiments shown in the main document (Sec. 7.5). For the corresponding experiments in Fig. 7-1, we evaluated on the configuration $(\textsc{Scratch}, 10k, 20k, 110k)$ for ImageNet, $(\textsc{Scratch}, 500, 200, 4000)$ for SVHN, $(\textsc{Scratch}, 3k, 1k, 15k)$ for CIFAR10, and $(\textsc{Scratch}, 100, 300, 3000)$ for FashionMNIST. For the evaluations in Fig. 7-3, we used $(\textsc{Scratch}, 128, 96, 200)$ and $(\textsc{Scratch}, 128, 64, 2000)$ for FashionMNIST and SVHN, respectively. The models were trained with standard data normalization with respect to the mean and standard deviation of the entire training set. For ImageNet, we used random cropping to 224×224 and random horizontal flips for data augmentation; for the remaining data sets, we used random cropping to 32×32 (28×28 for FashionMNIST) with 4 pixels of padding and random horizontal flips.

All presented results were averaged over 10 trials with the exception of those for ImageNet[8], where we averaged over 3 trials due to the observed low variance in our results. We used the uncertainty loss metric as defined in Sec. 7.2 for all of the experiments presented in this work – with the exception of results related to boosting prior approaches (Fig. 7-3). The initial set of points and the sequence of random network initializations (one per sample size for the Scratch option) were fixed across all algorithms to ensure fairness.

7.8.2 Setting for Experiments in Sec. 7.5.5

In this subsection, we describe the setting for the evaluations in Sec. 7.5.5, where we compared the performance of AdaProd$^+$ to modern algorithms for learning with prediction advice. Since our approach is intended to compete with time-varying competitors (see Sec. 7.6), we compare it to existing methods that ensure low regret with respect to time-varying competitors (via adaptive regret). In particular, we compare our approach to the following algorithms:

1. **Optimistic AMLProd** [WHL17]: we implement the (stronger) variant of Optimistic Adapt-ML-Prod that ensures dynamic regret (outlined at the end of Sec. 3.3 in [WHL17]). This algorithm uses the sleeping experts reduction of [GSVE14] and consequently, requires initially creating $\tilde{n} = nT$ sleeping experts and updating them with similar updates as in our algorithm (except the cost of the t^{th} update is $\tilde{\mathcal{O}}(nT)$ rather than $\tilde{\mathcal{O}}(N_t t)$ as in ours). Besides the computational costs, we emphasize that the only true functional difference between our algorithm and Optimistic AMLProd lies in the thresholding of the learning rates (Line 10 in Alg. 11). In our approach, we impose the upper bound

[8]We were not able to run Coreset or BatchBALD on ImageNet due to resource constraints and the high computation requirements of these algorithms.

$\min\{\eta_{t-1,i}, 2/(3(1+\hat{r}_{t+1,i}))\}$ for $\eta_{t,i}$ for any $t \in [T]$, whereas [WHL17] imposes the (smaller) bound of $1/4$.

2. **AdaNormalHedge(.TV)** [LS15]: we implement the time-varying version of AdaNormalHedge, AdaNormalHedge.TV as described in Sec. 5.1 of [LS15]. The only slight modification we make in our setting where we already have a sleeping experts problem is to incorporate the indicator $\mathcal{I}_{t,i}$ in our predictions (as suggested by [LS15] in their sleeping experts variant). In other words, we predict[9] $p_{t,i} \propto \mathcal{I}_{t,i} \sum_{\tau=1}^{t} \frac{1}{\tau^2} w(R_{[\tau,t-1],i}, C_{[\tau,t-1]})$ rather than the original $p_{t,i} \propto \sum_{\tau=1}^{t} \frac{1}{\tau^2} w(R_{[\tau,t-1],i}, C_{[\tau,t-1]})$, where $R_{[t_1,t_1],i} = \sum_{t=t_1}^{t_2} r_{t,i}$ and $C_{[t_1,t_1],i} = \sum_{t=t_1}^{t_2} |r_{t,i}|$ (note that the definition of C is different than ours).

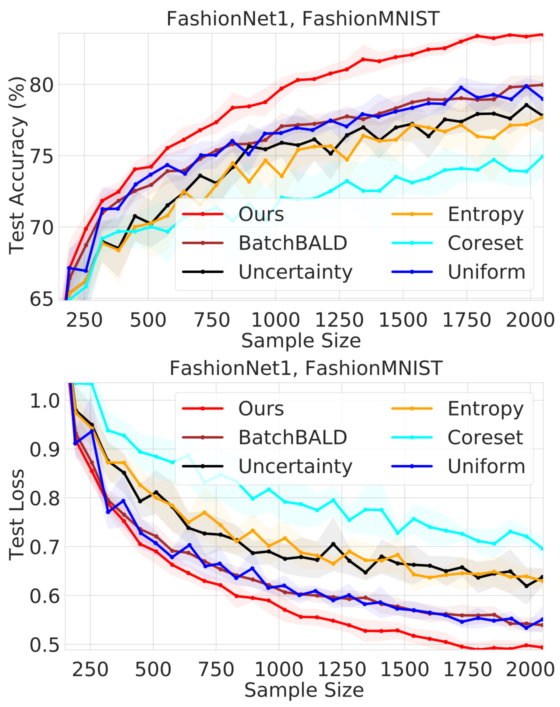
3. **Squint(.TV)** [KVE15]: Squint is a parameter-free algorithm like AdaNormalHedge in that it can also be extended to priors over an initially unknown number of experts. Hence, we use the same idea as in AdaNormalHedge.TV (also see [Luo17]) and apply the extension of the Squint algorithm for adaptive regret.

We used the $(\textsc{Scratch}, 500, 200, 400)$ and $(\textsc{Scratch}, 4000, 1000, 2000)$ configurations for the evaluations on the SVHN and CIFAR10 datasets, respectively.

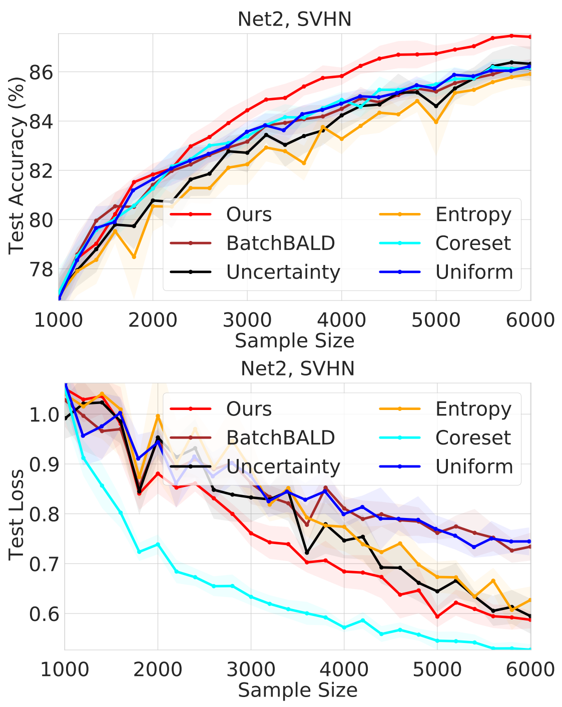
7.8.3 Results on Data-Starved Settings

Figure 7-5 shows the results of our additional evaluations on ImageNet and FashionMNIST in the *data-starved* setting where we begin with a very small (relatively) set of data points and can only query the labels of a small set of points at each time step. For both data sets, our approach outperforms competing ones in the various metrics considered – yielding up to 4% increase in test accuracy compared to the second-best performing method.

[9] We also implemented and evaluated the method with uniform prior over time intervals, i.e., $p_{t,i} \propto \mathcal{I}_{t,i} \sum_{\tau=1}^{t} w(R_{[\tau,t-1],i}, C_{[\tau,t-1]})$ (without the prior $\frac{1}{\tau^2}$), but found that it performed worse than with the prior in practice. The same statement holds for the Squint algorithm.



(a) FashionMNIST (Scratch, 128, 64, 2000)



(b) SVHN (Scratch, 500, 200, 6000)

Figure 7-6: Evaluations on the FashionNet and SVHNNet [AZK$^+$19, Ash21] architectures, which are different convolutional networks than those used in Sec. 7.5. Despite this architecture shift, our approach remains the overall top-performer on the evaluated data sets, even exceeding the relative performance of our approach on the previously used architectures.

7.8.4 Shifting Architectures

In this section, we consider the performance on FashionMNIST and SVHN when we change the network architectures from those used in Sec. 7.5. In particular, we conduct experiments on the *FashionNet* and *SVHNNet* architectures[10], convolutional neural networks that were used for benchmark evaluations in recent active learning work [AZK$^+$19, Ash21]. Our goal is to evaluate whether the performance of our algorithm degrades significantly when we vary the model we use for active learning.

Fig. 7-6 depicts the results of our evaluations using the same training hyperparameters as FashionCNN for FashionNet, and similarly, those for SVHNCNN for SVHNNet (see Table 7.1). For both architectures, our algorithm uniformly outperforms the competing approaches in virtually all sample sizes and scenarios; our approach achieves

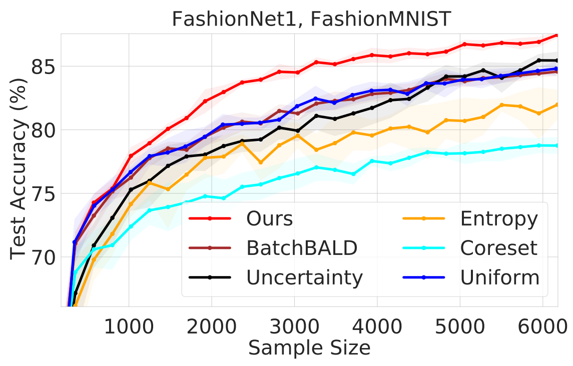
[10]Publicly available implementation and details of the architectures [AZK$^+$19, Ash21]: `https://github.com/JordanAsh/badge/blob/master/model.py` .

up to 5% and 2% higher test accuracy than the second best-performing method on FashionMNIST and SVHN, respectively. The sole exception is the SVHN test loss, where we come second to Coreset – which performs surprisingly well on the test loss despite having uniformly lower test accuracy than Ours on SVHN (top right, Fig. 7-6). Interestingly, the relative performance of our algorithm is even better on the alternate architectures than on the models used in the main body (compare Fig. 7-6to Fig. 7-1 of Sec. 7.5), where we performed only modestly better than competing approaches in comparison.

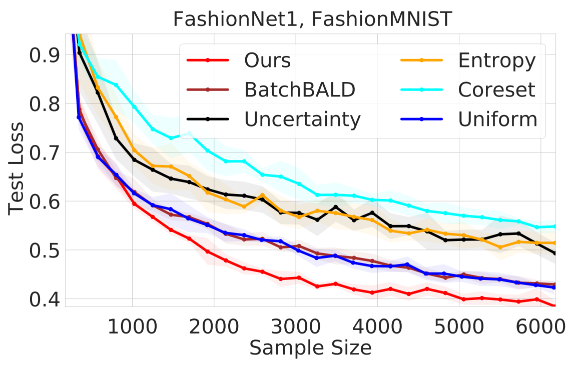
7.8.5 Robustness Evaluations on Shifted Architecture

Having shown that the resiliency of our approach for both data sets for the configuration shown in Fig. 7-6, we next investigate whether we can also remain robust to varying active learning configurations on alternate architectures. To this end, we fix the FashionMNIST dataset, the FashionNet architecture, and the Scratch option and consider varying the batch sizes and the initial and final number of labeled points. Most distinctly, we evaluated sample (active learning batch) sizes of 16, 96, and 224 points for varying sample budgets.

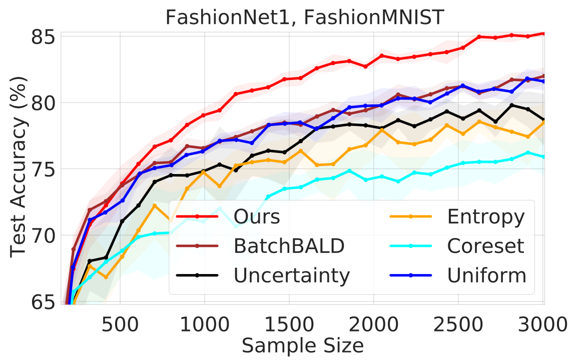
We present the results of our evaluations in Fig. 7-7, where each row corresponds to a differing configuration. For the first row of results corresponding to a batch size of 224, we see that we significantly (i.e., up to 3.5% increased test accuracy) outperform all compared methods for all sample sizes with respect to both test accuracy and loss. The same can be said for the second row of results corresponding to a batch size of 96, where we observe consistent improvements over prior work. For the smallest batch size 16 (last row of Fig. 7-7) and sampling budget (600), Ours still bests the compared methods, but the relative improvement is more modest (up to $\approx 1.5\%$ improvement in test accuracy) than it was for larger batch sizes. We conjecture that this is due to the fact that the sampling budget (600) is significantly lower than in the first two scenarios (up to 6000); in this data-starved regime, even a small set of uniformly sampled points from FashionMNIST is likely to help training since the points in the



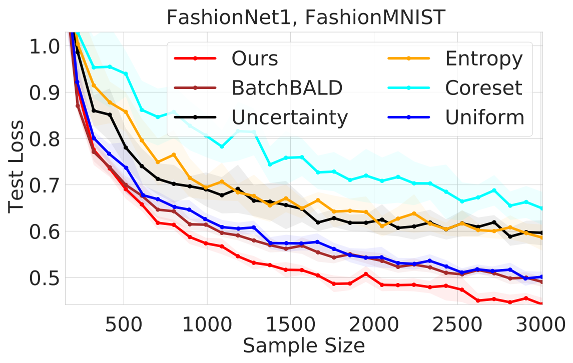
(a) Test accuracy (Scratch, 128, 224, 6000)



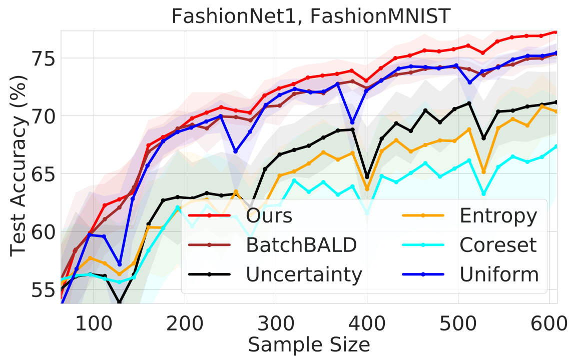
(b) Test loss (Scratch, 128, 224, 6000)



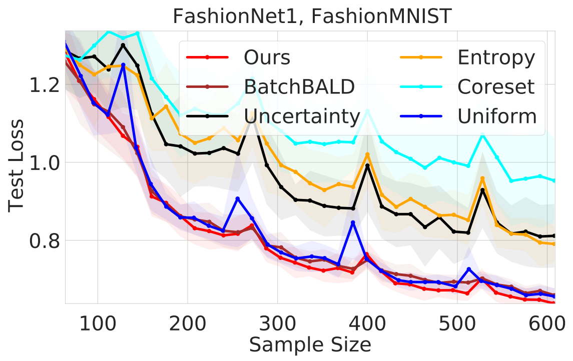
(c) Test accuracy (Scratch, 128, 96, 3000)



(d) Test loss (Scratch, 128, 96, 3000)



(e) Test accuracy (Scratch, 64, 16, 600)



(f) Test loss (Scratch, 64, 16, 600)

Figure 7-7: Evaluations with varying active learning configurations using the alternate FashionNet model trained on the FashionMNIST dataset.

small set of selected points will most likely be sufficiently distinct from one another.

7.9 Limitations and Concluding Remarks

In this chapter, we introduced AdaProd$^+$, an optimistic algorithm for prediction with expert advice that was tailored to the active learning. Our comparisons showed that AdaProd$^+$ fares better than Greedy and competing algorithms for learning with prediction advice. Nevertheless, from an online learning lens, AdaProd$^+$ can itself be improved so that it can be more widely applicable to active learning. For one, we currently require the losses to be bounded to the interval $[0, 1]$. This can be achieved by scaling the losses by their upper bound $\ell_{\max}$ (as we did for the Entropy metric), however, this quantity $\ell_{\max}$ may not be available beforehand for all loss metrics. Ideally, we would want a scale-free algorithm that works with any loss to maximize the applicability of our approach.

In the same vein, in future work we plan to extend the applicability of our framework to clustering-based active-learning, e.g., Coreset [SS17a] and Badge [AZK$^+$19], where it is more difficult to quantify what the loss should be for a given clustering. One idea could be to define the loss of an unlabeled point to be proportional to its distance – with respect to some metric – to the center of the cluster that the point belongs to (e.g., ≈ 0 loss for points near a center). However, it is not clear that the points near the cluster center should be prioritized over others as we may want to prioritize cluster outliers too. It is also not clear what the distance metric should be, as the Euclidean distance in the clustering space may be ill-suited. In future work, we would like to explore these avenues and formulate losses capable of appropriately reflecting each point's importance with respect to a given clustering.

Overall, our empirical evaluations on large-scale real-world data sets and architectures substantiate the reliability of our approach in outperforming naive uniform sampling and show that it leads to consistent and significant improvements over existing work. Our analysis and evaluations suggest that AdaProd$^+$ can be applied off-the-shelf with existing informativeness measures to improve upon greedy selection, and likewise can scale with future advances in uncertainty or informativeness quantification. In this regard, we hope that this work can contribute to the advancement of reliably effective active learning approaches that can one day become an ordinary part of every practitioner's toolkit, just like Adam and SGD have for stochastic optimization.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation (NSF) under Awards 1723943 and 1526815, Office of Naval Research (ONR) Grant N00014-18-1-2830, Microsoft, and JP Morgan Chase.

Chapter 8

Conclusion

In this chapter, we summarize the contributions of the work presented in this thesis, discuss the lessons learned along the way, and outline avenues for future work.

8.1 Thesis Summary

In this thesis, we presented provable, sampling-based algorithms to enable efficient AI and alleviate its exponentially increasing computational and environmental costs. The unifying theme of our work is to use representative samples to accelerate existing algorithms, enable real-time performance, and generally alleviate the high computational, labeling, and environmental costs of modern AI. Our theoretical guarantees and empirical evaluations suggest that we can efficiently train and obtain compact AI models using importance sampling at the cost of a relatively small approximation error.

We began our exposition with a motivating application of sampling in Chapter 3 in the context of reachability analysis. The problem was to compute the set of reachable states of an AI agent, e.g., an autonomous vehicle, by evaluating the initial set of states using a computationally-expensive reachability function. Given that the set of initial states was infinite in size, we had to settle for an approximation in the form of evaluating a small, finite subset. This motivated a careful subset selection procedure that would approximate the evaluation on the entire set. We showed that provably approximate coverage of reachable states can be efficiently computed using a packing of $\mathcal{X}$ with a carefully chosen packing parameter and demonstrated its effectiveness in practice.

In Chapter 4, we focused on subsampling training points in order to accelerate SVM training. By bridging k-means clustering with the SVM objective function, we established tight bounds on the relative importance of each data point that can be computed efficiently. Combining our analysis with the sensitivity framework led to compact coresets on real-world data sets.

Next, in Chapter 5, we considered downsizing data sets to compressing large deep learning models for efficiency and ease of deployment. Here, our attempt to apply the sensitivity framework proved to be theoretically futile due to the worst-case nature of the sensitivity definition – which involves a supremum over queries. Our key insight was that the queries in the setting of model compression are data points drawn from a distribution and that we can exploit this probabilistic nature by considering the sensitivity of most likely inputs, rather than a worst-case one across the entire support of the data distribution. This led to the development of the *empirical sensitivity* framework, which we used to prune the edges of large neural network models. We established bounds on the size of the pruned network as a function of the desired approximation accuracy. Our bounds also led to an automatic budget allocation procedure that emphasized the preservation of parameters in important layers over others that could be pruned more aggressively. We presented evaluations on large-scale models and data sets and showed that our approach can prune up to 90% of a network's parameters without any degradation in test accuracy.

We extended the empirical sensitivity framework for parameter pruning in Chapter 6, where we considered the problem of pruning entire neurons and filters rather than just weights of a neural network. Here, we observed that building on the empirical sensitivity definition enabled *structured pruning* of filters and neurons, leading to efficient models that can perform fast inference. As a prominent use case, we demonstrated the benefits of structured pruning on a real-world end-to-end autonomous driving scenario, where real-time inference was crucial to the safe navigation of the vehicle. We conclude Chapter 6 with a discussion of the next frontier in network pruning, including the incorporation of other objectives beyond test accuracy alone, like robustness to noisy or adversarial data.

We returned the problem of subsampling training data in the context of deep learning in Chapter 7, this time with the additional constraint that the data is unlabeled. This reduces to the active learning problem where the goal is to select the smallest, most informative set of inputs to label. Here, we built on previous work on online learning to formulate a low-regret approach framework that can be used with any desired notion of informativeness to robustify and improve upon state-of-the-art greedy methods.

8.2 Lessons Learned

Here, we discuss valuable insights we obtained along the way during our work on sampling for scalable AI.

8.2.1 Theory-guided Practice & Practice-guided Theory

The algorithms introduced in this thesis were generally based on constructive proofs that translated to concrete sampling procedures in practice. For example, for reachability analysis, the idea of having a diverse set of states (a covering) from the input space was motivated by the Lipschitz condition. Moreover, the packing distance was entirely determined by the theory. Similarly, the importance-sampling procedure was entirely based on the analytical sensitivity upper bounds and corresponding concentration analysis. Similar statements can be said for the development of the methods for pruning (Chapters 5 & 6) and for active learning (Chapter 7). However, this is not the full story. In fact, many of our insights were enabled by our observations in practice which culminated in advancements in theory.

In the work presented here, for example, our analysis on coresets for SVMs was based on the intuition we gained through practical toy examples. Similarly, the automatic budget allocation scheme in our pruning work was inspired by the empirical observation that some layers required more samples to preserve the output than other layers. The same can be said for the development of our low-regret active learning approach in Chapter 7, where our observations in typical active learning settings motivated the development of a low-regret approach that could patch the shortcomings of existing greedy methods.

8.2.2 Toy Examples and Lower Bounds First

When we first started our work on coresets for Support Vector Machines (Chapter 4), the sensitivity framework had at the time been shown to be quite effective in obtaining provably small data summaries for popular methods in machine learning like M-estimator clustering (e.g., k-means), among others [Fel19, BLK17, BFL16, HCB16]. This was generally enabled by an elegant bound on the sensitivity of each point using a triangle inequality argument with an approximate (bicriteria) solution. These recent advances led us to believe that an analogous approach should likewise work for the SVM problem, or at the very least, to think that compact coresets for SVMs should exist. So, we began to work towards bounding the sensitivities in a similar way. The goal was to obtain quantities sharp enough so that their sum would be sublinear in n just like in the case of k-means coresets — a necessary condition for sublinear-sized coresets according to Bernstein's inequality.

After a significant amount of work, it seemed that no matter what we tried, we could not achieve bounds that were tight enough. We tried techniques far beyond those used for k-means coresets, including the use of convex hulls and other geometric methods to bound the sensitivity, but nothing seemed to suit our needs. The main challenge was that the SVM objective function did not respect even an approximate version of the triangle inequality, unlike that of k-means. Only after a lot of work and effort on our attempt did we consider the possibility that sublinear SVM coresets may not exist for all data sets. We began to think we were too optimistic given the success and ease of constructing coresets for other problems like clustering. We began to think, could we instead prove a *lower* bound on the sensitivities rather than an upper bound? After some brainstorming for pathological examples, we were able to indeed prove that the sum of sensitivities is lower bounded by $\Omega(n)$ for a particular set of points, implying that in the worst case, no small coresets exist even for linear SVMs.

This taught us a valuable lesson in the importance of considering toy scenarios and the existence of small data summaries first. In our case, the pathological example enabling the lower bound ultimately guided our algorithm design. It led to insights on the sensitivity upper bound, which enabled small data summaries in practice due to the structure of real-world data. This also taught us the lesson that despite the worst-case pathological inputs, data-dependent bounds may enable compact summaries in practice under real-world conditions. We believe that this lesson will also be relevant for future work and developments in scalable AI, and would encourage further researchers to think about the existence of coresets using toy scenarios or lower bounds prior to diving deep into their construction.

8.2.3 Importance of Algorithms with Non-worst-case Performance

Throughout our work, we learned the importance of methods with non-worst-case guarantees and that two algorithms with the same worst-case guarantees can in fact perform very differently in practice. This was most prominently made clear to us during our work on active learning (Chapter 7). Here, our initial attempt to regret minimization was to use the well-known HEDGE algorithm with an instance-independent[1] learning rate of $\eta = \sqrt{\log n / T}$. This tuning of the learning rate leads to a bound of $\mathcal{O}(\sqrt{T \log n})$ on the static regret in the classical setting which is only a

[1]Here instance-independent means that η and the regret bound are independent of the actual losses $\ell_1, \ldots, \ell_T$ that the algorithm receives.

logarithmic factor away from a matching minimax lower bound. Given this guarantee and the popularity of HEDGE, we had high expectations for its performance in practice. However, we quickly found out that the algorithm performed quite poorly: it was simply too conservative in its plays.

This was because the algorithm was based on a worst-case analysis where the losses $\|\ell_t\|_\infty$ is equal to 1 at every time step t. To obtain better bounds in practice, we needed a tightened analysis with the instance-dependent losses in mind. So, we used an adaptive version of HEDGE with a time-varying learning rate $\eta_t = \sqrt{\log n / \sum_{t \in [T]} \|\ell_t\|_\infty^2}$ which leads to the bound $\mathcal{O}\big(\sqrt{\log n \sum_{t \in [T]} \|\ell_t\|_\infty^2}\big)$ [Ora19]. Although in the worst-case this bound matched that of the previous algorithm, it performed substantially better in practice. Going further, we built on previous methods in online learning to develop ADAPROD$^+$, an approach that removes the pessimistic infinity norm from the losses and guarantees second-order bounded regret on a per-expert basis. This led to even further improvements in practical performance, despite the regret bound matching that of our initial attempt with HEDGE in the worst-case. This led to the further improved results presented in Chapter 7. Overall, we envision that future development in scalable AI can focus on instance-dependent bounds and algorithms that leverage the structure inherent in real-world problems.

8.3 Future Work

The work in this thesis paves the way for future research in provable algorithms for scalable AI. To this end, one avenue for future work is to combine the data and model compression techniques presented here in a synergistic way. For example, combining the work in Chapters 6 for model compression and 7 for active learning would simultaneously enable the use of smaller sets of labeled data and compact models for fast and label-efficient AI models. This would also have implications for ease of deployment to resource-constrained platforms and real-time inference. We plan to investigate this direction in future work. Likewise, we would like to extend

Sensitivity-informed Provable Pruning (SiPP) to sparsify the parameters of other Machine Learning models such as SVMs. At a high level, our vision for future work is to build on the techniques presented in this thesis so that reliable and efficient sampling algorithms can one day become a part of the Machine Learning toolkit, just as Stochastic Gradient Descent and Adam have for deep learning.

Bibliography

[AANR17] Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems*, pages 3180–3189, 2017.

[AAR20] Alireza Aghasi, Afshin Abdi, and Justin Romberg. Fast convex pruning of deep neural networks. *SIAM Journal on Mathematics of Data Science*, 2(1):158–188, 2020.

[ACH$^+$95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[AD14] M. Althoff and J. M. Dolan. Online Verification of Automated Road Vehicles Using Reachability Analysis. *IEEE Transactions on Robotics*, 30(4):903–918, 2014.

[ADH$^+$19] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332, 2019.

[ADI06] R. Alur, T. Dang, and F. Ivančić. Predicate Abstraction for Reachability Analysis of Hybrid Systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 5(1):152–199, 2006.

[AGJT14] A. Alam, A. Gattami, K. H. Johansson, and C. J. Tomlin. Guaranteeing safety for heavy duty vehicle platooning: Safe set computations and experimental evaluations. *Control Engineering Practice*, 24:33–41, 2014.

[AGNZ18] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*, 2018.

[ALHS18] Menachem Adelman, Kfir Y Levy, Ido Hakimi, and Mark Silberstein. Faster neural network training with approximate tensor operations. *arXiv preprint arXiv:1805.08079*, 2018.

[Alt15] M. Althoff. An introduction to CORA 2015. In *Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[APB$^+$18] Alexander Amini, Liam Paull, Thomas Balch, Sertac Karaman, and Daniela Rus. Learning steering bounds for parallel autonomous systems. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[AS10] Pankaj K Agarwal and R Sharathkumar. Streaming algorithms for extent problems in high dimensions. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1481–1489. Society for Industrial and Applied Mathematics, 2010.

[AS17] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017.

[Ash21] Jordan Ash. Badge. https://github.com/JordanAsh/badge, 2021.

[ASSR19] Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. *arXiv preprint arXiv:1910.02600*, 2019.

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[AZK$^+$19] Jordan T Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. *arXiv preprint arXiv:1906.03671*, 2019.

[AZLL19] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in Neural Information Processing Systems 32*, pages 6158–6169. Curran Associates, Inc., 2019.

[BC03] Mihai Badoiu and Kenneth L Clarkson. Smaller core-sets for balls. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 801–802. Society for Industrial and Applied Mathematics, 2003.

[BF04] A. Bhatia and E. Frazzoli. Incremental Search Methods for Reachability Analysis of Continuous and Hybrid Systems. In *International Workshop on Hybrid Systems: Computation and Control*, 2004.

[BFL16] Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *arXiv preprint arXiv:1612.00889*, 2016.

[BGM$^+$14] D. Bresolin, L. Geretti, R. Muradore, P. Fiorini, and T. Villa. Verification of Robotic Surgery Tasks by Reachability Analysis: A Comparison of Tools. In *Euromicro Conference on Digital System Design*, 2014.

[BGMMS21] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623, 2021.

[BGOFG20] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems 2020*, pages 129–146, 2020.

[BHPI02] Mihai Bādoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 250–257, 2002.

[BKML17] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017.

[BLFR21] Cenk Baykal, Lucas Liebenwein, Dan Feldman, and Daniela Rus. Low-regret active learning. *arXiv preprint arXiv:2104.02822*, 2021.

[BLG$^+$18] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2018.

[BLG$^+$19a] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2019.

[BLG$^+$19b] Cenk Baykal*, Lucas Liebenwein*, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Sipping neural networks: Sensitivity-informed provable pruning of neural networks. *arXiv preprint arXiv:1910.05422*, 2019.

[BLG$^+$21] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Sipping neural networks: Sensitivity-informed provable pruning of neural networks. *arXiv preprint arXiv:1910.05422*, 2021.

[BLK17] Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017.

[BM20] Adam Bohr and Kaveh Memarzadeh. The rise of artificial intelligence in healthcare applications. In *Artificial Intelligence in Healthcare*, pages 25–60. Elsevier, 2020.

[BMR$^+$20] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[BTFR20] Cenk Baykal*, Murad Tukan*, Dan Feldman, and Daniela Rus. On coresets for support vector machines. In *International Conference on Theory and Applications of Models of Computation*, pages 287–299. Springer, 2020.

[BTFR21] Cenk Baykal, Murad Tukan, Dan Feldman, and Daniela Rus. On coresets for support vector machines. In *Special Issue Invite to Theoretical Computer Science (TCS), Under Review*, 2021.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[CAP$^+$19] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.

[CÁS13] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An Analyzer for Non-Linear Hybrid Systems. In *International Conference on Computer Aided Verification*, 2013.

[CBL06] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

[CCB$^+$16] Anna Choromanska, Krzysztof Choromanski, Mariusz Bojarski, Tony Jebara, Sanjiv Kumar, and Yann LeCun. Binary embeddings with structured hashed projections. In *International Conference on Machine Learning*, pages 344–353, 2016.

[CEM+15] Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 163–172. ACM, 2015.

[CGJ+00] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *International Conference on Computer Aided Verification*, 2000.

[CGL93] E. Clarke, O. Grumberg, and D. Long. Verification Tools for Finite-State Concurrent Systems. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*, 1993.

[Che20] A. Chen. Pytorch playground. https://github.com/aaron-xichen/pytorch-playground, 2020.

[CHW12] Kenneth L Clarkson, Elad Hazan, and David P Woodruff. Sublinear optimization for machine learning. *Journal of the ACM (JACM)*, 59(5):23, 2012.

[CK99] A. Chutinan and B.H. Krogh. Verification of Polyhedral-Invariant Hybrid Automata Using Polygonal Flow Pipe Approximations. In *International workshop on hybrid systems: computation and control*, 1999.

[CK08] P. Cheng and V. Kumar. Sampling-based Falsification and Verification of Controllers for Continuous Dynamic Systems. *The International Journal of Robotics Research*, 27(11-12):1232–1245, 2008.

[Cla10] Kenneth L Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. *ACM Transactions on Algorithms (TALG)*, 6(4):63, 2010.

[CMF+19] Beidi Chen, Tharun Medini, James Farwell, Sameh Gobriel, Charlie Tai, and Anshumali Shrivastava. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. *arXiv preprint arXiv:1903.03129*, 2019.

[CPI18] Miguel A Carreira-Perpinán and Yerlan Idelbayev. "learning-compression" algorithms for neural net pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8532–8541, 2018.

[CSM+15] X. Chen, S. Schupp, I.B. Makhlouf, E. Ábrahám, G. Frehse, and S. Kowalewski. A Benchmark Suite for Hybrid Systems Reachability Analysis. In *NASA Formal Methods Symposium*, 2015.

[CT15] M. Chen and C. J. Tomlin. Exact and efficient hamilton-jacobi reachability for decoupled systems. In *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*, pages 1297–1303. IEEE, 2015.

[CXS19] Beidi Chen, Yingchen Xu, and Anshumali Shrivastava. Fast and accurate stochastic gradient estimation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[CYF$^+$15] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865, 2015.

[Dav59] P.J. Davis. Leonhard Euler's Integral: A Historical Profile of the Gamma Function. *The American Mathematical Monthly*, 66(10):849–869, 1959.

[DCP17] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4860–4874, 2017.

[DDS$^+$09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[DJP$^+$20] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[DP18] Melanie Ducoffe and Frederic Precioso. Adversarial active learning for deep networks: a margin based approach. *arXiv preprint arXiv:1802.09841*, 2018.

[DSD$^+$13] Misha Denil, Babak Shakibi, Laurent Dinh, Marc Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26*, pages 2148–2156, 2013.

[Dub57] L. E. Dubins. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[DZB$^+$14] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *CoRR*, abs/1404.0736, 2014.

[EFG16] S. M. Erlien, S. Fujita, and J. C. Gerdes. Shared steering control using safe envelopes for obstacle avoidance and vehicle stability. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):441–451, 2016.

[EGSD20] Sayna Ebrahimi, William Gan, Kamyar Salahi, and Trevor Darrell. Minimax active learning. *arXiv preprint arXiv:2012.10467*, 2020.

[EWR+15] K. Edelberg, D. Wai, J. Reid, E. Kulczycki, and P. Backes. Workspace and Reachability Analysis of a Robotic Arm for Sample Cache Retrieval from a Mars Rover. In *AIAA SPACE Conference and Exposition*, 2015.

[FCTS15] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry. Reach-avoid problems with time-varying dynamics, targets and constraints. In *Proceedings of the 18th international conference on hybrid systems: computation and control*, pages 11–20. ACM, 2015.

[Fed69] H. Federer. *Geometric Measure Theory.* Springer, 1969.

[Fel19] Dan Feldman. Core-sets: An updated survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1335, 2019.

[FJY+19] Xin Feng, Youni Jiang, Xuejiao Yang, Ming Du, and Xin Li. Computer vision algorithms and hardware implementations: A survey. *Integration*, 69:309–320, 2019.

[FL11] Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578. ACM, 2011.

[GBR18] Stephanie Gil, Cenk Baykal, and Daniela Rus. Resilient multi-agent consensus using wi-fi signals. *IEEE control systems letters*, 3(1):126–131, 2018.

[GEH19] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[GEY17] Yonatan Geifman and Ran El-Yaniv. Deep active learning over the long tail. *arXiv preprint arXiv:1711.00941*, 2017.

[GHH+11] J.H. Gillula, G.M. Hoffmann, H. Huang, M.P. Vitus, and C.J. Tomlin. Applications of Hybrid Reachability Analysis to Robotic Aerial Vehicles. *The International Journal of Robotics Research*, 30(3):335–354, 2011.

[GIG17] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pages 1183–1192. PMLR, 2017.

[GIM$^+$99] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, 1999.

[GJ09] Bernd Gärtner and Martin Jaggi. Coresets for polytope distance. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 33–42. ACM, 2009.

[GKDP20] Noah Gamboa, Kais Kudrolli, Anand Dhoot, and Ardavan Pedram. Campfire: Compressible, regularization-free, structured sparse training for hardware accelerators. *arXiv preprint arXiv:2001.03253*, 2020.

[GKPS06] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM (JACM)*, 53(3):324–360, 2006.

[GO05] R. Geraerts and M.H. Overmars. Reachability Analysis of Sampling Based Planners. In *Robotics and Automation (ICRA)*, 2005.

[GP17] Bolin Gao and Lacra Pavel. On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv preprint arXiv:1704.00805*, 2017.

[GSS19] Daniel Gissin and Shai Shalev-Shwartz. Discriminative active learning. *arXiv preprint arXiv:1907.06347*, 2019.

[GSVE14] Pierre Gaillard, Gilles Stoltz, and Tim Van Erven. A second-order bound with excess losses. In *Conference on Learning Theory*, pages 176–196. PMLR, 2014.

[GYC16] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[GYK$^+$21] Amir Gholami, Zhewei Yao, Sehoon Kim, Michael Mahoney, and Kurt Keutzer. Ai and memory wall. *RiseLab Medium Post*, 2021.

[HCB16] Jonathan H Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. *arXiv preprint arXiv:1605.06423*, 2016.

[HHWT97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A Model Checker for Hybrid Systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2):110–122, 1997.

[HKD$^+$18] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2234–2240. AAAI Press, 2018.

[HKS11] Elad Hazan, Tomer Koren, and Nati Srebro. Beating sgd: Learning svms in sublinear time. In *Advances in Neural Information Processing Systems*, pages 1233–1241, 2011.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[HMD15] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[HPM04] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300, 2004.

[HPRZ07] Sariel Har-Peled, Dan Roth, and Dav Zimak. Maximum margin coresets for active and noise tolerant learning. In *IJCAI*, pages 836–841, 2007.

[HZDS$^+$18] Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G Edward Suh. Channel gating neural networks. *arXiv preprint arXiv:1805.12549*, 2018.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IHM$^+$16] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and$<$ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[Imm15] F. Immler. Verified Reachability Analysis of Continuous Systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2015.

[IN07] Piotr Indyk and Assaf Naor. Nearest-neighbor-preserving embeddings. *ACM Transactions on Algorithms (TALG)*, 3(3):31–es, 2007.

[IRS$^+$15] Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*, 2015.

[IST$^+$19] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *arXiv preprint arXiv:1905.02175*, 2019.

[JNM$^+$19] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. *arXiv preprint arXiv:1912.02178*, 2019.

[Joa06] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

[JVZ14] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[KALL17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[KDSA14] J. Kapinski, J.V. Deshmukh, S. Sankaranarayanan, and N. Arechiga. Simulation-guided Lyapunov Analysis for Hybrid Dynamical Systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, 2014.

[KH$^+$09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical Report*, 2009.

[KNMS10] Robert Kleinberg, Alexandru Niculescu-Mizil, and Yogeshwer Sharma. Regret bounds for sleeping experts and bandits. *Machine learning*, 80(2):245–272, 2010.

[KPY$^+$15] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[KVAG19] Andreas Kirsch, Joost Van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *arXiv preprint arXiv:1906.08158*, 2019.

[KVE15] Wouter M Koolen and Tim Van Erven. Second-order quantile methods for experts and combinatorial games. In *Conference on Learning Theory*, pages 1155–1175. PMLR, 2015.

[LAT18] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBC$^+$21] Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. Lost in pruning: The effects of pruning neural networks beyond test accuracy. *arXiv preprint arXiv:2103.03014*, 2021.

[LBG$^+$18] Lucas Liebenwein, Cenk Baykal, Igor Gilitschenski, Sertac Karaman, and Daniela Rus. Sampling-based approximation algorithms for reachability analysis with provable guarantees. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[LBL$^+$20] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

[LBS$^+$19] Harry Lang*, Cenk Baykal*, Najib Abu Samra, Tony Tannous, Dan Feldman, and Daniela Rus. Deterministic coresets for stochastic matrices with applications to scalable sparse pagerank. In *International Conference on Theory and Applications of Models of Computation*, pages 410–423. Springer, 2019.

[LC07] GaÃĢlle Loosli and StÃŠphane Canu. Comments on the“core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 8(Feb):291–301, 2007.

[LDS90] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[LFKF17] Mario Lucic, Matthew Faulkner, Andreas Krause, and Dan Feldman. Training mixture models at scale via coresets. *arXiv preprint arXiv:1703.08110*, 2017.

[LGGT19] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5623–5632, 2019.

[Li20] Chuan Li. Openai's gpt-3 language model: A technical overview, 2020.

[Lic13] M. Lichman. UCI machine learning repository, 2013.

[Lie21] L. Liebenwein. Provable pruning. https://github.com/lucaslie/provable_pruning, 2021.

[LJL$^+$19] Shaohui Lin, Rongrong Ji, Yuchao Li, Cheng Deng, and Xuelong Li. Toward compact convnets via structure-sparsity regularized filter pruning. *IEEE transactions on neural networks and learning systems*, 31(2):574–588, 2019.

[LKD$^+$16] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[LL16] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2554–2564. IEEE, 2016.

[LLS01] Yi Li, Philip M Long, and Aravind Srinivasan. Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences*, 62(3):516–527, 2001.

[LRH$^+$17] S.B. Liu, H. Roehm, C. Heinzemann, I. Lütkebohle, J. Oehlerking, and M. Althoff. Provably Safe Motion of Mobile Robots in Human Environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[LRLZ17] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2178–2188, 2017.

[LS10] Michael Langberg and Leonard J Schulman. Universal ε-approximators for integrals. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 598–607. SIAM, 2010.

[LS15] Haipeng Luo and Robert E Schapire. Achieving all with no parameters: Adanormalhedge. In *Conference on Learning Theory*, pages 1286–1304, 2015.

[LSB$^+$20] Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020.

[LSV$^+$17] L. Liebenwein, W. Schwarting, C.-I. Vasile, J. DeCastro, J. Alonso-Mora, S. Karaman, and D. Rus. Compositional and contract-based verification for autonomous driving on road networks. *International Symposium on Robotics Research (ISRR)*, 2017.

[Luo17] Haipeng Luo. Interval Regret. https://haipeng-luo.net/courses/CSCI699/lecture9.pdf, 2017. [Online; accessed November-2020].

[LUW17] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3290–3300, 2017.

[M+65] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[MBT05] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin. A Time-Dependent Hamilton-Jacobi Formulation of Reachable Sets for Continuous Dynamic Games. *Transactions on Automatic Control*, 50(7):947–957, 2005.

[MCL+19] Yuzhe Ma, Ran Chen, Wei Li, Fanhua Shang, Wenjian Yu, Minsik Cho, and Bei Yu. A unified approximation framework for compressing and accelerating deep neural networks. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 376–383. IEEE, 2019.

[MF14] Alireza Makhzani and Brendan Frey. Winner-take-all autoencoders. *arXiv preprint arXiv:1409.2752*, 2014.

[MIGR19] Sepideh Mahabadi, Piotr Indyk, Shayan Oveis Gharan, and Alireza Rezaei. Composable core-sets for determinant maximization: A simple near-optimal algorithm. In *International Conference on Machine Learning*, pages 4254–4263. PMLR, 2019.

[MMK18] Alejandro Molina, Alexander Munteanu, and Kristian Kersting. Core dependency networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI). AAAI Press Google Scholar*, 2018.

[MMR19] Konstantin Makarychev, Yury Makarychev, and Ilya Razenshteyn. Performance of johnson-lindenstrauss transform for k-means and k-medians clustering. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1027–1038, 2019.

[MMS+17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[MMS+18] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

[MMT$^+$19] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[Mor16] F. Morgan. *Geometric Measure Theory: A Beginner's Guide*. Academic Press, 2016.

[MS18] Alexander Munteanu and Chris Schwiegelshohn. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI-Künstliche Intelligenz*, 32(1):37–53, 2018.

[MSB$^+$17] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[MSSW18] Alexander Munteanu, Chris Schwiegelshohn, Christian Sohler, and David P Woodruff. On coresets for logistic regression. *arXiv preprint arXiv:1805.08571*, 2018.

[MTK$^+$16] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[Mun14] J. Munkres. *Topology*. Pearson Education, 2014.

[Mus18] Christopher Paul Musco. *Faster linear algebra for data analysis and machine learning*. PhD thesis, Massachusetts Institute of Technology, 2018.

[Mut19] Hariank Muthakana. *Uncertainty and diversity in deep active image classification*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 2019.

[MW19] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pages 4646–4655. PMLR, 2019.

[MY15] Mehryar Mohri and Scott Yang. Accelerating optimization via adaptive prediction. *arXiv preprint arXiv:1509.05760*, 2015.

[NBS18] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A PAC-bayesian approach to spectrally-normalized margin bounds for neural networks. In *International Conference on Learning Representations*, 2018.

[NKB$^+$19] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019.

[NKB$^+$20] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2020.

[NKT14] Manu Nandan, Pramod P Khargonekar, and Sachin S Talathi. Fast svm training using approximate extreme points. *Journal of Machine Learning Research*, 15(1):59–98, 2014.

[NLB$^+$19] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019.

[NR14] Vikram Nathan and Sharath Raghvendra. Accurate streaming support vector machines. *arXiv preprint arXiv:1412.2485*, 2014.

[NWC$^+$11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *Google Research*, 2011.

[Ope19] OpenAI. Ai and compute. https://openai.com/blog/ai-and-compute, 2019.

[Ora19] Francesco Orabona. A modern introduction to online learning. *arXiv preprint arXiv:1912.13213*, 2019.

[Pan18] Pankajj. Fashion MNIST with PyTorch. https://www.kaggle.com/pankajj/fashion-mnist-with-pytorch-93-accuracy, 2018. [Online; accessed November-2020].

[PBB$^+$19] Brian Plancher, Camelia D Brumar, Iulian Brumar, Lillian Pentecost, Saketh Rama, and David Brooks. Application of approximate matrix multiplication to neural networks and distributed slam. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2019.

[PBD18] Anastasia Pentina and Shai Ben-David. Multi-task {K} ernel {L} earning based on {P} robabilistic {L} ipschitzness. In *Algorithmic Learning Theory*, pages 682–701. PMLR, 2018.

[PF05] Daniel Pérez Palomar and Javier Rodríguez Fonollosa. Practical algorithms for a family of waterfilling solutions. *IEEE transactions on Signal Processing*, 53(2):686–695, 2005.

[PGL$^+$21] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[PGM$^+$19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[PKV09] E. Plaku, L.E. Kavraki, and M.Y. Vardi. Falsification of LTL Safety Properties in Hybrid Systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2009.

[PLA$^+$15] O. Porges, R. Lampariello, J. Artigas, A. Wedler, C. Borst, and M. A. Roa. Reachability and Dexterity: Analysis and Applications for Space Robotics. In *Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA)*, 2015.

[PPF03] V. S. Patsko, S. G. Pyatko, and A. A. Fedotov. Three-dimensional reachability set for a nonlinear control system. *Journal of Computer and Systems Sciences International*, 42(3):320–328, 2003.

[PyT20a] PyTorch contributors. Non-linear activations (weighted sum, nonlinearity). https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity, 2020. [Online; accessed 4-June-2020].

[PyT20b] PyTorch contributors. Unfold. https://pytorch.org/docs/master/generated/torch.nn.Unfold.html, 2020. [Online; accessed 9-June-2020].

[RDIV09] Piyush Rai, Hal Daumé III, and Suresh Venkatasubramanian. Streamed learning: one-pass svms. *arXiv preprint arXiv:0908.0572*, 2009.

[RDS$^+$15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet

Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RFC20] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *International Conference on Learning Representations*, 2020.

[RM51] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[RS13] Alexander Rakhlin and Karthik Sridharan. Online learning with predictable sequences. In *Conference on Learning Theory*, pages 993–1019. PMLR, 2013.

[RXC$^+$20] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A survey of deep active learning. *arXiv preprint arXiv:2009.00236*, 2020.

[SAH$^+$20] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[SAM19] Hamid Shayestehmanesh, Sajjad Azami, and Nishant A Mehta. Dying experts: Efficient algorithms with optimal regret bounds. *arXiv preprint arXiv:1910.13521*, 2019.

[SAMR18] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

[SAN20] Taiji Suzuki, Hiroshi Abe, and Tomoaki Nishimura. Compression based bound for non-compressed network: unified generalization error analysis of large compressible deep neural network. In *International Conference on Learning Representations*, 2020.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[Ser95] H. Seraji. Reachability Analysis for Base Placement in Mobile Manipulators. *Journal of Field Robotics*, 12(1):29–43, 1995.

[SGV20] Aadirupa Saha, Pierre Gaillard, and Michal Valko. Improved sleeping bandits with stochastic action sets and adversarial rewards. In *International Conference on Machine Learning*, pages 8357–8366. PMLR, 2020.

[SHK$^+$14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[SHM$^+$16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[SL14] Jacob Steinhardt and Percy Liang. Adaptivity and optimism: An improved exponentiated gradient algorithm. In *International Conference on Machine Learning*, pages 1593–1601. PMLR, 2014.

[SMR$^+$19] Li Shen, Laurie R Margolies, Joseph H Rothstein, Eugene Fluder, Russell McBride, and Weiva Sieh. Deep learning to improve breast cancer detection on screening mammography. *Scientific reports*, 9(1):1–12, 2019.

[Sri99] Aravind Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM Journal on Computing*, 29(2):648–670, 1999.

[SRMW17] Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting. In *International Conference on Machine Learning*, pages 3299–3308. PMLR, 2017.

[SS17a] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

[SS17b] Ryan Spring and Anshumali Shrivastava. A new unbiased and efficient class of lsh-based samplers and estimators for partition function computation in log-linear models. *arXiv preprint arXiv:1703.05160*, 2017.

[SS17c] Ryan Spring and Anshumali Shrivastava. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 445–454, 2017.

[SS20] Ryan Spring and Anshumali Shrivastava. Mutual information estimation using lsh sampling. In *IJCAI*, pages 2807–2815, 2020.

[SSK15] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*, pages 3088–3096, 2015.

[SSSSC11] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[SZ14a] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[SZ14b] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Tab09] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach.* Springer Science & Business Media, 2009.

[TBFR20] Murad Tukan, Cenk Baykal, Dan Feldman, and Daniela Rus. On coresets for support vector machines. In *International Conference on Theory and Applications of Models of Computation*, pages 287–299. Springer, 2020.

[TCBM20] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *arXiv preprint arXiv:2002.08347*, 2020.

[TFF08] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

[TGLM20] Neil C Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F Manso. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 2020.

[TKC05] Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6(Apr):363–392, 2005.

[TKK07] Ivor W Tsang, Andras Kocsor, and James T Kwok. Simpler core vector machines with enclosing balls. In *Proceedings of the 24th international conference on Machine learning*, pages 911–918. ACM, 2007.

[TXZ$^+$15] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[UNK10] Taishi Uchiya, Atsuyoshi Nakamura, and Mineichi Kudo. Algorithms for adversarial bandit problems with multiple plays. In *International Conference on Algorithmic Learning Theory*, pages 375–389. Springer, 2010.

[VBC$^+$19] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[Ver16] Roman Vershynin. High-dimensional probability. *An Introduction with Applications*, 2016.

[VV98] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[WHL17] Chen-Yu Wei, Yi-Te Hong, and Chi-Jen Lu. Tracking the best expert in non-stationary stochastic environments. *arXiv preprint arXiv:1712.00578*, 2017.

[WK08] Manfred K Warmuth and Dima Kuzmin. Randomized online pca algorithms with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9(Oct):2287–2320, 2008.

[Wu16] Y. Wu. Yale ECE598, Lecture Notes: Information-Theoretic Methods in High-Dimensional Statistics, March 2016.

[WWW$^+$16] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[XD10] Z. Xue and R. Dillmann. Efficient Grasp Planning with Reachability Analysis. In *International Conference on Intelligent Robotics and Applications*, 2010.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[YCRM17] Jiyan Yang, Yin-Lam Chow, Christopher Ré, and Michael W Mahoney. Weighted sgd for \ell_p regression with randomized preconditioning. *arXiv preprint arXiv:1502.03571*, 2017.

[YLC$^+$17] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. *Preprint at https://arxiv. org/abs/1711.05908*, 2017.

[YLC$^+$18] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[YLLW18] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.

[YLWT17] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

[ZG17] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[ZLW$^+$17] Liang Zhao, Siyu Liao, Yanzhi Wang, Jian Tang, and Bo Yuan. Theoretical properties for neural networks with weight matrices of low displacement rank. *CoRR*, abs/1703.00144, 2017.

[ZVA$^+$18] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Non-vacuous generalization bounds at the imagenet scale: a pac-bayesian compression approach. In *International Conference on Learning Representations*, 2018.

[ZYZ$^+$18] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 184–199, 2018.

[ZZWT19] Yuefu Zhou, Ya Zhang, Yanfeng Wang, and Qi Tian. Accelerate cnn via recursive bayesian pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3306–3315, 2019.