ABSTRACT

Title of Dissertation: IMPROVED TRAINING OF DEEP NETWORKS FOR COMPUTER VISION

Abhay Kumar Yadav
Doctor of Philosophy, 2021

Dissertation Directed by: Professor David W. Jacobs
Department of Computer Science

Deep neural networks have become the state-of-the-art tool to solve many computer vision problems. However, these algorithms face a lot of computational and optimization challenges. For example, a) the training of deep networks is not only computationally intensive but also requires a lot of manual effort and parameter turning, b) for some particular use-cases, such as adversarial deep networks, it's even challenging to optimize to achieve good or stable performance. In this dissertation, we address these challenges by targeting the following closely related problems.

First, we focus on the problem of automating the step-size and decay parameters in the training of deep networks. Classical stochastic gradient methods for optimization rely on noisy gradient approximations that become progressively less accurate as iterates approach a solution. The large noise and small signal in the resulting gradients makes it difficult to use them for adaptive step-size selection. We propose alternative "big batch" SGD schemes that adaptively grow the batch size over time to maintain a nearly constant signal-to-noise ratio in the gradient approximation. The

high fidelity gradients enable automated learning rate selection and do not require stepsize decay. Also, big batches can be parallelized across many machines, reducing training time and efficiently utilizing resources.

Second, in the similar pursuit of automated and efficient training of deep networks, we explore the use of L-BFGS for large-scale machine learning applications. L-BFGS, a very successful second-order optimization method for convex problems, is not even considered an algorithm of choice for these applications. Recent work has shown that a stochastic version of L-BFGS can perform comparably to the current state-of-the-art solvers such as SGD or Adam for classification tasks. However, their work is limited to deep networks that do not use batch normalization. Since batch normalization is a de facto standard and essential for good performance in practical industrial-strength deep networks, this renders their work somewhat less practical. To this end, we propose a new variant of stochastic L-BFGS, which can work for deep networks that use batch normalization. We demonstrate the effectiveness of the proposed method by providing both convergence analysis and empirical results on standard deep networks and image classification. The proposed method outperforms Adam and existing approaches for L-BFGS by a large margin (10% in some cases) and is comparable to carefully tuned SGD for some cases. Although we do not surpass the generalization performance of carefully tuned SGD, this work marks another significant step towards considering L-BFGS as an effective algorithm for large-scale machine learning.

Third, we propose a stable training method for adversarial deep networks. Adversarial neural networks solve many important problems in data science, but are notoriously difficult to train. These difficulties come from the fact that optimal weights for adversarial nets correspond to saddle points, and not minimizers, of the loss function. The alternating stochastic gradient methods typically used for such problems do not reliably converge to saddle points, and when convergence does happen it is often highly sensitive to learning rates. We propose a simple modification of stochastic gradient descent that stabilizes adversarial networks. We show, both in theory and practice, that the proposed method reliably converges to saddle points, and is stable with a wider range of training parameters than a non-prediction method. This makes adversarial networks less likely to “collapse”, and enables faster training with larger learning rates.

Finally, we propose to efficiently compute the Neural Tangent Kernel (NTK) by establishing (both theoretically and empirically) that for most practical use-cases, NTK can be replaced by the well-known Laplace kernel, which is computationally much cheaper than NTK. NTK is interesting and important because it can reasonably well approximate the solution of a massively overparameterized neural network that is trained using SGD. So, another advantage of this finding is that one can get more insight into infinite width real neural networks by analyzing the Laplace kernel, which has a simple closed form (which NTK does not have).

IMPROVED TRAINING OF DEEP NETWORKS FOR COMPUTER VISION

by

Abhay Kumar Yadav

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

Dedication

To the existence

Acknowledgments

I owe my gratitude to all the people who have made this dissertation possible and because of whom my graduate experience has been one that I will cherish forever. I apologize in advance to whom I could have inadvertently left out in this acknowledgement.

First and foremost, I would like to thank my advisor, Professor David W. Jacobs, an excellent research mentor who has provided more than everything I could ask for from an advisor. He gave me a lot of freedom to explore my research, take internships, and spend time with my family. He always helped me, whether its academia or personal, or professional. He would suggest that I might be wrong but would still encourage me to see my own mistake without forcing his views, which I believe is an integral part of the learning process. I think not only he is a wonderful mentor but also a wonderful human being. Probably only thanking him would not be sufficient.

It is an honor to have Professor Abhinav Shrivastava, Professor Behtash Babadi, Professor Ramani Duraiswami, and Professor Tom Goldstein on my dissertation committee. I am thankful to them for serving on my committee and providing insightful and diverse suggestions to improve this dissertation. I also would like to thank Professor Rama Chellappa and Professor Tom Goldstein for their insightful and valuable suggestions during my dissertation proposal.

I want to extend my special thank to Professor Tom Goldstein for the intense and fruitful research discussions that expanded my understanding of seeing things from different perspectives, which led to many publications. I would like to thank Dr. Carlos, Dr. Soham De, Dr. Sohil Shah, Dr. Rajeev Ranjan, and my research colleagues at UMD for making my Ph.D. years memorable.

I would like to thank Professor Ronen Basri for his insightful discussion of deep networks and kernel methods. I would also like to thank my collaborators at Weizmann Institute of Science, Israel. It was a pleasure to work with them all.

I would like to acknowledge the help and support from the staff members from Computer Science Department, Prof. Jeff Foster, Jeniffer Story, Tom Hurst, Jodie Gray, Sharron McElroy, and Janice Perrone. I would like to thank UMIACS staff members for supporting my GPU workstations.

Last but not least, I owe my deepest thanks to my family for their unconditional love and support. They always stood behind me and helped me get through all those challenging times.

Lastly, thank you all and myself!

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Deep neural networks [2, 3] have become indispensable tools in computer vision and emerged as a clear winner in many important applications such as image classification [2, 4, 5], object detection [6, 7], Generative Adversarial Networks [8, 9], etc. Despite their enormous success, one faces a lot of computational and optimization challenges to apply deep networks [10] successfully. For example: a) For image classification, one has to empirically decide many hyper-parameters such as the number of layers, filter size, batch size, training algorithm, learning rate schedule, and the list goes on. And deciding even a single factor from this list is quite challenging due to the large manual effort and computation involved, b) Adversarial deep networks [9, 11, 12] solve many significant problems in computer vision, but are difficult to train because their training involves alternating minimization and maximization operations on a multi-task objective function, c) Recent works have shown that infinite width deep networks are equivalent to Neural Tangent Kernels (NTK) which makes it possible to approximate the function learned by an infinite width deep network using an equivalent NTK-regressor. However, memory and time requirements for NTK computation grow linearly with the number of layers (because of its recursive definition), which makes it expensive especially for large-scale data. This hinders the further exploration and application of NTK for large-scale datasets to some extent (For example, 5x time and memory computation is needed for a 5 layered infinite width deep network equivalent NTK).

In this work, we target the problems mentioned above using principled optimization methods for deep networks. In particular, Chapter 2 focuses on efficient and automated first-order training methods that are empirically comparable or better performing than other standard methods, but without requiring an expert user to choose learning rates and decay parameters. In a similar direction, in Chapter 3, we propose a new version of L-BFGS, a second-order method for the training of deep networks for classification tasks. In Chapter 4, we present a simple modification to the alternating stochastic gradient descent (SGD) method, called a prediction step, that improves stability in the training of adversarial deep networks. In Chapter 5, we establish that NTK is equivalent to the classic standard Laplace kernel for most of the practical use-cases, thus for those use-cases, one can compute the Laplace kernel, which is computationally cheap as compared to NTK. Finally, in Chapter 6, we summarize the current work and suggest some possible future directions.

1.1 Big Batch SGD: First-order method

In Chapter 2, we study a "big batch" strategy for the SGD methods [13] and its impact in the efficient and automated training of deep networks. It is common practice to progressively reduce the step size (learning rate) while using SGD methods, especially while training deep networks [14, 15], but finding the right learning rate and decay schedule is manual and inefficient. Rather than decreasing the step size, we let the minibatch *adaptively grow* in size to maintain a constant signal-to-noise ratio of the gradient approximation. This approach has two main advantages: 1) Big batches can be parallelized across many machines, reducing training time and efficiently using resources, 2) Using this batching strategy, we can keep the step size constant, or let it adapt using a simple Armijo backtracking line search [16], making the method much more manageable to automate and requiring much less user oversight than classical small batch SGD. Recent results [17] have shown that large *fixed* batch sizes fail to find good minimizers for non-convex problems like deep neural networks. Adaptively increasing the batch size over time overcomes this limitation: intuitively, in the initial iterations, the increased stochasticity (corresponding to smaller batches) can help land the iterates near a good minimizer, and larger batches, later on, can increase the speed of convergence towards this minimizer. The proposed automated methods are empirically comparable or better performing than other standard methods, but without requiring an expert user to choose learning rates and decay parameters.

1.2 Frozen-Batch L-BFGS (FbLBFGS): Second-order method

In Chapter 3, we explore and promote the use of second-order methods for large-scale machine learning applications in a stochastic setting. L-BFGS, a second-order optimization method, has had much success and been celebrated in convex optimization because of "ease of training" (For example, one can use line search to choose learning rates that need almost no parameter tuning, and it converges reliably even for poorly conditioned objectives where SGD fails). But it is not even considered an algorithm of choice for large-scale machine learning applications, and hence is not explored compared to existing first-order methods such as SGD and Adam. Recent work [18] has shown that a new stochastic version of L-BFGS can get comparable or even better results than SGD or Adam for classification tasks. However, that can only work when batch normalization (BatchNorm) is not used. Since BatchNorm is a de facto standard and important for good performance in practical deep networks, this still limits the use of L-BFGS in these scenarios.

This motivates and warrants further exploration of L-BFGS for more practical deep nets.

In particular, we propose to freeze and reuse the same batch twice for a stable curvature pair update in the stochastic L-BFGS method for deep networks with BatchNorm. This approach is generic enough to be applied to any L-BFGS method that employs a finite gradient differencing approach, making it even more appealing as one can borrow existing tricks and trades of the existing L-BFGS variants. We also provide convergence proof and empirical results on the classification task. The proposed approach performs consistently better than existing L-BFGS approaches and Adam by a large margin (more than 10% generalization accuracy for some problems) and also achieves results that are on par with carefully manually tuned SGD (SOTA). Although the proposed method is not able to surpass the state-of-the-art results achieved by SGD, this work marks another significant step towards making L-BFGS competitive for large-scale machine learning.

1.3 Stabilizing Adversarial Nets With Prediction Methods

In Chapter 4, we propose a simple modification of stochastic gradient descent that stabilizes adversarial networks. Adversarial networks are widely used in many applications such as image generation [11, 19], domain adaptation [20, 21, 22], fair representation [12, 23], etc. One widely used instance of adversarial networks is the generative adversarial network (GAN [9], Figure 1.1). Adversarial networks are difficult to train because adversarial nets try to accomplish two objectives simultaneously; weights are adjusted to maximize performance on one task while minimizing performance on another. Mathematically, this corresponds to finding a *saddle point* of a loss

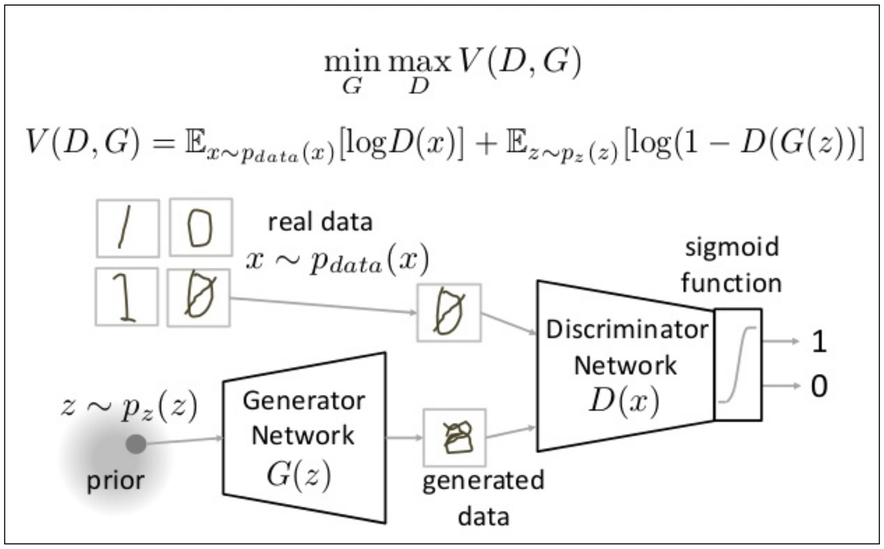


Figure 1.1: An example of Adversarial Networks (Generative Adversarial Network).

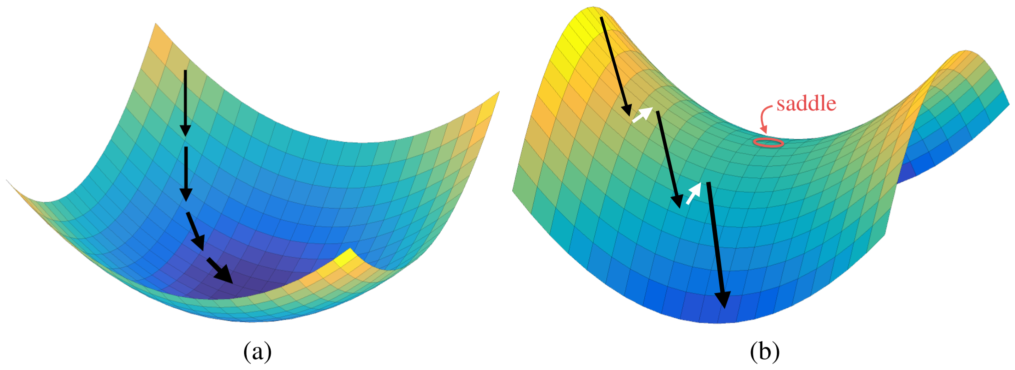


Figure 1.2: A schematic depiction of gradient methods. (a) Classical networks are trained by marching down the loss function until a minimizer is reached. Because classical loss functions are bounded from below, the solution path gets stopped when a minimizer is reached, and the gradient method remains stable. (b) Adversarial net loss functions may be unbounded from below, and training alternates between minimization and maximization steps. If minimization (or, conversely, maximization) is more powerful, the solution path "slides off" the loss surface and the algorithm becomes unstable, resulting in a sudden "collapse" of the network.

function and can be written as the following optimization problem

$$\min_u \max_v \mathcal{L}(u,v) \tag{1.1}$$

for some loss function $\mathcal{L}$ and variables u and v. A schematic depiction of the difference between finding a minimizer and finding a saddle point is shown in the Figure 1.2.

In this work, we present a simple "prediction" step (1.2) that is easily added to many training algorithms for adversarial nets. We present both theoretical analysis as well as experiments to show the efficacy of the proposed method.

Prediction Method

$$\begin{aligned}
u^{k+1} &= u^k - \alpha_k \mathcal{L}'_u(u^k, v^k) && | \quad \text{gradient descent in } u\text{, starting at } (u^k, v^k) \\
\bar{u}^{k+1} &= u^{k+1} + (u^{k+1} - u^k) && | \quad \textit{predict} \text{ future value of } u \\
v^{k+1} &= v^k + \beta_k \mathcal{L}'_v(\bar{u}^{k+1}, v^k) && | \quad \text{gradient ascent in } v\text{, starting at } (\bar{u}^{k+1}, v^k)\ .
\end{aligned} \tag{1.2}$$

1.4 Similarity between the Laplace and Neural Tangent Kernels

In Chapter 5, we first show that Neural Tangent Kernel (NTK) has the same set of functions as the Laplace kernel for an overparameterized fully connected neural network. The underlying assumptions are that the activation function used in the neural network should be ReLU, and data should be distributed on the hypersphere. These assumptions are realistic because most practical neural networks use ReLU activation and normalize the dataset. Experiments show the similar performance of NTK and Laplace, and indicate a slight advantage to the more general γ-exponential kernel achieving state-of-the-art results for well-known 102 UCI data sets. This work has two direct implications: 1) one can replace Neural Tangent kernel with Laplace kernel and thus can avoid expensive NTK computation, 2) one can also gain much insight about neural networks from analysis of the well-known Laplace kernel, which has a simple closed form.

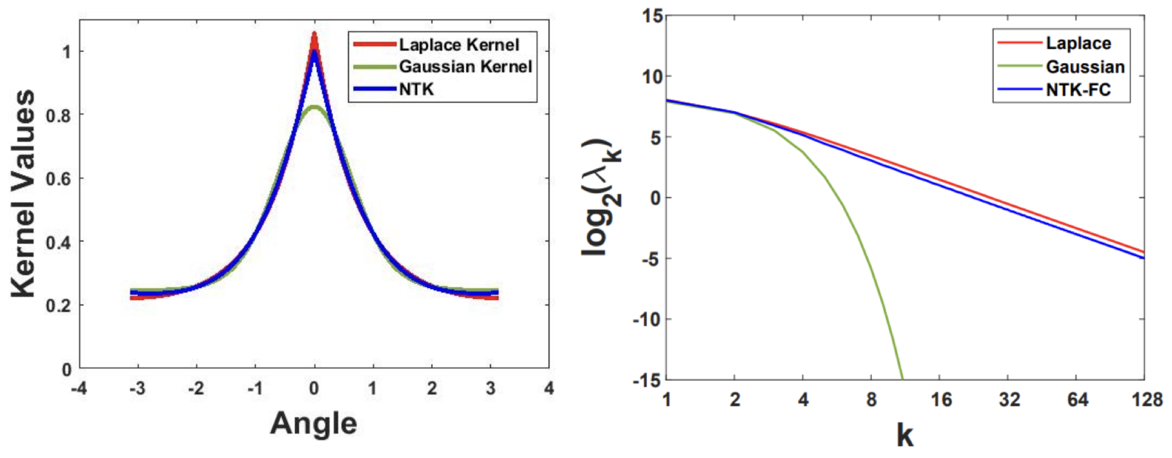


Figure 1.3: Left: An overlay of the NTK for a 6-layer FC network with ReLU activation with the Laplace and Gaussian kernels, as a function of the angle between their arguments. The exponential kernels are modulated by an affine transformation to achieve a least squares fit to the NTK. Note the high degree of similarity between the Laplace kernel and NTK. Right: eigenvalues as a function of frequency in $\mathbb{S}^1$. The slopes in these log-log plots indicate the rate of decay, which is similar for both the Laplace kernel and for NTK for the FC network with 6 layers. (Empirical slopes are -1.94 for both Laplace and NTK-FC.) The eigenvalues of the Gaussian kernel, in contrast, decay exponentially.

1.5 Summary

Deep learning has become an almost essential algorithm to solve many computer vision problems. Training is still cumbersome and needs a lot of manual effort to tune many knobs. In this work, we try to make it easier by focusing on automated and efficient training methods. Efficiency matters because of the considerable energy consumed by deep networks, and automation matters because of much manual effort involved. Automatic parameter tuning further reduces the computation required to reach similar results.

Keeping this broad picture in mind, we first focus on efficient and automated first-order methods for training deep networks for the classification task and then also explore the most popular second-order method, L-BFGS in a similar direction. Then we focus on different types of deep networks: adversarial networks and fully connected networks with infinite width. For adversarial networks, we propose a novel training algorithm based on prediction methods which improves stability in the training of adversarial deep networks across a wide range of learning rates. For fully connected deep networks with infinite width, recent works have shown that those are equivalent to kernel regressors that use Neural Tangent Kernels (NTKs). However, the computation of NTK is expensive in terms of memory and time. Thus, we focus on efficient methods for the computation of NTK and show that NTK can be replaced by classic standard Laplace kernel for most practical use-cases. The computation of the Laplace kernel, which has a simple closed-form, is way cheaper than NTK kernels. Another added benefit of this finding is that one can gain more insight by leveraging existing literature on the well-known Laplace kernel. In the following chapters, we discuss these research works in detail.

Chapter 2: Automated Inference using Adaptive Batch Sizes

In this chapter, we focus on first-order stochastic gradient methods. Classical stochastic gradient methods for optimization rely on noisy gradient approximations that become progressively less accurate as iterates approach a solution. The large noise and small signal in the resulting gradients make it difficult to use them for adaptive stepsize selection and automatic stopping. In this chapter, we propose alternative "big batch" SGD schemes that adaptively grow the batch size over time to maintain a nearly constant signal-to-noise ratio in the gradient approximation. The resulting methods have similar convergence rates to classical SGD, and do not require convexity of the objective. The high fidelity gradients enable automated learning rate selection and do not require stepsize decay. Big batch methods are thus easily automated and can run with little or no oversight. This work [24] is in collaboration with Soham De, David Jacobs, and Tom Goldstein.

2.1 Introduction

We are interested in problems of the form

$$\min_{x \in \mathcal{X}} \ell(x) := \begin{cases} \mathbb{E}_{z \sim p}[f(x; z)], \\ \frac{1}{N} \sum_{i=1}^{N} f(x; z_i), \end{cases} \tag{2.1}$$

where $\{z_i\}$ is a collection of data drawn from a probability distribution p. We assume that ℓ and f are differentiable, but possibly non-convex, and domain $\mathcal{X}$ is convex. In typical applications, each term $f(x; z)$ measures how well a model with parameters x fits one particular data observation z. The expectation over z measures how well the model fits the entire corpus of data on average.

When N is large (or even infinite), it becomes intractable to exactly evaluate $\ell(x)$ or its gradient $\nabla \ell(x)$, which makes classical gradient methods impossible. In such situations, the method of choice for minimizing (2.1) is the stochastic gradient descent (SGD) algorithm [13]. On iteration t, SGD selects a batch $\mathcal{B} \subset \{z_i\}$ of data uniformly at random, and then computes

$$x_{t+1} = x_t - \alpha_t \nabla_x \ell_{\mathcal{B}}(x_t), \tag{2.2}$$
$$\text{where} \quad \ell_{\mathcal{B}}(x) = \frac{1}{|\mathcal{B}|} \sum_{z \in \mathcal{B}} f(x; z),$$

where α_t denotes the stepsize used on the t-th iteration. Note that $\mathbb{E}_{\mathcal{B}}[\nabla_x \ell_{\mathcal{B}}(x_t)] = \nabla_x \ell(x_t)$, and so the calculated gradient $\nabla_x \ell_{\mathcal{B}}(x_t)$ can be interpreted as a "noisy" approximation to the true gradient. Because the gradient approximations are noisy, the stepsize α_t must vanish as $t \to \infty$ to guarantee convergence of the method. Typical stepsize rules require the user to find the optimal decay rate schedule, which usually requires an expensive grid search over different possible parameter values.

In this chapter, we consider a "big batch" strategy for SGD. Rather than letting the stepsize vanish over time as the iterates approach a minimizer, we let the minibatch $\mathcal{B}$ *adaptively grow* in size to maintain a constant signal-to-noise ratio of the gradient approximation. This prevents the algorithm from getting overwhelmed with noise, and guarantees convergence with an appropriate constant stepsize. Recent results [17] have shown that large *fixed* batch sizes fail to find good minimizers for non-convex problems like deep neural networks. Adaptively increasing the batch size over time overcomes this limitation: intuitively, in the initial iterations, the increased stochasticity (corresponding to smaller batches) can help land the iterates near a good minimizer, and larger batches later on can increase the speed of convergence towards this minimizer.

Using this batching strategy, we show that we can keep the stepsize constant, or let it adapt using a simple Armijo backtracking line search, making the method completely adaptive with no user-defined parameters.

Big batch methods that adaptively grow the batch size over time have several potential advantages over conventional small-batch SGD:

- Big batch methods don't require the user to choose stepsize decay parameters. Larger batch sizes with less noise enable easy estimation of the accuracy of the approximate gradient, making it straightforward to adaptively scale up the batch size and maintain fast convergence.
- Higher order methods like stochastic L-BFGS typically require more work per iteration than simple SGD. When using big batches, the overhead of more complex methods like L-BFGS can be amortized over more costly gradient approximations. Furthermore, better Hessian approximations can be computed using less noisy gradient terms.
- For a restricted class of non-convex problems (functions satisfying the Polyak-Łojasiewicz Inequality), the per-iteration complexity of big batch SGD is linear and the approximate gradients vanish as the method approaches a solution, which makes it easy to define automated stopping conditions. In contrast, small batch SGD exhibits sub-linear convergence, and the noisy gradients are not usable as a stopping criterion.
- Big batch methods are much more efficient than conventional SGD in massively parallel/distributed settings. Bigger batches perform more computation between parameter updates, and thus

allow a much higher ratio of computation to communication.

For the reasons above, big batch SGD is potentially much easier to automate and requires much less user oversight than classical small batch SGD.

2.2 Related work

In this section, we focus on automating stochastic optimization methods by reducing the noise in SGD. We do this by adaptively growing the batch size to control the variance in the gradient estimates, maintaining an approximately constant signal-to-noise ratio, leading to automated methods that do not require vanishing stepsize parameters. While there has been some work on adaptive stepsize methods for stochastic optimization [25, 26, 27, 28, 29], the methods are largely heuristic without any kind of theoretical guarantees or convergence rates. The work in Tan et al. [27] was a first step towards provable automated stochastic methods, and we explore in this direction to show provable convergence rates for the automated big batch method.

While there has been relatively little work in provable automated stochastic methods, there has been recent interest in methods that control gradient noise. These methods mitigate the effects of vanishing stepsizes, though choosing the (constant) stepsize still requires tuning and oversight. There have been a few papers in this direction that use dynamically increasing batch sizes. In Friedlander and Schmidt [30], the authors propose to increase the size of the batch by a constant factor on *every* iteration, and prove linear convergence in terms of the iterates of the algorithm. In Byrd et al. [31], the authors propose an adaptive strategy for growing the batch size; however, the authors do not present a theoretical guarantee for this method, and instead prove linear convergence for a continuously growing batch, similar to Friedlander and Schmidt [30].

Variance reduction (VR) SGD methods use an error correction term to reduce the noise in stochastic gradient estimates. The methods enjoy a provably faster convergence rate than SGD and have been shown to outperform SGD on convex problems [32, 33, 34, 35], as well as in parallel [36] and distributed settings [37]. A caveat, however, is that these methods require either extra storage or full gradient computations, both limiting factors when the dataset is very large. In a recent paper [38], the authors propose a growing batch strategy for a VR method that enjoys the same convergence guarantees. However, as mentioned above, choosing the constant stepsize still requires tuning. Another conceptually related approach is importance sampling, i.e., choosing training points such that the variance in the gradient estimates is reduced [39, 40, 41].

2.3 Big Batch SGD

2.3.1 Preliminaries and motivation

Classical stochastic gradient methods thrive when the current iterate is far from optimal. In this case, a small amount of data is necessary to find a descent direction, and optimization progresses efficiently. As x_t starts approaching the true solution $x^\star$, however, noisy gradient estimates frequently fail to produce descent directions and do not reliably decrease the objective. By choosing larger batches with less noise, we may be able to maintain descent directions on each iteration and uphold fast convergence. This observation motivates the proposed "big batch" method. We now explore this idea more rigorously.

To simplify notation, we hereon use $\nabla \ell$ to denote $\nabla_x \ell$. We wish to show that a noisy gradient approximation produces a descent direction when the noise is comparable in magnitude to the true gradient.

Lemma 1. *A sufficient condition for $-\nabla \ell_{\mathcal{B}}(x)$ to be a descent direction is*

$$\|\nabla \ell_{\mathcal{B}}(x) - \nabla \ell(x)\|^2 < \|\nabla \ell_{\mathcal{B}}(x)\|^2.$$

This is a standard result in stochastic optimization (see the Appendix A.1). In words, if the error $\|\nabla \ell_{\mathcal{B}}(x) - \nabla \ell(x)\|^2$ is small relative to the gradient $\|\nabla \ell_{\mathcal{B}}(x)\|^2$, the stochastic approximation is a descent direction. But how big is this error and how large does a batch need to be to guarantee this condition? By the weak law of large numbers[1]

$$\begin{aligned}\mathbb{E}[\|\nabla \ell_{\mathcal{B}}(x) - \nabla \ell(x)\|^2] &= \frac{1}{|\mathcal{B}|}\mathbb{E}[\|\nabla f(x;z) - \nabla \ell(x)\|^2] \\ &= \frac{1}{|\mathcal{B}|}\operatorname{Tr}\operatorname{Var}_z \nabla f(x;z),\end{aligned}$$

and so we can estimate the error of a stochastic gradient if we have some knowledge of the variance of $\nabla f(x;z)$. In practice, this variance could be estimated using the sample variance of a batch $\{\nabla f(x;z)\}_{z\in\mathcal{B}}$. However, we would like some bounds on the magnitude of this gradient to show that it is well-behaved, and also to analyze worst-case convergence behavior. To this end, we make the following assumption.

Assumption 1. *We assume f has L_z-Lipschitz dependence on data z, i.e., given two data points $z_1, z_2 \sim p(z)$, we have: $\|\nabla f(x;z_1) - \nabla f(x;z_2)\| \le L_z\|z_1 - z_2\|$.*

Under this assumption, we can bound the error of the stochastic gradient. The bound is uniform with respect to x, which makes it rather useful in analyzing the convergence rate for big batch methods.

[1] We assume the random variable $\nabla f(x;z)$ is measurable and has bounded second moment. These conditions will be guaranteed by the hypothesis of Theorem 1.

Theorem 1. *Given the current iterate x, suppose Assumption 1 holds and that the data distribution p has bounded second moment. Then the estimated gradient $\nabla \ell_{\mathcal{B}}(x)$ has variance bounded by*

$$\begin{aligned} \mathbb{E}_{\mathcal{B}} \|\nabla \ell_{\mathcal{B}}(x) - \nabla \ell(x)\|^2 &:= \operatorname{Tr} \operatorname{Var}_{\mathcal{B}}(\nabla \ell_{\mathcal{B}}(x)) \\ &\leq \frac{4L_z^2 \operatorname{Tr} \operatorname{Var}_z(z)}{|\mathcal{B}|}, \end{aligned}$$

where $z \sim p(z)$. Note the bound is uniform in x.

The proof is in the Appendix A.2. Note that, using a finite number of samples, one can easily approximate the quantity $\operatorname{Var}_z(z)$ that appears in our bound.

2.3.2 A template for big batch SGD

Theorem 1 and Lemma 1 together suggest that we should expect $d = -\nabla \ell_{\mathcal{B}}$ to be a descent direction reasonably often provided

$$\begin{aligned} & \theta^2 \|\nabla \ell_{\mathcal{B}}(x)\|^2 \geq \frac{1}{|\mathcal{B}|} [\operatorname{Tr} \operatorname{Var}_z(\nabla f(x; z_i))], \\ \text{or} \quad & \theta^2 \|\nabla \ell_{\mathcal{B}}(x)\|^2 \geq \frac{4L_z^2 \operatorname{Tr} \operatorname{Var}_z(z)}{|\mathcal{B}|}, \end{aligned} \tag{2.3}$$

for some $\theta < 1$. Big batch methods capitalize on this observation.

On each iteration t, starting from a point x_t, the big batch method performs the following steps:

1. Estimate the variance $\operatorname{Tr}\operatorname{Var}_z[\nabla f(x_t; z)]$, and a batch size K large enough that

$$\begin{aligned} \theta^2\mathbb{E}\|\nabla \ell_{\mathcal{B}_t}(x_t)\|^2 &\geq \mathbb{E}\|\nabla \ell_{\mathcal{B}_t}(x_t) - \nabla \ell(x_t)\|^2 \\ &= \frac{1}{K}\operatorname{Tr}\operatorname{Var}_z f(x_t; z), \end{aligned} \tag{2.4}$$

   where $\theta \in (0, 1)$ and $\mathcal{B}_t$ denotes the selected batch on the t-th iteration with $|\mathcal{B}| = K$.

2. Choose a stepsize α_t.

3. Perform the update $x_{t+1} = x_t - \alpha_t \nabla \ell_{\mathcal{B}_t}(x_t)$.

Clearly, we have a lot of latitude in how to implement these steps using different variance estimators and different stepsize strategies. In the following section, we show that, if condition (2.4) holds, then linear convergence can be achieved using an appropriate constant stepsize. In subsequent sections, we address the issue of how to build practical big batch implementations using automated variance and stepsize estimators that require no user oversight.

2.4 Convergence Analysis

We now present convergence bounds for big batch SGD methods (2.5). We study stochastic gradient updates of the form

$$x_{t+1} = x_t - \alpha \nabla \ell_{\mathcal{B}_t}(x_t) = x_t - \alpha(\nabla \ell(x_t) + e_t), \tag{2.5}$$

where $e_t = \nabla \ell_{\mathcal{B}_t}(x_t) - \nabla \ell(x_t)$, and $\mathbb{E}_{\mathcal{B}}[e_t] = 0$. Let us also define $g_t = \nabla \ell(x_t) + e_t$.

Before we present our results, we first state two assumptions about the loss function $\ell(x)$.

Assumption 2. *We assume that the objective function ℓ has L-Lipschitz gradients:*

$$\ell(x) \leq \ell(y) + \nabla \ell(y)^T (x - y) + \frac{L}{2} \|x - y\|^2.$$

This is a standard smoothness assumption used widely in the optimization literature. Note that a consequence of Assumption 2 is the property:

$$\|\nabla \ell(x) - \nabla \ell(y)\| \leq L \|x - y\|.$$

Assumption 3. *We also assume that the objective function ℓ satisfies the Polyak-Łojasiewicz Inequality:*

$$\|\nabla \ell(x)\|^2 \geq 2\mu(\ell(x) - \ell(x^\star)).$$

Note that this inequality does *not* require ℓ to be convex, and is, in fact, a weaker assumption than what is usually used. It does, however, imply that every stationary point is a global minimizer [42, 43].

We now present a result that establishes an upper bound on the objective value in terms of the error in the gradient of the sampled batch. We present all the proofs in the Appendix A.

Lemma 2. *Suppose we apply an update of the form (2.5) where the batch $\mathcal{B}_t$ is uniformly sampled from the distribution p on each iteration t. If the objective ℓ satisfies Assumption 2, then we have*

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \mathbb{E}\big[\ell(x_t) - \ell(x^\star) \\ - \big(\alpha - \frac{L\alpha^2}{2}\big)\|\nabla \ell(x_t)\|^2 + \frac{L\alpha^2}{2}\|e_t\|^2\big].$$

Further, if the objective ℓ satisfies the PL Inequality (Assumption 3), we have:

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \\ \leq \Big(1 - 2\mu\big(\alpha - \frac{L\alpha^2}{2}\big)\Big)\mathbb{E}[\ell(x_t) - \ell(x^\star)] + \frac{L\alpha^2}{2}\mathbb{E}\|e_t\|^2.$$

Using Lemma 2, we now provide convergence rates for big batch SGD.

Theorem 2. *Suppose ℓ satisfies Assumptions 2 and 3. Suppose further that on each iteration the batch size is large enough to satisfy (2.4) for $\theta \in (0, 1)$. If $0 \leq \alpha < \frac{2}{L\beta}$, where $\beta = \frac{\theta^2 + (1-\theta)^2}{(1-\theta)^2}$, then we get the following linear convergence bound for big batch SGD using updates of the form 2.5:*

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \gamma \cdot \mathbb{E}[\ell(x_t) - \ell(x^\star)],$$

where $\gamma = \big(1 - 2\mu(\alpha - \frac{L\alpha^2\beta}{2})\big)$. Choosing the optimal stepsize of $\alpha = \frac{1}{\beta L}$, we get

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \big(1 - \frac{\mu}{\beta L}\big) \cdot \mathbb{E}[\ell(x_t) - \ell(x^\star)].$$

Note that the above linear convergence rate bound holds without requiring convexity. Comparing it with the convergence rate of deterministic gradient descent under similar assumptions, we see that big batch SGD suffers a slowdown by a factor β, due to the noise in the estimation of the gradients.

2.4.1 Comparison to classical SGD

Conventional small batch SGD methods can attain only $O(1/t)$ convergence for strongly convex problems, thus requiring $O(1/\epsilon)$ gradient evaluations to achieve an optimality gap less than ϵ, and this has been shown to be *optimal* in the online setting (i.e., the infinite data setting) [44]. In the previous section, however, we have shown that big batch SGD methods converge linearly in the number of iterations, under a weaker assumption than strong convexity, in the online setting. Unfortunately, per-iteration convergence rates are not a fair comparison between these methods because the cost of a big batch iteration grows with the iteration count, unlike classical SGD. For this reason, it is interesting to study the convergence rate of big batch SGD as a function of *gradient evaluations*.

From Lemma 2, we see that we should not expect to achieve an optimality gap less than ϵ until we have: $\frac{L\alpha^2}{2}\mathbb{E}_{\mathcal{B}_t}\|e_t\|^2 < \epsilon$. In the worst case, by Theorem 1, this requires $\frac{L\alpha^2}{2}\frac{4L_z^2 \operatorname{Tr}\operatorname{Var}_z(z)}{|\mathcal{B}|} < \epsilon$, or $|\mathcal{B}| \geq O(1/\epsilon)$ gradient evaluations. Note that in the online or infinite data case, this is an optimal bound, and matches that of other SGD methods.

We choose to study the infinite sample case since the finite sample case is fairly trivial with a growing batch size: asymptotically, the batch size becomes the whole dataset, at which point we get the same asymptotic behavior as deterministic gradient descent, achieving linear convergence rates.

2.5 Practical Implementation with Backtracking Line Search

While one could implement a big batch method using analytical bounds on the gradient and its variance (such as that provided by Theorem 1), the purpose of big batch methods is to enable automated adaptive estimation of algorithm parameters. Furthermore, the stepsize bounds provided by our convergence analysis, like the stepsize bounds for classical SGD, are fairly conservative and more aggressive stepsize choices are likely to be more effective.

The framework outlined in Section 2.3.2 requires two ingredients: estimating the batch size and estimating the stepsize. Estimating the batch size needed to achieve (2.4) is fairly straightforward. We start with an initial batch size K, and draw a random batch $\mathcal{B}$ with $|\mathcal{B}| = K$. We then compute the stochastic gradient estimate $\nabla \ell_{\mathcal{B}}(x_t)$ and the sample variance

$$\begin{aligned} V_{\mathcal{B}} &:= \frac{1}{|\mathcal{B}| - 1} \sum_{z \in \mathcal{B}} \|\nabla f(x_t; z) - \nabla \ell_{\mathcal{B}}(x_t)\|^2 \\ &\approx \operatorname{Tr} \operatorname{Var}_{z \in \mathcal{B}}(\nabla f(x_t; z)). \end{aligned} \tag{2.6}$$

We then test whether $\|\nabla \ell_{\mathcal{B}}(x_t)\|^2 > V_{\mathcal{B}}/|\mathcal{B}|$ as a proxy for (2.4). If this condition holds, we proceed with a gradient step, else we increase the batch size $K \leftarrow K + \delta_K$, and check our condition again. We fix $\delta_K = 0.1K$ for all our experiments. Our aggressive implementation also simply chooses $\theta = 1$. The fixed stepsize big batch method is listed in Algorithm 1.

We also consider a backtracking variant of SGD that adaptively tunes the stepsize. This method selects batch sizes using the same criterion (2.6) as in the constant stepsize case. However, after a batch has been selected, a backtracking Armijo line search is used to select a stepsize. In the Armijo line search, we keep decreasing the stepsize by a constant factor (in our case, by a factor

Algorithm 1 Big batch SGD: fixed stepsize

```
1: initialize starting pt. $x_0$, stepsize $\alpha$, initial batch size $K > 1$, batch size increment $\delta_k$
2: while not converged do
3:     Draw random batch with size $|\mathcal{B}| = K$
4:     Calculate $V_\mathcal{B}$ and $\nabla \ell_\mathcal{B}(x_t)$ using (2.6)
5:     while $\|\nabla \ell_\mathcal{B}(x_t)\|^2 \leq V_\mathcal{B}/K$ do
6:         Increase batch size $K \leftarrow K + \delta_K$
7:         Sample more gradients
8:         Update $V_\mathcal{B}$ and $\nabla \ell_\mathcal{B}(x_t)$
9:     end while
10:    $x_{t+1} = x_t - \alpha \nabla \ell_\mathcal{B}(x_t)$
11: end while
```

of 2) until the following condition is satisfied on each iteration:

$$\ell_\mathcal{B}(x_{t+1}) \leq \ell_\mathcal{B}(x_t) - c\alpha_t \|\nabla \ell_\mathcal{B}(x_t)\|^2, \tag{2.7}$$

where c is a parameter of the line search usually set to $0 < c \leq 0.5$. We now present a convergence result of big batch SGD using the Armijo line search.

Theorem 3. *Suppose that ℓ satisfies Assumptions 2 and 3 and on each iteration, and the batch size is large enough to satisfy (2.4) for $\theta \in (0, 1)$. If an Armijo line search, given by (2.7), is used, and the stepsize is decreased by a factor of 2 failing (2.7), then we get the following linear convergence bound for big batch SGD using updates of the form 2.5:*

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \gamma \cdot \mathbb{E}[\ell(x_t) - \ell(x^\star)],$$

where $\gamma = \left(1 - 2c\mu \min\left(\alpha_0, \frac{1}{2\beta L}\right)\right)$ and $0 < c \leq 0.5$. If the initial stepsize α_0 is set large enough *such that* $\alpha_0 \geq \frac{1}{2\beta L}$*, then we get:*

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \Big(1 - \frac{c\mu}{\beta L}\Big)\mathbb{E}[\ell(x_t) - \ell(x^\star)].$$

In practice, on iterations where the batch size increases, we double the stepsize before running line search to prevent the stepsizes from decreasing monotonically (algorithm is listed in the Appendix A.6).

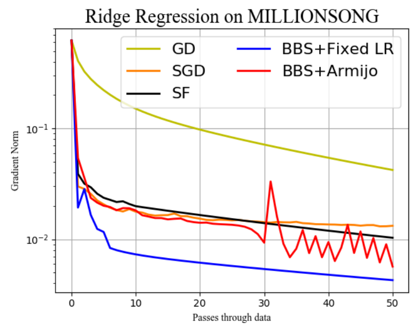
2.6 Experiments

In this section, we present our experimental results. We explore big batch methods with both convex and non-convex (neural network) experiments on large and high-dimensional datasets.

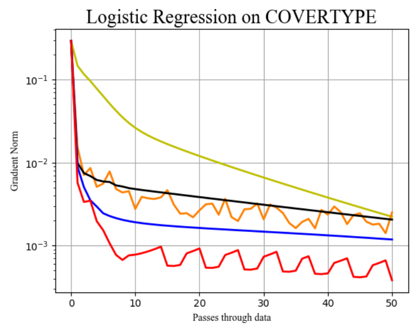
2.6.1 Convex Experiments

For the convex experiments, we test big batch SGD on a binary classification problem with logistic regression: $\min_x \frac{1}{n}\sum_{i=1}^n \log(1+\exp(-b_i a_i^T x))$, and a linear regression problem: $\min_x \frac{1}{n}\sum_{i=1}^n (a_i^T x - b_i)^2$.

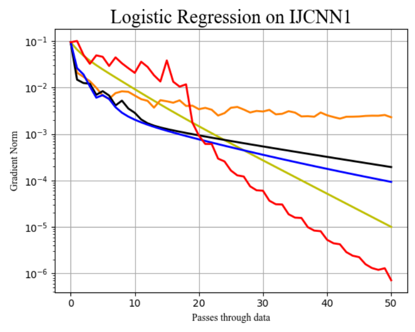
Figure 2.1 presents the results of our convex experiments on three standard real world datasets: IJCNN1 [45] and COVERTYPE [46] for logistic regression, and MILLIONSONG [47] for linear regression. As a preprocessing step, we normalize the features for each dataset. We compare deterministic gradient descent (GD) and SGD with stepsize decay ($\alpha_t = a/(b+t)$) to big batch SGD using a fixed stepsize (BBS+Fixed LR) and with backtracking line search (BBS+Armijo), as well as the growing batch method described in Friedlander and Schmidt [30] (denoted as SF; while the authors propose a quasi-Newton method, we adapt their algorithm to a first-order



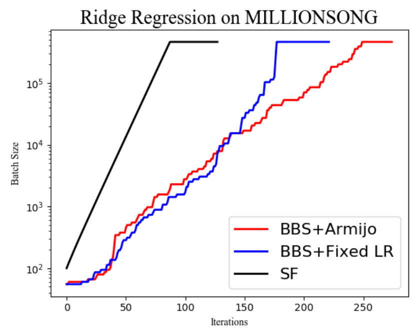
(a)



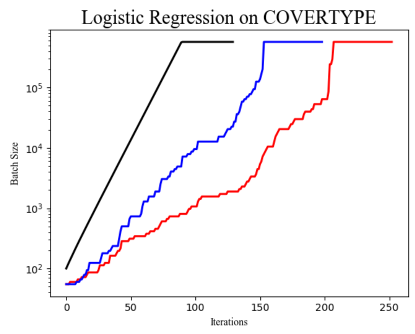
(b)



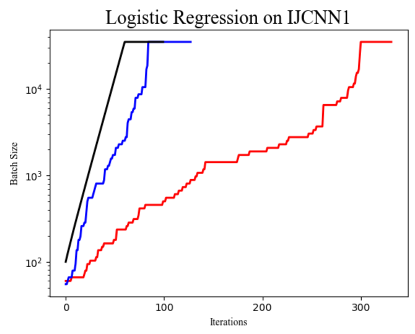
(c)



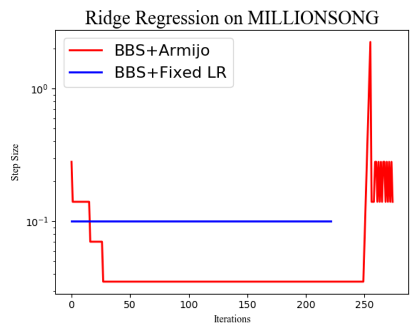
(d)



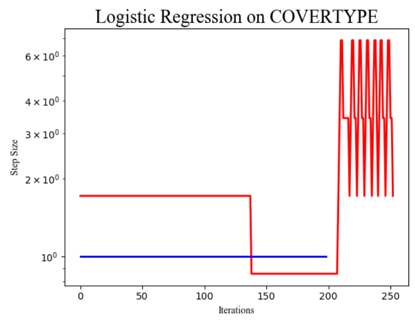
(e)



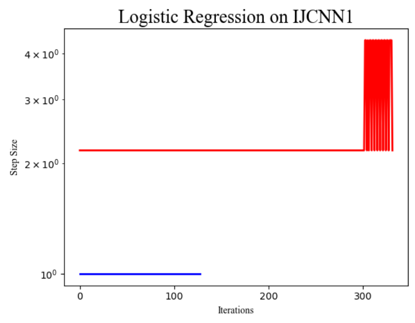
(f)



(g)



(h)



(i)

Figure 2.1: Convex experiments. Left to right: Ridge regression on MILLIONSONG; Logistic regression on COVERTYPE; Logistic regression on IJCNN1. The top row shows how the norm of the true gradient decreases with the number of epochs, the middle and bottom rows show the batch sizes and stepsizes used on each iteration by the big batch methods. Here 'passes through the data' indicates number of epochs, while 'iterations' refers to the number of parameter updates used by the method (there may be multiple iterations during one epoch).

method). We selected stepsize parameters using a comprehensive grid search for all algorithms, except BBS+Armijo which require no parameter tuning.

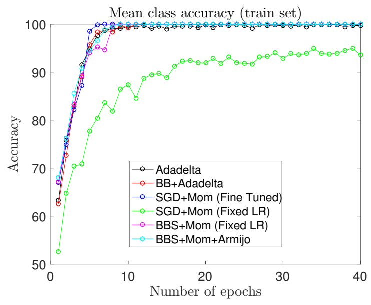
We see that across all three problems, the big batch methods outperform the other algorithms. We also see that the automated method (BBS+Armijo) is always comparable to or better than fixed stepsize methods. The automated method increase the batch size more slowly than BBS+Fixed LR and SF, and thus, this method can take more steps with smaller batches, leveraging its advantages longer. Further, note that the stepsizes derived by the automated method is very close to the optimal fixed stepsize rate.

2.6.2 Neural Network Experiments

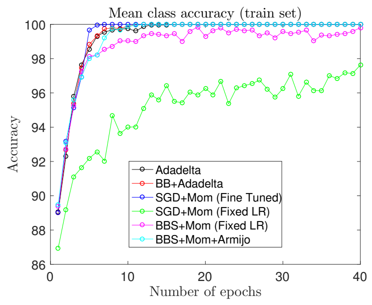
To demonstrate the versatility of the big batch SGD framework, we also present results on neural network experiments. We compare big batch SGD against SGD with finely tuned stepsize schedules and fixed stepsizes. We also compare with Adadelta [29], and combine the big batch method with AdaDelta (BB+AdaDelta) to show that more complex SGD variants can benefit from growing batch sizes. In addition, we had also compared big batch methods with L-BFGS. However, we found L-BFGS to consistently yield poorer generalization error on neural networks, and thus we omitted these results.

We train a convolutional neural network [48] (ConvNet) to classify three benchmark image datasets: CIFAR-10 [49], SVHN [50], and MNIST [48]. Our ConvNet is composed of 4 layers, excluding the input layer, with over 4.3 million weights. To compare against fine-tuned SGD, we used a comprehensive grid search on the stepsize schedule to identify the optimal schedule. Fixed stepsize methods use the default decay rule of the *Torch* library: $\alpha_t = \alpha_0/(1+10^{-7}t)$, where α_0 was chosen to be the stepsize used in the fine tuned experiments. We also tune the hyper-parameter ρ in the Adadelta algorithm. Details of the ConvNet and exact hyper-parameters used for training are presented in the supplemental.

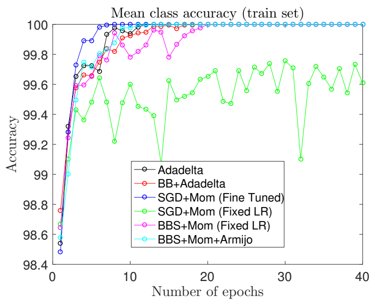
We plot the accuracy on the train and test set vs the number of epochs (full passes through the



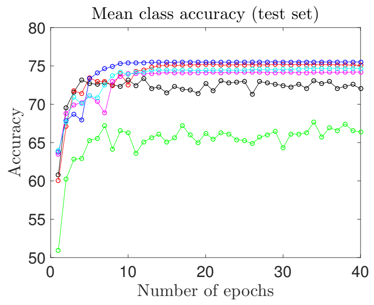
(a)



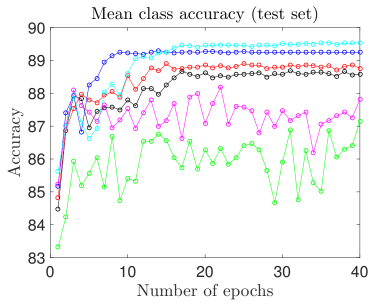
(b)



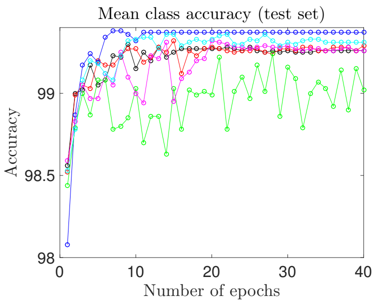
(c)



(d)



(e)



(f)

Figure 2.2: Neural Network Experiments. The three columns from left to right correspond to results for CIFAR-10, SVHN, and MNIST, respectively. The top row presents classification accuracies on the training set, while the bottom row presents classification accuracies on the test set.

dataset) in Figure 2.2. We notice that the big batch SGD with backtracking performs better than both Adadelta and SGD (Fixed LR) in terms of both train and test error. Big batch SGD even performs comparably to fine tuned SGD but without the trouble of fine tuning. This is interesting because most state-of-the-art deep networks (like AlexNet [2], VGG Net [51], ResNets [5]) were trained by their creators using standard SGD with momentum, and training parameters were tuned over long periods of time (sometimes months). Finally, we note that the big batch AdaDelta performs consistently better than plain AdaDelta on both large scale problems (SVHN and CIFAR-10), and performance is nearly identical on the small-scale MNIST problem.

2.7 Summary

In this chapter, we analyzed and studied the behavior of alternative SGD methods in which the batch size increases over time. Unlike classical SGD methods, in which stochastic gradients quickly become swamped with noise, these “big batch” methods maintain a nearly constant signal to noise ratio of the approximate gradient. As a result, big batch methods are able to adaptively adjust batch sizes without user oversight. The proposed automated methods are shown to be empirically comparable or better performing than other standard methods, but without requiring an expert user to choose learning rates and decay parameters.

Chapter 3:   Making L-BFGS Work with Industrial-Strength Nets

In chapter 2, we discussed first-order SGD methods, and now we will explore second-order methods. L-BFGS has been one of the most popular second-order methods for convex optimization. However, good performance by L-BFGS in deep learning has been elusive. Recent work has modified L-BFGS for deep networks for classification tasks and shown performance competitive with SGD and Adam (the most popular current algorithms) when batch normalization is not used. However, this work cannot be applied with batch normalization. Since batch normalization is a de facto standard and essential to good performance in deep networks, this still limits the use of L-BFGS. In this chapter, we address this issue. Our proposed method can be used as a drop-in replacement without changing the existing code. The proposed method performs consistently better than Adam and existing L-BFGS approaches and is comparable to carefully tuned SGD. We show results on three datasets, CIFAR-10, CIFAR-100, and STL-10, using three different popular deep networks ResNet, DenseNet, and Wide ResNet. This work marks another significant step towards making L-BFGS competitive in the deep learning community. This work [52] is in collaboration with Tom Goldstein and David W. Jacobs.

3.1 Introduction

Second-order methods like L-BFGS [53] (perhaps the most commonly used second-order method in machine learning) have a proven track record of performance for simple classifiers. They automatically select learning rates and provide fast convergence along with several other advantages over SGD [54]. Recently, several attempts have been made to explore L-BFGS for deep networks as well [18, 55, 56]. For example, Bollapragada et al. [18] proposed a new stochastic version of L-BFGS that can get comparable or even better results than SGD or Adam for classification tasks. One would expect that the advantages of L-BFGS should have popularized the use (or at least promoted further exploration) of the method for training deep networks. However, the applicability of L-BFGS has been limited because it does not play well with the state-of-the-art networks [57, 58, 59] that rely on Batch Normalization [60, 61, 62].

Batch Normalization (BatchNorm) is an integral component of almost all modern deep networks (Ioffe and Szegedy [60] has more than $15,000$ citations according to Google Scholar). To put things in perspective, without BatchNorm the classification accuracy of deep networks drops to almost 70% [18] from 92% [14] on CIFAR-10 using ResNet [57], and one can notice a similar drop in accuracy for many other popular deep networks and datasets. This discourages researchers from trying L-BFGS on deep networks without BatchNorm.

In this work we show how to get L-BFGS to work well with BatchNorm, obtaining performance comparable to SGD. Our method is general enough to apply to a wide range of L-BFGS variants [63, 64, 65], assuming only that curvature pair updates are done using finite gradient differencing. These updates consist of estimating a component of curvature by taking the difference between the gradient at different locations. Hopefully, our results will encourage more use of L-BFGS in deep learning, and more research on how to further improve its performance.

Contributions: We describe a novel scheme for stable curvature pair updates in the stochastic L-BFGS method for deep networks, in a way that is robust to the presence of BatchNorm. The scheme is generic enough to be applied to any variant of L-BFGS that employs a finite gradient differencing approach. This makes our method even more appealing, as it opens the possibility of borrowing other existing L-BFGS variants and their tricks. An additional advantage is that the method needs almost no parameter tuning, which is one of the benefits of using L-BFGS, and very important while training deep networks. Our numerical experiments show that the method proposed in this chapter – which we call the Frozen-Batch L-BFGS (FbLBFGS) method – outperforms existing L-BFGS approaches by a large margin (more than 10% in generalization accuracy for some problems) and also achieves performance comparable to SGD on standard large scale networks such as ResNet [57], DenseNet [59], and Wide ResNet [58] as demonstrated on standard datasets including CIFAR-10 [66], STL-10 [67], and CIFAR-100 [68].

Frozen-Batch L-BFGS (FbLBFGS) Method: In this chapter, we study how to design a stable curvature pair update in a stochastic setting when BatchNorm is used. We first note that it is crucial to take two gradient steps with the same training batch to obtain a consistent curvature estimate to use in updating the inverse Hessian. If two different batches are used, the gradient noise may dominate the curvature computation, destabilizing the update. In the frozen batch method, each time we select a new batch, we freeze it and take two gradient steps before making the Hessian update. This is a simple trick, and has been used before in a closely related method for online L-BFGS [69], namely oLBFGS, which also computes the finite difference of gradients using the same batch. The key difference is that oLBFGS always uses a fresh batch to take the actual descent gradient step (the recycled batch is only used for the Hessian update), whereas the proposed method uses the recycled batch for both the Hessian update and gradient step. This subtle difference has a huge impact on performance when BatchNorm is used.

3.2 Background and Related Work

Second-order methods have been studied in both convex and non-convex optimization, for more details see [1, 18, 24, 55, 65, 69, 70, 71, 72, 73, 74, 75]. Schraudolph et al. [69] proposed an online L-BFGS (oLBFGS) method to ensure a stable quasi-Newton curvature pair update by computing gradients on the same batch at the beginning and end of the iteration. Since this results in an extra computation of the gradients, Berahas et al. [72] proposed to use overlapping batches that share a subset of their data samples. In this case, the Hessian is updated using only samples that are shared between two adjacent batches, while the graident descent step uses all samples in a batch. This idea was further explored for large scale machine learning problems by Berahas and Takáč [55] and Bollapragada et al. [18]. Wang et al. [1] further extended oLBFGS and proposed a damped version (SdLBFGS) of it to maintain stable convergence in stochastic setting. Other approaches approximate curvature using the Fisher information matrix [76, 77, 78]. Krishnan et al. [79] approximately computes the inverse Hessian by expanding the matrices as the Neumann power series.

3.2.1 Multi-Batch L-BFGS

Let us consider the problem

$$\min_{x\in\mathbb{R}^d} F(x) \triangleq \frac{1}{N}\sum_{i=1}^{N} F_i(x) = \frac{1}{N}\sum_{i=1}^{N} f(x; z^i), \tag{3.1}$$

where $F_i(x) = f(x; z^i)$, f is a function (parametrized by x), and (z^i) is a collection of data drawn from an unknown probability distribution $P(z)$. A stochastic quasi-Newton method is given by

$$x_{k+1} = x_k - \alpha_k H_k g_k^{S_k}, \tag{3.2}$$

where the batch gradient is

$$g_k^{S_k} = \nabla F_{S_k}(x_k) \triangleq \frac{1}{|S_k|} \sum_{i \in S_k} \nabla F_i(x_k), \tag{3.3}$$

the set $S_k \subset \{1, 2, \cdots\}$ indexes data points $\{z^i\}$ sampled from the distribution P, and H_k is a positive definite approximation to the inverse Hessian.

3.2.2 Stable Quasi-Newton Updates

In the L-BFGS methods, the inverse Hessian approximation is updated using the following recursive formula,

$$\begin{aligned} H_{k+1} &= V_k^T H_k V_k + \rho_k \theta_k \theta_k^T \\ \rho_k &= (y_k^T \theta_k)^{-1} \\ V_k &= I - \rho_k y_k \theta_k^T \end{aligned} \tag{3.4}$$

where $\theta_k = x_{k+1} - x_k$ and $y_k = \nabla F_{S_{k+1}}(x_{k+1}) - \nabla F_{S_k}(x_k)$ is the difference in the gradients at x_{k+1} and x_k. When the batch changes from one iteration to the next ($S_{k+1} \neq S_k$), y_k is computed using different samples and the updating process (which is very sensitive to noise) may be unstable. To fix this, one approach is to repeat the same batch twice [1, 69] to compute gradient at both the iterates (x_k and x_{k+1}), given by

$$y_k = g_{k+1}^{S_k} - g_k^{S_k}. \tag{3.5}$$

However, this comes at the additional cost of wasted gradient computation. To avoid this, Berahas et al. [72] and Bollapragada et al. [18] propose to use overlapping batches, and compute Hessian updates using only the overlapping samples using the formula

$$y_k = g_{k+1}^{O_k} - g_k^{O_k}, \tag{3.6}$$

where $O_k = S_k \cap S_{k+1}$. This needs no extra computation since the two gradients in this case are subsets of the gradients corresponding to the samples S_k and S_{k+1}.

3.2.3 Stochastic Line Search

Historically, L-BFGS is combined with a line search method that automatically selects a stepsize by checking that the objective decreases sufficiently on each iteration, and cutting the stepsize if not. Bollapragada et al. [18] propose to perform a backtracking line search that aims to satisfy the Armijo condition

$$F_{S_k}(x_k - \alpha_k H_k g_k^{S_k}) \leq F_{S_k}(x_k) - c_1 \alpha_k (g_k^{S_k})^T H_k g_k^{S_k}, \tag{3.7}$$

where $0 < c_1 < 1$. This condition checks whether the observed decrease in the loss function is at least c_1 times the decrease predicted by a local linear approximation. This condition guarantees convergence in the deterministic setting, but not in the stochastic setting. The initial value of α_k is given by,

$$\alpha_k = \left(1 + \frac{\mathrm{Var}_{i \in S_k^v}\{g_k^i\}}{|S_k| \left\| g_k^{S_k} \right\|^2}\right)^{-1}, \tag{3.8}$$

where $\mathrm{Var}_{i \in S_k^v}\{g_k^i\} = \frac{1}{|S_k^v|-1} \sum_{i \in S_k^v} \left\| g_k^i - g_k^{S_k} \right\|^2$, and $S_k^v \subseteq S_k$.

Other authors suggest using a decaying learning rate such as $1/\sqrt{k}$ [1, 69]. This decaying learning rate is more theoretically justified in that convergence is guaranteed if the Hessian approximation is constant, but in practice this may be slow.

3.2.4 Batch Normalization

Batch normalization[1] (BatchNorm) normalizes the activation output $f_l(x_k; z^i)$ of a given layer l as follows:

$$\begin{aligned}
\mu^{S_k} &\leftarrow \frac{1}{|S_k|} \sum_{i=1}^{|S_k|} f_l(x_k; z^i) \\
\mathrm{Var}^{S_k} &\leftarrow \frac{1}{|S_k|} \sum_{i=1}^{|S_k|} (f_l(x_k; z^i) - \mu^{S_k})^2 \\
f_l(x_k; z^i; \mu^{S_k}, \mathrm{Var}^{S_k}) &\leftarrow \frac{f_l(x_k; z^i) - \mu^{S_k}}{\sqrt{\mathrm{Var}^{S_k} + \epsilon}} \\
&\equiv \mathrm{BN}^{S_k}(f_l(x_k; z^i))
\end{aligned} \tag{3.9}$$

where $z^i \in S_k$, S_k is the batch at iteration k, ϵ is a small number used for numerical stability and f_l is the transformation function of the layer l. From Eq 3.9, it is evident that the BN transform does not independently process each training example. Rather, $\mathrm{BN}^{S_k}(f_l(x_k; z^i))$ is a function of both the training example and the other examples in that batch. For more details refer to [80].

[1]For clarity, we omit the learnable parameters γ and β, which produce an affine transformation applied on top of the batch-norm layer. This is just another trainable layer, not affecting the proposed analysis. See [80] for more details.

3.3 Proposed Method

3.3.1 Stable Quasi-Newton Updates With BatchNorm:

We now explain the challenges BatchNorm poses for existing L-BFGS approaches and then we discuss our proposed solution. As per the overlapping batch approach suggested by Bollapragada et al. [18], Berahas et al. [72], the stochastic gradient difference y_k with BatchNorm can be written as:

$$\begin{aligned} y_k = &\frac{1}{|O_k|}\sum_{i\in O_k}\nabla F_i(x_{k+1};\mathrm{BN}^{\mathcal{S}_{k+1}}) \\ &- \frac{1}{|O_k|}\sum_{i\in O_k}\nabla F_i(x_k;\mathrm{BN}^{\mathcal{S}_k}), \end{aligned} \tag{3.10}$$

where $\mathrm{BN}^{\mathcal{S}_k}$ represents the batch normalization statistics for the batch S_k. From (3.10), it is clear that because of the different BatchNorm parameters, the gradients used to compute y_k are not consistent; even though only overlapping samples are used, the batch norm statistics depend on the non-overlapping samples. This breaks the gradient consistency for the overlapping approach.

To address this issue, one obvious solution is to repeat the same batch twice [1, 69]. However, this requires that the gradient be evaluated twice for every batch S_k at x_k and x_{k+1}. We make use of the extra gradient computation by actually taking another gradient step. We update the Hessian only using gradients from the same batch. Specifically, we propose to freeze the batch for two consecutive iterations, take two gradient steps, and then update the curvature pair (y_k, θ_k). The newly updated Hessian is applied to a gradient from the same batch. Then in the next iteration, when changing the batch from S_k to S_{k+1}, we do not update the curvature pair. Put another way, we compute two gradients for each batch, and we take a gradient descent step using both of these gradients. We find that this approach works significantly better than the existing approach SdLBFGS [1], improving L-BFGS performance by a large margin. We believe this is due to the fact that the gradient direction in our method gets pre-conditioned by a Hessian that was updated on the same batch. While with Wang et al. [1], one takes a (consistent and probably stable) Hessian update on one batch, and then use it to pre-condition on another.

Algorithm 2 Frozen-Batch L-BFGS (FbLBFGS)

```
Input: $x_0$ (initial iterate), $D = \{(z^i, t^i), \text{for } i = 1, \dots, n\}$ (training data), $m$ (memory parameter), $U$ = False (Flag to control curvature update).
 1: Create initial batch $S_1$
 2: for $k = 1, 2, ...$ do
 3:     if $k == 1$ then
 4:         Set the search direction $p_k = -g_k^{S_k}$
 5:     else
 6:         Calculate the search direction $p_k = -H_k g_k^{S_k}$ {Using L-BFGS Two-Loop
            Recursion (Procedure 3.1 in [1])}
 7:     end if
 8:     Normalize the search direction $p_k = \frac{p_k}{||p_k||_2}$
 9:     Set $\alpha_k = 1$
10:     while the Armijo condition (3.7) not satisfied do
11:         Set $\alpha_k = \alpha_k/2$
12:     end while
13:     Compute $x_{k+1} = x_k + \alpha_k p_k$
14:     if $U$ is True then
15:         Compute the curvature pairs $\theta_k = x_{k+1} - x_k$ and $y_k = g_{k+1}^{S_k} - g_k^{S_k}$
16:         Replace the oldest pair $(\theta_i, y_i)$ by $\theta_k, y_k$
17:         Create the next batch $S_{k+1}$
18:         Set $U$ = False {Do not update curvature in next iteration}
19:     else
20:         Set $U$ = True {Update curvature in next iteration}
21:         Set $S_{k+1} = S_k$ {Freeze the sample in next iteration}
22:     end if
23: end for
```

3.3.2 Line Search:

The stochastic line search (eq 3.7) proposed by Bollapragada et al. [18] requires computation of the initial value of α_k, which in turn requires computation of the variance of the gradient for

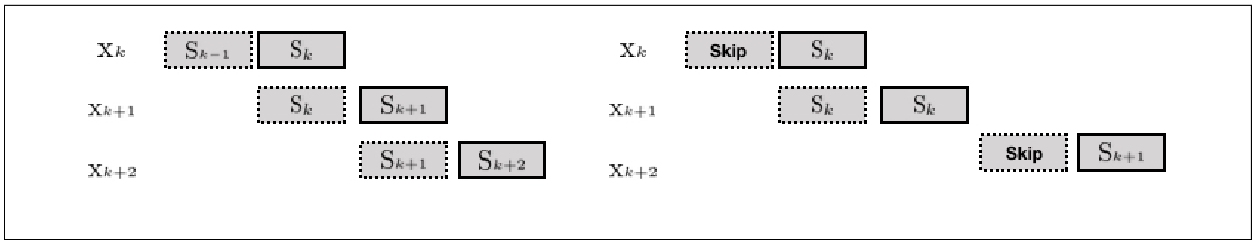


Figure 3.1: A side-by-side schematic depiction of curvature pair (y_k, θ_k) update scheme in SdLBFGS [1] and FbLBFGS (ours). Here, the dotted box represents the batch used for a curvature pair update and the solid box represents the batch used for the gradient step. The SdLBFGS method computes an auxiliary stochastic gradient at x_k using the sample S_{k-1} from the $(k-1)$-st iteration, which is used for gradient differencing only (to update y_k). Then a gradient step is taken on a new batch S_k and the process repeats. On the other hand, FbLBFGS (proposed) does not compute any auxiliary gradients but skips the first curvature pair update (k-th iteration), takes a gradient step, and in the next iteration ($(k+1)$-st) the same frozen batch S_k is used both to update the curvature pair and to take another gradient step. In the following next iteration (x_{k+2}), when the batch changes to S_{k+1}, the Hessian update is skipped, and the process repeats.

each example. This is not possible when using BatchNorm, since estimation of the variance of the gradient requires a forward pass for each example one by one, which will change the batch statistics.

In the stochastic case, sometimes the norm of the search direction can be too large, causing the algorithm to be very unstable. To cater to this, we propose a heuristic: we normalize the search direction (-$H_k g_k^{S_k}$) before doing the standard Armijo line search. This heuristic works surprisingly well in practice for all the datasets and all the models we tried.

3.3.3 On Convergence

In this section, we discuss the convergence of the proposed mehtod (FbLBFGS). The main challenge in designing a stochastic L-BFGS method for non-convex problem lies in the difficulty in preserving the positive-definiteness of the inverse Hessian approximation H_k, due to the non-convexity of the problem and the noise in gradient estimation. Wang et al. [1] proposed to address this issue by using damped curvature pair update. We leverage the convergence proof by Wang et al. [1] and show that we do not break their convergence conditions.

When frozen batch is used, we update the inverse Hessian approximation following [1]:

$$\begin{aligned} H_{k+1} &= V_k^T H_k V_k + \rho_k \theta_k \theta_k^T \\ \hat{y_k} &= \lambda_k y_k + (1-\lambda_k) H_k^{-1} \theta_k \\ \rho_k &= (\hat{y_k}^T \theta_k)^{-1} \\ V_k &= I - \rho_k \hat{y_k} \theta_k^T, \end{aligned} \tag{3.11}$$

where $\theta_k = x_{k+1} - x_k$ and $y_k = \nabla F_{S_k}(x_{k+1}) - \nabla F_{S_k}(x_k)$ is the difference in the gradients at x_{k+1} and x_k. The damping factor λ_k is give by,

$$\lambda_k = \begin{cases} \frac{0.75 \theta_k^\top H_k^{-1} \theta_k}{\theta_k^\top H_k^{-1} \theta_k - \theta_k^\top y_k}, & \text{if } \theta_k^\top y_k < 0.25 \theta_k^\top H_k^{-1} \theta_k, \\ 1, & \text{otherwise.} \end{cases} \tag{3.12}$$

When the batch is changed, we do not update the inverse Hessian approximation, i.e., we skip the update and set $H_{k+1} = H_k$. So, this ensures that the inverse Hessian approximation always remains positive-definite.

3.4 Experiments

In this section, we empirically demonstrate the proposed method's effectiveness on three benchmark datasets: STL-10 [67], CIFAR-10 [66], and CIFAR-100 [68]. Our results show three main points. First, without fine tuning, FbLBFGS can obtain generalization performance competitive with carefully tuned SGD using practical state-of-the-art architectures that use BatchNorm [80]. Second, FbLBFGS outperforms existing adaptive optimizers that automatically set the learning rate with minimal or no user supervision. Third, we show that our simple approach substantially outperforms existing L-BFGS methods.

It is impressive that FbLBFGS's performance approaches that of highly tuned SGD. For each model and dataset, untold grad student hours have been spent on grid search for good hyperparameters. Often these are chosen based on performance on the test set, instead of a separate held out dataset, overfitting the data [81]. To get a better sense of this, we also test SGD with a standard method of automatic parameter tuning. S1 and S2 are defined as follows: the learning rate starts with 0.1 for S1 or 1 for S2 and is reduced by $1/10$ after every $100, 200, 300$ epochs. FbLBFGS often outperforms these approaches.

We also compare our method against Adam [28]. Adam is the most popular and effective adaptive optimizer. We find that other adaptive optimizers [29, 82] perform similarly. FbLBFGS always significantly outperforms Adam.

Finally, we also compare the proposed method with existing L-BFGS methods. Specifically, we compare against the overlapping batch approach by Bollapragada et al. [18] and the stochastic damped L-BFGS (SdLBFGS) method by Wang et al. [1]. We use our line search with both of these so that we can compare our approach of freezing batches to prior approaches that address the same issue.

In all experiments, the batch size used is 128, history size is 5, and default parameters for Adam and other L-BFGS methods are used. We ran all the methods for 350 epochs. It is observed that as the optimizer reaches towards the solution, the gradients might be too noisy, and to get high fidelity gradients its recommended to increase the batch size [18, 24]. To address this issue, we set the batch size to 25% of the dataset towards the end of the training, for all the experiments. The deep networks used are WRN-28-10 [58], ResNet-18 [57], and DenseNet-40-12 [59]. The computational cost of FbLBFGS is increased because it makes two passes from the same frozen batch twice for each epoch. For a fair comparisons, we ran all other methods with a similar frozen batch as well (but did not notice any change in the performance as compared to using the batch only once per epoch). We report results over 5 different random seeds, as shown in Table 3.1, Table 3.2, and Table 3.3, for STL-10, CIFAR-10 and CIFAR-100, respectively. Figures 3.2, 3.4, and 3.6 show the training and test accuracy for STL-10, CIFAR-10, and CIFAR-100 on the deep networks Wide ResNet, DenseNet, and ResNet, respectively[2].

From the results, one can observe that our approach (FbLBFGS) always outperforms all the adaptive methods. Its performance compared with tuned SGD is comparable; sometimes it performs better than tuned SGD, sometimes about the same, and sometimes a bit worse. This shows the effectiveness of our approach to BatchNorm and also the practical usefulness of L-BFGS when applying deep networks to problems in which SGD has not been carefully tuned, since FbLBFGS has no free parameters.

Figures 3.3, 3.5, and 3.7 show the number of function evaluations per line search for each L-BFGS method. The overlap and SdLBFGS methods take nearly 10 times more backtracking steps than our method, which is likely due to a better Hessian approximation by our method. We also observed that the Armijo condition almost always accepts step-length 1, which means there is little overhead for using the line search. We believe normalizing the search direction before doing a line search is an effective heuristic. Without normalizing the search direction, all L-BFGS methods perform poorly. Hence, we ignored those results.

[2]All the training curves are in the Appendix B.1

Method	ResNet		DenseNet		Wide ResNet	
	Train	Test	Train	Test	Train	Test
SGD*	99.9	73.7 ±0.3	99.9	**73.5** ±0.2	99.9	**77.1**±0.5
SGD(S1)	99.8	68.0 ±0.2	99.9	63.9 ±0.2	99.9	71.1 ±0.2
SGD(S2)	99.9	72.5 ±0.3	99.9	71.9 ±0.3	99.9	71.8 ±0.6
Adam	99.9	69.5 ±0.1	99.9	70.2 ±0.2	99.9	69.2 ±0.2
Bollapragada et al. [18]	95.8	65.8 ±0.1	98.2	67.2 ±0.4	93.2	62.5 ±0.7
SdLBFGS [1]	96.7	66.8 ±0.3	96.7	69.8 ±0.2	94.8	65.1 ±0.8
FbLBFGS (ours)	99.8	**75.1** ±0.1	99.9	73.4 ±0.5	99.9	76.4 ±0.4

Table 3.1: Comparison of train/test accuracy for STL-10 on ResNet, DenseNet, and Wide ResNet respectively. The results are shown in the format of 'mean $\pm std$' computed over 5 random seeds. Higher are better. $\star$ denotes highly tuned SGD with grid search and potentially over-fitting the test set for a particular model. We report these results for complete perspective.

Method	ResNet		DenseNet		Wide ResNet	
	Train	Test	Train	Test	Train	Test
SGD*	99.9	92.1 ±0.2	99.6	91.2 ±0.3	99.9	**95.2** ±0.5
SGD(S1)	99.8	90.6 ±0.3	99.5	89.9 ±0.4	99.9	93.2 ±0.4
SGD(S2)	90.6	89.2 ±0.2	99.9	**93.1** ±0.1	99.9	94.8 ±0.2
Adam	97.1	91.4 ±0.5	95.2	89.2 ±0.1	95.1	90.1 ±0.3
Bollapragada et al. [18]	93.7	84.6±0.6	95.0	85.6 ±0.4	95.8	90.1 ±0.3
SdLBFGS [1]	94.5	86.2 ±0.2	95.5	87.4 ±0.3	96.1	88.1 ±0.2
FbLBFGS (ours)	99.9	**92.9** ±0.4	99.5	91.2 ±0.1	99.8	94.2 ±0.3

Table 3.2: Comparison of train/test accuracy for CIFAR-10 on ResNet, DenseNet, and Wide ResNet respectively. The results are shown in the format of 'mean $\pm std$' computed over 5 random seeds. Higher are better. $\star$ denotes highly tuned SGD with grid search and potentially over-fitting the test set for a particular model. We report these results for complete perspective.

Method	ResNet		DenseNet		Wide ResNet	
	Train	Test	Train	Test	Train	Test
SGD*	96.5	**66.6** ±0.2	97.1	**68.4**±0.1	99.9	**79.4** ±0.3
SGD(S1)	93.1	63.5 ±0.3	97.9	67.5 ±0.3	99.9	76.7 ±0.4
SGD(S2)	96.0	66.2 ±0.4	96.9	67.0 ±0.2	99.9	79.4 ±0.3
Adam	94.9	63.2 ±0.5	97.5	63.4 ±0.2	97.4	67.0 ±0.4
Bollapragada et al. [18]	89.6	58.8 ±0.3	72.1	62.2 ±0.5	91.2	66.4 ±0.3
SdLBFGS [1]	89.1	59.5 ±0.3	75.3	64.8 ±0.3	95.8	66.6 ±0.1
FbLBFGS (ours)	93.8	65.2 ±0.3	95.2	67.9 ±0.3	99.5	75.4±0.2

Table 3.3: Comparison of train/test accuracy for CIFAR-100 on ResNet, DenseNet, and Wide ResNet respectively. The results are shown in the format of 'mean $\pm std$' computed over 5 random seeds. Higher are better. $\star$ denotes highly tuned SGD with grid search and potentially over-fitting the test set for a particular model. We report these results for complete perspective.

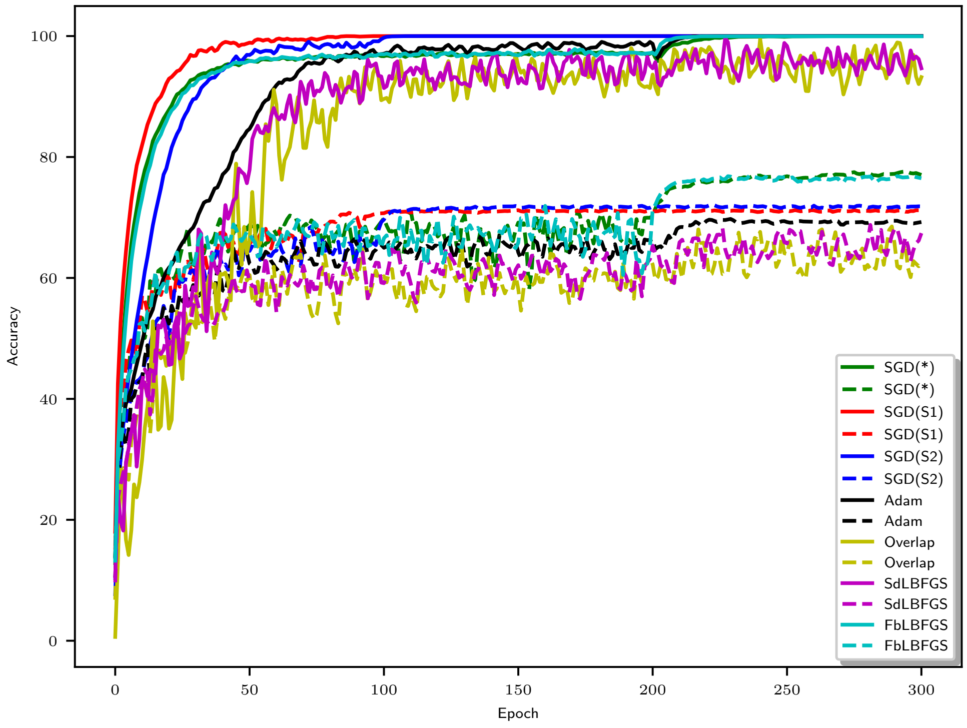


Figure 3.2: Overview of the performance of STL-10 on Wide ResNet. The solid lines represent train accuracy and dashed lines represent test accuracy, respectively.

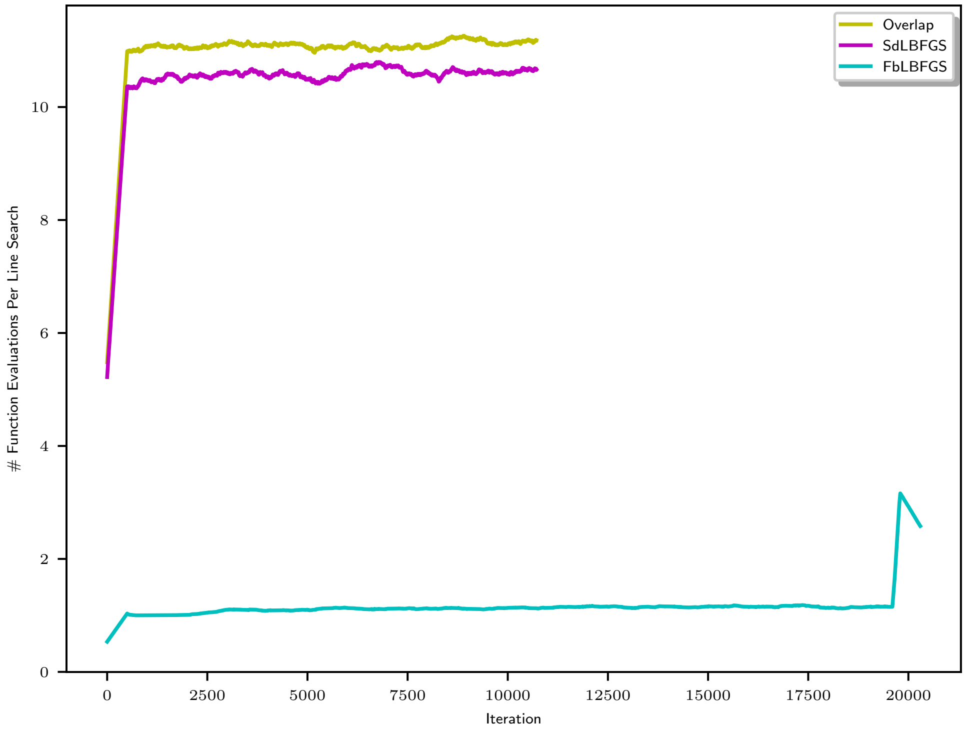


Figure 3.3: The number of function evaluations per line search for STL-10 on Wide ResNet. The average of every consecutive 100 iteration is plotted for display purpose.

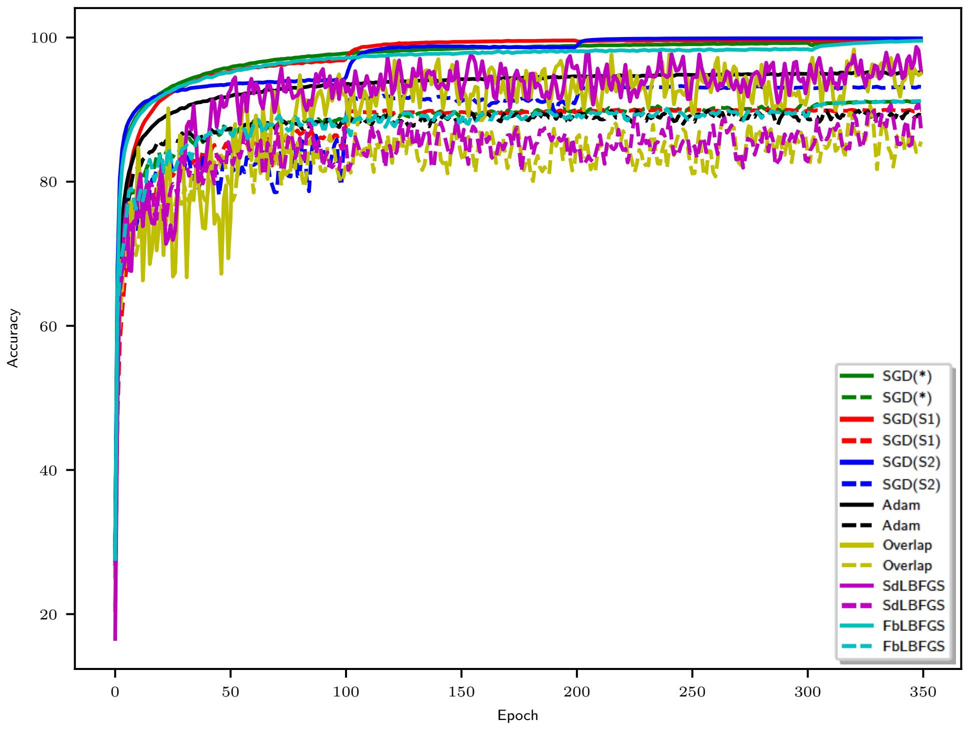


Figure 3.4: Overview of the performance of CIFAR-10 on DenseNet. The solid lines represent train accuracy and dashed lines represent test accuracy, respectively.

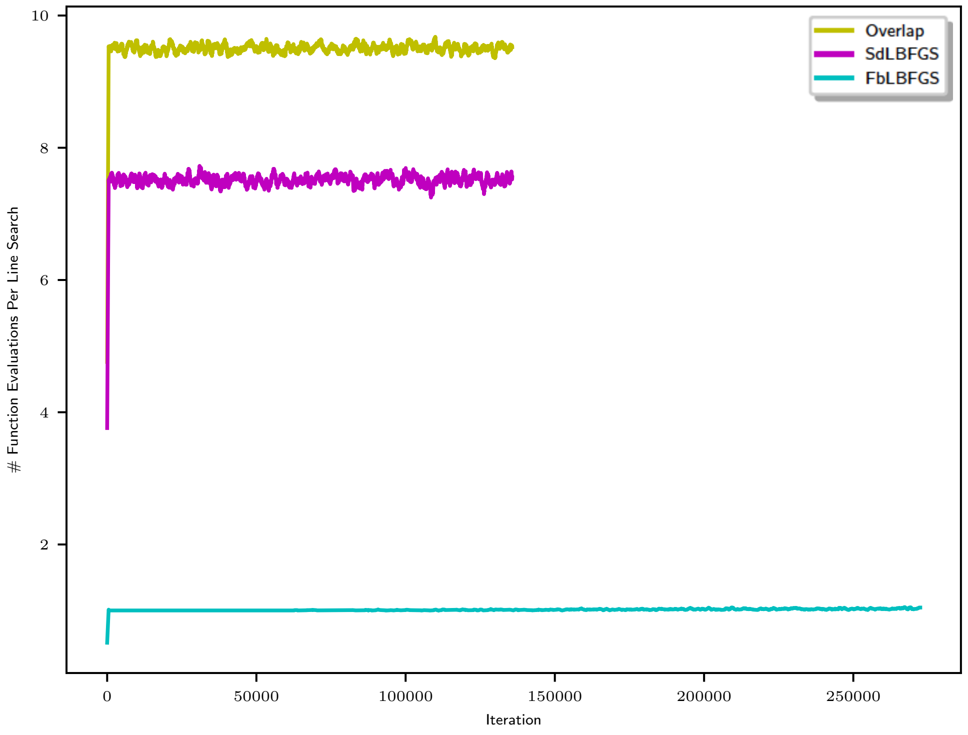


Figure 3.5: The number of function evaluations per line search for CIFAR-10 on DenseNet. The average of every consecutive 100 iteration is plotted for display purpose.

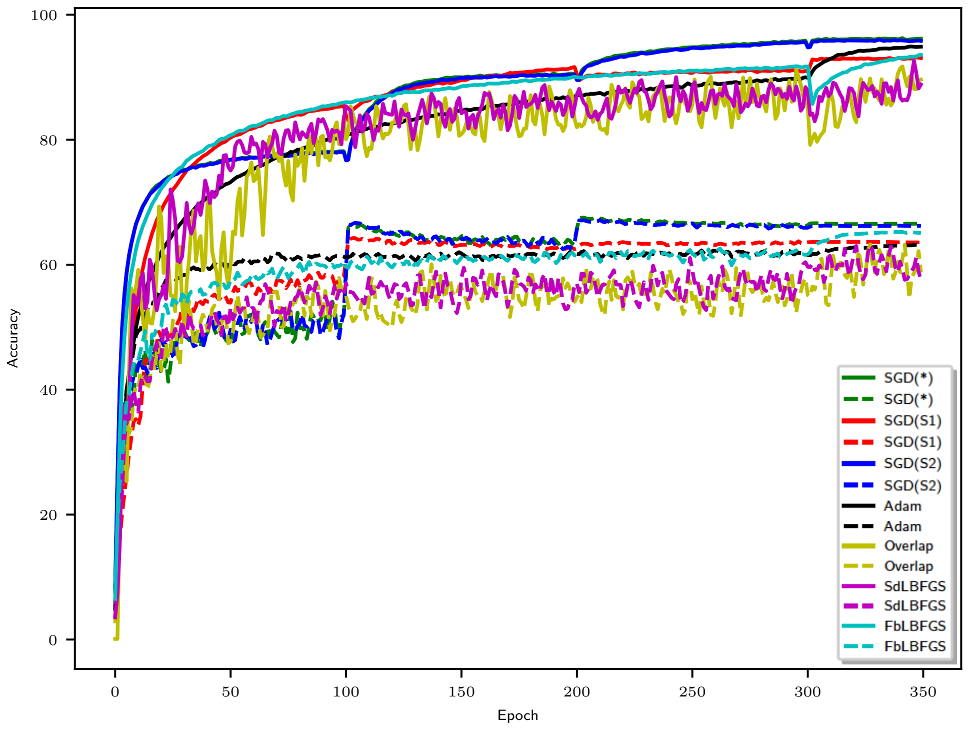


Figure 3.6: Overview of the performance of CIFAR-100 on ResNet. The solid lines represent train accuracy and dashed lines represent test accuracy, respectively.

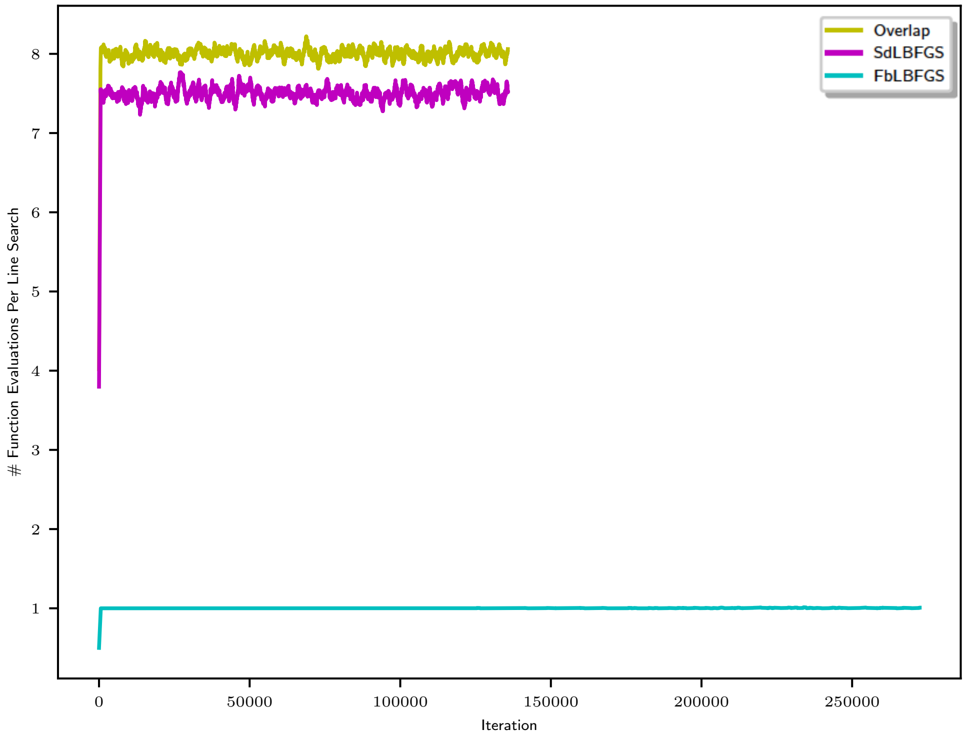


Figure 3.7: The number of function evaluations per line search for CIFAR-100 on ResNet. The average of every consecutive 100 iteration is plotted for display purpose.

3.5 Summary

It has proven challenging to effectively apply L-BFGS training methods to neural networks with batch normalization. In this chapter, we have shown how to make a simple and extremely effective modification to L-BFGS that makes it competitive with well-tuned SGD on classification tasks with BatchNorm. Our approach uses a frozen batch to ensure that all elements of the Hessian update are based on the same batch statistics. Along with a new approach to line search that reduces the number of expensive backtracking steps, we achieve results that considerably improve on previous L-BFGS implementations for training neural networks with batch normalization.

We find it a bit surprising to see L-BFGS compete with tuned SGD; the community often assumes that L-BFGS is overly aggressive, and line search methods get stuck in local minima. Interestingly, when applied to high-performance networks with BatchNorm (with the proposed modifications to maintain stability), L-BFGS does not suffer from these problems. This further supports the intuition that BatchNorm, while not understood theoretically, promotes more well behaved loss functions.

Chapter 4: Stabilizing Adversarial Nets With Prediction Methods

Adversarial neural networks solve many important problems in data science, but are notoriously difficult to train. These difficulties come from the fact that optimal weights for adversarial nets correspond to saddle points, and not minimizers, of the loss function. The alternating stochastic gradient methods typically used for such problems do not reliably converge to saddle points, and when convergence does happen it is often highly sensitive to learning rates. In this chapter, we propose a simple modification of stochastic gradient descent that stabilizes adversarial networks. We show, both in theory and practice, that the proposed method reliably converges to saddle points, and is stable with a wider range of training parameters than a non-prediction method. This makes adversarial networks less likely to "collapse," and enables faster training with larger learning rates. This work [83] is in collaboration with Sohil Shah, Zheng Xu, David Jacobs, and Tom Goldstein.

4.1 Introduction

Adversarial networks play an important role in a variety of applications, including image generation [11, 19], style transfer [19, 20, 84, 85], domain adaptation [20, 21, 22], imitation learning [86], privacy [23, 87], fair representation [12, 23], etc. One particularly motivating application of adversarial nets is their ability to form generative models, as opposed to the classical discriminative models [9, 88, 89, 90].

While adversarial networks have the power to attack a wide range of previously unsolved problems, they suffer from a major flaw: they are difficult to train. This is because adversarial nets try to accomplish two objectives simultaneously; weights are adjusted to maximize performance on one task while minimizing performance on another. Mathematically, this corresponds to finding a *saddle point* of a loss function - a point that is minimal with respect to one set of weights, and maximal with respect to another.

Conventional neural networks are trained by marching down a loss function until a minimizer is reached (Figure 1.2a). In contrast, adversarial training methods search for saddle points rather than a minimizer, which introduces the possibility that the training path "slides off" the objective functions and the loss goes to $-\infty$ (Figure 1.2b), resulting in "collapse" of the adversarial network. As a result, many authors suggest using early stopping, gradients/weight clipping [91], or specialized objective functions [9, 91, 92] to maintain stability.

In this chapter, we present a simple "prediction" step that is easily added to many training algorithms for adversarial nets. We present theoretical analysis showing that the proposed prediction method is asymptotically stable for a class of saddle point problems. Finally, we use a wide range of experiments to show that prediction enables faster training of adversarial networks using large learning rates without the instability problems that plague conventional training schemes.

4.2 Proposed Method

Saddle-point optimization problems have the general form

$$\min_u \max_v \mathcal{L}(u, v) \tag{4.1}$$

for some loss function $\mathcal{L}$ and variables u and v. Most authors use the alternating stochastic gradient method to solve saddle-point problems involving neural networks. This method alternates between updating u with a stochastic gradient *descent* step, and then updating v with a stochastic gradient *ascent* step. When simple/classical SGD updates are used, the steps of this method can be written

$$\begin{aligned} u^{k+1} &= u^k - \alpha_k \mathcal{L}'_u(u^k, v^k) \quad && | \quad \text{gradient descent in } u \text{, starting at } (u^k, v^k) \\ v^{k+1} &= v^k + \beta_k \mathcal{L}'_v(u^{k+1}, v^k) \quad && | \quad \text{gradient ascent in } v \text{, starting at } (u^{k+1}, v^k)\ . \end{aligned} \tag{4.2}$$

Here, $\{\alpha_k\}$ and $\{\beta_k\}$ are learning rate schedules for the minimization and maximization steps, respectively. The vectors $\mathcal{L}'_u(u, v)$ and $\mathcal{L}'_v(u, v)$ denote (possibly stochastic) gradients of $\mathcal{L}$ with respect to u and v. In practice, the gradient updates are often performed by an automated solver, such as the Adam optimizer [28], and include momentum updates.

We propose to stabilize the training of adversarial networks by adding a *prediction* step. Rather than calculating v^{k+1} using u^{k+1}, we first make a prediction, $\bar{u}^{k+1}$, about where the u iterates will be in the future, and use this predicted value to obtain v^{k+1}.

Prediction Method

$$\begin{aligned} u^{k+1} &= u^k - \alpha_k \mathcal{L}'_u(u^k, v^k) \quad && | \quad \text{gradient descent in } u \text{, starting at } (u^k, v^k) \\ \bar{u}^{k+1} &= u^{k+1} + (u^{k+1} - u^k) \quad && | \quad \textit{predict} \text{ future value of } u \\ v^{k+1} &= v^k + \beta_k \mathcal{L}'_v(\bar{u}^{k+1}, v^k) \quad && | \quad \text{gradient ascent in } v \text{, starting at } (\bar{u}^{k+1}, v^k)\ . \end{aligned} \tag{4.3}$$

The Prediction step (4.3) tries to estimate where u is going to be in the future by assuming its trajectory remains the same as in the current iteration.

4.3 Background

4.3.1 Adversarial Networks as a Saddle-Point Problem

We now discuss a few common adversarial network problems and their saddle-point formulations. *Generative Adversarial Networks* (GANs) fit a generative model to a dataset using a game in which a generative model competes against a discriminator [9]. The generator, $\mathbf{G}(\mathbf{z};\theta_g)$, takes random noise vectors $\mathbf{z}$ as inputs, and maps them onto points in the target data distribution. The discriminator, $\mathbf{D}(\mathbf{x};\theta_d)$, accepts a candidate point $\mathbf{x}$ and tries to determine whether it is really drawn from the empirical distribution (in which case it outputs 1), or fabricated by the generator (output 0). During a training iteration, noise vectors from a Gaussian distribution $\mathcal{G}$ are pushed through the generator network $\mathbf{G}$ to form a batch of generated data samples denoted by $\mathcal{D}_{fake}$. A batch of empirical samples, $\mathcal{D}_{real}$, is also prepared. One then tries to adjust the weights of each network to solve a saddle point problem, which is popularly formulated as,

$$\min_{\theta_g}\max_{\theta_d} \quad \mathbb{E}_{x\sim\mathcal{D}_{real}}\, f(\mathbf{D}(\mathbf{x};\theta_d)) + \mathbb{E}_{z\sim\mathcal{G}}\, f(1-\mathbf{D}(\mathbf{G}(\mathbf{z};\theta_g);\theta_d)). \tag{4.4}$$

Here $f(.)$ is any monotonically increasing function. Initially, [9] proposed using $f(x)=\log(x)$.

Domain Adversarial Networks (DANs) [22, 23, 93] take data collected from a "source" domain, and extract a feature representation that can be used to train models that generalize to another "target" domain. For example, in the domain adversarial neural network (DANN [22]), a set of feature layers maps data points into an embedded feature space, and a classifier is trained on these embedded features. Meanwhile, the adversarial discriminator tries to determine, using only the embedded features, whether the data points belong to the source or target domain. A good embedding yields a better task-specific objective on the target domain while fooling the discriminator, and is found by solving

$$\min_{\theta_f, \theta_{y^k}} \max_{\theta_d} \quad \sum_k \alpha_k \mathcal{L}_{y^k}\left(\mathbf{x}_s; \theta_f, \theta_{y^k}\right) - \lambda \mathcal{L}_d\left(\mathbf{x}_s, \mathbf{x}_t; \theta_f, \theta_d\right). \tag{4.5}$$

Here $\mathcal{L}_d$ is any adversarial discriminator loss function and $\mathcal{L}_{y^k}$ denotes the task specific loss. θ_f, θ_d, and θ_{y^k} are network parameter of feature mapping, discriminator, and classification layers.

4.3.2 Stabilizing saddle point solvers

It is well known that alternating stochastic gradient methods are unstable when using simple logarithmic losses. This led researchers to explore multiple directions for stabilizing GANs; either by adding regularization terms [91, 92, 94, 95], a myriad of training "hacks" [96, 97], re-engineering network architectures [92], and designing different solvers [98]. Specifically, the Wasserstein GAN (WGAN) [91] approach modifies the original objective by replacing $f(x) = \log(x)$ with $f(x) = x$. This led to a training scheme in which the discriminator weights are "clipped." However, as discussed in Arjovsky et al. [91], the WGAN training is unstable at high learning rates, or when used with popular momentum based solvers such as Adam. Currently, it is known to work well only with RMSProp [91].

The unrolled GAN [98] is a new solver that can stabilize training at the cost of more expensive gradient computations. Each generator update requires the computation of multiple extra discriminator updates, which are then discarded when the generator update is complete. While avoiding GAN collapse, this method requires increased computation and memory.

In the convex optimization literature, saddle point problems are more well studied. One popular solver is the primal-dual hybrid gradient (PDHG) method [99, 100], which has been popularized by Chambolle and Pock [101], and has been successfully applied to a range of machine learning and statistical estimation problems [102]. PDHG relates closely to the method proposed here - it achieves stability using the same prediction step, although it uses a different type of gradient update and is only applicable to bi-linear problems.

Stochastic methods for convex saddle-point problems can be roughly divided into two categories: stochastic coordinate descent [103, 104, 105, 106, 107, 108, 109] and stochastic gradient descent [110, 111]. Similar optimization algorithms have been studied for reinforcement learning [112, 113]. Recently, a "doubly" stochastic method that randomizes both primal and dual updates was proposed for strongly convex bilinear saddle point problems [114]. For general saddle point problems, "doubly" stochastic gradient descent methods are discussed in Nemirovski et al. [115], Palaniappan and Bach [116], in which primal and dual variables are updated simultaneously based on the previous iterates and the current gradients.

4.4 Interpretations of the prediction step

We present three ways to explain the effect of prediction: an intuitive, non-mathematical perspective, a more analytical viewpoint involving dynamical systems, and finally a rigorous proof-based approach.

4.4.1 An intuitive viewpoint

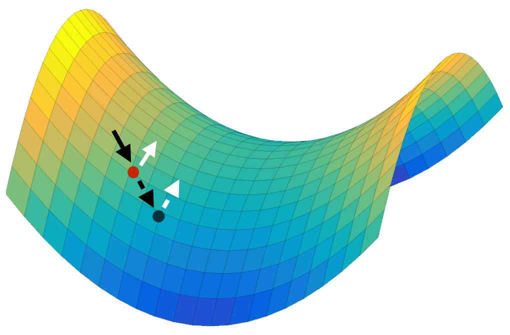
The standard alternating SGD switches between minimization and maximization steps. In this algorithm, there is a risk that the minimization step can overpower the maximization step, in which case the iterates will "slide off" the edge of saddle, leading to instability (Figure 1.2b). Conversely, an overpowering maximization step will dominate the minimization step, and drive the iterates to extreme values as well.

The effect of prediction is visualized in Figure 4.1. Suppose that a maximization step takes place starting at the red dot. Without prediction, the maximization step has no knowledge of the algorithm history, and will be the same regardless of whether the previous minimization update was weak (Figure 4.1a) or strong (Figure 4.1b). Prediction allows the maximization step to exploit information about the minimization step. If the previous minimization step was weak (Figure 4.1a), the prediction step (dotted black arrow) stays close to the red dot, resulting in a weak predictive maximization step (white arrow). But if we arrived at the red dot using a strong minimization step (Figure 4.1b), the prediction moves a long way down the loss surface, resulting in a stronger maximization step (white arrows) to compensate.

4.4.2 A more mathematical perspective

To get stronger intuition about prediction methods, let's look at the behavior of Algorithm (4.3) on a simple bi-linear saddle of the form

$$\mathcal{L}(u, v) = v^T K u \tag{4.6}$$


(a)

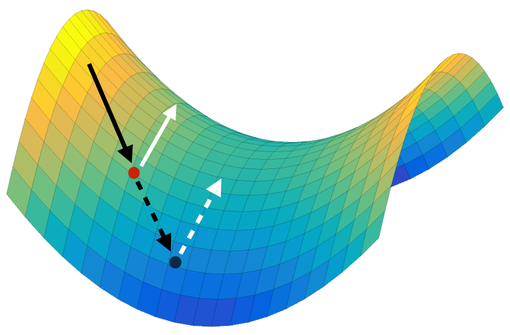

(b)

Figure 4.1: A schematic depiction of the prediction method. When the minimization step is powerful and moves the iterates a long distance, the prediction step (dotted black arrow) causes the maximization update to be calculated further down the loss surface, resulting in a more dramatic maximization update. In this way, prediction methods prevent the maximization step from getting overpowered by the minimization update.

where K is a matrix. When exact (non-stochastic) gradient updates are used, the iterates follow the path of a simple dynamical system with closed-form solutions. We give here a sketch of this argument: a detailed derivation is provided in the Appendix C.1.

When the (non-predictive) gradient method (4.2) is applied to the linear problem (4.6), the resulting iterations can be written

$$\frac{u^{k+1} - u^k}{\alpha} = -K^T v^k, \qquad \frac{v^{k+1} - v^k}{\alpha} = (\beta/\alpha)Ku^{k+1}.$$

When the stepsize α gets small, this behaves like a discretization of the system of differential equations

$$\dot{u} = -K^T v, \qquad \dot{v} = \beta/\alpha K u$$

where $\dot{u}$ and $\dot{v}$ denote the derivatives of u and v with respect to time. These equations describe a simple harmonic oscillator, and the closed form solution for u is

$$u(t) = C\cos(\Sigma^{1/2}t + \phi)$$

where Σ is a diagonal matrix, and the matrix C and vector ϕ depend on the initialization. We can see that, for small values of α and β, the non-predictive algorithm (4.2) approximates an undamped harmonic motion, and the solutions orbit around the saddle without converging.

The prediction step (4.3) improves convergence because it produces *damped* harmonic motion that sinks into the saddle point. When applied to the linearized problem (4.6), we get the dynamical system

$$\dot{u} = -K^T v, \qquad \dot{v} = \beta/\alpha K(u + \alpha\dot{u}) \tag{4.7}$$

which has solution

$$u(t) = UA\exp(-\frac{t\alpha}{2}\sqrt{\Sigma})\sin(t\sqrt{(1-\alpha^2/4)\Sigma} + \phi).$$

From this analysis, we see that the damping caused by the prediction step causes the orbits to converge into the saddle point, and the error decays exponentially fast.

4.4.3 A rigorous perspective

While the arguments above are intuitive, they are also informal and do not address issues like stochastic gradients, non-constant stepsize sequences, and more complex loss functions. We now provide a rigorous convergence analysis that handles these issues.

We assume that the function $\mathcal{L}(u, v)$ is convex in u and concave in v. We can then measure convergence using the "primal-dual" gap, $P(u, v) = \mathcal{L}(u, v^\star) - \mathcal{L}(u^\star, v)$ where $(u^\star, v^\star)$ is a saddle. Note that $P(u, v) > 0$ for non-optimal (u, v), and $P(u, v) = 0$ if (u, v) is a saddle. Using these definitions, we formulate the following convergence result. The proof is in the Appendix C.1.1.

Theorem 4. *Suppose the function $\mathcal{L}(u, v)$ is convex in u, concave in v, and that the partial gradient $\mathcal{L}'_v$ is uniformly Lipschitz smooth in u ($\|\mathcal{L}'_v(u_1, v) - \mathcal{L}'_v(u_2, v)\| \le L_v \|u_1 - u_2\|$). Suppose further that the stochastic gradient approximations satisfy $\mathbb{E}\|\mathcal{L}'_u(u, v)\|^2 \le G_u^2$, $\mathbb{E}\|\mathcal{L}'_v(u, v)\|^2 \le G_v^2$ for scalars G_u and G_v, and that $\mathbb{E}\|u^k - u^\star\|^2 \le D_u^2$, and $\mathbb{E}\|v^k - v^\star\|^2 \le D_v^2$ for scalars D_u and D_v. If we choose decreasing learning rate parameters of the form $\alpha_k = \frac{C_\alpha}{\sqrt{k}}$ and $\beta_k = \frac{C_\beta}{\sqrt{k}}$, then the SGD method with prediction converges in expectation, and we have the error bound*

$$\mathbb{E}[P(\hat{u}^l, \hat{v}^l)] \le \frac{1}{2\sqrt{l}} \left(\frac{D_u^2}{C_\alpha} + \frac{D_v^2}{C_\beta} \right) + \frac{\sqrt{l+1}}{l} \left(\frac{C_\alpha G_u^2}{2} + C_\alpha L_v G_u^2 + C_\alpha L_v D_v^2 + \frac{C_\beta G_v^2}{2} \right)$$

where $\hat{u}^l = \frac{1}{l} \sum_{k=1}^{l} u^k$, $\hat{v}^l = \frac{1}{l} \sum_{k=1}^{l} v^k$.

4.5 Experiments

We present a wide range of experiments to demonstrate the benefits of the proposed prediction step for adversarial nets. We consider a saddle point problem on a toy dataset constructed using MNIST images, and then move on to consider state-of-the-art models for three tasks: GANs, domain adaptation, and learning of fair classifiers. Additional results, and additional experiments involving mixtures of Gaussians, are presented in the Appendix C.2.

4.5.1 MNIST Toy problem

We consider the task of classifying MNIST digits as being even or odd. To make the problem interesting, we corrupt 70% of odd digits with salt-and-pepper noise, while we corrupt only 30% of even digits. When we train a LeNet network [48] on this problem, we find that the network encodes and uses information about the noise; when a noise vs no-noise classifier is trained on the deep features generated by LeNet, it gets 100% accuracy. The goal of this task is to force LeNet to ignore the noise when making decisions. We create an adversarial model of the form (4.5) in which $\mathcal{L}_y$ is a softmax loss for the even vs odd classifier. We make $\mathcal{L}_d$ a softmax loss for the task of discriminating whether the input sample is noisy or not. The classifier and discriminator were both pre-trained using the default LeNet implementation in Caffe [117]. Then the combined adversarial net was jointly trained both with and without prediction. For implementation details, see the Appendix C.2.1.

Figure 4.2 summarizes our findings. In this experiment, we considered applying prediction to both the classifier and discriminator. We note that our task is to retain good classification accuracy while preventing the discriminator from doing better than the trivial strategy of classifying odd digits as noisy and even digits as non-noisy. This means that the discriminator accuracy should ideally be ~ 0.7. As shown in Figure 4.2a, the prediction step hardly makes any difference when evaluated at the small learning rate of 10^{-4}. However, when evaluated at higher rates, Figures 4.2b and 4.2c show that the prediction solvers are very stable while one without prediction collapses (blue solid line is flat) very early. Figure 4.2c shows that the default learning rate (10^{-3}) of the Adam solver is unstable unless prediction is used.

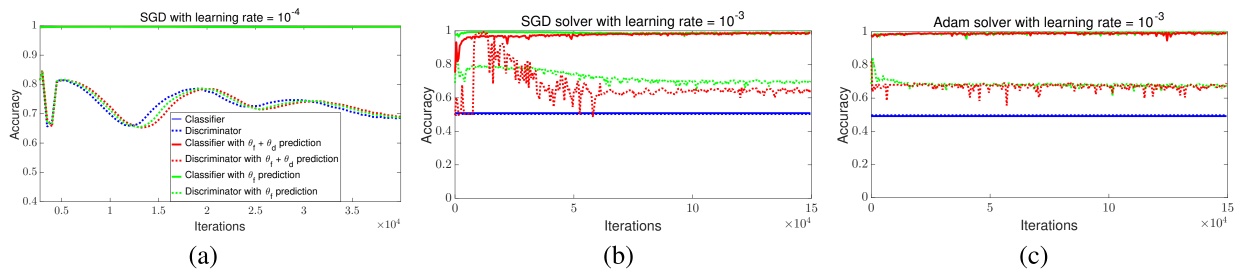


Figure 4.2: Comparison of the classification accuracy (digit parity) and discriminator (noisy vs. no-noise) accuracy using SGD and Adam solver with and without prediction steps. θ_f and θ_d refers to variables in eq. (4.5). (a) Using SGD with learning rate $lr = 10^{-4}$. Note that the solid lines of red, blue and green are overlapped. (b) SGD solver with higher learning rate of $lr = 10^{-3}$, and (c) using Adam solver with its default parameter.

4.5.2 Generative Adversarial Networks

Next, we test the efficacy and stability of our proposed predictive step on generative adversarial networks (GAN), which are formulated as saddle point problems (4.4) and are popularly solved using a heuristic approach [9]. We consider an image modeling task using CIFAR-10 [118] on the recently popular convolutional GAN architecture, DCGAN [88]. We compare our predictive method with that of DCGAN and the unrolled GAN [98] using the training protocol described in Radford et al. [88]. Note that we compared against the unrolled GAN with stop gradient switch[1] and $K = 5$ unrolling steps. All the approaches were trained for five random seeds and 100 epochs each.

We start with comparing all three methods using the default solver for DCGAN (the Adam optimizer) with learning rate=0.0002 and β_1=0.5. Figure 4.3 compares the generated sample images (at the 100^{th} epoch) and the training loss curve for all approaches. The discriminator and generator loss curves in Figure 4.3e show that without prediction, the DCGAN collapses at the 45^{th} and 57^{th} epochs. Similarly, Figure 4.3f shows that the training for unrolled GAN collapses in at

[1]We found the unrolled GAN without stop gradient switch as well as for smaller values of K collapsed when used on the DCGAN architecture.

least three instances. The training procedure using predictive steps never collapsed during any epochs. Qualitatively, the images generated using prediction are more diverse than the DCGAN and unrolled GAN images.

Figure 4.4 compares all approaches when trained with $5\times$ higher learning rate (0.001) (the default for the Adam solver). As observed in Radford et al. [88], the standard and unrolled solvers are very unstable and collapse at this higher rate. However, as shown in Figure 4.4d, & 4.4a, training remains stable when a predictive step is used, and generates images of reasonable quality. The training procedure for both DCGAN and unrolled GAN collapsed on all five random seeds. The results on various additional intermediate learning rates are in the Appendix C.2.4.2.

In the the Appendix C.2.4.2, we present one additional comparison showing results on a higher momentum of β_1=0.9 (learning rate=0.0002). We observe that all the training approaches are stable. However, the quality of images generated using DCGAN is inferior to that of the predictive and unrolled methods.

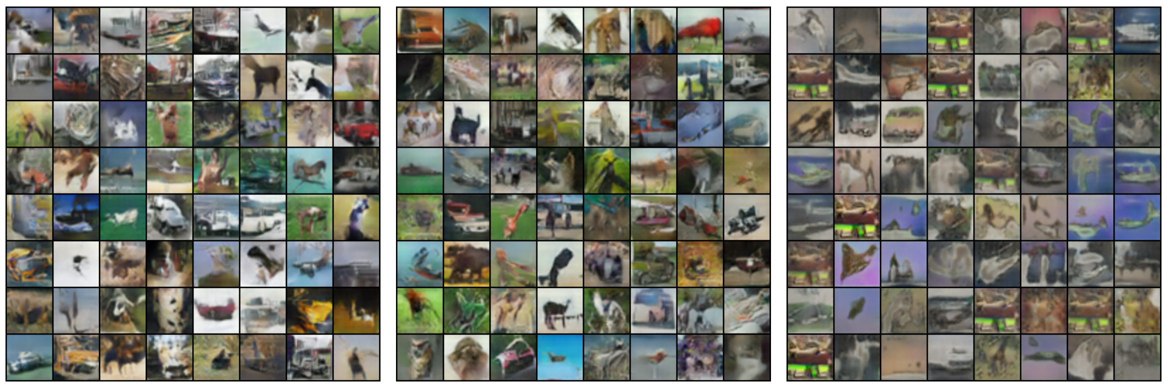
Overall, of the 25 training settings we ran on (each of five learning rates for five random seeds), the DCGAN training procedure collapsed in 20 such instances while unrolled GAN collapsed in 14 experiments (not counting the multiple collapse in each training setting). On the contrary, we find that our simple predictive step method collapsed only once.

Note that prediction adds trivial cost to the training algorithm. Using a single TitanX Pascal, a training epoch of DCGAN takes 35 secs. With prediction, an epoch takes 38 secs. The unrolled GAN method, which requires extra gradient steps, takes 139 secs/epoch.

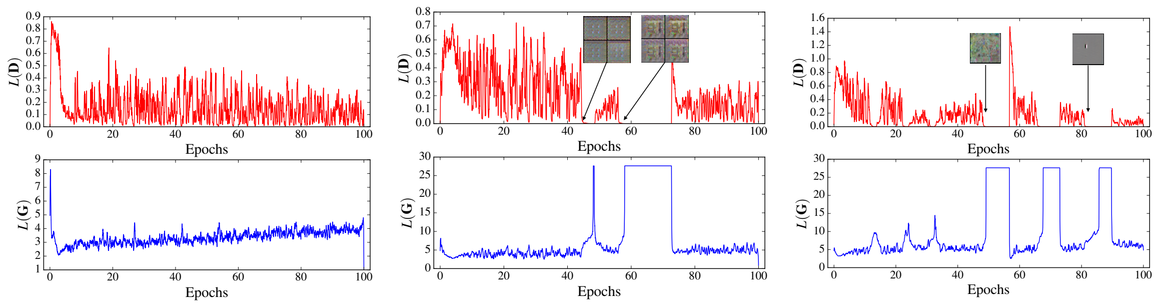
Finally, we draw quantitative comparisons based on the inception score [96], which is a widely used metric for visual quality of the generated images. For this purpose, we consider the current state-of-the-art Stacked GAN [119] architecture. Table 4.1 lists the inception scores computed on the generated samples from Stacked GAN trained (200 epochs) with and without prediction at different learning rates. The joint training of Stacked GAN collapses when trained at the default learning rate of adam solver (i.e., 0.001). However, reasonably good samples are generated if the same is trained with prediction on both the generator networks. The right end of Table 4.1 also list the inception score measured at fewer number of epochs at higher learning rates. It suggest that the model trained with prediction methods are not only stable but also allows faster convergence using higher learning rates. For reference the inception score on real images of CIFAR-10 dataset is 11.51 ± 0.17.

Table 4.1: Comparison of Inception Score on Stacked GAN network with and w/o **G** prediction.

Learning rate	0.0001	0.0005	0.001	0.0005 (40)	0.001 (20)
Stacked GAN (joint)	8.44 ± 0.11	7.90 ± 0.08	1.52 ± 0.01	5.80 ± 0.15	1.42 ± 0.01
Stacked GAN (joint) + prediction	$\mathbf{8.55} \pm 0.12$	$\mathbf{8.13} \pm 0.09$	$\mathbf{7.96} \pm 0.11$	$\mathbf{8.10} \pm 0.10$	$\mathbf{7.79} \pm 0.07$



(a) With **G** prediction (b) DCGAN (c) Unrolled GAN



(d) With **G** prediction (e) DCGAN (f) Unrolled GAN

Figure 4.3: Comparison of GAN training algorithms for DCGAN architecture on Cifar-10 image datasets. Using default parameters of DCGAN; $lr = 0.0002, \beta_1 = 0.5$.

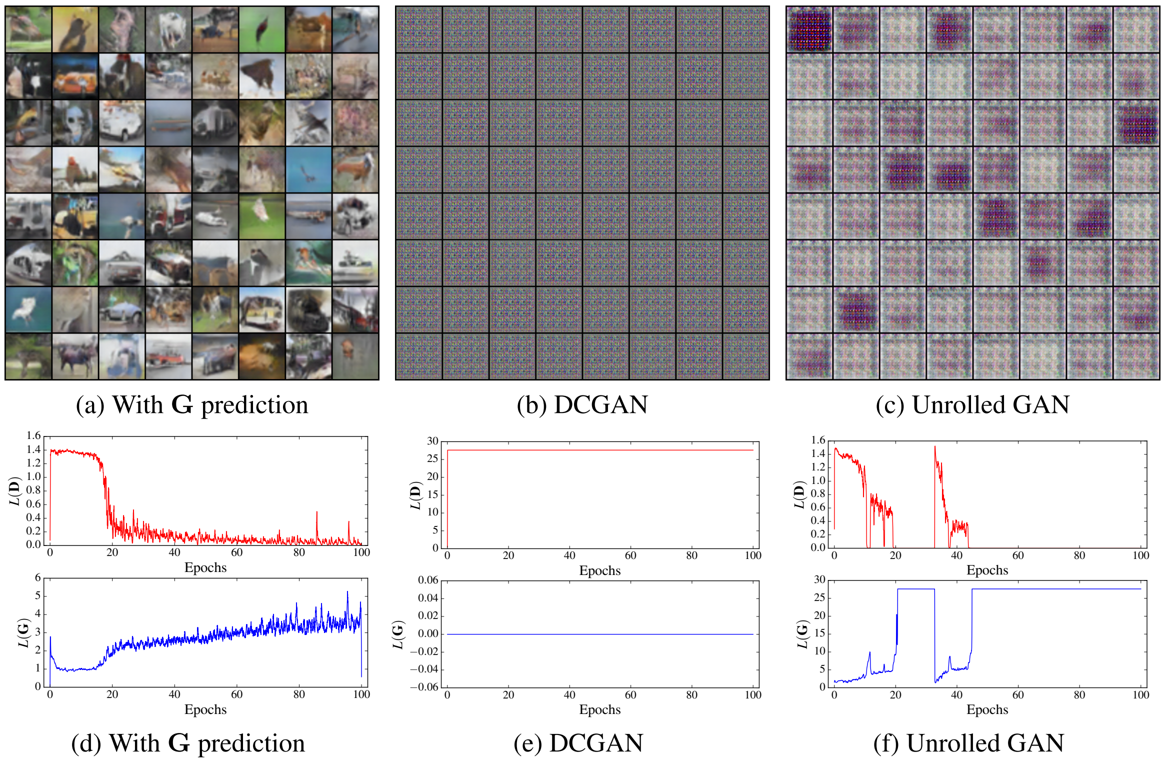


Figure 4.4: Comparison of GAN training algorithms for DCGAN architecture on Cifar-10 image datasets with higher learning rate, $lr = 0.001, \beta_1 = 0.5$.

4.5.3 Domain Adaptation

We consider the domain adaptation task [21, 22, 120] wherein the representation learned using the source domain samples is altered so that it can also generalize to samples from the target distribution. We use the problem setup and hyper-parameters as described in [22] using the OFFICE dataset [120] (experimental details are shared in the Appendix C.2.3). In Table 4.2, comparisons are drawn with respect to target domain accuracy on six pairs of source-target domain tasks. We observe that the prediction step has mild benefits on the "easy" adaptation tasks with very similar source and target domain samples. However, on the transfer learning tasks of AMAZON-to-WEBCAM, WEBCAM-to-AMAZON, and DSLR-to-AMAZON which has noticeably distinct data samples, an extra prediction step gives an absolute improvement of $1.3 - 6.9\%$ in predicting target domain labels.

Table 4.2: Comparison of target domain accuracy on OFFICE dataset.

Method	Source	AMAZON	WEBCAM	DSLR	WEBCAM	AMAZON	DSLR
	Target	WEBCAM	AMAZON	WEBCAM	DSLR	DSLR	AMAZON
DANN [22]		73.4	51.6	95.5	**99.4**	**76.5**	51.7
DANN + prediction		**74.7**	**58.5**	**96.1**	99.0	73.5	**57.6**

4.5.4 Fair Classifier

Finally, we consider a task of learning fair feature representations [12, 23, 121] such that the final learned classifier does not discriminate with respect to a sensitive variable. As proposed in Edwards and Storkey [23] one way to measure fairness is using discrimination,

$$y_{disc} = \left| \frac{1}{N_0} \sum_{i:s_i=0} \eta(x_i) - \frac{1}{N_1} \sum_{i:s_i=1} \eta(x_i) \right| . \tag{4.8}$$

Here s_i is a binary sensitive variable for the i^{th} data sample and N_k denotes the total number of samples belonging to the k^{th} sensitive class. Similar to the domain adaptation task, the learning of each classifier can be formulated as a minimax problem in (4.5) [12, 23]. Unlike the previous example though, this task has a model selection component. From a pool of hundreds of randomly generated adversarial deep nets, for each value of t, one selects the model that maximizes the difference

$$y_{t,Delta} = y_{acc} - t * y_{disc}. \tag{4.9}$$

The "Adult" dataset from the UCI machine learning repository is used. The task (y_{acc}) is to classify whether a person earns $\geq \$50k$/year. The person's gender is chosen to be the sensitive variable. Details are in the Appendix C.2.3. To demonstrate the advantage of using prediction for model selection, we follow the protocol developed in Edwards and Storkey [23]. In this work, the search space is restricted to a class of models that consist of a fully connected autoencoder, one task specific discriminator, and one adversarial discriminator. The encoder output from the autoencoder acts as input to both the discriminators. In our experiment, 100 models are randomly selected. During the training of each adversarial model, $\mathcal{L}_d$ is a cross-entropy loss while $\mathcal{L}_y$ is a linear combination of reconstruction and cross-entropy loss. Once all the models are trained, the best model for each value of t is selected by evaluating (4.9) on the validation set.

Figure 4.5a plots the results on the test set for the AFLR approach with and without prediction steps in their default Adam solver. For each value of t, Figure 4.5b, 4.5c also compares the number of layers in the selected encoder and discriminator networks. When using prediction for training, relatively stronger encoder models are produced and selected during validation, and hence the prediction results generalize better on the test set.

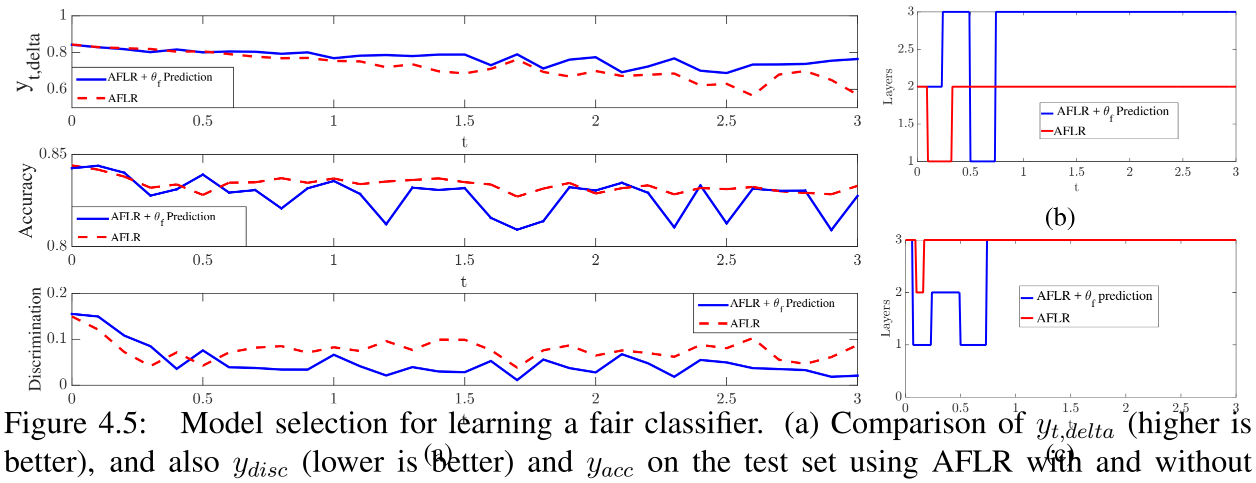


Figure 4.5: Model selection for learning a fair classifier. (a) Comparison of $y_{t,delta}$ (higher is better), and also y_{disc} (lower is better) and y_{acc} on the test set using AFLR with and without predictive steps. (b) Number of encoder layers in the selected model. (c) Number of discriminator layers (both adversarial and task-specific) in the selected model.

4.6 Summary

In this chapter, we presented a simple modification to the alternating SGD method, called a prediction step, that improves the stability of adversarial networks. We also presented theoretical results showing that the prediction step is asymptotically stable for solving saddle point problems. We demonstrated, using a variety of test problems, that prediction steps prevent network collapse and enable training with a wider range of learning rates than plain SGD methods.

Chapter 5: On the Similarity between the Laplace and Neural Tangent Kernels

Recent theoretical work has shown that massively overparameterized neural networks are equivalent to kernel regressors that use *Neural Tangent Kernels* (NTKs). Experiments show that these kernel methods perform similarly to real neural networks. However, memory and time requirements for NTK computation grow linearly with the number of layers (because of its recursive definition), limiting its further exploration and application. This chapter caters to this problem by showing that NTK for fully connected networks with ReLU activation is closely related to the standard Laplace kernel.

In particular, we show theoretically that for normalized data on the hypersphere both kernels have the same eigenfunctions and their eigenvalues decay polynomially at the same rate, implying that their Reproducing Kernel Hilbert Spaces (RKHS) include the same sets of functions. This means that both kernels give rise to classes of functions with the same smoothness properties. The two kernels differ for data off the hypersphere, but experiments indicate that when data is properly normalized these differences are not significant. Finally, we provide experiments on real data comparing NTK and the Laplace kernel, along with a larger class of γ-exponential kernels. We show that these perform almost identically. Our results also suggest that much insight about neural networks can be obtained from analysis of the well-known Laplace kernel, which has a simple closed form. This work [122] is in collaboration with Amnon Geifman, Yoni Kasten, Meirav

Galun, David Jacobs, and Basri Ronen.

5.1 Introduction

Neural networks with significantly more parameters than training examples have been successfully applied to a variety of tasks. Somewhat contrary to common wisdom, these models typically generalize well to unseen data. It has been shown that in the limit of infinite model size, these neural networks are equivalent to kernel regression using a family of novel *Neural Tangent Kernels* (NTK) [123, 124]. NTK methods can be analyzed to explain many properties of neural networks in this limit, including their convergence in training and ability to generalize [125, 126, 127, 128]. Recent experimental work has shown that in practice, kernel methods using NTK perform similarly, and in some cases better, than neural networks [129], and that NTK can be used to accurately predict the dynamics of neural networks [124, 130, 131]. However, NTK computation is expensive. For example, roughly 5x time and memory computation is needed for a 5 layered infinite width deep network equivalent NTK. These observations suggest that a better understanding of NTK can lead to new ways to compute NTK as well as analyze neural networks.

To this end, we ask the following important question: Is NTK significantly different from standard kernels? For the case of fully connected (FC) networks, [129] provides experimental evidence that NTK is especially effective, showing that it outperforms the Gaussian kernel on a large suite of machine learning problems. Consequently, they argue that NTK should be added to the standard machine learning toolbox. [126] has shown empirically that the dynamics of neural networks on randomly labeled data more closely resembles the dynamics of learning through stochastic gradient descent with the Laplace kernel than with the Gaussian kernel. In this chapter, we show

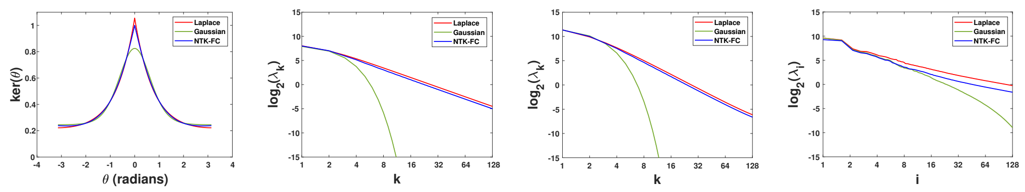


Figure 5.1: Left: An overlay of the NTK for a 6-layer FC network with ReLU activation with the Laplace and Gaussian kernels, as a function of the angle between their arguments. The exponential kernels are modulated by an affine transformation to achieve a least squares fit to the NTK. Note the high degree of similarity between the Laplace kernel and NTK. Middle left: eigenvalues as a function of frequency in $\mathbb{S}^1$. The slopes in these log-log plots indicate the rate of decay, which is similar for both the Laplace kernel and for NTK for the FC network with 6 layers. (Empirical slopes are -1.94 for both Laplace and NTK-FC.) The eigenvalues of the Gaussian kernel, in contrast, decay exponentially. Middle right: Same for $\mathbb{S}^2$. (Empirical slopes are -2.75 for the Laplace and NTK-FC.) Right: Same estimated for the UCI Abalone dataset (here we show eigenvalues as function of eigenvalue index).

theoretically and experimentally that NTK does closely resemble the Laplace kernel, already a standard tool of machine learning.

Kernels are mainly characterized by their corresponding Reproducing Kernel Hilbert Space (RKHS), which determines the set of functions they can produce [132]. They are further characterized by the RKHS norm they induce, which is minimized (implicitly) in every regression problem. Our main result is that when restricted to the hypersphere $\mathbb{S}^{d-1}$, NTK for a fully connected (FC) network with ReLU activation and bias has the same RKHS as the Laplace kernel, defined as $\boldsymbol{k}^{\mathrm{Lap}}(\mathbf{x},\mathbf{z}) = e^{-c\|\mathbf{x}-\mathbf{z}\|}$ for points $\mathbf{x},\mathbf{z} \in \mathbb{S}^{d-1}$ and constant $c > 0$. (In general, NTK for deeper networks is more sharply peaked, corresponding to larger values of c, see Appendix D.) This equivalence of RKHSs is shown by establishing that on the hypersphere the eigenfunctions of NTK and the Laplace kernels coincide and their eigenvalues decay at the same rate (see Figure 5.1), implying in turn that gradient descent (GD) with both kernels should have the same dynamics, explaining [126]'s experiments. In previous work, the eigenfunctions and eigenvalues of NTK have been derived on the hypersphere for networks with only one hidden layer, while these properties of the Laplace kernel have been studied in $\mathbb{R}^d$. We derive new results for the Laplace kernel on the hypersphere, and for NTK for deep networks on the hypersphere and in $\mathbb{R}^d$. In $\mathbb{R}^d$, NTK gives rise to radial eigenfunctions, forgoing the shift invariance property of exponential kernels. Experiments indicate that this difference is not significant in practice.

Finally, we show experiments indicating that the Laplace kernel achieves similar results to those obtained with NTK on real-world problems. We further show that by using the more general, γ-exponential kernel [133], which allows for one additional parameter, $\boldsymbol{k}^{\gamma}(\mathbf{x}, \mathbf{z}) = e^{-c\|\mathbf{x}-\mathbf{z}\|^{\gamma}}$, we achieve slightly better performance than NTK on a number of standard datasets.

5.2 Related Works

The connection between neural networks and kernel methods has been investigated for over two decades. Early works have noted the equivalence between neural networks with single hidden layers of infinite width and Gaussian Processes (GP) [134, 135], where GP prior can be used to achieve exact Bayesian inference. Recently [136, 137] have extended the results to deep fully-connected neural networks in which all but the last layer retain their initial values. In this context, [138] introduced the Arc-cosine kernel, while [139] showed a duality between neural networks and compositional kernels.

More recent work introduced the family of neural tangent kernels (NTK) [123, 124]. This work showed that for massively overparameterized and fully trained networks, their training dynamics closely follows the path of kernel gradient descent, and that training converges to the solution of kernel regression with NTK. Follow-up work defined analogous kernels for residual [140] and convolutional networks [124, 141]. Recent work also showed empirically that classification with NTK achieves performance similar to deep neural networks with the corresponding architecture

[124, 141].

The equivalence between kernels and overparameterized neural networks opened the door to studying inductive bias in neural networks. For two layer, FC networks, [131, 142, 143] investigated the spectral property of the NTK when the data is distributed uniformly on the hypersphere, showing in particular that with GD low frequencies are learned before higher ones. [144] extended these results to non-uniform distributions. [145] analyzed the eigenvalues of NTK over the Boolean cube, and [146] analyzed its spectrum under approximate pairwise orthogonality. [142, 147] further leveraged the spectral properties of the kernels to investigate their RKHS in the case of bias-free two layer networks. Our results apply to deep networks with bias. [148] studied approximation bounds for two layer neural networks, and [125, 126, 127, 128] studied generalization properties of kernel methods in the context of neural networks.

Positive definite kernels and their associated RKHSs have been studied extensively, see, e.g., [149, 150] for reviews. The spectral properties of classic kernels, e.g., the Gaussian and Laplace kernels, are typically derived for input in $\mathbb{R}^d$ [151]. Several papers examine the RKHS of common kernels (e.g., the Gaussian and polynomial) on the hypersphere [152, 153, 154].

Recent work compares the performance of NTK to that of common kernels. Specifically, [129]'s experiments suggest that NTK is superior to the Gaussian and low degree polynomial kernels. [126] compares the learning speed of GD for randomly mislabeled data, showing that NTK learns such data as fast as the Laplace kernel and much faster than the Gaussian kernel. Our analysis provides a theoretical justification of this result.

5.3 NTK vs. the Exponential Kernels

Our aim is to compare NTK to common kernels. In comparing kernels we need to consider two main properties: first, what functions are included in their respective RKHS and secondly, how their respective norms behave. (These concepts are reviewed below in Sec. 5.3.1.) The answer to the former question determines the set of functions considered for regression, while the answer to the latter determines the result of regression. Together, these will determine how a kernel generalizes to unseen data points. Below we see that on the hypersphere both NTK and exponential kernels (e.g., Gaussian and Laplace) give rise to the same set of eigenfunctions. Therefore, the answers to the questions above are determined fully by the corresponding eigenvalues. Moreover, the asymptotic decay rate of the eigenvalues of each kernel determines their RKHS.

As an example consider the exponential kernels in $\mathbb{R}^d$, i.e., the kernels $e^{-c\|\mathbf{x}-\mathbf{z}\|^\gamma}$, where $c > 0$ and $0 < \gamma \leq 2$ [133]. These shift invariant kernels have the Fourier transform as their eigenfunctions. The eigenvalues of the Gaussian kernel, i.e., $\gamma = 2$, decay exponentially, implying that its respective RKHS includes only infinitely smooth functions. In contrast, the eigenvalues of the Laplace kernel, i.e., $\gamma = 1$, decay polynomially, forming a space of continuous, but not necessarily smooth functions.

Our main theoretical result is that when restricted to the hypersphere $\mathbb{S}^{d-1}$

$$\mathcal{H}^{\text{Gauss}} \subset \mathcal{H}^{\text{Lap}} = \mathcal{H}^{\text{FC}_\beta(2)} \subseteq \mathcal{H}^{\text{FC}_\beta(\text{L})},$$

where $\mathcal{H}^{\text{Gauss}}$ and $\mathcal{H}^{\text{Lap}}$ denote the RKHSs associated with the Gaussian and Laplace kernels, and $\mathcal{H}^{\text{FC}_\beta(\text{L})}$ denotes the NTK for a FC network with L layers, ReLU activation, and bias. Further empirical results indicate that $\mathcal{H}^{\text{Lap}} = \mathcal{H}^{\text{FC}_\beta(\text{L})}$ for the entire range $L \geq 2$. Indeed, the subsequent work of [155] proves that $\mathcal{H}^{\text{FC}_\beta(2)} \supseteq \mathcal{H}^{\text{FC}_\beta(\text{L})}$, thus together with our results proving that $\mathcal{H}^{\text{FC}_\beta(2)} = \mathcal{H}^{\text{FC}_\beta(\text{L})}$.

Next we briefly recall basic concepts in kernel regression. We subsequently characterize the RKHS of NTK and the Laplace kernel and show their equivalence in $\mathbb{S}^{d-1}$. Finally, we discuss how these kernels extend outside of the sphere to the entire $\mathbb{R}^d$ space. All lemmas and theorems are proved in the Appendix D.

5.3.1 Preliminaries

We consider positive definite kernels $\boldsymbol{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ defined over a compact metric space $\mathcal{X}$ endowed with a finite Borel measure $\mathcal{V}$. Each such kernel is associated with a Reproducing Kernel Hilbert Space (RKHS) of functions, $\mathcal{H}$, which includes the set of functions the kernel reproduces, i.e., $f(\mathbf{x}) = \langle f, \boldsymbol{k}(\cdot, \mathbf{x})\rangle_{\mathcal{H}}$ where the inner product is inherited from the respective Hilbert space. For such kernels the following holds:

1. For all $x \in \mathcal{X}$ we have that the $\boldsymbol{k}(\cdot, x) \in \mathcal{H}$.
2. Reproducing property: for all $x \in \mathcal{X}$ and for all $f \in \mathcal{H}$ it holds that $f(x) = \langle f, \boldsymbol{k}(\cdot, x)\rangle_{\mathcal{H}}$.

Moreover, RKHSs and positive definite kernels are uniquely paired.

According to Mercer's theorem $\boldsymbol{k}$ can be written as

$$\boldsymbol{k}(\mathbf{x}, \mathbf{z}) = \sum_{i \in I} \lambda_i \Phi_i(\mathbf{x}) \Phi_i(\mathbf{z}), \quad \mathbf{x}, \mathbf{z} \in \mathcal{X}, \tag{5.1}$$

where $\{(\lambda_i, \Phi_i)\}_{i\in I}$ are the eigenvalues and eigenfunctions of $\boldsymbol{k}$ with respect to the measure $\mathcal{V}$, i.e.,

$$\int k(\mathbf{x},\mathbf{z})\Phi_i(\mathbf{z})d\mathcal{V}(\mathbf{z}) = \lambda_i\Phi_i(\mathbf{x}).$$

The RKHS $\mathcal{H}$ is the space of functions $f \in \mathcal{H}$ of the form $f(\mathbf{x}) = \sum_{i\in I}\alpha_i\Phi_i(\mathbf{x})$ whose RKHS norm is finite, i.e., $\|f\|_{\mathcal{H}} = \sum_{i\in I}\frac{\alpha_i^2}{\lambda_i} < \infty$. The latter condition restricts the set of functions in an RKHS, allowing only functions that are sufficiently smooth in accordance with the asymptotic decay of λ_k.

The literature considers many different kernels (see, e.g., [156]). Here we discuss the family of γ-exponential kernels $\boldsymbol{k}^{\gamma}(\mathbf{x},\mathbf{z}) = e^{-c\|\mathbf{x}-\mathbf{z}\|^{\gamma}}$, $0 < \gamma \leq 2$, which include the Laplace ($\gamma = 1$) and the Gaussian ($\gamma = 2$) kernels.

Neural Tangent Kernel. Let $f(\theta,\mathbf{x})$ denote a neural network function with ReLU activation and trainable parameters θ. Then the corresponding NTK is defined as

$$\boldsymbol{k}^{\mathrm{NTK}}(\mathbf{x},\mathbf{z}) = \mathbb{E}_{\theta\sim\mathcal{P}}\left\langle \frac{\partial f(\theta,\mathbf{x})}{\partial\theta}, \frac{\partial f(\theta,\mathbf{z})}{\partial\theta}\right\rangle,$$

where expectation is taken over the probability distribution $\mathcal{P}$ of the initialization of θ, and we assume that the width of each layer tends to infinity. Our results focus on NTK kernels corresponding to deep, fully connected network architectures that may or may not include bias, where bias, if it exists, is initialized at zero. We denote these kernels by $\boldsymbol{k}^{\mathrm{FC}_0(\mathrm{L})}$ for the bias-free version and $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$ for NTK with bias and define them in the Appendix D.1.

Kernel regression. Given training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathbb{R}$, kernel ridge regression is the solution to

$$\min_{f \in \mathcal{H}} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2 + \lambda \|f\|_{\mathcal{H}}^2. \tag{5.2}$$

When $\lambda \to 0$ this problem is called minimum norm interpolant, and the solution satisfies

$$\min_{f \in \mathcal{H}} \|f\|_{\mathcal{H}} \quad \text{s.t.} \quad \forall i,\ f(\mathbf{x}_i) = y_i. \tag{5.3}$$

The solution of (5.2) is given by $f(\mathbf{x}) = \boldsymbol{k}_{\mathbf{x}}^T (K + \lambda I)^{-1} \mathbf{y}$, where the entries of $\boldsymbol{k}_{\mathbf{x}} \in \mathbb{R}^n$ are $\boldsymbol{k}(\mathbf{x}, \mathbf{x}_i)$, K is the $n \times n$ matrix with $K_{ij} = \boldsymbol{k}(\mathbf{x}_i, \mathbf{x}_j)$, I denotes the identity matrix, and $\mathbf{y} = (y_1, ..., y_n)^T$. Further review of kernel methods can be found, e.g., in [132, 149].

5.3.2 NTK in $\mathbb{S}^{d-1}$

We next consider the NTK for fully connected networks applied to data restricted to the hypersphere $\mathbb{S}^{d-1}$. To characterize the kernel, we first aim to determine the eigenvectors of NTK. This will be a direct consequence of Lemma 3. Subsequently in Theorem 5 we will characterize the decay rate of the corresponding eigenvalues.

Lemma 3. *Let $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}(\mathbf{x}, \mathbf{z})$, $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$, denote the NTK kernels for FC networks with $L \geq 2$ layers, possibly with bias initialized with zero. This kernel is zonal, i.e., $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}(\mathbf{x}, \mathbf{z}) = \boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}(\mathbf{x}^T\mathbf{z})$. (Note the abuse of notation, which should be clear by context.)*

We note that for the bias-free $\boldsymbol{k}^{\mathrm{FC}_0(\mathrm{L})}$ this lemma was proven in [144] and we extend the proof to allow for bias. It is well known that the spherical harmonics are eigenvectors for any zonal kernel with respect to the uniform measure on $\mathbb{S}^{d-1}$ with $d \geq 3$. (For background on Spherical Harmonics see, e.g., [157]). Therefore, due to Mercer's Theorem (5.1), any zonal kernel $\boldsymbol{k}$ can be written as

$$\boldsymbol{k}(\mathbf{x}, \mathbf{z}) = \sum_{k=0}^{\infty} \lambda_k \sum_{j=1}^{N(d,k)} Y_{k,j}(\mathbf{x}) Y_{k,j}(\mathbf{z}), \tag{5.4}$$

where $Y_{k,j}(.)$ denotes the spherical harmonics of $\mathbb{S}^{d-1}$, $N(d,k)$ denotes the number of harmonics of order k in $\mathbb{S}^{d-1}$, and λ_k are the respective eigenvalues. On the circle $\mathbb{S}^1$ the eigenvectors are the Fourier series, and $\boldsymbol{k}(\mathbf{x}, \mathbf{z}) = \sum_{k=0}^{\infty} \frac{1}{c_k} \lambda_k \cos(k\theta)$, where $\theta = \arccos(\mathbf{x}^T \mathbf{z})$ and c_k is a normalization factor, $c_0 = 4\pi^2$ and $c_k = \pi^2$ when $k \geq 1$.

Deriving the eigenvalues for NTK for deep networks is complicated, due to its recursive definition. For a two-layer network without bias, [142, 147] proved that the eigenvalues decay at a rate of $O(k^{-d})$. With no bias, however, two-layer networks are nonuniversal, and in particular $\lambda_k = 0$ for odd $k \geq 3$ [131]. To avoid this issue Theorem 5 establishes that with bias NTK is universal for any number of layers $L \geq 2$, and its eigenvalues decay at a rate no faster than $O(k^{-d})$. Moreover, with $L = 2$ the eigenvalues decay exactly at the rate of $O(k^{-d})$.

Theorem 5. *Let $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$. With bias initialized at zero:*

1. *$\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$ decomposes according to (5.4) with $\lambda_k > 0$ for all $k \geq 0$, and*

2. *$\exists k_0$ and constants $C_1, C_2, C_3 > 0$ that depend on the dimension d such that $\forall k > k_0$*

   (a) $C_1 k^{-d} \leq \lambda_k \leq C_2 k^{-d}$ if $L = 2$, and

   (b) $C_3 k^{-d} \leq \lambda_k$ if $L \geq 3$.

The proof of this theorem for $L = 2$ borrows techniques from [142]. The proof for $L \geq 3$ relies mainly on showing that the algebraic operations in the recursive definition of NTK (including addition, product and composition) do not increase the rate of decay. The consequence of Theorem 5 is that NTK for FC networks gives rise to an infinite size feature space and that its eigenvalues decay no faster than $O(k^{-d})$. While our proofs only establish a bound for the case that $L \geq 3$, empirical results suggest that the eigenvalues for these kernels decay exactly as $\Theta(k^{-d})$, as can be seen in Figure 5.1.

5.3.3 NTK vs. exponential kernels in $\mathbb{S}^{d-1}$

The polynomial decay of the eigenvalues of NTK suggests that NTK is closely related to the Laplace kernel, as we show next. Indeed, any shift invariant and isotropic kernel, i.e., $\boldsymbol{k}(\mathbf{x}, \mathbf{y}) = \boldsymbol{k}(\|\mathbf{x}-\mathbf{y}\|)$, in $\mathbb{R}^d$ is zonal when restricted to the hypersphere, since $\mathbf{x}, \mathbf{y} \in \mathbb{S}^{d-1}$ implies $\|\mathbf{x}-\mathbf{y}\|^2 = 2(1-\mathbf{x}^T\mathbf{y})$. Therefore, in $\mathbb{S}^{d-1}$ the spherical harmonics are the eigenvectors of the exponential kernels.

[152] shows that the Gaussian kernel restricted to the hypersphere yields eigenvalues that decay exponentially fast. In contrast we next prove that the eigenvalues of the Laplace kernel restricted to the hypersphere decay polynomially as $\Theta(k^{-d})$, the same decay rate shown for NTK in Theorem 5 and in Figure 5.1.

Theorem 6. *Let $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$ and write the Laplace kernel as $\boldsymbol{k}^{\mathrm{Lap}}(\mathbf{x}^T\mathbf{z}) = e^{-c\sqrt{1-\mathbf{x}^T\mathbf{z}}}$, restricted to $\mathbb{S}^{d-1}$. Then $\boldsymbol{k}^{\mathrm{Lap}}$ can be decomposed as in (5.4) with the eigenvalues λ_k satisfying $\lambda_k > 0$, and $\exists k_0$ such that $\forall k > k_0$ it holds that:*

$$B_1 k^{-d} \leq \lambda_k \leq B_2 k^{-d}$$

where $B_1, B_2 > 0$ are constants that depend on the dimension d and the parameter c.

Our proof uses the decay rate of the Laplace kernel in $\mathbb{R}^d$, and results due to [153, 154] that relate Fourier expansions in $\mathbb{R}^d$ to their corresponding spherical harmonic expansions in $\mathbb{S}^{d-1}$. This allows us to state our main theoretical result.

Theorem 7. *Let $\mathcal{H}^{\text{Lap}}$ denote the RKHS for the Laplace kernel restricted to $\mathbb{S}^{d-1}$, and let $\mathcal{H}^{\text{FC}_\beta(\text{L})}$ denote the NTK corresponding respectively to a FC network with L layers, ReLU activation, and bias, restricted to $\mathbb{S}^{d-1}$, then $\mathcal{H}^{\text{Lap}} = \mathcal{H}^{\text{FC}_\beta(2)} \subseteq \mathcal{H}^{\text{FC}_\beta(\text{L})}$.*

The common decay rates of NTK and the Laplace kernel in $\mathbb{S}^{d-1}$ imply that the set of functions in their RKHSs are identical, having the same smoothness properties. In particular, due to the norm equivalence of RKHSs and Sobolev spaces both spaces include functions that have weak derivatives up to order $d/2$ [154]. We recall that empirical results suggest further that $\boldsymbol{k}^{\text{FC}_\beta(\text{L})}$ decays exactly as $\Theta(k^{-d})$, and so we conjecture that $\mathcal{H}^{\text{Lap}} = \mathcal{H}^{\text{FC}_\beta(\text{L})}$. We note that despite this asymptotic similarity, the eigenvalues of NTK and the Laplace kernel are not identical even if we correct for shift and scale. Consequently, each kernel may behave slightly differently. Our experiments in Section 5.4 suggest that this results in only small differences in performance.

The similarity between NTK and the Laplace kernel has several implications. First, the dynamics of gradient descent for solving regression (5.2) with both kernels [158] should be similar. For a kernel with eigenvalues $\{\lambda_i\}_{i=1}^{\infty}$ a standard calculation shows that GD requires $O(1/\lambda_i)$ time steps to learn the ith eigenfunction (e.g., [131, 159]). For both NTK and the Laplace kernel in $\mathbb{S}^{d-1}$ this implies that $O(k^d)$ time steps are needed to learn a harmonic of frequency k. This is in contrast for instance with the Gaussian kernel, where the time needed to learn a harmonic of frequency k grows exponentially with k. This in particular explains the empirical results of [126], where it was shown that fitting noisy class labels with the Laplace kernel or neural networks requires a similar number of SGD steps. The authors of [126] conjectured that "optimization performance is controlled by the type of non-smoothness," as indeed is determined by the identical RKHS for NTK and the Laplace kernel.

The similarity between NTK and the Laplace kernel also implies that they have similar generalization properties. Indeed various generalization bounds rely explicitly on spectral properties of kernels. For example, given a set of training points $X \subseteq \mathbb{S}^{d-1}$ and a target function $f : \mathbb{S}^{d-1} \to \mathbb{R}$, then the error achieved by the kernel regression estimator given X, denoted $\hat{f}_X$, is (see, e.g., [160])

$$\left\| f - \hat{f}_X \right\|_\infty \leq C \cdot h(X)^\alpha \left\| f \right\|_{\mathcal{H}_k}, \;\; f \in \mathcal{H}_k,$$

where $h(X) := \sup_{\mathbf{z} \in \mathbb{S}^{d-1}} \inf_{\mathbf{x} \in X} \arccos(\mathbf{z}^T \mathbf{x})$ is the mesh norm of X (and thus depends on the density of the points), and α depends on the smoothness property of the kernel. Specifically, for both the Laplace kernel and NTK $\alpha = 1/2$.

Likewise, with n training points and $f \in \mathcal{H}_{\boldsymbol{k}}$, [161] derived the following lower bound

$$\mathbb{E}_X \left((f - \hat{f}_X)^2 \right) \geq \sum_{i=n+1}^{\infty} \alpha_i,$$

where α_i are the eigenvalues of f. Both of these bounds are equivalent asymptotically up to a constant for NTK (with bias) and the Laplace kernel.

5.3.4 NTK vs. exponential kernels in $\mathbb{R}^d$

While theoretical discussions of NTK largely assume the input data is normalized to lie on the sphere, such normalization is not the common practice in neural network applications. Instead, most often each feature is normalized separately by setting its mean to zero and variance to 1. Other normalizations are also common. It is therefore important to examine how NTK behaves outside of the hypersphere, compared to common kernels.

Below we derive the eigenfunctions of NTK for deep FC networks with ReLU activation with and without bias. We note that [142, 147] derived the eigenfunctions of NTK for two-layer FC networks with no bias. We will show that the same eigenfunctions are obtained with deep, bias-free networks, and that additional eigenfunctions appear when bias is added. We begin with a definition.

Definition 1. A kernel $\boldsymbol{k}$ is homogeneous of order α if $\boldsymbol{k}(\mathbf{x}, \mathbf{z}) = \|\mathbf{x}\|^\alpha \|\mathbf{z}\|^\alpha \boldsymbol{k}\left(\frac{\mathbf{x}^T\mathbf{z}}{\|\mathbf{x}\|\|\mathbf{z}\|}\right)$.

Theorem 8. *(1) Bias-free $\boldsymbol{k}^{\mathrm{FC}_0(\mathrm{L})}$ is homogeneous of order 1. (2) With bias initialized at zero, let $\boldsymbol{k}^{\mathrm{Bias(L)}} = \boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})} - \boldsymbol{k}^{\mathrm{FC}_0(\mathrm{L})}$. Then, $\boldsymbol{k}^{\mathrm{Bias(L)}}$ is homogeneous of order 0.*

The two kernels $\boldsymbol{k}^{\mathrm{FC}_0(\mathrm{L})}$ and $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$ (but not $\boldsymbol{k}^{\mathrm{Bias(L)}}$) are unbounded. Therefore, their Mercer's representation (5.1) exists under measures that decay sufficiently fast as $\|\mathbf{x}\| \to \infty$. Examples include the uniform distribution on the $\|\mathbf{x}\| \leq 1$ disk or the standard normal distribution. Such distributions have the virtue of being uniform on all concentric spheres. The following theorem determines the eigenfunctions for these kernels.

Theorem 9. *Let $p(r)$ be a decaying density on $[0, \infty)$ such that $0 < \int_0^\infty p(r) r^2 dr < \infty$ and $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$.*

1. *Let $\boldsymbol{k}_0(\mathbf{x}, \mathbf{z})$ be homogeneous of order 1 such that $\boldsymbol{k}_0(\mathbf{x}, \mathbf{z}) = \|\mathbf{x}\|\,\|\mathbf{z}\|\,\hat{\boldsymbol{k}}_0(\frac{\mathbf{x}^T\mathbf{z}}{\|\mathbf{x}\|\|\mathbf{z}\|})$. Then its eigenfunctions with respect to $p(\|\mathbf{x}\|)$ are given by $\Psi_{k,j} = a\|\mathbf{x}\| Y_{k,j}\left(\frac{\mathbf{x}}{\|\mathbf{x}\|}\right)$ where $Y_{k,j}$ are the spherical harmonics in $\mathbb{S}^{d-1}$ and $a \in \mathbb{R}$.*

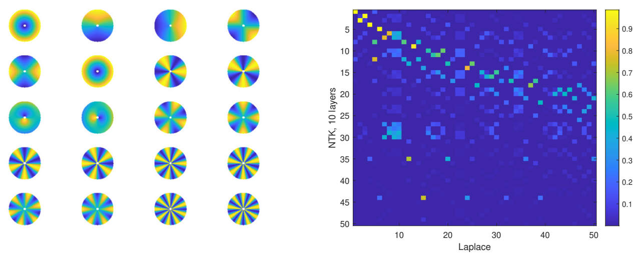


Figure 5.2: Left: plots of the eigenfunctions of NTK for a two layer FC network with bias on the unit disk, arranged in decreasing order of the eigenvalues. The radial shape of the eigenfunctions is evident. For two layers, the eigenvalues of $\boldsymbol{k}^{\mathrm{FC}_0(2)}$ are zero for odd $k \geq 3$, while those of $\boldsymbol{k}^{\mathrm{Bias}(2)}$ are zero for even $k \geq 2$. Therefore we see two “DC components" (top left and 2nd in 2nd row) and four $k = 1$ components (2nd and 3rd in 1st row and 1st and 2nd in third row). The rest of the frequencies are represented twice each. Right: Absolute correlation between the eigenfunctions of NTK and those of the Laplace kernel for data sampled uniformly on the unit disk. It can be seen that eigenfunctions of higher frequency for NTK correlate with eigenfunctions of higher frequency for the Laplace. However, relatively low order components for NTK contain higher frequency components of the Laplace.

2. *Let $\boldsymbol{k}(\mathbf{x},\mathbf{z}) = \boldsymbol{k}_0(\mathbf{x},\mathbf{z}) + \boldsymbol{k}_1(\mathbf{x},\mathbf{z})$ so that $\boldsymbol{k}_0$ as in 1 and $\boldsymbol{k}_1$ is homogeneous of order 0. Then the eigenfunctions of $\boldsymbol{k}$ are of the form $\Psi_{k,j} = (a\,\|\mathbf{x}\| + b)\,Y_{k,j}\left(\frac{\mathbf{x}}{\|\mathbf{x}\|}\right)$.*

The eigenfunctions of NTK in $\mathbb{R}^d$, therefore, are similar to those in $\mathbb{S}^{d-1}$; they are the spherical harmonics scaled radially in the bias free case, or linearly with the norm when bias is used. With bias, $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$ has up to $2N(d,k)$ eigenfunctions for every frequency k. Compared to the eigenvalues in $\mathbb{S}^{d-1}$, the eigenvalues can change, depending on the radial density $p(r)$, but they maintain their overall asymptotic behavior.

In contrast to NTK, the Laplace kernel is shift invariant, and therefore its eigenfunctions are the Fourier transform. The two kernels hence cannot be compared merely by their eigenvalues. Figure 5.2 shows the eigenfunctions of NTK along with their correlation to the eigenfunctions of the Laplace kernel. While these differences are large, they seem to make only little difference in experiments, see Section 5.4 below. It is possible to produce a homogeneous version of the Laplace kernel as follows

$$\boldsymbol{k}^{\mathrm{HLap}}(\mathbf{x},\mathbf{z}) = \|\mathbf{x}\|\|\mathbf{z}\| \exp\left(-c\sqrt{1-\frac{\mathbf{x}^T\mathbf{z}}{\|\mathbf{x}\|\|\mathbf{z}\|}}\right). \tag{5.5}$$

Following Thm. 9 the eigenfunctions of this kernel are the scaled spherical harmonics and, following Thm. 6, its eigenvalues decay at the rate of k^{-d}, much like the NTK.

5.4 Experiments

We compare the performance of NTK with Laplace, Gaussian, and γ-exponential kernels on both small and large scale real datasets. Our goal is to demonstrate: a) Results with the Laplace kernel are quite similar to those obtained by NTK, and b) The γ-exponential kernel can achieve slightly better results than NTK. Experimental details are provided in the Appendix D.5.

5.4.1 UCI dataSet

We compare methods using the same set of 90 small scale UCI datasets (with less than 5000 data points) as in [129]. The results are provided in Table 5.3 for the exponential kernels and their homogeneous versions, denoted by the "H-" prefix, as well as for NTK. For completeness, we also cite the results for Random forest (RF), the top classifier identified in [162], and neural networks from [129]. Further comparison of the accuracies obtained with NTK vs. the H-Laplace kernel on each of the 90 datasets is shown in Figure 5.4.

We report the same metrics as used in [129]: Friedman Ranking, Average Accuracy, P90/P95, and PMA. A superior classifier is expected to have lower Friedman rank and higher P90, P95, and PMA. Friedman Ranking [163] reports the average ranking of a given classifier compared to other classifiers. P90/P95 denotes the fraction of datasets on which a classifier achieves more than

$90/95\%$ of the maximum achievable accuracy (i.e., maximum accuracy among all the classifiers [162]). PMA represents the percentage of maximum accuracy.

From Table 5.3, one can observe that the H-Laplace kernel results are the closest to NTK on all the metrics. In fact, as seen in Figure 5.4, these methods seem to achieve highly similar accuracies in each of the 90 datasets. Furthermore, the H-γ-exponential outperforms all the classifiers including NTK on all metrics. Moreover, the homogeneous versions slightly outperform the standard kernels. All these methods have hyperparameters that can be optimized. In [129], they search 105 hyperparameters for NTK. For a fair comparison, we search for the same number for the γ-exponential and fewer (70) for the Laplace kernels. We note finally that deeper networks yield NTK shapes that are more sharply peaked, corresponding to Laplace kernels with higher values of c. This is shown in Fig. 5.5 below.

Classifier	F-Rank	Average Accuracy	P90	P95	PMA
H-γ-exp.	**26.26**	**82.25%±14.07%**	**92.22%**	**73.33%**	**96.07% ±4.83%**
γ-exp.	32.98	81.80%±14.21%	85.56%	73.33%	95.49% ±5.31%
H-Laplace	29.60	81.74%±13.82%	88.89%	66.67%	95.53% ±4.84%
Laplace	33.28	81.12%±14.16%	86.67%	65.56%	94.88% ±6.85%
H-Gaussian	32.66	81.46% ± 14.83%	84.44%	67.77%	94.95% ±6.25%
Gaussian	35.76	81.03% ± 15.09%	85.56%	72.22%	94.56% ±8.22%
NTK [129]	28.34	81.95%±14.10%	88.89%	72.22%	95.72% ±5.17%
NN [129]	38.06	81.02%±14.47%	85.56%	60.00%	94.55% ±5.89%
RF [129]	33.51	81.56% ±13.90%	85.56%	67.78%	95.25% ±5.30%

Figure 5.3: Performance on the UCI dataset. Lower F-Rank and higher P90, P95, PMA are better numbers.

5.4.2 Large scale datasets

We leverage FALKON [164], an efficient approximate kernel method to conduct large scale regression and classification tasks following the setup of [164]. The results and datasets details are reported in Table 5.1. We searched for hyperparameters based on a small validation dataset for all the methods

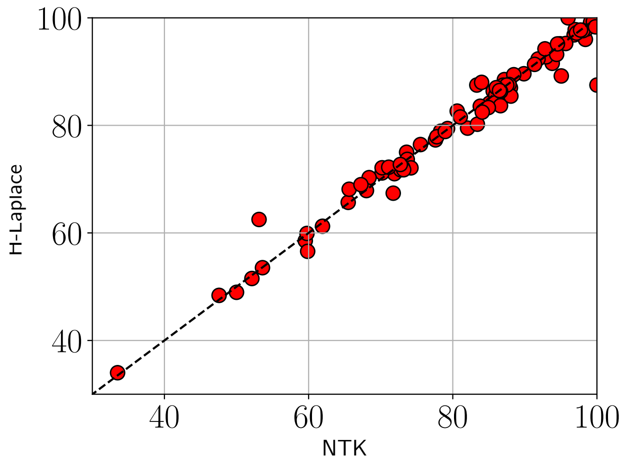


Figure 5.4: Performance comparisons between NTK and H-Laplace on the UCI dataset.

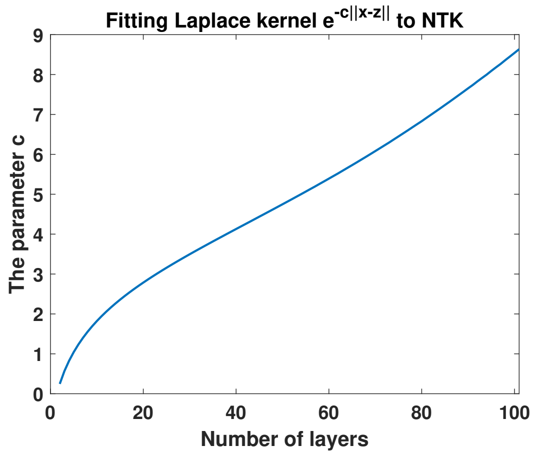


Figure 5.5: Fitting the Laplace kernel to NTK. The graph shows optimal width (c) of the Laplace kernel that is fitted to NTK with different number of layers.

and used the standard train/test partition provided on the UCI repository. From Table 5.1, one can notice that NTK and H-Laplace perform similarly. For each dataset, either the γ-exponential or Gaussian kernels slightly outperforms these two kernels.

5.4.3 Hierarchical convolutional kernels

Convolutional NTKs (CNTK) were shown to express the limit of convolutional neural networks when the number of channels tends to infinity, and recent empirical results showed that the two

	MillionSongs [165]	SUSY [166]	HIGGS [166]
#Training Data	4.6×10^5	5×10^6	1.1×10^7
#Features	90	18	28
Problem Type	Regression	Classification	Classification
Performance Metric	MSE	AUC	AUC
H-γ-exp.	**78.6417**	87.686	**82.281**
H-Laplace	79.7941	87.670	81.995
NTK	79.9666	87.673	82.089
H-Gaussian	79.6255	**87.689**	81.967
Neural Network [166]	-	87.500	81.600
Deep Neural Network [166]	-	**87.900***	**88.500***

Table 5.1: Performance on the large scale datasets. We report MSE (lower is better) for the regression problem, and AUC (higher is better) for the classification problems.

achieve similar accuracy on test data [124, 141]. CNTK is defined roughly by recursively applying NTK to image patches. For our final experiment we constructed alternative hierarchical kernels, in the spirit of [167, 168], by recursively applying exponential kernels in a manner similar to CNTK. The new kernel, denoted C-Exp, is applied first to pairs of 3×3 image patches, then to 3×3 patches of kernel values, and so forth. A detailed algorithm is provided in the Appendix D.5.3. We applied the kernel (using the homogeneous versions of the Laplace, Gaussian and γ-exponential kernels) to the Cifar-10 dataset and compared it to CNTK. Our experimental conditions and results for CNTK are identical to those of [124]. (Note that these do not include global average pooling.) Consistent with our previous experiments, Table 5.2 shows that these kernels are on par with the CNTK with small advantage to the γ-exponential kernel. This demonstrates that the four kernels maintain similar performance even after repeated application.

Method	Accuracy (50k)	Accuracy(2k)
CNTK	66.4%	43.9%
C-Exp Laplace	65.2%	44.2%
C-Exp γ-exponential	**67.0%**	**45.2%**
C-Exp Gaussian	66.8%	45.0%

Table 5.2: Classification accuracy for the CIFAR-10 dataset for our C-Exp hierarchical kernels, compared to CNTK. The two columns show results with training on the full dataset and on the first 2000 examples.

5.5 Summary

In this chapter, we considered the relationship between NTK and the classic Laplace kernel. Our main result is to show that for data normalized on the unit hypersphere, these two kernels have the same RKHS. Experiments show that the two kernels perform almost identically on a wide range of real-world applications. This enables one to replace expensive NTK with the standard Laplace kernel, which is computationally much cheaper than NTK. Moreover, coupled with prior results that show that kernel methods using NTK mimic the behavior of FC neural networks, our results suggest that much insight about neural networks can be obtained from analysis of the well-known Laplace kernel, which has a simple closed-form.

Chapter 6: Conclusion

In this dissertation, we have improved training of three types of deep network models from efficiency, automation and stability point of views, with the main focus on important computer vision applications:

- Big batch SGD (first-order) and Frozen-Batch L-BFGS (second-order) methods to efficiently train deep networks with almost no parameter turning and little or no user oversight.
  - Big batch SGD schemes adaptively grow the batch size to maintain a nearly constant signal-to-noise ratio in gradient approximation with the intuition that the increased stochasticity can help land the iterates near a good minimizer in the initial iterations. Later on, larger batches can increase the speed of convergence towards this minimizer. The resulting high-fidelity gradients also enable automated learning rate selection, parallel training, and reliable stopping creation. We show comparable results with carefully manually tuned SGD and perform consistently better than other standard methods, but without requiring an expert user to choose learning rates and decay parameters.
  - The frozen batch scheme has enabled L-BFGS to work reasonably well on more practical well-known deep networks that use BatchNorm and achieved comparable results with SOTA methods, which has further bolstered confidence that second-order method such as L-BFGS can also be considered for image classification tasks (which is not even considered as an option to explore for those applications). We achieved consistently

better results than existing L-BFGS methods and comparable results (in most cases) with carefully tuned SGD. This work marks another step toward making L-BFGS an algorithm of choice for large-scale stochastic machine learning (just like L-BFGS is a default choice for convex optimization).

- The prediction step method stabilizes training of adversarial deep networks and prevents network mode collapse. We have shown results on various test problems and demonstrated the method's stability across a wide range of learning rates. Consequently, one can train with a faster learning rate for faster convergence, and another advantage is that the method leads to a simple modification that can be easily integrated with other sophisticated methods.
- At last, we propose to efficiently compute NTK by establishing that for most practical use-cases (ReLU activation and data distributed on a sphere), NTK can be replaced by the well-known Laplace kernel, which is computationally way cheaper than NTK. Prior results have shown that NTK mimics the behavior of real infinite width neural networks, so another takeaway from this work is that one can get more insight into neural networks from the analysis of the Laplace kernel, which has a simple closed-form.

Future directions: Though our proposed methods achieve significant progress, there is still scope for improvement, and we list a few possible research directions below:

- Although we are able to achieve good performance with large batches, the generalization performance is still not better than SGD. Many recent works have tried to understand the mysterious properties that make SGD favorable for generalization, but still, it remains inconclusive. More recent work has shown that some explicit form of regularization can be used with large batch to improve the large batch generalization. It would be a direction worth exploring to combine the proposed methods (Big batch SGD and FbLBFGS) with those explicit regularizations.

- GANs are one of the most important deep networks in computer vision, and the spectral normalization [169] is a state-of-art regularization technique to achieve good results in GANs. So, it would be worthwhile to combine the prediction method with spectral normalization, specifically when it does not incur any significant implementation or computation burden. Also, to achieve better performance, one would expect to remove the regularization as the model converges towards the solution, and the prediction method can play a significant role in that regime due to its asymptotically stable performance for solving saddle point problems.

Appendix A: Automated Inference using Adaptive Batch Sizes

A.1 Proof of Lemma 1

Proof. We know that $-\nabla \ell_{\mathcal{B}}(x)$ is a descent direction iff the following condition holds:

$$\nabla \ell_{\mathcal{B}}(x)^T \nabla \ell(x) > 0. \tag{A.1}$$

Expanding $\|\nabla \ell_{\mathcal{B}}(x) - \nabla \ell(x)\|^2$ we get

$$\|\nabla \ell_{\mathcal{B}}(x)\|^2 + \|\nabla \ell(x)\|^2 - 2\nabla \ell_{\mathcal{B}}(x)^T \nabla \ell(x) < \|\nabla \ell_{\mathcal{B}}(x)\|^2,$$
$$\implies -2\nabla \ell_{\mathcal{B}}(x)^T \nabla \ell(x) < -\|\nabla \ell(x)\|_2^2 \leq 0,$$

which is always true for a descent direction (A.1). □

A.2 Proof of Theorem 1

Proof. Let $\bar{z} = \mathbb{E}[z]$ be the mean of z. Given the current iterate x, we assume that the batch $\mathcal{B}$ is sampled uniformly with replacement from p. We then have the following bound:

$$\begin{aligned}
\|\nabla f(x;z) - \nabla \ell(x)\|^2 &\leq 2\|\nabla f(x;z) - \nabla f(x,\bar{z})\|^2 + 2\|\nabla f(x,\bar{z}) - \nabla \ell(x)\|^2 \\
&\leq 2L_z^2\|z-\bar{z}\|^2 + 2\|\nabla f(x,\bar{z}) - \nabla \ell(x)\|^2 \\
&= 2L_z^2\|z-\bar{z}\|^2 + 2\|\mathbb{E}_z[\nabla f(x,\bar{z}) - \nabla f(x,z)]\|^2 \\
&\leq 2L_z^2\|z-\bar{z}\|^2 + 2\mathbb{E}_z\|\nabla f(x,\bar{z}) - \nabla f(x,z)\|^2 \\
&\leq 2L_z^2\|z-\bar{z}\|^2 + 2L_z^2\mathbb{E}_z\|\bar{z}-z\|^2 \\
&= 2L_z^2\|z-\bar{z}\|^2 + 2L_z^2 \operatorname{Tr}\operatorname{Var}_z(z),
\end{aligned}$$

where the first inequality uses the property $\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2$, the second and fourth inequalities use Assumption 1, and the third line uses Jensen's inequality. This bound is *uniform* in x. We then have

$$\begin{aligned}
\mathbb{E}_z\|\nabla f(x;z) - \nabla \ell(x)\|^2 &\leq 2L_z^2\mathbb{E}_z\|z-\bar{z}\|^2 + 2L_z^2 \operatorname{Tr}\operatorname{Var}_z(z) \\
&= 4L_z^2 \operatorname{Tr}\operatorname{Var}_z(z)
\end{aligned}$$

uniformly for all x. The result follows from the observation that

$$\mathbb{E}_{\mathcal{B}}\|\nabla f_{\mathcal{B}}(x) - \nabla \ell(x)\|^2 = \frac{1}{|\mathcal{B}|}E_z\|\nabla f(x;z) - \nabla \ell(x)\|^2.$$

□

A.3 Proof of Lemma 2

Proof. From (2.5) and Assumption 2 we get

$$\ell(x_{t+1}) \leq \ell(x_t) - \alpha {g_t}^T \nabla \ell(x_t) + \frac{L\alpha^2}{2}\|g_t\|^2.$$

Taking expectation with respect to the batch $\mathcal{B}_t$ and conditioning on x_t, we get

$$\begin{aligned}
\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq& \ell(x_t) - \ell(x^\star) - \alpha \mathbb{E}[g_t]^T \nabla \ell(x_t) + \frac{L\alpha^2}{2}\mathbb{E}\|g_t\|^2 \\
=& \ell(x_t) - \ell(x^\star) - \alpha\|\nabla \ell(x_t)\|^2 + \frac{L\alpha^2}{2}(\|\nabla \ell(x_t)\|^2 + \mathbb{E}\|e_t\|^2 + \mathbb{E}[e_t]^T \nabla \ell(x_t)) \\
=& \ell(x_t) - \ell(x^\star) - \big(\alpha - \frac{L\alpha^2}{2}\big)\|\nabla \ell(x_t)\|^2 + \frac{L\alpha^2}{2}\mathbb{E}\|e_t\|^2 \\
\leq& \Big(1 - 2\mu\big(\alpha - \frac{L\alpha^2}{2}\big)\Big)(\ell(x_t) - \ell(x^\star)) + \frac{L\alpha^2}{2}\mathbb{E}\|e_t\|^2,
\end{aligned}$$

where the second inequality follows from Assumption 3. Taking expectation, the result follows.

□

A.4 Proof of Theorem 2

Proof. We begin by applying the reverse triangle inequality to (2.4) to get

$$(1-\theta)\mathbb{E}\|\nabla \ell_{\mathcal{B}}(x)\| \leq \mathbb{E}\|\nabla \ell(x)\|$$

which applied to (2.4) yields

$$\frac{\theta^2}{(1-\theta)^2}\mathbb{E}\|\nabla\ell(x_t)\|^2 \geq \mathbb{E}\|\nabla\ell_{\mathcal{B}}(x_t) - \nabla\ell(x_t)\|^2 = \mathbb{E}\|e_t\|^2. \tag{A.2}$$

Now, we apply (A.2) to the result in Lemma 2 to get

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \mathbb{E}[\ell(x_t) - \ell(x^\star)] - \big(\alpha - \frac{L\alpha^2\beta}{2}\big)\mathbb{E}\|\nabla\ell(x_t)\|^2,$$

where $\beta = \frac{\theta^2+(1-\theta)^2}{(1-\theta)^2} \geq 1$. Assuming $\alpha - \frac{L\alpha^2\beta}{2} \geq 0$, we can apply Assumption 3 to write

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \Big(1 - 2\mu\big(\alpha - \frac{L\alpha^2\beta}{2}\big)\Big)\mathbb{E}[\ell(x_t) - \ell(x^\star)],$$

which proves the theorem. Note that $\max_\alpha\{\alpha - \frac{L\alpha^2\beta}{2}\} = \frac{1}{2L\beta}$, and $\mu \leq L$. It follows that

$$0 \leq \Big(1 - 2\mu\big(\alpha - \frac{L\alpha^2\beta}{2}\big)\Big) < 1.$$

The second result follows immediately. □

A.5 Proof of Theorem 3

Proof. Applying the reverse triangle inequality to (2.4) and using Lemma 2 we get, as in Theorem 2:

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \mathbb{E}[\ell(x_t) - \ell(x^\star)] - \big(\alpha - \frac{L\alpha^2\beta}{2}\big)\mathbb{E}\|\nabla\ell(x_t)\|^2, \tag{A.3}$$

where $\beta = \frac{\theta^2+(1-\theta)^2}{(1-\theta)^2} \geq 1$.

We will show that the backtracking condition in (2.7) is satisfied whenever $0 < \alpha_t \leq \frac{1}{\beta L}$. First notice that:

$$0 < \alpha_t \leq \frac{1}{\beta L} \quad \implies \quad -\alpha_t + \frac{L\alpha_t^2\beta}{2} \leq -\frac{\alpha_t}{2}.$$

Thus, we can rewrite (A.3) as

$$\begin{aligned}\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] &\leq \mathbb{E}[\ell(x_t) - \ell(x^\star)] - \frac{\alpha_t}{2}\mathbb{E}\|\nabla\ell(x_t)\|^2 \\ &\leq \mathbb{E}[\ell(x_t) - \ell(x^\star)] - c\alpha_t\mathbb{E}\|\nabla\ell(x_t)\|^2,\end{aligned}$$

where $0 < c \leq 0.5$. Thus, the backtracking line search condition (2.7) is satisfied whenever $0 < \alpha_t \leq \frac{1}{L\beta}$.

Now we know that either $\alpha_t = \alpha_0$ (the initial stepsize), or $\alpha_t \geq \frac{1}{2\beta L}$, where the stepsize is decreased by a factor of 2 each time the backtracking condition fails. Thus, we can rewrite the above as

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \mathbb{E}[\ell(x_t) - \ell(x^\star)] - c\min\left(\alpha_0, \frac{1}{2\beta L}\right)\mathbb{E}\|\nabla\ell(x_t)\|^2.$$

Using Assumption 3 we get

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \left(1 - 2c\mu\min\left(\alpha_0, \frac{1}{2\beta L}\right)\right)\mathbb{E}[\ell(x_t) - \ell(x^\star)].$$

Assuming we start off the stepsize at a large value such that $\min(\alpha_0, \frac{1}{2\beta L}) = \frac{1}{2\beta L}$, we can rewrite this as:

$$\mathbb{E}[\ell(x_{t+1}) - \ell(x^\star)] \leq \Big(1 - \frac{c\mu}{\beta L}\Big)\mathbb{E}[\ell(x_t) - \ell(x^\star)].$$

□

A.6 Algorithmic Details for Automated Big Batch Methods

The complete details of the backtracking Armijo line search we used with big batch SGD are explained in detail in Algorithm 3.

Algorithm 3 Big batch SGD: backtracking line search

```
1: initialize starting pt. $x_0$, initial stepsize $\alpha$, initial batch size $K > 1$, batch size increment $\delta_k$,
   backtracking line search parameter $c$, flag $F = 0$
2: while not converged do
3:     Draw random batch with size $|\mathcal{B}| = K$
4:     Calculate $V_\mathcal{B}$ and $\nabla \ell_\mathcal{B}(x_t)$ using (2.6)
5:     while $\|\nabla \ell_\mathcal{B}(x_t)\|^2 \leq V_\mathcal{B}/K$ do
6:         Increase batch size $K \leftarrow K + \delta_K$
7:         Sample more gradients
8:         Update $V_\mathcal{B}$ and $\nabla \ell_\mathcal{B}(x_t)$
9:         Set flag $F = 1$
10:    end while
11:    if flag $F == 1$ then
12:        $\alpha \leftarrow \alpha * 2$
13:        Reset flag $F = 0$
14:    end if
15:    while $\ell_\mathcal{B}(x_t - \alpha \nabla \ell_\mathcal{B}(x_t)) > \ell_\mathcal{B}(x_t) - c\alpha_t \|\nabla \ell_\mathcal{B}(x_t)\|^2$ do
16:        $\alpha \leftarrow \alpha/2$
17:    end while
18:    $x_{t+1} = x_t - \alpha \nabla \ell_\mathcal{B}(x_t)$
19: end while
```

A.7 Details of Neural Network Experiments and Additional Results

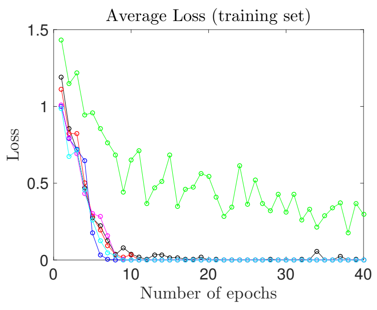
Here we present details of the ConvNet and exact hyper-parameters used for training the neural network models for our experiments.

We train a convolutional neural network [48] (ConvNet) to classify three benchmark image datasets: CIFAR-10 [49], SVHN [50] and MNIST [48]. The ConvNet used in our experiments is composed of 4 layers, excluding the input layer. We use 32×32 pixel images as input. The first layer of the ConvNet contains $16 \times 3 \times 3$, and the second layer contains $256 \times 3 \times 3$ filters. The third and fourth layers are fully connected [48] with 256 and 10 outputs respectively. Each layer except the last one is followed by a ReLu non-linearity [2] and a max pooling stage [170] of size 2×2. This ConvNet has over 4.3 million weights.

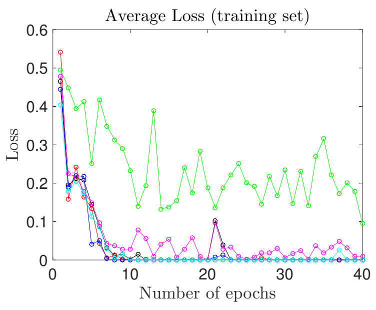
To compare against fine-tuned SGD, we used a comprehensive grid search on the stepsize schedule to identify optimal parameters (up to a factor of 2 accuracy). For CIFAR10, the stepsize starts from 0.5 and is divided by 2 every 5 epochs with 0 stepsize decay. For SVHN, the stepsize starts from 0.5 and is divided by 2 every 5 epochs with $1e{-}05$ learning rate decay. For MNIST, the learning rate starts from 1 and is divided by 2 every 3 epochs with 0 stepsize decay. All algorithms use a momentum parameter of 0.9, and SGD and AdaDelta use mini-batches of size 128.

Fixed stepsize methods use the default decay rule of the *Torch* library: $\alpha_t = \alpha_0/(1 + 10^{-7}t)$, where α_0 was chosen to be the stepsize used in the fine-tuned experiments. We also tune the hyper-parameter ρ in the Adadelta algorithm, and we found 0.9, 0.9 and 0.8 to be best-performing parameters for CIFAR10, SVHN and MNIST respectively.

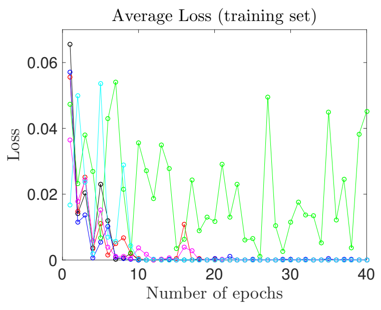
Figure A.1 shows the change in the loss function over time for the same neural network experiments shown in the Chapter 2 (on CIFAR-10, SVHN and MNIST).



(a)



(b)



(c)

Figure A.1: Neural Network Experiments. Figure shows the change in the loss function for CIFAR-10, SVHN, and MNIST (left to right).

Appendix B: Making L-BFGS Work with Industrial-Strength Nets

B.1 Performance overview of FbLBFGS for different datasets and deep networks

B.1.1 Overview of FbLBFGS for STL10 on different deep networks

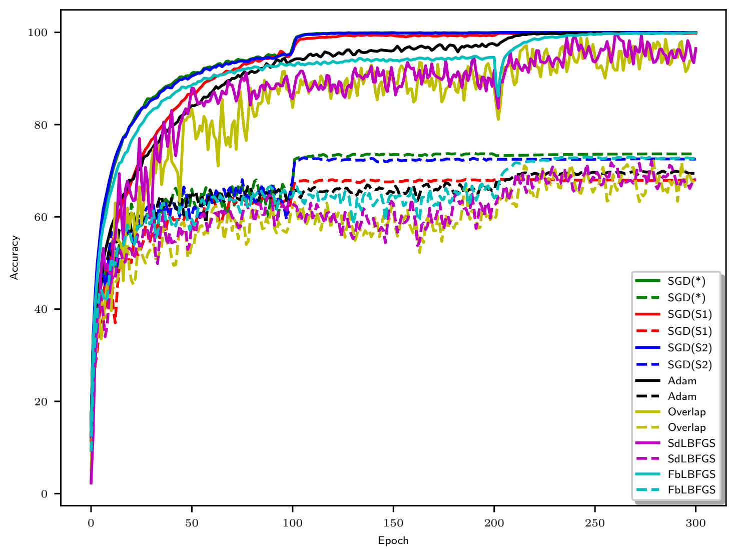


Figure B.1: Overview of the performance of STL-10 on ResNet. The solid lines represent train accuracy and dashed lines represent the test accuracy, respectively.

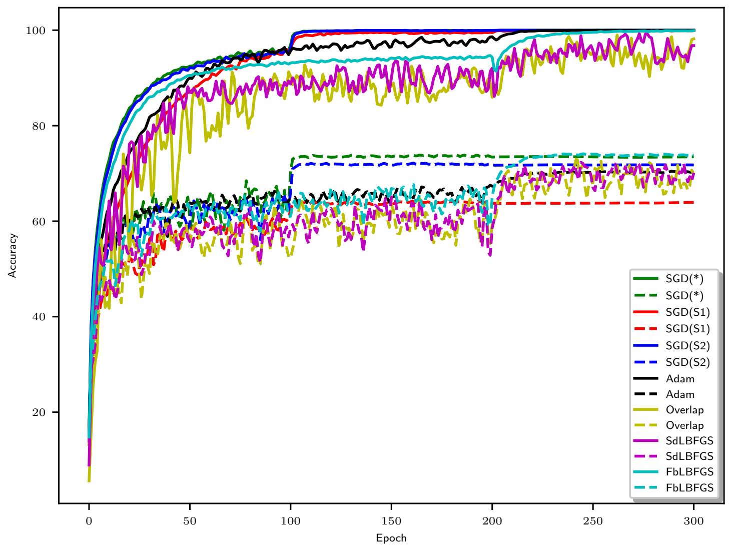


Figure B.2: Overview of the performance of STL-10 on DenseNet. The solid lines represent train accuracy and dashed lines represent the test accuracy, respectively.

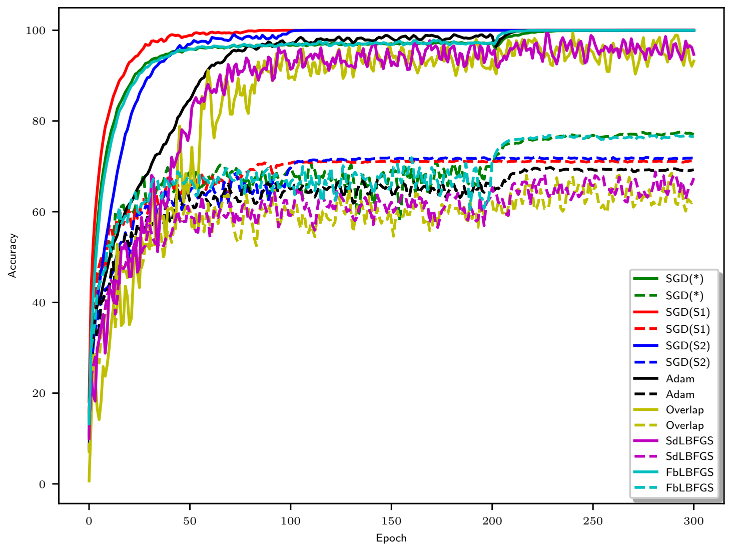


Figure B.3: Overview of the performance of STL-10 on Wide ResNet. The solid lines represent train accuracy and dashed lines represent the test accuracy, respectively.

B.1.2 Overview of FbLBFGS for CIFAR-10 on different deep networks

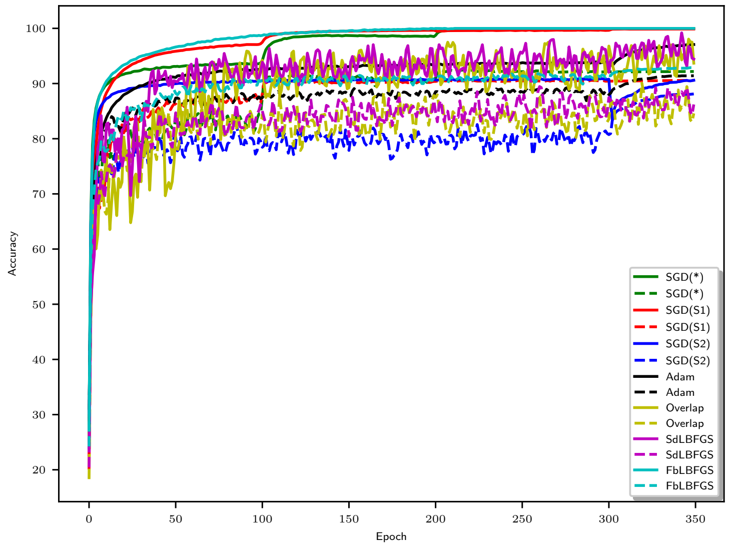


Figure B.4: Overview of the performance of CIFAR-10 on ResNet. The solid lines represent train accuracy and dashed lines represent the test accuracy, respectively.

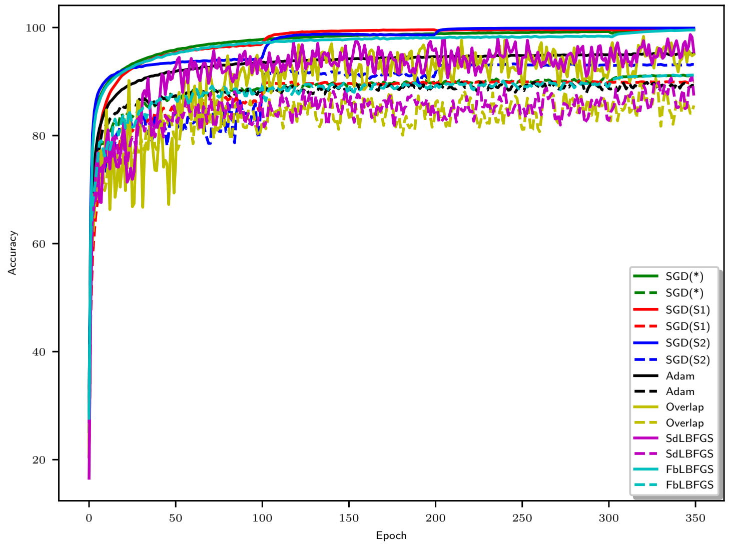


Figure B.5: Overview of the performance of CIFAR-10 on DenseNet. The solid lines represent train accuracy and dashed lines represent the test accuracy, respectively.

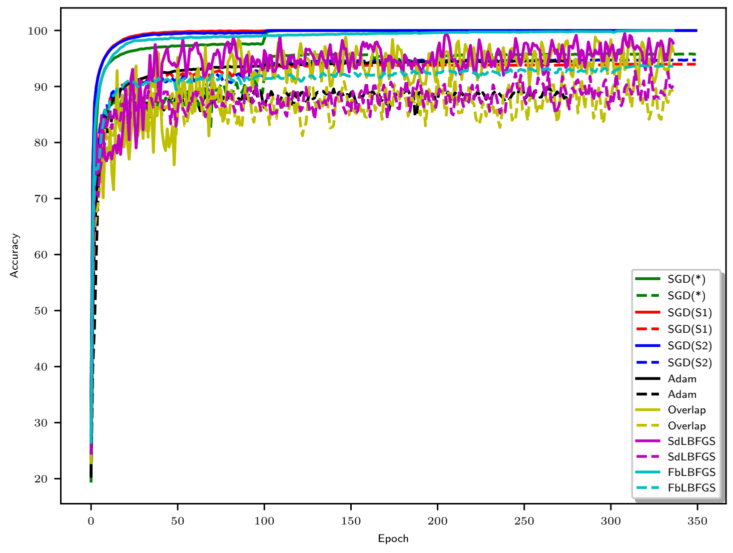


Figure B.6: Overview of the performance of CIFAR-10 on Wide ResNet. The solid lines represent train accuracy and dashed lines represent the test accuracy, respectively.

B.1.3 Overview of FbLBFGS for CIFAR-100 on different deep networks

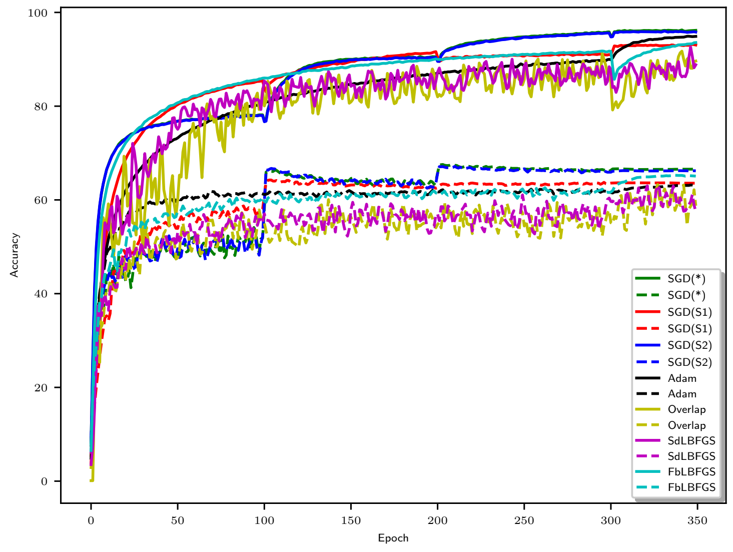


Figure B.7: Overview of the performance of CIFAR-100 on ResNet. The solid lines represent train accuracy and dashed lines represent the test accuracy, respectively.

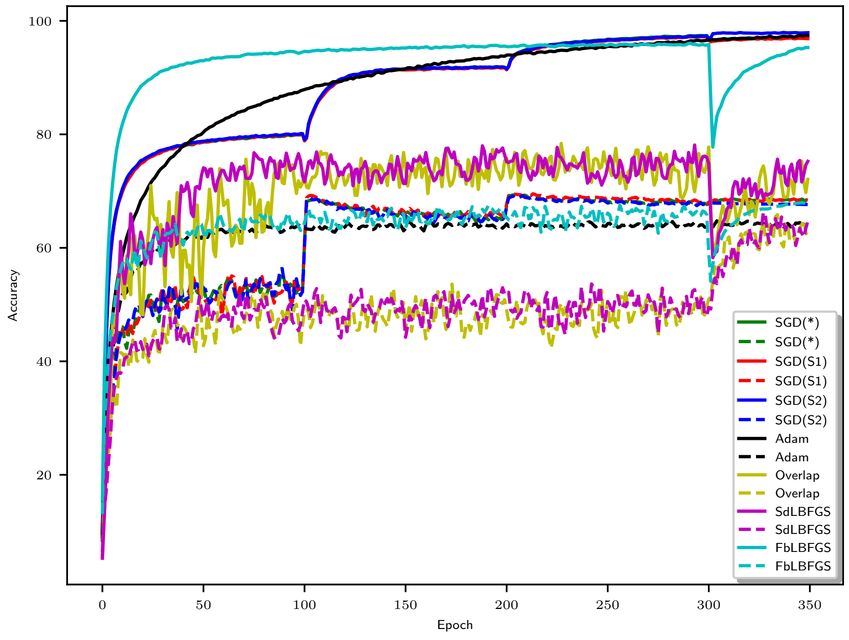


Figure B.8: Overview of the performance of CIFAR-100 on DenseNet. The solid lines represent train accuracy and dashed lines represent the test accuracy, respectively.

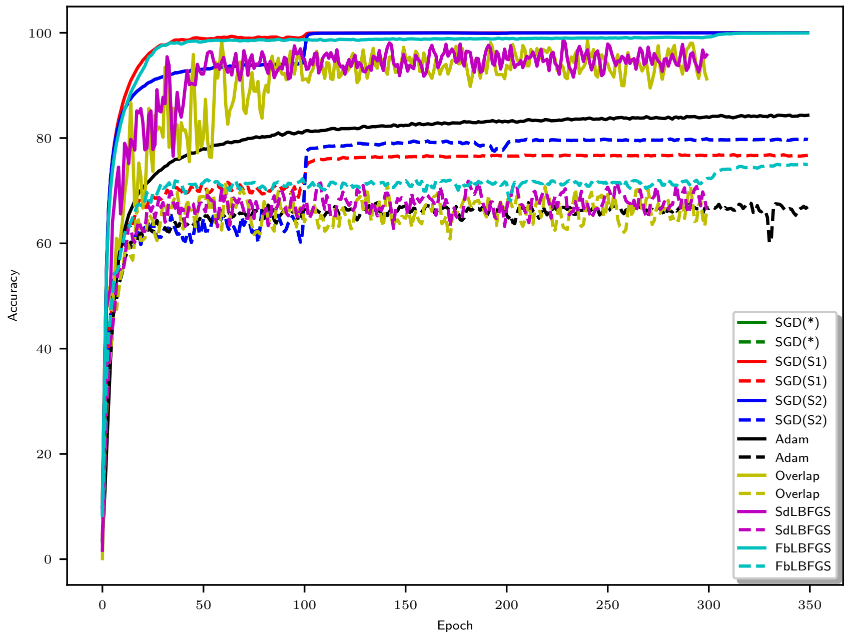


Figure B.9: Overview of the performance of CIFAR-100 on Wide ResNet. The solid lines represent train accuracy and dashed lines represent the test accuracy, respectively.

Appendix C: Stabilizing Adversarial Nets With Prediction Methods

C.1 Detailed derivation of the harmonic oscillator equation

Here, we provide a detailed derivation of the harmonic oscillator behavior of Algorithm (4.3) on the simple bi-linear saddle of the form

$$\mathcal{L}(x, y) = y^T K x$$

where K is a matrix. Note that, within a small neighborhood of a saddle, all smooth weakly convex objective functions behave like (4.6).To see why, consider a smooth objective function $\mathcal{L}$ with a saddle point at $x^* = 0$, $y^* = 0$. Within a small neighborhood of the saddle, we can approximate the function $\mathcal{L}$ to high accuracy using its Taylor approximation

$$\mathcal{L}(x, y) \approx \mathcal{L}(x^*, y^*) + y^T \mathcal{L}'_{xy} x + O(\|x\|^3 + \|y\|^3)$$

where $\mathcal{L}'_{xy}$ denotes the matrix of mixed-partial derivatives with respect to x and y. Note that the first-order terms have vanished from this Taylor approximation because the gradients are zero at a saddle point. The $O(\|x\|^2)$ and $O(\|y\|^2)$ terms vanish as well because the problem is assumed to be weakly convex around the saddle. Up to third-order error (which vanishes quickly near the saddle), this Taylor expansion has the form (4.6). For this reason, stability on saddles of the form (4.6) is a necessary condition for convergence of (4.3), and the analysis here describes the asymptotic behavior of the prediction method on any smooth problem for which the method converges.

We will show that, as the learning rate gets small, the iterates of the non-prediction method (4.2) rotate in orbits around the saddle without converging. In contrast, the iterates of the prediction method fall into the saddle and converge.

When the conventional gradient method (4.2) is applied to the linear problem (4.6), the resulting iterations can be written

$$\frac{x^{k+1}-x^k}{\alpha} = -K^T y^k, \qquad \frac{y^{k+1}-y^k}{\alpha} = (\beta/\alpha)Kx^{k+1}.$$

When the stepsize α gets small, this behaves like a discretization of the differential equation

$$\dot{x} = -K^T y \tag{C.1}$$

$$\dot{y} = \beta/\alpha K x \tag{C.2}$$

where $\dot{x}$ and $\dot{y}$ denote the derivatives of x and y with respect to time.

The differential equations (C.1,C.2) describe a harmonic oscillator. To see why, differentiate (C.1) and plug (C.2) into the result to get a differential equation in x alone

$$\ddot{x} = -K^T\dot{y} = -\beta/\alpha K^T K x. \tag{C.3}$$

We can decompose this into a system of independent single-variable problems by considering the eigenvalue decomposition $\beta/\alpha K^T K = U\Sigma U^T$. We now multiply both sides of (C.3) by U^T, and make the change of variables $z \leftarrow U^T x$ to get

$$\ddot{z} = -\Sigma z.$$

where Σ is diagonal. This is the standard equation for undamped harmonic motion, and its solution is $z = A\cos(\Sigma^{1/2}t + \phi)$, where $\cos$ acts entry-wise, and the diagonal matrix A and vector ϕ are constants that depend only on the initialization. Changing back into the variable x, we get the solution

$$x = UA\cos(\Sigma^{1/2}t + \phi).$$

We can see that, for small values of α and β, the non-predictive algorithm (4.2) approximates an undamped harmonic motion, and the solutions orbit around the saddle without converging.

The prediction step (4.3) improves convergence because it produces *damped* harmonic motion that sinks into the saddle point. When applied to the linearized problem (4.6), the iterates of the predictive method (4.3) satisfy

$$\begin{aligned}
\frac{x^{k+1} - x^k}{\alpha} &= -K^T y^k \\
\frac{y^{k+1} - y^k}{\alpha} &= \beta/\alpha K(x^{k+1} + x^{k+1} - x^k) = \beta/\alpha K x^{k+1} + \beta K \frac{x^{k+1} - x^k}{\alpha}.
\end{aligned}$$

For small α, this approximates the dynamical system

$$\dot{x} = -K^T y \tag{C.4}$$

$$\dot{y} = \beta/\alpha K(x + \alpha\dot{x}). \tag{C.5}$$

Like before, we differentiate (C.4) and use (C.5) to obtain

$$\ddot{x} = -K^T\dot{y} = -\beta/\alpha K^T(Kx + \alpha A\dot{x}) = -\beta/\alpha K^T Kx - \beta/K^T K\dot{x}. \tag{C.6}$$

Finally, multiply both sides by U^T and perform the change of variables $z \leftarrow U^T x$ to get

$$\ddot{z} = -\Sigma z - \alpha\Sigma\dot{z}.$$

This equation describes a damped harmonic motion. The solutions have the form of $z(t) = A\exp(-\frac{t\alpha}{2}\sqrt{\Sigma})\sin(t\sqrt{(1-\alpha^2/4)\Sigma}+\phi)$. Changing back to the variable x, we see that the iterates of the original method satisfy

$$x(t) = UA\exp(-\frac{t\alpha}{2}\sqrt{\Sigma})\sin(t\sqrt{(1-\alpha^2/4)\Sigma}+\phi).$$

where A and ϕ depend on the initialization.

From this analysis, we see that for small constant α the orbits of the lookahead method converge into the saddle point, and the error decays exponentially fast.

C.1.1 Proof of Theorem 4

Assume the optimal solution $(u^\star, v^\star)$ exists, then $\mathcal{L}'_u(u^\star, v) = \mathcal{L}'_v(u, v^\star) = 0$. In the following proofs, we use $g_u(u, v)$, $g_v(u, v)$ to represent the stochastic approximation of gradients, where $\mathbb{E}[g_u(u, v)] = \mathcal{L}'_u(u, v)$, $\mathbb{E}[g_v(u, v)] = \mathcal{L}'_v(u, v)$. We show the convergence of the proposed stochastic primal-dual gradients for the primal-dual gap $P(u^k, v^k) = \mathcal{L}(u^k, v^\star) - \mathcal{L}(u^\star, v^k)$. We prove the $O(1/\sqrt{k})$ convergence rate in Theorem 4 by using Lemma 4 and Lemma 5, which present the contraction of primal and dual updates, respectively.

Lemma 4. *Suppose $\mathcal{L}(u, v)$ is convex in u and $\mathbb{E}[\|g_u(u, v)\|^2] \leq G_u^2$, we have*

$$\mathbb{E}[\mathcal{L}(u^k, v^k)] - \mathbb{E}[\mathcal{L}(u^\star, v^k)] \leq \frac{1}{2\alpha^k}\left(\mathbb{E}[\|u^k - u^\star\|^2] - \mathbb{E}[\|u^{k+1} - u^\star\|^2]\right) + \frac{\alpha_k}{2} G_u^2 \tag{C.7}$$

Proof. Use primal update in (4.3), we have

$$\|u^{k+1} - u^\star\|^2 = \|u^k - \alpha_k\, g_u(u^k, v^k) - u^\star\|^2 \tag{C.8}$$

$$= \|u^k - u^\star\|^2 - 2\alpha_k\, \langle g_u(u^k, v^k),\, u^k - u^\star\rangle + \alpha_k^2\, \|g_u(u^k, v^k)\|^2. \tag{C.9}$$

Take expectation on both side of the equation, substitute with $\mathbb{E}[g_u(u, v)] = \mathcal{L}'_u(u, v)$ and apply $\mathbb{E}[\|g_u^2(u, v)\|] \leq G_u^2$ to get

$$\mathbb{E}[\|u^{k+1} - u^\star\|^2] \leq \mathbb{E}[\|u^k - u^\star\|^2] - 2\alpha_k\, \mathbb{E}[\langle \mathcal{L}'_u(u^k, v^k),\, u^k - u^\star\rangle] + \alpha_k^2 G_u^2. \tag{C.10}$$

Since $\mathcal{L}(u,v)$ is convex in u, we have

$$\langle \mathcal{L}'_u(u^k, v^k),\, u^k - u^\star \rangle \geq \mathcal{L}(u^k, v^k) - \mathcal{L}(u^\star, v^k). \tag{C.11}$$

(C.7) is proved by combining (C.10) and (C.11). □

Lemma 5. *Suppose $\mathcal{L}(u,v)$ is concave in v and has Lipschitz gradients, $\|\mathcal{L}'_v(u_1,v) - \mathcal{L}'_v(u_2,v)\| \leq L_v\|u_1 - u_2\|$; and bounded variance, $\mathbb{E}[\|g_u(u,v)\|^2] \leq G_u^2$, $\mathbb{E}[\|g_v(u,v)\|^2] \leq G_v^2$; and $\mathbb{E}[\|v^k - v^\star\|^2] \leq D_v^2$, we have*

$$\begin{aligned}&\mathbb{E}[\mathcal{L}(u^k, v^\star)] - \mathbb{E}[\mathcal{L}(u^k, v^k)] \leq \\ &\qquad \frac{1}{2\beta_k}\left(\mathbb{E}[\|v^k - v^\star\|^2] - \mathbb{E}[\|v^{k+1} - v^\star\|^2]\right) + \frac{\beta_k}{2}G_v^2 + \alpha_k L_v\,(G_u^2 + D_v^2).\end{aligned} \tag{C.12}$$

Proof. From the dual update in (4.3), we have

$$\|v^{k+1} - v^\star\|^2 = \|v^k + \beta_k\, g_v(\bar{u}^{k+1}, v^k) - v^\star\|^2 \tag{C.13}$$

$$= \|v^k - v^\star\|^2 + 2\beta_k\, \langle g_v(\bar{u}^{k+1}, v^k),\, v^k - v^\star\rangle + \beta_k^2\, \|g_v(\bar{u}^{k+1}, v^k)\|^2. \tag{C.14}$$

Take expectation on both sides of the equation, substitute $\mathbb{E}[g_v(u,v)] = \mathcal{L}'_v(u,v)$, and apply $\mathbb{E}[\|g_v^2(u,v)\|] \leq G_v^2$ to get

$$\mathbb{E}[\|v^{k+1} - v^\star\|^2] \leq \mathbb{E}[\|v^k - v^\star\|^2] + 2\beta_k\, \mathbb{E}[\langle \mathcal{L}'_v(\bar{u}^{k+1}, v^k),\, v^k - v^\star\rangle] + \beta_k^2\, G_v^2. \tag{C.15}$$

Reorganize (C.15) to get

$$\mathbb{E}[\|v^{k+1} - v^\star\|^2] - \mathbb{E}[\|v^k - v^\star\|^2] - \beta_k^2 \, G_v^2 \le 2\beta_k \, \mathbb{E}[\langle \mathcal{L}_v'(\bar{u}^{k+1}, v^k), \, v^k - v^\star\rangle]. \tag{C.16}$$

The right hand side of (C.16) can be represented as

$$\mathbb{E}[\langle \mathcal{L}_v'(\bar{u}^{k+1}, v^k), \, u^k - v^\star\rangle] \tag{C.17}$$

$$= \mathbb{E}[\langle \mathcal{L}_v'(\bar{u}^{k+1}, v^k) - \mathcal{L}_v'(u^k, v^k) + \mathcal{L}_v'(u^k, v^k), \, v^k - v^\star\rangle] \tag{C.18}$$

$$= \mathbb{E}[\langle \mathcal{L}_v'(\bar{u}^{k+1}, v^k) - \mathcal{L}_v'(u^k, v^k), \, v^k - v^\star\rangle] + \mathbb{E}[\langle \mathcal{L}_v'(u^k, v^k), \, v^k - v^\star\rangle], \tag{C.19}$$

where

$$\mathbb{E}[\langle \mathcal{L}_v'(\bar{u}^{k+1}, v^k) - \mathcal{L}_v'(u^k, v^k), \, v^k - v^\star\rangle] \tag{C.20}$$

$$\le \mathbb{E}[\|\mathcal{L}_v'(\bar{u}^{k+1}, v^k) - \mathcal{L}_v'(u^k, v^k)\| \, \|v^k - v^\star\|] \tag{C.21}$$

$$\le \mathbb{E}[L_v \, \|\bar{u}^{k+1} - u^k\| \, \|v^k - v^\star\|] \tag{C.22}$$

$$= \mathbb{E}[2L_y \, \|u^{k+1} - u^k\| \, \|v^k - v^\star\|] \tag{C.23}$$

$$= \mathbb{E}[2L_y \, \|\alpha_k g_u(u^k, v^k)\| \, \|v^k - v^\star\|] \tag{C.24}$$

$$\le L_y \alpha_k \, \mathbb{E}[\, \|g_u(u^k, v^k)\|^2 + \|v^k - v^\star\|^2] \tag{C.25}$$

$$\le L_y \alpha_k \, (G_u^2 + D_v^2). \tag{C.26}$$

Lipschitz smoothness is used for (C.22); the prediction step in (4.3) is used for (C.23); the primal update in (4.3) is used for (C.24); bounded assumptions are used for (C.26).

Since $\mathcal{L}(u,v)$ is concave in v, we have

$$\langle \mathcal{L}'_v(u^k, v^k),\, v^k - v^\star\rangle \leq \mathcal{L}(u^k, v^k) - \mathcal{L}(u^k, v^\star). \tag{C.27}$$

Combine equations (C.16, C.19, C.26 to getC.27)

$$\begin{aligned} &\frac{1}{2\beta_k}\left(\mathbb{E}[\|v^{k+1} - v^\star\|^2] - \mathbb{E}[\|v^k - v^\star\|^2]\right) - \frac{\beta_k}{2}G_v^2 \\ &\leq L_v\alpha_k\,(G_u^2 + D_v^2) + \mathbb{E}[\mathcal{L}(u^k, v^k)] - \mathbb{E}[\mathcal{L}(u^k, v^\star)]. \end{aligned} \tag{C.28}$$

Rearrange the order of (C.28) to achieve (C.12). □

We now present the proof of Theorem 4.

Proof. Combining (C.7) and (C.12) in the Lemmas, the primal-dual gap $P(u^k, v^k) = \mathcal{L}(u^k, v^\star) - \mathcal{L}(u^\star, v^k)$ satisfies,

$$\begin{aligned} \mathbb{E}[P(u^k, v^k)] \leq &\frac{1}{2\alpha_k}\left(\mathbb{E}[\|u^k - u^\star\|^2] - \mathbb{E}[\|u^{k+1} - u^\star\|^2]\right) + \frac{\alpha_k}{2}G_u^2 \\ &+ \frac{1}{2\beta_k}\left(\mathbb{E}[\|v^k - v^\star\|^2] - \mathbb{E}[\|v^{k+1} - v^\star\|^2]\right) + \frac{\beta_k}{2}G_v^2 + \alpha_k L_v\,(G_u^2 + D_v^2). \end{aligned} \tag{C.29}$$

Accumulate (C.29) from $k = 1, \ldots, l$ to obtain

$$\begin{aligned} &\sum_{k=1}^{l}\mathbb{E}[P(u^k, v^k)] \leq \\ &\frac{1}{2\alpha_1}\mathbb{E}[\|u^1 - u^\star\|^2] + \sum_{k=2}^{l}\left(\frac{1}{2\alpha_k} - \frac{1}{2\alpha_{k-1}}\right)\mathbb{E}[\|u^k - u^\star\|^2] + \sum_{k=1}^{l}\alpha_k\left(\frac{G_u^2}{2} + L_vG_u^2 + L_vD_v^2\right) \\ &+ \frac{1}{2\beta_1}\mathbb{E}[\|v^1 - v^\star\|^2] + \sum_{k=2}^{l}\left(\frac{1}{2\beta_k} - \frac{1}{2\beta_{k-1}}\right)\mathbb{E}[\|v^k - v^\star\|^2] + \sum_{k=1}^{l}\beta_k\frac{G_v^2}{2}. \end{aligned} \tag{C.30}$$

Assume $\mathbb{E}[||u^k - u^\star\|^2] \leq D_u^2$, $\mathbb{E}[||v^k - v^\star\|^2] \leq D_v^2$ are bounded, we have

$$\begin{aligned}\sum_{k=1}^{l} \mathbb{E}[P(u^k, v^k)] \leq & \frac{1}{2\alpha_1} D_u^2 + \sum_{k=2}^{l} (\frac{1}{2\alpha_k} - \frac{1}{2\alpha_{k-1}}) D_u^2 + \sum_{k=1}^{l} \alpha_k (\frac{G_u^2}{2} + L_v G_u^2 + L_v D_v^2) \\ & + \frac{1}{2\beta_1} D_v^2 + \sum_{k=2}^{l} (\frac{1}{2\beta_k} - \frac{1}{2\beta_{k-1}}) D_v^2 + \sum_{k=1}^{l} \beta_k \frac{G_v^2}{2}.\end{aligned} \tag{C.31}$$

Since α_k, β_k are decreasing and $\sum_{k=1}^{l} \alpha_k \leq C_\alpha \sqrt{l+1}$, $\sum_{k=1}^{l} \beta_k \leq C_\beta \sqrt{l+1}$, we have

$$\sum_{k=1}^{l} \mathbb{E}[P(u^k, v^k)] \leq \frac{\sqrt{l}}{2} \left(\frac{D_u^2}{C_\alpha} + \frac{D_v^2}{C_\beta} \right) + \sqrt{l+1} \left(\frac{C_\alpha G_u^2}{2} + C_\beta L_v G_u^2 + C_\alpha L_v D_v^2 + \frac{C_\beta G_v^2}{2} \right) \tag{C.32}$$

For $\hat{u}^l = \frac{1}{l} \sum_{k=1}^{l} u^k$, $\hat{v}^l = \frac{1}{l} \sum_{k=1}^{l} v^k$, because $\mathcal{L}(u, v)$ is convex-concave, we have

$$\mathbb{E}[P(\hat{u}^l, \hat{v}^l)] = \mathbb{E}[\mathcal{L}(\hat{u}^l, v^\star) - \mathcal{L}(u^\star, \hat{v}^l)] \tag{C.33}$$

$$\leq \mathbb{E}[\frac{1}{l} \sum_{k=1}^{l} (\mathcal{L}(u^k, v^\star) - \mathcal{L}(u^\star, v^k))] \tag{C.34}$$

$$= \frac{1}{l} \sum_{k=1}^{l} \mathbb{E}[\mathcal{L}(u^k, v^\star) - \mathcal{L}(u^\star, v^k)] \tag{C.35}$$

$$= \frac{1}{l} \sum_{k=1}^{l} \mathbb{E}[P(u^k, v^k)]. \tag{C.36}$$

Combine (C.32) and (C.36) to prove

$$\mathbb{E}[P(\hat{x}^l, \hat{y}^l)] \leq \frac{1}{2\sqrt{l}} \left(\frac{D_u^2}{C_\alpha} + \frac{D_v^2}{C_\beta} \right) + \frac{\sqrt{l+1}}{l} \left(\frac{C_\alpha G_u^2}{2} + C_\alpha L_v G_u^2 + C_\alpha L_v D_v^2 + \frac{C_\beta G_v^2}{2} \right). \tag{C.37}$$

□

C.2 Experiments

C.2.1 MNIST Toy example

Experimental details: We consider a classic MNIST digits dataset [48] consisting of 60,000 training images and 10,000 testing images each of size 28×28. For simplicity, let us consider a task (T1) of classifying into odd and even numbered images. Let's say, that $\sim 50\%$ of data instances were corrupted using salt and pepper noise of probability 0.2 and this distortion process was biased. Specifically, only 30% of even numbered images were distorted as against the 70% of odd-numbered images. We have observed that any feature representation network θ_f trained using the binary classification loss function for task T1 has noise bias also encoded within it. This was verified by training an independent noise classifier on the learned features. This lead us to design of simple adversarial network to "unlearn" the noise bias from the feature learning pipeline. We formulate this using the minimax objective in (4.5).

In our model, $\mathcal{L}_d$ is a softmax loss for the task (T2) of classifying whether the input sample is noisy or not. $\mathcal{L}_y$ is a softmax loss for task T1 and $\lambda = 1$. A LeNet network [48] is considered for training on task T1 while a two-layer MLP is used for training on task T2. LeNet consist of two convolutional (conv) layers followed by two fully connected (FC) layers at the top. The parameters of conv layers form θ_f while that of FC and MLP layers forms θ_y and θ_d respectively. We train the network in three stages. Following the training on task T1, θ_f were fixed and MLP is trained independently on task T2. The default learning schedule of the LeNet implementation in Caffe [117] were followed for both the tasks. The total training iterations on each task were set to $10,000$. After pretraining, the whole network is jointly finetuned using the adversarial approach.

(4.5) is alternatively minimized w.r.t. $\theta_{\mathbf{f}}, \theta_{\mathbf{y}}$ and maximized w.r.t. $\theta_{\mathbf{d}}$. The predictive steps were only used during the finetuning phase.

Our finding is summarized in Figure 4.2. In addition, Figure C.1 provides head-to-head comparison of two popular solvers Adam and SGD using the predictive step. Not surprisingly, the Adam solver shows relatively better performance and convergence even with an additional predictive step. This also suggests that the default hyper-parameter for the Adam solver can be retained and utilized for training this networks without resorting to any further hyper-parameter tuning (as it is currently in practice).

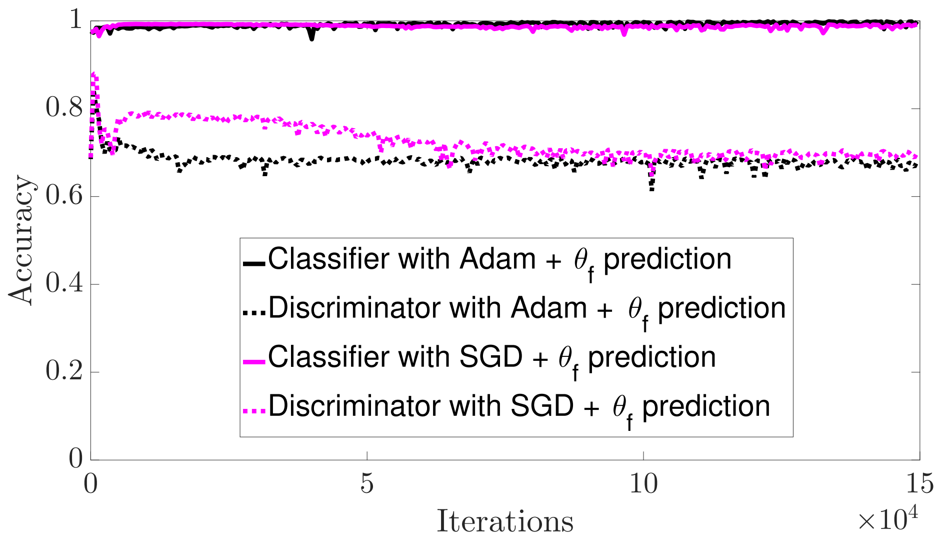


Figure C.1: Comparison of the classification accuracy of parity classification and noise discrimination using the SGD and Adam solvers with and without prediction step.

C.2.2 Domain Adaptation

Experimental details: To evaluate a domain adaptation task, we consider the OFFICE dataset [120]. OFFICE is a small scale dataset consisting of images collected from three distinct domains: AMAZON, DSLR and WEBCAM. For such a small scale dataset, it is non-trivial to learn features from images of a single domain. For instance, consider the largest subset AMAZON, which contains only 2,817 labeled images spread across 31 different categories. However, one can leverage the power of domain adaptation to improve cross domain accuracy. We follow the protocol listed in [22] and the same network architecture is used. Caffe [117] is used for implementation. The training procedure from [22] is kept intact except for the additional prediction step. In Table 4.2 comparisons are drawn with respect to target domain accuracy on three pairs of source-target domain tasks. The test accuracy is reported at the end of 50,000 training iterations.

C.2.3 Fair Classifier

Experimental details: The "Adult" dataset from the UCI machine learning repository is used, which consists of census data from $\sim 45,000$ people. The task is to classify whether a person earns $\geq \$50k$/year. The person's gender is chosen to be the sensitive variable. We binarize all the category attributes, giving us a total of 102 input features per sample. We randomly split data into 35,000 samples for training, 5000 for validation and 5000 for testing. The result reported here is an average over five such random splits.

C.2.4 Generative Adversarial Networks

C.2.4.1 Toy Dataset

To illustrate the advantage of the prediction method, we experiment on a simple GAN architecture with fully connected layers using the toy dataset. The constructed toy example and its architecture is inspired by the one presented in Metz et al. [98]. The two dimensional data is sampled from the mixture of eight Gaussians with their means equally spaced around the unit circle centered at

$(0, 0)$. The standard deviation of each Gaussian is set at 0.01. The two dimensional latent vector $\mathbf{z}$ is sampled from the multivariate Gaussian distribution. The generator and discriminator networks consist of two fully connected hidden layers, each with 128 hidden units and tanh activations. The final layer of the generator has linear activation while that of discriminator has sigmoid activation. The solver optimizes both the discriminator and the generator network using the objective in (4.4). We use adam solver with its default parameters (i.e., learning rate = 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$) and with input batch size of 512. The generated two dimensional samples are plotted in the figure (C.2). The straightforward utilization of the adam solver fails to construct all the modes of the underlying dataset while both unrolled GAN and our method are able to produce all the modes.

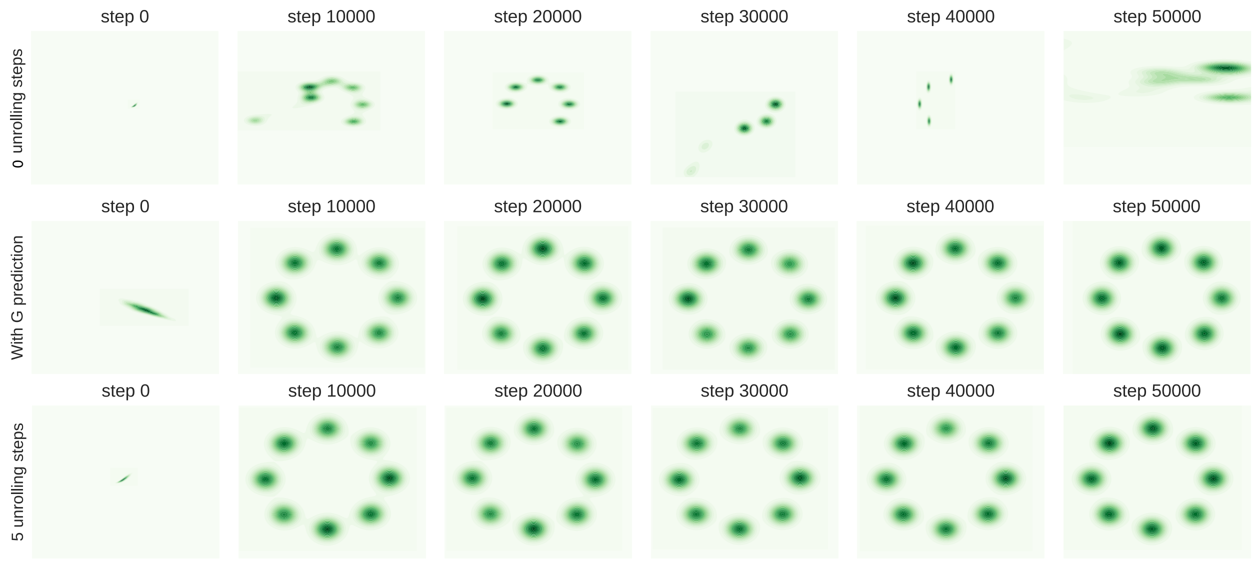


Figure C.2: Comparison of GAN training algorithms on toy dataset. Results on, from top to bottom, GAN, GAN with $\mathbf{G}$ prediction, and unrolled GAN.

We further investigate the performance of GAN training algorithms on data sampled from a mixture of a large number of Gaussians. We use 100 Gaussian modes which are equally spaced around a circle of radius 24 centered at $(0, 0)$. We retain the same experimental settings as described above and train GAN with two different input batch sizes, a small (64) and a large batch (6144) setting.

The Figure (C.3) plots the generated sample output of GAN trained (for fixed number of epochs) under the above setting using different training algorithms. Note that for small batch size input, the default as well as the unrolled training for GAN fails to construct actual modes of the underlying dataset. We hypothesize that this is perhaps due to the batch size, 64, being smaller than the number of input modes (100). When trained with small batch the GAN observe samples only from few input modes at every iteration. This causes instability leading to the failure of training algorithms. This scenario is pertinent to real datasets wherein the number of modes are relatively high compared to input batch size.

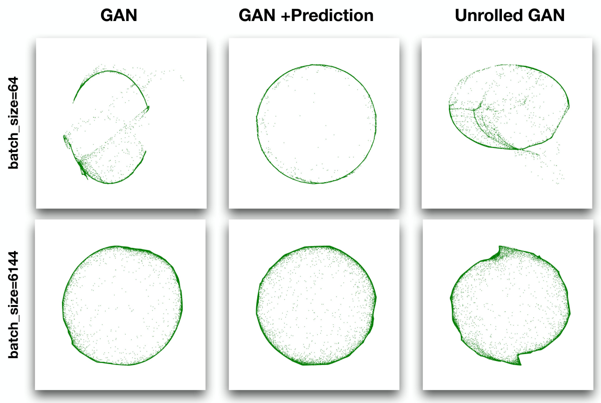
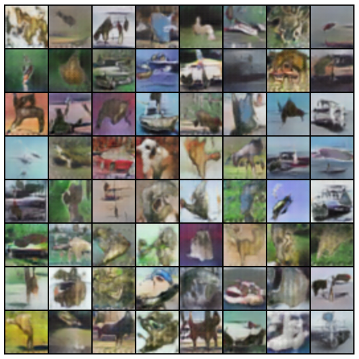


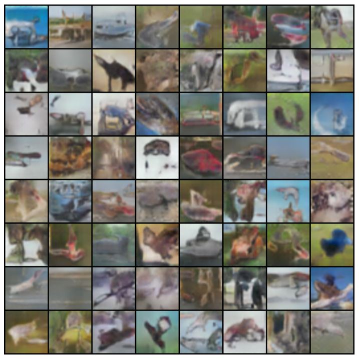
Figure C.3: Comparison of GAN training algorithms on toy dataset of mixture of 100 Gaussians. Results on, from top to bottom, batch size of 64 and 6144.

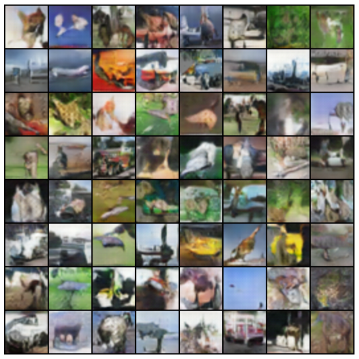
C.2.4.2 Additional DCGAN Results

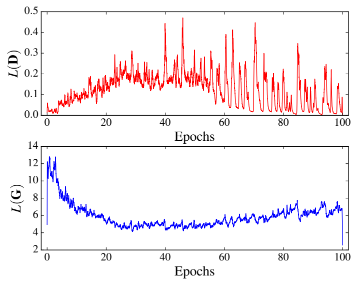
DCGAN Architecture details: For our experiments, we use publicly available code for DCGAN [88] and their implementation for Cifar-10 dataset. The random noise vector is of 100 dimensional and output of the generator network is a 64x64 image of 3 channels.

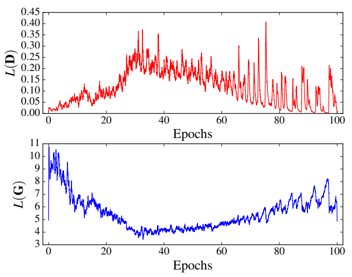
Experiments on Imagenet: In this subsection we demonstrate the advantage of prediction methods for generating higher resolution images of size 128 x 128. For this purpose, the state-of-the-


(a) With **G** prediction


(b) DCGAN


(c) Unrolled GAN


(d) With **G** prediction


(e) DCGAN

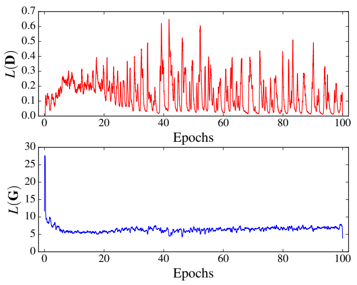
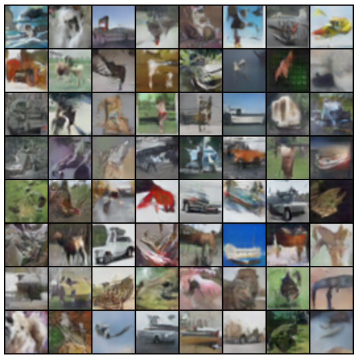
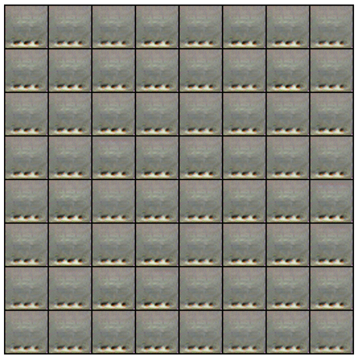
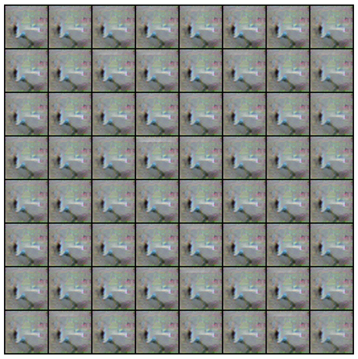
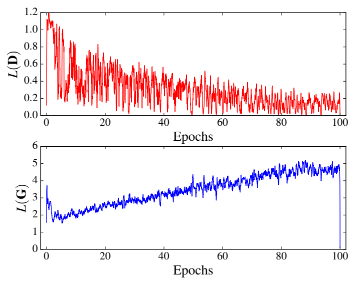

(f) Unrolled GAN

Figure C.4: Comparison of GAN training algorithms for DCGAN architecture on Cifar-10 image datasets. Using higher momentum, $lr = 0.0002, \beta_1 = 0.9$.

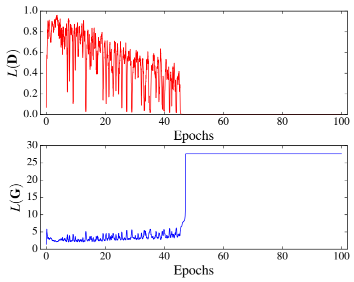

(a) With **G** prediction


(b) DCGAN

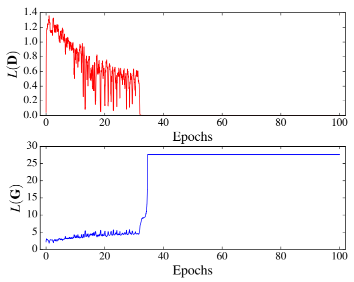

(c) Unrolled GAN



(d) With **G** prediction

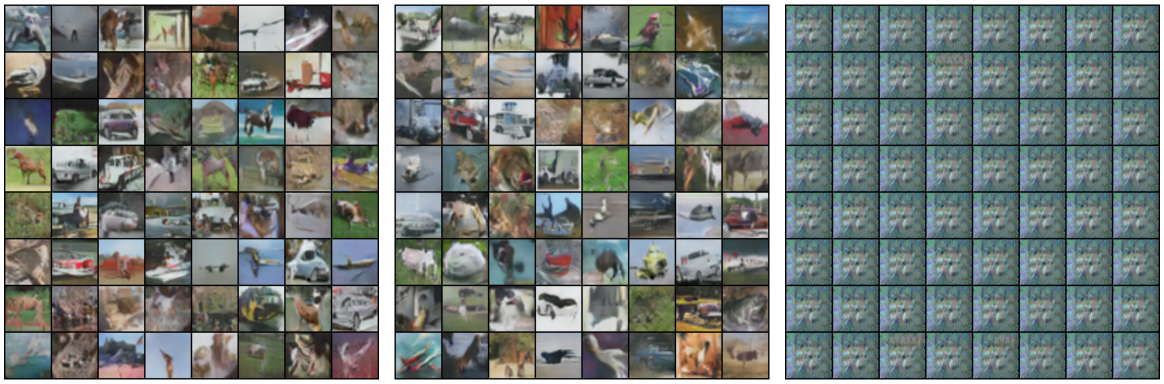


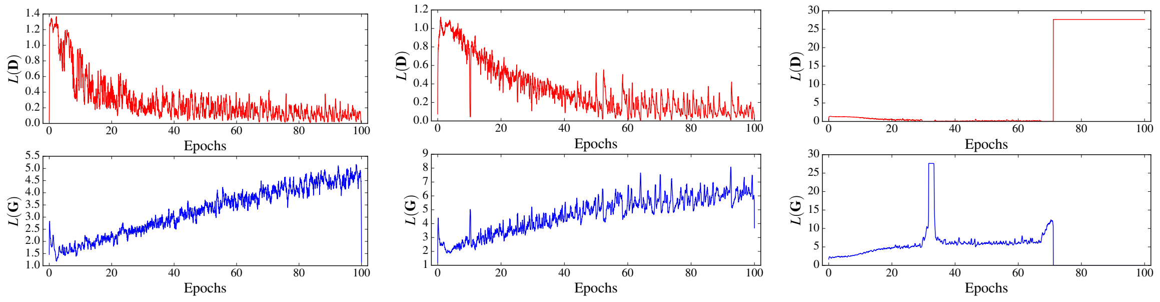
(e) DCGAN



(f) Unrolled GAN

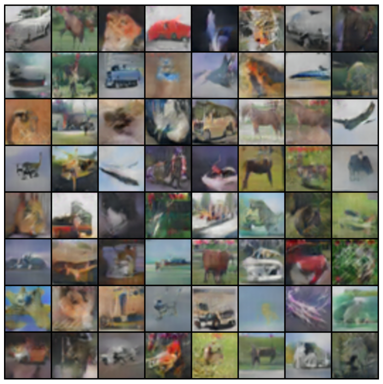
Figure C.5: Comparison of GAN training algorithms for DCGAN architecture on Cifar-10 image datasets. $lr = 0.0004, \beta_1 = 0.5$.


(a) With **G** prediction (b) DCGAN (c) Unrolled GAN

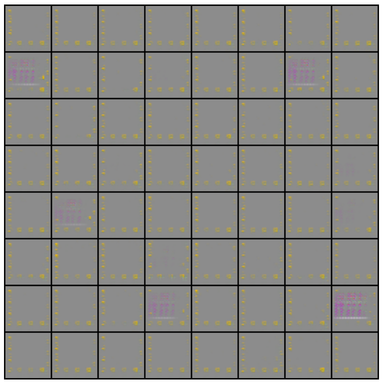


(d) With **G** prediction (e) DCGAN (f) Unrolled GAN

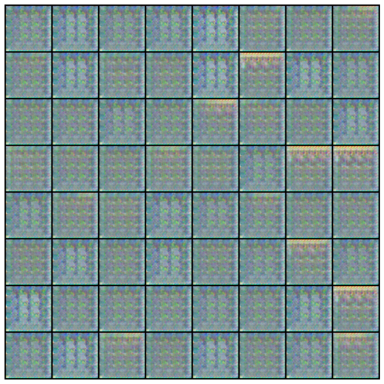
Figure C.6: Comparison of GAN training algorithms for DCGAN architecture on Cifar-10 image datasets. $lr = 0.0006, \beta_1 = 0.5$.



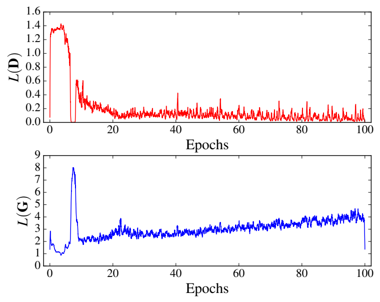
(a) With **G** prediction



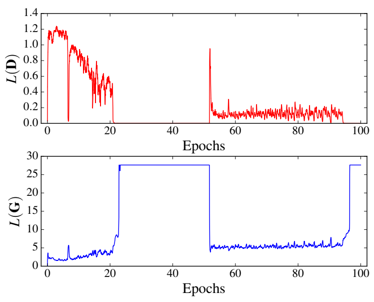
(b) DCGAN



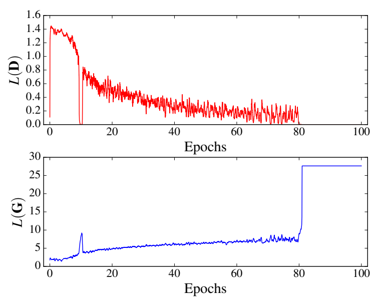
(c) Unrolled GAN



(d) With **G** prediction



(e) DCGAN



(f) Unrolled GAN

Figure C.7: Comparison of GAN training algorithms for DCGAN architecture on Cifar-10 image datasets. $lr = 0.0008, \beta_1 = 0.5$.

art AC-GAN [171] architecture is considered and conditionally learned using images of all 1000 classes from Imagenet dataset. We have used the publicly available code for AC-GAN and all the parameter were set to it default as in Odena et al. [171]. The figure C.8 plots the inception score measured at every training epoch of AC-GAN model with and without prediction. The score is averaged over five independent runs. From the figure, it is clear that even at higher resolution with large number of classes the prediction method is stable and aids in speeding up the training.

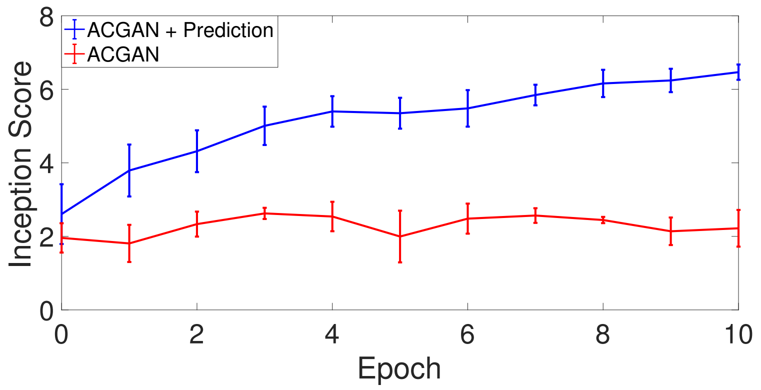


Figure C.8: Comparison of Inception scores on high resolution Imagenet datasets measured at each training epoch of ACGAN model with and without prediction.

Appendix D: On the Similarity between the Laplace and Neural Tangent Kernels

D.1 Formulas for NTK

We begin by providing the recursive definition of NTK for fully connected (FC) networks with bias initialized at zero. The formulation includes a parameter β that when set to zero the recursive formula coincides with the formula given in [124] for bias-free networks.

The network model. We consider a L-hidden-layer fully-connected neural network (in total $L+1$ layers) with bias. Let $\mathbf{x} \in \mathbb{R}^d$ (and denote $d_0 = d$), we assume each layer $l \in [L]$ of hidden units includes d_l units. The network model is expressed as

$$
\begin{aligned}
\mathbf{g}^{(0)}(\mathbf{x}) &= \mathbf{x} \\
\mathbf{f}^{(l)}(\mathbf{x}) &= W^{(l)}\mathbf{g}^{(l-1)}(\mathbf{x}) + \beta \mathbf{b}^{(l)} \in \mathbb{R}^{d_l}, \quad l = 1, \dots L \\
\mathbf{g}^{(l)}(\mathbf{x}) &= \sqrt{\frac{c_\sigma}{d_l}}\sigma\left(\mathbf{f}^{(l)}(\mathbf{x})\right) \in \mathbb{R}^{d_l}, \quad l = 1, \dots L \\
f(\theta, \mathbf{x}) &= f^{(L+1)}(\mathbf{x}) = W^{(L+1)} \cdot \mathbf{g}^{(L)}(\mathbf{x}) + \beta b^{(L+1)}
\end{aligned}
$$

The network parameters θ include $W^{(L+1)}, W^{(L)}, ..., W^{(1)}$, where $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$, $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l \times 1}$, $W^{(L+1)} \in \mathbb{R}^{1 \times d_L}$, $b^{(L+1)} \in \mathbb{R}$, σ is the activation function and $c_\sigma = 1/\left(\mathbb{E}_{z\sim\mathcal{N}(0,1)}[\sigma(z)^2]\right)$. The network parameters are initialized with $\mathcal{N}(0, I)$, except for the biases $\{\mathbf{b}^{(1)}, \dots, \mathbf{b}^{(L)}, b^{(L+1)}\}$, which are initialized with zero.

The recursive formula for NTK. The recursive formula in [123] assumes the bias is initialized with a normal distribution. Here we assume the bias is initialized at zero, yielding a sightly different formulation, which can be readily derived from [123]'s formulation.

Given $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$, we denote the NTK for this fully connected network with bias by $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L+1})}(\mathbf{x}, \mathbf{z}) := \Theta^{(L)}(\mathbf{x}, \mathbf{z})$. The kernel $\Theta^{(L)}(\mathbf{x}, \mathbf{z})$ is defined using the following recursive definition. Let $h \in [L]$ then

$$\Theta^{(h)}(\mathbf{x}, \mathbf{z}) = \Theta^{(h-1)}(\mathbf{x}, \mathbf{z})\dot{\Sigma}^{(h)}(\mathbf{x}, \mathbf{z}) + \Sigma^{(h)}(\mathbf{x}, \mathbf{z}) + \beta^2, \tag{D.1}$$

where

$$\Sigma^{(0)}(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$$

$$\Theta^{(0)}(\mathbf{x}, \mathbf{z}) = \Sigma^{(0)}(\mathbf{x}, \mathbf{z}) + \beta^2.$$

and we define

$$\Sigma^{(h)}(\mathbf{x}, \mathbf{z}) = c_\sigma \mathbb{E}_{(u,v)\backsim N(0,\Lambda^{(h-1)})}\left(\sigma(u)\sigma(v)\right)$$

$$\dot{\Sigma}^{(h)}(\mathbf{x}, \mathbf{z}) = c_\sigma \mathbb{E}_{(u,v)\backsim N(0,\Lambda^{(h-1)})}\left(\dot{\sigma}(u)\dot{\sigma}(v)\right)$$

$$\Lambda^{(h-1)} = \begin{pmatrix} \Sigma^{(h-1)}(\mathbf{x}, \mathbf{x}) & \Sigma^{(h-1)}(\mathbf{x}, \mathbf{z}) \\ \Sigma^{(h-1)}(\mathbf{z}, \mathbf{x}) & \Sigma^{(h-1)}(\mathbf{z}, \mathbf{z}) \end{pmatrix}.$$

Now, let

$$\lambda^{(h-1)}(\mathbf{x}, \mathbf{z}) = \frac{\Sigma^{(h-1)}(\mathbf{x}, \mathbf{z})}{\sqrt{\Sigma^{(h-1)}(\mathbf{x}, \mathbf{x})\Sigma^{(h-1)}(\mathbf{z}, \mathbf{z})}}. \tag{D.2}$$

By definition $|\lambda^{(h-1)}| \le 1$, and for ReLU activation we have $c_\sigma = 2$ and

$$\Sigma^{(h)}(\mathbf{x},\mathbf{z}) = c_\sigma \frac{\lambda^{(h-1)}(\pi - \arccos(\lambda^{(h-1)})) + \sqrt{1-(\lambda^{(h-1)})^2}}{2\pi}\sqrt{\Sigma^{(h-1)}(\mathbf{x},\mathbf{x})\Sigma^{(h-1)}(\mathbf{z},\mathbf{z})} \quad \text{(D.3)}$$

$$\dot{\Sigma}^{(h)}(\mathbf{x},\mathbf{z}) = c_\sigma \frac{\pi - \arccos(\lambda^{(h-1)})}{2\pi}. \quad \text{(D.4)}$$

The parameter β allows us to consider a fully-connected network either with ($\beta > 0$) or without bias ($\beta = 0$). When $\beta = 0$, the recursive formulation is the same as existing derivations, e.g., [123]. Finally, the normalized NTK of a FC network with $L+1$ layers, without bias, is given by $\frac{1}{L+1}\boldsymbol{k}^{\mathrm{FC}_0(\mathrm{L}+1)}(\mathbf{x}_i,\mathbf{x}_j)$.

NTK for a two-layer FC network on $\mathbb{S}^{d-1}$. Using the recursive formulation above, for points on the hypersphere $\mathbb{S}^{d-1}$ NTK for a two-layer FC network with bias initialized at 0, is as follows. Let $u = \mathbf{x}^T\mathbf{z}$, with $\mathbf{x},\mathbf{z} \in \mathbb{S}^{d-1}$. Then,

$$\begin{aligned}\boldsymbol{k}^{\mathrm{FC}_\beta(2)}(\mathbf{x},\mathbf{z}) &= \Theta^{(1)}(\mathbf{x},\mathbf{z}) \\ &= \Theta^{(0)}(\mathbf{x},\mathbf{z})\dot{\Sigma}^{(1)}(\mathbf{x},\mathbf{z}) + \Sigma^{(1)}(\mathbf{x},\mathbf{z}) + \beta^2 \\ &= (u+\beta^2)\frac{\pi - \arccos(u)}{\pi} + \frac{u(\pi - \arccos(u)) + \sqrt{1-u^2}}{\pi} + \beta^2.\end{aligned}$$

Rearranging, we get

$$\boldsymbol{k}^{\mathrm{FC}_\beta(2)}(\mathbf{x},\mathbf{z}) = \boldsymbol{k}^{\mathrm{FC}_\beta(2)}(u) = \frac{1}{\pi}\left((2u+\beta^2)(\pi - \arccos(u)) + \sqrt{1-u^2}\right) + \beta^2. \quad \text{(D.5)}$$

D.2 NTK on $\mathbb{S}^{d-1}$

This section provides a characterization of NTK on the hypersphere $\mathbb{S}^{d-1}$ under the uniform measure. The recursive formulas of the kernels are given in Appendix D.1.

Lemma 6. *Let $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}(\mathbf{x}, \mathbf{z})$, $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$, denote the NTK kernels for FC networks with $L \geq 2$ layers, possibly with bias initialized with zero. This kernel is zonal, i.e., $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}(\mathbf{x}, \mathbf{z}) = \boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}(\mathbf{x}^T\mathbf{z})$.*

Proof. See Appendix D.4.

□

To prove the next theorem, we recall several results on the the arithmetics of RKHS, following [139, 149].

D.2.1 RKHS for sums and products of kernels.

Let $\boldsymbol{k}_1, \boldsymbol{k}_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be kernels with RKHS $\mathcal{H}_{\boldsymbol{k}_1}$ and $\mathcal{H}_{\boldsymbol{k}_2}$, respectively. Then,

1. **Aronszajn's kernel sum theorem.** The RKHS for $\boldsymbol{k} = \boldsymbol{k}_1 + \boldsymbol{k}_2$ is given by $\mathcal{H}_{\boldsymbol{k}_1+\boldsymbol{k}_2} = \{f_1 + f_2 \mid f_1 \in \mathcal{H}_{\boldsymbol{k}_1},\ f_2 \in \mathcal{H}_{\boldsymbol{k}_2}\}$

2. This yields the **kernel sum inclusion.** $\mathcal{H}_{\boldsymbol{k}_1}, \mathcal{H}_{\boldsymbol{k}_2} \subseteq \mathcal{H}_{\boldsymbol{k}_1+\boldsymbol{k}_2}$

3. **Norm addition inequality.** $\|f_1 + f_2\|_{\mathcal{H}_{\boldsymbol{k}_1+\boldsymbol{k}_2}} \leq \|f_1\|_{\mathcal{H}_{\boldsymbol{k}_1}} + \|f_2\|_{\mathcal{H}_{\boldsymbol{k}_2}}$

4. **Norm product inequality.** $\|f_1 \cdot f_2\|_{\mathcal{H}_{\boldsymbol{k}_1 \cdot \boldsymbol{k}_2}} \leq \|f_1\|_{\mathcal{H}_{\boldsymbol{k}_1}} \cdot \|f_2\|_{\mathcal{H}_{\boldsymbol{k}_2}}$

5. **Aronszajn's inclusion theorem.** $\mathcal{H}_{\boldsymbol{k}_1} \subseteq \mathcal{H}_{\boldsymbol{k}_2}$ if and only if $\exists s > 0$, such that $\boldsymbol{k}_1$

Corollary 1. *Let $\mathcal{H}_1, \mathcal{H}_2$ be RKHS on $\mathcal{X}$ with reproducing kernels $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$, respectively and let $\mathcal{H}_{\boldsymbol{k}}$ denote the RKHS of $\boldsymbol{k} = \boldsymbol{k}_1 + \boldsymbol{k}_2$, then $\mathcal{H}_1, \mathcal{H}_2 \subseteq \mathcal{H}_{\boldsymbol{k}}$*

This is true since $\mathcal{H}_1$ and $\mathcal{H}_2$ are linear spaces, therefore $f \equiv 0 \in \mathcal{H}_1, \mathcal{H}_2$. Using Aronszajn's sum of kernels theorem, it yields $\mathcal{H}_1, \mathcal{H}_2 \subseteq \mathcal{H}_{\boldsymbol{k}}$.

D.2.2 The decay rate of the eigenvalues of NTK

Theorem 10. *Let $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$. With bias initialized at zero and $\beta > 0$:*

1. *$\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$ can be decomposed according to*

$$\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}(\mathbf{x}, \mathbf{z}) = \sum_{k=0}^{\infty} \lambda_k \sum_{j=1}^{N(d,k)} Y_{k,j}(\mathbf{x}) Y_{k,j}(\mathbf{z}), \tag{D.6}$$

   with $\lambda_k > 0$ for all $k \geq 0$ and into $Y_{k,j}$ are the spherical harmonics of $\mathbb{S}^{d-1}$, and

2. *$\exists k_0$ and constants $C_1, C_2, C_3 > 0$ that depend on the dimension d such that $\forall k > k_0$*

   (a) $C_1 k^{-d} \leq \lambda_k \leq C_2 k^{-d}$ if $L = 2$, and

   (b) $C_3 k^{-d} \leq \lambda_k$ if $L \geq 3$.

We split the theorem into the next two lemmas. The first lemma handles NTK of two-layer FC networks with bias, and the second lemma handles NTK for deep networks.

Lemma 7. *Let $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$ and $\boldsymbol{k}^{\mathrm{FC}_\beta(2)}(\mathbf{x}^T\mathbf{z})$ as defined in (D.5) with $\beta > 0$. Then, $\boldsymbol{k}^{\mathrm{FC}_\beta(2)}$ decomposes according to (D.6) where $\lambda_k > 0$ for all $k \geq 0$ and $\exists k_0$ such that $\forall k \geq k_0$*

$$C_1 k^{-d} \leq \lambda_k \leq C_2 k^{-d},$$

where $C_1, C_2 > 0$ are constants that depend on the dimension d.

Proof. To prove the lemma we leverage the results of [142, 147]. First, under the assumption of the uniform measure on $\mathbb{S}^{d-1}$, we can apply Mercer decomposition to $\boldsymbol{k}^{\mathrm{FC}_{\beta}(2)}(\mathbf{x}, \mathbf{z})$, where the eigenfunctions are the spherical harmonics. This is due to the observation that $\boldsymbol{k}^{\mathrm{FC}_{\beta}(2)}(\mathbf{x}, \mathbf{z})$ is positive and zonal in $\mathbb{S}^{d-1}$. It is zonal by Lemma 6 and positive, since $\boldsymbol{k}^{\mathrm{FC}_{\beta}(2)}$ can be decomposed as

$$\begin{aligned}
\boldsymbol{k}^{\mathrm{FC}_{\beta}(2)}(u) &= \frac{1}{\pi}\left((2u+\beta^2)(\pi - \arccos(u)) + \sqrt{1-u^2}\right) + \beta^2 \\
&= \frac{1}{\pi}\left(2u(\pi - \arccos(u)) + \sqrt{1-u^2}\right) + \frac{1}{\pi}\beta^2\left(\pi - \arccos(u)\right) + \beta^2 \\
&:= \kappa(\mathbf{x}^T\mathbf{z}) + \beta^2\kappa_0(\mathbf{x}^T\mathbf{z}) + \beta^2,
\end{aligned}$$

where $\kappa(\mathbf{x}^T\mathbf{z})$ is the NTK for a bias-free, two-layer network introduced in [147] and $\kappa_0(\mathbf{x}^T\mathbf{z})$ is known to be the zero-order arc-cosine kernel [172]. By kernel arithmetic, this yields another kernel and this means that $\boldsymbol{k}^{\mathrm{FC}_{\beta}(2)}$ is a positive kernel.

Furthermore, according to Proposition 5 in [147]

$$\kappa(\mathbf{x}^T\mathbf{z}) = \sum_{k=0}^{\infty} \mu_k \sum_{j=1}^{N(d,k)} Y_{k,j}(\mathbf{x})Y_{k,j}(\mathbf{z}),$$

where $Y_{k,j}, j = 1, \ldots, N(d,k)$ are spherical harmonics of degree k, and the eigenvalues μ_k satisfy $\mu_0, \mu_1 > 0$, $\mu_k = 0$ if $k = 2j+1$ with $j \geq 1$ and otherwise, $\mu_k > 0$ and $\mu_k \sim C(d)k^{-d}$ as $k \to \infty$, with $C(d)$ a constant depending only on d. Next, following Lemma 17 in [147] the eigenvalues of $\kappa_0(\mathbf{x}^T\mathbf{z})$, denoted η_k satisfy $\eta_0, \eta_1 > 0$, $\eta_k > 0$ if $k = 2j+1$, with $j \geq 1$ and behave asymptotically as $C_0(d)k^{-d}$. Consequently, $\boldsymbol{k}^{\mathrm{FC}_{\beta}(2)} = \kappa + \beta^2\kappa_0 + \beta^2$, and since both κ and κ_0 have the spherical harmonics as their eigenfunctions, their eigenvalues are given by $\lambda_k = \mu_k + \beta^2\eta_k > 0$ for $k > 0$ and $\lambda_0 = \mu_0 + \beta^2\eta_0 + \beta^2 > 0$, and asymptotically $\lambda_k \sim \tilde{C}(d)k^{-d}$, where $\tilde{C}(d) = C(d) + \beta^2 C_0(d)$. To conclude, this implies that $\exists k_0, C_1(d) > 0$ and $C_2(d) > 0$, such that for all $k \geq k_0$ it holds that

$$C_1 k^{-d} \leq \lambda_k \leq C_2 k^{-d}$$

and also, unless $\beta = 0$, for all $k \geq 0$

$$\lambda_k > 0.$$

$\square$

Next, we prove the second part of Theorem 10 that relates to deep FC networks with bias, $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$, i.e. we prove the following lemma.

Lemma 8. *Let $\mathbf{x}, \mathbf{z} \in \mathbb{S}^{d-1}$ and $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}(\mathbf{x}^T\mathbf{z})$ as defined in Appendix D.1. Then*

1. *$\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$ decomposes according to (D.6) with $\lambda_k > 0$ for all $k \geq 0$*
2. *$\exists k_0$ such that $\forall k > k_0$ it holds that $C_3 k^{-d} \leq \lambda_k$ in which $C_3 > 0$ depends on the dimension d*
3. *$\mathcal{H}^{\mathrm{FC}_\beta(\mathrm{L}-1)} \subseteq \mathcal{H}^{\mathrm{FC}_\beta(\mathrm{L})}$*

Proof. Following Lemma 6, it holds that $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$ is zonal, and therefore can be decomposed according to (D.6). In order to prove the lemma we look at the recursive formulation of the NTK kernel, i.e.,

$$\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l+1})} = \boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l})}\dot{\Sigma}^{(l)} + \Sigma^{(l)} + \beta^2. \tag{D.7}$$

Now, following Lemma 17 in [147] all of the eigenvalues of $\dot{\Sigma}^{(l)}$ are positive, including $\lambda_0 > 0$. This implies that the constant function $g(\mathbf{x}) \equiv 1 \in \mathcal{H}_{\dot{\Sigma}^{(l)}}$.

Now, we use the norm multiplicity inequality in Sec. D.2.1 and show that $\mathcal{H}_{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l})}} \subseteq \mathcal{H}_{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l})} \cdot \dot{\Sigma}^{(l)}}$. Let $f \in \mathcal{H}_{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l})}}$, i.e., $\|f\|_{\mathcal{H}_{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l})}}} < \infty$. We showed that $1 \in \mathcal{H}_{\dot{\Sigma}^{(l)}}$. Therefore, $\|f \cdot 1\|_{\mathcal{H}_{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l})} \cdot \dot{\Sigma}^{(l)}}} \leq \|f\|_{\mathcal{H}_{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l})}}} \|1\|_{\mathcal{H}_{\dot{\Sigma}^{(l)}}} < \infty$, implying that $f \in \mathcal{H}_{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l})} \cdot \dot{\Sigma}^{(l)}}$.

Finally, according to the kernel sum inclusion in Sec. D.2.1, relying on the recursive formulation (D.7) we have $\mathcal{H}_{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l})}} \subseteq \mathcal{H}_{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l})} \cdot \dot{\Sigma}^{(l)}} \subseteq \mathcal{H}_{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{l+1})}}$. Therefore,

$$\mathcal{H}^{\mathrm{FC}_\beta(2)} \subseteq \ldots \subseteq \mathcal{H}^{\mathrm{FC}_\beta(\mathrm{L}-1)} \subseteq \mathcal{H}^{\mathrm{FC}_\beta(\mathrm{L})}. \tag{D.8}$$

This completes the proof, by using Aronszan's inclusion theorem as follows. Since $H^{k^{FC(2)}} \subseteq H^{k^{FC(L)}}$, then by Aronszajn's inclusion theorem $\exists s > 0$ such that $\boldsymbol{k}^{\mathrm{FC}_\beta(2)} << s^2 \boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$. Since the kernels are zonal on the sphere (with uniform distribution of the data) their corresponding RKHS share the same eigenfunctions, namely the spherical harmonics.

Therefore, for all $k \geq 0$ it holds

$$s^2 \lambda_k^{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}} \geq \lambda_k^{\boldsymbol{k}^{\mathrm{FC}_\beta(2)}} > 0$$

and for $k \to \infty$ it holds that

$$s^2 \lambda_k^{\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}} \geq \lambda_k^{\boldsymbol{k}^{\mathrm{FC}_\beta(2)}} \geq \frac{C_1}{k^d}$$

completing the proof.

□

D.3 Laplace Kernel in $\mathbb{S}^{d-1}$

The Laplace kernel $\boldsymbol{k}(\mathbf{x},\mathbf{y}) = e^{-\bar{c}\|\mathbf{x}-\mathbf{y}\|}$ restricted to the sphere $\mathbb{S}^{d-1}$ is defined as

$$K(\mathbf{x},\mathbf{y}) = \boldsymbol{k}(\mathbf{x}^T\mathbf{y}) = e^{-c\sqrt{1-x^Ty}} \tag{D.9}$$

where $c > 0$ is a tuning parameter. We next prove an asymptotic bound on its eigenvalues.

Theorem 11. *Let $\mathbf{x},\mathbf{y} \in \mathbb{S}^{d-1}$ and $\boldsymbol{k}(\mathbf{x}^T\mathbf{y}) = e^{-c\sqrt{1-\mathbf{x}^T\mathbf{y}}}$ be the Laplace kernel, restricted to $\mathbb{S}^{d-1}$. Then $\boldsymbol{k}$ can be decomposed as in (D.6) with the eigenvalues λ_k satisfying $\lambda_k > 0$ for all $k \geq 0$ and $\exists k_0$ such that $\forall k > k_0$ it holds that:*

$$B_1 k^{-d} \leq \lambda_k \leq B_2 k^{-d}$$

where $B_1, B_2 > 0$ are constants that depend on the dimension d and the parameter c.

Our proof relies on several supporting lemmas.

Lemma 9. *([150] Thm 1.14 page 6) For all $\alpha > 0$ it holds that*

$$\int_{\mathbb{R}^d} e^{-2\pi\|\mathbf{x}\|\alpha} e^{-2\pi i\mathbf{t}\cdot\mathbf{x}} d\mathbf{x} = c_d \frac{\alpha}{(\alpha^2 + \|\mathbf{t}\|^2)^{(d+1)/2}}, \tag{D.10}$$

where $c_d = \Gamma(\frac{d+1}{2})/(\pi^{(d+1)/2})$

Lemma 10. *Let $f(\mathbf{x}) = e^{-c\|\mathbf{x}\|}$ with $\mathbf{x} \in \mathbb{R}^d$. Then, its Fourier transform $\Phi(\mathbf{w})$ with $\mathbf{w} \in \mathbb{R}^d$ is $\Phi(\mathbf{w}) = \Phi(\|\mathbf{w}\|) = C(1 + \|\mathbf{w}\|^2/c^2)^{-(d+1)/2}$ for some constant $C > 0$.*

Proof. To calculate the Fourier transform we need to calculate the following integral

$$\Phi(\mathbf{w}) = \frac{1}{(2\pi)^d} \int_{\mathbb{R}^d} e^{-c\|\mathbf{x}\|} e^{-i\mathbf{x}\cdot\mathbf{w}} d\mathbf{x}.$$

According to the Lemma 9, plugging $\alpha = \frac{c}{2\pi}$ and $\mathbf{t} = \frac{\mathbf{w}}{2\pi}$ into (D.10) yields

$$\Phi(\mathbf{w}) = c_d \frac{c}{(c^2 + \|\mathbf{w}\|^2)^{(d+1)/2}} = \frac{c_d}{c^{(d+1)}} \frac{1}{\left(1 + \frac{\|\mathbf{w}\|^2}{c^2}\right)^{(d+1)/2}} = C\left(1 + \frac{\|\mathbf{w}\|^2}{c^2}\right)^{-(d+1)/2}$$

with $C = \frac{c_d}{c^{(d+1)}} > 0$.

$\square$

Lemma 11. *([154] Thm. 4.1) Let $f(\mathbf{x})$ be defined as $f(\|\mathbf{x}\|)$ for all $\mathbf{x} \in \mathbb{R}^d$, and let $\Phi(\mathbf{w}) = \Phi(\|\mathbf{w}\|)$ denote its Fourier Transform in $\mathbb{R}^d$. Then, its corresponding kernel on $\mathbb{S}^{d-1}$ is defined as the restriction $\boldsymbol{k}(\mathbf{x}^T\mathbf{y}) = f(\|\mathbf{x} - \mathbf{y}\|)$ with $\mathbf{x}, \mathbf{y} \in \mathbb{S}^{d-1}$. By Mercer's Theorem the spherical harmonic expansion of $\boldsymbol{k}(\mathbf{x}^T\mathbf{y})$ is of the form*

$$\boldsymbol{k}(\mathbf{x}^T\mathbf{y}) = \sum_{k=0}^{\infty} \lambda_k \sum_{j=1}^{N(d,k)} Y_{k,j}(\mathbf{x}) Y_{k,j}(\mathbf{y}).$$

Then, the eigenvalues in the spherical harmonic expansion λ_k are related to the Fourier coefficients of f, $\Phi(t)$, as follows

$$\lambda_k = \int_o^{\infty} t\Phi(t) J^2_{k+\frac{d-2}{2}}(t) dt, \tag{D.11}$$

where $J_v(t)$ is the usual Bessel function of the first kind of order v.

Having, these supporting Lemmas, we can now prove **Theorem 11**.

Proof. First, $\boldsymbol{k}(\cdot,\cdot)$ is a positive zonal kernel and hence can be written as

$$\boldsymbol{k}(\mathbf{x}^T\mathbf{y}) = \sum_{k=0}^{\infty} \lambda_k \sum_{j=1}^{N(d,k)} Y_{k,j}(\mathbf{x})Y_{k,j}(\mathbf{y}).$$

Next, to derive the bounds we plug the Fourier coefficients, $\Phi(\omega)$, computed in Lemma 10, into the expression for the harmonic coefficients, λ_k (D.11), obtaining

$$\lambda_k = C\int_0^{\infty} \frac{t}{\left(1+\frac{t^2}{c^2}\right)^{\frac{d+1}{2}}} J^2_{k+\frac{d-2}{2}}(t)dt.$$

Applying a change of variables $t = cx$ we get

$$\lambda_k = c^2 C\int_0^{\infty} \frac{x}{(1+x^2)^{\frac{d+1}{2}}} J^2_{k+\frac{d-2}{2}}(cx)dx. \tag{D.12}$$

We next bound this integral from both above and below. To get an upper bound we observe that for $x \in [0,\infty)$ $x^2 < 1+x^2$, implying that $x(1+x^2)^{-(d+1)/2} < x^{-d}$, and consequently

$$\lambda_k < c^2 C\int_0^{\infty} x^{-d} J^2_{k+\frac{d-2}{2}}(cx)dx := c^2 C A(k,d,c).$$

The above integral $A(k,d,c)$ was computed in [173] (Sec. 13.41 page 402 with $a := c$, $\lambda := d$, and $\mu = \nu := k+(d-2)/2$) which gives

$$A(k,d,c) = \int_0^{\infty} x^{-d} J^2_{k+\frac{d-2}{2}}(cx)dx = \frac{(\frac{c}{2})^{d-1}\Gamma(d)\Gamma(k-\frac{1}{2})}{2\Gamma^2(\frac{d+1}{2})\Gamma(k+d-\frac{1}{2})}. \tag{D.13}$$

Using Stirling's formula $\Gamma(x) = \sqrt{2\pi}x^{x-1/2}e^{-x}(1+O(x^{-1}))$ as $x \to \infty$. Consequently, for sufficiently large $k >> d$

$$\begin{aligned}
\lambda_k < c^2 C A(k,d,c) &= c^2 C \frac{(\frac{c}{2})^{d-1}\Gamma(d)\Gamma(k-\frac{1}{2})}{2\Gamma^2(\frac{d+1}{2})\Gamma(k+d-\frac{1}{2})} \\
&\sim c^2 C \frac{(\frac{c}{2})^{d-1}\Gamma(d)}{2\Gamma^2(\frac{d+1}{2})} \cdot \frac{(k-\frac{1}{2})^{k-1}e^{-k+\frac{1}{2}}}{(k+d-\frac{1}{2})^{k+d-1}e^{-k-d+\frac{1}{2}}}(1+O(k^{-1})) \\
&= B_2 k^{-d}, \qquad \text{(D.14)}
\end{aligned}$$

where B_2 depends on c, C and the dimension d.

We use again the relation (D.12) to derive a lower bound for λ_k. First, note that since $t, 1+t^2, J_v^2(t)$ are all non-negative for $t \in [0,\infty)$ and therefore

$$\begin{aligned}
\lambda_k &\geq c^2 C \int_1^\infty \frac{x}{(1+x^2)^{\frac{d+1}{2}}} J^2_{k+\frac{d-2}{2}}(cx)dx \geq c^2 C \int_1^\infty \frac{1}{2^{\frac{d+1}{2}}x^d} J^2_{k+\frac{d-2}{2}}(cx)dx \\
&= \frac{Cc^2}{2^{\frac{d+1}{2}}}\left(\int_0^\infty x^{-d}J^2_{k+\frac{d-2}{2}}(cx)dx - \int_0^1 x^{-d}J^2_{k+\frac{d-2}{2}}(cx)dx\right) \\
&= \frac{Cc^2}{2^{\frac{d+1}{2}}}\int_0^\infty x^{-d}J^2_{k+\frac{d-2}{2}}(cx)dx \left(1 - \frac{\int_0^1 x^{-d}J^2_{k+\frac{d-2}{2}}(cx)dx}{\int_0^\infty x^{-d}J^2_{k+\frac{d-2}{2}}(cx)dx}\right) \\
&= \frac{Cc^2}{2^{\frac{d+1}{2}}}A(k,d,c)\left(1 - \frac{B(k,d,c)}{A(k,d,c)}\right),
\end{aligned}$$

where $B(k,d,c) := \int_0^1 x^{-d}J^2_{k+\frac{d-2}{2}}(cx)dx$. The first integral, $A(k,d,c)$, was shown in (D.14) to converge asymptotically to B_2k^{-d}. To bound the second integral, $B(k,d,c)$, we use an inequality from [173] (Section 3.31, page 49), which states that for $v,t \in \mathbb{R}$, $v > -\frac{1}{2}$,

$$|J_v(t)| \leq \frac{2^{-v}t^v}{\Gamma(v+1)}.$$

This gives an upper bound for $B(k,d,c)$

$$B(k,d,c) = \int_0^1 x^{-d} J^2_{k+\frac{d-2}{2}}(cx)dx \le \int_0^1 x^{-d} \frac{2^{-2(k+\frac{d-2}{2})}(cx)^{2(k+\frac{d-2}{2})}}{\Gamma^2(k+\frac{d}{2})} dx \le \frac{(\frac{c}{2})^{2(k+\frac{d-2}{2})}}{\Gamma^2(k+\frac{d}{2})}.$$

Applying Stirling's formula we obtain $B(k,d,c) \le O\left(\frac{(\frac{ce}{2})^{2(k+\frac{d}{2})}(k+d)}{(k+\frac{d}{2})^{2(k+\frac{d}{2})}}\right)$, which implies that as k grows, $\frac{B(k,d,c)}{A(k,d,c)} \to 0$. Therefore, asymptotically for large k

$$\lambda_k \ge \frac{Cc^2}{2^{\frac{d+1}{2}}} A(k,d,c) \left(1 - \frac{B(k,d,c)}{A(k,d,c)}\right) \ge \frac{Cc^2}{2^{\frac{d+1}{2}}} A(k,d,c),$$

from which we conclude that $\lambda_k > B_1 k^{-d}$, where the constant B_1 depends on c, C, and d. We have therefore shown that there exists k_0 such that $\forall k > k_0$

$$B_1 k^{-d} \le \lambda_k \le B_2 k^{-d}.$$

Finally, to show that $\lambda_k > 0$ for all $k \ge 0$ we use again (D.11) in Lemma 11 which states that

$$\lambda_k = \int_0^\infty t\Phi(t) J^2_{k+\frac{d-2}{2}}(t)dt.$$

Note that in the interval $(0,\infty)$ it holds that $t > 0$ and $\Phi(t) > 0$ due to Lemma 10. Therefore $\lambda_k = 0$ implies that $J^2_{k+\frac{d-2}{2}}(t)$ is identically 0 on $(0,\infty)$, contradicting the properties of the Bessel function of the first kind. Hence, $\lambda_k > 0$ for all k. $\square$

D.3.1 Proof of main theorem

Theorem 12. *Let $\mathcal{H}^{\mathrm{Lap}}$ denote the RKHS for the Laplace kernel restricted to $\mathbb{S}^{d-1}$, and let $\mathcal{H}^{\mathrm{FC}_\beta(\mathrm{L})}$ denote the NTK corresponding to a FC network with L layers with bias, restricted to $\mathbb{S}^{d-1}$, then $\mathcal{H}^{\mathrm{Lap}} = \mathcal{H}^{\mathrm{FC}_\beta(2)} \subseteq \mathcal{H}^{\mathrm{FC}_\beta(\mathrm{L})}$.*

Proof. Let λ_k^{Lap}, $\lambda_k^{\mathrm{FC}_\beta(2)}$, and $\lambda_k^{\mathrm{FC}_\beta(\mathrm{L})}$ denote the eigenvalues of the three kernel, $\boldsymbol{k}^{\mathrm{Lap}}$, $\boldsymbol{k}^{\mathrm{FC}_\beta(2)}$, and $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$ in their Mercer's decomposition, i.e.,

$$\boldsymbol{k}(\mathbf{x}^T\mathbf{z}) = \sum_{k=0}^{\infty} \lambda_k \sum_{j=1}^{N(d,k)} Y_{k,j}(\mathbf{x}) Y_{k,j}(\mathbf{z}).$$

Denote by k_0 the smallest k for which Theorems 10 and 11 hold simultaneously. We first show that $\mathcal{H}^{\mathrm{Lap}} \subseteq \mathcal{H}^{\mathrm{FC}_\beta(2)}$. Let $f(\mathbf{x}) \in \mathcal{H}^{\mathrm{Lap}}$, and let $f(\mathbf{x}) = \sum_{k=0}^{\infty}\sum_{j=0}^{N(d,k)} \alpha_{k,j} Y_{k,j}(\mathbf{x})$ denote its spherical harmonic decomposition. Then $\|f\|_{\mathcal{H}^{\mathrm{Lap}}} < \infty$ implies, due to Theorem 11, that

$$\sum_{k=k_0}^{\infty}\sum_{j=0}^{N(d,k)} \frac{1}{B_2} k^d \alpha_{k,j}^2 \leq \sum_{k=k_0}^{\infty}\sum_{j=0}^{N(d,k)} \frac{\alpha_{k,j}^2}{\lambda_k^{\mathrm{Lap}}} < \infty.$$

Combining this with Theorem 10, and recalling that $\lambda_k^{\mathrm{FC}_\beta(2)} > 0$ for all $k \geq 0$), we have

$$\sum_{k=k_0}^{\infty}\sum_{j=0}^{N(d,k)} \frac{\alpha_{k,j}^2}{\lambda_k^{\mathrm{FC}_\beta(2)}} \leq \sum_{k=k_0}^{\infty}\sum_{j=0}^{N(d,k)} \frac{1}{C_1} k^d \alpha_{k,j}^2 = \frac{B_2}{C_1} \sum_{k=k_0}^{\infty}\sum_{j=0}^{N(d,k)} \frac{1}{B_2} k^d \alpha_{k,j}^2 < \infty,$$

implying that $\|f\|^2_{\mathcal{H}^{\mathrm{FC}_\beta(2)}} < \infty$, and so $\mathcal{H}^{\mathrm{Lap}} \subseteq \mathcal{H}^{\mathrm{FC}_\beta(2)}$. Similar arguments can be used to show that $\mathcal{H}^{\mathrm{FC}_\beta(2)} \subseteq \mathcal{H}^{\mathrm{Lap}}$, proving that $\mathcal{H}^{\mathrm{FC}_\beta(2)} = \mathcal{H}^{\mathrm{Lap}}$. Finally, following the inclusion relation (D.8) the theorem is proved. $\square$

D.4 NTK in $\mathbb{R}^d$

In this section we denote $r_x = \|\mathbf{x}\|$, $r_z = \|\mathbf{z}\|$ and by $\hat{\mathbf{x}} = \mathbf{x}/r_x$, $\hat{\mathbf{z}} = \mathbf{z}/r_z$. We first prove Theorem 13 and as a consequence Lemma 12 is proved.

Theorem 13. *Let $\boldsymbol{k}^{\mathrm{FC_0(L)}}(\mathbf{x},\mathbf{z}), \boldsymbol{k}^{\mathrm{FC_\beta(L)}}(\mathbf{x},\mathbf{z})$, $\mathbf{x},\mathbf{z} \in \mathbb{R}^d$, denote the NTK kernel with L layers without bias and with bias initialized at zero, respectively. It holds that (1) Bias-free $\boldsymbol{k}^{\mathrm{FC_0(L)}}$ is homogeneous of order 1. (2) Let $\boldsymbol{k}^{\mathrm{Bias(L)}} = \boldsymbol{k}^{\mathrm{FC_\beta(L)}} - \boldsymbol{k}^{\mathrm{FC_0(L)}}$. Then, $\boldsymbol{k}^{\mathrm{Bias(L)}}$ is homogeneous of order 0.*

Lemma 12. *Let $\boldsymbol{k}^{\mathrm{FC_\beta(L)}}(\mathbf{x},\mathbf{z})$, $\mathbf{x},\mathbf{z} \in \mathbb{S}^{d-1}$, denote the NTK kernels for FC networks with $L \geq 2$ layers, possibly with bias initialized with zero. This kernel is zonal, i.e., $\boldsymbol{k}^{\mathrm{FC_\beta(L)}}(\mathbf{x},\mathbf{z}) = \boldsymbol{k}^{\mathrm{FC_\beta(L)}}(\mathbf{x}^T\mathbf{z})$.*

To that end, we first prove the following supporting Lemma.

Lemma 13. *For $\mathbf{x},\mathbf{z} \in \mathbb{R}^d$ it holds that*

$$\Theta^{(L)}(\mathbf{x},\mathbf{z}) = r_x r_z \Theta^{(L)}\left(\hat{\mathbf{x}},\hat{\mathbf{z}}\right) = r_x r_z \Theta^{(L)}(\hat{\mathbf{x}}^T\hat{\mathbf{z}}),$$

where $\Theta^{(L)} = \boldsymbol{k}^{\mathrm{FC_0(L+1)}}$, as defined in Appendix D.1.

Proof. We prove this by induction over the recursive definition of $\boldsymbol{k}^{\mathrm{FC_0(L+1)}} = \Theta^{(L)}(\mathbf{x},\mathbf{z})$. Let $\mathbf{x},\mathbf{z} \in \mathbb{R}^d$, then by definition

$$\Theta^{(0)}(\mathbf{x},\mathbf{z}) = \mathbf{x}^T\mathbf{z} = r_x r_z \Theta^{(0)}\left(\hat{\mathbf{x}},\hat{\mathbf{z}}\right) = r_x r_z \Theta^{(0)}\left(\hat{\mathbf{x}}^T\hat{\mathbf{z}}\right)$$

and

$$\Sigma^{(0)}(\mathbf{x},\mathbf{z}) = \mathbf{x}^T\mathbf{z} = r_x r_z \Sigma^{(0)}\left(\hat{\mathbf{x}},\hat{\mathbf{z}}\right) = r_x r_z \Sigma^{(0)}\left(\hat{\mathbf{x}}^T\mathbf{z}\right)$$

Assuming the induction hypothesis holds for l, i.e.,

$$\Theta^{(l)}(\mathbf{x},\mathbf{z}) = r_x r_z \Theta^{(l)}\left(\hat{\mathbf{x}},\hat{\mathbf{z}}\right) = r_x r_z \Theta^{(l)}\left(\hat{\mathbf{x}}^T\mathbf{z}\right)$$

and

$$\Sigma^{(l)}(\mathbf{x},\mathbf{z}) = r_x r_z \Sigma^{(l)}\left(\hat{\mathbf{x}},\hat{\mathbf{z}}\right) = r_x r_z \Sigma^{(l)}\left(\hat{\mathbf{x}}^T\hat{\mathbf{z}}\right)$$

we prove that those equalities are also true for $l+1$.

By the definition of $\lambda^{(l)}$ (D.2) and the induction hypothesis for $\Sigma^{(l)}$ we have that

$$\lambda^{(l)}(\mathbf{x},\mathbf{z}) = \frac{\Sigma^{(l)}(\mathbf{x},\mathbf{z})}{\sqrt{\Sigma^{(l)}(\mathbf{x},\mathbf{x})\Sigma^{(l)}(\mathbf{z},\mathbf{z})}} = \frac{\Sigma^{(l)}\left(\hat{\mathbf{x}},\hat{\mathbf{z}}\right)}{\sqrt{\Sigma^{(l)}\left(\hat{\mathbf{x}}_i,\hat{\mathbf{x}}\right)\Sigma^{(l)}\left(\hat{\mathbf{z}},\hat{\mathbf{z}}\right)}} = \lambda^{(l)}\left(\hat{\mathbf{x}},\hat{\mathbf{z}}\right) = \lambda^{(l)}\left(\hat{\mathbf{x}}^T\hat{\mathbf{z}}\right)$$

Plugging this result in the definitions of Σ (D.3) and $\dot{\Sigma}$ (D.4), using the induction hypothesis we obtain

$$\begin{aligned}\Sigma^{(l+1)}(\mathbf{x},\mathbf{z}) &= r_x r_z \Sigma^{(l+1)}\left(\hat{\mathbf{x}},\hat{\mathbf{z}}\right) = r_x r_z \Sigma^{(l+1)}\left(\hat{\mathbf{x}}^T\hat{\mathbf{z}}\right) \\ \dot{\Sigma}^{(l+1)}(\mathbf{x},\mathbf{z}) &= \dot{\Sigma}^{(l+1)}\left(\hat{\mathbf{x}},\hat{\mathbf{z}}\right) = \dot{\Sigma}^{(l+1)}\left(\hat{\mathbf{x}}^T\hat{\mathbf{z}}\right) \qquad \text{(D.15)}\end{aligned}$$

Finally, using the recursion formula (D.1) ($\beta = 0$) and the induction hypothesis for $\Theta^{(l)}$, we obtain

$$\Theta^{(l+1)}(\mathbf{x},\mathbf{z}) = r_x r_z \Theta^{(l+1)}\left(\hat{\mathbf{x}},\hat{\mathbf{z}}\right) = r_x r_z \Theta^{(l+1)}\left(\hat{\mathbf{x}}^T\hat{\mathbf{z}}\right)$$

□

A corollary of this Lemma is that $\boldsymbol{k}^{\mathrm{FC}_0(\mathrm{L})}$ is homogeneous of order 1 in $\mathbb{R}^d$, proving the first part of Theorem 13. Also, it is homogeneous of order 0 in $\mathbb{S}^{d-1}$, proving Lemma 12 for $\beta = 0$.

We next turn to proving the second part of Theorem 13, i.e., that $\boldsymbol{k}^{\mathrm{Bias(L)}} = \boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})} - \boldsymbol{k}^{\mathrm{FC}_0(\mathrm{L})}$ is homogeneous of order 0 in $\mathbb{R}^d$. By rewriting the recursive definition of $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$, shown in Appendix D.1, we can express $\boldsymbol{k}^{\mathrm{Bias(L)}}$ in the following recursive manner $\boldsymbol{k}^{\mathrm{Bias(1)}} = \beta^2$, and $\boldsymbol{k}^{\mathrm{Bias(l+1)}} = \boldsymbol{k}^{\mathrm{Bias(l)}}\dot{\Sigma} + \beta^2$. Therefore, $\boldsymbol{k}^{\mathrm{Bias(L)}}$ is homogeneous of order zero, since it depends only on $\dot{\Sigma}$, which is by itself homogeneous of order zero (D.15). This concludes Theorem 13.

Finally, Lemma 12 is proved, since $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})} = \boldsymbol{k}^{\mathrm{FC}_0(\mathrm{L})} + \boldsymbol{k}^{\mathrm{Bias(L)}}$, and when restricted to $\mathbb{S}^{d-1}$ both components are homogeneous of order 0.

Theorem 14. *Let $p(r)$ be a decaying density on $[0, \infty)$ such that $0 < \int_0^\infty p(r) r^2 dr < \infty$ and $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$.*

1. *Let $\boldsymbol{k}_0(\mathbf{x}, \mathbf{z})$ be homogeneous of order 1 such that $\boldsymbol{k}_0(\mathbf{x}, \mathbf{z}) = r_x r_z \hat{\boldsymbol{k}}_0(\hat{\mathbf{x}}^T \hat{\mathbf{z}})$. Then its eigenfunctions with respect to $p(r_x)$ are given by $\Psi_{k,j} = a r_x Y_{k,j}(\hat{\mathbf{x}})$, where $Y_{k,j}$ are the spherical harmonics in $\mathbb{S}^{d-1}$ and $a \in \mathbb{R}$.*

2. *Let $\boldsymbol{k}(\mathbf{x}, \mathbf{z}) = \boldsymbol{k}_0(\mathbf{x}, \mathbf{z})$ + $\boldsymbol{k}_1(\mathbf{x}, \mathbf{z})$ so that $\boldsymbol{k}_0$ as in 1 and $\boldsymbol{k}_1$ is homogeneous of order 0. Then the eigenfunctions of $\boldsymbol{k}$ are of the form $\Psi_{k,j} = (a r_x + b)\, Y_{k,j}(\hat{\mathbf{x}})$.*

Proof. 1. Since $\hat{\boldsymbol{k}}_0$ is zonal, its Mercer's representation reads

$$\hat{\boldsymbol{k}}_0(\hat{\mathbf{x}}, \hat{\mathbf{z}}) = \sum_{k=0}^{\infty} \lambda_k \sum_{j=1}^{N(d,k)} Y_{k,j}(\hat{\mathbf{x}}) Y_{k,j}(\hat{\mathbf{z}}),$$

where the spherical harmonics $Y_{k,j}$ are the eigenfunctions of $\hat{\boldsymbol{k}}_0$. Consequently, as noted also in [147],

$$\boldsymbol{k}_0(\mathbf{x},\mathbf{z}) = a^2 \sum_{k=0}^{\infty} \lambda_k \sum_{j=1}^{N(d,k)} r_x Y_{k,j}(\hat{\mathbf{x}}) r_z Y_{k,j}(\hat{\mathbf{z}}).$$

The orthogonality of the eigenfunctions $\Psi_{k,j}(\mathbf{x}) = a r_x Y_{k,j}(\hat{\mathbf{x}})$ is verified as follows. Let $\bar{p}(\mathbf{x})$ denote a probability density on $\mathbb{R}^d$ such that $\bar{p}(\mathbf{x}) = p(r_x)/A(r_x)$, where $A(r_x)$ denotes the surface area of a sphere of radius r_x in $\mathbb{R}^d$. Then,

$$\int_{\mathbb{R}^d} \Psi_{k,j}(\mathbf{x})\Psi_{k',j'}(\mathbf{x})\bar{p}(\mathbf{x})d\mathbf{x} = a^2 \int_0^{\infty} \frac{r_x^{d+1} p(r_x)}{A(r_x)} dr_x \int_{\mathbb{S}^{d-1}} Y_{k,j}(\hat{\mathbf{x}}) Y_{k',j'}(\hat{\mathbf{x}}) d\hat{\mathbf{x}} = \delta_{k,k'}\delta_{j,j'},$$

where the rightmost equality is due to the orthogonality of the spherical harmonics and by setting

$$a^2 = \left(\int_0^{\infty} \frac{r_x^{d+1} p(r_x)}{A(r_x)} dr_x \right)^{-1}.$$

Clearly this integral is positive, and the conditions of the theorem guarantee that it is finite.

2. By the conditions of the theorem we can write

$$\boldsymbol{k}(\mathbf{x},\mathbf{z}) = r_x r_z \hat{\boldsymbol{k}}_0(\hat{\mathbf{x}}^T\hat{\mathbf{z}}) + \hat{\boldsymbol{k}}_1(\hat{\mathbf{x}}^T\hat{\mathbf{z}}),$$

where $\hat{\mathbf{x}}, \hat{\mathbf{z}} \in \mathbb{S}^{d-1}$. On the hypersphere the spherical harmonics are the eigenfunctions of $\boldsymbol{k}_0$ and $\boldsymbol{k}_1$. Denote their eigenvalues respectively by λ_k and μ_k, so that

$$\int_{\mathbb{S}^{d-1}} \boldsymbol{k}_0(\hat{\mathbf{x}}^T\hat{\mathbf{z}}) \bar{Y}_k(\hat{\mathbf{z}}) d\hat{\mathbf{z}} = \lambda_k \bar{Y}_k(\hat{\mathbf{x}}) \tag{D.16}$$

$$\int_{\mathbb{S}^{d-1}} \boldsymbol{k}_1(\hat{\mathbf{x}}^T\hat{\mathbf{z}}) \bar{Y}_k(\hat{\mathbf{z}}) d\hat{\mathbf{z}} = \mu_k \bar{Y}_k(\hat{\mathbf{x}}), \tag{D.17}$$

where $\bar{Y}_k(\hat{\mathbf{x}})$ denote the zonal spherical harmonics. We next show that the space spanned by the functions $r_x\bar{Y}_k(\mathbf{x})$ and $\bar{Y}_k(\mathbf{x})$ is fixed under the following integral transform

$$\int_{\mathbb{R}^d} \boldsymbol{k}(\mathbf{x},\mathbf{z})(\alpha r_z+\beta)\bar{Y}_k(\hat{\mathbf{z}})\bar{p}(\mathbf{z})d\mathbf{z} = (ar_x+b)\bar{Y}_k(\hat{\mathbf{x}}), \tag{D.18}$$

$\alpha, \beta, a, b \in \mathbb{R}$ are constants. The left hand side can be written as the application of an integral operator $T(\mathbf{x},\mathbf{z})$ to a function $\Phi^k_{\alpha,\beta}(\mathbf{z}) = (\alpha r_z+\beta)\bar{Y}_k(\hat{\mathbf{z}})$. Expressing this operator application in spherical coordinates yields

$$T(\mathbf{x},\mathbf{z})\Phi^k_{\alpha,\beta}(\mathbf{z}) = \int_0^\infty \frac{p(r_z)r_z^{d-1}}{A(r_z)}dr_z \int_{\hat{\mathbf{z}}\in\mathbb{S}^{d-1}} \left(r_xr_z\boldsymbol{k}_0(\hat{\mathbf{x}}^T\hat{\mathbf{z}}) + \boldsymbol{k}_1(\hat{\mathbf{x}}^T\hat{\mathbf{z}})\right)(\alpha r_z+\beta)\bar{Y}_k(\hat{\mathbf{z}})d\hat{\mathbf{z}}.$$

We use (D.16) and (D.17) to substitute for the inner integral, obtaining

$$T(\mathbf{x},\mathbf{z})\Phi^k_{\alpha,\beta}(\mathbf{z}) = \int_0^\infty \frac{p(r_z)r_z^{d-1}}{A(r_z)}(\lambda_k r_x r_z+\mu_k)(\alpha r_z+\beta)\bar{Y}_k(\hat{\mathbf{x}})dr_z.$$

Together with (D.18), this can be written as

$$T(\mathbf{x},\mathbf{z})\Phi_{\alpha,\beta}(\mathbf{z}) = \Phi_{a,b}(\mathbf{x}),$$

where

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \lambda_k & 0 \\ 0 & \mu_k \end{pmatrix} \begin{pmatrix} M_2 & M_1 \\ M_1 & M_0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

where $M_q = \int_0^\infty \frac{r_z^{q+d-1}p(r_z)}{A(r_z)}dr_z$, $0 \le q \le 2$. By the conditions of the theorem these moments are finite. This proves that the space spanned by $\{r_x \bar{Y}(\hat{\mathbf{x}}), \bar{Y}(\hat{\mathbf{x}})\}$ is fixed under $T(\mathbf{x}, \mathbf{z})$, and therefore the eigenfunctions of $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}(\mathbf{x}, \mathbf{z})$ take the form $(\bar{a} r_x + \bar{b})\bar{Y}(\hat{\mathbf{x}})$ for some constants $\bar{a}, \bar{b}$.

$\square$

The implication of Theorem 14 is that the eigenvectors of $\boldsymbol{k}^{\mathrm{FC}_0(\mathrm{L})}$ are the spherical harmonic functions, scaled by the norm of their arguments. With bias, $\boldsymbol{k}^{\mathrm{FC}_\beta(\mathrm{L})}$ has up to $2N(d,k)$ eigenfunctions for every frequency k, of the general form $(a r_x + b) Y_{k,j}(\hat{\mathbf{x}})$ where a, b are constants that differ from one eigenfunction to the next.

D.5 Experimental Details

D.5.1 The UCI dataSet

In this section, we provide experimental details for the UCI dataset. We use precisely the same pre-processed datasets, and follow the same performance comparison protocol as in [129].

NTK Specifications We reproduced the results of [129] using the publicly available code[1], and followed the same protocol as in [129]. The total number of kernels evaluated in [129] are 15 and the SVM cost value parameter $\mathbf{C}$ is tuned from 10^{-2} to 10^4 by powers of 10. Hence, the total number of hyper-parameter combinations searched using cross-validation is 105 (15×7).

Exponential Kernels Specifications For the Laplace and Gaussian kernels, we searched for 10 kernel width values $(1/c)$ from $2^{-2} \times \nu$ to ν in the log space with base 2, where ν is chosen heuristically as the median of pairwise l_2 distances between data points (known as the *median*

[1]`https://github.com/LeoYu/neural-tangent-kernel-UCI`

trick [174]). So, the total number of kernel evaluations is 10. For γ-exponential, we searched through 5 equally spaced values of γ from 0.5 to 2. Since we wanted to keep the number of the kernel evaluations the same as for NTK in [129], we searched through only three kernel bandwidth values $(1/c)$ which are 1, ν and #features (default value in the **sklearn** package[2]). So, the total number of kernel evaluations is 15 (5×3).

For a fair comparison with [129], we swept the same range of SVM cost value parameter $\mathbf{C}$ as in [129], i.e., from 10^{-2} to 10^{4} by powers of 10. Hence, the total number of hyper-parameter search using cross-validation is 70 (10×7) for Laplace and 105 (15×7) for γ-exponential which is the same as for NTK in [129].

D.5.2 Large scale datasets

We used the experimental setup mentioned in [164] and the publicly available code [3]. [164] solves kernel ridge regression (KRR [175]) using the FALKON algorithm, which solves the following linear system

$$(K_{nn} + \lambda n I)\,\alpha = \hat{\mathbf{y}},$$

where K is an $n \times n$ kernel matrix defined by $(K)_{ij} = K(x_i, x_j)$, $\hat{\mathbf{y}} = (y_1, \dots y_n)^T$, and λ is the regularization parameter. Refer to [164] for more details.

In Table D.1, we provide the hyper parameters chosen with cross validation.

[2]`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.rbf_kernel.html`

[3]`https://github.com/LCSL/FALKON_paper`

	MillionSongs [165]	SUSY [166]	HIGGS [166]
H-γ-exp.	$\gamma = 1.4, \sigma = 5, \lambda = 1e^{-6}$	$\gamma = 1.8, \sigma = 5, \lambda = 1e^{-7}$	$\gamma = 1.6, \sigma = 8, \lambda = 1e^{-8}$
H-Laplace	$\sigma = 3, \lambda = 1e^{-6}$	$\sigma = 4, \lambda = 1e^{-7}$	$\sigma = 8, \lambda = 1e^{-8}$
NTK	$L = 9, \lambda = 1e^{-9}$	$L = 3, \lambda = 1e^{-8}$	$L = 3, \lambda = 1e^{-6}$
H-Gaussian	$\sigma = 8, \lambda = 1e^{-6}$	$\sigma = 3, \lambda = 1e^{-7}$	$\sigma = 8, \lambda = 1e^{-8}$

Table D.1: Hyper-parameters chosen with cross validation for the different kernels.

D.5.3 C-Exp: Convolutional Exponential Kernels

Let $\mathbf{x} = (x_1, ..., x_d)^T$ and $\mathbf{z} = (z_1, ..., z_d)^T$ denote two vectorized images. Let P denote a window function (we used 3×3 windows). Our hierarchical exponential kernels are defined by $\bar{\Theta}(\mathbf{x}, \mathbf{z})$ as follows:

$$\begin{aligned}
\Theta_{ij}^{[0]}(\mathbf{x}, \mathbf{z}) &= x_i z_j \\
s_{ij}^{[h]}(\mathbf{x}, \mathbf{z}) &= \sum_{m \in P} \Theta^{[h]}(x_{i+m}, z_{j+m}) + \beta^2 \\
\Theta_{ij}^{[h+1]}(\mathbf{x}, \mathbf{z}) &= K(s_{ij}^{[h]}(\mathbf{x}, \mathbf{z}), s_{ii}^{[h]}(\mathbf{x}, \mathbf{x}), s_{jj}^{[h]}(\mathbf{z}, \mathbf{z})) \\
\bar{\Theta}(\mathbf{x}, \mathbf{z}) &= \sum_i \Theta_{ii}^{[L]}(\mathbf{x}, \mathbf{z})
\end{aligned}$$

where $\beta \geq 0$ denotes the bias and the last step is analogous to a fully connected layer in networks, and we set

$$K(s_{ij}, s_{ii}, s_{jj}) = \sqrt{s_{ii} s_{jj}}\, \boldsymbol{k}\left(\frac{s_{ij}}{\sqrt{s_{ii} s_{jj}}}\right)$$

where $\boldsymbol{k}$ can be any kernel defined on the sphere. In the experiments we applied this scheme to the three exponential kernels, Laplace, Gaussian and γ-exponential.

Technical details We used the following four kernels:

CNTK [124] $L = 6, \beta = 3$.

C-Exp Laplace. $L = 3, \beta = 3$, $\boldsymbol{k}(\mathbf{x}^T\mathbf{z}) = a + be^{-c\sqrt{2-2\mathbf{x}^T\mathbf{z}}}$ with $a = -11.491, b = 12.606, c =$

0.048.

C-Exp $\gamma-$exponential. $L = 8, \beta = 3$, $\boldsymbol{k}(\mathbf{x}^T\mathbf{z}) = a + be^{-c(2-2\mathbf{x}^T\mathbf{z})^{\gamma/2}}$ with $a = -0.276, b = 1.236, c = 0.424, \gamma = 1.888$.

C-Exp Gaussian. $L = 12, \beta = 3$, $\boldsymbol{k}(\mathbf{x}^T\mathbf{z}) = a + be^{-c(2-2\mathbf{x}^T\mathbf{z})}$ with $a = -0.22, b = 1.166, c = 0.435$.

We set β in these experiments with cross validation in $\{1, ..., 10\}$. For each kernel $\boldsymbol{k}$ above, the parameters a, b, c and γ were chosen using non-linear least squares optimization with the objective $\sum_{u \in U}(\boldsymbol{k}(u) - \boldsymbol{k}^{\mathrm{FC}_\beta(2)}(u))^2$, where $\boldsymbol{k}^{\mathrm{FC}_\beta(2)}$ is the NTK for a two-layer network defined in (D.5) with bias $\beta = 1$, and the set U included (inner products between) pairs of normalized $3 \times 3 \times 3$ patches drawn uniformly from the CIFAR images. The number of layers L is chosen by cross validation.

For the training phase we used 1-hot vectors from which we subtracted 0.1, as in [176]. For the classification phase, as in [141], we normalized the kernel matrices such that all the diagonal elements are ones. To avoid ill conditioned kernel matrices we applied ridge regression with a regularization factor of $\lambda = 5 \cdot 10^{-5}$. Finally, to reduce overall running times, we parallelized the kernel computations on NVIDIA Tesla V100 GPUs.

Appendix E: Biconvex Relaxation for Semidefinite Programming in Computer Vision

Semidefinite programming (SDP) is an indispensable tool in computer vision, but general-purpose solvers for SDPs are often too slow and memory intensive for large-scale problems. Our framework, referred to as biconvex relaxation (BCR), transforms an SDP consisting of PSD constraint matrices into a specific biconvex optimization problem, which can then be approximately solved in the original, low-dimensional variable space at low complexity. The resulting problem is solved using an efficient alternating minimization (AM) procedure. Since AM has the potential to get stuck in local minima, we propose a general initialization scheme that enables BCR to start close to a global optimum—this is key for BCR to quickly converge to optimal or near-optimal solutions. We showcase the efficacy of our approach on three applications in computer vision, namely segmentation, co-segmentation, and manifold metric learning. BCR achieves solution quality comparable to state-of-the-art SDP methods with speedups between $4\times$ and $35\times$. This work [177] is in collaboration with Sohil Shah, Carlos D. Castillo, David W. Jacobs, Christoph Studer, and Tom Goldstein.

E.1 Introduction

Optimization problems involving either integer-valued vectors or low-rank matrices are ubiquitous in computer vision. Graph-cut methods for image segmentation, for example, involve optimization problems where integer-valued variables represent region labels [178, 179, 180, 181]. Problems in multi-camera structure from motion [182], manifold embedding [183], and matrix completion [184] all rely on optimization problems involving matrices with low rank constraints. Since these constraints are non-convex, the design of efficient algorithms that find globally optimal solutions is a difficult task.

For a wide range of applications [183, 185, 186, 187, 188, 189], non-convex constraints can be handled by *semidefinite relaxation* (SDR) [185]. In this approach, a non-convex optimization problem involving a vector of unknowns is "lifted" to a higher dimensional convex problem that involves a positive semidefinite (PSD) matrix, which then enables one to solve a SDP [190]. While SDR delivers state-of-the-art performance in a wide range of applications [180, 181, 183, 184, 185, 191], the approach significantly increases the dimensionality of the original optimization problem (i.e., replacing a vector with a matrix), which typically results in exorbitant computational costs and memory requirements. Nevertheless, SDR leads to SDPs whose global optimal solution can be found using robust numerical methods.

A growing number of computer-vision applications involve high-resolution images (or videos) that require SDPs with a large number of variables. General-purpose (interior point) solvers for SDPs do not scale well to such problem sizes; the worst-case complexity is $O(N^{6.5}\log(1/\varepsilon))$ for an $N \times N$ problem with ε objective error [192]. In imaging applications, N is often proportional to the number of pixels, which is potentially large.

The prohibitive complexity and memory requirements of solving SDPs exactly with a large number of variables has spawned interest in fast, non-convex solvers that avoid lifting. For example, recent progress in phase retrieval by Netrapalli *et al.* [193] and Candès *et al.* [194] has shown that non-convex optimization methods provably achieve solution quality comparable to exact SDR-based methods with significantly lower complexity. These methods operate on the original dimensions of the (un-lifted) problem, which enables their use on high-dimensional problems. Another prominent example is max-norm regularization by Lee *et al.* [195], which was proposed for solving high-dimensional matrix-completion problems and to approximately perform max-cut clustering. This method was shown to outperform exact SDR-based methods in terms of computational complexity, while delivering acceptable solution quality. While both of these examples outperform classical SDP-based methods, they are limited to very specific problem types, and cannot handle more complex SDPs that typically appear in computer vision.

E.1.1 Contributions

We introduce a novel framework for approximately solving SDPs with positive semi-definite constraint matrices in a computationally efficient manner and with small memory footprint. Our proposed *bi-convex* relaxation (BCR), transforms an SDP into a biconvex optimization problem, which can then be solved in the original, low-dimensional variable space at low complexity. The resulting biconvex problem is solved using a computationally-efficient AM procedure. Since AM is prone to get stuck in local minima, we propose an initialization scheme that enables BCR to start close to the global optimum of the original SDP—this initialization is key for our algorithm to quickly converge to an optimal or near-optimal solution. We showcase the effectiveness of the BCR framework by comparing to highly-specialized SDP solvers for a selected set of problems in computer vision involving image segmentation, co-segmentation, and metric learning on manifolds. Our results demonstrate that BCR enables high-quality results while achieving speedups ranging from $4\times$ to $35\times$ over state-of-the-art competitor methods [196, 197, 198, 199, 200] for the studied applications.

E.2 Background and Relevant Prior Art

We now briefly review semidefinite programs (SDPs) and discuss prior work on fast, approximate solvers for SDPs in computer vision and related applications.

E.2.1 Semidefinite Programs (SDPs)

SDPs find use in a large and growing number of fields, including computer vision, machine learning, signal and image processing, statistics, communications, and control [190]. SDPs can be written in the following general form:

$$\begin{aligned} \underset{\mathbf{Y}\in\mathcal{S}^{+}_{N\times N}}{\text{minimize}} \quad & \langle \mathbf{C}, \mathbf{Y}\rangle \\ \text{subject to} \quad & \langle \mathbf{A}_i, \mathbf{Y}\rangle = b_i, \quad \forall i \in \mathcal{E}, \\ & \langle \mathbf{A}_j, \mathbf{Y}\rangle \leq b_j, \quad \forall j \in \mathcal{B}, \end{aligned} \tag{E.1}$$

where $S^{+}_{N\times N}$ represents the set of $N \times N$ symmetric positive semidefinite matrices, and $\langle \mathbf{C}, \mathbf{Y}\rangle = \mathrm{tr}(\mathbf{C}^T\mathbf{Y})$ is the matrix inner product. The sets $\mathcal{E}$ and $\mathcal{B}$ contain the indices associated with the equality and inequality constraints, respectively; $\mathbf{A}_i$ and $\mathbf{A}_j$ are symmetric matrices of appropriate dimensions.

The key advantages of SDPs are that (i) they enable the transformation of certain non-convex constraints into convex constraints via semidefinite relaxation (SDR) [185] and (ii) the resulting problems often come with strong theoretical guarantees.

In computer vision, a large number of problems can be cast as SDPs of the general form (E.1). For example, [183] formulates image manifold learning as an SDP, [189] uses an SDP to enforce a non-negative lighting constraint when recovering scene lighting and object albedos, [201] uses an SDP for graph matching, [182] proposes an SDP that recovers the orientation of multiple cameras from point correspondences and essential matrices, and [184] uses low-rank SDPs to solve matrix-completion problems that arise in structure-from-motion and photometric stereo.

E.2.2 SDR for Binary-Valued Quadratic Problems

Semidefinite relaxation is commonly used to solve binary-valued labeling problems. For such problems, a set of variables take on binary values while minimizing a quadratic cost function that depends on the assignment of pairs of variables. Such labeling problems typically arise from Markov random fields (MRFs) for which many solution methods exist [202]. Spectral methods, e.g., [178], are often used to solve such binary-valued quadratic problems (BQPs)—the references [179, 180] used SDR inspired by the work of [181] that provides a generalized SDR for the max-cut problem. BQP problems have wide applicability to computer vision problems, such as segmentation and perceptual organization [179, 196, 203], semantic segmentation [204], matching [180, 205], surface reconstruction including photometric stereo and shape from defocus [188], and image restoration [206].

BQPs can be solved by lifting the binary-valued label vector $\mathbf{b} \in \{\pm 1\}^N$ to an N^2-dimensional matrix space by forming the PSD matrix $\mathbf{B} = \mathbf{b}\mathbf{b}^T$, whose non-convex rank-1 constraint is relaxed to PSD matrices $\mathbf{B} \in S^+_{N\times N}$ with an all-ones diagonal [185]. The goal is then to solve a SDP for $\mathbf{B}$ in the hope that the resulting matrix has rank 1; if $\mathbf{B}$ has higher rank, an approximate solution must be extracted which can either be obtained from the leading eigenvector or via randomization methods [185, 207].

E.2.3 Specialized Solvers for SDPs

General-purpose solvers for SDPs, such as SeDuMi [208] or SDPT3 [209], rely on interior point methods with high computational complexity and memory requirements. Hence, their use is restricted to low-dimensional problems. For problems in computer vision, where the number of variables can become comparable to the number of pixels in an image, more efficient algorithms are necessary. A handful of special-purpose algorithms have been proposed to solve specific problem types arising in computer vision. These algorithms fit into two classes: (i) convex algorithms that solve the original SDP by exploiting problem structure and (ii) non-convex methods that avoid lifting.

For certain problems, one can exactly solve SDPs with much lower complexity than interior point schemes, especially for BQP problems in computer vision. Ecker *et al.* [188] deployed a number of heuristics to speed up the Goemans-Williamson SDR [181] for surface reconstruction. Olsson *et al.* [206] proposed a spectral subgradient method to solve BQP problems that include a linear term, but are unable to handle inequality constraints. A particularly popular approach is the SDCut algorithms of Wang *et al.* [196]. This method solves BQP for some types of segmentation problems using dual gradient descent. SDCut leads to a similar relaxation as for BQP problems, but enables significantly lower complexity for graph cutting and its variants. To the best of our knowledge, the method by Wang *et al.* [196] yields state-of-the-art performance—nevertheless, our proposed method is at least an order of magnitude faster, as shown in Section E.4.

Another algorithm class contains non-convex approximation methods that avoid lifting altogether. Since these methods work with low-dimensional unknowns, they are potentially more efficient than lifted methods. Simple examples include the Wiberg method [210] for low-rank matrix approximation, which uses Newton-type iterations to minimize a non-convex objective. A number of methods have been proposed for SDPs where the objective function is simply the trace-norm of $\mathbf{Y}$ (i.e., problem (E.1) with $\mathbf{C} = \mathbf{I}$) and without inequality constraints. Approaches include replacing the trace norm with the max-norm [195], or using the so-called Wirtinger flow to solve phase-retrieval problems [194]. One of the earliest approaches for non-convex methods are due to Burer and Montiero [211], who propose an augmented Lagrangian method. While this method is able to handle arbitrary objective functions, it does not naturally support inequality constraints (without introducing auxiliary slack variables). Furthermore, this approach uses convex methods for which convergence is not well understood and is sensitive to the initialization value.

While most of the above-mentioned methods provide best-in-class performance at low computational complexity, they are limited to very specific problems and cannot be generalized to other, more general SDPs.

E.3 Biconvex Relaxation (BCR) Framework

We now present the proposed *biconvex relaxation (BCR)* framework. We then propose an alternating minimization procedure and a suitable initialization method.

E.3.1 Biconvex Relaxation

Rather than solving the general SDP (E.1) directly, we exploit the following key fact: any matrix $\mathbf{Y}$ is symmetric positive semidefinite if and only if it has an expansion of the form $\mathbf{Y} = \mathbf{X}\mathbf{X}^T$. By substituting the factorization $\mathbf{Y} = \mathbf{X}\mathbf{X}^T$ into (E.1), we are able to remove the semidefinite constraint and arrive at the following problem:

$$\begin{aligned}
\underset{\mathbf{X}\in\mathbb{R}^{N\times r}}{\text{minimize}} \quad & \text{tr}(\mathbf{X}^T\mathbf{C}\mathbf{X}) \\
\text{subject to} \quad & \text{tr}(\mathbf{X}^T\mathbf{A}_i\mathbf{X}) = b_i, \quad \forall i \in \mathcal{E}, \\
& \text{tr}(\mathbf{X}^T\mathbf{A}_j\mathbf{X}) \leq b_j, \quad \forall j \in \mathcal{B},
\end{aligned} \tag{E.2}$$

where $r = \text{rank}(\mathbf{Y})$.[1] Note that any symmetric semi-definite matrix $\mathbf{A}$ has a (possibly complex-valued) square root $\mathbf{L}$ of the form $\mathbf{A} = \mathbf{L}^T\mathbf{L}$. Furthermore, we have $\text{tr}(\mathbf{X}^T\mathbf{A}\mathbf{X}) = \text{tr}(\mathbf{X}^T\mathbf{L}^T\mathbf{L}\mathbf{X}) = \|\mathbf{L}\mathbf{X}\|_F^2$, where $\|\cdot\|_F$ is the Frobenius (matrix) norm. This formulation enables us to rewrite (E.2) as follows:

$$\begin{aligned}
\underset{\mathbf{X}\in\mathbb{R}^{N\times r}}{\text{minimize}} \quad & \text{tr}(\mathbf{X}^T\mathbf{C}\mathbf{X}) \\
\text{subject to} \quad & \mathbf{Q}_i = \mathbf{L}_i\mathbf{X}, \quad \|\mathbf{Q}_i\|_F^2 = b_i, \quad \forall i \in \mathcal{E}, \\
& \mathbf{Q}_j = \mathbf{L}_j\mathbf{X}, \quad \|\mathbf{Q}_j\|_F^2 \leq b_j, \quad \forall j \in \mathcal{B}.
\end{aligned} \tag{E.3}$$

If the matrices $\{\mathbf{A}_i\}$, $\{\mathbf{A}_j\}$, and $\mathbf{C}$ are themselves PSDs, then the objective function in (E.3) is convex and quadratic, and the inequality constraints in (E.3) are convex—non-convexity of the problem is only caused by the equality constraints. The core idea of BCR explained next is to relax these equality constraints. Here, we assume that the factors of these matrices are easily obtained from the underlying problem structure. For some applications, where these factors are not readily available this could be a computational burden (worst case $\mathcal{O}(N^3)$) rather than an asset.

In the formulation (E.3), we have lost convexity. Nevertheless, whenever $r < N$, we achieved a (potentially large) dimensionality reduction compared to the original SDP (E.1). We now relax (E.3) in a form that is biconvex, i.e., convex with respect to a group of variables when the remaining

[1]Straightforward extensions of our approach allow us to handle constraints of the form $\text{tr}(\mathbf{X}^T\mathbf{A}_k\mathbf{X}) \geq b_k, \forall k \in \mathcal{A}$, as well as complex-valued matrices and vectors.

variables are held constant. By relaxing the convex problem in biconvex form, we retain many advantages of the convex formulation while maintaining low dimensionality and speed. In particular, we propose to approximate (E.3) with the following *biconvex relaxation (BCR)*:

$$\begin{aligned}
&\underset{\mathbf{X},\mathbf{Q}_i,i\in\{\mathcal{B}\cup\mathcal{E}\}}{\text{minimize}} \; \operatorname{tr}(\mathbf{X}^T\mathbf{C}\mathbf{X}) + \frac{\alpha}{2}\sum_{i\in\{\mathcal{E}\cup\mathcal{B}\}}\|\mathbf{Q}_i - \mathbf{L}_i\mathbf{X}\|_F^2 - \frac{\beta}{2}\sum_{j\in\mathcal{E}}\|\mathbf{Q}_j\|_F^2 \\
&\text{subject to} \quad \|\mathbf{Q}_i\|_F^2 \le b_i, \quad \forall i \in \{\mathcal{B}\cup\mathcal{E}\},
\end{aligned} \tag{E.4}$$

where $\alpha > \beta > 0$ are relaxation parameters (discussed in detail below). In this BCR formulation, we relaxed the equality constraints $\|\mathbf{Q}_i\|_F^2 = b_i$, $\forall i \in \mathcal{E}$, to inequality constraints $\|\mathbf{Q}_i\|_F^2 \le b_i$, $\forall i \in \mathcal{E}$, and added negative quadratic penalty functions $-\frac{\beta}{2}\|\mathbf{Q}_i\|$, $\forall i \in \mathcal{E}$, to the objective function. These quadratic penalties attempt to force the inequality constraints in $\mathcal{E}$ to be satisfied exactly. We also replaced the constraints $\mathbf{Q}_i = \mathbf{L}_i\mathbf{X}$ and $\mathbf{Q}_j = \mathbf{L}_j\mathbf{X}$ by quadratic penalty functions in the objective function.

The relaxation parameters are chosen by freezing the ratio α/β to 2, and following a simple, principled way of setting β. Unless stated otherwise, we set β to match the curvature of the penalty term with the curvature of the objective i.e., $\beta = \|\mathbf{C}\|_2$, so that the resulting bi-convex problem is well-conditioned.

Our BCR formulation (E.4) has some important properties. First, if $\mathbf{C} \in \mathcal{S}^+_{N\times N}$ then the problem is biconvex, i.e., convex with respect to $\mathbf{X}$ when the $\{\mathbf{Q}_i\}$ are held constant, and vice versa. Furthermore, consider the case of solving a constraint feasibility problem (i.e., problem (E.1) with $\mathbf{C} = \mathbf{0}$). When $\mathbf{Y} = \mathbf{X}\mathbf{X}^T$ is a solution to (E.1) with $\mathbf{C} = \mathbf{0}$, the problem (E.4) assumes objective value $-\frac{\beta}{2}\sum_j b_j$, which is the global minimizer of the BCR formulation (E.4). Likewise, it is easy to see that any global minimizer of (E.4) with objective value $-\frac{\beta}{2}\sum_j b_j$ must be a solution to the original problem (E.1).

E.3.2 Alternating Minimization (AM) Algorithm

One of the key benefits of biconvexity is that (E.4) can be globally minimized with respect to $\mathbf{Q}$ or $\mathbf{X}$. Hence, it is natural to compute approximate solutions to (E.4) via alternating minimization. Note the convergence of AM for biconvex problems is well understood [212, 213]. The two stages of the proposed method for BCR are detailed next.

Stage 1: Minimize with respect to $\{\mathbf{Q}_i\}$. The BCR objective in (E.4) is quadratic in $\{\mathbf{Q}_i\}$ with no dependence between matrices. Consequently, the optimal value of $\mathbf{Q}_i$ can be found by minimizing the quadratic objective, and then reprojecting back into a unit Frobenius-norm ball of radius $\sqrt{b_i}$. The minimizer of the quadratic objective is given by $\frac{\alpha}{\alpha-\beta_i}\mathbf{L}_i\mathbf{X}$, where $\beta_i = 0$ if $i \in \mathcal{B}$ and $\beta_i = \beta$ if $i \in \mathcal{E}$. The projection onto the unit ball then leads to the following *expansion–reprojection* update:

$$\mathbf{Q}_i \leftarrow \frac{\mathbf{L}_i\mathbf{X}}{\|\mathbf{L}_i\mathbf{X}\|_F} \min\left\{\sqrt{b_i}, \frac{\alpha}{\alpha-\beta_i}\|\mathbf{L}_i\mathbf{X}\|_F\right\}. \tag{E.5}$$

Intuitively, this expansion–reprojection update causes the matrix $\mathbf{Q}_i$ to expand if $i \in \mathcal{E}$, thus encouraging it to satisfy the relaxed constraints in (E.4) with equality.

Stage 2: Minimize with respect to $\mathbf{X}$. This stage solves the least-squares problem:

$$\mathbf{X} \leftarrow \underset{\mathbf{X}\in\mathbb{R}^{N\times r}}{\arg\min}\ \operatorname{tr}(\mathbf{X}^T\mathbf{C}\mathbf{X}) + \frac{\alpha}{2}\sum_{i\in\{\mathcal{E}\cup\mathcal{B}\}}\|\mathbf{Q}_i - \mathbf{L}_i\mathbf{X}\|_F^2. \tag{E.6}$$

Algorithm 4 AM for Biconvex Relaxation

```
1: inputs: $\mathbf{C}$, $\{\mathbf{L}_i\}$, $b_i$, $\alpha$, and $\beta$, output: $\mathbf{X}$
2: Compute an initializer for $\mathbf{X}$ as in Section E.3.3
3: Precompute $\mathbf{M} = \left(\mathbf{C} + \alpha \sum_{i\in\{E\cup\mathcal{B}\}} \mathbf{L}_i^T\mathbf{L}_i\right)^{-1}$
4: while not converged do
5:     $\mathbf{Q}_i \leftarrow \frac{\mathbf{L}_i\mathbf{X}}{\|\mathbf{L}_i\mathbf{X}\|_F} \min\left\{\sqrt{b_i}, \frac{\alpha}{\alpha-\beta_i}\|\mathbf{L}_i\mathbf{X}\|_F\right\}$
6:     $\mathbf{X} \leftarrow \mathbf{M}\left(\sum_{i\in\{\mathcal{E}\cup\mathcal{B}\}} \mathbf{L}_i^T\mathbf{Q}_i\right)$,
7: end while
```

The optimality conditions for this problem are linear equations, and the solution is

$$\mathbf{X} \leftarrow \left(\mathbf{C} + \alpha \sum_{i\in\{\mathcal{E}\cup\mathcal{B}\}} \mathbf{L}_i^T\mathbf{L}_i\right)^{-1} \left(\sum_{i\in\{\mathcal{E}\cup\mathcal{B}\}} \mathbf{L}_i^T\mathbf{Q}_i\right), \tag{E.7}$$

where the matrix inverse (one-time computation) may be replaced by a pseudo-inverse if necessary. Alternatively, one may perform a simple gradient-descent step with a suitable step size, which avoids the inversion of a potentially large-dimensional matrix.

The resulting AM algorithm for the proposed BCR (E.4) is summarized in Algorithm 4.

E.3.3 Initialization

The problem (E.4) is biconvex and hence, a global minimizer can be found with respect to either $\{\mathbf{Q}_i\}$ or $\mathbf{X}$, although a global minimizer of the joint problem is not guaranteed. We hope to find a global minimizer at low complexity using the AM method, but in practice AM may get trapped in local minima, especially if the variables have been initialized poorly. We now propose a principled method for computing an initializer for $\mathbf{X}$ that is often close to the global optimum of the BCR problem—our initializer is key for the success of the proposed AM procedure and enables fast convergence.

The papers [193, 194] have considered optimization problems that arise in phase retrieval where

$\mathcal{B} = \varnothing$ (i.e., there are only equality constraints), $\mathbf{C} = \mathbf{I}$ being the identity, and $\mathbf{Y}$ being rank one. For such problems, the objective of (E.1) reduces to $\operatorname{tr}(\mathbf{Y})$. By setting $\mathbf{Y} = \mathbf{x}\mathbf{x}^T$, we obtain the following formulation:

$$\underset{\mathbf{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\mathbf{x}\|_2^2 \quad \text{subject to} \ \ \mathbf{q}_i = \mathbf{L}_i\mathbf{x}, \quad \|\mathbf{q}_i\|_2^2 = b_i, \quad \forall i \in \mathcal{E}. \tag{E.8}$$

Netrapali *et al.* [193] proposed an iterative algorithm for solving (E.8), which has been initialized by the following strategy. Define

$$\mathbf{Z} = \frac{1}{|\mathcal{E}|}\sum_{i\in\mathcal{E}} b_i \mathbf{L}_i^T \mathbf{L}_i. \tag{E.9}$$

Let $\mathbf{v}$ be the leading eigenvector of $\mathbf{Z}$ and λ the leading eigenvalue. Then $\mathbf{x} = \lambda\mathbf{v}$ is an accurate approximation to the true solution of (E.8). In fact, if the matrices $\mathbf{L}_i$ are sampled from a random normal distribution, then it was shown in [193, 194] that $\mathbb{E}\|\mathbf{x}^\star - \lambda\mathbf{x}\|_2^2 \to 0$ (in expectation) as $|\mathcal{E}| \to \infty$, where $\mathbf{x}^\star$ is the true solution to (E.8).

We are interested in a good initializer for the general problem in (E.3) where $\mathbf{X}$ can be rank one or higher. We focus on problems with equality constraints only—note that one can use slack variables to convert a problem with inequality constraints into the same form [190]. Given that $\mathbf{C}$ is a symmetric positive definite matrix, it can be decomposed into $\mathbf{C} = \mathbf{U}^T\mathbf{U}$. By the change of variables $\widetilde{\mathbf{X}} = \mathbf{U}\mathbf{X}$, we can rewrite (E.1) as follows:

$$\underset{\mathbf{X}\in\mathbb{R}^{N\times r}}{\text{minimize}} \ \|\widetilde{\mathbf{X}}\|_F^2 \quad \text{subject to} \ \ \langle\widetilde{\mathbf{A}}_i, \widetilde{\mathbf{X}}\widetilde{\mathbf{X}}^T\rangle = b_i, \quad \forall i \in \mathcal{E}, \tag{E.10}$$

where $\widetilde{\mathbf{A}}_i = \mathbf{U}^{-T}\mathbf{A}_i\mathbf{U}^{-1}$, and we omitted the inequality constraints. To initialize the proposed AM procedure in Algorithm 4, we make the change of variables $\widetilde{\mathbf{X}} = \mathbf{U}\mathbf{X}$ to transform the BCR formulation into the form of (E.10). Analogously to the initialization procedure in [193] for phase retrieval, we then compute an initializer $\widetilde{\mathbf{X}}_0$ using the leading r eigenvectors of $\mathbf{Z}$ scaled by the leading eigenvalue λ. Finally, we calculate the initializer for the original problem by reversing the change of variables as $\mathbf{X}_0 = \mathbf{U}^{-1}\widetilde{\mathbf{X}}_0$. For most problems the initialization time is a small fraction of the total runtime.

E.3.4 Advantages of Biconvex Relaxation

The proposed framework has numerous advantages over other non-convex methods. First and foremost, BCR can be applied to general SDPs. Specialized methods, such as Wirtinger flow [194] for phase retrieval and the Wiberg method [210] for low-rank approximation are computationally efficient, but restricted to specific problem types. Similarly, the max-norm method [195] is limited to solving trace-norm-regularized SDPs. The method of Burer and Montiero [211] is less specialized, but does not naturally support inequality constraints. Furthermore, since BCR problems are biconvex, one can use numerical solvers with guaranteed convergence. Convergence is guaranteed not only for the proposed AM least-squares method in Algorithm 4 (for which the objective decreases monotonically), but also for a broad range of gradient-descent schemes suitable to find solutions to biconvex problems [214]. In contrast, the method in [211] uses augmented Lagrangian methods with non-linear constraints for which convergence is not guaranteed.

E.4 Benchmark Problems

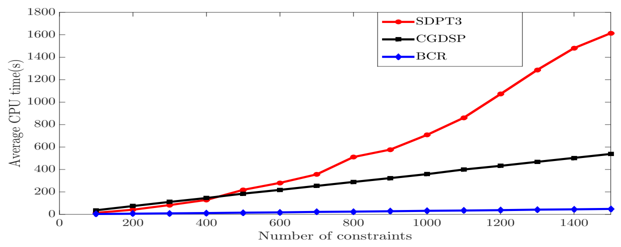
We now evaluate our solver using both synthetic and real-world data. We begin with a brief comparison showing that biconvex solvers outperform both interior-point methods for general

SDPs and also state-of-the-art low-rank solvers. Of course, specialized solvers for specific problem forms achieve superior performance to classical interior point schemes. For this reason, we evaluate our proposed method on three important computer vision applications, i.e., segmentation, co-segmentation, and manifold metric learning, using public datasets, and we compare our results to state-of-the-art methods. These applications are ideal because (i) they involve large scale SDPs and (ii) customized solvers are available that exploit problem structure to solve these problems efficiently. Hence, we can compare our BCR framework to powerful and optimized solvers.

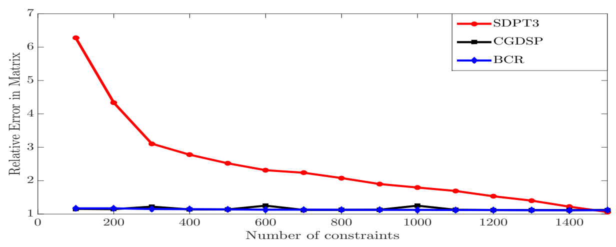
E.4.1 General-Form Problems

We briefly demonstrate that BCR performs well on general SDPs by comparing to the widely used SDP solver, SDPT3 [209] and the state-of-the-art, low-rank SDP solver CGDSP [215]. Note that SDPT3 uses an interior point approach to solve the convex problem in (E.1) whereas the CGDSP solver uses gradient-descent to solve a non-convex formulation. For fairness, we initialize both algorithms using the proposed initializer and the gradient descent step in CGDSP was implemented using various acceleration techniques [216]. Since CGDSP cannot handle inequality constraints we restrict our comparison to equality constraints only.

Experiments: We randomly generate a 256×256 rank-3 data matrix of the form $\mathbf{Y}_{\text{true}} = \mathbf{x}_1\mathbf{x}_1^T + \mathbf{x}_2\mathbf{x}_2^T + \mathbf{x}_3\mathbf{x}_3^T$, where $\{\mathbf{x}_i\}$ are standard normal vectors. We generate a standard normal matrix $\mathbf{L}$ and compute $\mathbf{C} = \mathbf{L}^T\mathbf{L}$. Gaussian matrices $\mathbf{A}_i \in \mathbb{R}^{250\times 250}$ form equality constraints. We report the relative error in the recovered solution $\mathbf{Y}_{\text{rec}}$ measured as $\|\mathbf{Y}_{\text{rec}} - \mathbf{Y}_{\text{true}}\|/\|\mathbf{Y}_{\text{true}}\|$. Average runtimes for varying numbers of constraints are shown in Figure E.1a, while Figure E.1b plots the average relative error. Figure E.1a shows that our method has the best runtime of all the schemes. Figure



(a) Average solver runtime



(b) Average relative error

Figure E.1: Results on synthetic data for varying number of linear constraints.

E.1b shows convex interior point methods do not recover the correct solution for small numbers of constraints. With few constraints, the full lifted SDP is under-determined, allowing the objective to go to zero. In contrast, the proposed BCR approach is able to enforce an additional rank-3 constraint, which is advantageous when the number of constraints is low.

E.4.2 Image Segmentation

Consider an image of N pixels. Segmentation of foreground and background objects can be accomplished using graph-based approaches, where graph edges encode the similarities between pixel pairs. Such approaches include normalized cut [178] and ratio cut [217]. The graph cut problem can be formulated as an NP-hard integer program [181]

$$\underset{\mathbf{x}\in\{-1,1\}^N}{\text{minimize}} \ \mathbf{x}^T\mathbf{L}\mathbf{x}, \tag{E.11}$$

where $\mathbf{L}$ encodes edge weights and $\mathbf{x}$ contains binary region labels, one for each pixel. This problem can be "lifted" to the equivalent higher dimensional problem

$$\underset{\mathbf{X}\in S^+_{N\times N}}{\text{minimize}} \ \operatorname{tr}(\mathbf{L}^T\mathbf{X}) \quad \text{subject to } \operatorname{diag}(\mathbf{X}) = \mathbf{1}, \ \operatorname{rank}(\mathbf{X}) = 1. \tag{E.12}$$

After dropping the non-convex rank constraint, (E.12) becomes an SDP that is solvable using convex optimization [179, 191, 205]. The SDP approach is computationally intractable if solved using off-the-shelf SDP solvers (such as SDPT3 [209] or other interior point methods). Furthermore, exact solutions cannot be recovered when the solution to the SDP has rank greater than 1. In contrast, BCR is computational efficient for large problems and can easily incorporate rank constraints, leading to efficient spectral clustering.

BCR is also capable of incorporating annotated foreground and background pixel priors [218] using linear equality and inequality constraints. We consider the SDP based segmentation presented in [218], which contains three grouping constraints on the pixels: $(\mathbf{t}_f^T\mathbf{P}\mathbf{x})^2 \geq \kappa\|\mathbf{t}_f^T\mathbf{P}\mathbf{x}\|_1^2$, $(\mathbf{t}_b^T\mathbf{P}\mathbf{x})^2 \geq \kappa\|\mathbf{t}_b^T\mathbf{P}\mathbf{x}\|_1^2$ and $((\mathbf{t}_f - \mathbf{t}_b)^T\mathbf{P}\mathbf{x})^2 \geq \kappa\|(\mathbf{t}_f - \mathbf{t}_b)^T\mathbf{P}\mathbf{x}\|_1^2$, where $\kappa \in [0, 1]$. $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}$ is the normalized pairwise affinity matrix and $\mathbf{t}_f$ and $\mathbf{t}_b$ are indicator variables denoting the foreground and background pixels. These constraints enforce that the segmentation respects the pre-labeled pixels given by the user, and also pushes high similarity pixels to have the same label. The affinity matrix $\mathbf{W}$ is given by

$$W_{i,j} = \begin{cases} \exp\left(-\frac{\|\mathbf{f}_i - \mathbf{f}_j\|_2^2}{\gamma_f^2} - \frac{d(i,j)^2}{\gamma_d^2}\right), & \text{if } d(i,j) < r \\ 0, & \text{otherwise,} \end{cases} \tag{E.13}$$

where $\mathbf{f}_i$ is the color histogram of the ith super-pixel and $d(i, j)$ is the spatial distance between i and j. Considering these constraints and letting $\mathbf{X} = \mathbf{Y}\mathbf{Y}^T$, (E.12) can be written in the form of

(E.2) as follows:

$$\begin{aligned}
\underset{\mathbf{Y}\in\mathbb{R}^{N\times r}}{\text{minimize}} \quad & \operatorname{tr}(\mathbf{Y}^T\mathbf{L}\mathbf{Y}) \\
\text{subject to} \quad & \operatorname{tr}(\mathbf{Y}^T\mathbf{A}_i\mathbf{Y}) = 1, \quad \forall i = 1,\ldots,N \\
& \operatorname{tr}(\mathbf{Y}^T\mathbf{B}_2\mathbf{Y}) \geq \kappa\|\mathbf{t}_f^T\mathbf{P}\mathbf{x}\|_1^2,\ \operatorname{tr}(\mathbf{Y}^T\mathbf{B}_3\mathbf{Y}) \geq \kappa\|\mathbf{t}_b^T\mathbf{P}\mathbf{x}\|_1^2 \\
& \operatorname{tr}(\mathbf{Y}^T\mathbf{B}_4\mathbf{Y}) \geq \kappa\|(\mathbf{t}_f-\mathbf{t}_b)^T\mathbf{P}\mathbf{x}\|_1^2,\ \operatorname{tr}(\mathbf{Y}^T\mathbf{B}_1\mathbf{Y}) = 0.
\end{aligned} \tag{E.14}$$

Here, r is the rank of the desired solution, $\mathbf{B}_1 = \mathbf{1}\mathbf{1}^T$, $\mathbf{B}_2 = \mathbf{P}\mathbf{t}_f\mathbf{t}_f^T\mathbf{P}$, $\mathbf{B}_3 = \mathbf{P}\mathbf{t}_b\mathbf{t}_b^T\mathbf{P}$, $\mathbf{B}_4 = \mathbf{P}(\mathbf{t}_f - \mathbf{t}_b)(\mathbf{t}_f - \mathbf{t}_b)^T\mathbf{P}$, $\mathbf{A}_i = \mathbf{e}_i\mathbf{e}_i^T$, $\mathbf{e}_i \in \mathbb{R}^n$ is an elementary vector with a 1 at the ith position. After solving (E.14) using BCR (E.4), the final binary solution is extracted from the score vector using the swept random hyperplanes method [207].

We compare the performance of BCR with the highly customized BQP solver SDCut [196] and biased normalized cut (BNCut) [197]. BNCut is an extension of the Normalized cut algorithm [178] whereas SDCut is currently the most efficient and accurate SDR solver but limited only to solving BQP problems. Also, BNCut can support only one quadratic grouping constraint per problem.

Experiments: We consider the Berkeley image segmentation dataset [219]. Each image is segmented into super-pixels using the VL-Feat [220] toolbox. For SDCut and BNCut, we use the publicly available code with hyper-parameters set to the values suggested in [196]. For BCR, we set $\beta = \lambda/\sqrt{|\mathcal{B}\cup\mathcal{E}|}$, where λ controls the coarseness of the segmentation by mediating the tradeoff between the objective and constraints, and would typically be chosen from $[1, 10]$ via cross validation. For simplicity, we just set $\lambda = 5$ in all experiments reported here.

We compare the runtime and quality of each algorithm. Figure E.2 shows the segmentation

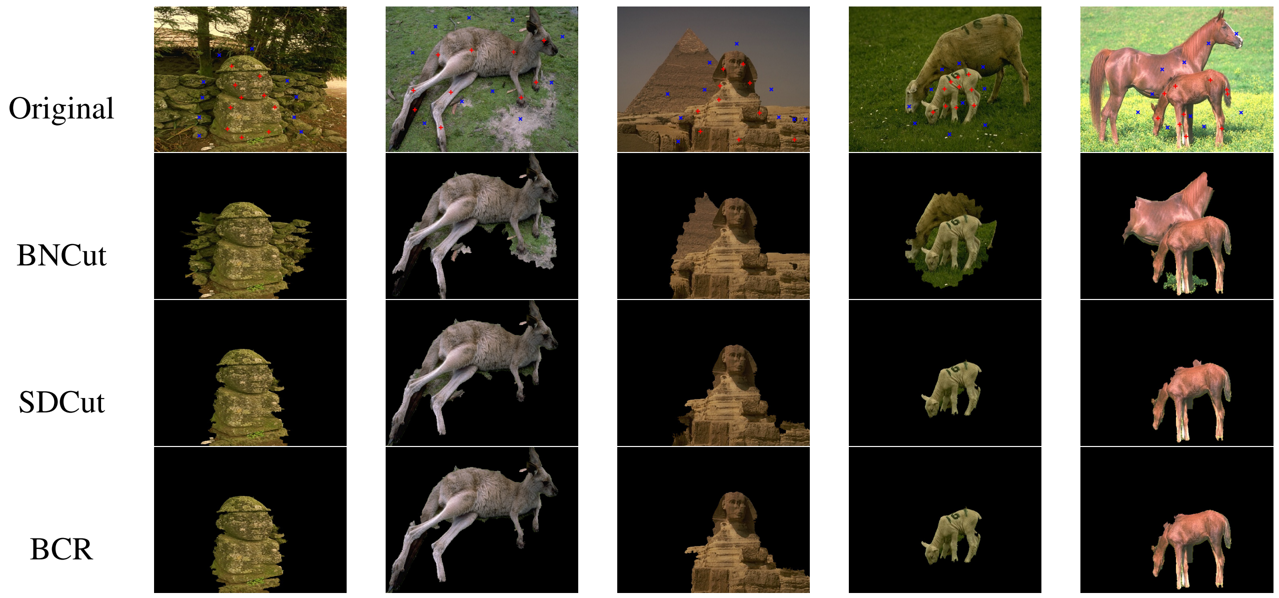


Figure E.2: Image segmentation results on the Berkeley dataset. The red and blue marker indicates the annotated foreground and background super-pixels, respectively.

results while the quantitative results are displayed in Table E.1. For all the considered images, our approach gives superior foreground object segmentation compared to SDCut and BNCut. Moreover, as seen in Table E.1, our solver is $35\times$ faster than SDCut and yields lower objective energy. Segmentation using BCR is achieved using only rank 2 solutions whereas SDCut requires rank 7 solutions to obtain results of comparable accuracy.[2] Note that while BNCut with rank 1 solutions is much faster than SDP based methods, the BNCut segmentation results are not on par with SDP approaches.

E.4.3 Co-segmentation

We next consider image co-segmentation, in which segmentation of the same object is jointly computed on multiple images simultaneously. Because co-segmentation involves multiple images, it provides a testbed for large problem instances. Co-segmentation balances a tradeoff between

[2]The optimal solutions found by SDCut all had rank 7 except for one solution of rank 5.

two criteria: (i) color and spatial consistency within a single image and (ii) discrimination between foreground and background pixels over multiple images. We closely follow the work of Joulin *et al.* [203], whose formulation is given by

$$\underset{\mathbf{x}\in\{\pm 1\}^N}{\text{minimize}}\ \mathbf{x}^T\mathbf{A}\mathbf{x} \quad \text{subject to}\ \ (\mathbf{x}^T\delta_i)^2 \le \lambda^2, \quad \forall i = 1,\dots,M, \tag{E.15}$$

where M is the number of images and $N = \sum_{i=1}^{M} N_i$ is the total number of pixels over all images. The matrix $\mathbf{A} = \mathbf{A}_b + \frac{\mu}{N}\mathbf{A}_w$, where $\mathbf{A}_w$ is the intra-image affinity matrix and $\mathbf{A}_b$ is the inter-image discriminative clustering cost matrix computed using the χ^2 distance between SIFT features in different images (see [203] for a details).

To solve this problem with BCR, we re-write (E.15) in the form (E.2) to obtain

$$\begin{aligned} \underset{\mathbf{X}\in\mathbb{R}^{N\times r}}{\text{minimize}} \quad & \operatorname{tr}(\mathbf{X}^T\mathbf{A}\mathbf{X}) \\ \text{subject to:} \quad & \operatorname{tr}(\mathbf{X}^T\mathbf{Z}_i\mathbf{X}) = 1, \quad \forall i = 1,\dots,N \\ & \operatorname{tr}(\mathbf{X}^T\mathbf{\Delta}_i\mathbf{X}) \le \lambda^2,\ \forall i = 1,\dots,M, \end{aligned} \tag{E.16}$$

where $\mathbf{\Delta}_i = \delta_\mathbf{i}\delta_\mathbf{i}{}^T$ and $\mathbf{Z}_i = \mathbf{e}_i\mathbf{e}_i^T$. Finally, (E.16) is solved using BCR (E.4), following which one can recover the optimal score vector $\mathbf{x}_p^*$ as the leading eigenvector of $\mathbf{X}^*$. The final binary solution is extracted by thresholding $\mathbf{x}_p^*$ to obtain integer-valued labels [198].

Experiments: We compare BCR to two well-known co-segmentation methods, namely low-rank factorization [198] (denoted LR) and SDCut [196]. We use publicly available code for LR and SDCut. We test on the Weizman horses[3] and MSRC[4] datasets with a total of four classes (horse,

[3]www.msri.org/people/members/eranb/

[4]www.research.microsoft.com/en-us/projects/objectclassrecognition/

Table E.1: Results on image segmentation. Numbers are the mean over the images in Fig. E.2. Lower numbers are better. The proposed algorithm and the best performance are highlighted.

Method	BNCut	SDCut	**BCR**
Time (s)	**0.08**	27.64	0.97
Objective	10.84	6.40	**6.34**
Rank	1	7	2

Table E.2: Co-segmentation results. The proposed algorithm and the best performance is highlighted.

		Test Cases			
Dataset		horse	face	car-back	car-front
Number of images		10	10	6	6
Variables in BQPs		4587	6684	4012	4017
Time (s)	LowRank	2647	1614	724	749
	SDCut	220	274	180	590
	BCR	**18.8**	**61.8**	**46.7**	**44.7**
Objective	LowRank	4.84	4.48	5.00	4.17
	SDCut	5.24	4.94	4.53	4.27
	BCR	**4.64**	**3.29**	**4.36**	**3.94**
Rank	LowRank	18	11	7	10
	SDCut	3	3	3	3
	BCR	**2**	**2**	**2**	**2**

car-front, car-back, and face) containing $6 \sim 10$ images per class. Each image is over-segmented to $400 \sim 700$ SLIC superpixels using the VLFeat [220] toolbox, giving a total of around $4000 \sim 7000$ super-pixels per class. Relative to image segmentation problems, this application requires $10\times$ more variables.

Qualitative results are presented in Figure E.3 while Table E.2 provides a quantitative comparison. From Table E.2, we observe that on average our method converges $\sim 9.5\times$ faster than SDCut and $\sim 60\times$ faster than LR. Moreover, the optimal objective value achieved by BCR is significantly lower than that achieved by both SDCut and LR methods. Figure E.3 displays the visualization of the final score vector $\mathbf{x}_p^*$ for selected images, depicting that in general SDCut and BCR produce similar results. Furthermore, the optimal BCR score vector $\mathbf{x}_p^*$ is extracted from a rank-2 solution, as compared to rank-3 and rank-7 solutions needed to get comparable results with SDCut and LR.

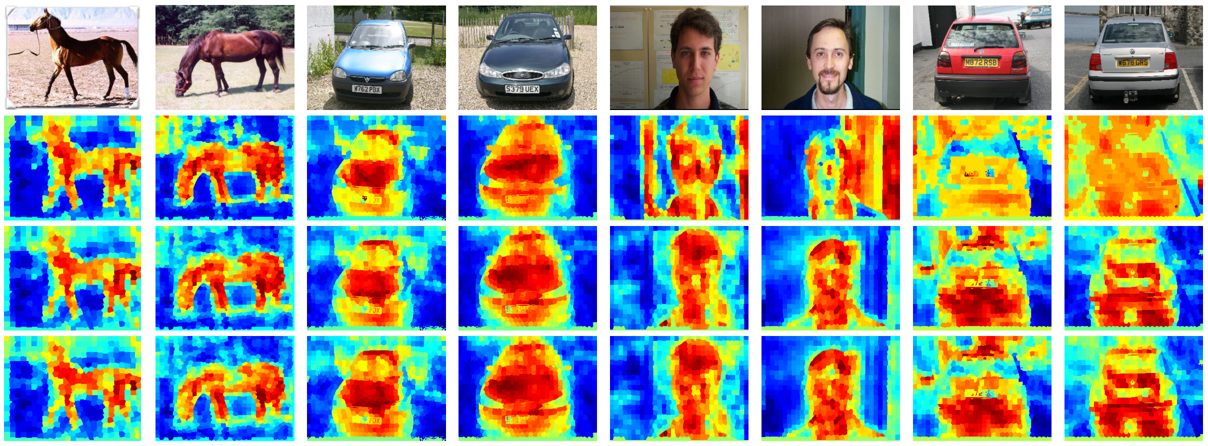


Figure E.3: Co-segmentation results on the Weizman horses and MSRC datasets. From top to bottom: the original images, the results of LR, SDCut, and BCR, respectively.

E.4.4 Metric Learning on Manifolds

Large SDPs play a central role in manifold methods for classification and dimensionality reduction on image sets and videos [199, 200, 221]. Manifold methods rely heavily on covariance matrices, which accurately characterize second-order statistics of variation between images. Typical methods require computing distances between matrices along a Riemannian manifold—a task that is expensive for large matrices and limits the applicability of these techniques. It is of interest to perform dimensionality reduction on SPD matrices, thus enabling the use of covariance methods on very large problems.

In this section, we discuss dimensionality reduction on manifolds of SPD matrices using BCR, which is computationally much faster than the state-of-the-art while achieving comparable (and often better) performance. Consider a set of high-dimensional SPD matrices $\{\mathrm{S}_1, \ldots, \mathrm{S}_n\}$ where

$\mathbf{S}_i \in S^+_{N\times N}$. We can project these onto a low-dimensional manifold of rank $K < N$ by solving

$$\begin{aligned} \underset{\mathbf{X}\in S^+_{N\times N}, \eta_{ij}\geq 0}{\text{minimize}} \quad & \operatorname{tr}(\mathbf{X}) + \mu \textstyle\sum_{i,j} \eta_{ij} \\ \text{subject to} \quad & \mathbb{D}_X(\mathbf{S}_i, \mathbf{S}_j) \leq u + \eta_{ij}, \quad \forall (i,j) \in \mathcal{C} \\ & \mathbb{D}_X(\mathbf{S}_i, \mathbf{S}_j) \geq l - \eta_{ij}, \quad \forall (i,j) \in \mathcal{D} \end{aligned} \tag{E.17}$$

where $\mathbf{X}$ is a (low-dimensional) SPD matrix, $\mathbb{D}_X$ is Riemannian distance metric, and η_{ij} are slack variables. The sets $\mathcal{C}$ and $\mathcal{D}$ contain pairs of similar/dissimilar matrices labeled by the user, and the scalars u and l are given upper and lower bounds. For simplicity, we measure distance using the log-Euclidean metric (LEM) defined by [199]

$$\mathbb{D}(\mathbf{S}_i, \mathbf{S}_j) = \|\log(\mathbf{S}_i) - \log(\mathbf{S}_j)\|_F^2 = \operatorname{tr}\big((\mathbf{R}_i - \mathbf{R}_j)^T(\mathbf{R}_i - \mathbf{R}_j)\big), \tag{E.18}$$

where $\mathbf{R}_i = \log(\mathbf{S}_i)$ is a matrix logarithm. When $\mathbf{X}$ has rank K, it is a transformation onto the space of rank K covariance matrices, where the new distance is given by [199]

$$\mathbb{D}_X(\mathbf{S}_i, \mathbf{S}_j) = \operatorname{tr}\big(\mathbf{X}(\mathbf{R}_i - \mathbf{R}_j)^T(\mathbf{R}_i - \mathbf{R}_j)\big). \tag{E.19}$$

We propose to solve the semi-definite program (E.17) using the representation $\mathbf{X} = \mathbf{Y}\mathbf{Y}^T$ which puts our problem in the form (E.2) with $\mathbf{A}_{ij} = (\mathbf{R}_i - \mathbf{R}_j)^T(\mathbf{R}_i - \mathbf{R}_j)$. This problem is then solved using BCR, where the slack variables $\{\eta_{ij}\}$ are removed and instead a hinge loss penalty approximately enforces the inequality constraints in (E.4). In our experiments we choose $u = \rho - \xi\tau$ and $l = \rho + \xi\tau$, where ρ and τ are the mean and standard deviation of the pairwise distances between $\{S_i\}$ in the original space, respectively. The quantities ξ and μ are treated as hyper-parameters.

Experiments: We analyze the performance of our approach (short BCRML) against state-of-the-art manifold metric learning algorithms using three image set classification databases: ETH-80, YouTube Celebrities (YTC), and YouTube Faces (YTF) [222]. The ETH-80 database consists of a 10 image set for each of 8 object categories. YTC contains 1,910 video sequences for 47 subjects from YouTube. YTF is a face verification database containing 3,425 videos of 1,595 different people. Features were extracted from images as described in [199]. Faces were cropped from each dataset using bounding boxes, and scaled to size 20×20 for the ETH and YTC datasets. For YTF we used a larger 30×30 scaling, as larger images were needed to replicate the results reported in [199].

We compare BCR to three state-of-the-art schemes: LEML [199] is based on a log-Euclidean metric, and minimizes the logdet divergence between matrices using Bregman projections. SPDML [200] optimizes a cost function on the Grassmannian manifold while making use of either the affine-invariant metric (AIM) or Stein metric. We use publicly available code for LEML and SPDML and follow the details in [199, 200] to select algorithm specific hyper-parameters using cross-validation. For BCRML, we fix α to be $1/\sqrt{|\mathcal{C} \cup \mathcal{D}|}$ and μ as $\alpha/2$. The ξ is fixed to 0.5, which performed well under cross-validation. For SPDML, the dimensionality of the target manifold K is fixed to 100. In LEML, the dimension cannot be reduced and thus the final dimension is the same as the original. Hence, for a fair comparison, we report the performance of BCRML using full target dimension (BCRML-full) as well as for $K = 100$ (BCRML-100).

Table E.3 summarizes the classification performance on the above datasets. We observe that BCRML performs almost the same or better than other ML algorithms. One can apply other algorithms to gain a further performance boost after projecting onto the low-dimensional manifold.

Table E.3: Image set classification results for state-of-the-art metric learning algorithms. The last three columns report computation time in seconds. The last 3 rows report performance using CDL-LDA after dimensionality reduction. Methods using the proposed BCR are listed in bold.

Method	ETH-80	YTC	YTF	Train (s)	Test (s)	Total (s)
AIM	89.25 ± 1.69	62.77 ± 2.89	59.82 ± 1.63	-	5.189	1463.3
Stein	89.00 ± 2.42	62.02 ± 2.71	57.56 ± 2.17	-	3.593	1013.3
LEM	90.00 ± 2.64	62.06 ± 3.04	59.78 ± 1.69	-	1.641	462
SPDML-AIM [200]	91.00 ± 3.39	65.32 ± 2.77	61.64 ± 1.46	3941	0.227	4005
SPDML-Stein [200]	90.75 ± 3.34	66.10 ± 2.92	61.66 ± 2.09	1447	**0.024**	1453.7
LEML [199]	92.00 ± 2.18	62.13 ± 3.13	60.92 ± 1.95	93	1.222	437.7
BCRML-full	92.00 ± 3.12	64.40 ± 2.92	60.58 ± 1.75	189	1.222	669.7
BCRML-100	92.25 ± 3.78	64.61 ± 2.65	$\mathbf{62.42 \pm 2.14}$	**45**	0.291	127
CDL-LDA [221]	$\mathbf{94.25 \pm 3.36}$	72.94 ± 1.81	N/A	-	1.073	302.7
LEML+CDL-LDA [199]	94.00 ± 3.57	73.01 ± 1.67	N/A	93	0.979	369
BCRML-100+CDL-LDA	93.75 ± 3.58	$\mathbf{73.48 \pm 1.83}$	N/A	45	0.045	**57.7**

Hence, we also provide a performance evaluation for LEML and BCRML using the LEM based CDL-LDA recognition algorithm [221]. The last three columns of Table E.3 display the runtime measured on the YTC dataset. We note that BCRML-100 trains roughly $2\times$ faster and overall runs about $3.5\times$ faster than the next fastest method. Moreover, on testing using CDL-LDA, the overall computation time is approximately $5\times$ faster in comparison to the next-best performing approach.

E.5 Summary

We have presented a novel biconvex relaxation framework (BCR) that enables the solution of general semidefinite programs (SDPs) at low complexity and with a small memory footprint. We have provided an alternating minimization (AM) procedure along with a new initialization method that, together, are guaranteed to converge, computationally efficient (even for large-scale problems), and able to handle a variety of SDPs. Comparisons of BCR with state-of-the-art methods for specific computer vision problems, such as segmentation, co-segmentation, and metric learning, show that BCR provides similar or better solution quality with significantly lower runtime. While this chapter only shows applications for a select set of computer vision problems, determining the efficacy of BCR for other problems in signal processing, machine learning, control, etc. is left for future work.

Bibliography

[1] Xiao Wang, Shiqian Ma, Donald Goldfarb, and Wei Liu. Stochastic quasi-newton methods for nonconvex stochastic optimization. *SIAM Journal on Optimization*, 27(2):927–956, 2017.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[4] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[5] 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.

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[8] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[10] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[11] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, 2017.

[12] Michael F Mathieu, Junbo Jake Zhao, Junbo Zhao, Aditya Ramesh, Pablo Sprechmann, and Yann LeCun. Disentangling factors of variation in deep representation using adversarial training. In *NIPS*, pages 5041–5049, 2016.

[13] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[14] 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.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

[17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[18] Raghu Bollapragada, Dheevatsa Mudigere, Jorge Nocedal, Hao-Jun Michael Shi, and Ping Tak Peter Tang. A progressive batching l-bfgs method for machine learning. In *International Conference on Machine Learning*, pages 619–628, 2018.

[19] Xiaolong Wang and Abhinav Gupta. Generative image modeling using style and structure adversarial networks. In *ECCV*, pages 318–335, 2016.

[20] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. In *ICLR*, 2017.

[21] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *ICLR Workshop*, 2017.

[22] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, pages 675–678, 2014.

[23] Harrison Edwards and Amos Storkey. Censoring representations with an adversary. In *ICLR*, 2016.

[24] Soham De, Abhay Yadav, David Jacobs, and Tom Goldstein. Automated inference with adaptive batches. In *International Conference on Artificial Intelligence and Statistics*, 2017.

[25] Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. In *Advances In Neural Information Processing Systems*, pages 181–189, 2015.

[26] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *Proceedings of The 30th International Conference on Machine Learning*, pages 343–351, 2013.

[27] Conghui Tan, Shiqian Ma, Yu-Hong Dai, and Yuqiu Qian. Barzilai-borwein step size for stochastic gradient descent. *arXiv preprint arXiv:1605.04131*, 2016.

[28] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[29] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[30] Michael P Friedlander and Mark Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.

[31] Richard H Byrd, Gillian M Chin, Jorge Nocedal, and Yuchen Wu. Sample size selection in optimization methods for machine learning. *Mathematical programming*, 134(1):127–155, 2012.

[32] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2014.

[33] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[34] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *arXiv preprint arXiv:1309.2388*, 2013.

[35] Aaron J Defazio, Tibério S Caetano, and Justin Domke. Finito: A faster, permutable incremental gradient method for big data problems. *arXiv preprint arXiv:1407.2710*, 2014.

[36] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex J Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, pages 2647–2655, 2015.

[37] Soham De and Tom Goldstein. Efficient distributed SGD with variance reduction. In *2016 IEEE International Conference on Data Mining*. IEEE, 2016.

[38] Reza Harikandeh, Mohamed Osama Ahmed, Alim Virani, Mark Schmidt, Jakub Konečnỳ, and Scott Sallinen. Stopwasting my gradients: Practical svrg. In *Advances in Neural Information Processing Systems*, pages 2251–2259, 2015.

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

[40] Dominik Csiba and Peter Richtárik. Importance sampling for minibatches. *arXiv preprint arXiv:1602.02283*, 2016.

[41] Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, pages 1017–1025, 2014.

[42] Hamed Karimi, Julie Nutini, and Mark Schmidt. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 795–811. Springer, 2016.

[43] Boris Teodorovich Polyak. Gradient methods for minimizing functionals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 3(4):643–653, 1963.

[44] Alexander Rakhlin, Ohad Shamir, and Karthik Sridharan. Making gradient descent optimal for strongly convex stochastic optimization. *arXiv preprint arXiv:1109.5647*, 2011.

[45] Danil Prokhorov. Ijcnn 2001 neural network competition. *Slide presentation in IJCNN*, 1, 2001.

[46] Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

[47] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[48] 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.

[49] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[50] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 4. Granada, Spain, 2011.

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[52] Abhay Kumar Yadav, Tom Goldstein, and David W Jacobs. Making l-bfgs work with industrial-strength nets. In *British Machine Vision Conference 2020*. British Machine Vision Association, 2020.

[53] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[54] Adrian J Shepherd. *Second-order methods for neural networks: Fast and reliable training methods for multi-layer perceptrons*. Springer Science & Business Media, 2012.

[55] Albert S Berahas and Martin Takáč. A robust multi-batch l-bfgs method for machine learning. *arXiv preprint arXiv:1707.08552*, 2017.

[56] Jacob Rafati and Roummel F Marcia. Improving l-bfgs initialization for trust-region methods in deep learning. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 501–508. IEEE, 2018.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[58] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[59] 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.

[60] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[61] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[62] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[63] Yun Fei, Guodong Rong, Bin Wang, and Wenping Wang. Parallel l-bfgs-b algorithm on gpu. *Computers & Graphics*, 40:1–9, 2014.

[64] Wenbo Gao and Donald Goldfarb. Quasi-newton methods: superlinear convergence without line searches for self-concordant functions. *Optimization Methods and Software*, 34(1):194–217, 2019.

[65] Chaoxu Zhou, Wenbo Gao, and Donald Goldfarb. Stochastic adaptive quasi-newton methods for minimizing expected values. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4150–4159. JMLR. org, 2017.

[66] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

[67] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

[68] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6, 2009.

[69] Nicol N Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-newton method for online convex optimization. In *Artificial intelligence and statistics*, pages 436–443, 2007.

[70] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In *International Conference on Machine Learning*, pages 604–612, 2014.

[71] Aryan Mokhtari and Alejandro Ribeiro. Global convergence of online limited memory bfgs. *The Journal of Machine Learning Research*, 16(1):3151–3181, 2015.

[72] Albert S Berahas, Jorge Nocedal, and Martin Takác. A multi-batch l-bfgs method for machine learning. In *Advances in Neural Information Processing Systems*, pages 1055–1063, 2016.

[73] Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2): 1008–1031, 2016.

[74] Nitish Shirish Keskar and Albert S Berahas. adaqn: An adaptive quasi-newton algorithm for training rnns. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 1–16. Springer, 2016.

[75] Frank Curtis. A self-correcting variable-metric algorithm for stochastic optimization. In *International Conference on Machine Learning*, pages 632–641, 2016.

[76] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[77] Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582, 2016.

[78] Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations. 2016.

[79] Shankar Krishnan, Ying Xiao, and Rif A Saurous. Neumann optimizer: A practical optimization algorithm for deep neural networks. 2018.

[80] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[81] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018.

[82] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Technical Report*, 2012.

[83] Abhay Yadav, Sohil Shah, Zheng Xu, David Jacobs, and Tom Goldstein. Stabilizing adversarial nets with prediction methods. In *International Conference on Learning Representations*, 2018.

[84] Andrew Brock, Theodore Lim, JM Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. In *ICLR*, 2017.

[85] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[86] Jonathan Ho, Jayesh Gupta, and Stefano Ermon. Model-free imitation learning with policy optimization. In *International Conference on Machine Learning*, pages 2760–2769, 2016.

[87] Martín Abadi and David G Andersen. Learning to protect communications with adversarial neural cryptography. *arXiv preprint arXiv:1610.06918*, 2016.

[88] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

[89] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015.

[90] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[91] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

[92] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. In *ICLR*, 2017.

[93] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. In *ICLR*, 2016.

[94] Yujia Li, Kevin Swersky, and Richard S Zemel. Generative moment matching networks. In *ICML*, pages 1718–1727, 2015.

[95] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. In *ICLR*, 2017.

[96] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NIPS*, pages 2234–2242, 2016.

[97] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[98] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. In *ICLR*, 2017.

[99] Mingqiang Zhu and Tony Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. *UCLA CAM Report*, pages 08–34, 2008.

[100] Ernie Esser, Xiaoqun Zhang, and Tony Chan. A general framework for a class of first order primal-dual algorithms for tv minimization. *UCLA CAM Report*, pages 09–67, 2009.

[101] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40 (1):120–145, 2011.

[102] Tom Goldstein, Min Li, and Xiaoming Yuan. Adaptive primal-dual splitting methods for statistical learning and image processing. In *NIPS*, pages 2089–2097, 2015.

[103] Cong Dang and Guanghui Lan. Randomized first-order methods for saddle point optimization. *arXiv preprint arXiv:1409.8625*, 2014.

[104] Guanghui Lan and Yi Zhou. An optimal randomized incremental gradient method. *arXiv preprint arXiv:1507.02000*, 2015.

[105] Yuchen Zhang and Xiao Lin. Stochastic primal-dual coordinate method for regularized empirical risk minimization. In *ICML*, pages 353–361, 2015.

[106] Zhanxing Zhu and Amos J Storkey. Adaptive stochastic primal-dual coordinate descent for separable saddle point problems. In *ECML-PKDD*, pages 645–658, 2015.

[107] Zhanxing Zhu and Amos J Storkey. Stochastic parallel block coordinate descent for large-scale saddle point problems. In *AAAI*, 2016.

[108] Jialei Wang and Lin Xiao. Exploiting strong convexity from data with primal-dual first-order algorithms. *ICML*, 2017.

[109] Atsushi Shibagaki and Ichiro Takeuchi. Stochastic primal dual coordinate method with non-uniform sampling based on optimality violations. *arXiv preprint arXiv:1703.07056*, 2017.

[110] Yunmei Chen, Guanghui Lan, and Yuyuan Ouyang. Optimal primal-dual methods for a class of saddle point problems. *SIAM Journal on Optimization*, 24(4):1779–1814, 2014.

[111] Linbo Qiao, Tianyi Lin, Yu-Gang Jiang, Fan Yang, Wei Liu, and Xicheng Lu. On stochastic primal-dual hybrid gradient approach for compositely regularized minimization. In *ECAI*, 2016.

[112] Mengdi Wang and Yichen Chen. An online primal-dual method for discounted markov decision processes. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 4516–4521. IEEE, 2016.

[113] Simon S Du, Jianshu Chen, Lihong Li, Lin Xiao, and Dengyong Zhou. Stochastic variance reduction methods for policy evaluation. *ICML*, 2017.

[114] Adams Wei Yu, Qihang Lin, and Tianbao Yang. Doubly stochastic primal-dual coordinate method for empirical risk minimization and bilinear saddle-point problem. *arXiv preprint arXiv:1508.03390*, 2015.

[115] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

[116] Balamurugan Palaniappan and Francis Bach. Stochastic variance reduction methods for saddle-point problems. In *NIPS*, pages 1408–1416, 2016.

[117] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, pages 675–678, 2014.

[118] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[119] Xun Huang, Yixuan Li, Omid Poursaeed, John Hopcroft, and Serge Belongie. Stacked generative adversarial networks. In *CVPR*, 2017.

[120] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. *ECCV*, pages 213–226, 2010.

[121] Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. The variational fair autoencoder. In *ICLR*, 2016.

[122] Amnon Geifman, Abhay Yadav, Yoni Kasten, Meirav Galun, David Jacobs, and Basri Ronen. On the similarity between the laplace and neural tangent kernels. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1451–1461. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1006ff12c465532f8c574aeaa4461b16-Paper.pdf`.

[123] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[124] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pages 8139–8148, 2019.

[125] Mikhail Belkin, Daniel J Hsu, and Partha Mitra. Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate. In *Advances in neural information processing systems*, pages 2300–2311, 2018.

[126] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. *arXiv preprint arXiv:1802.01396*, 2018.

[127] Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. Spectrum dependent learning curves in kernel regression and wide neural networks. *arXiv preprint arXiv:2002.02561*, 2020.

[128] Tengyuan Liang and Alexander Rakhlin. Just interpolate: Kernel" ridgeless" regression can generalize. *arXiv preprint arXiv:1808.00387*, 2018.

[129] Sanjeev Arora, Simon S. Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. In *International Conference on Learning Representations*, 2020.

[130] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 242–252, 2019.

[131] Ronen Basri, David Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. In *Advances in Neural Information Processing Systems*, pages 4763–4772, 2019.

[132] Motonobu Kanagawa, Philipp Hennig, Dino Sejdinovic, and Bharath K Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences. *arXiv preprint arXiv:1807.02582*, 2018.

[133] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[134] Christopher KI Williams. Computing with infinite networks. In *Advances in neural information processing systems*, pages 295–301, 1997.

[135] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[136] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[137] Alexander G de G Matthews, Mark Rowland, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*, 2018.

[138] Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.

[139] Amit Daniely, Roy Frostig, and Yoram Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *Advances In Neural Information Processing Systems*, pages 2253–2261, 2016.

[140] Kaixuan Huang, Yuqing Wang, Molei Tao, and Tuo Zhao. Why do deep residual networks generalize better than deep feedforward networks?–a neural tangent kernel perspective. *arXiv preprint arXiv:2002.06262*, 2020.

[141] Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels. *arXiv preprint arXiv:1911.00809*, 2019.

[142] Francis Bach. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research*, 18(1):629–681, 2017.

[143] Yuan Cao, Zhiying Fang, Yue Wu, Ding-Xuan Zhou, and Quanquan Gu. Towards understanding the spectral bias of deep learning. *arXiv preprint arXiv:1912.01198*, 2019.

[144] Ronen Basri, Meirav Galun, Amnon Geifman, David Jacobs, Yoni Kasten, and Shira Kritchman. Frequency bias in neural networks for input of non-uniform density. In *International Conference on Machine Learning*, 2020.

[145] Greg Yang and Hadi Salman. A fine-grained spectral perspective on neural networks. *arXiv preprint arXiv:1907.10599*, 2019.

[146] Zhou Fan and Zhichao Wang. Spectra of the conjugate kernel and neural tangent kernel for linear-width neural networks. *arXiv preprint arXiv:2005.11879*, 2020.

[147] Alberto Bietti and Julien Mairal. On the inductive bias of neural tangent kernels. In *Advances in Neural Information Processing Systems*, pages 12873–12884, 2019.

[148] Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Linearized two-layers neural networks in high dimension. *arXiv preprint arXiv:1904.12191*, 2019.

[149] Saburou Saitoh and Yoshihiro Sawano. *Theory of reproducing kernels and applications*. Springer, 2016.

[150] Elias M Stein and Guido Weiss. *Introduction to Fourier analysis on Euclidean spaces (PMS-32)*, volume 32. Princeton university press, 2016.

[151] George S Kimeldorf and Grace Wahba. A correspondence between bayesian estimation on stochastic processes and smoothing by splines. *The Annals of Mathematical Statistics*, 41 (2):495–502, 1970.

[152] Ha Quang Minh, Partha Niyogi, and Yuan Yao. Mercer's theorem, feature maps, and smoothing. In *International Conference on Computational Learning Theory*, pages 154–168. Springer, 2006.

[153] Francis J Narcowich, Xinping Sun, and Joseph D Ward. Approximation power of rbfs and their associated sbfs: a connection. *Advances in Computational Mathematics*, 27(1):107–124, 2007.

[154] Francis J Narcowich and Joseph D Ward. Scattered data interpolation on spheres: error estimates and locally supported basis functions. *SIAM Journal on Mathematical Analysis*, 33(6):1393–1410, 2002.

[155] Lin Chen and Sheng Xu. Deep neural tangent kernel and laplace kernel have the same RKHS. *arXiv preprint arXiv:2009.10683*, 2020.

[156] Marc G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312, 2001.

[157] Jean Gallier. Notes on spherical harmonics and linear representations of lie groups. *preprint*, 2009.

[158] Olivier Chapelle. Training a support vector machine in the primal. *Neural computation*, 19 (5):1155–1178, 2007.

[159] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*, 2019.

[160] Kurt Jetter, Joachim StÃķckler, and Joseph Ward. Error estimates for scattered data interpolation on spheres. *Mathematics of Computation*, 68(226):733–747, 1999.

[161] Charles A Micchelli and Grace Wahba. Design problems for optimal surface interpolation. Technical report, Wisconsin University Madison, Dept. of Statistics, 1979.

[162] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research*, 15(1):3133–3181, 2014.

[163] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[164] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. Falkon: An optimal large scale kernel method. In *Advances in Neural Information Processing Systems*, pages 3888–3898, 2017.

[165] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR)*, 2011.

[166] Pierre Baldi Peter Sadowski and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5, 2014.

[167] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Kernel descriptors for visual recognition. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 244–252. 2010.

[168] Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2627–2635. 2014.

[169] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[170] Marc Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[171] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *ICLR*, 2017.

[172] Youngmin Cho and Lawrence K Saul. Analysis and extension of arc-cosine kernels for large margin classification. *arXiv preprint arXiv:1112.3712*, 2011.

[173] George Neville Watson. *A treatise on the theory of Bessel functions*. Cambridge university press, 1966.

[174] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina F Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, pages 3041–3049, 2014.

[175] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[176] Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Jiri Hron, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. *arXiv preprint arXiv:1810.05148*, 2018.

[177] Sohil Shah, Abhay Kumar Yadav, Carlos D Castillo, David W Jacobs, Christoph Studer, and Tom Goldstein. Biconvex relaxation for semidefinite programming in computer vision. In *European Conference on Computer Vision*, pages 717–735. Springer, 2016.

[178] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[179] Jens Keuchel, Christoph Schno, Christian Schellewald, and Daniel Cremers. Binary partitioning, perceptual grouping, and restoration with semidefinite programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(11):1364–1379, 2003.

[180] Philip HS Torr. Solving markov random fields using semi definite programming. In *Artificial Intelligence and Statistics*, 2003.

[181] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[182] Mica Arie-Nachimson, Shahar Z Kovalsky, Ira Kemelmacher-Shlizerman, Amit Singer, and Ronen Basri. Global motion estimation from point matches. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 81–88. IEEE, 2012.

[183] Kilian Q Weinberger and Lawrence K Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70(1):77–90, 2006.

[184] Kaushik Mitra, Sameer Sheorey, and Rama Chellappa. Large-scale matrix factorization with missing data under additional constraints. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1651–1659, 2010.

[185] Zhi-Quan Luo, Wing-kin Ma, Anthony Man-Cho So, Yinyu Ye, and Shuzhong Zhang. Semidefinite relaxation of quadratic optimization problems. *IEEE Signal Processing Magazine*, 27(3):20–34, May 2010.

[186] Jean B Lasserre. An explicit exact sdp relaxation for nonlinear 0-1 programs. In *Integer Programming and Combinatorial Optimization*, pages 293–303. Springer, 2001.

[187] Stephen Boyd and Lieven Vandenberghe. Semidefinite programming relaxations of non-convex problems in control and combinatorial optimization. In *Communications, Computation, Control, and Signal Processing*, pages 279–287. Springer, 1997.

[188] Ady Ecker, Allan D Jepson, and Kiriakos N Kutulakos. Semidefinite programming heuristics for surface reconstruction ambiguities. In *Computer Vision–ECCV 2008*, pages 127–140. Springer, 2008.

[189] Sameer Shirdhonkar and David W Jacobs. Non-negative lighting and specular object recognition. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1323–1330. IEEE, 2005.

[190] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM review*, 38(1): 49–95, July 1996.

[191] Matthias Heiler, Jens Keuchel, and Christoph Schnörr. Semidefinite clustering for image segmentation with a-priori knowledge. In *Pattern Recognition*, pages 309–317. Springer, 2005.

[192] Chunhua Shen, Junae Kim, and Lei Wang. A scalable dual approach to semidefinite metric learning. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2601–2608. IEEE, 2011.

[193] Praneeth Netrapalli, Prateek Jain, and Sujay Sanghavi. Phase retrieval using alternating minimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2796–2804, 2013.

[194] Emmanuel J Candès, Xiaodong Li, and Mahdi Soltanolkotabi. Phase retrieval via wirtinger flow: Theory and algorithms. *Information Theory, IEEE Transactions on*, 61(4):1985–2007, 2015.

[195] Jason D Lee, Ben Recht, Nathan Srebro, Joel Tropp, and Ruslan R Salakhutdinov. Practical large-scale optimization for max-norm regularization. In *Advances in Neural Information Processing Systems*, pages 1297–1305, 2010.

[196] Peng Wang, Chunhua Shen, and Anton van den Hengel. A fast semidefinite approach to solving binary quadratic problems. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[197] Subhransu Maji, Nisheeth K Vishnoi, and Jitendra Malik. Biased normalized cuts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2057–2064. IEEE, 2011.

[198] Michel Journée, F Bach, P-A Absil, and Rodolphe Sepulchre. Low-rank optimization on the cone of positive semidefinite matrices. *SIAM Journal on Optimization*, 20(5):2327–2351, 2010.

[199] Zhiwu Huang, Ruiping Wang, Shiguang Shan, Xianqiu Li, and Xilin Chen. Log-euclidean metric learning on symmetric positive definite manifold with application to image set classification. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 720–729, 2015.

[200] Mehrtash T Harandi, Mathieu Salzmann, and Richard Hartley. From manifold to manifold: Geometry-aware dimensionality reduction for spd matrices. In *European Conference on Computer Vision*, pages 17–32. Springer, 2014.

[201] Xiao Bai, Hang Yu, and Edwin R Hancock. Graph matching using spectral embedding and alignment. In *International Conference on Pattern Recognition (ICPR)*, volume 3, pages 398–401. IEEE, 2004.

[202] Chaohui Wang, Nikos Komodakis, and Nikos Paragios. Markov random field modeling, inference & learning in computer vision & image understanding: A survey. *Computer Vision and Image Understanding*, 117(11):1610–1627, 2013.

[203] A. Joulin, F. Bach, and J. Ponce. Discriminative clustering for image co-segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[204] Peng Wang, Chunhua Shen, and Anton van den Hengel. Efficient SDP inference for fully-connected CRFs based on low-rank decomposition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[205] Christian Schellewald and Christoph Schnörr. Probabilistic subgraph matching based on convex relaxation. In *Energy minimization methods in computer vision and pattern recognition*, pages 171–186. Springer, 2005.

[206] Carl Olsson, Anders P Eriksson, and Fredrik Kahl. Solving large scale binary quadratic problems: Spectral methods vs. semidefinite programming. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[207] Kevin Lang. Fixing two weaknesses of the spectral method. In *Advances in Neural Information Processing Systems (NIPS)*, pages 715–722, 2005.

[208] Jos F Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625–653, 1999.

[209] K. C. Toh, M.J. Todd, and R.H. Tutuncu. Sdpt3 - a matlab software package for semidefinite programming. *Optimization Methods and Software*, 11:545–581, 1998.

[210] Takayuki Okatani and Koichiro Deguchi. On the wiberg algorithm for matrix factorization in the presence of missing components. *International Journal of Computer Vision*, 72(3): 329–337, 2007.

[211] Samuel Burer and Renato DC Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2):329–357, 2003.

[212] John C. Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934, 2009. doi: 10.1145/1577069.1755882. URL `http://doi.acm.org/10.1145/1577069.1755882`.

[213] Jim Douglas and James E Gunn. A general formulation of alternating direction methods. *Numerische Mathematik*, 6(1):428–453, 1964.

[214] Yangyang Xu and Wotao Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences*, 6(3):1758–1789, 2013.

[215] Qinqing Zheng and John Lafferty. A convergent gradient descent algorithm for rank minimization and semidefinite programming from random linear measurements. In *Neural Information Processing Systems (NIPS)*. 2015. URL `http://papers.nips.cc/paper/5830-a-convergent-gradient-descent-algorithm-for-rank-minimization-and-` `pdf`.

[216] Tom Goldstein, Christoph Studer, and Richard Baraniuk. A field guide to forward-backward splitting with a FASTA implementation. *arXiv eprint*, abs/1411.3406, 2014. URL `http://arxiv.org/abs/1411.3406`.

[217] Song Wang and Jeffrey Mark Siskind. Image segmentation with ratio cut. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:675–690, 2003.

[218] Stella X Yu and Jianbo Shi. Segmentation given partial grouping constraints. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2):173–183, 2004.

[219] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[220] Andrea Vedaldi and Brian Fulkerson. Vlfeat: An open and portable library of computer vision algorithms (2008), 2012.

[221] Ruiping Wang, Huimin Guo, Larry S Davis, and Qionghai Dai. Covariance discriminative learning: A natural and efficient approach to image set classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2496–2503. IEEE, 2012.

[222] Lior Wolf, Tal Hassner, and Itay Maoz. Face recognition in unconstrained videos with matched background similarity. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 529–534. IEEE, 2011.

[223] Tom Goldstein and Simon Setzer. High-order methods for basis pursuit. *UCLA CAM Report*, pages 10–41, 2010.