Novel Stochastic Solvers for Image Classification, Generation, and Further Explorations

Thesis submitted in partial
fulfillment of the requirements of the degree of

Master of Science
in
Computer Science and Engineering
by Research

by

Neel Mishra
2020701017
neel.mishra@research.iiit.ac.in

Advised by Dr. Pawan Kumar

International Institute of Information Technology
Hyderabad - 500032, India
November, 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled " **Optimizing Minimization and Minimax with Structural Information, Approximations, and Architectural Modifications** " by **Neel Mishra**, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Dr. Pawan Kumar

Bhagavad Gita (Chapter 3, Verse 8)

Perform your duty, for action is superior to inaction.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my parents, who have provided unwavering support throughout my life. In times of difficulty, both my father and mother have been there for me without hesitation. My father has always been prompt in addressing any situation, regardless of potential consequences. Their consistent presence and care have been incredibly valuable to me.

I am deeply thankful to my brother and sister, as they have played distinct roles that have greatly influenced my life. My brother's selflessness, always prioritizing others, has been a guiding light for me. Meanwhile, my sister has been a constant companion since my childhood. From helping me get ready for school to offering guidance throughout my journey, her nurturing presence has had a significant impact on shaping my character.

I am indebted to my guide, Dr. Pawan Kumar, whose guidance has been invaluable. He has not only encouraged me to explore new and daring ideas but has also unwaveringly believed in my abilities. Even during challenging research endeavors, his unwavering support and thorough training have given me the confidence to navigate the research process. I am forever grateful for his mentorship and dedication.

I would also like to express my gratitude to my friends, including Swayatta, Abhinaba, Ritam, Kinal, Arpan, Rudresh, Kaustabh, Uzefa, Gopichand, Nikhil, Anish, Avneesh, Ankit, Keshav, and others. Their unwavering support and influence have played a crucial role in my personal growth and character development. I deeply appreciate their friendship and the profound impact they have had on my life.

Lastly, I want to express my gratitude to God, whose presence and guidance have been paramount during my most challenging times. When I believed I had reached a dead end, it was the glimpse of his grace that helped me persevere and ultimately overcome adversity. I will always cherish the support and strive to follow the path guided by the divine presence.

Abstract

This thesis presents research on deep learning optimization, emphasizing three separate yet related topics: adaptive learning rates, first-order optimization for generative adversarial networks (GANs), and the effects of label smoothing.

Firstly, we propose a novel approach for obtaining adaptive learning rates in gradient-based descent methods for classification tasks. Departing from traditional methods that rely on decayed expectations of gradient-based terms, our approach leverages the angle between the current gradient and a new gradient computed from the orthogonal direction. By incorporating angle history, we determine adaptive learning rates that lead to superior accuracy compared to existing state-of-the-art optimizers. We provide empirical evidence of convergence and evaluate our approach on diverse benchmark datasets, employing prominent image classification architectures.

Secondly, we introduce a groundbreaking first-order optimization method tailored specifically for training GANs. Our method builds upon the Gauss-Newton method, approximating the min-max Hessian, and utilizes the Sherman-Morrison inversion formula to calculate the inverse. Operating as a fixed-point method that ensures necessary contraction, our approach produces high-fidelity images with enhanced diversity across multiple datasets. Notably, it outperforms state-of-the-art second-order methods, including achieving the highest inception score for CIFAR10. Additionally, our method demonstrates comparable execution times to first-order min-max methods.

Furthermore, we investigate the effects of label smoothing on GAN training, examining various optimizer variants and learning rates. Our research reveals that employing label smoothing with a high learning rate and the CGD optimizer yields results surpassing the quality attained by using ADAM with the same learning rate. Importantly, we establish that label smoothing plays a vital role, as its absence fails to generate comparable results. We also explore the impact of architectural changes on the generator's conditioning, providing valuable insights into the factors influencing GAN performance.

Our research advances the deep learning optimization field by delving into these interconnected areas. We present novel methodologies for adaptive learning rates, first-order optimization for GANs, and the importance of label smoothing. These advancements offer improved accuracy in classification tasks, enhanced image generation quality, and a deeper understanding of the nuances of GAN training.

Research Papers from the Thesis Work

Conference Papers

1. **Neel Mishra**, Pawan Kumar *"Angle Based Dynamic Learning Rate for Gradient Descent"*, **International Joint Conference on Neural Networks (IJCNN)**, 2023

2. **Neel Mishra**, Sukhjinder Kumar, Bamdev Mishra, Pratik Jawanpuria, *"A Gauss-Newton Approach for Min-Max Optimization in GANs"*, **Submitted at WACV**, 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"The greatest glory in living lies not in never falling, but in rising every time we fall."

-Nelson Mandela

The quote's emphasis on persistence and perseverance in the face of failure reflects the iterative nature of numerical optimization, which involves a series of ups and downs in the solver's trajectory. It's crucial to maintain resilience and learn from setbacks to refine the optimization approach and eventually reach the optimal solution.

1.1 Motivation

Numerical optimization is a powerful tool that has found its applications in a variety of sectors including science, engineering, economics, and many others.

The primary motivation for numerical optimization is to improve efficiency and performance in various applications. In engineering and physics, optimization is used to design structures, devices, and systems that are more energy-efficient, safer, and more reliable, and in chemical engineering, optimization is used to design chemical processes that maximize the yield of a desired product while minimizing the use of raw materials and energy.

Numerical optimisation is often used in artificial intelligence to train models that are capable of performing a variety of tasks, including as image recognition, natural language

processing, and autonomous navigation. By optimizing the parameters of these models, we can improve their accuracy, speed, and generalization capability. In conclusion, numerical optimization is a critical tool for solving many complex problems in various domains. Its applications are widespread and diverse, and its benefits are significant. As such, numerical optimization is an area that will continue to attract significant attention and research in the future.

Minimax optimization is a branch of optimization that has gained significant attention in various fields, including game theory, control theory, machine learning, and many others. The objective of minimax optimization is to find a solution that minimizes the maximum possible loss, given certain constraints and objectives. This approach is particularly useful in situations where the worst-case scenario needs to be taken into account, such as in competitive settings or in uncertain environments.

One of the main motivations for minimax optimization is to provide robust solutions that can withstand worst-case scenarios. In game theory, for instance, minimax optimization is used to determine optimal strategies for two or more players in a zero-sum game. A zero-sum game is a situation where one player's gain is the other player's loss. The objective of each player is to minimize their maximum possible loss, which can be achieved by maximizing their minimum possible gain. This approach ensures that the player's strategy is robust against any strategy that the opponent might choose.

1.2 Problem Addressed

Line search methods are an essential component of optimization algorithms that seek to identify the optimal learning rate for a given objective function. However, traditional line search methods can be time-consuming and may require many iterations to converge to a satisfactory solution. Our approach provides an alternative to these methods by using structural information and an additional gradient to enable a more efficient line search.

In addition to improving line search methods, we have also focused on developing solvers to facilitate better GAN training. Second-order solvers that compute a form of full Newton method can be quite effective in GAN training, but they are also notoriously slow. We have created a solver to deal with this problem, our solver can approximate a second-order solution while maintaining the execution complexity of a first-order method.

In addition to improving line search methods and developing efficient solvers, we have also explored label smoothing in GAN training. While label smoothing can mitigate the

need for momentum in some cases, we have found that in certain scenarios, momentum is absolutely necessary to enable the training of GANs that generate meaningful images. Our research has identified the limitations of label smoothing and shown where momentum is needed to compensate for these limitations. By understanding the interactions between label smoothing and momentum, we can develop more effective strategies for GAN training that can overcome these limitations and generate high-quality images. Our research has also examined how various architectural changes in the embedding layer can affect GAN training. Surprisingly, we discovered that restricting the embedding layer's number of layers can actually reduce the range of modes that a GAN can learn.

In summary, our research has focused on developing more efficient and effective methods for line search and GAN training. By leveraging structural information and examining the effects of label smoothing and architectural changes, we have made significant strides in improving these critical optimization processes.

1.3 Challenges

- Line search methods take a long time to find the optimal learning rate.
- Second order solvers that compute a form of full Newton take an excruciatingly long time to train, which can be a significant challenge for GAN training.
- The development of an efficient solver that approximates a second-order solution, but with the execution complexity of a first-order method, is a challenging task.
- Most solvers for GANs use momentum, but it is unclear whether it is possible to remove momentum altogether and still achieve good performance.

1.4 Contributions

- Development of a method that improves line search by exploiting structural information using an additional gradient.
- Design of an efficient solver for GAN training that approximates a second-order solution, while maintaining the execution complexity of a first-order method.

- Exploration of label smoothing as a strategy for GAN training and identification of its limitations.
- Analysis of the interactions between label smoothing and momentum, and identification of scenarios where momentum is necessary for GAN training.
- Investigation of the effects of architectural changes in the embedding layer on GAN training and identification of a surprising result that reducing the number of layers can limit the number of modes that a GAN can learn.

Chapter 2

Literature Survey

2.1 Introduction to Optimization

A constrained optimization problem is formulated as follows [57, Equation 1.1]:

$$\begin{aligned} &\text{minimize } f(\boldsymbol{x}) \\ &\text{subject to } g_i(\boldsymbol{x}) \leq 0, \quad i = 1, 2, \ldots, m \\ &\qquad\qquad h_j(\boldsymbol{x}) = 0, \quad j = 1, 2, \ldots, p, \end{aligned}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the vector of decision variables, $f(\boldsymbol{x})$ is the objective function to be minimized, $g_i(\boldsymbol{x})$ are the inequality constraints, and $h_j(\boldsymbol{x})$ are the equality constraints. The problem is to find the values of $\boldsymbol{x}$ that minimize $f(\boldsymbol{x})$ subject to the constraints $g_i(\boldsymbol{x}) \leq 0$ and $h_j(\boldsymbol{x}) = 0$.

2.2 Non stochastic optimization techniques

2.2.1 Gradient Descent

A popular optimisation technique for locating a function's minimum is gradient descent. Gradient descent works by iteratively updating the values of the choice variables in the direction of the objective function's negative gradient. In other words, the algorithm tries to find the steepest descent direction at each iteration and moves the decision variables in that direction to reach the minimum of the function. The gradient of the objective function

$f(\boldsymbol{x})$ is defined as the vector of partial derivatives with respect to each decision variable:

$$\nabla f(\boldsymbol{x}) = \left[\frac{\partial f}{\partial x_1} \ \frac{\partial f}{\partial x_2} \ \dots \ \frac{\partial f}{\partial x_n} \right]. \tag{2.1}$$

The gradient descent algorithm starts with an initial guess of the decision variable values, denoted as $\boldsymbol{x}^{(0)}$. At each iteration k, the algorithm updates the decision variable values using the following update rule:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \alpha_k \nabla f(\boldsymbol{x}^{(k)}),$$

where α_k is the step size or learning rate at iteration k. The learning rate determines the size of the step taken in the descent direction and contributes much to the convergence of the algorithm. The algorithm may converge very slowly if the learning rate is too low, and it may fail to converge or even diverge if it is too high. The update rule for gradient descent can be interpreted as follows: at each iteration, the algorithm determines the negative gradient of the objective function at the current point and scales the decision variables in that direction. The algorithm continues to iterate until it reaches a stopping criterion, such as a maximum number of iterations or a small enough change in the objective function value. Gradient descent has a number of variations, such as batch, stochastic, and mini-batch gradient descents, which vary in how the gradient is computed and the size of the updates. While stochastic gradient descent only uses one randomly chosen data point at a time, batch gradient descent computes the gradient utilising all of the data points. A middle ground between the two is mini-batch gradient descent, in which the gradient is calculated using a very small batch of data points at each iteration.

Algorithm 1 Gradient Descent Algorithm

Require: Initial guess $\boldsymbol{x}^{(0)}$, learning rate η, tolerance ϵ
Ensure: Optimal solution $\boldsymbol{x}^*$
1: Initialize $k \leftarrow 0$
2: **repeat**
3: Compute gradient $\nabla f(\boldsymbol{x}^{(k)})$
4: Update variables: $\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} - \alpha_k \nabla f(\boldsymbol{x}^{(k)})$
5: $k \leftarrow k + 1$
6: **until** $||\nabla f(\boldsymbol{x}^{(k)})|| \leq \epsilon$

In the algorithm 1, the objective function is denoted as $f(\boldsymbol{x})$, where $\boldsymbol{x}$ is the decision variable vector. The gradient of the objective function is denoted as $\nabla f(\boldsymbol{x})$, which is a vector of partial derivatives with respect to each decision variable. The initial guess of the decision variable values is denoted as $\boldsymbol{x}^{(0)}$. The decision variable values at each iteration are denoted as $\boldsymbol{x}^{(k)}$. The step size or learning rate at iteration k is denoted as α_k. The stopping criterion is a condition that determines when to stop the iteration process, such as a maximum number of iterations or a small enough change in the objective function value. The optimal solution is denoted as $\boldsymbol{x}^*$.

2.2.2 Newton's Method

Newton's method is an iterative method that requires the computation of the function's first and second derivatives, denoted as $\nabla f(\boldsymbol{x})$ and $\nabla^2 f(\boldsymbol{x})$ respectively. Iteratively updating the decision variables is done using a second-order Taylor approximation of the objective function. The fundamental principle of Newton's approach is to solve for the minimum of the quadratic function that is used to approximate the objective function at each iteration. The decision variables are updated by subtracting the inverse of the Hessian matrix, multiplied by the gradient vector, from the current decision variable values. The Hessian matrix is denoted as $\boldsymbol{H}(\boldsymbol{x})$, which is a symmetric positive definite matrix if the objective function is convex.

The Newton's method algorithm can be written as follows:

Algorithm 2 Newton's Method Algorithm

Require: Initial guess $\boldsymbol{x}^{(0)}$, tolerance ϵ
Ensure: Optimal solution $\boldsymbol{x}^*$
1: Initialize $k \leftarrow 0$
2: **repeat**
3: Compute gradient and Hessian: $\nabla f(\boldsymbol{x}^{(k)})$ and $\boldsymbol{H}(\boldsymbol{x}^{(k)})$
4: Solve for $\boldsymbol{p}$: $\boldsymbol{H}(\boldsymbol{x}^{(k)})\boldsymbol{p} = -\nabla f(\boldsymbol{x}^{(k)})$
5: Update variables: $\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}$
6: $k \leftarrow k + 1$
7: **until** $||\nabla f(\boldsymbol{x}^{(k)})|| \leq \epsilon$

In this algorithm, the initial guess of the decision variable values is denoted as $\boldsymbol{x}^{(0)}$, and the decision variable values at each iteration are denoted as $\boldsymbol{x}^{(k)}$. The Hessian matrix is denoted as $\boldsymbol{H}(\boldsymbol{x})$. The step direction at each iteration, denoted as $\boldsymbol{p}$, is obtained by solving

the linear system of equations

$$\boldsymbol{H}(\boldsymbol{x}^{(k)})\boldsymbol{p} = -\nabla f(\boldsymbol{x}^{(k)}).$$

Newton's method is known to converge quadratically [57, Theorem 3.5] when the objective function is twice continuously differentiable and has a positive definite and Lipschitz continous Hessian matrix. The Hessian matrix must be positive definite for the inverse to exist, and computing it can be computationally expensive. Therefore, modifications to Newton's method have been proposed, such as the quasi-Newton methods, which approximate the Hessian matrix instead of computing it exactly, or the Newton-CG method, which uses a conjugate gradient method to solve for the step direction instead of solving a linear system of equations.

It is also worth noting that Newton's method can be extended to handle constraints by using constrained optimization techniques such as the interior-point method or the augmented Lagrangian method. These methods incorporate the constraints into the objective function and use Newton's method to find the optimal solution subject to the constraints.

Newton's method is a robust optimization algorithm that can converge quickly to the optimal solution if certain conditions are met. However, it can also be computationally expensive and sensitive to the choice of initial guess. Therefore, it is important to carefully consider the problem at hand and choose the appropriate optimization algorithm for the specific application.

2.2.3 Krylov subspace solvers

Krylov subspace methods are iterative algorithms used to solve large sparse systems of linear equations of the form $Ax = b$, where A is a large, sparse matrix, b is a vector of constants, and x is the unknown solution vector. These techniques are especially helpful for resolving problems where A is either too big to fit in memory or requires too much processing power to directly invert.

The key idea behind Krylov subspace methods is to construct a sequence of subspaces K_n of increasing dimension that are generated by repeated multiplication of A by a given vector. The vectors in K_n are then used to construct an approximate solution to the linear system.

Conjugate gradient (CG) is a common Krylov subspace technique [49, Chapter 2]. The CG method is made to solve symmetric positive-definite systems, which are widespread in many applications, including optimisation and finite element analysis issues. The CG method generates a sequence of vectors $x_1, x_2, \ldots, x_n$ that converge to the exact solution x.

2.2.3.1 Conjugate Gradient(CG) Method

The CG method works by iteratively minimizing the residual $r_k = b - Ax_k$ over the subspace K_k, where x_k is the current approximate solution. The search directions are chosen to be conjugate with respect to the matrix A, meaning that the dot product of any two search directions is zero. This property ensures that the algorithm converges in at most n iterations, where n is the size of the system.

The Conjugate Gradient (CG) algorithm is derived by minimizing the error function $\phi(x)$ in the A-norm, as shown in the Equation (2.2)

$$\phi(x) = \frac{1}{2}x^T Ax - b^T x. \tag{2.2}$$

To minimize $\phi(x)$, we need to find a set of search directions $p_1, p_2, \ldots, p_n$ that are mutually conjugate with respect to the matrix A, i.e they must satisfy the Equation (2.3):

$$p_i^T Ap_j = 0 \qquad \text{for } i \neq j. \tag{2.3}$$

We also need to find a set of step sizes $\alpha_1, \alpha_2, \ldots, \alpha_n$ that minimize $\phi(x)$ in the A-norm along each search direction p_i. We can find these step sizes by solving the Equation (2.4), which is a one-dimensional (in α) optimization problem:

$$\alpha_i = \arg\min_{\alpha \in \mathbb{R}} \phi(x_i + \alpha p_i). \tag{2.4}$$

To solve this optimization problem, we differentiate $\phi(x_i + \alpha p_i)$ with respect to α and set the derivative equal to zero:

$$\frac{d}{d\alpha}\phi(x_i + \alpha p_i) = (x_i + \alpha p_i)^T Ap_i - b^T p_i = 0. \tag{2.5}$$

Solving for α yields:

$$\alpha_i = \frac{r_i^T r_i}{p_i^T A p_i}, \tag{2.6}$$

where $r_i = b - Ax_i$ is the residual vector at iteration i.

Using these step sizes, we can update our approximation of the solution as follows:

$$x_{i+1} = x_i + \alpha_i p_i. \tag{2.7}$$

Substituting this into the definition of the residual vector, we get:

$$\begin{aligned} r_{i+1} &= b - Ax_{i+1} \\ &= b - A(x_i + \alpha_i p_i) \\ &= b - Ax_i - \alpha_i A p_i \\ &= r_i - \alpha_i A p_i. \end{aligned}$$

To ensure that our search directions are mutually conjugate, we need to update our search direction as follows:

$$p_{i+1} = r_{i+1} + \beta_i p_i. \tag{2.8}$$

Substituting the expression for p_{i+1} as p_i, and p_i as p_j in the equation (2.3), we get:

$$(r_{i+1} + \beta_i p_i)^T A p_i = 0.$$

Expanding and simplifying this expression, we get:

$$\beta_i = -\frac{r_{i+1}^T A p_i}{p_i^T A p_i}.$$

Incorporating all of this will give the CG algorithm that is shown in the Algorithm (3).

Algorithm 3 Conjugate Gradient Method

Require: Linear system $Ax = b$, initial guess $\boldsymbol{x}_0$, tolerance ϵ, and symmetric A
Ensure: Solution $\boldsymbol{x}$

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
$\boldsymbol{p}_0 = \boldsymbol{r}_0$
$k = 0$
while $\|\boldsymbol{r}_k\| > \epsilon$ **do**
    $\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$
    $\boldsymbol{x_{k+1}} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
    $\boldsymbol{r_{k+1}} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k$
    $\beta_{k+1} = \frac{\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{r}_k}$
    $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1} \boldsymbol{p}_k$
    $k = k + 1$
end while
return $\boldsymbol{x}_k$

2.3 Stochastic Optimization

2.3.1 Introduction

Stochastic optimization is a crucial field in machine learning that deals with finding optimal solutions to problems that involve uncertainty. The importance of stochastic optimization in machine learning arises from the fact that many real-world machine learning problems are inherently stochastic. Additionally, in deep learning, the stochastic gradient descent algorithm is widely used to optimize the parameters of the neural network, which involves randomly selecting mini-batches of data to update the parameters.

Stochastic optimization provides a powerful framework for modeling and solving such problems by taking into account the randomness and uncertainty in the problem formulation. It allows for the incorporation of probabilistic models and statistical methods into the optimization process, which can lead to more robust and reliable solutions. Furthermore, stochastic optimization can help in exploring the solution space more efficiently and effectively, by leveraging the randomization in the optimization process to escape local optima and discover better solutions.

In summary, stochastic optimization is an essential tool in machine learning for addressing real-world problems that involve randomness and uncertainty. Its ability to model and

solve such problems can lead to more robust and reliable solutions, and can also help in exploring the solution space more efficiently and effectively.

The stochastic objective function is defined as:

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} f_i(\boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta}), \tag{2.9}$$

where $\boldsymbol{\theta}$ is a vector of model parameters, N is the number of data samples, $f_i(\boldsymbol{\theta})$ is the loss function associated with the i-th sample, $R(\boldsymbol{\theta})$ is a regularization term, and λ is a hyperparameter controlling the strength of regularization.

2.3.2 Non-Convexity

A *non-convex* function is one that doesn't satisfies the conditions of convexity mentioned in [8, Chapter 3]. Non-convexity, therefore, means that the function has some regions where the curvature is negative or flat, leading to multiple local minima and maxima. This can create problems in optimization because finding the global minimum or maximum of a non-convex function can be challenging.

The implications of non-convexity for optimization are significant because many real-world problems involve non-convex objective functions. Non-convexity can arise due to complex interactions between variables or nonlinear constraints, making the optimization problem much harder to solve. Moreover, finding the global optimum for non-convex functions is challenging since these functions may have several local optima.

To optimize non-convex functions, we need to use specialized algorithms that can deal with non-convexity. These algorithms can broadly be classified into two categories: deterministic and stochastic. Deterministic methods, such as gradient descent and Newton's method, search for the optimum by iteratively improving an initial solution. Stochastic methods, such as simulated annealing and genetic algorithms, use randomization to explore the search space and find the global optimum. Stochasticity and non-convexity can create a challenging and unpredictable training environment for machine learning models.

The challenges of optimizing non-convex objective functions include:

1. **Several local optima**: Finding the global optimum of non-convex functions is difficult because these functions might have several local optima.

2. **Computational item complexity**: The computational complexity of finding the global optimum increases exponentially with the number of variables, making it challenging to optimize high-dimensional non-convex functions.

3. **Initialization**: The performance of many optimization algorithms depends heavily on the initial solution. Finding a good initial solution for non-convex functions can be challenging, and the quality of the solution can affect the convergence rate.

4. **Convergence**: Non-convex optimization algorithms can converge to a local optimum instead of a global one, making it difficult to guarantee the solution's optimality.

In summary, non-convexity refers to functions that do not satisfy convexity conditions, leading to multiple local optima and challenging optimization. While stochasticity can introduce randomness and variability in the training process, non-convexity can make finding the optimal model parameters difficult. To overcome these challenges, specialized algorithms must deal with non-convexity and stochasticity.

2.3.3 Stochastic Gradient Descent (SGD)

The optimisation method known as Stochastic Gradient Descent (SGD) is utilised frequently for training machine learning models with stochastic objective functions. The fundamental concept of SGD is to update the model parameters in the direction of the objective function's negative gradient, which is computed using a mini-batch—a small batch—of training data that is randomly chosen. The update rule for SGD can be expressed as:

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t, x_{i:t}), \tag{2.10}$$

where θ_t is the parameter vector at time step t, $f(\theta_t, x_{i:t})$ is the objective function evaluated at θ_t using a mini-batch of data points $x_{i:t}$, $\nabla f(\theta_t, x_{i:t})$ is the gradient of the objective function with respect to θ_t, and α is the learning rate, which determines the step size for the update. In practice, SGD is typically used with a variant known as mini-batch SGD, where the objective function is evaluated using a small batch of data points at each iteration. This can help to reduce the variance of the gradient estimate and improve convergence. Mini-batch SGD can be summarized as follows:

1. Randomly sample a mini-batch of data points $x_{i:t}$

2. Compute the gradient estimate $\nabla f(\theta_t, x_{i:t})$

3. Update the parameters using the SGD update rule

4. Repeat until convergence

Overall, SGD is a simple but effective optimization technique for training machine learning models with stochastic objective functions. Its ease of implementation and scalability to large datasets have made it a popular choice for many applications, including, but not limited to image classification [46, 74], object detection [61, 63], natural language processing [11, 79], speech recognition [1, 23], recommender systems [42, 64], neural machine translation [3, 85], video analysis [19, 82], generative models [22, 67], reinforcement learning [54, 55], adversarial attacks [27, 31], semi-supervised learning [58, 38], transfer learning [16, 10], online learning [77, 79], federated learning [59, 68], graph neural networks [66], Bayesian deep learning[62, 26]. Momentum is a technique used to accelerate the convergence of Stochastic Gradient Descent (SGD). It helps the optimizer to move faster towards the minimum of the loss function by accumulating the previous gradients. The momentum technique updates the gradient descent with a fraction of the previous gradient. This fraction is a hyperparameter called momentum coefficient, denoted by γ. The update rule for the momentum technique is as follows:

$$v_t = \gamma v_{t-1} + \eta \nabla f(\theta_{t-1}, x_{i:t}),$$

where v_t is the current velocity, η is the learning rate, $\nabla f(\theta_{t-1}, x_{i:t})$ is the gradient of the loss function with respect to the parameters θ. The velocity is initialized to zero $v_0 = 0$. Then the parameters are updated as follows:

$$\theta_t = \theta_{t-1} - v_t.$$

The momentum technique helps to dampen the oscillations in the gradient descent path, which allows the optimizer to take larger steps in the correct direction. This technique is particularly useful for high-dimensional optimization problems with a lot of noise in the gradients.

2.3.4 RMSprop : Root Mean Square Propagation

In the lecture 6 of the online course "Neural Networks for Machine Learning" [78], RMSprop was first introduced. It is an optimization algorithm that adapts the learning rate for each parameter during training. It adapts the learning rates based on the historical gradient information, which allows it to converge faster and handle noisy gradients effectively.

Algorithm 4 RMSprop Optimization Algorithm

1: Initialize parameters: θ_0, moving average of squared gradients v_0, time step $t = 0$
2: Set hyperparameters: α (learning rate), β (exponential decay rate), ϵ (small constant to prevent division by zero)
3: **while** not converged **do**
4: Increment time step: $t \leftarrow t + 1$
5: Randomly sample a mini-batch of data points $x_{i:t}$
6: Compute the gradient: $\nabla f(\theta_{t-1}, x_{i:t})$
7: Update moving average of squared gradients: $v_t \leftarrow \beta v_{t-1} + (1-\beta)(\nabla f(\theta_{t-1}, x_{i:t}))^2$
8: Update parameters: $\theta_t \leftarrow \theta_{t-1} - \alpha \dfrac{\nabla f(\theta_{t-1}, x_{i:t})}{\sqrt{v_t} + \epsilon}$
9: **end while**

In the algorithm 9, θ_t represents the parameter vector at time step t, $f(\theta_t, x_{i:t})$ is the objective function evaluated using a mini-batch of data points $x_{i:t}$, and $\nabla f(\theta_{t-1}, x_{i:t})$ is the gradient of the objective function with respect to θ_{t-1}. The hyperparameters α, β, and ϵ control the learning rate, the exponential decay rate, and a small constant to prevent division by zero.

2.3.5 Adam : Adaptive Moment Estimation

In their work Kingma and Ba introduced Adam which is an optimization algorithm that combines the benefits of both momentum and RMSprop. It is widely used for training deep learning models. Adam dynamically adapts the learning rates for each parameter, making it well-suited for non-convex optimization problems.

The Algorithm of Adam is summarized in Algorithm 5, θ_t represents the parameter vector at time step t, $f(\theta_t, x_{i:t})$ is the objective function evaluated using a mini-batch of data points $x_{i:t}$, and $\nabla f(\theta_{t-1}, x_{i:t})$ is the gradient of the objective function with respect to θ_{t-1}. The hyperparameters α, β_1, β_2, and ϵ control the learning rate, the exponential

Algorithm 5 Adam Optimization Algorithm

Initialize parameters: θ_0, first moment vector m_0, second moment vector v_0, time step $t = 0$
Set hyperparameters: α (learning rate), β_1 (exponential decay rate for the first moment), β_2 (exponential decay rate for the second moment), ϵ (small constant to prevent division by zero)
while not converged **do**
    Increment time step: $t \leftarrow t + 1$
    Randomly sample a mini-batch of data points $x_{i:t}$
    Compute the gradient: $\nabla f(\theta_{t-1}, x_{i:t})$
    Update first moment estimate: $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)\nabla f(\theta_{t-1}, x_{i:t})$
    Update second moment estimate: $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)(\nabla f(\theta_{t-1}, x_{i:t}))^2$
    Correct for bias in first moment: $\hat{m}_t \leftarrow \dfrac{m_t}{1 - \beta_1^t}$
    Correct for bias in second moment: $\hat{v}_t \leftarrow \dfrac{v_t}{1 - \beta_2^t}$
    Update parameters: $\theta_t \leftarrow \theta_{t-1} - \alpha \dfrac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
end while

decay rates for the first and second moments, and a small constant to prevent division by zero.

Adam's adaptive learning rates, along with the momentum and RMSprop-like mechanisms, make it a popular choice for training deep neural networks. It helps accelerate convergence and handle noisy or sparse gradients effectively. Other variants of Adam for deep learning are NADAM [17], Rectified Adam (RAdam) [50], Nostalgic Adam [32], and AdamP [28].

2.3.6 Other notable stochastic solvers

Dubey et al. introduced a stochastic solver diffGrad, which uses a friction coefficient called diffGrad friction coefficient (DFC); They compute this coefficient using a non-linear sigmoid function on the change in gradients of current and immediate past iteration. Adabelief [91], another popular optimizer uses the square of the difference between the current gradient and the current exponential moving average to scale the current gradient. Many of the popular optimizers have played around with these four properties, namely, momentum, a decaying average of gradients or weight updates, gradient scaling, and a lookahead vector to derive their gradient update rules.

There have been few papers that have considered the angle information between successive gradients. Song et al. tries to correct the obtained gradient's direction by using the angle between the successive current and the past gradients. Roy et al. have also used the angle between successive gradients, but they control the step size based on the angle obtained from previous iterations and use it to calculate the angular coefficient, essentially the `tanh` of the angle between successive gradients. They claim that it reduces the oscillations between the successive gradient vectors; they also combine the gradient scaling, like RMSprop, and the expected weighted average of gradients as in ADAM.

2.3.7 Regularization

Optimization problems involving non-convex functions are notoriously challenging due to the presence of multiple local minima and non-linearities. Traditional optimization methods often struggle to converge to the global minimum in such cases. Regularization techniques offer a powerful approach to mitigate these issues and improve the optimization process. By introducing additional terms or constraints to the objective function, *regularization methods* guide the optimization algorithm towards better solutions.

2.3.8 L1 Regularization (Lasso)

L1 regularization, also known as Lasso regularization, is a technique that adds the absolute values of the model's coefficients as a penalty term to the objective function. It encourages sparsity in the solution by driving some coefficients to zero. The L1 regularization term can be defined as:

$$R_{\mathrm{L1}}(\boldsymbol{w}) = \lambda \sum_{i=1}^{n} |w_i|, \tag{2.11}$$

where $\boldsymbol{w}$ is the vector of model coefficients, w_i represents the i-th coefficient, and λ is the regularization parameter that controls the strength of the penalty.

2.3.8.1 L2 Regularization (Ridge)

L2 regularization, also known as Ridge regularization, adds the squared magnitudes of the model's coefficients to the objective function. Unlike L1 regularization, L2 regularization encourages small but non-zero values for all coefficients. The L2 regularization term can be defined as:

$$R_{\mathrm{L2}}(\boldsymbol{w}) = \lambda \sum_{i=1}^{n} w_i^2, \tag{2.12}$$

where $\boldsymbol{w}$ and w_i have the same meaning as in L1 regularization, and λ is the regularization parameter.

2.3.8.2 Elastic Net Regularization

Elastic Net regularization combines L1 and L2 regularization techniques to leverage their respective advantages. It adds both the absolute values and the squared magnitudes of the model's coefficients to the objective function. The Elastic Net regularization term can be defined as:

$$R_{\mathrm{ElasticNet}}(\boldsymbol{w}) = \lambda_1 \sum_{i=1}^{n} |w_i| + \lambda_2 \sum_{i=1}^{n} w_i^2, \tag{2.13}$$

where $\boldsymbol{w}$, w_i, λ_1, and λ_2 have the same meaning as in L1 and L2 regularization.

2.4 Wolfe conditions

The Wolfe conditions are a set of criteria used in line search methods to ensure the sufficient decrease and curvature properties of the step size. These conditions are commonly employed to determine an appropriate step length that guarantees convergence and maintains progress towards the optimal solution.

2.4.1 Sufficient Decrease Condition

The sufficient decrease condition, also known as the Armijo condition, ensures that the step size significantly decreases the objective function value. It is defined as follows:

$$f(\boldsymbol{x} + \alpha \boldsymbol{d}) \leq f(\boldsymbol{x}) + c_1 \alpha \nabla f(\boldsymbol{x})^T \boldsymbol{d}, \tag{2.14}$$

where α represents the step length, $\boldsymbol{x}$ is the current solution point, $\boldsymbol{d}$ is the search direction, $f(\boldsymbol{x})$ is the objective function value at $\boldsymbol{x}$, $\nabla f(\boldsymbol{x})$ is the gradient of f at $\boldsymbol{x}$, and c_1 is a constant satisfying $0 < c_1 < 1$.

The basic idea behind the sufficient decrease condition is to ensure that the step size decreases the objective function value that is proportional to the step length and aligned with the gradient direction. If this condition is not satisfied, the step length is reduced until the condition holds.

2.4.2 Curvature Condition

The curvature condition, also known as the curvature condition or the strong Wolfe condition, ensures that the step size does not lead to excessive curvature in the objective function. It is defined as follows:

$$\nabla f(\boldsymbol{x} + \alpha \boldsymbol{d})^T \boldsymbol{d} \geq c_2 \nabla f(\boldsymbol{x})^T \boldsymbol{d}, \tag{2.15}$$

where $\nabla f(\boldsymbol{x}+\alpha\boldsymbol{d})$ represents the gradient of f at $\boldsymbol{x}+\alpha\boldsymbol{d}$, and c_2 is a constant satisfying $c_1 < c_2 < 1$.

The curvature condition ensures that the step size does not result in excessive growth or sharp turns in the objective function. It verifies that the slope of the objective function in the new point is sufficiently aligned with the search direction. If this condition is not met, the step length is adjusted accordingly.

Algorithm 6 Wolfe Line Search Algorithm

Require: Initial solution $\boldsymbol{x}$, search direction $\boldsymbol{d}$, constants c_1 and c_2
Initialize step length α
Compute initial function value $f(\boldsymbol{x})$ and gradient $\nabla f(\boldsymbol{x})$
while Wolfe conditions are not satisfied **do**
    Compute $f(\boldsymbol{x} + \alpha\boldsymbol{d})$ and $\nabla f(\boldsymbol{x} + \alpha\boldsymbol{d})$
    if Sufficient decrease condition is not satisfied **then**
        Reduce step length: $\alpha \leftarrow \alpha/2$
    else
        if Curvature condition is not satisfied **then**
            Increase step length: $\alpha \leftarrow \alpha \times 2$
        end if
    end if
end while
Return Step length α

Advantages

1. **Balanced trade-off**: The Wolfe conditions strike a balance between ensuring a sufficient decrease in the objective function value and controlling the curvature of the function. By satisfying both conditions, the chosen step size tends to lead to convergence to a good solution.

2. **Convergence guarantee**: The strong Wolfe conditions, in particular, provide strong convergence guarantees for optimization algorithms. By imposing stricter criteria, these conditions ensure rapid convergence to a local minimum.

3. **Safety margin**: The Wolfe conditions introduce safety margins through the constants c_1 and c_2. These margins provide flexibility in selecting step sizes and help prevent overshooting or oscillation during the optimization process.

Disadvantages

1. **Computational cost**: Implementing the Wolfe conditions requires additional computations, such as evaluating the objective function and its gradient at each step length. This can increase the computational cost of the optimization algorithm.

2. **Sensitivity to constants**: The Wolfe conditions depend on the choice of the constants c_1 and c_2. Different values of these constants can lead to variations in the convergence behavior and performance of the optimization algorithm.

3. **Restrictiveness**: The strong Wolfe conditions can be overly restrictive in certain scenarios. These conditions may reject step sizes that could potentially lead to faster convergence or exploration of the solution space. In such cases, the weak Wolfe conditions can offer more flexibility.

2.5 Techniques for Stochastic Minimax Optimization

2.5.1 Formulation

We consider the problem of solving the following min max problem

$$\min_x \max_y f(x, y), \tag{2.16}$$

where $f : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$. This can be seen as a two-player game, where one agent tries to maximize the objective, whereas, the other agent tries to minimize the objective. In the following, sometimes, we may use the notation $f(x, y) = -g(x, y)$ to indicate zero sum game. Here $f(x, y)$ is the utility of the min player, and the $g(x, y)$ is the utility of the max player. A point (x^*, y^*) is defined to be a local nash equilibrium point if there exsist an epsilon ball around the point U, such that for all $(x, y) \in U$ satisfies the following

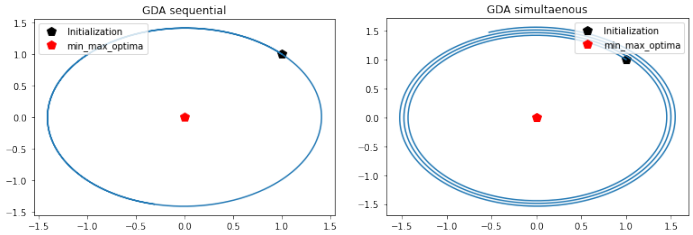
$$f(x^*, y) \leq f(x^*, y^*) \leq f(x, y^*). \tag{2.17}$$

The simple interpretation of the Equation (2.17) is that if (x^*, y^*) is a local nash equilibrium point then any perturbation in x while keeping the y fixed will only increase the function value, similarly any perturbation in y while keeping the x fixed can only decrease the function value. The authors in the paper [33] have also proposed local minimax equilibrium as an alternative to local nash equilibrium as is shown in the Equation 2.18.

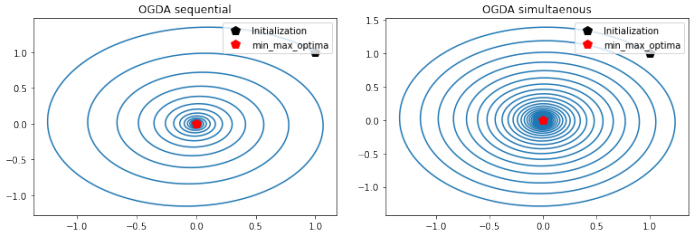
$$f(x^*, y) \leq f(x^*, y^*) \leq f(x, r(x)), \tag{2.18}$$

where $r(x)$ is an implicit function defined by $\nabla_y f(x, y) = 0$ in a neighbourhood of x^* with $r(x^*) = y^*$.

2.5.2 Optimistic Gradient Descent Ascent (OGDA)



(a) Trajectory of GDA



(b) Trajectory of OGDA

Figure 2.1: Comparison of GDA (top row) and OGDA (bottom row) techniques for the function $F(X, Y) = XY$. The top row shows sequential and simultaneous implementations of GDA, while the bottom row depicts sequential and simultaneous implementations of OGDA.

Traditional GDA (Algorithm 7 method is known to exhibit cyclic behavior around nash equilibrium, one such example is shown in Figure 2.1 (top row), to counter this The authors in the paper [13] propose the use of optimism to counter the rotations. The resulting algorithm with optimism is shown in the Algorithm 8, intuitively the past gradients are added to counteract the tangential motion, and the vector addition induces motion towards the center. This simple trick is very effective, as we can see in the Figure 2.1 (bottom row), where using OGDA, we are able to converge to nash equilibrium. The authors shows that OGDA effectively resolves the problem of limit cycling in WGAN training; they also emphasize the significant qualitative difference between GD and OGDA dynamics using illustrative toy examples, even when GD incorporates various adaptations proposed in recent literature, such as gradient penalty or momentum, and suggest that OGDA demonstrates faster regret rates in zero-sum games.

Algorithm 7 Gradient Descent Ascent

while not converged **do**
    $t \leftarrow t + 1$
    $x_{t+1} = x_t - \eta \nabla_x f(x, y)$
    $y_{t+1} = y_t + \eta \nabla_y f(x, y)$
end while

Algorithm 8 Optimistic Gradient Descent Ascent (OGDA)

while not converged **do**
    $t \leftarrow t + 1$
    $x_{t+1} = x_t - \eta \nabla_x f(x_t, y_t) + \frac{\eta}{2} \nabla_x f(x_{t-1}, y_{t-1})$
    $y_{t+1} = y_t + \eta \nabla_y f(x_t, y_t) - \frac{\eta}{2} \nabla_y f(x_{t-1}, y_{t-1})$
end while

2.5.3 Extra Gradient Method (EG)

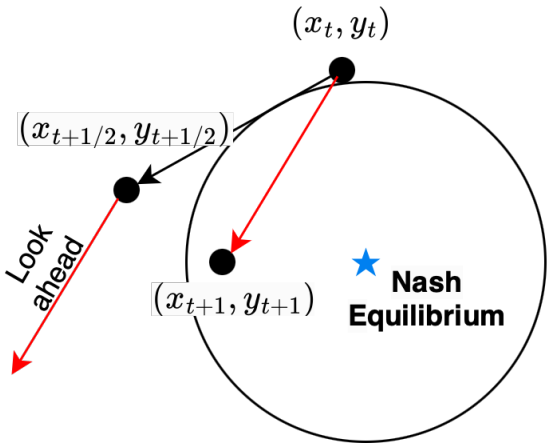


Figure 2.2: EG update

The computational complexity of EG method is twice of that of GDA, because it involves twice the gradient computations in a single update. As is shown in steps 2 and 3 of the Algorithm 9, the midpoint is computed first from the starting point (x_t, y_t) to estimate the lookahead vectors, once obtained we go back and use the lookahead vectors to perform the update on the points (x_t, y_t). As shown in Figure 2.2, it is also a first-order method that

counter the rotations of GDA. The work by Mokhtari et al. offers a thorough analysis of OGDA and EG, first-order methods designed to address the rotational challenges of GDA.

Algorithm 9 Extra Gradient Method (EG)

while not converged **do**
    $t \leftarrow t + 1$
    $x_{t+1/2} = x_t - \eta \nabla_x f(x_t, y_t)$
    $y_{t+1/2} = y_t + \eta \nabla_y f(x_t, y_t)$
    $x_{t+1} = x_t - \eta \nabla_x f(x_{t+1/2}, y_{t+1/2})$
    $y_{t+1} = y_t + \eta \nabla_y f(x_{t+1/2}, y_{t+1/2})$
end while

2.5.4 Follow the Ridge (FR)

In their paper, Wang et al. [83] presented an intriguing approach aimed at overcoming the limitations of Gradient Descent Ascent (GDA). They introduced the concept of local minimax, as defined in Equation 2.18, and assigned the roles of "leader" to the player who moves first and "follower" to the player who moves second. They have proposed the Algorithm 10(we will now refer $f(x, y)$, and $g(x, y)$ as f and g). Since a local minimax must by definition be located on a ridge, it makes sense to learn to follow the ridge, hence if a point (x_t, y_t) is on ridge, the leader in case of GDA will try to move away from the ridge by the update $\Delta x = -\eta \nabla_x f$. To counter this, the follower can then use the correcting term $\eta(\nabla_y f + (\nabla^2_{yy} f)^{-1} \nabla^2_{yx} \nabla_x)$, hence allowing both players to stay on the ridge.

Algorithm 10 Follow the Ridge (FR)

while not converged **do**
    $t \leftarrow t + 1$
    $\Delta x \leftarrow -\nabla_x f$
    $\Delta y \leftarrow \nabla_y f + (\nabla^2_{yy} f)^{-1} \nabla^2_{yx} \nabla_x f$
    $x_{t+1} \leftarrow x_t + \eta \Delta x$
    $y_{t+1} \leftarrow y_t + \eta \Delta y$
end while

2.5.5 Competitive Gradient Descent (CGD)

In the context of GDA, the strategies chosen by players for their next moves are influenced by the strategies selected by the other players in previous rounds. Various methods

Algorithm 11 Competitive gradient descent (CGD)

while not converged **do**
  $t \leftarrow t + 1$
  $\Delta x \leftarrow -\left(I - \eta^2 \nabla^2_{xy} f \nabla^2_{yx} g\right)^{-1} \left(\nabla_x f - \eta \nabla^2_{xy} f \nabla_y g\right)$
  $\Delta y \leftarrow -\left(I - \eta^2 \nabla^2_{yx} g \nabla^2_{xy} f\right)^{-1} \left(\nabla_y g - \eta \nabla^2_{yx} g \nabla_x f\right)$
  $x_{t+1} \leftarrow x_t + \eta \Delta x$
  $y_{t+1} \leftarrow y_t + \eta \Delta y$
end while

have been proposed to enhance the players' ability to predict each other's actions. These methods include following the regularized leader [73, 24], fictitious play [9], predictive updates [87], opponent learning awareness [20], and optimism [60, 13, 51]. Many of these techniques can be seen as variations of the extragradient method [Korpelevich], which shares algorithmic similarities with the approaches as mentioned above. Additionally, certain methods directly modify the dynamics of gradients to promote convergence. This can be achieved through techniques such as applying gradient penalties to encourage convergence [52], or by separating the potential convergent components from the rotational Hamiltonian components of the vector field [5, 48, 21]. In the paper "Competitive Gradient Descent (CGD)" [70], for the numerical computation of Nash equilibria of competitive two-player games, the authors present a new algorithm. Their approach is a straightforward extension of gradient descent to a two-player situation, where the update is provided by the Nash equilibrium of a regularised bilinear local approximation of the underlying game. This approach avoids the oscillatory and divergent behaviors seen in alternating gradient descent. The update rule of CGD is given by the Equation (2.19).

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = -\begin{pmatrix} I_d & \eta D^2_{xy} f \\ \eta D^2_{yx} g & I_d \end{pmatrix}^{-1} \begin{pmatrix} \nabla_x f \\ \nabla_y g \end{pmatrix}. \tag{2.19}$$

It is worth mentioning that CGD (Conjugate Gradient Descent) shares similarities with "Regularized Newton Descent." However, CGD does not include the diagonal blocks of the Hessian. If we were to incorporate all the blocks of the Hessian, the update rule would resemble Equation (2.20).

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = -\begin{pmatrix} I_d + \eta D^2_{xx} f & \eta D^2_{xy} f \\ \eta D^2_{yx} g & I_d + \eta D^2_{yy} g \end{pmatrix}^{-1} \begin{pmatrix} \nabla_x f \\ \nabla_y g \end{pmatrix}. \tag{2.20}$$

The solution to this linear system is a Nash equilibrium of the two-player game with consensus terms mentioned in Equation 2.21.

$$\begin{aligned} &\min_{x\in\mathbb{R}^m} x^T\nabla_x f + x^T\nabla^2_{xy} f y + x^T\nabla^2_{xx} f + y^T\nabla_y f + \frac{1}{2\eta}x^T x, \\ &\min_{y\in\mathbb{R}^n} y^T\nabla_y g + y^T\nabla^2_{yx} g x + y^T\nabla^2_{yy} g + x^T\nabla_x g + \frac{1}{2\eta}y^T y. \end{aligned} \tag{2.21}$$

The authors provide several reasons that are mentioned below for the impracticality of Equation (2.20), which we have also found out to be true through our experimentation.

1. **Matrix Inverse Conditioning:** Competitive gradient descent offers the advantage of bounded condition numbers for the matrix inverse in Algorithm 1. In many cases, such as zero-sum games, the condition number is upper-bounded by $\eta^2\|Dxy\|^2$. However, for non-convex-concave problems incorporating diagonal blocks of the Hessian, the matrix can become singular if $\eta\|D^2_{xx}f\| \geq 1$ or $\eta\|D^2_{yy}\| \geq 1$.

2. **Impact of Irrational Updates:** The effectiveness of the update rule (5) in reaching a local Nash equilibrium depends on certain conditions. In convex-concave problems or when $\eta\|D^2_{xx}f\|$ and $\eta\|D^2_{yy}\|$ are both less than 1, the update rule aligns with players' best strategies based on the quadratic approximation. Deviation from these conditions may lead to players adopting their worst strategies, contrary to the game interpretation of the problem.

3. **Regularity Constraints:** The incorporation of diagonal blocks of the Hessian relies on additional assumptions regarding the regularity of f.

The algorithm for competitive gradient descent is shown in Algorithm 11.

2.5.6 Implitict Competitive Regularization in GANs

The authors in the paper "Implicit Competitive Regularization in GANs" [72] propose an extension to CGD, which alone doesn't work on realistic dataset such as CIFAR (in our experiments), an extension named ACGD where adaptive learning is achieved using RMS-prop type heuristics. The improvement is observed when using both Wasserstein GAN (WGAN) loss and the original saturating GAN loss proposed by [12]. Moreover, the authors claimed that CGD imposes an implicit regularization when the minimax is

viewed as simultaneous minmax, whose solution would correspond to Nash equilibrium. Note that this also introduces mixed second derivative terms. When the generator and discriminator are trained simultaneously, a phenomenon known as implicit competitive regularisation (ICR) occurs that results in additional stable points or regions that do not appear when the generator and discriminator are trained separately using gradient descent and the discriminator is kept fixed. According to the authors, GAN is dependent on ICR and stabilises generators for whom the discriminator can only gradually reduce loss. The generators will create high-quality samples, according to the theory. Finding algorithms that provide stronger ICR than SGA is therefore desirable. For completeness, the full algorithm is shown in Algorithm 12. As was with CGD, the updates of ACGD requires solving linear system, and it can be very costly in practice.

Algorithm 12 ACGD, a variant of CGD with RMSProp-type heuristic to adjust learning rates. $\phi(\eta)$ denotes a diagonal matrix with η on the diagonal.

Require: α : Stepsize
Require: β_2 : Exponential decay rates for the second moment estimates, default value is 0.99
Require: $\max_y \min_x f(x, y)$: zero-sum game objective function with parameters x, y
Require: x_0, y_0 : ▷ Initial parameter vectors
$t \leftarrow 0$ ▷ Initialize time step
$v_{x,0}, v_{y,0} \leftarrow 0$ ▷ Initialize the 2nd moment estimate
repeat
  $t \leftarrow t + 1$
  $v_{x,t} \leftarrow \beta_2 \cdot v_{x,t-1} + (1 - \beta_2) \cdot g_{x,t}^2$
  $v_{y,t} \leftarrow \beta_2 \cdot v_{y,t-1} + (1 - \beta_2) \cdot g_{y,t}^2$
  $v_{x,t} \leftarrow v_{x,t}/(1 - \beta_2^t)$
  $v_{y,t} \leftarrow v_{y,t}/(1 - \beta_2^t)$
  $\eta_{x,t} \leftarrow \alpha/(\sqrt{v_{x,t}} + \epsilon)$
  $\eta_{y,t} \leftarrow \alpha/(\sqrt{v_{y,t}} + \epsilon)$
  $A_{x,t} = \phi(\eta_{x,t})$
  $A_{y,t} = \phi(\eta_{y,t})$
  $\Delta x_t \leftarrow -A_{x,t}^{\frac{1}{2}}(I + A_{x,t}^{\frac{1}{2}} D_{xy}^2 f A_{y,t} D_{yx}^2 f A_{x,t}^{\frac{1}{2}})^{-1} A_{x,t}^{\frac{1}{2}}(\Delta_x f + D_{xy}^2 f A_{y,t} \Delta_y f)$
  $\Delta y_t \leftarrow A_{y,t}^{\frac{1}{2}}(I + A_{y,t}^{\frac{1}{2}} D_{yx}^2 f A_{x,t} D_{xy}^2 f A_{y,t}^{\frac{1}{2}})^{-1} A_{y,t}^{\frac{1}{2}}(\Delta_y f + D_{yx}^2 f A_{x,t} \Delta_x f)$
  $x_t \leftarrow x_{t-1} + \Delta x_t$
  $y_t \leftarrow y_{t-1} + \Delta y_t$
until x_t, y_t converged

2.5.7 Consensus Optimization

Algorithm 13 Consensus gradient descent (ConOpt)

while not converged **do**
  $t \leftarrow t + 1$
  $\Delta x \leftarrow \nabla_x f - \gamma \nabla^2_{xy} \nabla_y f - \gamma \nabla^2_{xx} \nabla_x f$
  $\Delta y \leftarrow \nabla_y g - \gamma \nabla^2_{yx} \nabla_y g - \gamma \nabla^2_{xx} \nabla_x g$
  $x \leftarrow x + \eta \Delta x$
  $y \leftarrow y + \eta \Delta y$
end while

In [53], consensus gradient descent algorithm 13 was proposed. However, in the results shown in their paper, this was combined with Adam for adaptivity in the learning rate. As we can see, the primary update rule for ConOpt requires both $\nabla^2_{xy} f$ and $\nabla^2_{xx} f$ terms. The authors argue that two factors impede the effectiveness of current algorithms: i) the occurrence of eigenvalues in the Jacobian of the gradient vector field with a real-part equal to zero, and ii) eigenvalues exhibiting a significant imaginary part. The paper also uses fixed point theory framework to show that their method is convergent. Figure 2.3 presented by the authors in their paper displays the observed empirical distribution of eigenvalues for the Jacobian of the original vector field $v(x)$ associated with the simultaneous gradient ascent and the regularized vector field $w(x)$ that is given by the consensus optimization. Notably, it is evident that in proximity to the Nash equilibrium, a significant number of eigenvalues are closely aligned with the imaginary axis. Moreover, the suggested modification of the vector field employed in consensus optimization effectively shifts the eigenvalues toward the left.

2.5.8 Minimax optimization on Riemannian manifolds

In recent years, there has been a growing interest in min-max problems on Riemannian manifolds. In [90], authors introduced a generalization of Sion's minimax theorem to Riemannian manifolds and proposed the Riemannian extra-gradient method (RCEG) for solving geodesic-convex-geodesic-concave problems. Building on this work, Jordan et al. conducted a comprehensive analysis of both Riemannian gradient descent ascent (RGDA) and RCEG, focusing on geodesic-(strongly)-convex-geodesic-(strongly)-concave settings. Han et al. extended the Hamiltonian gradient methods to Riemannian min-max optimization

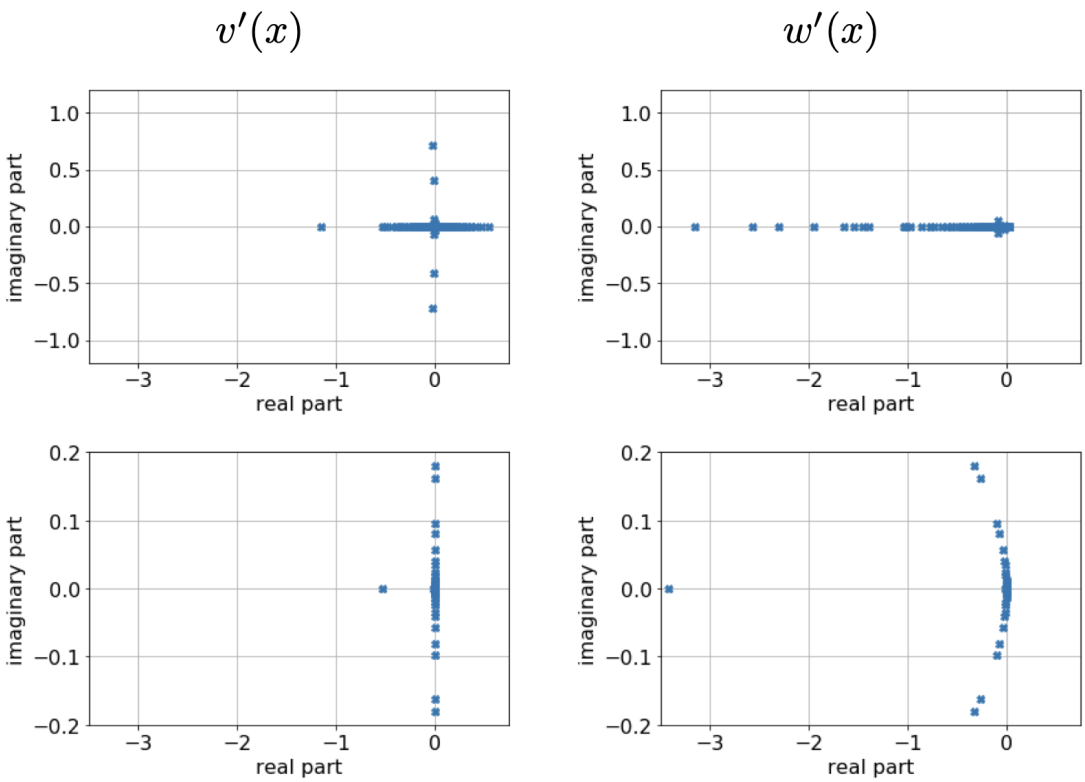


Figure 2.3: The distribution of eigenvalues before (top two) and after (bottom two) training using consensus optimization, as observed empirically [52].

and demonstrated convergence under the Riemannian PL condition on a proxy function. [29] and [84] made significant contributions. [2] has adapted several existing second-order methods on Riemannian manifolds.

2.6 Techniques for improved Deep Learning Training

2.6.1 Label smoothing

Label smoothing is a technique commonly used in deep learning to improve model generalization and reduce overfitting. It involves modifying the target labels used during training by introducing a small amount of uncertainty.

Let's consider a classification task with C classes. Normally, in a one-hot encoded representation, the ground truth label y for an input x is a vector of zeros with a single one at the index corresponding to the correct class. For example, if x belongs to class 3, $y = [0, 0, 1, 0, \ldots, 0]$.

Label smoothing replaces the one-hot encoded label with a smoothed distribution that assigns a small probability to incorrect classes. This encourages the model to be less confident and can help prevent overfitting. The smoothed label $\widetilde{y}$ is computed as follows:

$$\widetilde{y}_i = (1 - \epsilon) \cdot y_i + \frac{\epsilon}{C}, \tag{2.22}$$

where y_i is the i-th element of the one-hot encoded label y and ϵ is a small positive value representing the smoothing factor.

Label smoothing affects the choice of the loss function used for training. The commonly used cross-entropy loss needs to be modified to accommodate the smoothed labels. The cross-entropy loss with label smoothing is defined as:

$$\mathcal{L} = -\sum_{i=1}^{C} \widetilde{y}_i \log(p_i), \tag{2.23}$$

where p_i is the predicted probability for class i obtained from the model. The logarithmic term penalizes the model for assigning low probabilities to correct classes and high probabilities to incorrect classes, as guided by the smoothed labels.

Label smoothing provides several benefits in deep learning, including improved generalization, regularization, and reduced sensitivity to outliers. By introducing a small amount of uncertainty in the labels, label smoothing prevents the model from becoming overly confident and encourages it to generalize better to unseen data.

2.6.1.1 Modified Cross-Entropy Loss

The modified cross-entropy loss with label smoothing, as is mentioned in the Equation (2.22) is now modified as shown below:

$$\mathcal{L}_{\text{smoothed}} = -\sum_{i=1}^{C} \widetilde{y}_i \log(p_i).$$

To prove that label smoothing introduces a regularization factor, let's expand the modified loss function:

$$\begin{aligned}
\mathcal{L}_{\text{smoothed}} &= -\sum_{i=1}^{C} \widetilde{y}_i \log(p_i) \\
&= -\sum_{i=1}^{C} \left((1-\epsilon)\cdot y_i + \frac{\epsilon}{C}\right) \log(p_i) \\
&= -\sum_{i=1}^{C} (1-\epsilon)\cdot y_i \log(p_i) - \sum_{i=1}^{C} \frac{\epsilon}{C} \log(p_i) \\
&= (1-\epsilon)\cdot \left(-\sum_{i=1}^{C} y_i \log(p_i)\right) - \frac{\epsilon}{C}\sum_{i=1}^{C} \log(p_i) \\
&= (1-\epsilon)\cdot \mathcal{L}_{\text{original}} - \frac{\epsilon}{C}\sum_{i=1}^{C} \log(p_i).
\end{aligned}$$

We can observe that the second term, $\frac{\epsilon}{C}\sum_{i=1}^{C}\log(p_i)$, acts as a regularization factor in the loss function. It penalizes the model for assigning high probabilities to the predicted classes, encouraging the model to be less confident and reducing overfitting.

Therefore, we have mathematically proved that label smoothing introduces a regularization factor in the loss function, which helps to regularize the model and improve its generalization capabilities.

2.6.2 Early stopping

Early stopping is a popular technique used in deep learning to prevent overfitting and improve the generalization performance of a model. It involves monitoring the performance of a neural network during the training process and stopping the training when certain criteria are met. The idea is that as the model trains, it initially improves its performance on both the training and validation sets. However, at some point, the model's performance on the validation set may start to deteriorate, indicating that it is overfitting the training data

If the model is trained for too long, it may start to memorize the training data instead of learning general patterns. This leads to overfitting, where the model performs well on the training data but fails to generalize to new, unseen data. Early stopping helps

to address this issue by providing a way to determine the optimal point at which to stop training. It involves monitoring a validation set during the training process. The validation set is a separate dataset that is not used for training but is used to estimate the model's performance on unseen data.

During training, the model's performance on the validation set is evaluated periodically. The performance metric used for evaluation can vary depending on the task, but commonly used metrics include accuracy, loss, or any other relevant evaluation metric. The validation performance is tracked over multiple training iterations, and if it starts to degrade or no longer improves significantly, training is stopped.

Once early stopping triggers and training is stopped, the model's parameters at that point are typically used as the final model. These parameters strike a balance between fitting the training data well and generalizing to new data.

2.6.3 Embedding layers

The embedding layer is a fundamental component in deep learning models, particularly in natural language processing (NLP) tasks. It plays a crucial role in representing categorical variables, such as words or entities, as continuous vectors in a lower-dimensional space.

The embedding layer essentially maps discrete and high-dimensional inputs, such as words from a vocabulary, to dense vectors of fixed size. Each unique word is assigned a unique vector representation, often referred to as an embedding vector. The goal is to capture the semantic and syntactic relationships between words, enabling the model to learn meaningful patterns and generalize well to unseen data.

During the training process, the embedding layer's parameters are learned alongside the other model parameters through backpropagation. The optimization algorithm adjusts these parameters to minimize the loss function, which measures the model's performance on the task at hand. As a result, the embedding vectors gradually adapt to the context and the specific task, effectively encoding useful information about the data. The following steps illustrate the functioning of the embedding layer:

1. Initialization:

Embedding Matrix: The embedding layer initializes an embedding matrix E of size $V \times D$, where V is the vocabulary size and D is the embedding dimension. Each row of the matrix represents the embedding vector for a specific categorical variable (word).

2. Input representation:

Input sequence: Let's assume we have an input sequence of length T, where each element represents a categorical variable (word). We'll represent this sequence as $X = [x_1, x_2, \ldots, x_T]$, where x_i belongs to the range $[1, V]$, indicating the index of the word in the vocabulary.

3. Embedding lookup:

One-hot encoding: We first convert the input sequence X into a one-hot encoded matrix H of size $T \times V$, where each row corresponds to a word in the sequence. The one-hot encoding assigns a '1' to the position corresponding to the index of the word and '0' to all other positions.

Embedding lookup operation: The embedding lookup operation can be mathematically expressed as the matrix multiplication of H and E. The result is a matrix Z of size $T \times D$, where each row represents the embedding vector for a word in the input sequence.

$$Z = H \cdot E.$$

4. Training:

During training, the embedding layer learns the optimal values of the embedding matrix E through backpropagation and gradient descent. The gradients flow through the embedding layer and update the embedding vectors based on the loss function and the model's objective.

5. Embedding representation:

The output of the embedding layer is the matrix Z, which contains the dense numerical representations (embeddings) of the input sequence. These embeddings can then be passed to subsequent layers for further processing or prediction.

2.6.4 Data normalization

Data normalization is a crucial preprocessing step in machine learning and deep learning that involves transforming input data into a common scale or distribution. The goal of normalization is to ensure that different features or variables are treated equally and to prevent any particular feature from dominating the learning process. By normalizing the data, the range or distribution of values across features is adjusted, making it easier for the learning algorithm to find meaningful patterns or relationships.

2.6.4.1 Min-Max Scaling

Min-Max scaling, also known as normalization, rescales the data to a specific range, typically between 0 and 1. The normalized value x_{norm} is derived using the Equation (2.24) in case of Min-Max scaling.

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}. \tag{2.24}$$

here x is the feature value of the current sample, x_{min}, and x_{max} are the minimum, and maximum value of the feature out of all samples respectively.

2.6.4.2 Z-score Normalization

Z-score normalisation, sometimes referred to as standardisation, transforms the data so that the mean and standard deviation are both 0. The normalized value x_{norm} can be calculated using the Equation (2.25).

$$z = \frac{x - \mu}{\sigma}, \tag{2.25}$$

where z represents the standardized value, x is the original value, μ denotes the mean of the dataset, and σ represents the standard deviation of the dataset.

2.6.4.3 Decimal Scaling

$$x_{\text{norm}} = \frac{x}{10^d}. \tag{2.26}$$

In Equation (2.26), x represents the original value of the feature, and d represents the decimal shift.

2.6.4.4 Log Transformation

$$x_{\text{norm}} = \log(x). \tag{2.27}$$

In Equation (2.27), x represents the original value of the feature.

2.6.4.5 Unit Vector Scaling

$$x_{\text{norm}} = \frac{x}{\|X\|}. \tag{2.28}$$

In Equation (2.28), x represents the original value of the feature, and $\|X\|$ represents the magnitude of the vector formed by all the data points.

Chapter 3

Angle based dynamic learning rate for gradient descent

3.1 Formulation

To facilitate an adaptive and more accurate step size, we consider a snapshot in the global gradient vector field, it is well known that the gradient vectors will be perpendicular to the tangential plane of the level curve at each point of the vector field. The existing gradient descent methods go parallel to the negative gradient g_1 from the point X_1, instead, we first go perpendicular in the direction of ${g_1}^{\perp}$ towards a new point X_2 by a small h step (a hyperparameter); the direction ${g_1}^{\perp}$ is any random vector inside the tangential plane. At X_2 we calculate another negative gradient g_2. Let point O be the intersection point of these two gradients. We now simply use these information to calculate the height of the triangle as shown in the Figure 3.1 by calculating the step size $d = h \cdot \cot(\theta)$ in the step 8 of the Algorithm 14, where θ is the angle between the two vectors g_1 and g_2. We then travel

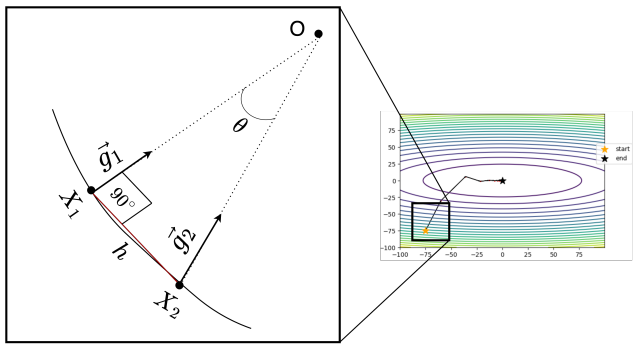


Figure 3.1: Illustration of one iteration of the proposed method.

in the direction of normalized g_1 with step size d as shown in step 13 of the algorithm 14, and essentially reach the intersection point O of the two gradients; we repeat this process until convergence.

Although the perpendicular step size is fixed and static, our learning rate changes with every new iteration. The perpendicular step h plays a critical role in our method; too large of a perpendicular step will lead g_1 and g_2 to point to different localities in non-convex settings. Hence, the calculation of step length will make no sense. On the other hand, if the value of h is too tiny, then the gradient vectors will overlap, and the angle between these gradients will be too small to compute precisely. So to tackle this issue, we keep a relatively small perpendicular step size h and add some ϵ to h because $\cot\theta$ is infinite when θ is zero. Figure 3.2 shows the resulting angle for the optimal h on different architectures. In steps 8 to 11 of algorithm 14, we use the decayed expectation (step 9) of the step size to double the current estimated step size (step 11), if the current estimated step size is lesser than the expected step size.

Algorithm 14 ASSIT-gradients: Adaptive Step Size through Intersection of Subsequent Transversed-gradients.

Input: X
Parameter: h, β

1: **while** not converged **do**
2: $X_1 \leftarrow X$ ▷ Initial random guess
3: $g_1 \leftarrow -\nabla_X f(X_1)$
4: $p_1 \leftarrow {g_1}^{\perp}$ ▷ Perpendicular unit vector to g_1
5: $X_2 \leftarrow X_1 - h \cdot p_1$
6: $g_2 \leftarrow -\nabla_X f(X_2)$
7: $\theta \leftarrow \angle(g_1, g_2) + \epsilon$
8: $d \leftarrow h \cdot \cot\theta$ ▷ Find current step size
9: $d_{avg} \leftarrow \beta \cdot d_{avg} + (1-\beta) \cdot d$ ▷ Average step size
10: **if** $d < d_{avg}$ **then**
11: $d \leftarrow 2 \cdot d$
12: **end if**
13: $X \leftarrow X_1 + d \cdot \dfrac{g_1}{\|g_1\|}$ ▷ Update step
14: **end while**

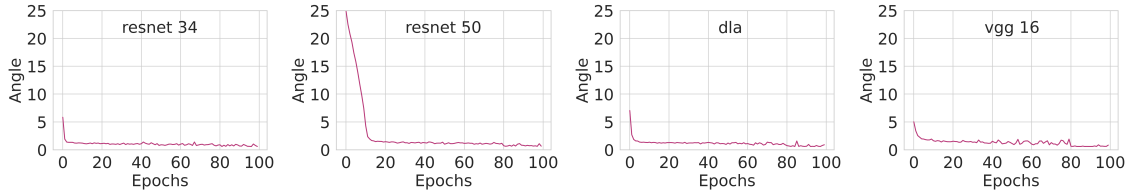


Figure 3.2: Progression of angles with increasing epochs on CIFAR 10 dataset. We observe that for most of the architectures, except of epochs less than 10, the angles are between 0-2 degrees. This is configured in such a way because too less of an angle and the cot becomes unstable, and attempting to have larger angles by having a larger h can lead to g_1 and g_2 (See Figure 14) in the algorithm 14, to lie in different localities.

3.2 Experiments on Some Toy Examples

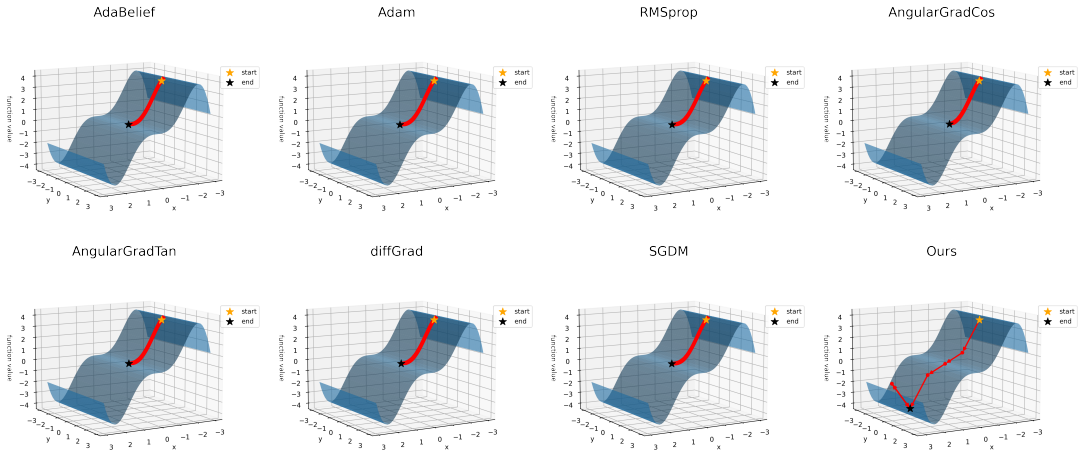
In this section, we experimentally demonstrate the potential of our method to overcome bad minima; for this, we have crafted two challenging surfaces using the tool [Joffe]. We have also shown the 3D visualization for the same in Figures 3.3a and 3.4a. For the experimental setup on these toy examples, we set the learning rate $\eta = 1e-2$, we keep the same initial point for all the methods, and we train them up to 1000 iterations.

3.2.1 Toy Example A

In the toy example A, we consider the following simple function

$$f(x, y) = -y^2 \sin x. \tag{3.1}$$

We see in Figure 3.3a that the surface corresponding to this function is wavy and has multiple local minima. The initialization point is $(-2.0, 0.0)$ as shown in the Figure 3.3a. Although our trajectory is the same as that of SGD without momentum, we can see that while all the optimizers get stuck at the relatively bad local minima, our proposed optimizer goes to a better local minimum, and it doesn't stop there; we can see that it tries to go further only to come back due to steep curvature ahead. Even though the gradient amplitude may be small in a region, since we take the unit direction of gradient to estimate the perpendicular direction, this perpendicular step is invariant to the current gradient amplitude. Hence, it can explore a region where the gradient is more substantial and use it to estimate an effective step length that may lead to further descent.



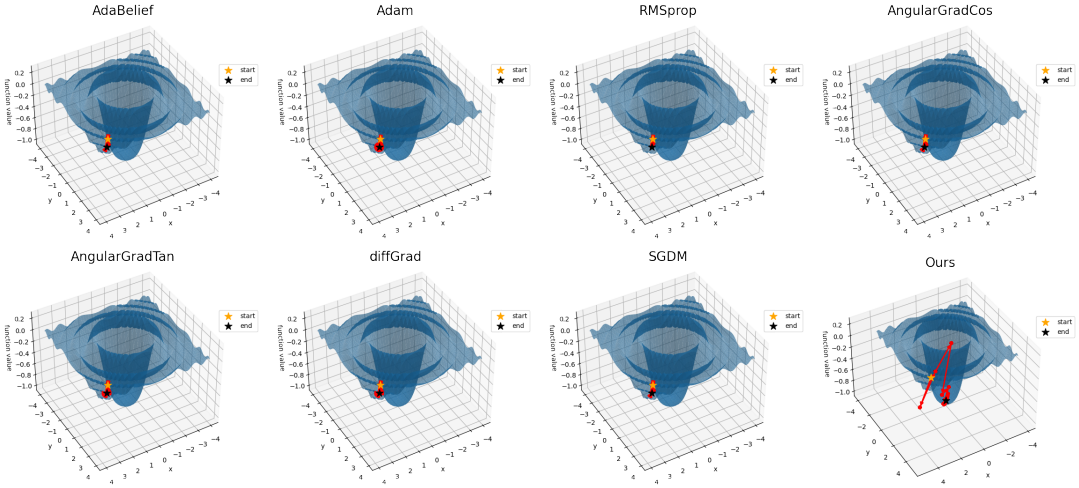
(a) Toy Example A: Our method is shown as bottom right figure. We observe that while all other methods get stuck in the local minima, our method can descents further. The black star denotes the terminal point, while the yellow star denotes the starting point.

3.2.2 Toy Example B

The toy example B is defined as follows

$$f(x, y) = -\frac{\sin\left(x^2 + y^2\right)}{x^2 + y^2}. \tag{3.2}$$

The shape of $f(x, y)$ as shown in Figure 3.4a is similar to a ripple in the water, the initialization point is kept at $(3.0, 3.0)$. We observe that other optimizers get stuck within the ripple of the function, while ours can explore further and eventually find a better minimum.



(a) Toy Example B: Here our method is shown in the bottom right. The black star denotes the terminal point, while the yellow star denotes the starting point. We observe that while all other methods cannot explore efficiently, our method reaches a better local minima.

3.3 Convergence Results

In this section, we prove convergence results. We show that with our choice of learning rate for SGD, our method converges. Moreover, later we also show that the Armijo's condition for our choice learning rate is satisfied.

Theorem 3.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function which is convex and differentiable, and let the gradient of the function be L Lipschitz continuous with Lipschitz constant $L > 0$, i.e., we have that $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ then the objective function value will be monotonously decreasing with each iteration of our method.*

Proof. The below equation gives the update rule of our method

$$y = x_t = x_{t-1} - h_{t-1} \cot \theta_{t-1} \frac{\nabla f(x_{t-1})}{\|\nabla f(x_{t-1})\|}. \tag{3.3}$$

And, let h be constrained as,

$$h_t \leq \frac{\|f(x_t)\|}{L \cot \theta_t}. \tag{3.4}$$

With assumptions on $f(x)$, we have

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} \nabla^2 f(x) \|y - x\|^2.$$

Since $\nabla f(x)$ is L-Lipschitz continuous, $\nabla^2 f(x) \leq L$

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} L \|y - x\|^2.$$

Substituting y as in equation (3.3), and x as x_{t-1}, we get:

$$\begin{aligned}
f(x_t) &\leq f(x_{t-1}) + \nabla f(x_{t-1})^T (-h_{t-1} \cot \theta_{t-1} \frac{\nabla f(x_{t-1})}{\|\nabla f(x_{t-1})\|}) \\
&\quad + \frac{1}{2} L \| - h_{t-1} \cot \theta_{t-1} \frac{\nabla f(x_{t-1})}{\|\nabla f(x_{t-1})\|} \|^2) \\
&\leq f(x_{t-1}) + \nabla f(x_{t-1})^T (-h_{t-1} \cot \theta_{t-1} \frac{\nabla f(x_{t-1})}{\|\nabla f(x_{t-1})\|}) \\
&\quad + \frac{1}{2} L h_{t-1}^2 \cot^2 \theta_{t-1}) \\
&\leq f(x_{t-1}) + h_{t-1} \cot \theta_{t-1} (\frac{L}{2} h_{t-1} \cot \theta_{t-1} - \|\nabla f(x_{t-1})\|).
\end{aligned}$$

Using equation (3.4), we get

$$\begin{aligned}
f(x_t) &\leq f(x_{t-1}) + \frac{\|\nabla f(x_{t-1})\|}{L} (\frac{\|\nabla f(x_{t-1})\|}{2} - \|\nabla f(x_{t-1})\|) \\
&\leq f(x_{t-1}) - \frac{1}{2L} (\|\nabla f(x_{t-1})\|^2).
\end{aligned}$$

Hence the objective function decreases with every iterate. □

A well-known condition on the step length for sufficient decrease in function value is given by Wolfe's condition as stated below. We will show that the first Wolfe's condition, which is Armijo's condition is satisfied, however, we show that second Wolfe's condition is not satisfied.

Theorem 3.2 (**Wolfe's Condition**). *Let α_k be the step length and p_k be the descent direction for minimizing the function $f(x) : \mathbb{R}^n \to \mathbb{R}$. Then the strong Wolfe's condition require α_k to satisfy the following two conditions:*

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k. \tag{3.5}$$

$$|\nabla f(x_k + \alpha_k p_k)^T p_k| \leq c_2 |\nabla f_k^T p_k|, \tag{3.6}$$

with $0 < c_1 < c_2 < 1$.

Proof. See [36, p. 39] for details. □

Theorem 3.3 (**First Wolfe condition**). *Under the assumptions of theorem 3.1, the update rule in Algorithm 14 satisfies the first Wolfe's sufficient decrease condition condition* (3.5) *with* $c_1 = \frac{1}{2L}$.

Proof. We recall that in Wolfe's conditions, p_k and α are the descent direction and step size, respectively. For our method in Algorithm 14, $p_k = -\nabla f(x_k)$ and $\alpha = h \cot \theta$. From theorem 3.1 we have the expression,

$$\begin{aligned} f(x_t) &\leq f(x_{t-1}) - \frac{1}{2L}(\|\nabla f(x_{t-1})\|^2) \\ &\leq f(x_{t-1}) + \frac{1}{2L} \nabla f(x_{t-1})^T p_{t-1}. \end{aligned}$$

Hence, for $c_1 = \frac{1}{2L}$ the above inequality holds true. □

Optimizer	EN-B0	EN-B0 wide	EN-B4	EN-B4 wide
SGD	59.53	59.99	**61.45**	**61.76**
ADAM	57.41	55.86	56.20	54.57
RMSPROP	57.98	56.52	57.00	55.77
DIFFGRAD	56.79	56.12	59.03	58.59
AGC	58.61	58.33	58.22	56.46
AGT	58.70	58.08	57.97	56.62
OURS	**60.21**	**60.51**	61.32	61.17

Table 3.1: Results on mini-ImageNet dataset. Since ImageNet is a relatively large dataset, we have results on EfficientNet only. We observe that our method has the best accuracy for two variants shown in **bold**, and the second best results for the other two variants shown as underlined. Despite hyperparameter tuning, most other methods except vanilla SGD perform poorly.

3.4 Numerical results

Method	RN18	RN34	RN50	D121	VGG16	DLA
SGD	93.18	93.63	93.40	93.85	92.57	93.29
Adam	93.85	93.99	93.88	94.28	92.66	93.57
RMSProp	93.63	94.07	93.42	93.84	92.36	93.29
AdaBelief	93.57	93.71	93.80	94.26	92.84	93.72
diffGrad	93.85	93.73	93.65	94.28	92.67	93.49
AGC	93.64	94.02	94.05	94.57	92.64	93.6
AGT	93.92	93.89	94.11	94.50	92.76	93.74
OURS	**94.00**	**94.24**	**94.39**	**94.75**	**93.18**	**94.38**

Table 3.2: Overall accuracies on CIFAR-10 first 100 epochs. The best results are in **bold**, and second best is underlined. We find that our method has the best accuracy among all the methods compared. Here RN stands for ResNet, D121 is DenseNet-121, and DLA is Deep Layer Aggregation.

We test the performance of the current state-of-the-art optimizers with ours on CIFAR-10 [Krizhevsky et al.], CIFAR-100 [Krizhevsky et al.], and mini-imagenet [80].

For CIFAR-10, and CIFAR-100, we take the prominent image classification architectures, these are ResNet18 [27], ResNet34 [27], ResNet50 [27], VGG-16 [75], DLA [89], and DenseNet121 [30], in the tables, we refer to these architectures as RN18, RN34, RN50,

VGG16, DLA, and D121 respectively. The current state-of-the-art optimizers are being compared with our method, i.e, Stochastic Gradient Descent with Momentum (SGDM), Adam [41], RMSprop, Adabelief, diffGrad, cosangulargrad (AGC), tanangulargrad (AGT).

To see long term result and stagnation of accuracy for CIFAR-100, a relatively large dataset, we run for 300 epochs and show accuracy results in Figure 3.7 for ResNet50, VGG-16, DenseNet-121, and DLA. Our method performs substantially better on DLA, and achieves best accuracy for resnet-50 and DenseNet on CIFAR-100.

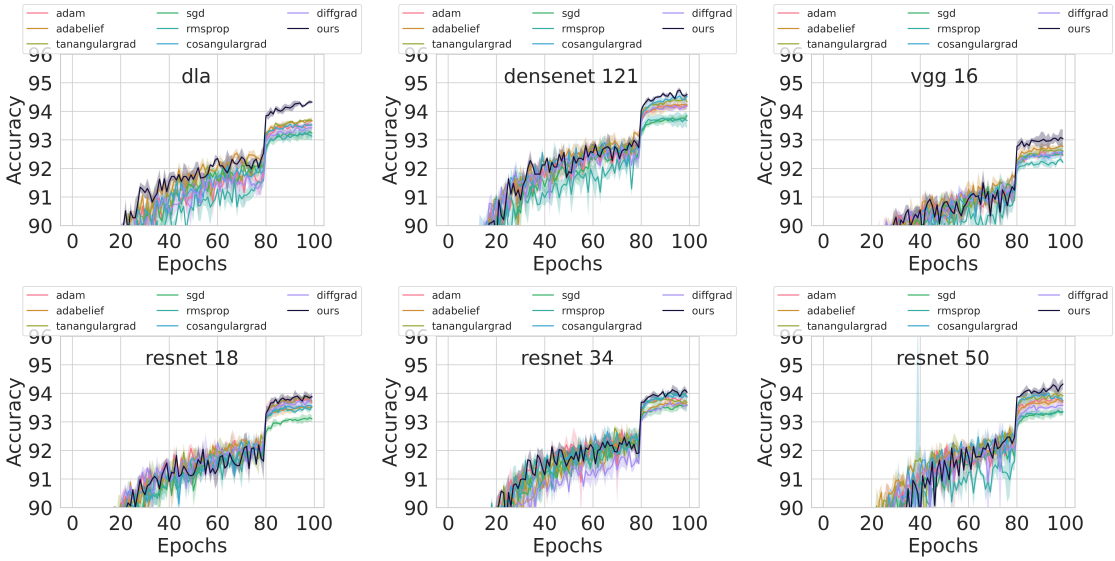


Figure 3.5: Accuracy plots versus epochs for image classification on the CIFAR-10 dataset. Our method is represented by dark purple curve, which is an average over 3 runs. The variance observed is not much around the mean. On DLA we observe a significantly better accuracy. On other architectures, our method maintains the highest accuracy.

Figure 3.5 shows accuracy versus epochs for CIFAR-10 for 3 runs and hence variations are seen around mean; we show the mean by bold lines. The best accuracy results for CIFAR-10 dataset are also shown in Table 3.2. We observe that our method performs the best across all the architectures; with more pronounced accuracy for the DLA and VGG. The observed jump in performance at epoch 80 is due to the learning rate scheduler, which is set to reduce the learning rate by 10 at the 80th epoch. The experimental results for CIFAR-100 are shown in the Table 3.3, and the plots for accuracy versus epochs for multiple runs are shown in the Figure 3.6. We perform the best in most of the architectures, except on ResNet-50 and our method performs the second best for VGG-16. Mini-imagenet [81]

dataset is a subset of the larget Imagenet dataset [15]. We use the efficient net architecture for mini-imagenet because as shown in Figure 1 in the paper [77], efficient-net provides the best accuracy per parameter. The best accuracy results of the mini-imagenet is shown in Table 3.1. We outperform all the standard state-of-the-art methods that use decaying expectations of the gradient terms (includes momentum based methods) to either scale the gradient or correct its direction. We perform best in EfficientNet-B0 and EfficientNet-B0 wide; second best in EfficientNet-B4 and EfficientNet-B4 wide while performing substantially better than the adaptive methods.

Method	RN18	RN34	RN50	D121	VGG16	DLA
SGD	72.94	73.31	73.82	74.82	69.24	73.84
Adam	71.98	71.92	73.52	74.50	67.78	71.33
RMSProp	68.90	69.66	71.13	71.74	65.79	67.91
AdaBelief	73.07	73.44	<u>75.47</u>	<u>75.60</u>	**69.63**	<u>74.09</u>
diffGrad	72.02	72.21	74.08	74.53	67.83	71.07
AGC	<u>73.25</u>	<u>73.64</u>	75.46	75.36	69.21	72.84
AGT	73.13	73.35	**75.75**	75.39	68.85	73.06
OURS	**73.33**	**74.22**	74.44	**76.79**	<u>69.46</u>	**74.92**

Table 3.3: Overall accuracies on CIFAR-100 first 100 epochs. The best results are shown in **bold**, and the second best results are <u>underlined</u>. We observe that our method has the best results on four architectures and the second best on one architecture. Here RN stands for ResNet, D121 is DenseNet-121, and DLA is Deep Layer Aggregation.

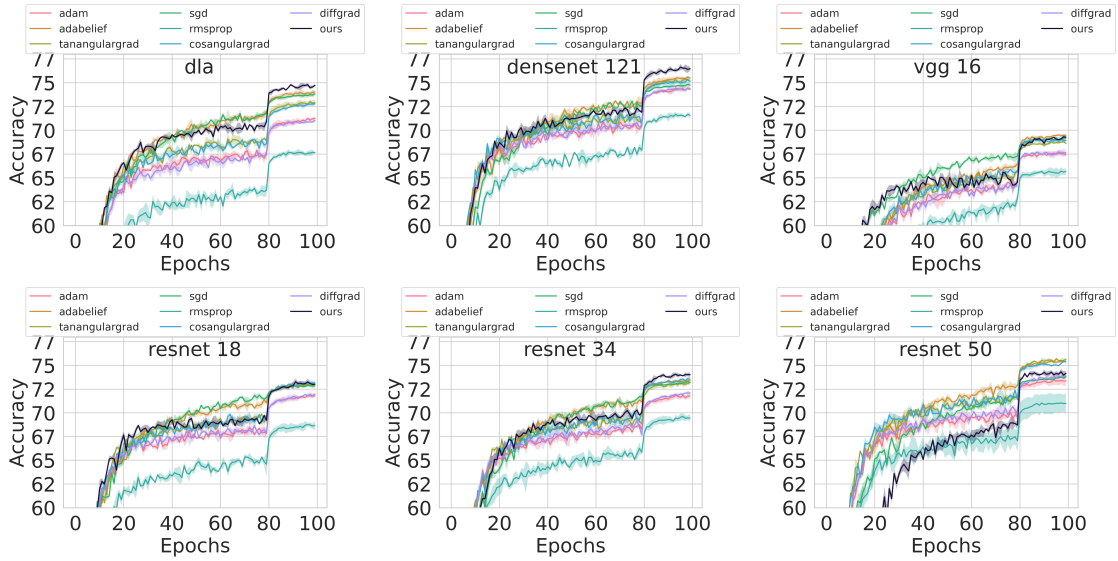


Figure 3.6: Accuracy plots versus epochs for image classification on CIFAR-100 dataset. Our method is represented by dark purple curve. Except for ResNet50 and VGG16, our method achieves the highest accuracy. The accuracy plots for longer number of epochs is shown in Figure 3.7. For hyper-parameters and ablation study, please refer to supplementary material.

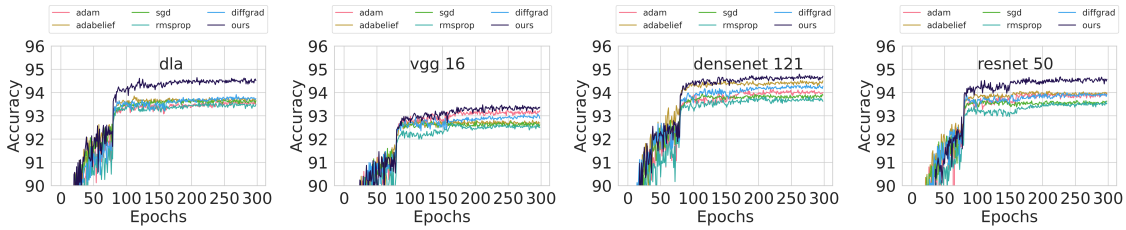


Figure 3.7: Accuracy versus epochs plot for image classification for CIFAR-100 results for 300 epochs. We verify that on long term our method maintains high accuracy, and most methods start to stagnate from epochs 100 onwards.

3.5 Code Repository

https://github.com/misterpawan/dycent.

3.6 Ablation study

As shown in figure 3.8, we provide the ablation study of our algorithm across multiple h on the CIFAR-10 dataset. Each row corresponds to a different architecture, and each column corresponds to a different step size h, decreasing as we go from left to right. For the ablation study, we take the same architectures that where in the main paper. We take the secondary hyperparameter of our method β to be 0.2, 0.4, 0.6, 0.8, and the performance related to each beta is reflected in the graph by their respective curves. The aim is to study the effects of the hyper-parameter β on different h. We also notice that for a lower learning rate, $\beta = 0.6$ dominates, while for a larger learning rate, the differences between β get less noticeable, hence any reasonable value of $\beta \in [0.2, 0.6]$ is good enough.

3.7 Experimental Setup

We adopt the code from the GitHub repository of the recent state-of-the-art paper [65]; we keep all their experimental setup the same for CIFAR-10 and CIFAR-100, i.e., keeping the batch size 128 and the same learning rate scheduler which divides the learning rate by 10 every 80th epoch; we performed three independent trials on three random seeds for each of those scripts. The training of each script is done on systems with one GeForce GTX 1080 Ti GPU, 10 CPUs, with 2G memory per CPU; each CPU is "Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz". We use the deep learning library PyTorch in our experiments, specifically PyTorch version 1.9.0. Most of the plotting is shown using matplotlib version 3.2.2; for the 3-D plots, we have used matplotlib version 3.5.0.

3.8 Hyper-parameters tuning setup

We show the hyperparameters related to our experiments in the tables below; these are the hyperparameters that gave the best performance out of the learning rates $\eta = 1e-1, 1e-2, 1e-3, 1e-4$. In each of the tables from 3.4 to 3.16, we show hyperparameters

for CIFAR-10, CIFAR-100, and Mini-imagenet for all the methods. The η_{init} represents the traditional learning rate; we have shown it as 1 in our method because we are traveling perpendicular with h. The hyperparameter α in RMSProp is the weight of the weighted average of the square of the gradients. In contrast, the hyperparameters β_1 and β_2 are the weights of the weighted average of gradient, and square of the gradient terms, respectively. While the γ is the hyperparameter for the SGD, it represents the momentum factor. We have marked NA (Not Applicable) in those entries where that particular hyperparameter doesn't apply to that respective method. The tables 3.4 to 3.9 show the hyperparameters for cifar-10 specifically, the hyperparameter for most of the methods remain unchanged, however, for the methods of AGC, AGT the stepsize oscillate between η = 1e-2, and 1e-3. And the tables 3.10 to 3.15 reflect similar trends. Hyperparameters for the Mini-imagenet are shown in the table 3.16.

ResNet-18							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGD	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.2

Table 3.4: Hyperparameter for ResNet-18 architecture.

ResNet-34							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.2

Table 3.5: Hyperparameter for ResNet-34 architecture.

ResNet-50							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.2

Table 3.6: Hyperparameter for ResNet-50 architecture.

DenseNet-121							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.2

Table 3.7: Hyperparameter for DenseNet-121 architecture.

VGG-16							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-2	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.2

Table 3.8: Hyperparameter for VGG-16 architecture.

DLA							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.2

Table 3.9: Hyperparameter for DLA architecture.

ResNet18							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.4

Table 3.10: Hyperparameter for ResNet-18 architecture.

ResNet34							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.4

Table 3.11: Hyperparameter for ResNet-34 architecture.

ResNet-50							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.4

Table 3.12: Hyperparameter for ResNet-50 architecture.

DenseNet-121							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.4

Table 3.13: Hyperparameter for DenseNet-121 architecture.

VGG-16							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-2	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.4

Table 3.14: Hyperparameter for VGG16 architecture.

DLA							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-2	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-2	NA	NA	NA	0	0.4

Table 3.15: Hyperparameter for DLA architecture.

B0, B0-Wide, B4, B4-wide							
Optimizer	η_{init}	h	α	β_1	β_2	γ	β
SGDM	1e-2	NA	NA	NA	NA	0.9	NA
ADAM	1e-3	NA	NA	0.9	0.99	0	NA
RMSprop	1e-3	NA	0.99	NA	NA	0	NA
AGC	1e-3	NA	NA	0.9	0.99	0	NA
AGT	1e-3	NA	NA	0.9	0.99	0	NA
diffGrad	1e-3	NA	NA	0.9	0.99	0	NA
Adabelif	1e-3	NA	NA	0.9	0.99	0	NA
Ours	1	1e-1	NA	NA	NA	0	0.2

Table 3.16: Hyperparameter for Mini-ImageNet

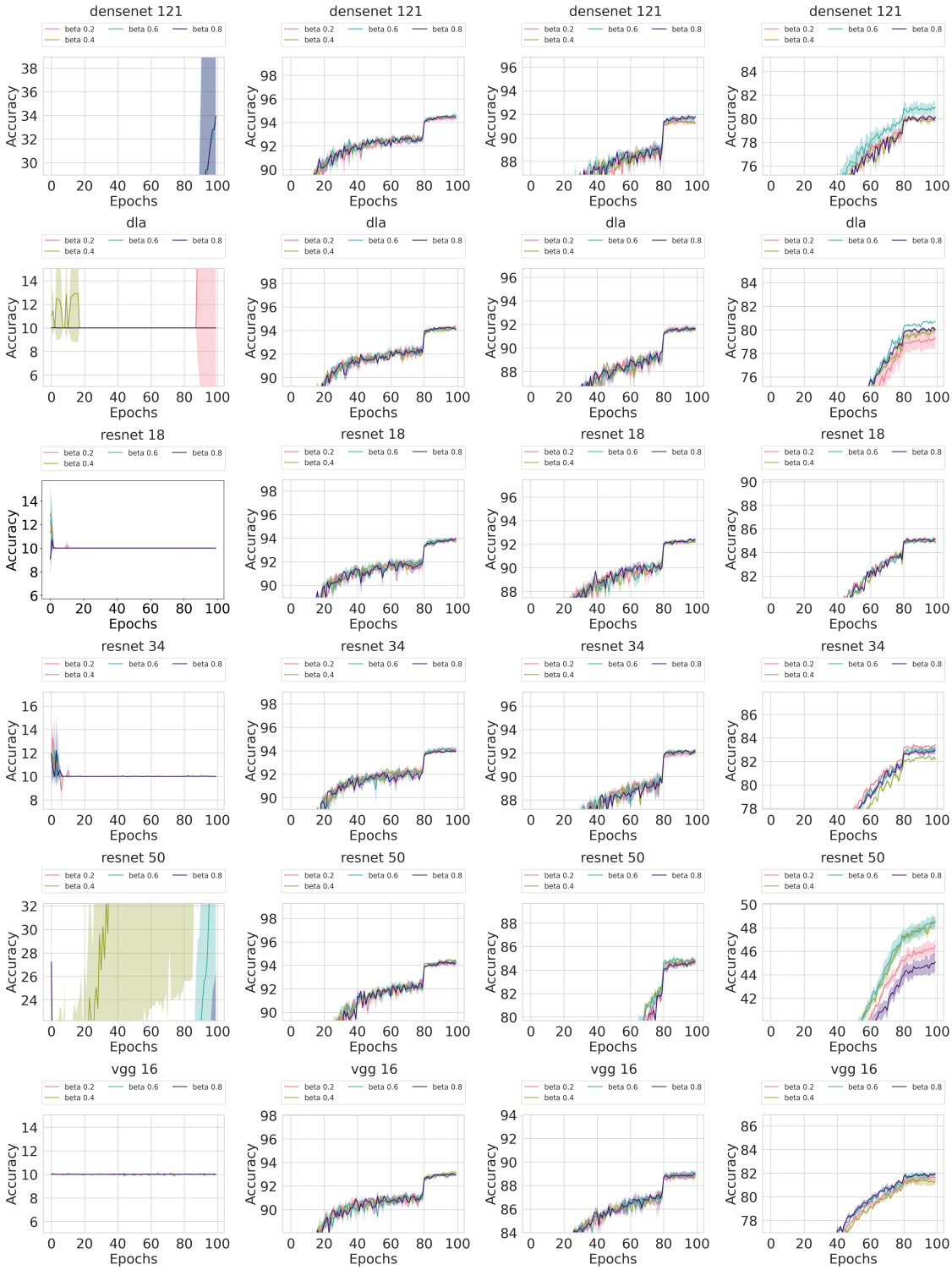


Figure 3.8: Ablation study of our method on CIFAR-10

Chapter 4

A Gauss-Newton Approach for Min-Max Optimization in GANs

4.1 Formulation

We adapt the Gauss-Newton method as a preconditioned solver for minmax problems and show that it fits within the fixed point framework [53]. As a result, we obtain a reliable, efficient, and fast first-order solver. We proceed to describe our method in detail. Consider the general form of the fixed point iterate, where $v(p)$ is the combined vector of gradients of both players at point $p = [x, y]^T$ (See Equation (4.5)).

$$F(p) = p + hA(p)v(p), \tag{4.1}$$

where h is the step size. The matrix $A(p)$ corresponding to our method is

$$A(p) = B(p)^{-1} - I, \tag{4.2}$$

where $B(p)$ denotes the Gauss-Newton preconditioning matrix defined as follows

$$B(p) = \lambda I + v(p)v(p)^T. \tag{4.3}$$

Algorithm 15 Proposed solver for min-max.

Require: $\min_x \max_y f(x, y)$: zero-sum game objective function with parameters x, y
Require: h: Step size
Require: λ: Fisher preconditioning parameter
Require: Initial parameter vectors $p_0 = [x_0, y_0]$
Require: Initialize time step $t \leftarrow 0$
Require: Compute initial gradient v_0 at p_0

1: **repeat**
2: $t \leftarrow t + 1$
3: $u_t \leftarrow \frac{1}{\sqrt{\lambda}} * v_t$
4: $z \leftarrow \frac{1}{\lambda} * \left(v_t - \frac{u_t (u_t^T v_t)}{(1 + u_t^T u_t)} \right)$
5: $\Delta \leftarrow -(v_t - z)$
6: Update

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} \leftarrow \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} + h * \Delta$$

7: **until** $p_t = [x_t, y_t]^T$ converged

With Sherman-Morison formula to compute $B(p)^{-1}$. We have

$$A(p) = \frac{1}{\lambda} \left(I - \frac{\frac{v(p) v(p)^T}{\lambda}}{1 + \frac{v(p)^T v(p)}{\lambda}} \right) - I. \tag{4.4}$$

A regularization parameter $\lambda > 0$ is added in (4.3) to keep $B(p)$ invertible. Our analysis in next section suggests to keep $\lambda < 1$ for convergence guarantee. Hence, a recommended choice is $\lambda \in (0, 1)$. To compute the inverse of Gauss-Newton matrix $B(p)$, we use the well-known Sherman–Morrison inversion formula to compute the inverse cheaply. Some of the previous second-order methods that used other approximation of Hessian require solving a large sparse linear system using a Krylov subspace solver [71, 69]; this makes these methods extremely slow. For larger GAN architectures with large model weights such as GigaGAN [37], we cannot afford to use such costly second-order methods.

4.2 Algorithm

In Algorithm 15, we show the steps corresponding to our method, and in Algorithm 16, we show our method with momentum. In the algorithms, we implement the fixed point iteration (4.1) with the choice of matrix given in (4.4) above. We use v_t to denote the gradient $v(p_t)$ computed at point $p_t = [x_t, y_t]^T$ at iteration t. We use the notation u_t in Line 3 of Algorithm 15 to denote the gradient v_t scaled by $\sqrt{\lambda}$. With these notations, our $A(p_t)$ matrix computed at p_t becomes

$$A(p_t) = \frac{1}{\lambda}\left(I - \frac{u_t u_t^T}{1 + u_t^T u_t}\right) - I.$$

Consequently, the matrix $A(p_t)v_t$ now becomes

$$A(p_t)v_t = \frac{1}{\lambda}\left(v_t - \frac{u_t u_t^T v_t}{1 + u_t^T u_t}\right) - v_t.$$

The computation of $A(p_t)v_t$ is achieved in Steps 4 and 5 of Algorithm 15. Finally, the fixed point iteration:

$$p_{t+1} = p_t + A(p_t)v_t.$$

for $p_t = [x_t, y_t]^T$ is performed in Step 6. Similarly, a momentum based variant with a heuristic approach is developed in Algorithm 16.

Algorithm 16 Adaptive solver

Require: $\min_x \max_y f(x, y)$: zero-sum game objective function with parameters x, y
Require: h: Step size
Require: β_2: Exponential decay rates for the second moment estimates
Require: λ: Fisher preconditioning parameter
Require: Initial parameter vectors $p_0 = [x_0, y_0]$
Require: Compute initial gradient v_0 at p_0
Require: Compute $\theta_0 = v_0^2$
Require: Initialize time-step: $t \leftarrow 0$

1: **repeat**
2: $t \leftarrow t + 1$
3: $\theta_t \leftarrow \beta_2 \cdot \theta_{t-1} + (1 - \beta_2) \cdot v_{t-1}^2$
4: $g_t \leftarrow v_t / (\sqrt{\theta_t} + \epsilon)$
5: $u_t \leftarrow \sqrt{\dfrac{h}{\lambda}} * g_t$
6: $z \leftarrow \dfrac{1}{\lambda} * \left(v_t - \dfrac{u_t (u_t^T v_t)}{(1 + u_t^T u_t)} \right)$
7: $\Delta \leftarrow -(g_t - z)$
8: Update

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} \leftarrow \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} + h * \Delta$$

9: **until** $p_t = [x_t, y_t]^T$ converged

4.3 Fixed point iteration and convergence

We wish to find a Nash equilibrium of a zero sum two player game associated with training GAN. For a differentiable two-player game, the associated vector field is given by

$$v(x, y) = \begin{bmatrix} \nabla_x f(x, y) \\ -\nabla_y f(x, y) \end{bmatrix}. \tag{4.5}$$

The derivative of the vector field is

$$v'(x, y) = \begin{bmatrix} \nabla_{xx}^2 f(x, y) & \nabla_{xy}^2 f(x, y) \\ -\nabla_{yx}^2 f(x, y) & -\nabla_{yy}^2 f(x, y) \end{bmatrix}.$$

The general update rule in the framework of fixed-point iteration is described in [53, Equation (16)]. Our method uses the fixed point iteration shown in (4.1). Recall that $p = (x, y)$. For the minmax problem, with minmax gradient $v(p)$ defined in (4.5), the fixed point iterative method in (4.1) well defined.

Lemma 4.1. *[53, Lemma 1 of Supplementary] For zero-sum games, $v'(p)$ is negative semi-definite if and only if $\nabla^2_{xx} f(x, y)$ is negative semi-definite and $\nabla^2_{yy} f(x, y)$ is positive semi-definite.*

Corollary 4.1. *[53, Corollary 2 in Appendix] For zero-sum games, $v'(\bar{p})$ is negative semi-definite for any local Nash equilibrium $\bar{p}$. Conversely, if $\bar{p}$ is a stationary point of $v(p)$ and $v'(\bar{p})$ is negative-definite, then $\bar{p}$ is a local Nash equilibrium.*

Proposition 4.1. *[7, Proposition 4.4.1] Let $F : \Omega \to \Omega$ be a continuously differentiable function on an open subset Ω of $\mathbb{R}^n$ and let $\bar{p} \in \Omega$ be so that*

1. *$F(\bar{p}) = \bar{p}$, and*

2. *the absolute values of the eigenvalues of the Jacobian $F'(\bar{p})$ are all smaller than 1 .*

Then, there is an open neighborhood U of $\bar{p}$ so that for all $p_0 \in U$, the iterates $F^{(k)}(p_0)$ converge to $\bar{p}$. The rate of convergence is at least linear. More precisely, the error $\left|F^{(k)}(p_0) - \bar{p}\right|$ is in $\mathcal{O}\left(|\lambda_{\max}|^k\right)$ for $k \to \infty$, where $\lambda_{\max}$ is the eigenvalue of $F'(\bar{p})$ with the largest absolute value.

The Jacobian of the fixed point iteration 4.1 is given as follows

$$F'(p) = I + hA(p)v'(p) + hA'(p)v(p). \tag{4.6}$$

At stationary point $p = \bar{p}$, $v(\bar{p}) = 0$, hence from (4.2), we have $A(\bar{p}) = (1/\lambda - 1)I$, hence, at equilibrium, the Jacobian of the fixed point iteration (4.6) reduces to

$$F'(\bar{p}) = I + hA(\bar{p})v'(\bar{p}) = I + \sigma v'(\bar{p}), \tag{4.7}$$

where

$$\sigma = h(1/\lambda - 1). \tag{4.8}$$

Lemma 4.2. *If $\bar{p}$ is a stationary point of $v(\bar{p})$ and $A(\bar{p})$ is defined as in equation (4.4), then $A(\bar{p})v'(\bar{p})$ is negative definite for some $\lambda < 1$.*

Proof.

$$A(\bar{p})v'(\bar{p}) = \left(\frac{1}{\lambda} \left(\frac{I - \dfrac{v(\bar{p})v(\bar{p})^T}{\lambda}}{1 + \dfrac{v(\bar{p})^T v(\bar{p})}{\lambda}} \right) - I \right) v'(\bar{p}).$$

At the fixed point, we have $v(\bar{p}) = 0$, hence, from above

$$A(\bar{p})v'(\bar{p}) = \left(\frac{1}{\lambda} (I) - I \right) v'(\bar{p}).$$

From Corollary 4.1, $v'(\bar{p})$ is negative definite at the local Nash equilibrium $\bar{p}$, hence, for $\lambda < 1$ the proof is complete. ■ □

At the stationary point, we do have $F(\bar{p}) = \bar{p}$, which satisfies the first item of Proposition 4.1. In the Corollary below, we show that for σ in (4.7) small enough, we satisfy the Item 2 of Proposition 4.1. To this end, we use the following Lemma.

Lemma 4.3. *[53, Lemma 4 in Supplementary] Let $U \in \mathbb{R}^{n \times n}$ only has all eigenvalues with negative real part, and let $h > 0$ (in Equation (4.8)), then the eigenvalues of the matrix $I + \sigma U$ lie in the unit ball if and only if*

$$\sigma < \frac{1}{|R(\xi)|} \frac{2}{1 + \left(\dfrac{I(\xi)}{R(\xi)} \right)^2} \tag{4.9}$$

for all eigenvalues ξ of U. Here, $I(\xi)$ and $R(\xi)$ denote the real and imaginary eigenvalues of the matrix U.

Table 4.1: Generator architecture for MNIST and FashionMNIST experiments. Here Conv1, Conv2, and Conv3 are convolution layers with batch normalization and ReLU activation, Conv4 is a convolution layer with Tanh activation.

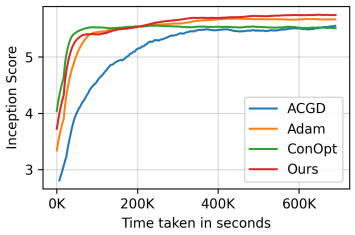
Module	Kernel	Stride	Pad	Shape
Input	N/A	N/A	N/A	$z \in \mathbb{R}^{100} \sim \mathcal{N}(0, I)$
Conv1	4×4	1	0	$100 \to 1024$
Conv2	4×4	2	1	$1024 \to 512$
Conv3	4×4	2	1	$512 \to 256$
Conv4	4×4	2	1	$256 \to 3$

Corollary 4.2. *Let $\bar{p}$ be a stationary point. If $v'(\bar{p})$ has only eigenvalues with negative real part, then the iterative method with the fixed point iteration* (4.1)*, with* (4.2) *and* (4.3) *is locally convergent to $\bar{p}$ for σ defined in* (4.7)*.*

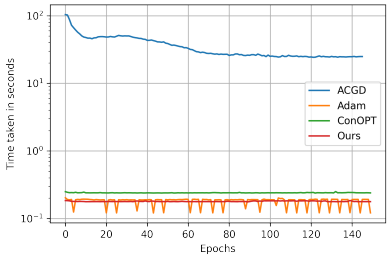
Proof. As mentioned above, $F(\bar{p}) = \bar{p}$ at stationary point $\bar{p}$ thereby satisfying item 1 of Proposition 4.1. We use Lemma 4.3 with σ same as in (4.7) and $U = v'(\bar{p})$. From corollary 4.1, we have that U has all eigenvalues with negative real part. Moreover, we can change h, and λ such that σ satisfies the upper bound (4.9), and we will have the eigenvalues of $F'(\bar{p}) = I + \sigma U = I + \sigma v'(\bar{p})$ lie in a unit ball, hence, we also satisfy item 2 of Proposition 4.1. Thereby, we have local convergence due to Proposition 4.1. ■ □

Table 4.2: Discriminator architecture for MNIST and FashionMNIST experiments. Here Conv1 is a convolution layer with LeakyReLU activation, Conv2, and Conv3 are convolution layers with batch normalization and ReLU activation, Conv4 is a convolution layer with sigmoid activation.

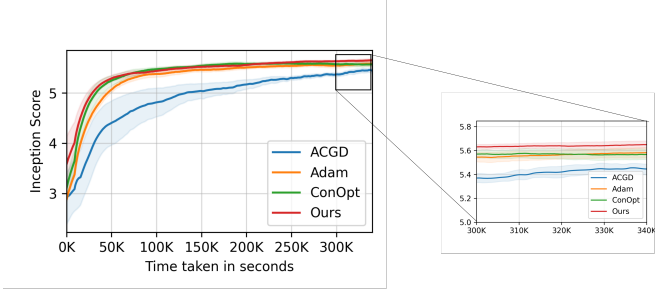
Module	Kernel	Stride	Pad	Shape
Input	N/A	N/A	N/A	$z \in \mathbb{R}^{100} \sim \mathcal{N}(0, I)$
Input	N/A	N/A	N/A	$x \in \mathbb{R}^{3 \times 32 \times 32}$
Conv1	4×4	2	1	$3 \to 256$
Conv2	4×4	2	1	$256 \to 512$
Conv3	4×4	2	1	$512 \to 1024$
Conv4	4×4	1	0	$1024 \to 1$



(a) CIFAR10: Inception score.



(b) CIFAR10: Time comparison.



(c) CIFAR10: Variance in inception scores for multiple runs.

Figure 4.1: Results for CIFAR10.

4.4 Numerical results

4.4.1 Experimental setup

In the following, we refer to Algorithm 16 as "ours". We compare our proposed method with ACGD [71] (a practical implementation of CGD), ConOpt [53], and Adam (GDA with gradient penalty and Adam optimizer). ACGD and ConOpt are second-order optimizers, whereas Adam is a first-order optimizer. We do not compare with other methods including SGA [4] and OGDA [14] as ACGD [71] has already been shown to be better than those.

All experiments were carried out on an `Intel Xeon E5-2640` v4 processor with 40 virtual cores, 128GB of RAM, and an `NVIDIA GeForce GTX 1080 Ti` GPU equipped with 14336 CUDA cores and 44 GB of `GDDR5X VRAM`. The implementations were done using `Python 3.9.1` and `PyTorch (torch-1.7.1)`. Model evaluations were performed using the Inception score, computed with the `inception_v3` module from `torchvision-0.8.2`. The WGAN loss with gradient penalty was employed. We tuned the methods on a variety of hyperparameters to ensure optimal results, following the recommendations provided in

Table 4.3: Generator architecture for CIFAR10 experiments. Here Conv1, Conv2, and Conv3 is a convolution layer with batch normalization and ReLU activation, Conv4 is a convolution layer with Tanh activation.

Module	Kernel	Stride	Pad	Shape
Input	N/A	N/A	N/A	$z \in \mathbb{R}^{100} \sim \mathcal{N}(0, I)$
Conv1	4×4	1	0	$100 \to 1024$
Conv2	4×4	2	1	$1024 \to 512$
Conv3	4×4	2	1	$512 \to 256$
Conv4	4×4	2	1	$256 \to 3$

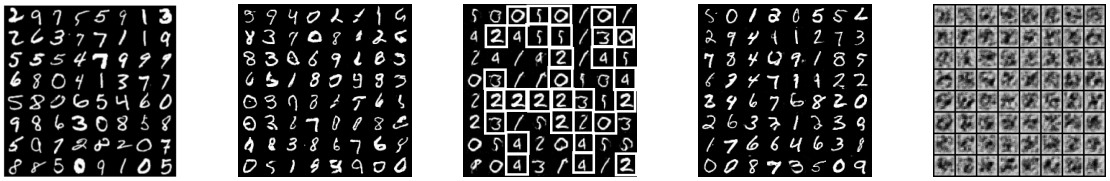
the official GitHub repositories or papers. For ACGD, we used the code provided in the official GitHub repository as a solver (`https://github.com/devzhk/cgds-package`), and followed their default recommendation for the *maxiter* parameter. The step size h was tested across multiple values $(1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 8e-5, 2e-4)$, and the gradient penalty constant λ_{gp} was varied as $(10, 1, 1e-1, 0)$. The best result for ACGD in our experimental setup was obtained with $h = 1e-4$ and $\lambda_{\text{gp}} = 1$. In the case of ConOpt, we tried the step size h across $1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 2e-5, 8e-6$, varied λ_{gp} as $(10, 1, 1e-1, 0)$, and adjusted the ConOpt parameter γ across $10, 1, 1e-1$. ConOpt performed best in our setup with $h = 1e-5$, $\lambda_{\text{gp}} = 1$, and $\gamma = 1e-1$. For Adam, we tested the same step sizes as with ConOpt and used the default parameter values recommended by PyTorch. Adam's best performance was achieved with a step size of $lr = 1e-5$. Our proposed method performed best with regularization parameter $\lambda = 1e-1$, step size $h = 1e-5$, and gradient penalty constant for WGAN $\lambda_{\text{gp}} = 10$.

4.4.2 Image generation on grey scale images

We investigate grayscale image generation using public datasets MNIST [47] and FashionMNIST [86], with the discriminator and generator architectures shown in Tables 4.1 and 4.2, respectively. In Figure 4.3, we present the MNIST dataset comparison. A careful inspection of the generated images reveals that the first-order optimizer Adam is capable of producing high-fidelity images closely resembling the original dataset shown in Figure 4.3a. Nonetheless, Adam's drawback is its tendency to generate images with characters of the same mode/class, such as 0 and 2, which might indicate mode-collapse, while also maintaining a uniform style across all images in Figure 4.3c. We highlighted the similar

Table 4.4: Discriminator architecture for CIFAR10 experiments. Here Conv1 is a convolution layer with LeakyReLU activation, Conv2 and Conv3 is a convolution layer with batch normalization, and LeakyReLU activation, Conv4 is a convolution layer with sigmoid activation.

Module	Kernel	Stride	Pad	Shape
Input	N/A	N/A	N/A	$z \in \mathbb{R}^{100} \sim \mathcal{N}(0, I)$
Input	N/A	N/A	N/A	$x \in \mathbb{R}^{3\times32\times32}$
Conv1	4×4	2	1	$3 \to 256$
Conv2	4×4	2	1	$256 \to 512$
Conv3	4×4	2	1	$512 \to 1024$
Conv4	4×4	1	0	$1024 \to 1$



(a) Real (b) Ours (c) Adam (d) ACGD (e) ConOpt

Figure 4.3: Images generated for MNIST. Samples inside white box show mode-collapse.

samples with white box illustrating mode-collapse for Adam. The second-order optimizer ACGD offers an alternative solution to this issue, as shown in Figure 4.3d. Although the images contain some minor artifacts, they display high quality. ACGD mitigates the consistent character style observed in first-order optimizers, enabling increased diversity in generated images while preserving visual fidelity to the original dataset. Our proposed method balances the advantages of both Adam and ACGD. By utilizing this approach, we can generate characters in 4.3b that closely mimic the original dataset and remain computationally efficient, similar to Adam, a first-order optimizer. Although some minor artifacts may be present in the generated images, our method overall provides a promising solution achieving high-quality and diverse results. As depicted in Figure 4.4, the generation of FashionMNIST images displays a trend similar to that observed in MNIST. For grey-scale images, ConOpt's performance is found to be suboptimal. We note that the ConOpt paper [53] did not present results for grey-scale images; thus, we acknowledge that we may not know the appropriate hyperparameters for these datasets.

(a) Real (b) Ours (c) Adam (d) ACGD (e) ConOpt

Figure 4.4: Images generated for Fashion MNIST. Samples inside white box show mode-collapse.

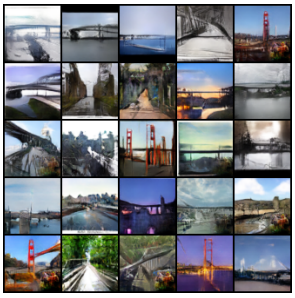
4.4.3 Image generation on color images

For color image datasets, we examine the well-known and widely-used datasets `CIFAR10` [44], `FFHQ` [39], and `LSUN` bridges, tower, and classroom datasets [88]. A comparison for the `CIFAR10` dataset [44] is displayed in Figure 4.1. Our method generates images of comparable quality to those produced by ACGD. A widely accepted metric for comparing methods on the `CIFAR10` dataset is the inception score. We analyze the inception scores of ACGD, Adam, ConOpt, and our method in Figure 4.1a and Figure 4.1c. The former demonstrates the performance over extended training durations, while the latter illustrates the performance variations across multiple runs. The generator and discriminator architectures for `CIFAR10` are presented in Tables 4.3 and 4.4, respectively. Table 4.5a displays the average time taken per epoch, while Table 4.5b showcases the maximum inception scores across multiple runs. Our method achieves the highest inception score of 5.82, followed closely by Adam with a score of 5.76, and then ConOpt and ACGD.

ACGD has a long runtime due to the costs involved in solving linear systems. However, it may achieve a higher inception score if allowed to run for an extended period. To investigate this, we conducted experiments for a long runtime of approximately eight days, as illustrated in Figure 4.1a. We used the same experimental setting as ACGD, including testing different values of GP and a case where GP was not used. For our architecture, ACGD gave better results than the results reported on a smaller architecture in ACGD paper; as shown in [69, Figure 9]. However, we find that Adam demonstrated better performance than ACGD on this architecture.

Figure 4.6a displays our generated images of FFHQ, created using a training dataset of FFHQ thumbnails downscaled to 64x64x3 dimensions. Our images demonstrate a high level of detail and texture in the skin and realistic facial features. Figure 4.6b shows our results on the LSUN classroom dataset, where our model successfully learned significant

(a) LSUN-tower.



(b) LSUN-bridges.

Figure 4.5: Images generated by our method on LSUN-tower and LSUN-bridges.

features, capturing the essence of classroom scenes. In Figure 4.5, the generated images of LSUN-tower and LSUN-bridges exhibit recognizable characteristics, such as the distinct shape of a tower and the features typical of bridges.

(a) FFHQ.

(b) Classroom.

Figure 4.6: Images generated by our method on FFHQ and LSUN-classroom.

Table 4.5: Timings and inception scores across multiple runs.

Method	Average time per epoch
ACGD	35.34
Adam	0.186
ConOpt	0.25
Ours	0.184

(a) Average time in seconds.

Method	Run1	Run2	Run3
ACGD	5.55	5.52	5.60
Adam	5.38	5.65	5.58
ConOpt	5.73	5.72	5.76
Ours	**5.82**	**5.77**	**5.79**

(b) Max inception scores across runs.

4.5 Computational complexity and timing comparisons

Assuming $x \in \mathbb{R}^m, y \in \mathbb{R}^n$, then due to the terms ∇^2_{xy}, and ∇^2_{yx} involved in ConOpt, there is additional computational cost of the order $O(m^2n^2)$, $O(m^3)$ and $O(n^3)$, these costs are associated with constructing these terms and for matrix-vector operation required. On the other hand, for ACGD, there are additional cost of order $O(m^3n^3)$ for solving the linear system with matrix of order $mn \times mn$ if direct method is used, and of order $O(mn)$ if iterative methods such as CG as in [71] is used. For empirically verifying the time complexity, for `CIFAR10` dataset, in Figure 4.1b, we observe that ACGD is the slowest, and ConOpt is also costlier than our method. Our method has similar complexity to that of Adam the first order methods. To compute the order of complexity of Algorithm 15, we have $m + n$ cost for scaling, $2(m + n)$ cost for $u(u^T g_t)$ computation, and $m + n$ cost for $u^T u$ computation, then subtraction with g_t costs $(m + n)$. The total cost per epoch is $O(m + n)$, which is same order as any first-order method such as Adam.

Table 4.1b displays a comparison of the time taken per epoch for various methods, including ACGD. It is noteworthy that ACGD consumes a significantly greater amount of time compared to other methods. Specifically, ACGD takes 35 seconds per epoch, which is approximately 200 times more than our method that takes only 0.184 seconds per epoch. Notably, this is almost identical to the time taken by Adam, which is 0.186 seconds per epoch.

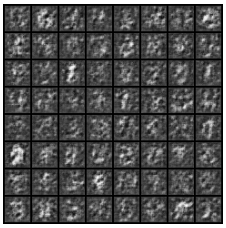
Chapter 5

Other explorations

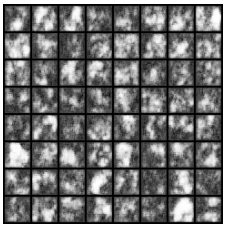
5.1 An attempt to overcome momentum

The majority of optimizers discussed in the Generative Adversarial Network (GAN) literature incorporate techniques such as momentum or gradient scaling. However, through our experiments, we have discovered that when such components are removed from the optimizer, these models fail to generate satisfactory results. In an effort to address this limitation, we endeavored to explore alternative methods that could effectively replace the use of momentum during training. Our experimentation revealed that the generator's loss function exhibited significant instability, while the discriminator's loss function quickly approached zero.

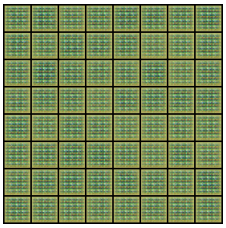
To mitigate this issue, we use a form of regularization known as label smoothing (LS). Previous chapter has demonstrated that LS introduces regularization effects. As depicted in Figure 5.1, the top row illustrates the generated images from a training session conducted without momentum and LS, showcasing highly noisy outputs. In contrast, the bottom row demonstrates that, specifically for black and white datasets, the generator was able to generate visually appealing images. Notably, for MNIST, the generated images bear a striking resemblance to the original dataset, while for FMNIST, although the image quality is subpar, it is an improvement over generating nothing. It should be acknowledged, however, that even after the application of LS, the generator continued to produce only noise when confronted with colored datasets such as CIFAR.



(a) CGD without LS - MNIST



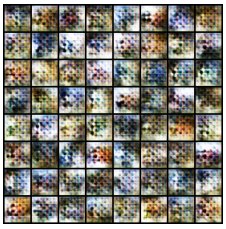
(b) CGD without LS - FMNIST



(c) CGD without LS - CIFAR

(d) CGD with LS - MNIST

(e) CGD with LS - FMNIST



(f) CGD with LS - CIFAR

Figure 5.1: Performance comparison between CGD with and without label smoothing(LS) on MNIST, Fashion MNIST, and CIFAR datasets.

5.2 Implicit Label Smoothing Regularization (ILSR)

We tried to train the GAN in a novel way as in shown in the Figure 5.2, where we are incorporating the label smoothing regularization(LSR) implicitly. We do this by conditioning the generator on the smoothed labels.

Instead of providing a fixed label, we are providing soft label from a random distribution on the interval $[l_1, l_2)$, where l_1, and l_2 is the lower, and upper label limit respectively. reflects the situation where no label smoothing is taking place.

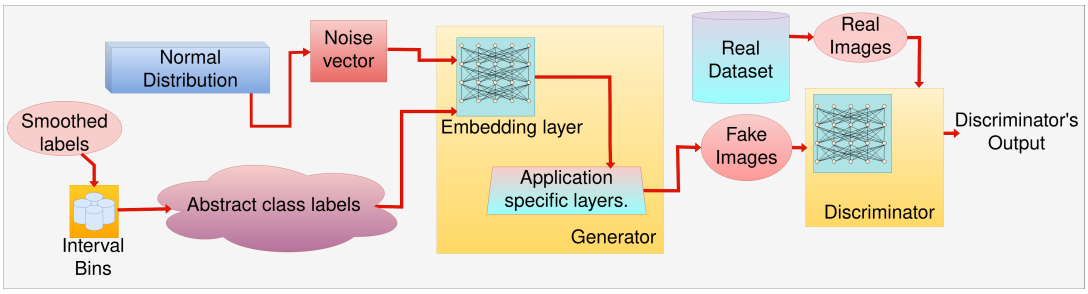


Figure 5.2: Implicit label smoothing supported generator

To condition the generator on these smoothed labels, we first obtain $\mathcal{L}$, which we call abstract class label using the smoothed labels via the 18 in the step 4 of the Algorithm 17, and then we feed these $\mathcal{L}$ into an embedding layer within the step 6 of Algorithm 17 to obtain a fresh batch of fake images, and then we train the GAN normally using these fake images.

Algorithm 17 Implicit label smoothing regularization

Require: l_1 , l_2, $batch\_size$, $abstract\_classes$, and λ.

1: $initial\_gap \leftarrow l_2 - l_1$
2: **for** $epoch \leftarrow 1$ to max_epochs **do**
3: $label \leftarrow (l_2 - l_1) * r(batch\_size) + l_2$ ▷ $r(x)$ is a uniform distribution on the interval [0,1)
4: $abstract\_class\_labels \leftarrow$ LABEL_GEN($label$)
5: $real\_images \leftarrow dataset(batch\_size)$
6: $fake\_images \leftarrow generator(abstract\_class\_labels, noise)$
7: Train generator using this $fake\_images$, and train discriminator normally.
8: **end for**

Algorithm 18 label_gen

Require: $label, l_1, l_2, abstract\_classes$

1: $interval \leftarrow [l_1, l_2]$
2: $n \leftarrow length(abstract\_classes)$
3: Divide the intervals in n equal bins, and assign each bin a unique value incrementally.
4: $abstract\_class\_labels \leftarrow []$
5: **for** $i \leftarrow 1$ to $length(label)$ **do**
6: $\mathcal{L}_i \leftarrow bin\_value[label_i]$
7: **end for**
8: **return** $\mathcal{L}$

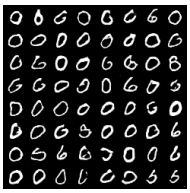
5.3 Disentangled representation learning

Disentangled representation is necessary to learn the general representation of the given distribution, a generator that learns a disentangled representation provides with more diverse images not only among different classes, but also within the same class. It is also responsible for allowing generator to learn salient features of the dataset.

Authors in [6] have experimentally agreed that a good disentangling leads to better yield of samples, and better mixing between the mode. We have witnessed something similar in our experiments.

In Figure 5.3 we can see that on fixing the abstract class label $\mathcal{L}$ to a specific value, we obtain a range of diverse images which have been learned as an umbrella representation under that $\mathcal{L}$ within the embedding layer. We can also witness mixing of modes pertaining to similar characteristic between the samples, this is a mark of generator which has successfully learned deeper representation due to disentanglement.

For example, The samples in 5.3a have very similar salient features, where we can observe similar classes are being learned as an umbrella representation under $\mathcal{L} = 0$, particularly samples very similar to the digit class 0,6,8. And even within a particular digit class we witness more diverse options among the sample where the thickness, height, roundness and alignment are differing. Hence an effective disentanglement has been achieved. This is also true for other values of $\mathcal{L}$ as can be seen in the other subfigures under figure 5.3.



(a) $\mathcal{L}$ fixed to 0

(b) $\mathcal{L}$ fixed to 1

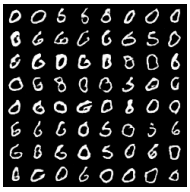
(c) $\mathcal{L}$ fixed to 2

(d) $\mathcal{L}$ fixed to 3

(e) $\mathcal{L}$ fixed to 4

(f) $\mathcal{L}$ fixed to 5

(g) $\mathcal{L}$ fixed to 6



(h) $\mathcal{L}$ fixed to 7

(i) $\mathcal{L}$ fixed to 8

(j) $\mathcal{L}$ fixed to 9

Figure 5.3: Good diversity, and mixing of modes from our method which is due to better disentangling

(a) MNIST (b) Fashion MNIST (c) CIFAR

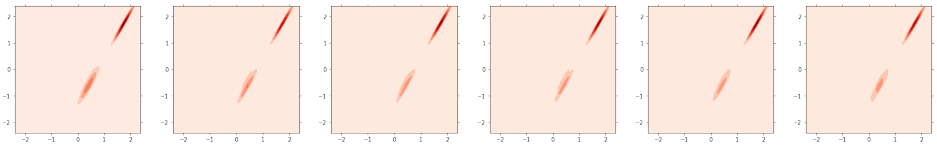
Figure 5.4: Performance of ILSR on MNIST, FMNIST, and CIFAR

5.4 Inducing Mode limiting with the Embedding Layer

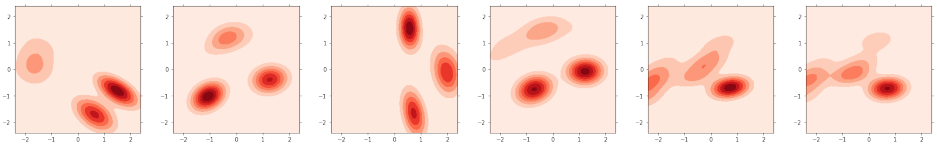
In this section we show that this setup is indeed conditioning the generator in a way that can limit the maximum number of modes learned during the training.

For this we are increasing the latent space of the embedding layer to 200, and changing the dictionary size of the embedding layer to see if we can limit the number of nodes produced by generator using this setup. This might not be ideal for a practical scenario where we would like to take into consideration the effect that noise could have had to train a higher quality GAN, instead this demonstration is just to reflect that the system is indeed getting conditioned by our method, and that the embedding layer needs at least the same number of rows as there are number of classes in the real distribution, i.e for MNIST where the number of classes are 10 (digits [0-9]) we need at least 10 rows in the embedding layer.

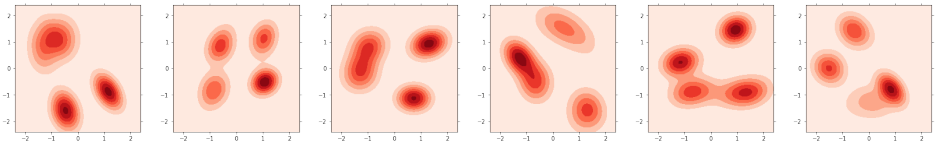
When limit the number of rows in the embedding layer to less than 8 for the 8-mode Gaussian distribution, in Figure 5.5 we witness that the maximum learned modes are exactly equal to the number of rows, hence effectively limiting the generator's capability to learn, only when we set the number of rows to 8 the generator is capable of producing all 8 modes. Hence this gives a clear intuition that the embedding rows must at least equal the number of classes in the real distribution, to learn all the modes effectively.



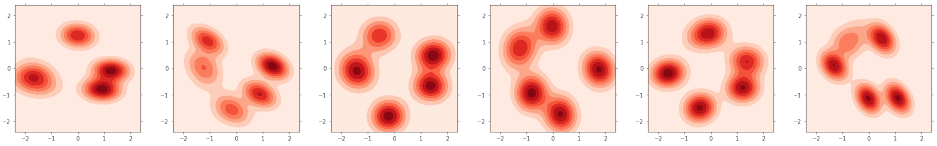
(a) Training output when the number of rows of the embedding layer is restricted to 2.



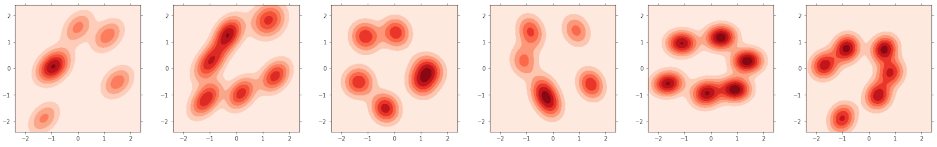
(b) Training output when the number of rows of the embedding layer is restricted to 3.



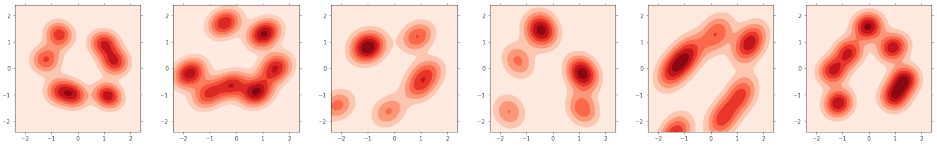
(c) Training output when the number of rows of the embedding layer is restricted to 4.



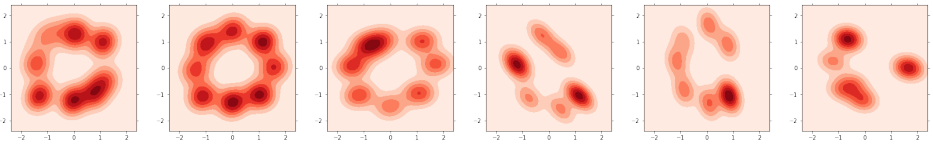
(d) Training output when the number of rows of the embedding layer is restricted to 5.



(e) Training output when the number of rows of the embedding layer is restricted to 6.



(f) Training output when the number of rows of the embedding layer is restricted to 7.



(g) Training output when the number of rows of the embedding layer is restricted to 8.

Figure 5.5: Learning can be restricted to few modes when number of rows of the embedding layers are restricted.

Chapter 6

Conclusion and Future Work

In conclusion, this thesis has explored and contributed to the field of deep learning optimization by investigating adaptive learning rates, first-order optimization for GANs, and the effects of label smoothing.

Our novel approach for obtaining adaptive learning rates surpasses traditional methods by leveraging the angle between gradients, leading to superior accuracy compared to state-of-the-art optimizers. We have demonstrated convergence and evaluated the approach on diverse benchmark datasets using prominent image classification architectures.

The introduction of our first-order optimization method for GANs, based on the modified Gauss-Newton method, has shown remarkable results. Our method generates high-fidelity images with enhanced diversity across multiple datasets, outperforming second-order methods and achieving the highest inception score for CIFAR10. Additionally, the execution time of our method is comparable to that of first-order min-max methods.

Through an investigation of label smoothing effects on GAN training, we have highlighted the significance of employing label smoothing with a high learning rate and the CGD optimizer. The results surpass those obtained with ADAM at the same learning rate, demonstrating the crucial role of label smoothing in achieving superior quality. We have also explored architectural changes and their impact on the conditioning of the generator, providing valuable insights into optimizing GAN performance.

Overall, our research has made significant contributions to the field of deep learning optimization. Our proposed methodologies for adaptive learning rates, first-order optimization for GANs, and the importance of label smoothing offer improved accuracy in classification tasks, enhanced image generation quality, and a deeper understanding of the intricacies involved in GAN training. These advancements pave the way for further advancements and

applications in the field of deep learning optimization. The following endeavors present potential avenues for future work based on the findings of this thesis:

1. **Generalization of the angular framework**: Expanding the application of the proposed adaptive learning rate approach to other optimization methods such as Adam, RMSprop, and exploring its efficacy in tasks beyond image classification, such as natural language processing. This would involve adapting the angular framework to different domains and analyzing its performance in diverse scenarios.

2. **Refinement and extension of the Gauss-Newton framework**: Further investigation of the proposed first-order optimization method for GANs by incorporating a detailed analysis of convergence rate. Conducting experimental evaluations on larger architectures to assess the scalability and generalizability of the method. This would provide a more comprehensive understanding of the method's performance and its potential applications in various GAN settings.

3. **Theoretical justification for label smoothing in GAN training**: Incorporating theoretical insights that support and elucidate the benefits of label smoothing in GAN training. Considering the widespread usage of momentum-based solvers in modern GAN optimization, exploring theoretical foundations can enhance our understanding of the role and impact of label smoothing. This theoretical justification would provide a stronger basis for the practical adoption of label smoothing techniques in GAN models.

By pursuing these future research directions, we can build upon the current thesis and advance the field of deep learning optimization, opening up new possibilities for improved optimization methods, expanded applications, and theoretical justifications.

Bibliography

[1] Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., et al. (2016). Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182. PMLR.

[2] Anonymous (2023). Nonconvex-nonconcave min-max optimization on riemannian manifolds. *Submitted to Transactions on Machine Learning Research*. Under review.

[3] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[4] Balduzzi, D., Racaniere, S., Martens, J., Foerster, J., Tuyls, K., , and Graepel, T. (2018a). The mechanics of n-player differentiable games. *arXiv:1802.05642*.

[5] Balduzzi, D., Racaniere, S., Martens, J., Foerster, J., Tuyls, K., and Graepel, T. (2018b). The mechanics of n-player differentiable games. In *International Conference on Machine Learning*, pages 354–363. PMLR.

[6] Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2013). Better mixing via deep representations. In *International conference on machine learning*, pages 552–560. PMLR.

[7] Bertsekas, D. (1999). *Nonlinear Programming*. Athena Scientific.

[8] Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

[9] Brown, G. W. (1951). Iterative solution of games by fictitious play. *Act. Anal. Prod Allocation*, 13(1):374.

[10] Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. (2020). Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

[11] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537.

[12] Cortes, C., Lawarence, N., Lee, D., Sugiyama, M., and Garnett, R. (2015). Advances in neural information processing systems 28. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems*.

[13] Daskalakis, C., Ilyas, A., Syrgkanis, V., and Zeng, H. (2017). Training gans with optimism. *arXiv preprint arXiv:1711.00141*.

[14] Daskalakis, C., Ilyas, A., Syrgkanis, V., and Zeng, H. (2018). Training gans with optimism. *ArXiv*, abs/1711.00141.

[15] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[16] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[17] Dozat, T. (2016). Incorporating nesterov momentum into adam. *ICLR Workshop*, 1:2013–2016.

[18] Dubey, S. R., Chakraborty, S., Roy, S. K., Mukherjee, S., Singh, S. K., and Chaudhuri, B. B. (2019). diffgrad: an optimization method for convolutional neural networks. *IEEE transactions on neural networks and learning systems*, 31(11):4500–4511.

[19] Feichtenhofer, C., Pinz, A., and Zisserman, A. (2016). Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1933–1941.

[20] Foerster, J. N., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., and Mordatch, I. (2017). Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*.

[21] Gemp, I. and Mahadevan, S. (2018). Global convergence to the equilibrium of gans using variational inequalities. *arXiv preprint arXiv:1808.01531*.

[22] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

[23] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee.

[24] Grnarova, P., Levy, K. Y., Lucchi, A., Hofmann, T., and Krause, A. (2017). An online learning approach to generative adversarial networks. *arXiv preprint arXiv:1706.03269*.

[25] Han, A., Mishra, B., Jawanpuria, P., Kumar, P., and Gao, J. (2022). Riemannian hamiltonian methods for min-max optimization on manifolds. *arXiv preprint arXiv:2204.11418*.

[26] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969.

[27] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[28] Heo, B., Chun, S., Oh, S. J., Han, D., Yun, S., Kim, G., Uh, Y., and Ha, J.-W. (2020). Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. *arXiv preprint arXiv:2006.08217*.

[29] Huang, F. and Gao, S. (2023). Gradient descent ascent for minimax problems on riemannian manifolds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[30] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2016). Densely connected convolutional networks.

[31] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[32] Huang, H., Wang, C., and Dong, B. (2018). Nostalgic adam: Weighting more of the past gradients when designing the adaptive learning rate. *arXiv preprint arXiv:1805.07557*.

[33] Jin, C., Netrapalli, P., and Jordan, M. (2020). What is local optimality in nonconvex-nonconcave minimax optimization? In *International Conference on Machine Learning*, pages 4880–4889. PMLR.

[Joffe] Joffe, B. Functions 3d: Examples. https://www.benjoffe.com/code/tools/functions3d/examples.

[35] Jordan, M., Lin, T., and Vlatakis-Gkaragkounis, E.-V. (2022). First-order algorithms for min-max optimization in geodesic metric spaces. *Advances in Neural Information Processing Systems*, 35:6557–6574.

[36] Jorge Nocedal, S. J. W. (2006). *Numerical Optimization*. Springer-Verlag, New York.

[37] Kang, M., Zhu, J.-Y., Zhang, R., Park, J., Shechtman, E., Paris, S., and Park, T. (2023). Scaling up gans for text-to-image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[38] Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

[39] Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410.

[40] Kingma, D. P. and Ba, J. (2014a). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[41] Kingma, D. P. and Ba, J. (2014b). Adam: A method for stochastic optimization.

[42] Koren, Y. (2009). Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456.

[Korpelevich] Korpelevich, G. Extragradient method for finding saddle points and other problems (in russian) economics and mathematical methods, 1976, vol.

[44] Krizhevsky, A. (2009). Cifar-10 (canadian institute for advanced research). Technical report, University of Toronto.

[Krizhevsky et al.] Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10, and cifar-100 (canadian institute for advanced research).

[46] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

[47] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[48] Letcher, A., Balduzzi, D., Racaniere, S., Martens, J., Foerster, J., Tuyls, K., and Graepel, T. (2019). Differentiable game mechanics. *The Journal of Machine Learning Research*, 20(1):3032–3071.

[49] Liesen, J. and Strakos, Z. (2013). *Krylov subspace methods: principles and analysis*. Numerical Mathematics and Scie.

[50] Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. (2019). On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*.

[51] Mertikopoulos, P., Lecouat, B., Zenati, H., Foo, C.-S., Chandrasekhar, V., and Piliouras, G. (2018). Optimistic mirror descent in saddle-point problems: Going the extra (gradient) mile. *arXiv preprint arXiv:1807.02629*.

[52] Mescheder, L., Nowozin, S., and Geiger, A. (2017a). The numerics of gans. *Advances in neural information processing systems*, 30.

[53] Mescheder, L., Nowozin, S., and Geiger, A. (2017b). The numerics of GANs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1823–1833, Red Hook, NY, USA. Curran Associates Inc.

[54] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[55] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

[56] Mokhtari, A., Ozdaglar, A., and Pattathil, S. (2020). A unified analysis of extra-gradient and optimistic gradient methods for saddle point problems: Proximal point approach. In *International Conference on Artificial Intelligence and Statistics*, pages 1497–1507. PMLR.

[57] Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer, New York, NY, USA, 2e edition.

[58] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

[59] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[60] Rakhlin, A. and Sridharan, K. (2013). Online learning with predictable sequences. In *Conference on Learning Theory*, pages 993–1019. PMLR.

[61] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

[62] Redmon, J. and Farhadi, A. (2017). Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271.

[63] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28.

[64] Rendle, S. (2010). Factorization machines. In *2010 IEEE International conference on data mining*, pages 995–1000. IEEE.

[65] Roy, S., Paoletti, M., Haut, J., Dubey, S., Kar, P., Plaza, A., and Chaudhuri, B. (2021). Angulargrad: A new optimization technique for angular convergence of convolutional neural networks. *arXiv preprint arXiv:2105.10190*.

[66] Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. *Advances in neural information processing systems*, 30.

[67] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242.

[68] Sarzynska-Wawer, J., Wawer, A., Pawlak, A., Szymanowska, J., Stefaniak, I., Jarkiewicz, M., and Okruszek, L. (2021). Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304:114135.

[69] Schaefer, F., Zheng, H., and Anandkumar, A. (2020). Implicit competitive regularization in GANs. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8533–8544. PMLR.

[70] Schäfer, F. and Anandkumar, A. (2019). Competitive gradient descent. *Advances in Neural Information Processing Systems*, 32.

[71] Schäfer, F. and Anandkumar, A. (2019). Competitive gradient descent. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA. Curran Associates Inc.

[72] Schäfer, F., Zheng, H., and Anandkumar, A. (2019). Implicit competitive regularization in gans. *arXiv preprint arXiv:1910.05852*.

[73] Shalev-Shwartz, S. and Singer, Y. (2006). Convex repeated games and fenchel duality. *Advances in neural information processing systems*, 19.

[74] Simonyan, K. and Zisserman, A. (2014a). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[75] Simonyan, K. and Zisserman, A. (2014b). Very deep convolutional networks for large-scale image recognition.

[76] Song, C., Pons, A., and Yen, K. (2021). Ag-sgd: Angle-based stochastic gradient descent. *IEEE Access*, 9:23007–23024.

[77] Tan, M. and Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR.

[78] Tieleman, T., Hinton, G., et al. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[79] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

[80] Vinyals, O., Blundell, C., Lillicrap, T., kavukcuoglu, k., and Wierstra, D. (2016a). Matching networks for one shot learning. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

[81] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016b). Matching networks for one shot learning. *Advances in neural information processing systems*, 29.

[82] Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., and Van Gool, L. (2016). Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer.

[83] Wang, Y., Zhang, G., and Ba, J. (2019). On solving minimax optimization locally: A follow-the-ridge approach. *arXiv preprint arXiv:1910.07512*.

[84] Wu, X., Hu, Z., and Huang, H. (2023). Decentralized riemannian algorithm for nonconvex minimax problems. *arXiv preprint arXiv:2302.03825*.

[85] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

[86] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

[87] Yadav, A., Shah, S., Xu, Z., Jacobs, D., and Goldstein, T. (2017). Stabilizing adversarial nets with prediction methods. *arXiv preprint arXiv:1705.07364*.

[88] Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., and Xiao, J. (2015). Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. In *arXiv preprint arXiv:1506.03365*.

[89] Yu, F., Wang, D., Shelhamer, E., and Darrell, T. (2018). Deep layer aggregation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2403–2412.

[90] Zhang, P., Zhang, J., and Sra, S. (2023). Sion's minimax theorem in geodesic metric spaces and a riemannian extragradient algorithm.

[91] Zhuang, J., Tang, T., Ding, Y., Tatikonda, S. C., Dvornek, N., Papademetris, X., and Duncan, J. (2020). Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in neural information processing systems*, 33:18795–18806.