UNIVERSITY OF CALIFORNIA, MERCED

Optimization for machine learning: Memory-efficient and tractible solutions to large-scale non-convex systems

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering and Computer Science

by

Aditya Ranganath

Committee in charge:

Dr. Mukesh Singhal, Chair
Dr. Roummel F. Marcia, Co-Chair
Dr. Meng Tang
Dr. Harish Bhat

2023

The dissertation of Aditya Ranganath is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

(Dr. Meng Tang)

(Dr. Harish Bhat)

(Dr. Roummel F. Marcia, Co-Chair)

(Dr. Mukesh Singhal, Chair)

University of California, Merced

2023

DEDICATION

I dedicate this thesis to my family - my brother Narayanan Ranganath, my mother Pavana Ranganath, my father Ranganath Kaushik Raghavan, my grandfather Nadamuni Ramaswamy Srinivasan and my grandmothers Sasi Srinivasan and Lakshmi Raghavan.

EPIGRAPH

The only true wisdom is in knowing you know nothing.

—Socrates

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Symbols

$\mathbf{g}$	Gradient information
$\mathbf{H}$	Hessian Matrix
$\mathbf{B}$	Hessian Approximation
$\mathbf{x}$	Input sample
$\mathbf{y}$	Output labels
$\odot$	Elementwise Product sum for vectors
$\mathcal{L}$	Risk of estimation
Θ	Neural network parameterization
η	Learning rate
ϵ	Small scalar value
$\mathcal{Q}$	Quadratic approximation of $\mathcal{L}$
Δ	Trust-region radius
$\mathbf{E}$	Encoder
$\mathbf{D}$	Decoder
Adam	Adaptive Moment estimation algorithm
ReLU	Rectified Linear Unit
RMSProp	Root-Mean-Square Propagation
SGD	Stochastic Gradient Descent
SR1	Limited Symmetric Rank-1 update
CG	

ACKNOWLEDGEMENTS

I would like to acknowledge the efforts of my advisors Dr. Mukesh Singhal and Dr. Roummel F. Marcia for giving me the opportunity to work on the projects elaborated in this thesis. They have shared my joy and celebrations during my achievements and have offered strength and wisdom in failures. Their guidance has been steady and strong through my journey and I hope I can carry on this legacy. My sincerest regards to both of you.

I would like to thank my committee member Dr. Harish Bhat, whose support has been really strong throught my association with him. I am appreciative of all the wisdom he has imparted on me over the years and all the discussions on deep learning and machine learning. I am extremely grateful to have served as a Graduate Research Assistant under the grant NSF DMS-1723272 guided by Dr. Bhat. It was an enriching experience in a fresh domain of Time-dependent Density Functional Theory. It was his special topics course that motivated some of the ideas that we will see later in this thesis.

I would like to thank Dr. Meng Tang for readily agreeing to be on my committee. His rich research experience will hopefully guide my research in a new direction during my future endeavours.

Moving on to my academic family, I would like to extend my wishes to Dr. Omar Deguchy, Jacqueline Alvarez, Irabiel Romero, Azar Alizadeh and Jocelyn Ornelas Munoz for their collaborative efforts and support. I appreciate all the conversations we have had and the ideas we have exchanged with each other.

I would like my collaborators from Lawrence Livermore National Laboratory (LLNL) - Jason Van Tuinen and Dr. Goran Konjevod.

In addition, I would like to thank Cindy Gonzalez, Brian Gallagher, Dr. Suzanne Sindi and Dr. Mikkel Landajuela, for the opportunity to work at LLNL during the summer of 2023. The experience of participating at LLNL data science was nothing short of fantastic.

Finally, I would like to extend my gratitude and appreciation to my family members - Narayanan Ranganath, Pavana Ranganath and Ranganath Kaushik Raghavan for their unwavering support and guidance through the years.

VITA

2010-2014	B. E. in Electrical and Electronics Engineering, Loyola-ICAM College of Engineering and Technology, Chennai, Tamil Nadu.
2016-2018	M. S. in Electrical Engineering and Computer Science, University of California, Merced, CA
2018-current	PhD. in Electrical Engineering and Computer Science, University of California, Merced, CA

PUBLICATIONS

Aditya Ranganath, Omar Deguchy, Mukesh Singhal, Roummel F. Marcia. Multi-Stage noise reduction with recurrent neural networls. In *2021 55th Asilomar Conference on Singals, Systems, and Computers*, pages 135-139, 2021b, doi:10.1109/IEEECONF53345.2021.9723266

Aditya Ranganath, Omar DeGuchy, Fabian Santiago, Mukesh Singhal, and Roummel Marcia. Recurrent neural imaging: An evolutionary approach for mixed possion-gaussian image denoising. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 484-489, 2022. doi: 10.1109/ICMLA55696.2022.00078.

Aditya Ranganath, Omar DeGuchy, Mukesh Singhal, and Roummel F. Marcia. Multi-stage gaussian noise reduction with recurrent neural networks. *In 2021 55th Asilomar Conference on Signals, Systems, and Computers*, pages 135-139, 2021b. doi: 10.1109/IEEECONF53345.2021.9723266.

Jason Van Tuinen, **Aditya Ranganath**, Goran Konjevod, Mukesh Singhal, and Roummel Marcia. Novel adversarial defense techniques for white-box attacks. *In 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 617-622, 2022. doi:10.1109/ICMLA55696.2022.00095.

Aditya Ranganath, Mukesh Singhal, Roummel Marcia, Stochastic Adaptive Regularization Using Cubics with L-SR1 Updates for Deep Neural Networks, *Manuscript under review, Association for Advancement of Artificial Intelligence (AAAI)*

Aditya Ranganath, Robert Smith, Jocelyn Ornelas Munoz, Mukesh Singhal, Roummel Marcia, Image Separation using Transformer attention models, *Manusrcript under review, IEEE International Conference on Acoustics, Speech and Signal Processing.*

Aditya Ranganath, Irabiel Romero, Mukesh Singhal, Roummel Marica, Quasi-Adam: Improving Adam using quasi-Newton approximation. *Manuscript under review, Internaltional Conference on Artificial Intelligence and Statistics (AISTATS)*

Boaz Ilan, **Aditya Ranganath**, Jacqueline Alvarez, Shilpa Khatri, and Roummel Marcia. Interpretability of relu for inversion. *In 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1190-1195, 2022. doi: 10.1109/ICMLA55696.2022.00192.

Azar Alizadeh, Mukesh Singhal, Vahid Behzadan, Pooya Tavallali, and **Aditya Ranganath**. Stochastic induction of decision trees with application to learning haar trees. *In 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 825-830, 2022b. doi: 10.1109/ICMLA55696.2022.00137.

Azar Alizadeh, Pooya Tavallali, Vahid Behzadan, **Aditya Ranganath**, and Mukesh Singhal. A novel approach for synthetic reduced nearest-neighbor leveraging neural networks. *In 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 831-836, 2022a. doi: 10.1109/ICMLA55696.2022.00138.

Azar Alizadeh, Mukesh Singhal, Vahid Behzadan, Pooya Tavallali, and **Aditya Ranganath**. Stochastic induction of decision trees with application to learning haar trees. *In 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 825-830, 2022b. doi: 10.1109/ICMLA55696.2022.00137.

ABSTRACT OF THE DISSERTATION

Optimization for machine learning: Memory-efficient and tractible solutions to large-scale non-convex systems

by

Aditya Ranganath
Doctor of Philosophy in Electrical Engineering and Computer Science
University of California Merced, 2023
Dr. Mukesh Singhal, Chair

Neural networks generally require large amounts of data to adequately model the domain space. In situations where the data are limited, the predictions from these models, which are typically obtained from stochastic gradient descent (SGD) minimization algorithms, can be poor. In addition, the data is commonly corrupted due to poor imaging appatus. In these cases, the use of more sophisticated optimization approaches and model architectures becomes crucial to increase the impact of each training iteration. Second-order methods can capture curvature information, providing a more informed guess on the direction and step length. However, they require vast amounts of storage and can be computationally time demanding.

To address the computational issue, we propose an optimization algorithm that uses second-derivative information, exploiting curvature information for avoiding saddle points. We utilize a Hessian-free approach where we do not explicitly store the second-derivative matrix, by applying a conjugate gradient method. The algorithm uses a trust-region method, which does not require the Hessian to be positive definite. We present numerical experiments which demonstrate the improvement in classification accuracy using our proposed approach over a standard SGD approach.

We propose using a limited-memory symmetric rank-one quasi-Newton approach which further addresses the time and space computational complexity. The

approach allows for indefinite Hessian approximations, enabling directions of negative curvature to be exploited. Furthermore, we use a modified adaptive regularized using cubics approach, which generates a sequence of cubic subproblems that have closed-form solutions with suitable regularization choices and investigate the performance of our proposed and compare our approach to state-of-the-art first-order and other quasi-Newton methods.

To incorporate the benefits of an exponential moving average algorithm to a quasi-Newton approach, we propose a quasi-Adam approach. Judicious choices of quasi-Newton matrices can lead to guaranteed descent in the objective function and improved convergence. In this work, we integrate search directions obtained from using these quasi-Newton Hessian approximations with the Adam optimization algorithm. We provide convergence guarantees and demonstrate improved performance through an extensive experimentation on a variety of applications.

Finally, to mitigate the issue of data corruption, we propose a variety of architectures for various applications in image processing. We propose a blind source signal separator, which involves separating image signals which have been superimposed by a common observing apparatus. We propose novel deep learning architectures for low photon count image denoising, which contains Gaussian noise in a low-photon count setting. Then we propose a novel architecture for low-photon count and downsampled imaging, where the signal is interfered with some Gaussian noise, Poisson noise and then downsampled. Finally, we propose a novel adversarial detection method for white-box attacks using Radial basis function and Discrete Cosine Transforms.

Chapter 1

Introduction

Optimization, in the context of deep learning, is the process of fine-tuning a neural network's parameters to minimize the loss function, which measures the disparity between the model's predictions and the target values. This crucial aspect of deep learning aims to improve a model's performance by adjusting weights and biases using various optimization algorithms. Effective optimization not only helps deep learning models generalize better to unseen data but also plays a pivotal role in training deep neural networks with millions of parameters, making them capable of solving complex real-world problems in fields like computer vision, natural language processing, and reinforcement learning.

First-order optimization methods such as Gradient Descent, form the foundation of many of mordern optimization algorithms in the field of deep learning. It computes the gradient of the loss function with respect to the model parameters and iteratively adjusts the model parameters in the direction of steepest descent. This simplicity makes it accessible and easy to implement. Each iteration involves computing gradients, which is a relatively inexpensive operation. This efficiency is crucial when working with large datasets and deep neural networks. More advanced optimization techniques like Adam, RMSprop, and stochastic gradient descent (SGD) with momentum enhance convergence speed and stability. However, it can be challenging to tune the hyperparameters of an gradient-based optimization scheme, which signficantly impacts the training process. In addition, the convergence of these schemes can be slow depending on the shape of the manifold.

Second-order optimization methods, utilize information from the second derivative of the loss function with respect to model parameters. These methods are employed to enhance the efficiency and convergence properties of optimization algorithms. While first-order methods rely solely on gradient information, second-order methods incorporate curvature information to navigate the optimization landscape more efficiently. This leads to faster convergence, better adaptation to varying learning rates, and improved stability during training. Despite these advantages, second-order methods face several limitations in the context of deep learning. Deep neural networks often have millions of parameters, making the computation and

storage of the Hessian highly demanding. Additionally, the non-convex nature and the prevalence of saddle points can sometimes cause second-order methods to converge to undesirable solutions.

Quasi-Newton methods are a family of optimization algorithms, which are used to approximate the Hessian information. These methods are designed to extract information from the Hessian matrix without the need to compute it. The most well-known quasi-Newton method is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) and the Symmetric Rank-one (SR-1) algorithms, but there are several other variants as well, including the Limited-Memory BFGS (L-BFGS) method and the Limited-Memory SR-1 (LSR-1) method. However, in deep learning and large-scale machine learning problems, quasi-Newton methods are less commonly used due to their high computationally complex nature. Quasi-Newton methods are not inherently designed to work with stochastic gradients and do not adapt well to the inherent noise in the gradients when using mini-batches. In addition, the non-convex nature of these manifolds can cause the method to converge to sub-optimal solutions. In Sec. 1.1, we briefly discuss about deep learning and their architectures. In Sec. 1.2, we discuss about different optimization techniques with their advantages and disadvantages. In Sec. 1.3, we identify the problems associated with these optimization techniques. In Sec. 1.4, we discuss the advantages of these different approaches and how to exploit these advantages in a deep learning setting.

1.1 Deep learning

Neural networks, are a foundational concept in the field of machine learning and artificial intelligence. They are computational models inspired by the structure and function of the human brain, designed to process information and make predictions. Neural networks have become a critical component of a wide range of applications, from image and speech recognition to natural language processing and autonomous vehicles.

At their core, neural networks consist of interconnected nodes, or "neurons",

organized into layers. These layers typically include an input layer, one or more hidden layers, and an output layer. Neurons within the network are analogous to biological neurons, processing and transmitting information through weighted connections. The power of neural networks lies in their ability to learn from data and adapt their internal parameters, known as "weights," to make accurate predictions or classifications.

The process of training a neural network involves presenting it with a dataset containing input-output pairs. During training, the network adjusts its weights using optimization techniques to minimize the difference between its predictions and the actual target values, typically quantified by a loss function. This iterative learning process allows neural networks to generalize from the training data, making them capable of handling new, unseen data and making predictions or decisions based on the patterns they have learned.

Neural networks come in various architectures, with feedforward neural networks (FNNs) being the simplest, where information flows in one direction, from the input layer to the output layer. More complex architectures include convolutional neural networks (CNNs), designed for image and spatial data, and recurrent neural networks (RNNs), which excel in sequential data processing, making them suitable for tasks like natural language understanding and time series analysis. For more information on the types of models used in this thesis, please refer to the Appendix Sec. A.1.

Neural networks have achieved remarkable success in recent years, especially in deep learning, where models with many hidden layers, known as deep neural networks, have demonstrated exceptional performance in a wide range of applications. This resurgence of interest in neural networks, fueled by advances in hardware and large datasets, has made them a key technology in modern artificial intelligence, revolutionizing fields like computer vision, speech recognition, and natural language understanding. As the field of neural networks continues to evolve, it holds the promise of even more breakthroughs in machine learning and AI applications.

1.2 Optimization

Optimization is of paramount importance in deep learning. Deep learning involves training neural networks with a vast number of parameters (weights and biases) to make predictions or classifications. It is the process of finding the optimal values for these parameters, leading to a model that performs well on the given task. The core objective in deep learning is to minimize a loss function, which estimates the disparity between the model's predictions and the actual target values. This disparity, often referred to as loss, is minimized by the neural network, enabling the model to make accurate predictions. The optimization setup for a neural network is defined as

$$\underset{\Theta \in \mathbb{R}^n}{\text{minimize}}\ \mathcal{L}(\Theta) \equiv \frac{1}{m} \sum_{j=1}^{j=m} \mathcal{L}_j(\Theta; \mathbf{x}_j, \mathbf{y}_j) \equiv \frac{1}{m} \sum_{j=1}^{m} \mathcal{L}_j(\Theta), \tag{1.1}$$

where $\mathcal{L}_j$ is a function that depends on the j^{th} observation in the training set $\{(x_j, y_j)\}_{j=1}^{n}$. These objective functions are generally large-scale (the dimension of Θ, n, and the number of data points, m, are typically in the order of millions), non-linear (the function $\mathcal{L}$ often involves nonlinear activation functions), and non-convex ($\mathcal{L}$ is a composition of functions that can result in non-convexity [1]). An optimized model is not only good at fitting the training data but also at generalizing its knowledge to new, unseen data. Effective optimization helps improve a model's ability to generalize, reducing overfitting and enhancing its performance on real-world data. For the methods discussed in this thesis, the optimization techniques are broadly divided into two categories - **First-order** methods, **Second-order** methods and **Quasi-Newton** methods.

1.2.1 First-order Methods

First-order methods involve computing the gradients of the objective function with respect to the parameterization of the function. In this thesis, the gradient, unless otherwise specified, is referred to as $\mathbf{g}_i$, where $\mathbf{g} = \nabla_\Theta \mathcal{L}$ at the i^{th} iteration. The gradient based method is an iterative process, where for each j^{th} input batch

or input of the dataset $\mathbf{x}_j$, the gradients are computed and the the parameters are adjusted according to the following update scheme:

$$\Theta_{i+1} = \Theta_i - \eta \mathbf{g}_i. \tag{1.2}$$

This $-\mathbf{g}_i$ is the negative gradient direction and η is referred to as *learning-rate*. For more details on origin and usage, see A.1. The iterate update 1.2 forms the foundation of most mordern gradient-based first-order approaches. One of the first optimization schemes to be used in a deep learning setting is the **Batch-Gradient Descent** method and **Stochastic-Gradient Descent (SGD)** method. In a Batch-Gradient Descent method, the objective function $\mathcal{L}$ is evaluated over a batch instead of the entire dataset. Each batch is iteratively fed to the network and the gradients are calculated over each batch. In a SGD setting, the dataset is divided into batches, with the caveat that only select sampled datapoints are used to evaluate the gradient of that batch. We discuss SGD in more detail in Sec 2.1.

Advantages:

1. $\mathbf{g}_i$ is fairly easy to compute. It is asymptotically equal to one forward-pass through the neural network.
2. $\mathbf{g}_i$ is fairly easy to store. It requires only $\mathcal{O}(n)$ memory.
3. In practice, first-order approaches always converge to the minimizer **eventually**.

Disadvantages:

1. First-order methods suffer from slow and linear-convergence.
2. These methods get stuck at saddle-points.

For more details on saddle-points and linear-convergence, see Sec. A.2.2.

1.2.2 Second-order Methods.

Second order approaches use the second derivative of the objective function $\mathcal{L}$ with respect to their parameters Θ. Often referred to as a *Newton method*, the

second-order approaches use the Hessian matrix $\mathbf{H} = \nabla^2_\Theta \mathcal{L}$, where $\mathbf{H} \in \mathbb{R}^{n \times n}$. The iterative algorithm for a Newton method is given by

$$\Theta_{i+1} = \Theta_i - [\mathbf{H}_i]^{-1}\mathbf{g}_i. \tag{1.3}$$

For more information on how this update is derived and formulated, see Appendix Sec. A.2.

Advantages:

1. The Newton's method enjoys quadratic convergence.
2. The step exploits curvature information of the function.

Disadvantages:

1. For large-scale non-convex problems like neural-networks, it is computationally difficult to store the Hessian information.
2. The matrix inversion operation can be computationally expensive.
3. Since $\mathcal{L}$ is highly non-convex and nonlinear, the Hessian can be singular or negative-definite.

For more details on convergence and computational budget, see Sec. A.2.1.

1.2.3 Quasi-Newton Methods

Quasi-Newton methods use the *secant-equation* to approximate the Hessian information. Given Θ_i, Θ_{i-1}, and their corresponding gradients $\mathbf{g}_i, \mathbf{g}_{i-1}$, the secant equation is defined by

$$\mathbf{y}_i = \mathbf{B}_{i+1}\mathbf{s}_i, \tag{1.4}$$

where $\mathbf{y}_i = \mathbf{g}_i - \mathbf{g}_{i-1}$, $\mathbf{s}_i = \Theta_i - \Theta_{i-1}$ and $\mathbf{B}_{i+1} \approx \mathbf{H}_{i+1}$ is the Hessian approximation. Quasi-Newton method became popular in a deep-learning setting due to their ease in computation and limited memory requirements. There are a variety of quasi-Newton approaches such as *Broyden-Fletcher-Goldfarb-Shanno* (**BFGS**),

Symmetric-Rank-1 (**SR-1**) methods, which approximate this Hessian information **B**. The update strategy for a quasi-Newton approach is given by

$$\Theta_{i+1} = \Theta_i - [\mathbf{B}_i]^{-1}\mathbf{g}_i. \tag{1.5}$$

Advantages:

1. These methods are computationally economic and enjoy a linear memory depenence.
2. The steps are easy to compute and require very few matrix-vector products.
3. They approximate the Hessian information such that the curvature information is induced.

Disadvantages:

1. The quasi-Newton matrix is only an approximation of the Hessian matrix. This means, the curvature information is only *approximated* under this setting.
2. Some quasi-Newton matrices such as BFGS always stay postive-definite. This may be an issue in regions where the true Hessian is negative-definite or singular.
3. The method has not been explicitly proven to enjoy a superlinear convergence.

1.3 Problem Statement

Optimization in deep learning is extremely crucial. However, due to limited memory constraints and highly non-convex nature of the objective functions, this task becomes intractible.

- First-order methods are cheap to compute and easy to store. However, due to lack of curvature information, their convergence can be slow and infficient.

- Second-order methods contain curvature information. However, storing and inverting a Hessian can be expensive, and in some cases, non-invertible.
- Limited memory Quasi-Newton methods are approximations to the second-order derivative and still remain a first-order method as they do not store the full-rank Hessian information, but merely a low-rank approximation to it.
- The current literature does not adequately address the optimization of neural networks using these Quasi-Newton approaches This is often attributed to the noisy nature of the gradients and highly non-convex nature of the objective functions.
- In addition, the architecture of the neural networks and their corresponding objective functions play a major role.
- In addition, neural networks are large-scale parameterized methods which require adequate data and a good optimzation scheme will prevent a network from overfitting, but allows the network to generalize well on unseen data. The mark of a good neural network is a good network model with adequate parameterization and architecture, which allows the network to train without bias.

1.4 Motivation

In this chapter, we have discussed the different deep learning neural networks used depending on the task. We have identified the optimization setup for these neural networks and specified the algorithm used for different methods. We have discussed the advantages and disadvantages of these approaches. The motivation of the thesis is presented as follows:

1. Exploit curvature information without explicitly storing or inverting the Hessian.

2. Approximate the Hessian information with appropriate safeguards and limited-memory strategies.

3. Maintain a positive definite matrix and influence the step to pull away from saddle-points.

4. Design networks which can expedite the training process with a smaller computational and memoryfootprint.

This thesis is orgaized as follows: In Chapter 2, we propose a Trust-region based second-order Hessian-Free approach. In Chapter 3, we discuss an Adaptive-Regularized Cubics methods which uses the Limited-Memory Symmetric-Rank-one method. In Chapter 4, we discuss a quasi-Newton and exponential moving average method. In Chapter 5, we propose various deep learning architectures across different image processing applications and how their performance improves over existing architectures. In Chapter 6, we summarize the contributions of the work presented here.

Chapter 2

Hessian-Free Trust-Region methods

The recent success in the application of neural networks to a variety of fields can be mostly attributed to two driving factors: improvements in network architeccture and the optimization techniques used during the learning process [2–4]. Since the inception of the back-propagation algorithm made the optimization of the network parameters viable, methods using first-derivative information have dominated the field (see e.g., [5–11]). With an increased access to powerful computational resources, there is greater potential for the field to shift toward considering more sophisticated algorithms. This is particularly true for data-limited inference, where the availability of data is limited, i.e., n in (1.1) is relatively small. In the case of an image classifier, this translates to a limited number of training images for classification.

The novelty of the proposed approach is as follows: **we exploit second-derivative information to improve the predictive capabilities of artificial neural networks for data-limited inference**. Our method combines the efficient computation of a true Hessian-vector product in a trust-region setting, thus allowing us to solve the trust-region subproblem using a conjugate based method.

The chapter is organized as follows. In Sec. 2.1, we describe existing and commonly used algorithms and then discuss some of the recent techniques which approximate second-order information. In Sec. 2.2, we describe the novelty of our proposed approach, which is based on trust-region methods, which are alternatives to the more commonly-used line-search methods in optimization. We describe our numerical experiments in Sec. 2.3 and the main results in Sec. 2.4. Finally, we summarize the chapter with concluding remarks in Sec. 2.5.

2.1 Related Methods

Hessian-Free Methods. Hessian-free methods look to improve on gradient-descent methods by using higher-order information. by capitalizing on approximations of second-derivative information or in some cases using the true Hessian itself. In either regime, the explicit storage of the associated matrix can be quite expensive when considering the large-scale optimization problem associated with training

a neural network. In order to minimize this cost, Hessian-free methods focus on the matrix vector product of the Hessian or Hessian approximation ($\mathbf{H}$) and an n-dimensional vector (d). To minimize the cost of storing second-derivative (Hessian) matrices (which can be potentially too large to store in memory), Hessian-free methods only require them for matrix-vector multiplication. In [12], the Hessian-free optimization method is implemented using the finite difference approximation of the matrix vector product: The approach by Martens [12] uses the finite difference approximation of the matrix vector product:

$$\mathbf{Hd} = \lim_{\epsilon \to 0} \frac{\nabla \mathcal{L}(\Theta + \epsilon \mathbf{d}) - \nabla \mathcal{L}(\Theta)}{\epsilon}, \tag{2.1}$$

where the operation is used in a conjugate gradient (CG) setting in order to provide the descent direction to the next iterate. Pearlmutter [13] offers an alternative approach which computes the actual Hessian-vector product as

$$\mathbf{Hd} = \frac{\partial}{\partial \epsilon} \nabla \mathcal{L}(\Theta + \epsilon \mathbf{d}) \Big|_{\epsilon = 0}. \tag{2.2}$$

For details on implementation, see [14–17]. To calculate $\mathbf{Hd}$ for a simple backpropagation network, see [14, 15], and for recurrent backpropagation networks, see [16, 17]. One requirement when using either operation in the context of typical CG methods is that $\mathbf{H}$ be positive definite. This is a requirement to ensure that the CG method or any linesearch methods result in a descent direction. A related approach utilizes Gauss-Newton approximations [18]. For example, the approach in [19] extends the approximation for use with a cross-entropy loss and a framework similar to [13].

$\mathbf{H}$ is often required to be positive definite to guarantee that the computed search direction is a descent direction. As they are stated, the matrix-vector products in (2.1) and (2.2) do not provide any guarantees that the matrix $\mathbf{H}$ is positive definite. One commonly used technique to guarantee to avoid the problems associated with an indefinite matrix is to shift or “dampen" the eigenvalues of $\mathbf{H}$ as $\mathbf{B} = \mathbf{H} + \lambda \mathbf{I}$, where $\lambda \geq 0$. The Hessian-vector product is now expressed as $\mathbf{Bd} = \mathbf{Hd} + \lambda \mathbf{d}$, where $\mathbf{Hd}$ is evaluated using the previously described techniques.

Gauss-Newton approximations have also been proposed as an alternative to the previous two methods. The Gauss-Newton approximation, $\mathbf{G}$, also known as the

squared Jacobian, is an outer product of the Hessian, typically used in least-squares problems. In [19], they extend the approximation for use with a cross-entropy loss using a similar framework to that presented in [13]. The author notes that as long as $\mathbf{G}$ is positive semi-definite, descent is guaranteed, but to overcome the possibility of indefiniteness, a similar dampening approach must be taken. In later sections, we will make use of Pearlmutter's method while relaxing the requirement that the Hessian be positive definite.

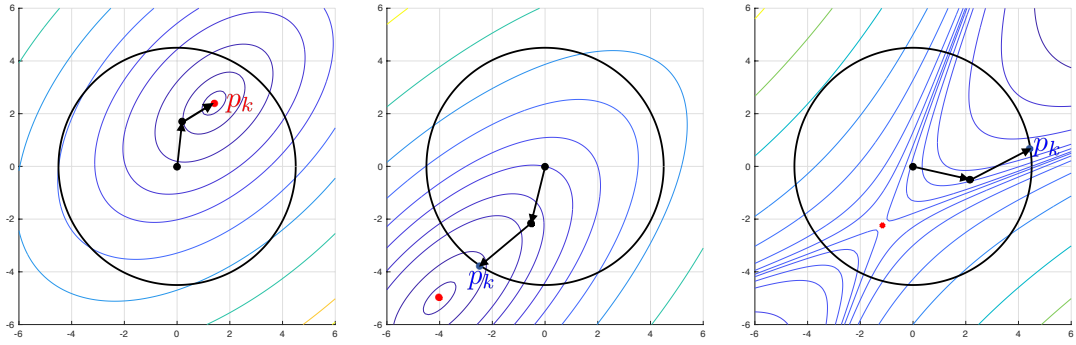
Stochastic Gradient Descent (SGD). First-derivative algorithms have emerged as the standard optimization techniques used for training deep neural networks. In particular, methods based on stochastic gradient descent (SGD) are preferred for their low computational cost and ease of implementation [5, 20]. This method differs from a classic gradient descent approach in that the gradient is computed based on a sample of the dataset. In the context of deep learning, the gradient is computed based on a sample batch. Specifically, at iteration k in SGD, a sample batch $S_k \subseteq \{1, 2, \ldots, n\}$ is randomly chosen and the current iterate w_k is updated using

$$\Theta_{k+1} = \Theta_k - \eta_k \frac{1}{|S_k|} \sum_{j \in S_k} \nabla \mathcal{L}_j(\Theta_k),$$

where η_k known as the *learning rate*.

2.2 Proposed Approach

In the methods described above, the Hessian $\mathbf{H}$ is often required to be positive definite to guarantee that the computed search direction is a descent direction. For non-convex problems, this requirement is not always satisfied. In [12], the eigenvalues of $\mathbf{H}$ are shifted by adding $\lambda\mathbf{I}$ to $\mathbf{H}$, where $\lambda > 0$ is obtained heuristically. Here, we propose using a trust-region approach that does not require modifying the eigenvalues of $\mathbf{H}$. As they are stated, the matrix-vector products in (2.1) and (2.2) do not provide any guarantees that the matrix $\mathbf{H}$ is positive definite. One commonly used technique to guarantee to avoid the problems associated with an indefinite matrix is to shift or "dampen" the eigenvalues of $\mathbf{H}$ as $\mathbf{B} = \mathbf{H} + \lambda\mathbf{I}$,



(a) (b) (c)

Figure 2.1: Illustration of the CG-Steihaug approach in two dimensions. (a) When $\mathcal{Q}_k(p)$ is convex and its unconstrained minimizer lies within the trust-region radius, then the CG iterates will converge to the unconstrained minimizer. (b) When $\mathcal{Q}_k(p)$ is convex but its unconstrained minimizer is outside the trust region, then the minimizer p_k is defined where the CG iterate crosses the boundary. (c) When $\mathcal{Q}_k(p)$ is not convex, i.e., $\mathbf{H}_k$ is not positive definite, then the CG-Steihaug method terminates when a direction of curvature is detected and the minimizer $\mathbf{p}_k$ is defined where $\mathcal{Q}_k(p)$ is minimized along the last computed CG iterate.

where $\lambda \geq 0$. The Hessian-vector product is now expressed as $\mathbf{Bd} = \mathbf{Hd} + \lambda\mathbf{d}$, where $\mathbf{Hd}$ is evaluated using the previously described techniques.

The proposed method presents a novel optimization routine designed to minimize (1.1). Unlike the previous methods described, the goal is computing fast Hessian-vector products within a trust-region setting. This allows us to approximately solve the trust-region subproblem using CG while allowing for negative curvature. By incorporating second-derivative information, we improve the impact of each iteration and avoid certain local minima and saddle points. The increase in the quality of the optimization routine will increase the value of each data point and allow us to reach better optima with fewer training instances. In the following subsections we describe the proposed approach in more detail.

Fast Exact Hessian-Vector Products. As stated in the previous section, computing the second-derivative or Hessian can be computationally intensive. Furthermore, in the context of neural networks, storing the Hessian can be infeasible. Using exact second derivative information allows us to create an accurate localized model of the true objective function which is used in the trust-region setting

Algorithm 1 Proposed Second-Order Trust-Region Method

Given: For some $\Delta_{\max} > 0, \Delta_0 \in (0, \Delta_{\max}), \eta \in [0, \frac{1}{4})$,

and $\epsilon > 0$

while $\|\nabla \mathcal{L}(\Theta_k)\|_2 > \epsilon$ **do**

Obtain $\mathbf{p}_k$ from CG-Steihaug in [21]

Perform line search using **Wolfe conditions**

Evaluate the ratio given by

$\rho_k = (\mathcal{L}(\Theta_k) - \mathcal{L}(\Theta_{k+1}))/(\mathcal{Q}_k(0) - \mathcal{Q}_k(\mathbf{p}_k))$

if $\rho_k < \frac{1}{4}$ **then**

$\Delta_{k+1} = \frac{1}{4}\Delta_k$

else

if $\rho_k > \frac{3}{4}$ **and** $\|\mathbf{p}_k\|_2 = \Delta_k$ **then**

$\Delta_{k+1} = \min(2\Delta_k, \Delta_{\max})$

else

$\Delta_{k+1} = \Delta_k$

end if

end if

if $\rho > \eta$: **then**

$\Theta_{k+1} = \Theta_k + \mathbf{p}_k$

else

$\Theta_{k+1} = \Theta_k$

end if

end while

described in the next section. At the same time we require this information to be available for use in the CG method. In both cases, a Hessian-vector multiplication is required, and so we chose to use Pearlmutter's algorithm, commonly referred to as Rop [13]. While we are not the first to use Rop in a CG setting, our approach is novel in that we no longer require dampening of the Hessian to guarantee positive definiteness.

The motivation for avoiding damped Hessian approximations comes in two parts. The first is that it requires the choice of another hyperparameter, λ. The choice of λ can greatly affect the convergence of the optimization routine and should be chosen for each update of the Hessian vector product. The second motivation is that the perturbation to the true Hessian imposed by λ results in an approximation of the second derivative and thus less accurate curvature information. *The proposed algorithm relaxes the requirement of the Hessian to be positive-definite and compensates for the possibility of negative curvature by using a trust-region setting.*

Trust-Region Methods. Trust-region methods [22] are alternative approaches to line-search methods for solving optimization problems. While line-search methods first compute a search direction and then determine a step length along that direction at each iteration, trust-region methods determine a quadratic model to the true objective function and a corresponding region over which the quadratic model can be trusted to be accurate. Specifically, trust-region methods solve a sequence of quadratic subproblems with a single constraint of the following form:

$$\begin{aligned} \mathbf{p}_k &= \underset{\mathbf{p}\in\Re^n}{\arg\min} \quad \mathcal{Q}_k(\mathbf{p}) \equiv \nabla\mathcal{L}(\Theta_k)^\top\mathbf{p} + \frac{1}{2}\mathbf{p}^\top\nabla^2\mathcal{L}(\Theta_k)\mathbf{p} \\ &\qquad \text{subject to} \quad \|\mathbf{p}\|_2 \leq \Delta_k \end{aligned} \tag{2.3}$$

where Δ_k is a scalar parameter referred to as the *trust-region radius*. The trust-region subproblem solution p_k is used to compute the next iterate given by $\mathbf{\Theta}_{k+1} = \Theta_k + \mathbf{p}_k$, provided $\mathbf{p}_k$ satisfies a certain property discussed below.

Conjugate Gradient (CG)-Steihaug Approach. An important component of the algorithm facilitates solving the trust-region subproblem in (2.3) to provide the directional step $\mathbf{p}_k$ to the next iteration. In a similar approach to [12], we use

Algorithm 2 CG-Steihaug

Given: Tolerance $\epsilon_k > 0$

Set: $\mathbf{p}_0 = 0$, $\mathbf{r}_0 = \mathbf{g}_k, \mathbf{d}_0 = -\mathbf{r}_0 = -\mathbf{g}_k$

if $\|\mathbf{r}_0\|_2 < \epsilon_k$ **then**

  return $\mathbf{p}_k = \mathbf{z}_0 = 0$

end if

for $j \in$ 0,1,2,3,4.. **do**

  if $\mathbf{d}_j^\top \mathbf{H}_k \mathbf{d}_j \leq 0$ **then**

    Find τ such that $\mathbf{p}_k = \mathbf{z}_j + \tau \mathbf{d}_j$ minimizes

      $\mathcal{Q}_k(\mathbf{p})$ in (2.3) with $\|\mathbf{p}_k\|_2 = \Delta_k$

    return $\mathbf{p}_k$

  end if

  Set $\alpha_j = \mathbf{r}_j^\top \mathbf{r}_j / \ \mathbf{d}_j^\top \mathbf{H}_k \mathbf{d}_j$

  Set $\mathbf{z}_{j+1} = \mathbf{z}_j + \alpha_j \mathbf{d}_j$

  if $\|\mathbf{z}_{j+1}\| \geq \Delta_k$ **then**

    Find $\tau \geq 0$ such that $\mathbf{p}_k = \mathbf{z}_j + \tau \mathbf{d}_j$ satisfies

      $\|\mathbf{p}_k\|_2 = \Delta_k$

    return $\mathbf{p}_k$

  end if

  Set $\mathbf{r}_{j+1} = \mathbf{r}_j + \alpha_j \mathbf{H}_k \mathbf{d}_j$

  if $\|\mathbf{r}_{j+1}\|_2 < \epsilon_k$ **then**

    return $\mathbf{p}_k = \mathbf{z}_{j+1}$

  end if

  Set $\beta_{j+1} = \mathbf{r}_{j+1}^\top \mathbf{r}_{j+1} / \mathbf{r}_j^\top \mathbf{r}_j$

  Set $\mathbf{d}_{j+1} = -\mathbf{r}_{j+1} + \beta_{j+1} \mathbf{d}_j$

end for

a conjugate gradient method with the fast exact Hessian-vector product in (2.2). Our algorithm uses a modified CG method known as the CG-Steihaug approach [21], or as the Steihaug-Toint truncated conjugate gradient method [22]. which we outline in Algorithm 1.

If the minimizer lies within the trust-region radius, then the CG iterates con-

verge to the unconstrained minimizer (see Fig. 2.1(a)). If the minimizer is outside of the trust region, then the CG iteration terminates where the iterate crosses the boundary 2.1(b)). Finally, if the Hessian is not positive definite, then CG-Steihaug terminates the algorithm and the minimizer is detected at the boundary of the trust region in the direction of the last computed CG iterate (see Fig. 2.1(c)). We re-iterate that the proposed method does not require the Hessian to be positive definite. Therefore, we do not have to use a dampening scalar λ, and consequently, the method allows directions of negative curvature to be detected to avoid saddle points. We describe our proposed approach in Algorithm 2. We note that the final iterate $\mathbf{p}_k$ in Algorithm 2 satisfies the following decrease in the quadratic model:

$$\mathcal{Q}_k(0) - \mathcal{Q}_k(\mathbf{p}_k) \geq c_1 \|\mathbf{g}_k\|_2 \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|_2}{\|\mathbf{H}_k\|_2}\right), \tag{2.4}$$

where c_1 is a constant with $c_1 \in (0, 1]$. Also, we note that the direction $\mathbf{p}_k$ satisfies the trust-region constraint, i.e., $\|\mathbf{p}_k\|_2 \leq \Delta_k$, which will be used for convergence results.

Summary of Proposed Approach. In summary, the proposed approach has three major components: (i) a trust-region method (outlined in Algorithm 2, which allows for indefinite Hessians that avoid saddle points; (ii) a fast and exact Hessian-vector product (2.2) which efficiently provides true second-derivative information at the current iterate; and (iii) the CG-Steihaug approach, which solves the trust-region subproblem without storing the Hessian matrix. To guarantee that the search direction provided by the CG-Steihaug method sufficiently decreases the value of the quadratic subproblem, we implement a Wolfe line search in the direction of $\mathbf{p}_k$ [21]. Furthermore, we evaluate the accuracy of the quadratic model of the objective function within the trust region using the ratio ρ_k in Algorithm 2 to determine whether the update is acceptable or not. The result is an algorithm that considers second derivative information and allows for the possibility of negative curvature. As such, the impact of each iteration is more valuable when compared to those of first-order methods.

Convergence. We conclude with the following convergence guarantee for our proposed method. We first define the level set $S = \{\Theta \colon \mathcal{L}(\Theta) \leq \mathcal{L}(\Theta_0)\}$, where Θ_0

is the initial point, and an open neighborhood of S by $S(R_0) = \{\Theta \colon \|\Theta - \Theta^*\| < R_0 \text{ for some } \Theta^* \in S\}$, where R_0 is a positive constant. Then we have the following result.

Theorem 1. Let $\eta = 0$ in Algorithm 2. Suppose that $\|\mathbf{H}_k\| \leq \beta$ for some constant $\beta > 0$, that $\mathcal{L}(\Theta)$ is bounded below on the level set L and Lipschitz continuously differentiable in the neighborhood $L(R_0)$ for some $R_0 > 0$, and that all feasible solutions $\mathbf{p}_k$ of (2.3) satisfy (2.4). Then we have

$$\liminf_{k \to \infty} \|\mathbf{g}_k\|_2 = 0.$$

The proof follows directly from Theorem 4.5 in [21].

2.3 Limited-Data Experimental Setup

To validate the effectiveness of the proposed algorithm, we implemented Algorithm 1 with the intent to train a neural network to perform a well established classification task. However, we imposed a limitation on the amount of data available for training to simulate data-starvation. Here, we describe our experimental setup.

Neural Network Architecture. In each of these experiments, we used a type of architecture known as a Multi-Layer Perceptron (MLP), which are a class of feed-forward artificial neural networks with the ability to separate data with a non-linear decision boundary [23]. The MLP used in our experiments was implemented using the deep learning package Theano and consisted of three layers of nodes/neurons. The input layer consisted of the vectorized version of the input image (784 nodes), where each node corresponds to each pixel in the sample image. The hidden layer contained 500 neurons and was followed by the non-linear hyperbolic tangent function (tanh) used as a non-linear activation on the output of the layer. Finally, the output layer consisted of 10 neurons corresponding to the 10 classes present in the the dataset. This layer was also followed by the same activation function as the previous layer. The softmax activation function was applied to the output of the neural network in order to provide a probability distribution of the possible

classes. The probability distribution was compared to the true one-hot encoding of the target class using a cross entropy loss function. The architecture was kept simple to demonstrate the effectiveness of the optimization method on the intended task rather than the complexity of the MLP.

Dataset. The dataset used for these experiments is known as the MNIST dataset [24], which consists of rasterized 28×28 pixel images of handwritten digits from 0-9 (10 classes). The collection of images is partitioned in a training set of 60,000 images and a test set of 10,000 images.

Hardware. Experiments were carried out on GPUs housed in a high-performance cluster. The cluster consists of 95 computer nodes with a total of 2116 cores and 2301 Mhz processing power. The GPUs include 4 NVIDIA Tesla K20 graphics cards and 2 Nvidia p100s with a total operating capacity of 62 TFLOPS.

Testing Procedure. The goal of our experiments is to show that the proposed approach can improve on SGD when training a neural network with **limited data**. The 70,000 images in the MNIST dataset were partitioned in the following manner: 50,000 images in the training set, 10,000 images in the validation set and 10,000 images in the testing set. From the training set, we randomly sampled subsets with sizes of 20, 100, 500, 1000 and 10,000 images. When implementing our algorithm, the entire subset is used in a single batch over a single epoch. The algorithm terminates when the norm of the gradient of the objective function reaches a sufficient tolerance (in our case, this was set to 10^{-5}). When using SGD, we approached training using the standard practice of using mini-batches. We chose a mini-batch size of 20 images for all data subsets

2.4 Main Results

The results of our experiments are presented in Fig. 2.2 and Table 1, which presents the average of 10 runs each with each run using a different initial point. When compared to SGD for various training times and various dataset sizes, the proposed method improves on the accuracy of the classification task. In particular, after a single pass (1 epoch) through the available data the proposed method is

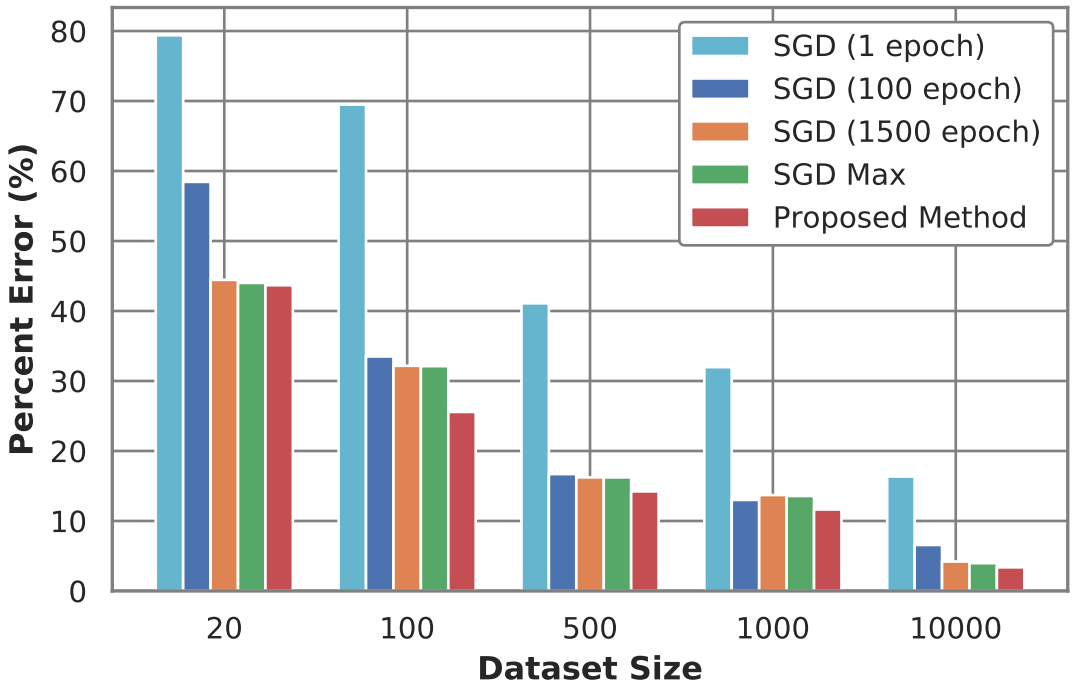


Figure 2.2: A comparison of our method to Stochastic Gradient Descent (SGD) with limited datasets. Here we report the accuracy error for datasets of 20, 100, 500, 1000 and 10000 images. Results are also shown for various numbers of epochs. The proposed approach was only trained for 1 epoch while SGD was allowed to run for 1, 100, and 1500 epochs and SGD Max, which refers to the number of epochs of SGD allowed to run within the time our proposed method runs 1 epoch.

almost twice as accurate as SGD. This confirms the notion that the iterations of the proposed method have a greater impact than that of the first-order method. As we continued to allow the network to train we can see that after almost 1500 epochs of SGD (which may be considered overtraining) we do not see a great improvement in the level of accuracy. **In fact, we even allowed the SGD algorithm to run in the same GPU time as our method, we still improve on the performance of SGD.** The results in Table 1 show that the improvement on SGD in the case of a 20 image dataset is less than 2%. In this case, 20 images might approach the lower limit on the minimum size of the data set you need in order for the network to learn. We can see that as we increase the number of images in the dataset, SGD still underperforms in comparison to our proposed method.

Dataset size	Percent Error				
	SGD 1 epoch	**SGD 100 epochs**	**SGD 1500 epochs**	**SGD Max**	**Proposed Method**
20	79.39	58.45	44.45	44.01	**43.68**
100	69.49	33.50	32.17	32.12	**25.58**
500	41.10	16.70	16.23	16.23	**14.23**
1000	31.98	13.01	13.71	13.57	**11.64**
10000	16.35	6.57	4.21	3.97	**3.38**

Table 2.1: Error table corresponding to the testing error/loss of the neural network for our proposed method in comparison to SGD over various epochs and for different dataset sizes. For our proposed method, the dataset is fed as a batch to the network. For SGD, the data are fed in mini-batches of 20 images. SGD Max corresponds to SGD trained over the same GPU run time as our proposed algorithm.

2.5 Concluding Remarks

In this chapter we proposed a novel algorithm for training neural networks **using second-order information for data-limited inference**. In particular, our algorithm improves on first-order methods by allowing the use of curvature information to improve the quality of each iteration in the optimization method. This is accomplished by using a fast exact Hessian operation, which allows us to efficiently compute matrix-vector multiplication with the exact second-derivative matrix. Unlike previous algorithms that have used this type of algorithm in a conjugate gradient setting, by using the CG-Steihaug approach in a trust-region setting it allows us to relax the requirement that the Hessian be positive definite. This allows the algorithm to detect negative curvature information and avoid saddle points. We provided numerical results in a standard implementation of an MLP classification problem where the training dataset was limited. In all cases, the proposed method improves upon a standard implementation of Stochastic Gradient Descent trained over various epochs.

Chapter 3

Adaptive-Regularized Cubics using Symmetric Rank-one updates

Gradient and adaptive gradient methods are the most widely used for solving (1.1). The most common approach is stochastic gradient descent (SGD) which, despite its simplicity, performs well over a wide range of applications. However, in a sparse training data setting (as can be seen in chapter 2), SGD performs poorly due to limited training speed [25]. To address this problem, *adaptive* methods such as AdaGrad [7], AdaDelta [26], RMSProp [27] and Adam [28] have been proposed. These methods take the root mean square of the past gradients to influence the current step.

In contrast, Newton's method has the potential to exploit curvature information from the second-order derivative (Hessian) matrix (see e.g., [29]). Generally, the iterates are defined by $\Theta_{k+1} = \Theta_k - \alpha_k \eta_k^2 \mathcal{L}(\Theta_k)^{-1} \nabla \mathcal{L}(\Theta_k)$, where $\eta_k > 0$ is a steplength defined by a linesearch criterion ([21]). In a DNN setting, the number of parameters (n) can be of the order of millions. Thus storing the Hessian, which takes $\mathcal{O}(n^2)$ memory units, and inverting it, which requires $\mathcal{O}(n^3)$ operations, become impractical. Thus, full Hessians are rarely ever computed. Instead, Hessian-vector products and Hessian-free methods are used (see e.g., [12]) which reduce the cost of storing the Hessian and inverting it. However, the computational cost remains expensive.

Alternatively, quasi-Newton methods compute Hessian approximations, $\mathbf{B}_k \approx \nabla^2 \mathcal{L}(\Theta_k)$, that satisfy the *secant condition* given by $\mathbf{y}_{k-1} = \mathbf{B}_k \mathbf{s}_{k-1}$, where

$$\mathbf{s}_{k-1} = \Theta_k - \Theta_{k-1} \quad \text{and} \quad \mathbf{y}_{k-1} = \nabla \mathcal{L}(\Theta_k) - \nabla \mathcal{L}(\Theta_{k-1}).$$

The most commonly used quasi-Newton method, including in the realm of deep learning, is the limited-memory BFGS update, or L-BFGS (see e.g., [30]), where the Hessian approximation is given by

$$\mathbf{B}_k = \mathbf{B}_{k-1} + \frac{\mathbf{y}_{k-1}\mathbf{y}_{k-1}^\top}{\mathbf{y}_{k-1}^\top \mathbf{s}_{k-1}} - \frac{\mathbf{B}_{k-1}\mathbf{s}_{k-1}\mathbf{s}_{k-1}\mathbf{B}_{k-1}^\top}{\mathbf{s}_{k-1}^\top \mathbf{B}_{k-1}\mathbf{s}_{k-1}}. \tag{3.1}$$

One advantage of using an L-BFGS update is that the Hessian approximation can be guaranteed to be positive definite, which is highly suitable in line-search settings because the update $\mathbf{s}_k$ is guaranteed to be a descent direction, meaning there is some step length along this direction that results in a decrease in the objective function (see [21], Algorithm 6.1).

Because the L-BFGS update is positive definite, it does not readily detect directions of negative curvature for avoiding saddle points. In contrast, the Symmetric Rank-One (SR1) quasi-Newton update is not guarateed to be positive definite and can result in *ascent* directions for line-search methods. However, in trust-region settings where indefinite Hessian approximations are an advantage because they can capture directions of negative curvature, the limited-memory SR1 (L-SR1) has been shown to outperform L-BFGS in DNNs for classification (see [31]). We discuss this in more detail in Sec. 3.1 but in the context of Adaptive Regularization using Cubics.

Dedication

We dedicate this paper to Oleg P. Burdakov, whose work on shape-changing norms and quasi-Newton methods inspired this work.

3.1 Adaptive Regularization using Cubics with L-SR1 Updates

Here, we describe our proposed approach by first discussing the L-SR1 update.

Limited-memory symmetric rank-one updates. Unlike the BFGS update (3.1), which is a rank-two update, the SR1 update is a rank-one update, which is given by

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)^\top}{\mathbf{s}_k^\top(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)} \tag{3.2}$$

(see [32]). As previously mentioned, $\mathbf{B}_{k+1}$ in (3.2) is not guaranteed to be definite. However, it can be shown that the SR1 matrices can converge to the true Hessian (see [33] for details). We note that the pair $(\mathbf{s}_k, \mathbf{y}_k)$ is accepted only when

$$|\mathbf{s}_k^\top(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)| > \varepsilon\|\mathbf{s}_k\|_2\|\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k\|_2, \tag{3.3}$$

for some constant $\varepsilon > 0$ (see [21], Sec. 6.2, for details). The SR1 update can be

defined recursively as

$$\mathbf{B}_{k+1} = \mathbf{B}_0 + \sum_{j=0}^{k} \frac{(\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j)(\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j)^\top}{\mathbf{s}_j^\top(\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j)}. \tag{3.4}$$

In limited-memory settings, only the last $m \ll n$ pairs of $(\mathbf{s}_j, \mathbf{y}_j)$ are stored and used. For ease of presentation, here we choose $k < m$. We define

$$\mathbf{S}_k = [\ \mathbf{s}_0\ \ \mathbf{s}_1\ \ \cdots\ \ \mathbf{s}_{k-1}\ ] \quad \text{and} \quad \mathbf{Y}_k = [\ \mathbf{y}_0\ \ \mathbf{y}_1\ \ \cdots\ \ \mathbf{y}_{k-1}\ ].$$

Then $\mathbf{B}_k$ admits a compact representation of the form

$$\mathbf{B}_k \ = \ \mathbf{B}_0 + \begin{bmatrix} \\ \boldsymbol{\Psi}_k \\ \\ \end{bmatrix} [\ \mathbf{M}_k] \begin{bmatrix} \quad \boldsymbol{\Psi}_k^\top \quad \end{bmatrix}, \tag{3.5}$$

where $\boldsymbol{\Psi}_k = \mathbf{Y}_k - \mathbf{B}_0\mathbf{S}_k$ and

$$\mathbf{M}_k = (\mathbf{D}_k + \mathbf{L}_k + \mathbf{L}_k^\top - \mathbf{S}_k^\top\mathbf{B}_0\mathbf{S}_k)^{-1},$$

where $\mathbf{L}_k$ is the strictly lower triangular part, $\mathbf{V}_k$ is the strictly upper triangular part, and $\mathbf{D}_k$ is the diagonal part of $\mathbf{S}_k^\top\mathbf{Y}_k = \mathbf{L}_k + \mathbf{D}_k + \mathbf{V}_k$ (see [34] for further details).

Because of the compact representation of $\mathbf{B}_k$, its partial eigendecomposition can be computed (see [35]). In particular, if we compute the QR decomposition of $\boldsymbol{\Psi}_k = \mathbf{QR}$ and the eigendecomposition $\mathbf{RMR}^\top = \mathbf{P}\hat{\boldsymbol{\Lambda}}_k\mathbf{P}^\top$, then we can write

$$\mathbf{B}_k = \mathbf{B}_0 + \mathbf{U}_\parallel\hat{\boldsymbol{\Lambda}}_k\mathbf{U}_\parallel^\top,$$

where $\mathbf{U}_\parallel = \mathbf{QP} \in \mathbb{R}^{n\times k}$ has orthonormal columns and $\hat{\boldsymbol{\Lambda}}_k \in \mathbb{R}^{k\times k}$ is a diagonal matrix. If $\mathbf{B}_0 = \delta_k\mathbf{I}$ (see e.g., Lemma 2.4 in [31]), where $0 < \delta_k < \delta_{\max}$ is some scalar and $\mathbf{I}$ is the identity matrix, then we obtain the eigendecomposition

$$\mathbf{B}_k = \mathbf{U}_k\boldsymbol{\Lambda}_k\mathbf{U}_k^\top = \begin{bmatrix} \mathbf{U}_\parallel & \mathbf{U}_\perp \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{\Lambda}}_k + \delta_k\mathbf{I} & 0 \\ 0 & \delta_k\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{U}_\parallel^\top \\ \mathbf{U}_\perp^\top \end{bmatrix} \tag{3.6}$$

where $\mathbf{U}_k = [\ \mathbf{U}_\parallel\ \ \mathbf{U}_\perp\ ]$ is an orthogonal matrix and $\mathbf{U}_\perp \in \mathbb{R}^{n\times(n-k)}$ is a matrix whose columns form an orthonormal basis orthogonal to the range space of $\mathbf{U}_\parallel$.

Here,

$$(\mathbf{\Lambda}_k)_i = \begin{cases} \delta_k + \hat{\lambda}_i & \text{if } i \leq k \\ \delta_k & \text{if } i > k \end{cases}. \tag{3.7}$$

Adaptive Regularization using Cubics. Since the SR1 Hessian approximation can be indefinite, some safeguard must be implemented to ensure that the resulting search direction $\mathbf{s}_k$ is a descent direction. One such safeguard is to use a "regularization" term. The Adaptive Regularization using Cubics (ARCs) method ([36–38]) can be viewed as an alternative to line-search and trust-region methods. At each iteration, an approximate global minimizer of a local (cubic) model,

$$\min_{\mathbf{s}\in\mathbb{R}^n} m_k(\mathbf{s}) \equiv \mathbf{g}_k^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top \mathbf{B}_k \mathbf{s} + \frac{\mu_k}{3}(\Phi_k(\mathbf{s}))^3, \tag{3.8}$$

is determined, where $\mathbf{g}_k = \nabla\mathcal{L}(\Theta_k)$, $\mu_k > 0$ is a regularization parameter, and Φ_k is a function (norm) that regularizes $\mathbf{s}$. Typically, the Euclidean norm is used. In this work, we use an alternative "shape-changing" norm that allows us to solve each subproblem (3.8) exactly. Proposed in [39], this shape-changing norm is based on the partial eigendecomposition of $\mathbf{B}_k$. Specifically, if $\mathbf{B}_k = \mathbf{U}_k\mathbf{\Lambda}_k\mathbf{U}_k^\top$ is the eigendecomposition of $\mathbf{B}_k$, then we can define the norm

$$\|\mathbf{s}\|_{\mathbf{U}_k} \stackrel{\text{def}}{=} \|\mathbf{U}_k^\top \mathbf{s}\|_3.$$

It can be shown using Hölder's Inequality that

$$n^{-1/6}\|\mathbf{s}\|_2 \leq \|\mathbf{s}\|_{\mathbf{U}_k} \leq \|\mathbf{s}\|_2.$$

Closed-form solution. Applying a change of basis with $\bar{\mathbf{s}} = \mathbf{U}_k^\top \mathbf{s}$ and $\bar{\mathbf{g}}_k = \mathbf{U}_k^\top \mathbf{g}_k$, we can redefine the cubic subproblem as

$$\min_{\bar{\mathbf{s}}\in\mathbb{R}^n} \bar{m}_k(\bar{\mathbf{s}}) = \bar{\mathbf{g}}_k^\top \bar{\mathbf{s}} + \frac{1}{2}\bar{\mathbf{s}}^\top \mathbf{\Lambda}_k \bar{\mathbf{s}} + \frac{\mu_k}{3}\|\bar{\mathbf{s}}\|_3^3. \tag{3.9}$$

With this change of basis, we can find a closed-form solution of (3.9) easily. Note that $\bar{m}_k(\bar{\mathbf{s}})$ is a separable function, meaning we can write $\bar{m}_k(\bar{\mathbf{s}})$ as

$$\bar{m}_k(\bar{\mathbf{s}}) = \sum_{i=1}^{n} \left\{ (\bar{\mathbf{g}}_k)_i(\bar{\mathbf{s}})_i + \frac{1}{2}(\mathbf{\Lambda}_k)_i(\bar{\mathbf{s}})_i^2 + \frac{\mu_k}{3}|(\bar{\mathbf{s}})_i|^3 \right\}.$$

Consequently, we can solve (3.9) by solving one-dimensional problems of the form

$$\min_{\bar{s}\in\mathbb{R}} \ \bar{m}(\bar{s}) = \bar{g}\bar{s} + \frac{1}{2}\lambda\bar{s}^2 + \frac{\mu_k}{3}|\bar{s}|^3, \tag{3.10}$$

where $\bar{g} \in \mathbb{R}$ corresponds to entries in $\bar{\mathbf{g}}_k$ and $\lambda \in \mathbb{R}$ corresponds to diagonal entries in $\mathbf{\Lambda}_k$. To find the minimizer of (3.10), we first write $\bar{m}(\bar{s})$ as follows:

$$\bar{m}(\bar{s}) = \begin{cases} \bar{m}_+(s) = \bar{g}\bar{s} + \frac{1}{2}\lambda\bar{s}^2 + \frac{\mu_k}{3}\bar{s}^3 & \text{if } \bar{s} \geq 0, \\ \bar{m}_-(\bar{s}) = \bar{g}\bar{s} + \frac{1}{2}\lambda\bar{s}^2 - \frac{\mu_k}{3}\bar{s}^3 & \text{if } \bar{s} \leq 0. \end{cases}$$

The minimizer $\bar{s}^*$ of $\bar{m}(\bar{s})$ is obtained by setting $\bar{m}'(\bar{s})$ to zero and will depend on the sign of $\bar{g}$ because $\bar{g}$ is the slope of $\bar{m}(\bar{s})$ at $\bar{s} = 0$, i.e., $\bar{m}'(0) = \bar{g}$. In particular, if $\bar{g} > 0$, then $\bar{s}^*$ is the minimizer of $\bar{m}_-(\bar{s})$, namely $\bar{s}^* = (-\lambda + \sqrt{\lambda^2 + 4\bar{g}\mu})/(-2\mu)$. If $\bar{g} < 0$, then $\bar{s}^*$ is the minimizer of $\bar{m}_+(\bar{s})$, which is given by $\bar{s}^* = (-\lambda + \sqrt{\lambda^2 - 4\bar{g}\mu})/(2\mu)$. Note that these two expressions for $\bar{s}^*$ are equivalent to the following formula:

$$\bar{s}^* = \frac{-2\bar{g}}{\lambda + \sqrt{\lambda^2 + 4|\bar{g}|\mu}},$$

In the original space, $\mathbf{s}^* = \mathbf{U}_k\bar{\mathbf{s}}^*$ and $\mathbf{g}_k = \mathbf{U}_k\bar{\mathbf{g}}_k$. Letting

$$\mathbf{C}_k = \operatorname{diag}(\bar{c}_1, \ldots, \bar{c}_n), \quad \text{where } \ \bar{c}_i = \frac{2}{\lambda_i + \sqrt{\lambda_i^2 + 4|\bar{\mathbf{g}}_i|\mu}}, \tag{3.11}$$

then the solution $\mathbf{s}^*$ in the original space is given by

$$\mathbf{s}^* = \mathbf{U}_k\bar{\mathbf{s}}^* = -\mathbf{U}_k\mathbf{C}_k\mathbf{U}_k^\top\mathbf{g}_k. \tag{3.12}$$

Practical implementation. While computing $\mathbf{U}_\| \in \mathbb{R}^{n\times k}$ in the matrix $\mathbf{U}_k = [\ \mathbf{U}_\| \ \ \mathbf{U}_\perp\ ]$ is feasible since $k \ll n$, computing $\mathbf{U}_\perp$ explicitly is not. Thus, we must be able to compute $\mathbf{s}^*$ without needing $\mathbf{U}_\perp$. First, we define the following quantities

$$\bar{\mathbf{s}}_\| = \mathbf{U}_\|^\top\mathbf{s} \quad \text{and} \quad \bar{\mathbf{s}}_\perp = \mathbf{U}_\perp^\top\mathbf{s},$$

$$\bar{\mathbf{g}}_\| = \mathbf{U}_\|^\top\mathbf{g}_k \quad \text{and} \quad \bar{\mathbf{g}}_\perp = \mathbf{U}_\perp^\top\mathbf{g}_k.$$

Then the cubic subproblem (3.9) becomes

$$\underset{\bar{\mathbf{s}}\in\mathbb{R}^n}{\text{minimize}}\ \bar{m}_k(\bar{\mathbf{s}}) \ = \ \underset{\bar{\mathbf{s}}_\|\in\mathbb{R}^k}{\text{minimize}}\ \bar{m}_\|(\bar{\mathbf{s}}_\|) + \underset{\bar{\mathbf{s}}_\perp\in\mathbb{R}^{n-k}}{\text{minimize}}\ \bar{m}_\perp(\bar{\mathbf{s}}_\perp), \tag{3.13}$$

where

$$\bar{m}_{\|}(\bar{\mathbf{s}}_{\|}) = \bar{\mathbf{g}}_{\|}^{\top}\bar{\mathbf{s}}_{\|} + \frac{1}{2}\bar{\mathbf{s}}_{\|}^{\top}\hat{\mathbf{\Lambda}}_k\bar{\mathbf{s}}_{\|} + \frac{\mu_k}{3}\|\bar{\mathbf{s}}_{\|}\|_3^3, \tag{3.14}$$

$$\bar{m}_{\perp}(\bar{\mathbf{s}}_{\perp}) = \bar{\mathbf{g}}_{\perp}^{\top}\bar{\mathbf{s}}_{\perp} + \frac{\delta_k}{2}\|\bar{\mathbf{s}}_{\perp}\|_2^2 + \frac{\mu_k}{3}\|\bar{\mathbf{s}}_{\perp}\|_3^3. \tag{3.15}$$

We minimize $\bar{m}_{\|}(\bar{s}_{\|})$ in (3.14) similar to how we solved (3.10). In particular, if we let

$$\mathbf{C}_{\|} = \mathrm{diag}(c_1, \ldots, c_n), \text{ where } c_i = \frac{2}{\lambda_i + \sqrt{\lambda_i^2 + 4|(\bar{\mathbf{g}}_{\|})_i|\mu}}, \tag{3.16}$$

then the solution is given by

$$\mathbf{s}_{\|}^* = -\mathbf{C}_{\|}\bar{\mathbf{g}}_{\|}. \tag{3.17}$$

Minimizing $\bar{m}_{\perp}(\bar{s}_{\perp})$ in (3.15) is more challenging. The only restriction on the matrix $\mathbf{U}_{\perp}$ is that its columns must form an orthonormal basis for the orthogonal complement of the range space of $\mathbf{U}_{\|}$. We are thus free to choose the columns of $\mathbf{U}_{\perp}$ as long as they satisfy this restrction. In particular, we can choose the first column of $\mathbf{U}_{\perp}$ to be the normalized orthogonal projection of $\mathbf{g}_k$ onto the orthogonal complement of the range space of $\mathbf{U}_{\|}$, i.e.,

$$(\mathbf{U}_{\perp})_1 = (\mathbf{I} - \mathbf{U}_{\|}\mathbf{U}_{\|}^{\top})\mathbf{g}_k / \|(\mathbf{I} - \mathbf{U}_{\|}\mathbf{U}_{\|}^{\top})\mathbf{g}_k\|_2.$$

If $\mathbf{g}_k \in \mathrm{Range}(\mathbf{U}_{\|})$, then $\bar{\mathbf{g}}_{\perp} = \mathbf{U}_{\perp}^{\top}\mathbf{g}_k = 0$ and the minimizer of (3.15) is $\bar{\mathbf{s}}_{\perp}^* = 0$ (since $\delta_k > 0$ and $\mu_k > 0$). If $\mathbf{g}_k \notin \mathrm{Range}(\mathbf{U}_{\|})$, then $(\mathbf{U}_{\perp})_1 \neq 0$ and we can choose vectors $(\mathbf{U}_{\perp})_i \in \mathrm{Range}(\mathbf{U}_{\|})^{\perp}$ such that $(\mathbf{U}_{\perp})_i^{\top}(\mathbf{U}_{\perp})_1 = 0$ for all $2 \leq i \leq n-k$. Consequently, $\mathbf{U}_{\perp}^{\top}(\mathbf{U}_{\perp})_1 = \kappa\mathbf{e}_1$, where κ is some constant and $\mathbf{e}_1$ is the first column of the identity matrix. Specifically,

$$\kappa\mathbf{e}_1 = \mathbf{U}_{\perp}^{\top}(\mathbf{U}_{\perp})_1 = \mathbf{U}_{\perp}^{\top}\left(\mathbf{U}_{\perp}\mathbf{U}_{\perp}^{\top}\mathbf{g}_k\right) = \mathbf{U}_{\perp}^{\top}\mathbf{g}_k = \bar{\mathbf{g}}_{\perp},$$

which implies $\kappa = \|\bar{\mathbf{g}}_{\perp}\|_2$. Thus $\bar{\mathbf{g}}_{\perp}$ has only one non-zero component (the first component) and therefore, the minimizer $\bar{\mathbf{s}}_{\perp}^*$ of $\bar{m}_{\perp}(\bar{\mathbf{s}}_{\perp})$ in (3.15) also has only one non-zero compoent (the first component as well). In particular,

$$(\bar{\mathbf{s}}_{\perp}^*)_i = \begin{cases} -\alpha^*\|\bar{\mathbf{g}}_{\perp}\|_2 & \text{if } i = 1 \\ 0 & \text{otherwise} \end{cases}$$

where

$$\alpha = \frac{2}{\delta_k + \sqrt{\delta_k^2 + 4\mu\|\bar{\mathbf{g}}_\perp\|_2}}. \tag{3.18}$$

Equivalently, $\bar{\mathbf{s}}_\perp^* = -\alpha^* \bar{\mathbf{g}}_\perp$. Note that the quantity $\|\bar{\mathbf{g}}_\perp\|_2$ can be computed without computing $\bar{\mathbf{g}}_\perp$ from the fact that $\|\mathbf{g}\|_2^2 = \|\bar{\mathbf{g}}_\parallel\|_2^2 + \|\bar{\mathbf{g}}_\perp\|_2^2$.

Combining the expressions for $\bar{s}_\parallel^*$ in (3.17) and for $\bar{\mathbf{s}}_\perp^*$, the solution in the original space is given by

$$\begin{aligned}
\mathbf{s}^* &= \mathbf{U}_\parallel \mathbf{s}_\parallel^* + \mathbf{U}_\perp \mathbf{s}_\perp^* \\
&= -\mathbf{U}_\parallel \mathbf{C}_\parallel \mathbf{U}_\parallel^\top \mathbf{g} - \alpha^* (\mathbf{I}_n - \mathbf{U}_\parallel \mathbf{U}_\parallel^\top)\mathbf{g} \\
&= -\alpha^* \mathbf{g} + \mathbf{U}_\parallel (\alpha^* \mathbf{I} - \mathbf{C}_\parallel)\mathbf{U}_\parallel^\top \mathbf{g}.
\end{aligned}$$

Note that computing $\mathbf{s}^*$ neither involves forming $\mathbf{U}_\perp$ nor computing $\bar{\mathbf{g}}_\perp$ explicitly.

Termination criteria. With each cubic subproblem solved, the iterations are terminated when the change in iterates, $\mathbf{s}_k$, is sufficiently small, i.e.,

$$\|\mathbf{s}_k\|_2 < \tilde{\epsilon}\|\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k\|_2, \tag{3.19}$$

for some $\tilde{\epsilon}$, or when the maximum number of allowable iterations is achieved. The proposed Adaptive Regularization using Cubics with L-SR1 (ARCs-LSR1) algorithm is given in Algorithm 3.

Convergence. Here, we prove convergence properties of the proposed method (ARCs-LSR1 in Algorithm 2). The following theoretical guarantees follow the ideas from [37, 40]. First, we make the following mild assumptions:

A1. The loss function $f(\Theta)$ is continuously differentiable, i.e., $f \in C^1(\mathbb{R}^n)$.
A2. The loss function $f(\Theta)$ is bounded below.

Next, we prove that the matrix $\mathbf{B}_k$ in (3.4) is bounded.

Lemma 3.1.1. *The SR1 matrix $\mathbf{B}_{k+1}$ in (3.4) satsifies*

$$\|\mathbf{B}_{k+1}\|_F \leq \kappa_B \quad \textit{for all } k \geq 1$$

for some $\kappa_B > 0$.

Algorithm 3 Adaptive Regularization using Cubics with Limited-Memory SR1 (ARCs-LSR1)

Given: $\Theta_0, \gamma_2 \geq \gamma_1, 1 > \eta_2 \geq \eta_1 > 0,\ \sigma_0 > 0, \tilde{\epsilon} > 0, k = 0$, and $k_{\max} > 0$

while $k < k_{\max}$ and $\|\mathbf{s}_k\|_2 \geq \tilde{\epsilon}\|\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k\|_2$ **do**

  Obtain $\mathbf{S}_k = [\ \mathbf{s}_0\ \ \cdots\ \ \mathbf{s}_k\ ]$ and $\mathbf{Y}_k = [\ \mathbf{y}_0\ \ \cdots\ \ \mathbf{y}_k\ ]$

  Solve the generalized eigenvalue problem $\mathbf{S}_k^\top \mathbf{Y}_k \mathbf{u} = \hat{\lambda} \mathbf{S}_k^\top \mathbf{S}_k \mathbf{u}$ and let $\delta_k = \min\{\hat{\lambda}_i\}$

  Compute $\mathbf{\Psi}_k = \mathbf{Y}_k - \delta_k \mathbf{S}_k$

  Perform QR decomposition of $\mathbf{\Psi}_k = \mathbf{QR}$

  Compute eigendecomposition $\mathbf{RMR}^\top = \mathbf{P\Lambda P}^\top$

  Assign $\mathbf{U}_\parallel = \mathbf{QP}$ and $\mathbf{U}_\parallel^\top = \mathbf{P}^\top\mathbf{Q}^\top$

  Define $\mathbf{C}_\parallel = \text{diag}(c_1, \ldots, c_k)$, where $c_i = \frac{2}{\lambda_i + \sqrt{\lambda_i^2 + 4\mu|(\bar{\mathbf{g}}_\parallel)_i|}}$ and $\bar{\mathbf{g}}_\parallel = \mathbf{U}_\parallel^\top \mathbf{g}$

  Compute α^* in (3.18)

  Compute step $\mathbf{s}^* = -\alpha^*\mathbf{g} + \mathbf{U}_\parallel(\alpha^*\mathbf{I} - \mathbf{C}_\parallel)\mathbf{U}_\parallel^\top \mathbf{g}$

  Compute $m_k(\mathbf{s}^*)$ and $\rho_k = (f(\Theta_k) - f(\Theta_{k+1}))/m_k(\mathbf{s}^*)$

  Set

$$\Theta_{k+1} = \begin{cases} \Theta_k + \mathbf{s}^* & \text{if } \rho_k \geq \eta_1 \\ \Theta_k, & \text{otherwise} \end{cases}, \quad \text{and}$$

$$\mu_{k+1} = \begin{cases} \frac{1}{2}\mu_k & \text{if } \rho_k > \eta_2, \\ \frac{1}{2}\mu_k(1 + \gamma_1) & \text{if } \eta_1 \leq \rho_k \leq \eta_2, \\ \frac{1}{2}\mu_k(\gamma_1 + \gamma_2) & \text{otherwise} \end{cases}$$

  $k \leftarrow k + 1$

end while

Proof: Using the limited-memory SR1 update with memory parameter m in (3.4), we have

$$\|\mathbf{B}_{k+1}\|_F \leq \|\mathbf{B}_0\|_F + \sum_{j=k-m+1}^{k} \frac{\|(\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j)(\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j)^\top\|_F}{|\mathbf{s}_j^\top(\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j)|}.$$

Because $\mathbf{B}_0 = \delta_k\mathbf{I}$ with $\delta_k < \delta_{\max}$ for some $\delta_{\max} > 0$, we have that $\|\mathbf{B}_0\|_F = \sqrt{n}\delta_{\max}$. Using a property of the Frobenius norm, namely, for real matrices $\mathbf{A}$, $\|\mathbf{A}\|_F^2 = \text{trace}(\mathbf{A}\mathbf{A}^\top)$, we have that $\|(\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j)(\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j)^\top\|_F = \|\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j\|_2^2$. Since the pair $(\mathbf{s}_j, \mathbf{y}_j)$ is accepted only when $|\mathbf{s}_j^\top(\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j)| > \varepsilon\|\mathbf{s}_j\|_2\|\mathbf{y}_j - \mathbf{B}_j\mathbf{s}_j\|_2$, for some constant $\varepsilon > 0$, and since $\|\mathbf{s}_k\|_2 \geq \tilde{\epsilon}\|\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k\|_2$, we have

$$\|\mathbf{B}_{k+1}\|_F \leq \sqrt{n}\delta_{\max} + \frac{m}{\varepsilon\tilde{\epsilon}} \equiv \kappa_B,$$

which completes the proof. □

Given the bound on $\|\mathbf{B}_{k+1}\|_F$, we obtain the following result, which is similar to Theorem 2.5 in [37].

Theorem 3.1.2. *Under Assumptions **A1** and **A2**, if Lemma 3.1.1 holds, then*

$$\liminf_{k\to\infty} \|\mathbf{g}_k\| = 0.$$

Finally, we consider the following assumption, which can be satisfied when the gradient, $\mathbf{g}(\Theta)$, is Lipschitz continuous on Θ.

A3. If $\{\Theta_{t_i}\}$ and $\{\Theta_{l_i}\}$ are subsequences of $\{\Theta_k\}$, then $\|\mathbf{g}_{t_i} - \mathbf{g}_{l_i}\| \to 0$ whenever $\|\Theta_{t_i} - \Theta_{l_i}\| \to 0$ as $i \to \infty$.

If we further make Assumption **A3**, we have the following stronger result (which is based on Corollary 2.6 in [37]):

Corollary 3.1.3. *Under Assumptions **A1**, **A2**, and **A3**, if Lemma 3.1.1 holds, then*

$$\lim_{k\to\infty} \|\mathbf{g}_k\| = 0.$$

Stochastic implementation. Because full gradient computation is very expensive to perform, we impement a stochastic version of the proposed ARCs-LSR1

method. In particular, we use the batch gradient approximation

$$\tilde{\mathbf{g}}_k \equiv \frac{1}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} \nabla \mathcal{L}_i(\Theta_k).$$

In defining the SR1 matrix, we use the quasi-Newton pairs $(\mathbf{s}_k, \tilde{\mathbf{y}}_k)$, where $\tilde{\mathbf{y}}_k = \tilde{\mathbf{g}}_{k+1} - \tilde{\mathbf{g}}_k$ (see e.g., [31]). We make the following additional assumption (similar to Assumption 4 in [31]) to guarantee that the loss function $\mathcal{L}(\Theta)$ decreases over time:

A4. The loss function $\mathcal{L}(\Theta)$ is fully evaluated at every $J > 1$ iterations (for example, at iterates $\Theta_{J_0}, \Theta_{J_1}, \Theta_{J_2}, \ldots$, where $0 \leq J_0 < J$ and $J = J_1 - J_0 = J_2 - J_1 = \cdots$) and nowhere else in the algorithm. The batch size d is increased monotonically if $\mathcal{L}(\Theta_{J_\ell}) > \mathcal{L}(\Theta_{J_{\ell-1}}) - \tau$ for some $\tau > 0$.

With this added assumption, we can show that the stochastic version of the proposed ARCs-LSR1 method converges.

Theorem 3.1.4. *The stochastic version of ARCs-LSR1 converges with*

$$\lim_{k \to \infty} \|\mathbf{g}_k\| = 0.$$

Proof: Let $\widehat{\Theta}_i = \Theta_{J_i}$. By Assumption 4, $\mathcal{L}(\Theta)$ must decrease monotonically over the subsequence $\{\widehat{\Theta}_i\}$ or $d \to |\mathcal{D}|$, where $|\mathcal{D}|$ is the size of the dataset. If the objective function is decreased ι_k times over the subsequence $\{\widehat{\Theta}_i\}_{i=0}^k$, then

$$\mathcal{L}(\widehat{\Theta}_k) = \mathcal{L}(\hat{\Theta}_0) + \sum_{i=1}^{\iota_k} \left\{ \mathcal{L}(\widehat{\Theta}_i) - \mathcal{L}(\widehat{\Theta}_{i-1}) \right\} \leq \mathcal{L}(\widehat{\Theta}_0) - \iota_k \tau.$$

If $d \to |\mathcal{D}|$, then $\iota_k \to \infty$ as $k \to \infty$. By Assumption **A2**, $f(\Theta)$ is bounded below, which implies ι_k is finite. Thus, $d \to |\mathcal{D}|$, and the algorithm reduces to the full ARCs-LSR1 method, whose convergence is guaranteed by Corollary 3.1.3. □

We note that the proof to Theorem 3.1.4 follows very closely the proof of Theorem 2.2 in [31].

Complexity analysis. SGD methods and the related adaptive methods require $\mathcal{O}(n)$ memory storage to store the gradient and $\mathcal{O}(n)$ computational complexity to update each iterate. Such low memory and computational requirements make these

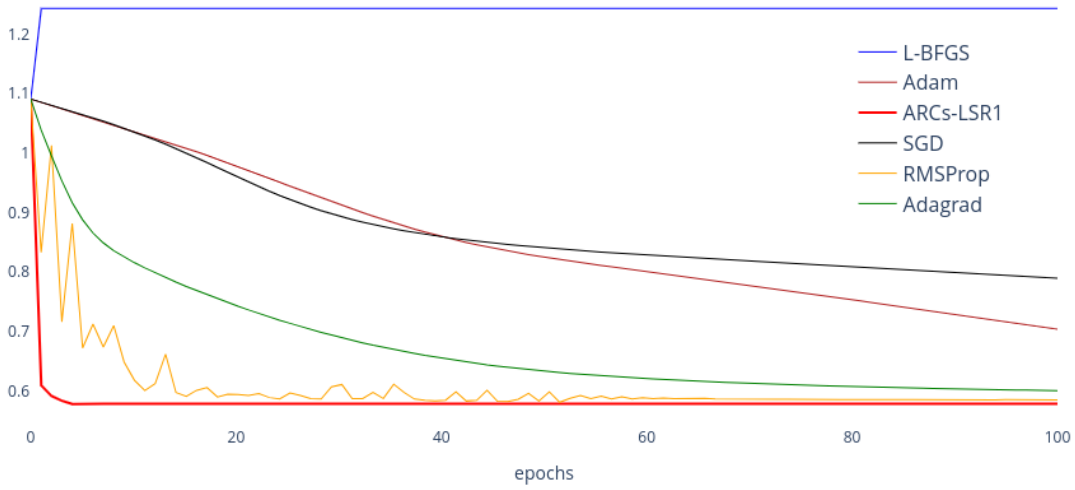
methods easily implementable. Quasi-Newton methods store the previous m gradients and use them to compute the update at each iteration. Consequently, L-BFGS methods require $\mathcal{O}(mn)$ memory storage to store the gradients and $\mathcal{O}(mn)$ computational complexity to update each iterate (see [39] for details). Our proposed ARCs-LSR1 approach also uses $\mathcal{O}(mn)$ memory storage to store the gradients, but the computational complexity to update each iterate requires an additional eigendecomposition of the $m \times m$ matrix $\mathbf{RMR}^\top$, so that the overall computational complexity at each iteration is $\mathcal{O}(m^3 + mn)$. However, since $m \ll n$, this additional factorization does not significantly increase the computational time.

3.2 Numerical Experiments

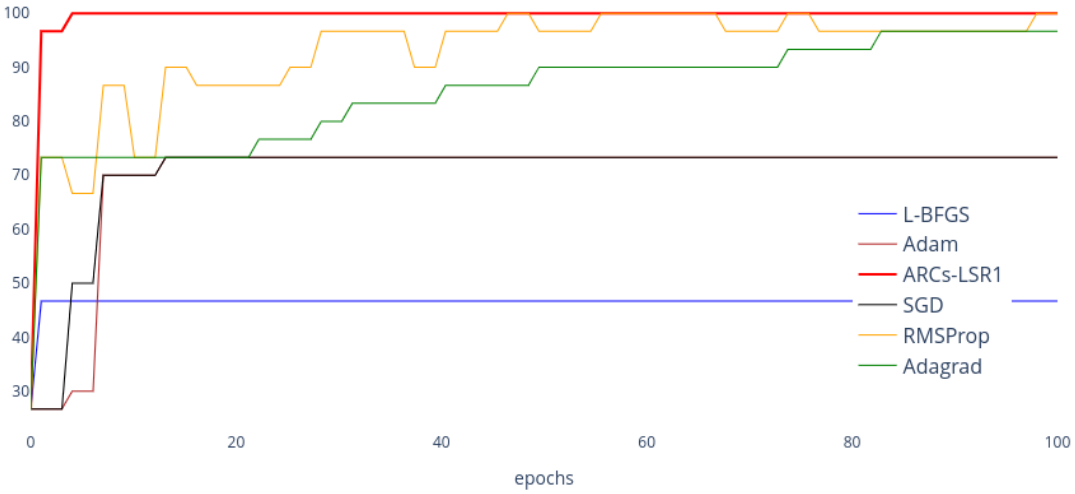
To empirically compare the efficiency of the proposed method with widely-used optimization methods, we focus on three broad deep learning problems: image classification, image reconstruction and language modeling. We choose these tasks due to their importance and availability of reproducible model architectures. We run each experiments on an average of 5 times with a random initialization in each experiment. We performed experiments in three categories: (1) classification, (2) image reconstruction, and (3) language modeling. We define the network achitecture for each experiment and present both the training and testing results for all methods. All experiments were conducted using open-source software PyTorch [41], SciPy [42], and NumPy [43]. We use an Intel Core i7-8700 CPU with a clock rate of 3.20 GHz and an NVIDIA RTX 2080 Ti graphics card.

Optimization approaches. We list the various optimization approaches to which we compared our proposed method. For the numerical experiments, we empirically fine-tuned the hyperparameters and select the best for each update scheme.

1. **Stochastic Gradient Descent (SGD) with Momentum** (see e.g., [44]). A gradient-descent algorithm that uses (i) an estimate of the gradient calculated from a randomly selected subset of the dataset, and (ii) a moving average of these gradient approximations . For the experiments, we used a momentum parameter of 0.9 and a learning rate of 1.0×10^{-1}.



(a) IRIS trainset



(a) IRIS testset

Figure 3.1: The classification accuracy results for **Experiment I**. (a) Training loss of the network. The y-axis represents the negative log-likelihood loss and the x-axis represents the number of epochs. (b) The classification accuracy for each method, i.e., the percentage of testing samples correctly predicted in the testing dataset for each method is presented. Note that the proposed method (ARCs-LSR1) achieves the highest classification accuracy within the fewest number of epochs.

2. **Adaptive Gradient Algorithm (Adagrad)** An algorithm similar to SGD but with an adaptive learning rate for each dimension at each iteration [7]. In our experiments, the initial accumulator value is set to 0, the perturbation ϵ is set to 1.0×10^{-10}, and the learning rate is set to 1.0×10^{-2}.

3. **Root Mean Square Propagation (RMSProp)** An algorithm similar to Adagrad but decays the contribution of older gradients at each iteration [27]. For our experiments, the perturbation ϵ is set to 1.0×10^{-8}. We set $\alpha = 0.99$, and used a learning rate of 1.0×10^{-2}.

4. **Adam** Related to RMSProp, this algorithm generates its parameter updates using a running average of first and second moment of the gradient [28]. For our experiments, we apply an ϵ perturbation of 1.0×10^{-6}. The momentum parameters β_0 and β_1 are chosen to be 0.9 and 0.999, respectively. The learning rate is set to 1.0×10^{-3}.

5. **Limited-memory BFGS (L-BFGS)**: We set the default learning rate to 1.0. The tolerance on function value/parameter change is set to 1.0×10^{-9} and the first-order optimality condition for termination is defined as 1.0×10^{-9}

6. **ARCs-LSR1 (Proposed method)**: For the experiments, we choose the same parameters as those used in L-BFGS.

Dataset. We measure the performance of each optimization method on the following four commonly-used datasets for training and testing in machine learning: (1) IRIS [45], (2) MNIST [24], (3) CIFAR10 [46], and (4) Fashion-MNIST [47].

1. **IRIS:** A dataset consisting of 50 samples from each of three species of the iris flower (iris setosa, iris virginica, and iris versicolor). The features correspond to the length and width of the sepals and petals for each sample [45].

2. **MNIST:** A database of 28×28 grayscale images of handwritten digits from 0 to 9, containing 60,000 training images and 10,000 testing images [24].

3. **CIFAR10:** A dataset consisting of 32×32 color images of 10 mutually exclusive classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship,

and truck) with 6,000 images per class, 50,000 training images, and 10,000 testing images [46].

4. **Fashion-MNIST:** A database of 28×28 grayscale images of articles of clothing associated with a label from 10 classes, containing 60,000 training images and 10,000 testing images [47].

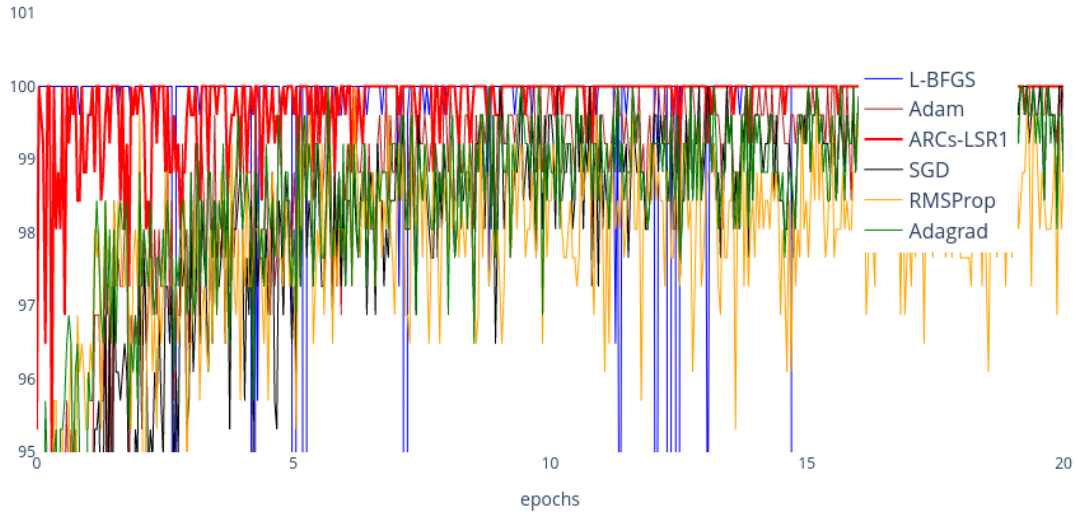
3.3 Results

3.3.1 Experiment I: Image classification

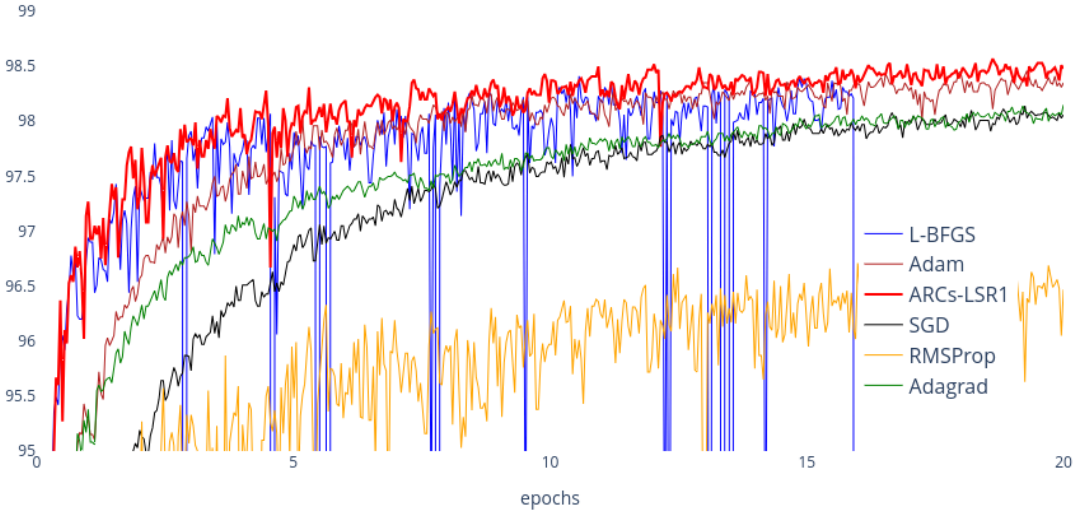
Experiment I.A: IRIS. The IRIS dataset consists of 50 samples of three species of the iris flower. The features correspond to the length and width of the sepals and petals for each sample. This dataset is relatively small; consequently, we only consider a shallow network with three fully connected layers and 2953 parameters. We set the history size and maximum iterations for the proposed approach and L-BFGS to 10. Fig. 3.1(a) shows the comparative performance of all the methods. Note that our proposed method (ARCs-LSR1) achieves the highest classification accuracy in the fewest number of epochs.

Experiment I.B: MNIST. The MNIST classifier is a shallow networks as well, with 3 fully connected layers and 397510 parameters. We train the network for 20 epochs with a batch size of 256 images, keeping the history size and maximum iterations the same as the IRIS experiments for the proposed approach and L-BFGS. Fig. 3.2(b) shows that the proposed ARCs-LSR1 outperforms the other methods.

Experiment I.C: CIFAR10. Because the CIFAR10 dataset contains color images (unlike the MNIST grayscale images), the network used has more layers compared to the previous experiments. The network has 6 convolutional layers and 3 fully connected layers with 62006 parameters. For ARCs-LSR1 and L-BFGS, we have a history size of 100 with a maximum number of iterations of 100 and a batch size of 1024. Fig. 3.3(a) represents the training loss (cross-entropy loss). Fig. 3.3(b) represents the testing accuracy, i.e., number of sample correctly predicted

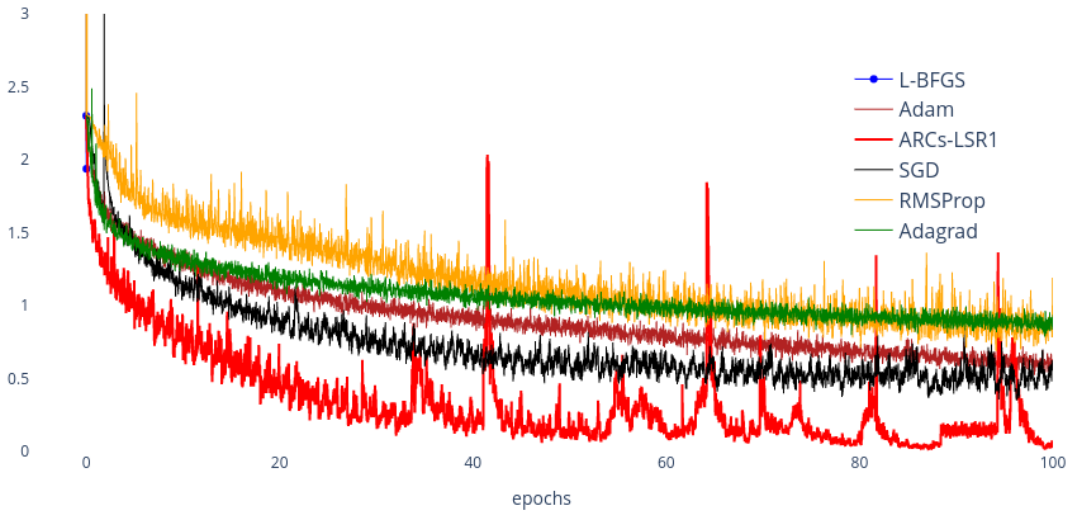


(a) Training accuracy for MNIST dataset

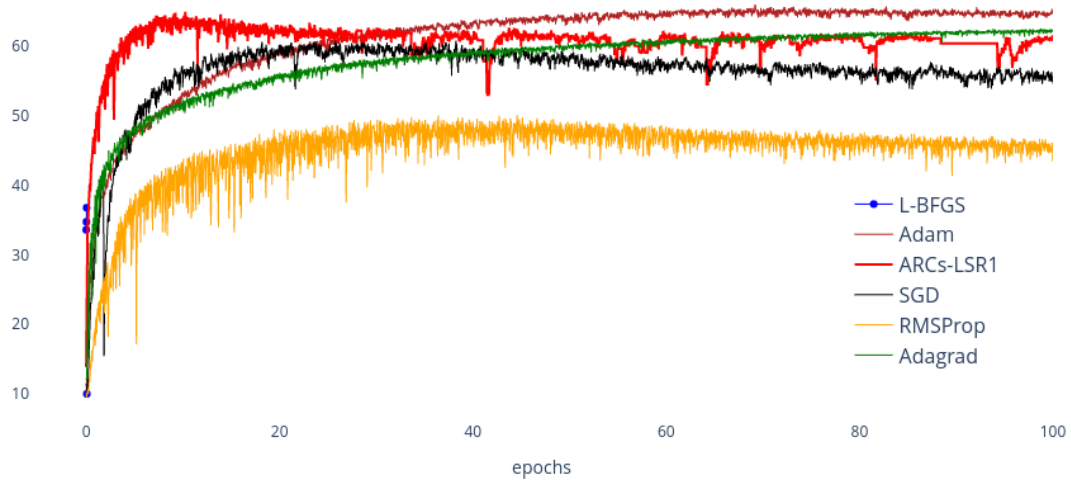


(b) Testing accuracy

Figure 3.2: **Experiment I.B**: MNIST classification. (a) Training accuracy of the network. The y-axis represents the percentage of samples predicted correctly. (b) The classification accuracy of the testing samples correctly predicted. Note that the proposed method (ARCs-LSR1) achieves the highest classification accuracy.



(a) Cross-entropy training loss for CIFAR10 dataset



(b) Classification accuracy

Figure 3.3: The classification results for Experiment I.C: CIFAR10. (a) Training accuracy of the network. The y-axis represents the negative log-likelihood loss, and the x-axis represents the number of epochs. (b) The classification accuracy of the testing samples correctly predicted. The proposed method (ARCs-LSR1) achieves the lowest training loss and highest classification accuracy within the fewest number of epochs.

in the testing set. As can be seen, the proposed approach is able to outperform the other methods.

3.3.2 Experiment II: Image reconstruction

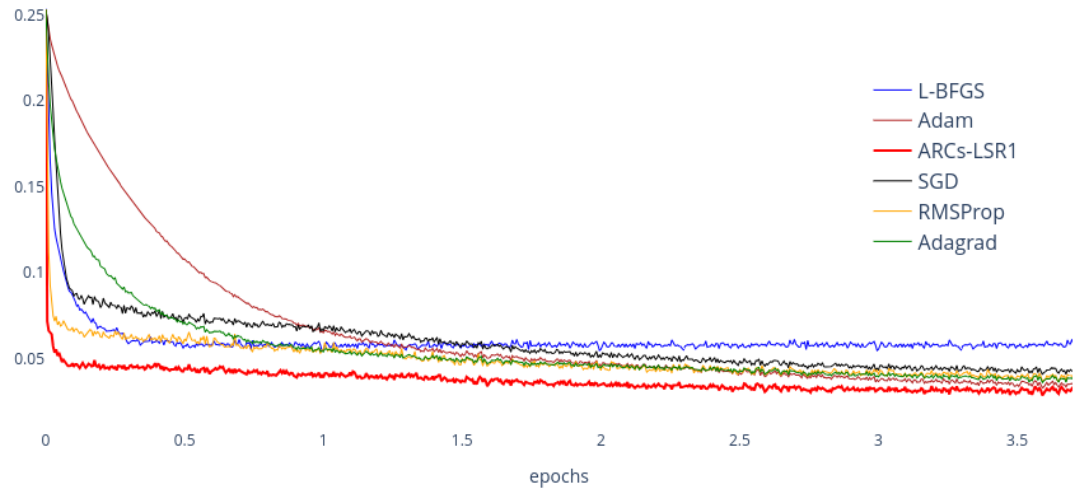
The image reconstruction problem involves feeding a feedforward convolutional autoencoder model a batch of the dataset. The loss function is defined between the reconstructed image and the original image. Additional details on the type of loss function and parameters of the model are described in the following experiments.

Experiment II.A: MNIST. An image $x \in \mathbb{R}^n$ is fed to the network, compressed into a latent space $z \in \mathbb{R}^l$, where $l \ll n$, and reconstructed back to its original image size $\bar{x} \in \mathbb{R}^n$. We compute the mean-squared loss error between the reconstruction and the true image. The network is shallow, with 53415 parameters, which are initialized randomly. We considered a batch size of 256 images and trained over 50 epochs. Each experiment has been conducted 5 times. We provide the training loss results for the early (Fig. 3.4(a)) and late epochs (Fig. 3.4(b)). This is empirical evidence that the method converges to the minimizer in fewer steps in comparison to the adaptive methods.

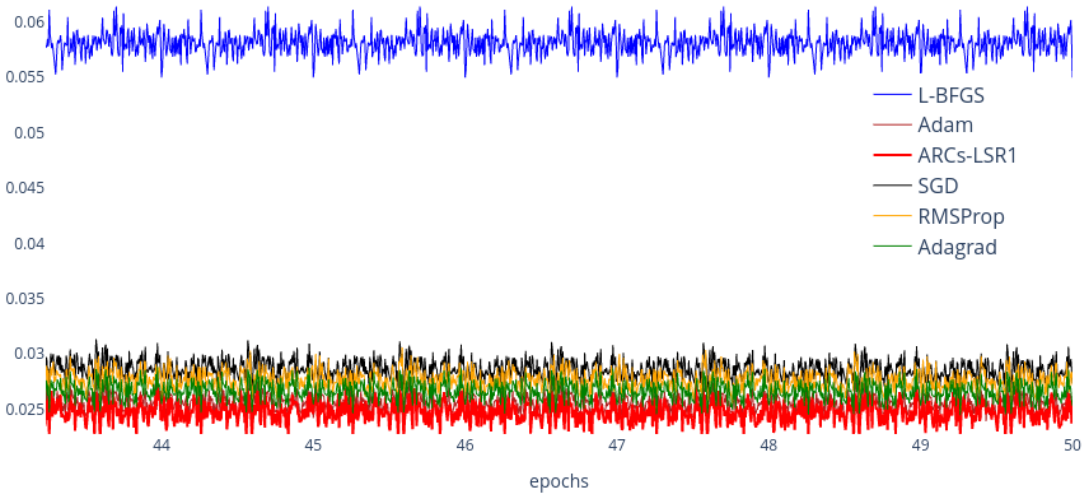
Experiment II.B: FMNIST. In this experiment, we follow the same approach as Experiment II.A and chose batch sizes of 256. Figs. 3.5(a) and 3.5(b) show the training response for the early and late epochs, respectively, and Figs. 3.5(a) and 3.5(b) show the testing response for the early and late epochs, respectively. We note that the convergence behavior of the methods are similar to those in Experiment II.A. The iterates generated by the proposed approach significantly decreases the objective function. Even after 50 epochs, we can see that the proposed approach has significantly reduced the objective loss function (see Fig. 3.5(b)). The network is capable of generalizing on a testing dataset as well in comparison to all other adaptive and quasi-Newton methods. For more details, please refer Fig. 3.5.

3.3.3 Experiment III: Natural language modeling

We conducted word level predictions on the Penn Tree Bank (PTB) dataset [48]. We used a state-of-the-art Long-Short Term Memory (LSTM) network which has 650 units per layer and its parameters are uniformly regularized in the range [-0.05, 0.05]. For more details on implementation, please refer [49]. For the ARCs-LSR1

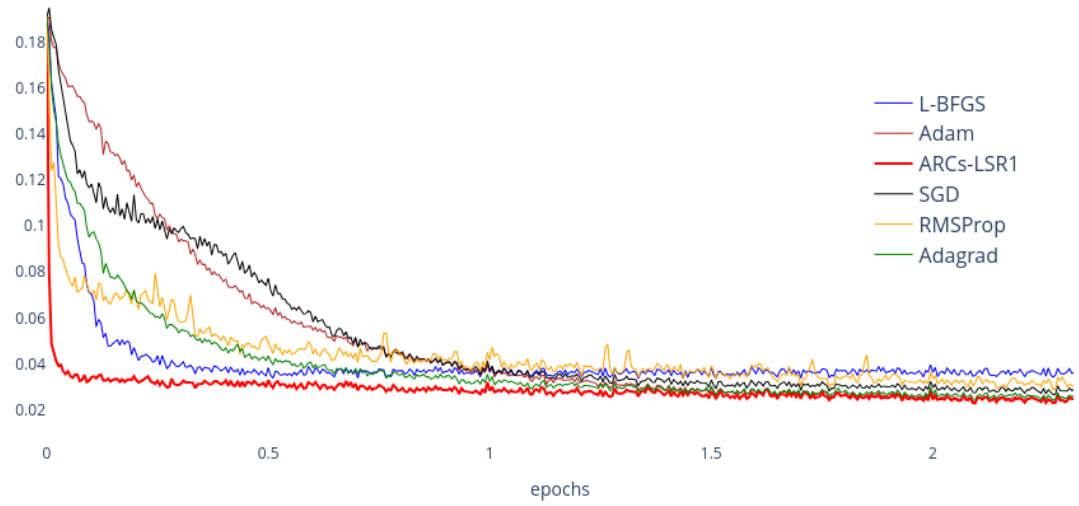


(a) MNIST training loss for early epochs

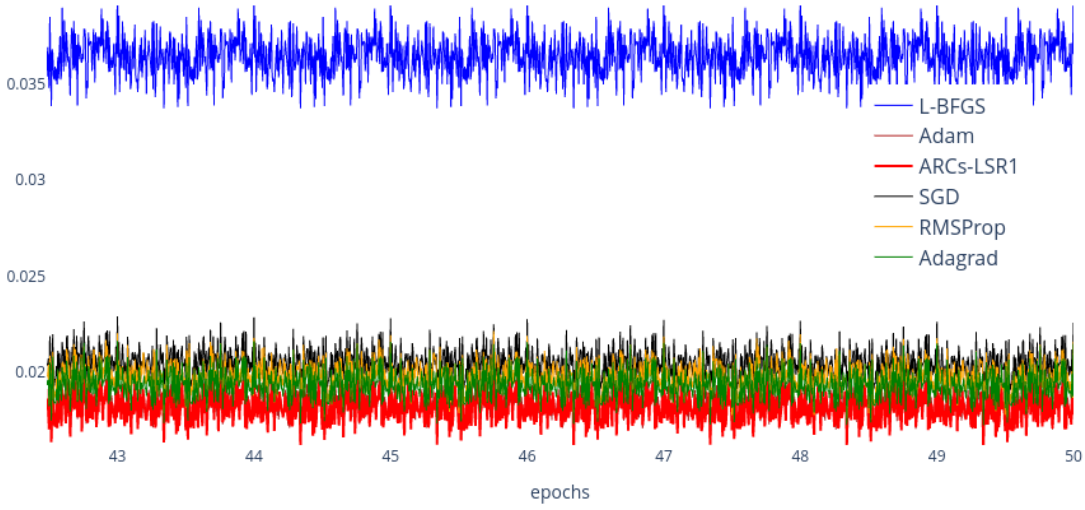


(b) MNIST training loss for late epochs

Figure 3.4: The image reconstruction results for Experiment II.A: MNIST. (a) Initial training loss. The y-axis represents the Mean-Squared Error (MSE) loss from the first four epochs. (b) Final training loss from epochs 43 to 50. Note that the proposed method (ARCs-LSR1) achieves the lowest training loss.



(a) Early epoch FMNIST training loss



(b) Late epoch FMNIST training loss

Figure 3.5: The image reconstruction results for Experiment II.B: FMNIST (a) Initial training loss. The y-axis represents the Mean-Squared Error (MSE) loss in the first three epochs. (b) Final training loss from epochs 42 to 50. Note that the proposed method (ARCs-LSR1) achieves the lowest training loss.

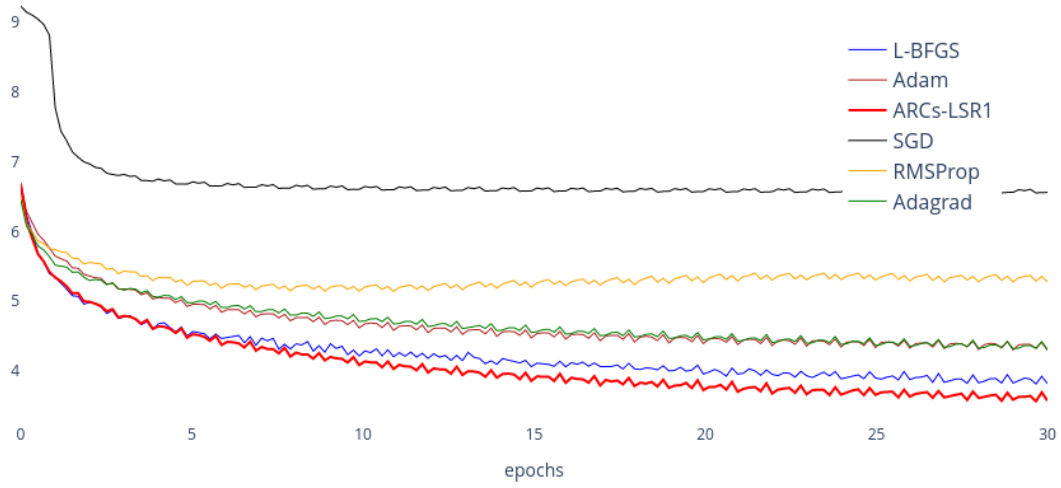


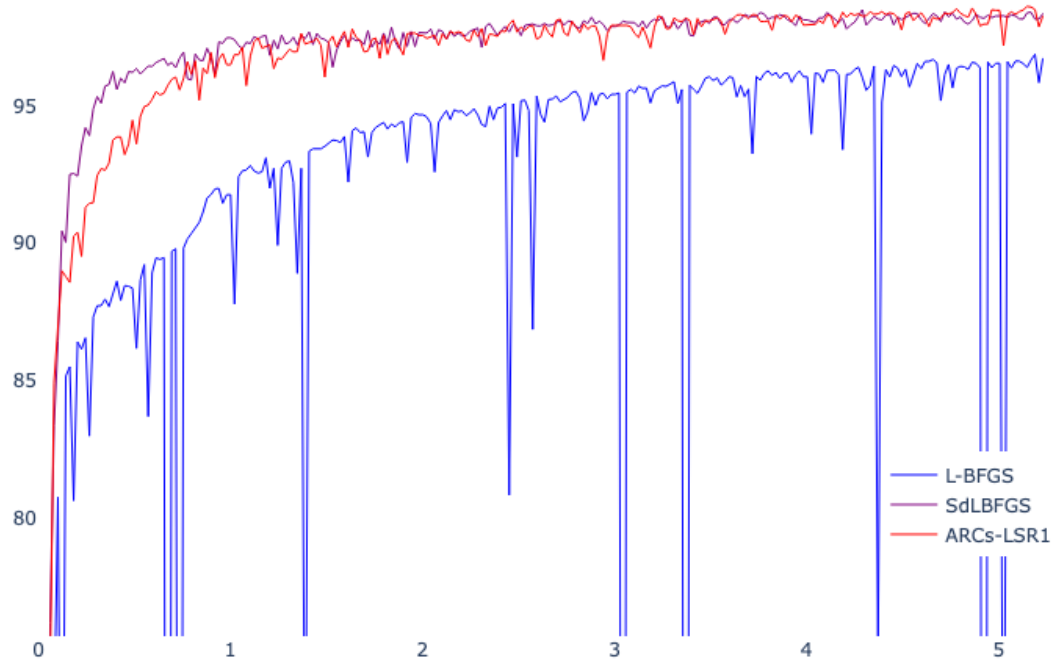
Figure 3.6: The prediction loss for **Experiment III: Penn Tree Bank**. The y-axis represents the cross-entropy loss, and the x-axis represents the number of epochs. Note that the proposed method (ARCs-LSR1) achieves the lowest loss.

method and the L-BFGS, we use a history size of 5 over 4 iterations. The prediction loss results are shown in Fig. 3.6. In contrast to the previous experiments, here, both quasi-Newton methods (L-BFGS and ARCs-LSR1) outperform the adaptive methods, with the proposed method (ARCs-LSR1) achieving the lowest cross-entropy prediction loss.

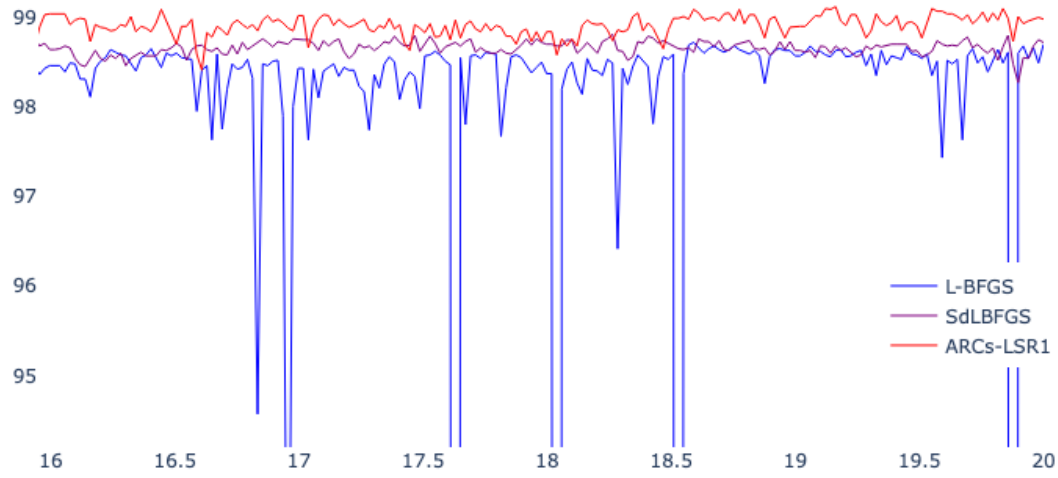
3.3.4 Experiment IV: Comparison with Stochastically Damped L-BFGS

In the experiments above on image classification and reconstruction, the L-BFGS approach performs poorly, which can be attributed to noisy gradient estimates and non-convexity of the problems. To tackle this, a *stochastically damped* L-BFGS (SdLBFGS) approach has been proposed [50], which adaptively generates a variance reduced, positive definite approximation of the Hessian. We compare the proposed approach to L-BFGS and SdLBFGS on the MNIST classification problem. As can be seen from Fig. 3.7(a), we are able to achieve a comparable performance to the stochastic version. In fact, in later epochs (see Fig. 3.7(b)), we

are able achieve higher accuracy.



(a) Epochs 0-5



(b) Epochs 16-20

Figure 3.7: The prediction loss for **Experiment IV**: Comparison with stochastically damped L-BFGS. The x-axis represents the number of epochs and the y-axis represents the accuracy of prediction. (a) Accuracy for epochs 0-5. (b) Accuracy for epochs 16-20.

Computational time analysis. We understand that the proposed approach performs competitively against all existing methods. We now analyze the time-constraints of each method. We choose to clock Experiment 3. We chose a maximum iterations of 100 with a history size of 100 for L-BFGS and ARCs-LSR1, with a batch size of 1024 images. Fig. 3.8 shows the time required by each of the methods to reach a non-overtrained minima. Note that the proposed approach reaches the desired minima in much less time than the other algorithms. L-BFGS does not converge perhaps due to a very noisy loss function and a small batch size, thus causing the algorithm to break. (see e.g., [51]). argue that a large batch size is required for quasi-Newton methods to perform well. However, the ARCs-LSR1 method performs well even with a small batch size.

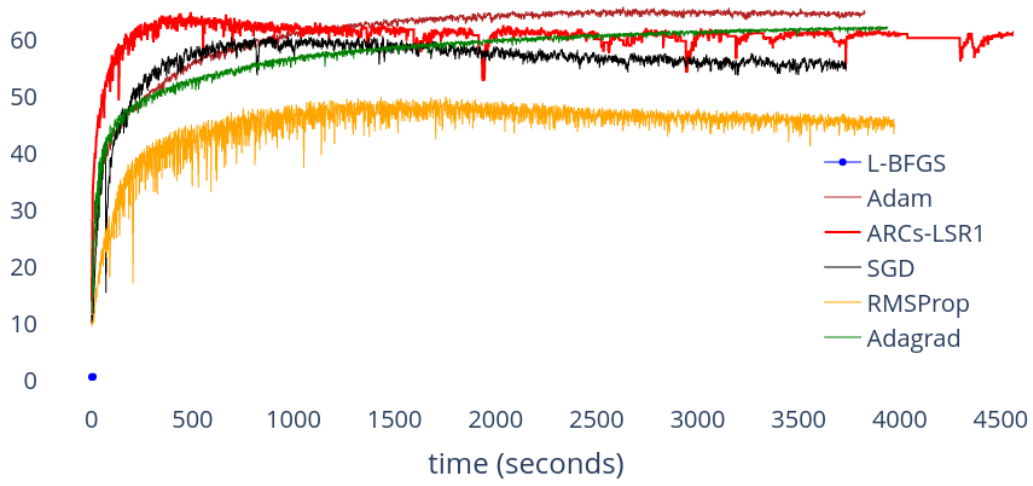


Figure 3.8: Timing analysis for Experiment 3: CIFAR10. The x-axis is time in seconds, and the y-axis is the accuracy of prediction in percentage. Note that the proposed method (ARCs-LSR1) achieves the highest accuracy within the shortest amount of time.

3.4 Conclusion

In this chapter, we proposed a novel quasi-Newton approach in an adaptive reguliarization using cubics (ARCs) setting using the less frequently used limited-

memory Symmetric Rank-1 (L-SR1) update and a shape-changing norm to define the regularizer. This shape-changing norm allowed us to solve for the minimizer exactly. We have provided convergence guarantees for the proposed ARCs-LSR1 method and analyzed its computational complexity. In a set of experiments in classification, image reconstruction, and language modeling, we demonstrated that ARCs-LSR1 achieves higher accuracy in fewer epochs than a variety of existing state-of-the-art optimization methods.

Chapter 4

Quasi-Adam

4.1 Introduction

Gradient-based optimization methods are one of the most commonly used approaches in deep learning. This is owing to their fast computational nature (computing the gradient is asymtotically equal to a forward pass through the neural network) and stochastic nature (the gradients are evaluated only at certain input points). Due to these factors, it was able to propel research advances in the intersection of deep learning and optimization (see [52]). In recent years, a mutlitude of exponentially moving average and moment approaches have been proposed to optimize neural networks (see e.g. [7, 28, 53, 54]). The main objective of exponentially moving average is to limit the reliance of the update on the past gradient information instead of recent gradient information.

Quasi-Newton approximations, on the other hand, explicitly use information from the past (steps and change in gradients) to build an approximation of the Hessian. This approximation induces the curvature information using the secant information. We discuss this in detail in Sec. 4.3. However, computing a step using quasi-Newton updates can be expensive due to its size and operations required. To overcome this, a limited-memory approach is generally used. Limited-memory BFGS (L-BFGS) is a very common approach where the Hessian approximation always stays positive-definite. We discuss this further in Sec. 4.3. In recent work, quasi-Newton approaches have proven to be more deep learning friendly (see [55]).

In this chapter, we propose *quasi-Adam*, a combination of an exponentially moving average and moment approach with the L-BFGS quasi-Newton update. The chapter is divided into the following sections: In Sec. 4.2, we discuss exponential moving average methods such as Adam and AdaGrad. In Sec. 4.3, we discuss the quasi-Newton approaches, and their compact-representations. In Sec. 4.4, we provide the pseudo-code of the proposed approach and discuss the space and time-complexity of it. In Sec. 4.6, we discuss the experimental setup, testbed, datasets we will be using and models used for each dataset and the results of these experiments. In Sec. 4.7, we discuss the results obtained in Sec. 4.6 and provide explanation and hypotheses based on the results. In Sec. 4.8, we finally provide our concluding statements.

Notation. We denote the gradient of $f(\Theta)$ in (1.1) at the t^{th} iteration by $\mathbf{g}_t = \nabla f(\Theta_t)$ and the Hessian approximation by $\mathbf{B}_t \approx \nabla^2 f(\Theta_t) \in \mathbb{R}^{n \times n}$. We denote the exact inverse of $\mathbf{B}_t$ by $\mathbf{H}_t$, i.e., $\mathbf{H}_t = \mathbf{B}_t^{-1}$. The learning rate is denoted by α_t, and the scalar g_t^2 corresponds to $g_t^2 = \|\mathbf{g}_t\|_2^2$. The matrix $\mathbf{I}$ is the $n \times n$ identity matrix.

4.2 Exponential moving average methods

Given a loss function $f(\Theta)$, gradient-based optimization approaches generate a sequence of iterates $\{\Theta_t\}$ that are computed using the following update:

$$\Theta_{t+1} = \Theta_t + \eta_t \mathbf{p}_t,$$

where η_t is the learning rate and $\mathbf{p}_t$ is the search direction. For gradient-descent methods, $\mathbf{p}_t = -\mathbf{g}_t$. For highly nonlinear and nonconvex functions, such as the typical neural network loss function, large learning rate (or step sizes) do not guarantee a reduction in the loss function, causing a non-monotone behavior within the loss function. Likewise, a small learning rate leads to slow convergence. To tackle this problem, the concept of momentum and moment was introduced.

Momentum is the process of weighted averaging the gradients $\mathbf{g}_t$ over time t. Given the initial momentum vector $\mathbf{m}_0$ be initialized as a vector of zeros, the expression for $\mathbf{m}_t$ can be written as

$$\mathbf{m}_t = \Psi(\mathbf{g}_t, \mathbf{m}_{t-1}),$$

where $t \in [1, T]$ and Ψ computes the weighted average between $\mathbf{g}_t$ and $\mathbf{m}_{t-1}$. This was first introduced by [56] and adapted by [7] and [Sutskever, unpublished 2012] in a deep learning setting, with some minor modifications. The motivation was to damp the non-monontone behaviour in regions of high curvature by averating over gradients with conflicting directions.

Hinton further improved upon the momentum based approach (see [27]) by employing a Root-Mean-Square (RMS) moving average which computes the weighted sum of g_t^2 over t. Given the initial moment $v_0 = 0$, the expression for v_t can be written as

$$v_t = \Phi(g_t^2; v_{t-1}),$$

where Φ is a weighted sum of v_{t-1} and g_t^2. The generalized expression for an exponential moving average update is written as

$$\mathbf{p}_t = -\frac{\Psi(\mathbf{g}_t, \mathbf{m}_{t-1})}{\Phi(g_t^2, v_t)}. \tag{4.1}$$

This moment term allows for the gradient to be normalized (in some way), helping the learning rate to work better. This propelled the use of momentum and moments in most mordern deep learning optimization algorithms.

The first major breakthrough was brought out by [28] who proposed Adam. The authors introduced an exponential decaying approach to the gradient update and the exponential moving average update employing and employed different weighted averages on the momentum and their moments.

Given $\beta_1, \beta_2 \in (0, 1)$, the momentum vector $\mathbf{m}_t$ is defined as

$$\mathbf{m}_t = \frac{1}{(1-\beta_1^t)}\bigg(\beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t\bigg), \tag{4.2}$$

and v_t is defined as

$$v_t = \frac{1}{(1-\beta_2^t)}\bigg(\beta_2 v_{t-1} + (1-\beta_2)g_t^2\bigg). \tag{4.3}$$

Using (4.2) and (4.3), the Adam update is given by

$$\mathbf{p}_{\text{Adam}} = -\frac{\mathbf{m}_t}{\sqrt{v_t + \epsilon}}, \tag{4.4}$$

where $\epsilon = 10^{-8}$ is a scalar.

Thus, Adam uses an exponential moving average of the momentum and moments. Through the remainder of the chapter, we explore the questions - Can we imporove upon an exponential moving average algorithm by inducing an approximation to the curvature information ? In Sec. 4.3, we explore quasi-Newton appraoches to answer these questions.

4.3 Quasi-Newton methods

Second-order approaches have the potential to exploit curvature information from second-order (Hessian) matrices. The iterate updates are defined using

$$\mathbf{p}_t = -[\nabla^2 f(\Theta_t)]^{-1}\mathbf{g}_t.$$

We note that the Hessian matrix $[\nabla^2 f(\Theta_t)]^{-1}$ is $n \times n$, which is computationally infeasible to form when n is very large. To tackle with the dimensionality problem, users generally resolve to a finite difference method (see [12]) or the Pearlmutter technique (see [13]). These methods can be used in conjunction with a trust-region type approach (see [57]), which safeguards the step size, and a conjugate-gradient method which requires only *Hessian-vector* products without explicitly forming the Hessian. However, using the true Hessian can give rise to other issues, such as matrix singularity and non-positive definiteness.

Quasi-Newton approaches, on the other hand, only use approximations to the Hessian, which satisfy the *secant equation* given by

$$\mathbf{y}_{t-1} = \mathbf{B}_t \mathbf{s}_{t-1}, \tag{4.5}$$

where

$$\mathbf{y}_{t-1} = \nabla\mathcal{L}(\Theta_t) - \nabla\mathcal{L}(\Theta_{t-1}) \quad \text{and} \quad \mathbf{s}_{t-1} = \Theta_t - \Theta_{t-1}.$$

The L-BFGS method (see [58]) is one of the most commonly used quasi-Newton updates for $\mathbf{B}_t$ due to the guaranteed positive-definiteness of $\mathbf{B}_t$. Since we only work with the inverse of the Hessian to find the direction of descent, we will be only working with $\mathbf{H}_t = \mathbf{B}_t^{-1}$.

The matrix $\mathbf{H}_t$ is recursively defined as

$$\mathbf{H}_t = \left(\mathbf{I} - \frac{\mathbf{s}_{t-1}\mathbf{y}_{t-1}^\top}{\mathbf{y}_{t-1}^\top \mathbf{s}_{t-1}}\right)\mathbf{H}_{t-1}\left(\mathbf{I} - \frac{\mathbf{y}_{t-1}\mathbf{s}_{t-1}^\top}{\mathbf{y}_{t-1}^\top \mathbf{s}_{t-1}}\right) + \frac{\mathbf{s}_{t-1}\mathbf{s}_{t-1}^\top}{\mathbf{y}_{t-1}^\top \mathbf{s}_{t-1}}, \tag{4.6}$$

with

$$\mathbf{H}_0 = \frac{\mathbf{y}_0^\top \mathbf{s}_0}{\mathbf{y}_0^\top \mathbf{y}_0}\mathbf{I}.$$

We observe here that $\mathbf{H}_t$ represents an $n \times n$ matrix, which can get computationally expensive to store. Hence, this matrix is never stored explicitly. Rather, we only store the steps $\mathbf{s}_{t-1}$ and the change in gradients $\mathbf{y}_{t-1}$. Since (4.6) is only a two-rank update, it can be written as

$$\mathbf{H}_t = \gamma_{t-1}\mathbf{I} + \mathbf{\Gamma}_{t-1}\mathbf{M}_{t-1}\mathbf{\Gamma}_{t-1}^\top, \tag{4.7}$$

where $\gamma_{t-1} = \mathbf{y}_{t-1}^\top \mathbf{s}_{t-1} / \mathbf{y}_{t-1}^\top \mathbf{y}_{t-1}$,

$$\mathbf{\Gamma}_{t-1} = \begin{bmatrix} \mathbf{s}_{t-1} & \gamma_{t-1}\mathbf{y}_{t-1} \end{bmatrix},$$

and

$$\mathbf{M}_{t-1} = \begin{bmatrix} \rho_{t-1} + \gamma_{t-1}\rho_{t-1}^2 \|\mathbf{y}_{t-1}\|_2^2 & -\rho_{t-1} \\ -\rho_{t-1} & 0 \end{bmatrix},$$

with $\rho_{t-1} = (\mathbf{s}_{t-1}^\top \mathbf{y}_{t-1})^{-1}$. The search direction computed using the L-BFGS update is given by

$$\mathbf{p}_{\text{L-BFGS}} = -\mathbf{H}_t \mathbf{g}_t. \tag{4.8}$$

The steps are only accepted when $\mathbf{H}_t$ is positive definite, which is imposed when $\mathbf{s}_{t-1}^\top \mathbf{y}_{t-1} > 0$. Thus the matrix is invertible and provides a direction of descent. The L-BFGS method is presented in Algorithm 4.

Algorithm 4 L-BFGS Method

Require: $\mathbf{s}_0, \mathbf{y}_0$ (First step and first change in gradient)

for $t \in 1, 2, \ldots$ **do**
  Compute $\gamma_{t-1} = \frac{\mathbf{s}_{t-1}^\top \mathbf{y}_{t-1}}{\mathbf{y}_{t-1}^\top \mathbf{y}_{t-1}}$
  Compute $\mathbf{s}_{t-1} = \Theta_t - \Theta_{t-1}$
  Compute $\mathbf{y}_{t-1} = \nabla\mathcal{L}(\Theta_t) - \nabla\mathcal{L}(\Theta_{t-1})$
  if $\mathbf{s}_{t-1}^\top \mathbf{y}_{t-1} > 0$ **then**
    Compute $\mathbf{p}_{\text{L-BFGS}}$ using (4.7)
    Replace $\mathbf{s}_{t-1} = \mathbf{s}_t$ and $\mathbf{y}_{t-1} = \mathbf{y}_t$
  end if
end for

Recently, practical L-BFGS methods have been proposed in a deep learning setting (see [50, 55, 59]). However, it is a common problem that L-BFGS performs very poorly on a variety of stochastic problems because of the use of stochastic gradients.

4.4 Proposed approach

From Sec. 4.2 and Sec. 4.3, we take motivation from both approaches and define our new update rule called **quasi-Adam**. We present the update step in Algorithm 5. The approach uses a combination of both directions $\mathbf{p}_{\text{Adam}}$ and

Algorithm 5 Quasi-Adam Method

Require: $\alpha, \beta_1, \beta_2 \in [0, 1), f(\Theta), \Theta_0, m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$

while $\Theta_t \neq \Theta^*$ **do**

Compute $\mathbf{p}_{\text{Adam}}$ step using (4.4).

Compute $\mathbf{p}_{\text{L-BFGS}}$ step using (4.8).

Update $\Theta_{t+1} \leftarrow \Theta_t - \eta_t(\mathbf{p}_{\text{Adam}} + \mathbf{p}_{\text{L-BFGS}})$.

end while

$\mathbf{p}_{\text{L-BFGS}}$ to improve the current step. In particular, the update step is given by

$$\Theta_{t+1} = \Theta_t - \eta_t \left(\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{v}_t + \epsilon}} + \mathbf{H}_t \mathbf{g}_t \right). \tag{4.9}$$

For the proposed approach, we use a memory size of 1. In the following sections, we discuss the space and time complexity of the proposed approach. The space and time complexity is presented as a modification to Adam - we only discuss the additional overhead for the proposed approach to Adam.

Space Complexity: Since we are only using 1 memory from the past, the space complexity is limited to $\mathcal{O}(n)$. We consume $\mathcal{O}(n)$ memory for saving the previous iterates weights from the model and $\mathcal{O}(n)$ for saving the gradients from past iterates, which gives us $\mathcal{O}(2n) \approx \mathcal{O}(n)$ asymptotically.

Time complexity: We need to perform the matrix vector product $\mathbf{H}_t \mathbf{g}_t$ in (4.8). The matrix $\mathbf{\Gamma}_{t-1}^\top$ is $2 \times n$, which means $\mathbf{\Gamma}_{t-1}^\top \mathbf{g}_t$ requires $\mathcal{O}(2n)$ operations. Each element in $\mathbf{M}_{t-1}$ is a scalar, which means $\mathbf{M}_{t-1}$ is a 2×2 matrix. Thus $\mathbf{M}_{t-1} \mathbf{\Gamma}_{t-1}^\top \mathbf{g}_t$ can be computed in $\mathcal{O}(4 + 2n)$ operations. Finally $\mathbf{\Gamma}_{t-1} \mathbf{M}_{t-1} \mathbf{\Gamma}_{t-1}^\top \mathbf{g}_t$ can be computed in $\mathcal{O}(4 + 4n)$ operations.

4.5 Convergence

We analyze the convergence of the proposed approach using the framework by [60]. Given an arbitrary sequence of convex functions $\mathcal{C} = \{c_1, c_2, \ldots, c_T\}$, the goal is to predict the parameter Θ_t by optimizing it over the previous function c_{t-1}. This process of identifying an optimal Θ_t over c_{t-1} is defined as an *online* algorithm. If Θ_t is selected by an algorithm A, we define the cost incurred by the

algorithm A as

$$L_A(T) = \sum_{t=1}^{T} c_t(\Theta_t). \tag{4.10}$$

In order to define the *offline* algorithm, we define a feasible convex set $\mathcal{F}$.

Definition 1. *A set $\mathcal{F} \subseteq \mathbb{R}^n$ is convex if for all $\Theta, \Theta' \in \mathcal{F}$, $r\Theta + (1-r)\Theta' \in \mathcal{F}$ for all $r \in [0,1]$.*

When the information on $\mathcal{C}$ and the convex subset $\mathcal{F}$ is available, the process of identifying the optimal $\Theta \in F$ is defined as *offline* algorithm (often also described as a static feasible solution).

Now, we formally introduce and define the *regret* function $R_A(T)$.

Definition 2. *Given an algorithm A and a convex programming problem $(\mathcal{F}, \mathcal{C})$, if $\{\Theta_1, \Theta_2 \ldots\}$ are vectors selected by algorithm A, then the cost of A until time T is given by* (4.10)*. The cost of a static feasible solution $\Theta \in \mathcal{F}$ until time T is given by*

$$L_\Theta(T) = \sum_{t=1}^{T} c_t(\Theta).$$

The regret of an algorithm A until time T is defined as

$$R_A(T) = L_A(T) - \min_{\Theta \in \mathcal{F}} L_\Theta(T).$$

The goal is to prove that the average regret $R_A(T)/T$ approaches 0 as $T \to \infty$.

For this, we begin by expanding the proposed update in (4.9):

$$\Theta_{t+1} = \Theta_t - \frac{\alpha_t}{1-\beta_1^t}\left(\frac{\beta_{1,t}}{\sqrt{\hat{v}_t}}\hat{\mathbf{m}}_{t-1} - \frac{\beta_{1,t}}{\sqrt{\hat{v}_t}}\mathbf{g}_t + \mathbf{H}\mathbf{g}_t\right),$$

where $\beta_{1,t} = \beta_1 \lambda^{t-1}, \lambda \in (0,1)$. Here, $\hat{v}_t$ is the exponential moving average, defined as

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t},$$

with $\beta_2 \in (0,1]$, $\hat{m}_t$ is the bias-corrected first moment estimate defined as

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t},$$

where $\beta_1 \in (0, 1]$, m_t is the biased first moment estimate given by

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t,$$

v_t is the biased second moment update given by

$$v_t = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} g_i^2,$$

We now make some mild assumptions for the iterates and their corresponding gradients.

Assumption 1. *The distance between any Θ_t generated by the proposed approach is bounded. This means that $\|\Theta_m - \Theta_n\|_2 \leq D$.*

Assumption 2. *The gradients of function f_t are bounded. This means, $\|\nabla f_t(\Theta)\|_2 \leq G$.*

For the L-BFGS update, a step is acceptable if the condition $\mathbf{s}_t^\top \mathbf{y}_t > 0$ holds. We formally state this as a theorem below.

Theorem 4.5.1. *For a convex set $\mathcal{F}$ and sequence of convex functions $\mathcal{C} = \{c_1, c_2, \ldots\}$, and for some step $\mathbf{s}_t = \Theta_{t+1} - \Theta_t$, where $\Theta_t, \Theta_{t+1} \in \mathcal{F}$, and change in gradient $\mathbf{y}_t = \nabla c_t(\Theta_{t+1}) - \nabla c_t(\Theta_t)$, where $c_t \in \mathcal{C}$ computed using a symmetric positive definite L-BFGS approximation, $\mathbf{s}_t$ and $\mathbf{y}_t$ will always satisfy the curvature condition $\mathbf{s}_t^\top \mathbf{y}_t > 0$.*

In practice, the condition $\mathbf{s}_t^\top \mathbf{y}_t > 0$ (please refer Sec. 4.3) is enforced by requiring $\mathbf{s}_t^\top \mathbf{y}_t \geq \varepsilon$ for some small $\varepsilon > 0$. It follows from Theorem 4.5.1 that $\|\mathbf{s}_t\|, \|\mathbf{y}_t\| \neq 0$ and that there exists some $c_l \in \mathbb{R}$ such that $0 < c_l \leq \mathbf{y}_t^\top \mathbf{y}_t$.

Given $\alpha_t = \alpha/\sqrt{t}$, we state the following theorem:

Theorem 4.5.2. *Given Assumptions 1 and 2 hold, we get and upper bound on the*

regret as

$$
\begin{aligned}
R(T) \;\leq\; & \\
& \frac{D^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{n} \sqrt{T\hat{v}_T} + \\
& \frac{(\beta_1+2)\alpha G}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{n} \|\mathbf{g}_{1:T,i}\| + \\
& \frac{n}{(1-\beta_1)(1-\lambda)^2}\Bigg[\frac{D^2G}{2\alpha} + \frac{D^2G}{2} + \\
& \qquad\qquad \frac{2D^2G^5}{c_l^2} + \frac{4\alpha D^2G^5}{c_l^2}\Bigg].
\end{aligned}
$$

From Theorem 4.5.2, the corollary follows:

Corollary 4.5.3. *Quasi-Adam achieves the following guarantee:*

$$R(T) = O(\frac{1}{\sqrt{T}}).$$

Thus, $\lim_{T\to\infty} R(T) \to 0$.

In order to understand the convergence properties, we will use the framework laid out by [60]. We revisit the definition of regret for the readers convenience. We modify the parameters such that it matches the notations section.

Lemma 4.5.4. *If a function f:* $\mathbb{R}^n \to \mathbb{R}$ *is convex, then* $\forall \Theta, \Theta' \in \mathbb{R}^n$,

$$f(\Theta') \geq f(\Theta) + \nabla f(\Theta)^\top (\Theta' - \Theta)$$

We introduce the notation $\mathbf{g}_{t,i}$ as the i^{th} component of the vector $\mathbf{g}_t$. Thus, $\mathbf{g}_{1:t,i} \in \mathbb{R}^t$ is the i^{th} component of the gradient, which means $\mathbf{g}_{1:t,i} = [\mathbf{g}_{1,i}, \mathbf{g}_{2,i}, \cdots, \mathbf{g}_{t,i}]$. Similarly, we define the i^{th} dimension of Θ_t as $\Theta_{1:t,i} \in \mathbb{R}^t$, which means $\Theta_{1:t,i} = [\Theta_{1,i}, \Theta_{2,i}, \cdots, \Theta_{t,i}]$

Lemma 4.5.5. *Let* $\gamma = \frac{\beta_1^2}{\sqrt{\beta_2}}$. *For* $\beta_1, \beta_2 \in [0,1)$ *that satisfy* $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$ *and bounded* $\mathbf{g}_t, \|\mathbf{g}_t\|_\infty \leq \|\mathbf{g}_t\|_2 \leq G$, *the following inequality holds*

$$\sum_{t=1}^{T} \frac{\hat{\mathbf{m}}_{t,i}^2}{\sqrt{t\hat{v}_t}} \leq \frac{2G}{(1-\gamma)^2\sqrt{1-\beta_2}} \|\mathbf{g}_{1:T,i}\|_2.$$

The proof can be found in [28].

From the definition of L-BFGS quasi-Newton matrices, if $\mathbf{s}_t^\top \mathbf{y}_t > \epsilon$, the Hessian approximation $\mathbf{H}_t$ remains positive-definite. Thus, for the two-rank update in (4.7), given $0 < \lambda_1 \leq \lambda_2$ and $\gamma > 0$, the inequality $\gamma < \lambda_1 + \gamma \leq \lambda_2 + \gamma$, holds. This means $\frac{1}{\gamma} > \frac{1}{\lambda_1+\gamma} \geq \frac{1}{\lambda_2+\gamma}$. Thus, we can write the following bound:

Lemma 4.5.6. *Given c_t has bounded gradients, $\|\nabla c_t(\Theta)\|_\infty \leq \|\nabla c_t(\Theta)\|_2 \leq G$ for all t, $\Theta \in \mathbb{R}^d$ and distance between any Θ_t generated by quasi-Adam is bounded, $\|\Theta_n - \Theta_m\| \leq D$ for any $n, m \in \{1, \cdots, T\}$. The following inequality holds*

$$\left|\frac{1}{\gamma_t}\right| \leq \frac{2DG}{c_l}.$$

Proof. By construction, we know that $\mathbf{y}_t^\top \mathbf{y}_t \geq c_l$. This implies

$$\frac{1}{\mathbf{y}_t^\top \mathbf{y}_t} \leq \frac{1}{c_l}.$$

Thus it follows that

$$\begin{aligned}\left|\frac{1}{\gamma_t}\right| = \left|\frac{\mathbf{y}_t^\top \mathbf{s}_t}{\mathbf{y}_t^\top \mathbf{y}_t}\right| &\leq \frac{|\mathbf{y}_t^\top \mathbf{s}_t|}{c_l} \leq \frac{\|\mathbf{y}_t\|_2 \|\mathbf{s}_t\|_2}{c_l} \\ &\leq \frac{(\|\nabla c_t(\Theta_{t+1})\| + \|\nabla c_t(\Theta_t)\|)D}{c_l} \\ &\leq \frac{2GD}{c_l}.\end{aligned}$$

□

Lemma 4.5.7. *Let $\mathbf{H}_t$ be defined in (4.6). Under the assumptions of Lemmas 4.5.5 and 4.5.6, the following inequality holds:*

$$|(\mathbf{H}_t \mathbf{g}_t)_i| \leq \frac{2DG^2}{c_l}.$$

Proof. Using vector- and matrix-norm inequalities,

$$\begin{aligned}|(\mathbf{H}_t \mathbf{g}_t)_i| &\leq \max_i |(\mathbf{H}_t \mathbf{g}_t)_i| = \|\mathbf{H}_t \mathbf{g}_t\|_\infty \\ &\leq \|\mathbf{H}_t \mathbf{g}_t\|_2 \leq |\lambda_{\max}| \|\mathbf{g}_t\|_2 \leq \lambda_{\max} G\end{aligned}$$

(see [61]).

Since $\mathbf{B}_t$ is always positive-definite, for $\lambda \in \{\lambda_1, \lambda_2\}$, $\lambda + \gamma_t > \gamma_t$ which means $\frac{1}{\lambda+\gamma_t} < \frac{1}{\gamma_t}$. Thus, $|\lambda_{\max}| = \frac{1}{\gamma_t}$. By Lemma 4.5.6, we get

$$|(\mathbf{H}_t \mathbf{g}_t)_i| \leq \frac{G}{\gamma_t} \leq \frac{2G^2 D}{c_l}.$$

Theorem 4.5.8. *Given Assumptions 1 and 2 hold, the upper bound on the regret is given by*

$$\begin{aligned} R(T) \;\leq\; & \\ & \frac{D^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{n} \sqrt{T\hat{v}_T} + \\ & \frac{(\beta_1+2)\alpha G}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{n} \|\mathbf{g}_{1:T,i}\| + \\ & \frac{n}{(1-\beta_1)(1-\lambda)^2} \Bigg[\frac{D^2 G}{2\alpha} + \frac{D^2 G}{2} + \\ & \qquad \frac{2D^2 G^5}{c_l^2} + \frac{4\alpha D^2 G^5}{c_l^2} \Bigg]. \end{aligned}$$

Proof. Using Lemma 4.5.4, we have,

$$c_t(\Theta_t) - c_t(\Theta_t^*) \leq \mathbf{g}_t^T (\Theta_t - \Theta^*) = \sum_{i=1}^{n} \mathbf{g}_{t,i} (\Theta_{t,i} - \Theta_{,i}^*)$$

We rewrite the proposed update from (4.9),

$$\begin{aligned} \Theta_{t+1} &= \Theta_t - \alpha_t (\hat{\mathbf{m}}_t / \sqrt{\hat{v}_t} + \mathbf{H}_t \mathbf{g}_t) \\ &= \Theta_t - \alpha_t \left(\frac{\beta_{1,t}}{(1-\beta_1^t)} \frac{\mathbf{m}_{t-1}}{\sqrt{\hat{v}_t}} + \frac{1-\beta_{1,t}}{(1-\beta_1^t)} \frac{\mathbf{g}_t}{\sqrt{\hat{v}_t}} + \mathbf{H}_t \mathbf{g}_t \right). \end{aligned}$$

Thus, for each

$$\begin{aligned} (\Theta_{t+1,i} - \Theta_{,i}^*)^2 = (\Theta_{t,i} - \Theta_{,i}^*)^2 & \\ - 2\alpha_t (\Theta_{t,i} - \Theta_{,i}^*) \frac{\beta_{1,t} \mathbf{m}_{t-1}}{(1-\beta_1^t)\sqrt{\hat{v}_t}} & \\ - 2\alpha_t (\Theta_{t,i} - \Theta_{,i}^*) \frac{(1-\beta_{1,t}) \mathbf{g}_{t,i}}{(1-\beta_1^t)\sqrt{\hat{v}_t}} & \\ - 2\alpha_t (\Theta_{t,i} - \Theta_{,i}^*) (\mathbf{H}\mathbf{g}_t)_i & \\ + \alpha_t^2 \left(\frac{\hat{\mathbf{m}}_{t,i}}{\sqrt{\hat{v}_t}} \right)^2 + \alpha_t^2 \left(\mathbf{H}_t \mathbf{g}_t \right)_i^2 & \\ + 2\alpha_t^2 \frac{\hat{\mathbf{m}}_{t,i}}{\sqrt{\hat{v}_t}} \left(\mathbf{H}_t \mathbf{g}_t \right)_i . & \end{aligned}$$

Rearranging the above equation and using Young's inequality $ab \leq a^2/2 + b^2/2$, it can be shown that $\sqrt{\hat{v}_t} = \sqrt{\sum_{j=1}^{t}(1-\beta_2)\beta_2^{t-j}\mathbf{g}_{j,i}^2}/\sqrt{1-\beta_2^t} \leq \|\mathbf{g}_{1:t,i}\|_2 \leq G\sqrt{t}$, and $\beta_{1,t} \leq \beta_1$. Using these inequalities and Lemma 4.5.4, we get the following bound for $\mathbf{g}_i(\Theta_{t,i} - \Theta^*_{,i})$:

$$
\begin{aligned}
&\mathbf{g}_i(\Theta_{t,i} - \Theta^*_{,i}) \\
&\quad= \frac{(1-\beta_1^t)\sqrt{\hat{v}_t}}{(1-\beta_{1,t})2\alpha_t}\left((\Theta_{t,i} - \Theta^*_{,i})^2 - (\Theta_{t+1,i} - \Theta^*_{,i})^2\right) \\
&\quad\quad- \frac{\beta_{1,t}}{(1-\beta_{1,t})}\frac{\hat{v}_{t-1}^{1/4}}{\sqrt{\alpha_{t-1}}}(\Theta_{t,i} - \Theta^*_{,i})\sqrt{\alpha_{t-1}}\frac{\mathbf{m}_{t-1,i}}{\hat{v}_{t-1}^{1/4}} \\
&\quad\quad- \frac{(1-\beta_1^t)\sqrt{\hat{v}_t}}{1-\beta_{1,t}}(\Theta_{t,i} - \Theta^*_{,i})(\mathbf{H}_t\mathbf{g}_t)_i \\
&\quad\quad+ \frac{(1-\beta_1^t)}{1-\beta_{1,t}}\frac{\alpha_t}{2}\frac{\hat{\mathbf{m}}_{t,i}^2}{\sqrt{\hat{v}_t}} + \frac{(1-\beta_1^t)}{1-\beta_{1,t}}\frac{\alpha_t\sqrt{\hat{v}_t}}{2}\left(\mathbf{H}_t\mathbf{g}_t\right)_i^2 \\
&\quad\quad+ \alpha_t\frac{(1-\beta_1^t)}{1-\beta_{1,t}}\frac{\hat{\mathbf{m}}_{t,i}}{\sqrt{\hat{v}_t}}\left(\mathbf{H}_t\mathbf{g}_t\right)_i \\
&\quad\leq \frac{\sqrt{\hat{v}_t}}{(1-\beta_{1,t})2\alpha_t}\left((\Theta_{t,i} - \Theta^*_{,i})^2 - (\Theta_{t+1,i} - \Theta^*_{,i})^2\right) \\
&\quad\quad+ \frac{\beta_{1,t}}{2(1-\beta_{1,t})}\frac{\alpha_{t-1}\mathbf{m}_{t-1}^2}{\sqrt{\hat{v}_{t-1}}} \\
&\quad\quad+ \frac{\beta_{1,t}\sqrt{\hat{v}_{t-1}}}{2(1-\beta_{1,t})\alpha_{t-1}}(\Theta_{t,i} - \Theta^*_{,i})^2 \\
&\quad\quad+ \frac{(1-\beta_{1,t})\sqrt{\hat{v}_t}}{2(1-\beta_{1,t})}[(\Theta_{t,i} - \Theta^*_{,i})^2 + (\mathbf{H}_t\mathbf{g}_t)_i^2] \\
&\quad\quad+ \frac{(1-\beta_1^t)\alpha_t}{1-\beta_{1,t}}\frac{\hat{\mathbf{m}}_{t,i}^2}{\sqrt{\hat{v}_t}} + \frac{(1-\beta_1^t)\alpha_t\sqrt{\hat{v}_t}}{1-\beta_{1,t}}\left(\mathbf{H}_t\mathbf{g}_t\right)_i^2.
\end{aligned}
$$

Using Lemma 4.5.5 and Lemma 4.5.7, we obtain the following regret bound:

$$
\begin{aligned}
R(T) \leq & \frac{1}{2\alpha_1(1-\beta_1)} \sum_{i=1}^{n} (\Theta_{1,i} - \Theta^*_{,i})^2 \sqrt{\hat{v}_1} + \\
& \sum_{i=1}^{n} \sum_{t=2}^{T} \frac{1}{2(1-\beta_1)} (\Theta_{t,i} - \Theta^*_{,i})^2 \left(\frac{\sqrt{\hat{v}_t}}{\alpha_t} - \frac{\sqrt{\hat{v}_{t-1}}}{\alpha_{t-1}} \right) \\
& + \frac{\beta_1 \alpha G}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{n} \|\mathbf{g}_{1:T,i}\|_2 \\
& + \sum_{i=1}^{n} \sum_{t=1}^{T} \frac{\beta_{1,t} \sqrt{\hat{v}_t}}{2(1-\beta_{1,t})\alpha_t} (\Theta_{t,i} - \Theta^*_{,i})^2 \\
& + \sum_{i=1}^{n} \sum_{t=1}^{T} \frac{(1-\beta_{1,t})\sqrt{\hat{v}_t}}{2(1-\beta_{1,t})} (\Theta_{t,i} - \Theta^*_{,i})^2 \\
& + \frac{2D^2 G^4}{c_l^2} \sum_{i=1}^{n} \sum_{t=1}^{T} \frac{\sqrt{\hat{v}_t}}{1-\beta_{1,t}} \\
& + \frac{2\alpha G}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{n} \|\mathbf{g}_{1:T,i}\|_2 \\
& + \frac{4D^2 G^4}{c_l^2} \sum_{i=1}^{n} \sum_{t=1}^{T} \frac{\alpha_t \sqrt{\hat{v}_t}}{1-\beta_{1,t}}.
\end{aligned}
$$

Using Assumption 1 and definition $\alpha_t = \alpha/\sqrt{t}$, the bound on the regret is simplified to the following:

$$
\begin{aligned}
R(T) \leq & \frac{D^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{n} \sqrt{T\hat{v}_t} \\
& + \frac{(\beta_1+2)\alpha G}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{n} \|\mathbf{g}_{1:T,i}\|_2 \\
& + \overbrace{\frac{D^2}{2\alpha} \sum_{i=1}^{n} \sum_{t=1}^{T} \frac{\beta_{1,t}\sqrt{t\hat{v}_t}}{(1-\beta_{1,t})}}^{\text{First-term}} \\
& + \overbrace{\left(\frac{D^2}{2} + \frac{2D^2 G^4}{c_l^2} \right) \sum_{i=1}^{n} \sum_{t=1}^{T} \frac{\sqrt{\hat{v}_t}}{(1-\beta_{1,t})}}^{\text{Second-term}} \\
& + \overbrace{\frac{4\alpha D^2 G^4}{c_l^2} \sum_{i=1}^{n} \sum_{t=1}^{T} \frac{\sqrt{\hat{v}_t}}{\sqrt{t}(1-\beta_{1,t})}}^{\text{Third-term}}.
\end{aligned}
\tag{4.11}
$$

Using the upper bound for the geometric series $\sqrt{\hat{v}} \leq G\sqrt{t}$, we find the upper bound of the three terms in (4.11) separately. The bound for the first-term is given by

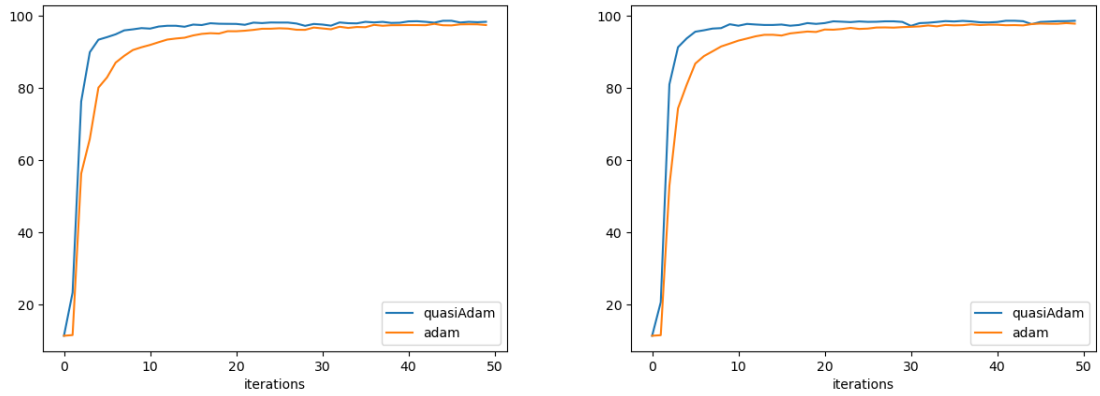
$$\begin{aligned}\sum_{t=1}^{T} \frac{\beta_{1,t}\sqrt{t\hat{v}_t}}{(1-\beta_{1,t})} &\leq G\sum_{t=1}^{T} \frac{t}{(1-\beta_{1,t})} \\ &\leq G\sum_{t=1}^{T} \frac{t\lambda^{t-1}}{(1-\beta_1)} \\ &\leq \frac{G}{(1-\beta_1)(1-\lambda)^2},\end{aligned}$$

the bound for the second-term is given by

$$\begin{aligned}\sum_{t=1}^{T} \frac{\sqrt{\hat{v}_t}}{(1-\beta_{1,t})} &\leq G\sum_{t=1}^{T} \frac{\sqrt{t}}{(1-\beta_{1,t})} \\ &\leq G\sum_{t=1}^{T} \frac{t}{(1-\beta_{1,t})} \\ &\leq \frac{G}{(1-\beta_1)(1-\lambda)^2},\end{aligned}$$

and the bound for the third-term is given by

$$\begin{aligned}\sum_{t=1}^{T} \frac{\sqrt{\hat{v}_t}}{\sqrt{t}(1-\beta_{1,t})} &\leq G\sum_{t=1}^{T} \frac{1}{1-\beta_{1,t}} \\ &\leq \frac{G}{1-\beta_1}\sum_{t=1}^{T} \lambda^{t-1} \\ &\leq \frac{G}{1-\beta_1}\sum_{t=1}^{T} \lambda^{t-1}t \\ &\leq \frac{G}{(1-\beta_1)(1-\lambda)^2}.\end{aligned}$$



(a) Batch-size = 256 (b) Batch-size = 512

Figure 4.1: **Experiment I:** MNIST image classification results for Adam and the proposed method, quasi-Adam. (a) Testing accuracy for batch-size of 256. (b) Testing accuracy for batch-size of 512. The y-axis represents the classification accuracy and the x-axis represents the batch-iteration. Note that for both batch-sizes, quasi-Adam outperforms Adam.

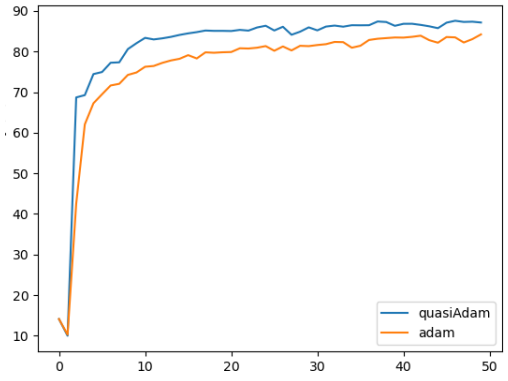
Thus, the final expression for the regret bound is given by

$$\begin{aligned} R(T) \quad \leq \\ & \frac{D^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{n} \sqrt{T\hat{v}_T} + \\ & \frac{(\beta_1+2)\alpha G}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{n} \|\mathbf{g}_{1:T,i}\| + \\ & \frac{n}{(1-\beta_1)(1-\lambda)^2} \left[\frac{D^2 G}{2\alpha} + \frac{D^2 G}{2} + \right. \\ & \qquad \left. \frac{2D^2 G^5}{c_l^2} + \frac{4\alpha D^2 G^5}{c_l^2} \right]. \end{aligned}$$

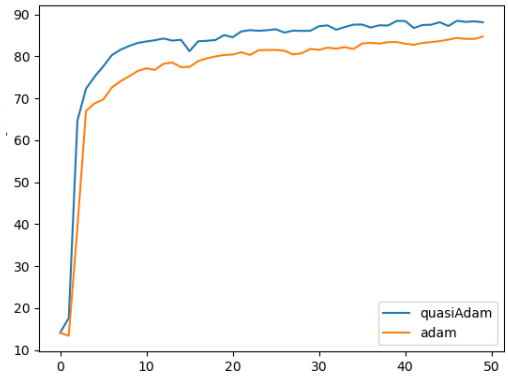
□

4.6 Experimental Setup

We evaluate the proposed approach over two datasets - CIFAR10 ([46]) and MNIST. We compare the proposed approach against Adam and SGD. We use these datsets due their reproducibiliy and ease of usage. We use a 2 convolutional

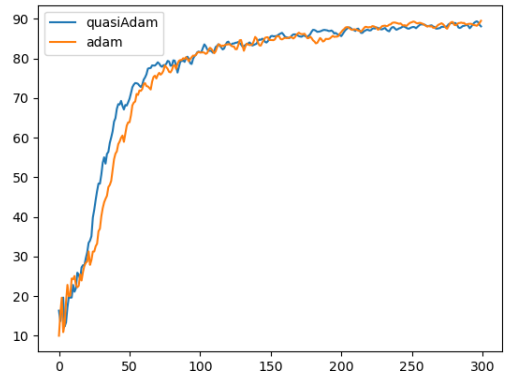


(a) Batch-size = 256

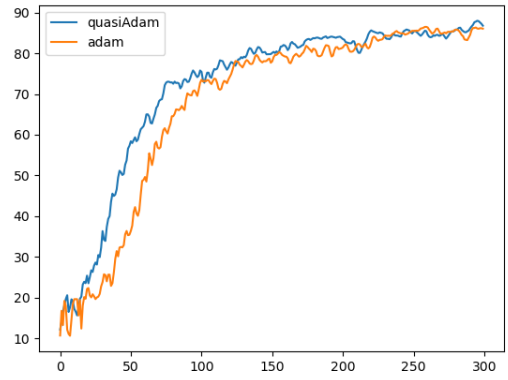


(b) Batch-size = 512

Figure 4.2: **Experiment II:** Fashion-MNIST image classification results for Adam and the proposed method, quasi-Adam. (a) Testing accuracy for batch-size of 256. (b) Testing accuracy for batch-size of 512. The y-axis represents the classification accuracy and the x-axis represents the batch-iteration. Note that for both batch-sizes, quasi-Adam outperforms Adam.



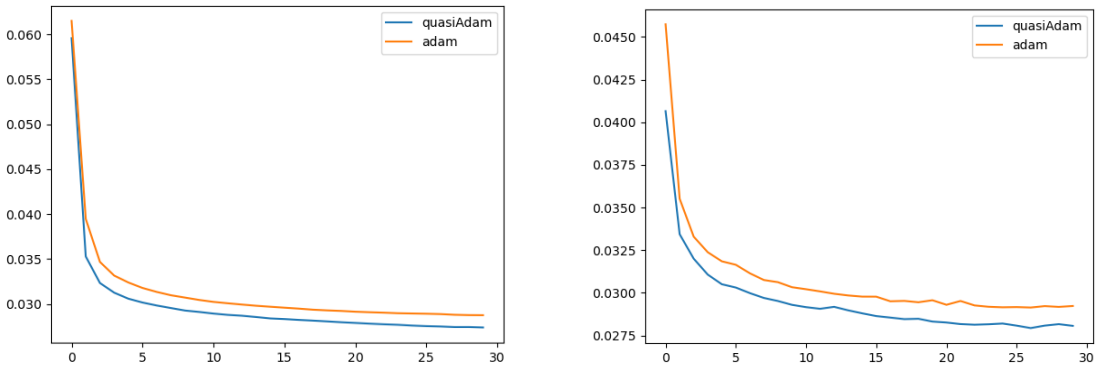
(a) Batch-size = 256



(b) Batch-size = 128

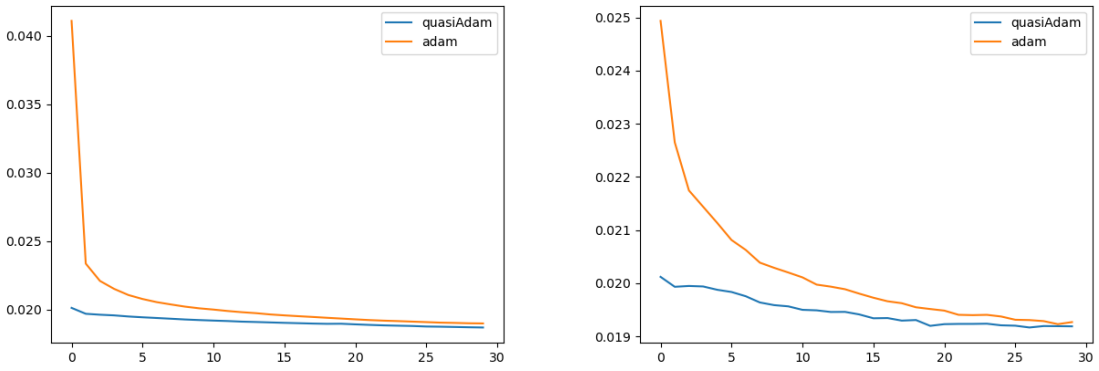
Figure 4.3: **Experiment III:** SVHN image classification results for Adam and the proposed method, quasi-Adam. (a) Testing accuracy for batch-size of 256. (b) Testing accuracy for batch-size of 128. The y-axis represents the classification accuracy and the x-axis represents the batch-iteration. Note that for both batch-sizes, quasi-Adam outperforms Adam.

networks (one shallow and one deep) to train over CIFAR10 and MNIST. We describe these datasets in Sec. 4.6.3. We use a small shallow network for the MNIST



(a) Training loss (b) Testing loss

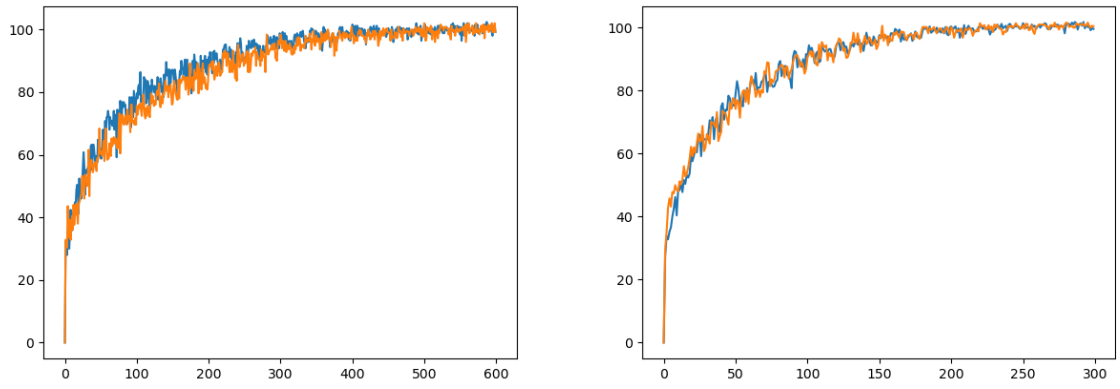
Figure 4.4: **Experiment IV:** Autoencoder MNIST Reconstruction for Adam and the proposed method, quasi-Adam. (a) Training loss. (b) Testing response. The x-axis represents the number of epochs and y-axis represents the average mean-squared error loss for each epoch. Note that in both training and testing responses, quasi-Adam outperforms Adam.



(a) Training loss (b) Testing loss

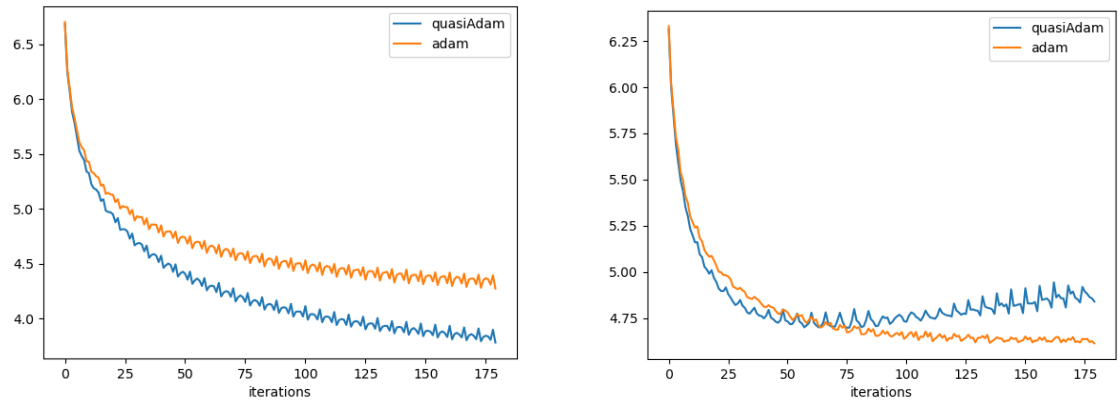
Figure 4.5: **Experiment V:** Autoencoder FMNIST Reconstruction results for Adam and the proposed method, quasi-Adam. (a) The training loss. (b) Testing response. The x-axis represents the number of epochs and the y-axis represents the average mean-squared error loss for each epoch. Note that in both training and testing responses, quasi-Adam outperforms Adam.

and FMNIST classification. Similarly, we use a small shallow network for the image reconstruction . We use two popular models: ResNet34 and DenseNet121, which



(a) batch-size = 256 (b) batch-size = 512

Figure 4.6: **Experiment VI**: CIFAR10 classification. (a) Batch-size = 256 images. (b) Batch-size = 512. The x-axis represents the number of iterations (batches) and the y-axis represents the average mean-squared error loss after each iteration. The proposed approach is able to outperform Adam with a batch-size of 256 images and has a comparable performance with a batch-size of 512 images.



(a) Training loss (b) Testing loss

Figure 4.7: **Experiment VII:** Pentree-Bank text prediction. (a) Training loss (b) the Testing loss. The x-axis-represents the number of iterations for each batch of sentences. The y-axis represents the loss for each batch of sentences. We note that the proposed approach is able to find a model with a lower training response than Adam. However, in the presence of overparametrization, overfitting can happen, which is evident in this case (see (b)).

are described in further detail in Sec. 4.6.2.

4.6.1 Testbed

All the experimenst were conducted using PyTorch (see [41]) libraries using two NVIDIA 1080 Ti graphics cards over an Intel i7-7700K CPU. For Adam, we conducted a variety of experiments to choose the hyperparameter based on the applications. The experiments included different batch size $\{256, 512, 1,024, 2,048, 8,192\}$, with different learning rates $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0.9\}$. We present results with the best hyperparameters for Adam and the proposed approach for each experiment. To understand the performance change between the proposed approach and Adam, we present the results where this change is most prominent. This is reflected in the first half of the training response. We observe that both the methods eventually converge to the same result after a large number of epochs.

4.6.2 Models

We use three different types of networks - an MNIST classifier for classifying MNIST and Fashion-MNIST dataset, ResNet34 to classifiy images in the SVHN dataset and an Autoencoder for MNIST and Fashion-MNIST reconstruction.

MNIST classifier: We design a shallow neural network to classify the MNIST dataset. The model has two convolutional layers, and three fully-connected dense layers. Each convolutional layer is followed by a maxpooling layer and ReLU activation function. Fully connected layers are followed by a ReLU activation function.

Resnet34: Resnet34 ([62]) is a deep learning model with 34 blocks, which contain two convolutional layers with skip connections between blocks and ReLU activation layers in between. The network contains approximately 21.8 million parameters. The network was designed to train over the ImageNet dataset. Since we are using SVHN here, the network has been modified accordingly.

Autoendcoder: we use a convolutional neural network. The network is comprised of two submodules - an encoder and a decoder. The encoder has a shallow 3

layer convolutional architecture and the decoder has a shallow 3 convolutional tranpose layer. The main purpose of the network is to reconstruct the images from its original image. The images are fed to the encoder, which compresses the image. This is then inflated/expanded by the decoder. The network has 87,125 parameters. We use the same network architecture for FMNIST reconstruction.

DenseNet121: The DenseNet121 network contains *densely* connected blocks followed by a transition layer. For more details on this network, see [63]. The network has 7,978,856 parameters and has been modified to operate on CIFAR10 dataset.

Language LSTM model: The language model contains an encoder, a decoder and an LSTM network. The embedding layer converts the input word tokens to embeddings. The LSTM contains 2 layers with 650 hidden units each. The network 19,780,400 parameters.

4.6.3 Datasets

We use three different datasets for two types of tasks - MNIST, FMNIST and SVHN. We provide a very brief description of these datasets below.

MNIST: MNIST is a dataset of 28×28 pixel handwritten digits from (0-9). These images are greyscaled single channel images with 5,000 training examples, 1,000 validation examples and 1,000 testing examples per class.

Fashion-MNIST: The Fashion-MNIST dataset (see [64]) consists of 28×28 pixel black-and-white images of clothing items such as shirt, automobile, shoes etc. The dataset has 10 classes and 6,000 images per class. This is further parted into 4,000 training examples, 2000 testing examples and 1,000 testing examples per class.

Street View House Number: The Street View House Number (SVHN) (see [65]) is a dataset with of street view images (numbers in addresses) extracted from Google Street View images. The dataset contains 32×32 RGB images of these street view images, cropped and separated into individual numbers ranging from (0-9). It can be viewed as the same flavor as MNIST, but significantly more relevant to mordern machine learning problems (recognizing digits and numbers in natural scene images). The training set contains 73,257 images while the testing

set contains 26,032 images.

CIFAR10: The CIFAR-10 dataset (Canadian Institute for Advanced Research, 10 classes) is a subset of the Tiny Images dataset and consists of 60,000 32×32 color images. The images are labelled with one of 10 mutually exclusive classes: airplane, automobile (but not truck or pickup truck), bird, cat, deer, dog, frog, horse, ship, and truck. There are 6000 images per class with 5,000 training and 1,000 testing images per class.

Penn Treebank: The English Penn Treebank (see [66]) corpus, and in particular the section of the corpus corresponding to the articles of Wall Street Journal (WSJ), is the most common corpus used for evaluation of models for sequence labelling. The task consists of annotating each word with its Part-of-Speech tag. The corpus is split into sections for training, validataion and testing - sections from 0 to 18 are used for training (38,219 sentences, 912,344 tokens), sections from 19 to 21 are used for validation (5,527 sentences, 131,768 tokens), and sections from 22 to 24 are used for testing (5,462 sentences, 129,654 tokens). The corpus is also commonly used for character-level and word-level Language Modelling.

4.6.4 Experiments

Experiment I: MNIST image classification. We use the MNIST classifier to classify the MNIST dataset and use cross-entropy as the loss function. We train the network with a batch-size of 256 and 512 images and a cross-entropy loss function. We use a learning rate of 1×10^{-2} for Adam and quasi-Adam. We present the testing response after each batch-iteration in Fig. 4.1.

Experiment II: Fashion-MNIST image classification. We use FashionMNIST dataset with the MNIST classifier. This is possible since the dimensions of the images are the same. We use a learning rate of 1×10^{-2} for both Adam and quasi-Adam, and a cross-entropy loss function. We present the testing accuracy for each batch-training iteration with a size of 256 and 512 images in Fig. 4.2.

Experiment III: SVHN image classification. For this task, we use the ResNet34 neural network. We train with a batch size of 256 and 128 images. We use the negative log-likelihood loss function with a learning rate of 1×10^{-2} for

both Adam and quasi-Adam. The results presented in Fig. 4.3 show the testing accuracy after each batch-training iteration.

Experiment IV: MNIST image reconstruction. For this task, we use the autoencoder to reconstruct the images. We use a batch-size of 256 images and the Mean-Squared Error (MSE) loss function with a learning rate of 10^{-2} for Adam and quasi-Adam. We observe both the training loss and the testing loss for each of the tasks. The results in Fig. 4.4 show the testing accuracy after each batch-iteration.

Experiment V: FMNIST image reconstruction. For the autoencoder reconstruction experiment, we use the autoencoder from Experiment IV (the images are of the same dimensions). We use the same learning rate of 1×10^{-3} for Adam and the proposed approach and the MSE loss function. We present the training and the testing response in Fig. 4.5.

Experiment VI: CIFAR10 image classification. We classify the images in CIFAR10 dataset (see [46]) with a DenseNet121 network (see [63]). We train a network with a batch-size of 256 images and 512 images. We report the testing accuracy in Fig. 4.6 (a). As can be seen from the testing reponses for both the batch sizes, the proposed approach is able to improve upon Adam.

Experiment VII: Language modelling. We use the Language LSTM model for the language modelling experiment. We run the experiment for 30 epochs. The results are presented in Fig. 4.7. From the training response depicted in Fig. 4.7 (a), it can be seen that the proposed approach is able to outperform Adam. However, the experiment reflects some form of overfitting under the effect of the proposed algorithm, which can cause the testing loss to diverge from the training response.

4.7 Discussion

The results from Sec. 4.6 elucidates the performance improvement of quasi-Adam over Adam. This can be attributed to both participating approaches - Adam and L-BFGS. The Adam approach uses an exponential moving momentum $\mathbf{m}_t$ and a exponential moving moment v_t. This allows for the learning rate to work

better at providing an adequate descent in the loss function in addition to treating the non-monotone behaviour of the loss function. However, it only exploits the gradient information, which is still a steepest descent direction. This may cause the iterates to dampen in a *plateau* region of the manifold, commonly referred to as saddle points. The L-BFGS approach, on the other hand, is able to provide a descent direction which exploits the curvature information using the secant equation. Since the Hessian information is induced in this matrix $\mathbf{H}_t$ in (4.8), the iterate potentially escapes this saddle point and addresses the sadle point problem more effectively. However, computing a large memory Hessian approximation can be computationally expensive.

The proposed approach was able to tackle both the problems - escpaing saddle points and containing computational expense. This combination of methods yielded an improvement in training performance across a variety of applications. Thus, convergence can be *expedited* with a very small computational overhead.

4.8 Conclusion

In this chapter, we proposed a new algorithm which uses a quasi-Newton update in conjunction with a moment estimation update. We show that the curvature information from the quasi-Newton approach improve upon an exponential moving average method. Through thorough experimentation, we were able to show that the the proposed approach was able to outperform Adam. We provided concrete convergence proofs and discuss the complexity analysis for space and time. We found quasi-Adam to be robust and suited across a variety of applications in the field of machine learning.

Chapter 5

Applications in deep learning

In this chapter, we will be exploring different applications of deep learning and suggest novel models for improving upon current architectures. In Sec. 5.1, we will explore Image disambigution. In Sec. 5.2, we will explore adversarial attacks and defense strategies against them. In Sec. 5.3, we discuss a Mult-stage denoising approach for Gaussian noisy images. In Sec. 5.4, we discuss a Multi-Stage mixed noise image denoising strategy.

5.1 Image Disambiguation

Signal recovery often involves separating and realizing multiple superimposed signals at once. Separating multiple images that have been superimposed is a challenging signal recovery problem. This situation arises when a detector, such as a microphone, receives multiple signals simultaneously. In order to recover the original signals, a signal separator needs to be applied. In this section, we will explore machine learning techniques for separating such signals. In particular, we investigate two approaches: an autoencoder approach and a transformer-based approach, and test their accuracy in recovering two separate images from noisy low-resolution superimposed measurements.

Image separation is a common signal separation problem in the domain of signal processing. Commonly referred to as 'blind source separation' (BSS), the problem involves separation of source signals with very little information about the sources or the multiplexing operation [67, 68].

Much of the early literature focuses on separation of temporal signals such as audio [69, 70] or video [71]. However, BSS has gained momentum in the field on images and tensors, which may have no temporal component whatsoever (see [72]). There is also literature which uses deep learning for blind source separation [73]. In contrast, for practical applications in digital imaging, noises can be caused by sudden change in light intensity, increase in temperature of the imaging apparatus or electrical fluctuations during transmission of the signal. Typically this type of noise is modeled as additive white Gaussian noise (AWGN). In the event that the imaging apparatus records the images with low resolution, the images may be

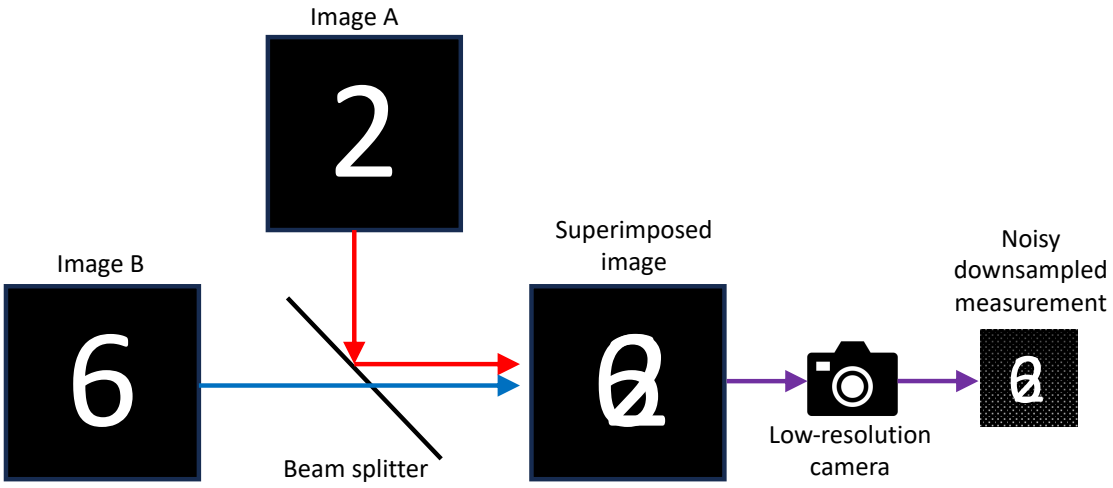


Figure 5.1: Schematic of the imaging system. Two images (A and B) are superimposed using a beam splitter observed at the detector of a low-resolution camera, resulting in a downsampled measurement with additive white Gaussian noise.

compressed as well. Thus, in addition to BSS, the image noise and compression need to be tackled.

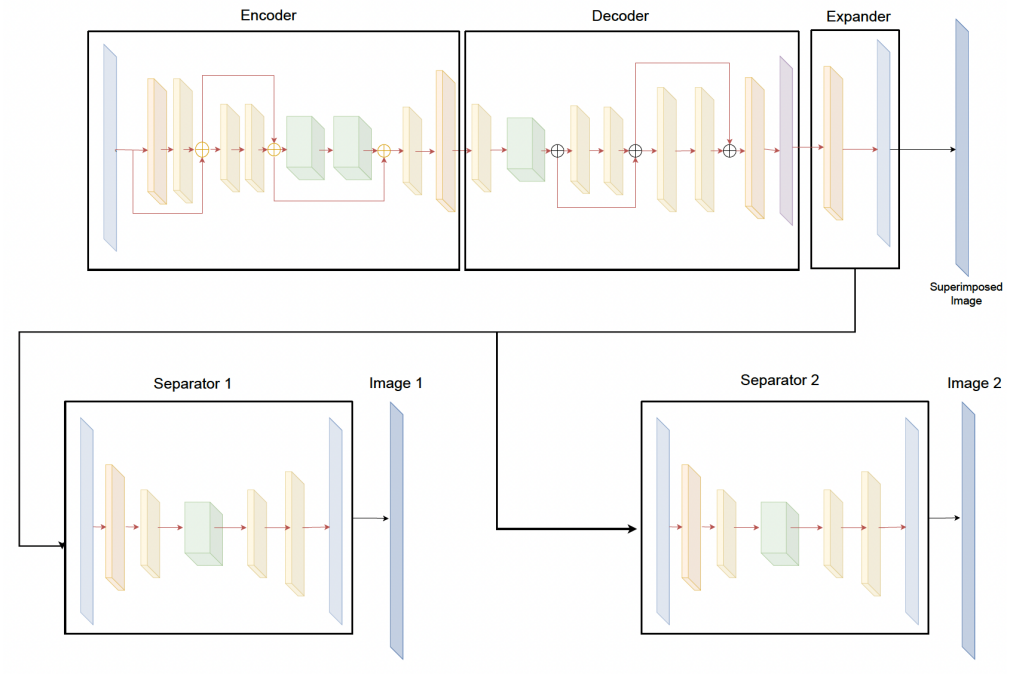
In this section, we will explore two deep learning strategies to address all of these issues simultaneously. The secion is organized as follows: In Sec. 5.1.1, we discuss the blind source problem formulation, in Sec. 5.1.2, we discuss the proposed approaches for separting the signals, in Sec. 5.1.3, we describe the numerical experiments of the proposed apporaches and in Sec. 5.1.4 and Sec. 5.1.5, we discuss the results and conclude respectively.

5.1.1 Problem Statement

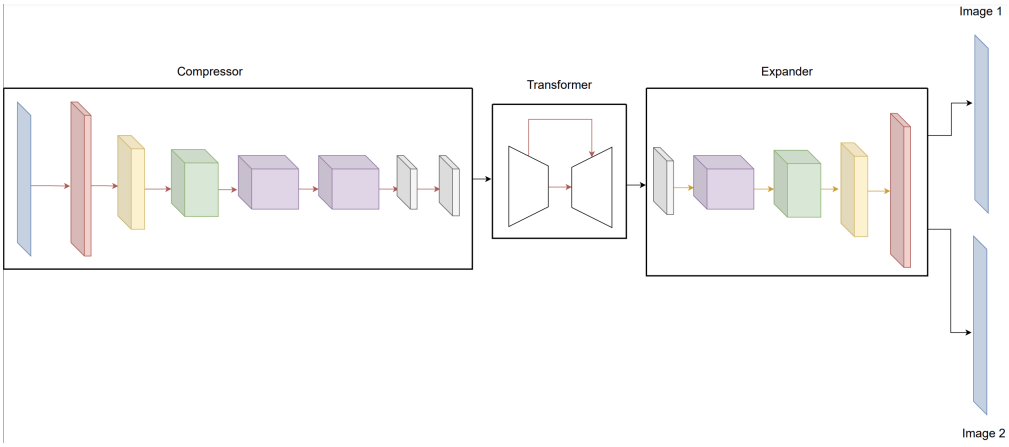
The blind source multiplexed problem can be formulated as

$$\mathbf{y} = \mathbf{D}(\mathbf{z}) + \mathbf{g}, \tag{5.1}$$

where $\mathbf{D}(\mathbf{z})$ is the downsampling operator, $\mathbf{z}$ is the resulting image of superimposing two images $\mathbf{x}^{(1)}$ an $\mathbf{x}^{(2)}$, i.e., $\mathbf{z} = \mathbf{x}^{(1)} + \mathbf{x}^{(2)}$, and $\mathbf{g} \sim \mathcal{N}(0, \sigma^2)$ is AWGN with zero mean and variance σ^2. These operations describe the linear model of observing noisy low-resolution images that are superimposed at the detector stage (see Fig. 5.1).



(a) Convolutional Separator (ConvSep) model.



(b) Limited informed Generative Transformer (LiGT) model.

Figure 5.2: The above diagram shows the two proposed approach. (a) The ConvSep model, which contains an encoder, a decoder, an expander, separator 1 ($\mathbf{S}_1$) and separator 2 ($\mathbf{S}_2$). (b) The model architecture of the transformer-based LiGT model, which comprises a compressor, a transformer and an expander. Each colored box represents the output from a convolutional operator

Related Work: In [74], the authors use a stacked autoencoder with fully connected layers. However, the size of the network can get prohibitively expensive for larger images. Also, the authors focus on weighted multiplexing problem without denoising and downsampling. There has been some work done using a convolutional neural network (see [75, 76]) for denoising images. They use two approaches to denoise the images - an autoencoder with convolutional layers and a recurrent neural network with convolutional layers as hidden units. This was able to tackle the problem of reducing the footprint of the network by replacing the linear layers with convolutional layers. In addition, the authors were able to realise the noise in the images as a temporal component. Our first method (ConvSep) is based on this approach.

Attention-based transformers have gained much momentum in the last few years. The concept of transformers was introduced by Vaswani et. al (see [77]) for natural language processing (NLP) [78, 79]. With increased improvement in NLP applications, the use of transformers was pervasive in many different fields such as image denoising [80, 81], protein structure prediction [82], and sentiment analysis [83]. Our second method (LiGT) is based on this approach.

5.1.2 Proposed Approach

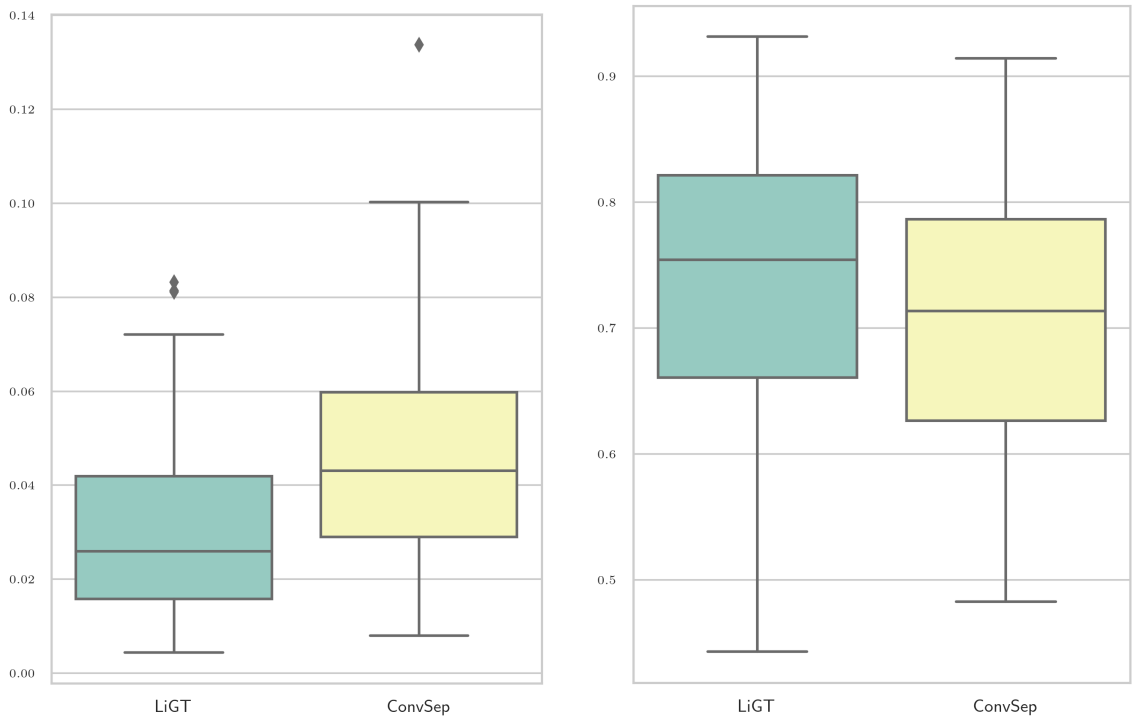
In this section, we describe the two approaches and their respective datasets.

Loss function: We optimize the network parameters using HuberLoss, which is defined as

$$\mathcal{L}_\delta(\hat{\mathbf{x}}, \mathbf{x}) = \begin{cases} \frac{1}{2}\|\hat{\mathbf{x}} - \mathbf{x}\|_1^2 & \text{if } \|\hat{\mathbf{x}} - \mathbf{x}\|_1 < \delta, \\ \delta(\|\hat{\mathbf{x}} - \mathbf{x}\|_1 - \frac{1}{2}\delta) & \text{otherwise,} \end{cases} \tag{5.2}$$

where $\delta \in \mathbb{R}$ helps in smoothing out the loss function at prohibitively small values of the absolute difference, $\hat{\mathbf{x}} \in \mathbb{R}^{n\times n}$ is the reconstructed realization and $\mathbf{x} \in \mathbb{R}^{n\times n}$ is the true image. We choose a value of $\delta = 1$ for our experiments.

Method I (ConvSep): ConvSep is a convolutional network which denoises, expands and separates the compressed noisy realization back to its original parent observations. In this approach, we take the compressed, noisy realization, and



(a). (b).

Figure 5.3: The above figure shows the collective results for the proposed approaches. Fig. (a). shows the distribution of Mean-Squared error and Fig. (b). shows the distribution of Structural similarity index metric (SSIM) values.

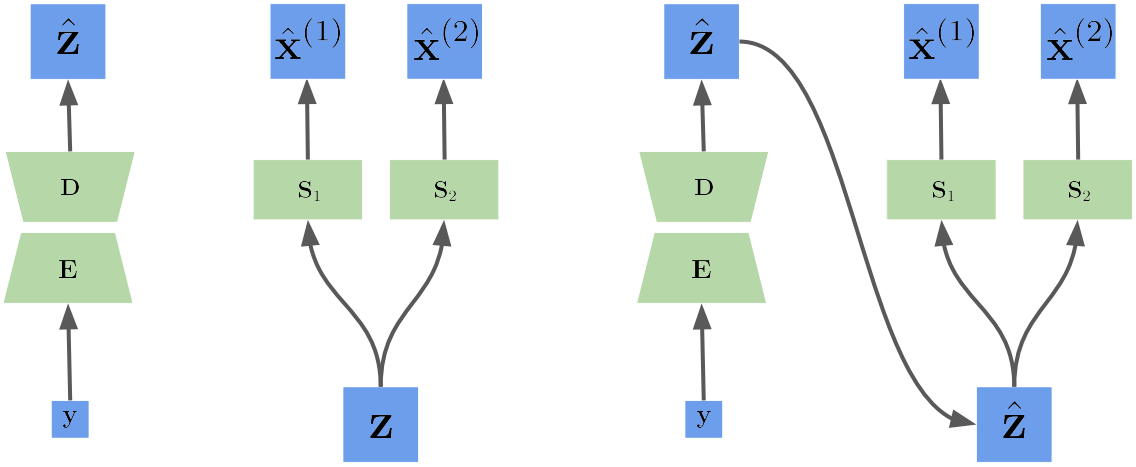
expand the dimensions using an Encoder (**E**)-Decoder (**D**) operation. Then we use two separators - $\mathbf{S}_1$ and $\mathbf{S}_2$ - that separate the image into two their two parent images. For more information on the networks, please see Fig. 5.2.

During the training operation, the superimposed image is available to perform the image expansion step. For more details on how this model operates, please refer Fig. 5.4. The loss function for ConvSep is given by

$$\mathcal{L}_{\text{ConvSep}} = \mathcal{L}(\hat{\mathbf{z}}, \mathbf{z}) + \mathcal{L}(\hat{\mathbf{x}}_1, \mathbf{x}_1) + \mathcal{L}(\hat{\mathbf{x}}_2, \mathbf{x}_2), \tag{5.3}$$

where $\mathcal{L}(\mathbf{z}, \hat{\mathbf{z}})$ is the loss between the denoised and upsampled superimposed ground truth and reconstructed images, $\mathcal{L}_1(\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_1)$ is the loss between the reconstructed image $\hat{\mathbf{x}}$ and the ground truth image $\mathbf{x}_1$. This model has a total of 2,846,224 parameters.

Method II (LiGT): LiGT is a transformer based model which expands, cleans



(a) ConvSep training procedure (b) ConvSep testing procedure

Figure 5.4: This is training and testing procedure for the ConvSep model. (a) During training, the superimposed and clean images are available to the ConvSep network to perform the separating operation. (b) During this stage, the the superimposed, clean images are not available to the network. Instead the output from the decoder is fed to the separators $\mathbf{S}_1$ and $\mathbf{S}_2$

and separates the mutliplexed signal at once. Unlike Method I, this model does not have access to the upsampled denoised superimposed realization. For more details on how the model works, please refer Fig. 5.2. The loss function for LiGT is given by

$$\mathcal{L}_{\text{LiGT}} = \mathcal{L}(\hat{\mathbf{x}}_1, \mathbf{x}_1) + \mathcal{L}(\hat{\mathbf{x}}_2, \mathbf{x}_2). \tag{5.4}$$

This model has a total of 2,301,865 parameters.

Dataset: We use the MNIST dataset [24] in our experiments. To generate our data, we randomly choose two images $\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)} \in \mathbb{R}^{28\times 28}$ from the MNIST data and superimpose them to obtain $\mathbf{z}_i \in \mathbb{R}^{28\times 28}$, which is then downsampled by a factor of 2 and to which AWGN is added to yield the noisy low-dimensional superimposed images $\mathbf{y}_i \in \mathbb{R}^{14\times 14}$. For Method I, the dataset is given by $\mathcal{D}_1 = \{\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}, \mathbf{z}_i, \mathbf{y}_i\}_{i=1}^N$. For Method II, the noisy, downsampled and superimposed observation is directly mapped to the two clean realizations and the intermediary data $\mathbf{z}_i$ is not used. The dataset is thus given by $\mathcal{D}_2 = \{\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}, \mathbf{y}_i\}_{i=1}^N$.

5.1.3 Experiments

In this section we describe the testbed and the training procedures for bot the models. All the architectures were implemented using PyTorch [84]. Training and testing were performed using two NVIDIA 1080 Ti GPUs. The networks were trained using the Adam optimizer [28].

Training: During training, the superimposed, compressed and noisy images are fed to both the models. For the LiGT model, these images are directly mapped to the clean and separated images. For the ConvSep model, the downsampled and noisy superimposed observation is fed to the encoder $\mathbf{E}$ for the $\mathbf{D}$ to yield the clean, upsampled superimposed construction $\hat{\mathbf{z}}$. The clean, upsampled superimposed image $\mathbf{z}$ is then fed to the separators $\mathbf{S}_1$ and $\mathbf{S}_2$ to yield $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$.

Testing: During testing, the operation of LiGT matches the training procedure. However, for the ConvSep model, the upsampled superimposed construction $\hat{\mathbf{z}}$ is directly fed to separators $\mathbf{S}_1$ and $\mathbf{S}_2$ (see Fig. 5.4 for illustration).

5.1.4 Results

In this section, we present the results from the two approaches. Fig. 5.5 shows the results from both the approaches. The first row shows the superimposed images, downsampled with added noise. The second and third row show the separated images using the LiGT model, the fourth and fifth row show the images separated using the ConvSep model. The last two images show the ground truth images. We can notice that the MSE for the LiGT model is much lower than the ConvSep model. It can also be noticed that the LiGT model was able to keep a lot of the structural integrity of the image better than the ConvSep model. Fig. 5.3 presents the overall MSE and SSIM results. The average MSE loss for the ConvSep model was 4.66×10^{-2}, and the average MSE loss for the LiGT model was 3.00×10^{-2}

5.1.5 Conclusion

In this section, we presented two approaches for image disambiguation. In the first approach, we presented ConvSep, which uses an RNN inspired convolutional

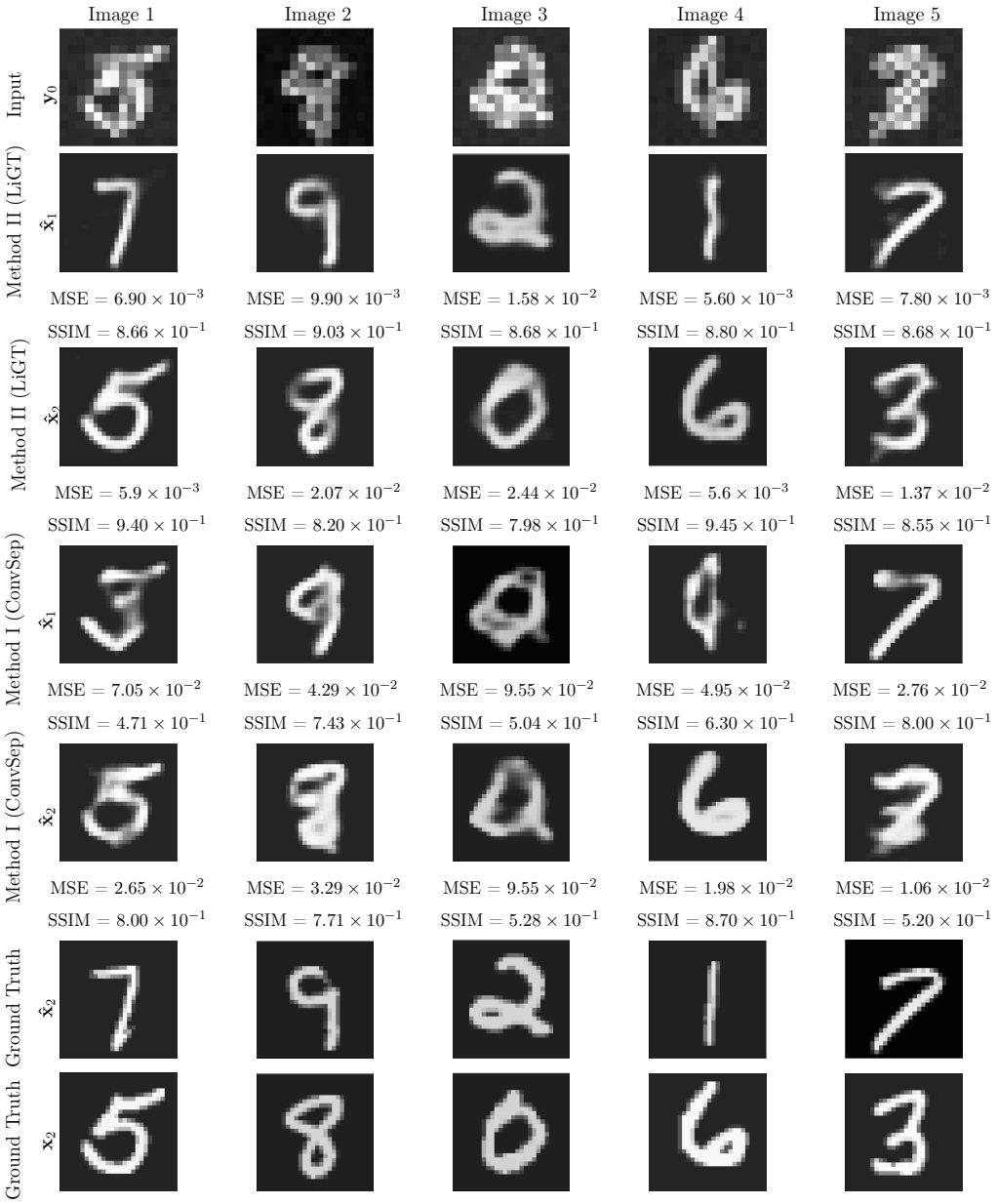


Figure 5.5: Numerical experiments on 5 images from the MNIST dataset. Row 1: Noisy input images $\mathbf{y}$. Rows 2 and 3: Final reconstructions $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_1$ using Method II (LiGT). Rows 4 and 5: Final reconstructions $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_1$ using Method I (ConvSep). Rows 6 and 7: Ground truth images $\mathbf{x}_1$, $\mathbf{x}_1$. MSE values for both Methods I (ConvSep) and II (LiGT) are presented for each image.

neural network to denoise and upsample in one stage and disambiguate the images in another stage. In the second approach, we presented LiGT, a Transformer based model which denoised, upsampled and cleaned the image simultaneously. Experi-

ments and results show that the transformer based model was able to outperform the RNN inspired approach with a smaller model footprint.

5.2 Novel defense techniques for White-Box Adversarial Attacks

Szegedy et. al [85] demonstrated that small perturbations to an image can fool deep neural networks into misclassifying the image. These perturbations, small enough that the perturbed image is indistinguishable from the originals to the human eye are often called "adversarial perturbations", or adversarial attacks.

Generally, adversarial attacks are categorized as either white-box or black-box attacks. In a black-box attack [86, 87], the attacking model has no knowledge of the predicting model. However, the output probabilities of a sample belonging to different classes may be available to the attacking model. Some examples of black-box attacks include gradient estimation, transferability, local search and combinatorics.
In a white-box attack [88, 89], the attacking model has knowledge of the weights and the neural network. Most white-box adversarial attacks are based on the Fast Gradient Sign Method (FGSM) [90], which adds to the input a (very small) vector in the direction of the gradient of the loss function with respect to the data. In FGSM's multistep variant, Projected Gradient Descent (PGD), the noise is added by taking multiple steps in the direction of the gradients and is often target at different classes in the dataset.

In this section, we explore adversarial defense mechanisms again white-box attacks, mainly PGD. For the reader's convenience, we have included a taxonomy of terms here.

5.2.1 Related Works

Defense against adversarial attacks has been an active area of research since the original paper [90].

Adversarial training: In 2015, Goodfellow et. al [90] proposed an adversarial training solution that attempts to solve a min-max optimization problem by training adversarial examples until the model learns to classify them. Given a dataset $\mathcal{X}$, with its corresponding labels y, some parameterization for the neural network Θ and some radius vector δ, the optimization setup can be written as

$$\Theta* = \arg\min_{\Theta} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in\mathcal{X}} \left[\max_{\delta\in[-\epsilon,\epsilon]} \mathcal{L}(\mathbf{x}+\delta;\mathbf{y};F_{\Theta}) \right].$$

Gao et. al [91] proved the convergence of these methods. However, the assumption here is that we know the noise level ϵ. A variety of related methods have been proposed since then ([92–96]). However, in a practical scenario, this information is not always available. In addition, depending on the size of the network and the size of the image, this method can get computationally expensive and time consuming.

Randomization: Another common method to defend a deep network against attacks is to randomize the input to the neural network [97]. This method uses two randomization layers in the neural network: a. The first layer randomly resizes the input image using a random scaling layer. b. The second layer adds a padding layer which is assigned around the boundary of the image at random. Zantedeschi et. al [93] showed that, by using a modified ReLU (called BReLU), and adding noise to the origin input to augment the training data, the learned model will gain some stability to adversarial examples. However, Carlini and Wagner [98] found that these defense mechanisms do not hold against strong attacks such as FGSM and PGD.

Projection: In this approach, a generative model is trained to classify on clean/-natural examples. The adversarial sample is fed to the generative model, an autoencoder which is trained on natural samples, before it is fed to the classifier. The idea here is that the network will "project" the adversarial sample to a point closely to the projection of the corresponding original image.

An extension to the above method is Generative Adversarial Networks (GANs) which aims at learning the distribution of the clean training data and attempts to project the adversarial sample onto this clean learned manifold [90]. The optimization setup, which is very similar to the adversarial training approach, is given

by

$$\min_{\mathbf{G}} \max_{\mathbf{D}} \{\mathbb{E}_{x\sim p_{\text{data}}(\mathbf{x})}[\log(\mathbf{D})(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log(1-\mathbf{D}(\mathbf{G}(\mathbf{z})))\right]\},$$

where $\mathbf{G}$ is the generator, $\mathbf{D}$ is the discriminator which predicts if $\mathbf{x}$ came from training data or $\mathbf{G}$, p_{data} is the distribution of the clean data, $\mathbf{z}$ is the Numerous adversarial defense schemes have been proposed using the above optimization setup (see [99, 100]). However, the drawback here is the similar to the adversarial training method—it may take a lot of time to train the GAN itself. In addition, GANs are much harder to train owing to their continual learning nature [101].

Detection: This approach only involves identifying an adversarial sample. A small binary classifier is built to separate the adversarial examples from the clean data [102]. However, this method suffers from a generalization limitation. Another way of detection commonly used is **KD-detection** [103], where a kernel $\mathbf{K}$ predicts whether a sample data $\mathbf{x}_i$ belongs to the same distribution as the dataset $\mathcal{X}$. The probability is given by

$$\hat{p}_{\mathcal{X}}\mathbf{x} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{K}_{\sigma}(\mathbf{x}, \mathbf{x}_i),$$

where $\mathbf{K}_{\sigma}(\cdot,\cdot)$ is a kernel function. Many recent works fall under this category, where either they compute the Gaussian discriminant analysis of features (Lee et. al [104]), view the features of the hidden layers between original and adversarial samples (Ma et. al [105]) or perform a joint statistical pooling amongst all the layers to detect adversarial samples (Raghuram et. al [106]). All of these methods have the ability to produce very reasonable results. However, applying kernel functions in the input space directly can be time-consuming depending on the dimensionality of the image. In addition, extracting the features of images from the layers of the neural network may cause some loss of information from the images itself.

5.2.2 Proposed Approach

We take inspiration from kernel based methods which use feature space information to train a kernel function. However, we also understand the motivation of

using a downsampled space—training a kernel function can be computationally inexpensive.

In addition to CNNs, our proposed approach uses a Discrete Cosine Transform (DCT) for dimensionality reduction and two Support Vector Machines, one each in the input and output space of the CNN classifier. In the following section, we refer to the CNN used by the proposed approach as an 'embedded' CNN.

Discrete Cosine Transform

Discrete Cosine Transform (DCT) is a form of matrix factorization and decomposition which generates a set of basis vectors of a class of Chebyshev polynomials [107, 108]. The DCT of some data $\mathbf{x_i} \sim \mathcal{X}$, where $x_i \in \Re^n$ is given by

$$G_x(k) = \frac{2}{M} \sum_{m=0}^{M-1} \mathbf{x}_i(m) \cos \frac{(2m+1)k\pi}{2M},$$
$$k = 1, 2 \ldots (M-1),$$

where $G_x(k)$ is the k^{th} DCT coefficient. The above form is for a single dimensional sequence. For a 2-dimensional matrix, where $\mathbf{x}_i \in \Re^{n_1 \times n_2}$ the DCT takes the form

$$G_x(k_1, k_2) = \sum_{m_1=0}^{n_1} \sum_{m_2=0}^{n_2} x_{m_1,m_2} \cos\left[\tfrac{\pi}{n_1}\left(n_1+\tfrac{1}{2}\right)k_1\right] \cos\left[\tfrac{\pi}{n_2}\left(n_2+\tfrac{1}{2}\right)k_2\right].$$

The time complexity of this method is $\mathcal{O}(n \log n)$, where n is the number of elements in the vector or matrix. Once the DCT is performed on the image, the resulting coefficients indicate the eigenvalues and their corresponding eigenvectors. In order to reduce the dimensionality, we remove the coefficients which are close to 0. This ends up being close to one-third of the entire image. To illustrate this, we show the example in figure. We use and MNIST ([109]) image, create the DCT transform of the image, remove a subsection of the image

Support Vector Machine (SVM)

We rely on the standard Support Vector Machine (SVM) [110] with a Radial Basis Function (RBF) kernel as an auxiliary classifier. An RBF is a strictly positive

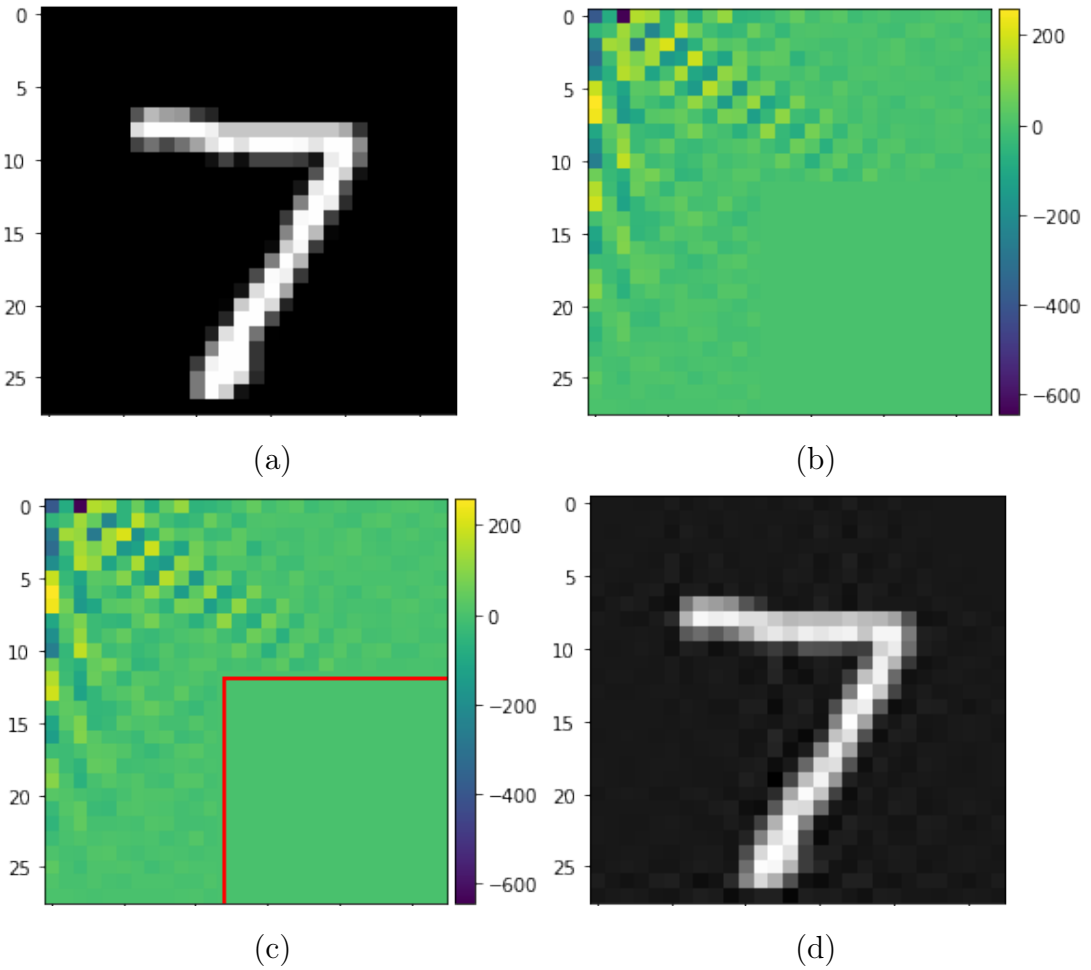


Figure 5.6: The effect of DCT with a subset of coefficients zeroed out on a sample image from MNIST. (a) Original image. (b) DCT coefficients computed by applying the DCT to the entire image along each dimension. (c) DCT coefficients with the lower left 16x16 block selected to be clipped. (d) Inverse DCT of the clipped coefficients from (c). Note that although there are some slight image artifacts, the main features of the digit remain the same.

definite functions whose values depends on some input and a central point. Given some dataset $\mathcal{X} = \{\mathbf{x}_0, \mathbf{x}_1 \ldots \mathbf{x}_m\}$ where $\mathbf{x}_i \in \mathbb{R}^n$ and some central point $\mathbf{x}_c$, an RBF yields a positive value $\phi(\|\mathbf{x}_i - \mathbf{x}_c\|)$. Here ϕ is the radial basis function that yields a radius about a central point $\mathbf{x}_c$. This function is often shape modified using a parameter ϵ. The Gaussian form of an RBF (which we have used in this section) is of the form

$$\phi(r) = e^{-\epsilon r^2},$$

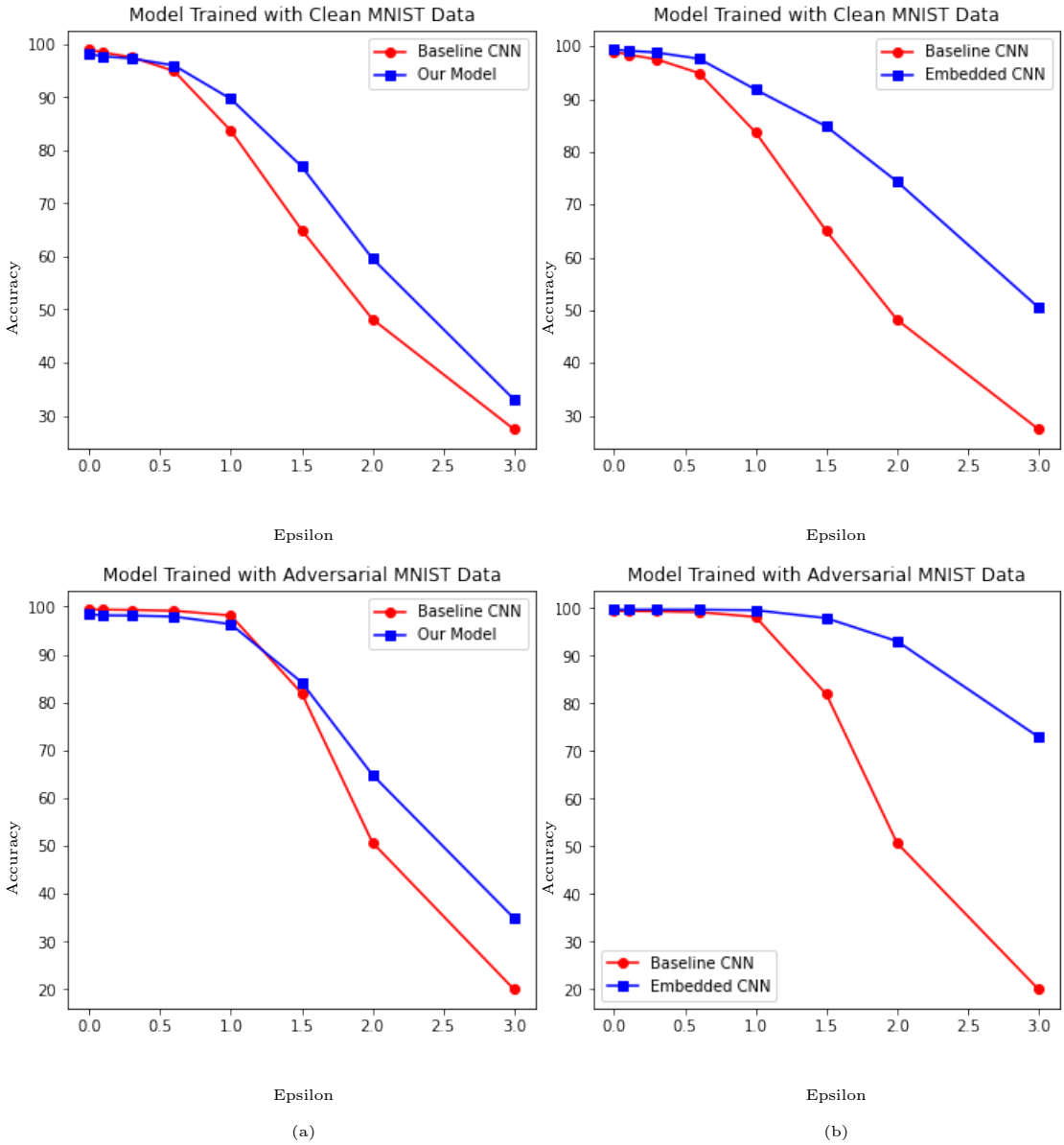


Figure 5.7: Accuracy of Baseline Model and Our Model on data samples with varying ϵ values, starting with clean data at $\epsilon = 0$. The baseline CNN and Embedded CNN were trained on clean (left two plots) and adversarial (right two plots) MNIST data.

where ϵ is a shape tuning parameter, r is the radial parameter defined by $\|\mathbf{x} - \mathbf{x}_i\|$, where $\mathbf{x}_i$ is an instance of a dataset $\mathcal{X}$. In the following sections, we elucidate the proposed approach by combining the discrete cosine transform, support vector machine and the neural networks.

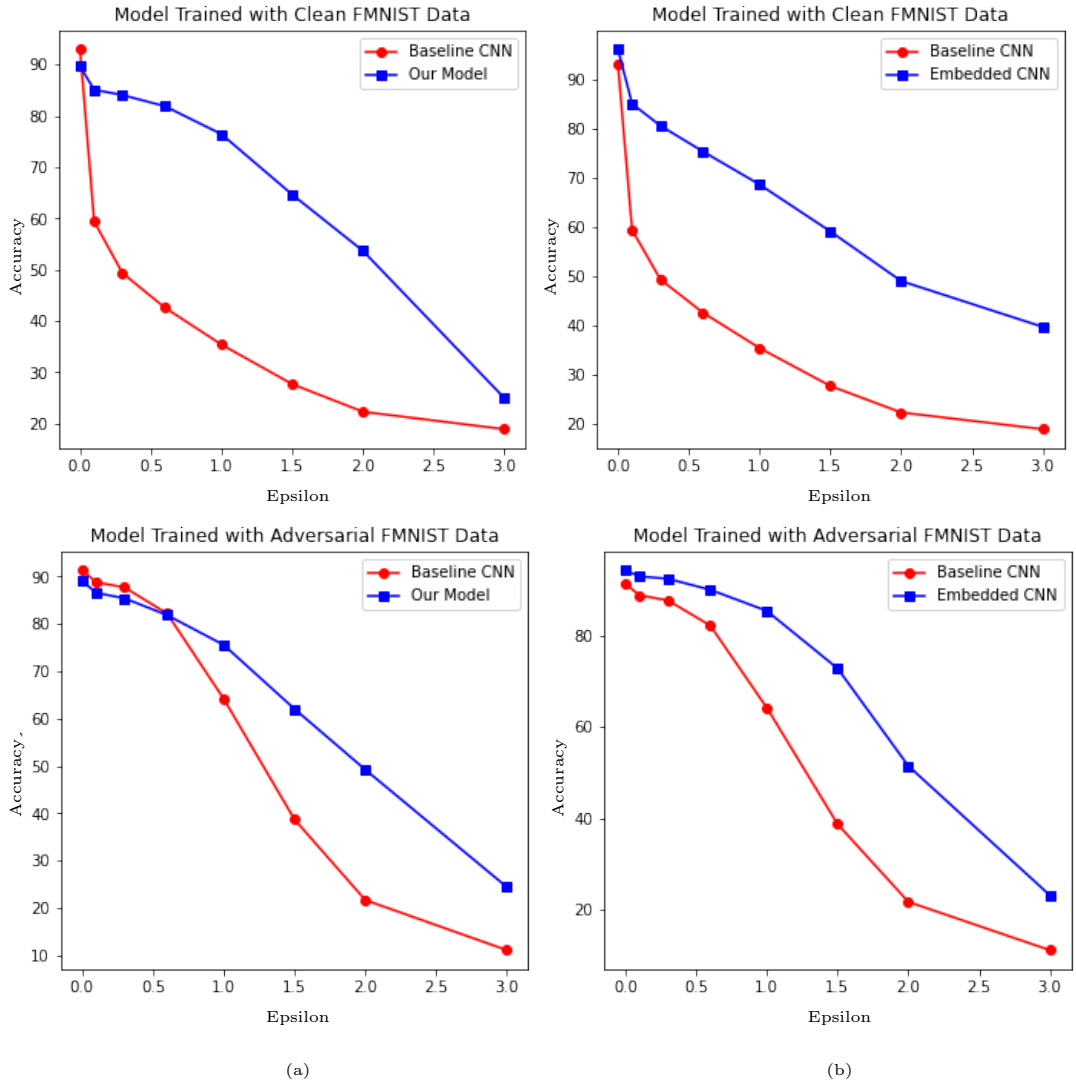


Figure 5.8: Accuracy of Baseline Model and Our Model on data samples with varying ϵ values, starting with clean data at $\epsilon = 0$. The baseline CNN and Embedded CNN were trained on clean (left two plots) and adversarial (right two plots) FMNIST data.

5.2.3 Experiments

For our experiments, we choose two very common dataset - MNIST and FMNIST. We train an input-space SVM using the full training dataset, with an RBF kernel and a C value of 100. We preprocess the data by normalizing it to 0 mean and unit variance, then applying the DCT transform to the entire image, removing the 16 highest-frequency coefficients along each dimension. The remaining coefficients are

then flattened into a single vector for training. The same preprocessing is applied during testing.

We train another, output-space SVM on the outputs of our target model (CNN) for natural (non-adversarial) data. We assume the target model applies a log softmax in its final layer and outputs a vector of size $n\_classes$, in our case 10. No preprocessing is done to these vectors before they are sent to the SVM for training or testing.

At test time, input to our model is classified by an RBF SVM in the input space and independently run through a CNN. After we receive CNN outputs, we also classify those with a different RBF SVM in our output space. If the SVM classification of the output disagrees with the SVM classification of the input, we flag this input as potentially adversarial and use its input space SVM classification to predict its class. If the data is not flagged as adversarial, we use the CNN's prediction.

The CNN architecture we use is a modified VGG [111] with 6 convolutional layers, each followed by a batch normalization and a ReLU layer. The convolutional layers have 64, 128, 256, 256, 512, and 512 filters, respectively. After the first, second, and fourth convolutional layers, maxpooling with a 2x2 filter is performed. Following the convolutional layers are three linear layers, each followed by a ReLU and dropout layer. The linear layers have 4096, 4096, and 10 units, respectively. We then take the log softmax of the output.

5.2.4 Abalation Study

We test our model alongside a normally trained CNN and an adversarially trained CNN, on both clean and adversarial data. The CNN used in our model is the same as the model we compare it to, since we treat the adversarial detector and input space classifier as supplementary to an existing model. To generate adversarial data, we apply a white-box attack to the CNN using PGD with 5 steps and a step-size of .01, using different total ϵ values. We tested on adversarial data with total $\epsilon \in [0.1, 0.3, 0.6, 1, 1.5, 2, 3]$. The adversarially trained CNN was trained on data with an ϵ of 0.3. We record the overall accuracy for each of the models (baseline CNN, our model), and the relative accuracies for our model's adversarial

Model	Trained with Clean MNIST Data							
Data (ϵ)	clean	0.1	0.3	0.6	1	1.5	2	3
Baseline Accuracy (%)	98.98	98.42	97.54	94.92	83.69	65.01	48.18	27.41
Our Model's Accuracy (%)	98.14	97.68	97.30	**95.98**	**89.68**	**76.96**	**59.61**	**33.02**
Detector Accuracy (%)	98.44	2.17	2.78	4.46	12.42	36.60	60.10	84.80
Input Cluster Relative Accuracy (%)	19.87	30.92	44.30	60.44	74.74	63.38	49.81	29.89
CNN Relative Accuracy (%)	99.38	99.17	98.82	97.64	91.80	84.81	74.36	50.49

Table 5.1: Results from model trained on clean MNIST data, and evaluated on clean and adversarial MNIST data. Accuracy of Baseline Model and Our Model on non-adversarial data and full sets of adversarial data with varying epsilon values refers to the overall percentage of correctly classified samples. On clean data, Detector Accuracy refers to the percentage of the dataset that the detector classifies as clean. On adversarial samples, it represents the percentage of the data that is classified as adversarial. Since the detector determines which samples are sent to each classifier embedded in our model, and this varies from run to run, we present the relative accuracy of each classifier on *only the data that it received in that run.*

Model	Trained with Adversarial MNIST Data							
Data epsilon	clean	0.1	0.3	0.6	1	1.5	2	3
Baseline Accuracy (%)	99.37	99.36	99.30	99.13	98.14	81.90	50.70	20.01
Our Model's Accuracy (%)	98.40	98.17	98.13	97.91	96.32	**84.19**	**64.86**	**34.88**
Detector Accuracy (%)	98.48	1.85	1.92	2.22	4.63	30.95	67.13	98.38
Input Cluster Relative Accuracy (%)	17.11	18.99	19.89	21.38	30.03	53.65	51.06	34.25
CNN Relative Accuracy (%)	99.65	99.66	99.66	99.65	99.54	97.87	93.04	72.94

Table 5.2: Results from model trained on adversarial MNIST data, and evaluated on clean and adversarial MNIST data.

Model	Trained with Clean FMNIST Data							
Data epsilon	clean	0.1	0.3	0.6	1	1.5	2	3
Baseline Accuracy (%)	92.96	59.37	49.33	42.58	35.39	27.67	22.29	18.89
Our Model's Accuracy (%)	89.63	85.09	84.04	81.87	76.41	64.63	53.65	24.92
Detector Accuracy (%)	89.66	39.08	49.84	59.72	67.80	75.23	81.28	94.80
Input Cluster Relative Accuracy (%)	33.56	85.18	87.54	86.28	80.11	66.46	54.73	24.12
CNN Relative Accuracy (%)	96.10	85.04	80.57	75.34	68.62	59.08	49.00	39.68

Table 5.3: Results from model trained on clean FMNIST data, and evaluated on clean and adversarial FMNIST data.

Model	Trained with Adversarial FMNIST Data							
Data epsilon	clean	0.1	0.3	0.6	1	1.5	2	3
Baseline Accuracy (%)	91.29	88.80	87.69	82.18	64.17	38.70	21.69	11.12
Our Model's Accuracy (%)	89.02	86.53	85.39	81.84	75.55	62.03	49.21	24.41
Detector Accuracy (%)	91.32	12.14	13.35	18.55	38.98	61.89	73.35	92.62
Input Cluster Relative Accuracy (%)	33.99	40.22	39.74	45.98	60.17	55.41	48.40	24.51
CNN Relative Accuracy (%)	94.25	92.93	92.42	90.00	85.37	72.78	51.43	23.12

Table 5.4: Results from model trained on adversarial FMNIST data, and evaluated on clean and adversarial FMNIST data.

data detector, SVM, and embedded CNN. To make this abalation study a little rigrous, we have used the same CNN architecture for the proposed approach and the adversarially trained model. All the methods have been trained on the same machine with the same conditions.

5.2.5 Results

For each ϵ value tested, we ran each version of our model and the baseline model 10 times, recording the overall accuracy for each, as well as the relative accuracies for the components of our model. Our model consists of a detector that first classifies the output data as adversarial or clean. The data that is flagged as adversarial is classified by the SVM, and we record the accuracy of the SVM on only that subset of the data. The data that is not flagged as adversarial is classified by the embedded CNN, and we record the accuracy of that model on that subset of the data. We refer to these accuracies as the relative accuracies for our model. We also record the accuracy of the adversarial data detector, and the time it takes for our model and the baseline CNN to make their predictions.

In the (a) portion of each plot in Figures 5.7 and , 5.8, we observe that our model has relatively high accuracy at low ϵ values, though it is often slightly less accurate than the baseline model on clean or lower ϵ data. We can consistently see, however, that as the magnitude of adversarial noise increases, our model's accuracy suffers with both the naturally trained and adversarially trained models. Further, by the threshold of $\epsilon = 1.5$, our model is consistently more accurate than a stan-

dalone CNN. The ability of our detector to accurately detect adversarial samples increases dramatically as ϵ increases, meaning that our model falls back on returning SVM predictions for the adversarial data rather than the CNN predictions, circumventing the target of the attack.

An interesting result to observe is in the disparity between the accuracy of a standalone CNN compared with the relative accuracy of the CNN embedded in our model at high ϵ values. In the (b) portion of each plot in Figure 5.7 and 5.8, for $\epsilon > 1$, the relative accuracy of the CNN embedded in our model is over 15% higher than the CNN on its own. Since the output space SVM boundaries are based on the outputs of clean data for the embedded CNN, we expect that this is a result of the output space SVM detecting erroneous predictions on the adversarial data that successfully receives an incorrect prediction from the CNN, but remains relatively similar in the input space.

In Tables 5.1–5.4, we observe that the input cluster relative accuracy is often very low compared to the embedded CNN relative accuracy. In principle, the input cluster SVM's prediction should only be used when the incoming data is adversarial, in which case we would expect CNN predictions to be unreliable while input images are relatively normal, so despite its low relative accuracy, we still see higher accuracies for the encompassing model. From our model's prediction times in Tables 5.1–5.4, we also observe that the model's inference is much slower than the baseline's, especially on FMNIST data. This likely comes down to the SVM inference time, which doesn't handle images or larger inputs as well as CNNs. As a supplement to a standalone CNN, our model offers some protection against white-box attacks, at the cost of the extra inference time to detect adversarial images.

5.2.6 Conclusion

In this section, we proposed two methods for defense agains adversarial attacks on image classifiers. In the first method, we adversarially trained the convolutional neural network model over adversarial samples generated from a distribution. In the second method, we downsampled the images to a lower dimension

using Discrete Cosine Transform and cluster those samples using a support vector machine (RBF). We investigated the robustness and training/testing response of these methods over various perturbation (ϵ) values on the MNIST and FMNIST datasets. We conclude that with some efficiency trade-off , we are able to achive a higher accuracy of prediction.

5.3 Multi-Stage Gaussian Denoising

In this section, we explain the model that we use to describe the noisy observations. Because we are operating under the assumption of adequate light levels during image acquisition, we use the AWGN model given by

$$\mathbf{y} = \mathbf{x} + \mathbf{g}$$

where $\mathbf{y} \in \mathbb{R}_+^n$ is the noisy observation vector, $\mathbf{x} \in \mathbb{R}_+^n$ is the true or clean signal and $g \in \mathbb{R}^n$ refers to the vector of AWGN with $g \sim \mathcal{N}(0, \sigma^2)$ where σ^2 denotes the variance of the Gaussian distribution. Our interest is in recovering the true noiseless signal $\mathbf{x}$ from the noisy observation $\mathbf{y}$.

Related work. Traditional denoising methods removing AWGN from images involve techniques which rely on operating in a different domain, such as the frequency domain [112, 113]. Baysian based approaches have also been very popular requiring not only the modeling of image priors but in many cases the hand tuning of parameters effecting the reconstruction quality [114–116]. In addition to parameter tuning, these algorithms are iterative and require computationally intensive optimization routines to arrive at a solution. As an alternative to optimization-based algorithms, deep neural networks have emerged as a viable alternative for solving the image denoising problem. A variety of architectures have been proposed, including convolutional neural networks (CNNs), residual neural networks, autoencoders, and multilayer perceptrons (MLPs) [117–120]. These methods are based on feed forward neural network architectures.

Diffusion models are probabilistic models designed to learn a data distribution $p(x)$ by gradually denoising a normally distributed variable, which corresponds to

learning the reverse process of a fixed Markov Chain of length T (see [121]). The authors of [122] designed an autoencoder and diffusion model where the images are downsampled by the encoder. This dowsampled space is denoised using the U-NET network. Once the denoising is done in the latent space, the decoder is used to upsample the image back to the original pixel space.

This work seeks an alternative to traditional optimization-based methods as well as typical feedforward based deep learning architectures by using a recurrent neural network (RNN) to approximate a mapping from the noisy observation $\mathbf{y}$ to the clean signal $\mathbf{x}$. The novelty of our approach is that we reformulate the Gaussian denoising problem as a multi-step denoising process allowing us to apply the RNN architecture.

5.3.1 Problem Formulation

In this section, we explain the model that we use to describe the noisy observations. Because we are operating under the assumption of adequate light levels during image acquisition, we use the AWGN model given by

$$\mathbf{y} = \mathbf{x} + \mathbf{g}$$

where $\mathbf{y} \in \mathbb{R}_+^n$ is the noisy observation vector, $\mathbf{x} \in \mathbb{R}_+^n$ is the true or clean signal and $g \in \mathbb{R}^n$ refers to the vector of AWGN with $g \sim \mathcal{N}(0, \sigma^2)$ where σ^2 denotes the variance of the Gaussian distribution. Our interest is in recovering the true noiseless signal $\mathbf{x}$ from the noisy observation $\mathbf{y}$.

Related work. Traditional denoising methods removing AWGN from images involve techniques which rely on operating in a different domain, such as the frequency domain [112, 113]. Baysian based approaches have also been very popular requiring not only the modeling of image priors but in many cases the hand tuning of parameters effecting the reconstruction quality [114–116]. In addition to parameter tuning, these algorithms are iterative and require computationally intensive optimization routines to arrive at a solution. As an alternative to optimization-based algorithms, deep neural networks have emerged as a viable alternative for

solving the image denoising problem. A variety of architectures have been proposed, including convolutional neural networks (CNNs), residual neural networks, autoencoders, and multilayer perceptrons (MLPs) [117–120]. These methods are based on feed forward neural network architectures. This work seeks an alternative to traditional optimization-based methods as well as typical feedforward based deep learning architectures by using a recurrent neural network (RNN) to approximate a mapping from the noisy observation $\mathbf{y}$ to the clean signal $\mathbf{x}$. The novelty of our approach is that we reformulate the Gaussian denoising problem as a multi-step denoising process allowing us to apply the RNN architecture.

5.3.2 Proposed Approach

In this section we describe the models implemented for image recovery from noisy observations. In particular we discuss two different architectures (labeled Methods I and II), both of which take advantage of the denoising properties of autoencoders. The first method attempts to directly map the noisy observation to the clean image, while the second method takes advantage of a specialized dataset which will be discussed in section 5.3.3 and reformulates the problem as an RNN.

Method I (Autoencoder). The first method is a direct approach using a convolutional autoencoder [123, 124]. This type of architecture has been successful in a variety of applications addressing different types of signal recovery applications [125, 126]. The encoder of the architecture, $\mathbf{E}$, consists of eight convolutional layers each followed by the leaky rectified linear unit (Leaky ReLU) [127, 128]. The result is a latent variable representation of the noisy image. The decoder, $\mathbf{D}$, then consists of eight convolutional transpose layers to bring back the image to the appropriate dimension. The output ($\hat{\mathbf{x}}$) is then compared to the target or clean image ($\mathbf{x}$) using the mean squared error (MSE) loss function given by

$$\text{MSE}(\hat{\mathbf{x}}, \mathbf{x}) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \|\hat{\mathbf{x}}_i - \mathbf{x}_i\|_2^2, \tag{5.5}$$

where $\mathcal{S}$ is the dataset and $|S|$ is its cardinality.

Method II (Recurrent Neural Network). The second method incorporates

the autoencoder of Method I in conjunction with the structure of a recurrent neural network. Recurrent neural networks have been shown to be effective in time dependent applications such as speech recognition, music transcription scene labeling and a variety of other applications requiring sequence learning [129–132]. As far as the authors of this work know, this is the first time that this type of architecture is being applied to the problem of denoising. The typical sequence to sequence RNN can be described by the following equations:

$$\begin{aligned} \mathbf{a}_t &= \mathbf{b} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{y}_t, \\ \mathbf{h}_t &= \tanh(\mathbf{a}_t), \\ \mathbf{o}_t &= \mathbf{c} + \mathbf{V}\mathbf{h}_t, \\ \hat{\mathbf{y}}_{t+1} &= \operatorname{softmax}(\mathbf{o}_t), \end{aligned} \tag{5.6}$$

where $\mathbf{b}$ and $\mathbf{c}$ are bias vectors and $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$ are weight matrices that map the input $\mathbf{y}_t$ to the approximation $\hat{\mathbf{y}}_\mathbf{t}$ using the hidden state $\mathbf{h}_t$ from times $t = 0$ to $t = \tau$ (see [23, 133] for details).

We modify and adapt the standard formulation (5.6) to a stage-wise denoising approach by treating the noise as a pseudo-temporal component. Given the noisy realization $\mathbf{y}$ with AWGN of intensity variance σ^2, we create a sequence of realizations $\{\mathbf{y}_0, \mathbf{y}_1 \dots \mathbf{y}_\tau\}$ with their associated values of σ^2 $\{\sigma_0^2, \sigma_1^2, \dots, \sigma_\tau^2\}$ arranged in a monotonically decreasing fashion. We initialize $\mathbf{y_0} = \mathbf{y}$ with $\boldsymbol{\sigma_0^2} = \boldsymbol{\sigma^2}$ and train the RNN to denoise the current realization to the next noisy realization in the sequence. For denoising at each stage, we use the Autoencoder from Method I. The following equation represents this model: a sequence of variances $\sigma_0^2 > \sigma_1^2 > \cdots > \sigma_\tau^2 > 0$, we create a sequence of noisy realizations $\{\mathbf{y}_0, \mathbf{y}_1 \dots \mathbf{y}_\tau\}$ of the ground truth image $\mathbf{x}$ and train the RNN through a sequence of denoising steps. We use the autoencoder from Method I for denoising at each step, which is modeled by the following equations:

$$\begin{aligned} \hat{\mathbf{z}}_t &= \mathbf{E}_t(\mathbf{y}_t, \mathbf{h}_t^\mathbf{e}), \\ \hat{\mathbf{y}}_{t+1} &= \mathbf{D}_t(\hat{\mathbf{z}}_t, \mathbf{h}_t^\mathbf{d}), \end{aligned} \tag{5.7}$$

where $\mathbf{E_t}$ and $\mathbf{D_t}$ are the encoder and decoder from Method I with corresponding weights $\mathbf{h}_t^\mathbf{e}$ and $\mathbf{h}_t^\mathbf{d}$, respectively, $\hat{\mathbf{z}}_\mathbf{t}$ is the latent space representation of the image,

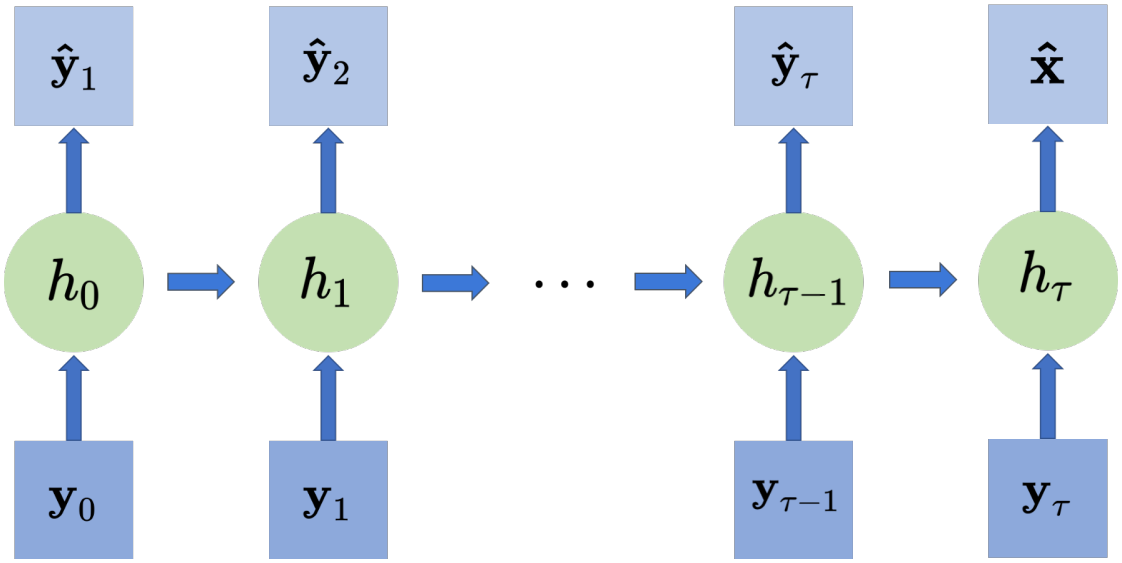
and $\hat{\mathbf{y}}_{t+1}$ is the approximation to $\mathbf{y}_{t+1}$. for a given noise σ_{t+1}. During Training, we feed an input $\mathbf{y_t}$ with some noise variance $\sigma_{\mathbf{t}}^{\mathbf{2}}$ through a hidden state at that step of the sequence $\mathbf{h_t} = [\mathbf{h_t^e}, \mathbf{h_t^d}]$ to approximate the true realization $\mathbf{y_{t+1}}$ in the next step with $\sigma^2 = \sigma_{\mathbf{t+1}}^{\mathbf{2}}$. The approximation ($\hat{\mathbf{y}}_{t+1}$) is then compared against the true realization using MSE. The weights $\mathbf{h_t} = [\mathbf{h}_t^{\mathbf{e}} \;\; \mathbf{h}_t^{\mathbf{d}}]$ are then propagated to the next step, and the process is repeated *mutatis mutandis*. We illustrate this RNN in Fig. 5.9. The last noisy realization in the sequence $\mathbf{y}_\tau$ (with the least amount of noise with variance σ_τ^2) is then mapped to an approximation of the clean image $\hat{\mathbf{x}}$ (see Fig. 5.9(a)). In a similar manner to the standard RNN, the approximations are all compared to their target using the MSE. All MSEs are summed and the loss is used to calculate the gradient across time with respect to the autoencoder weights. During testing, we no longer have access to this series of noisy realizations. In this scenario, we only provide the network with the initial noisy realization $\mathbf{y}_0$. The reconstructions from the previous input is used as input to create the next approximation (see Fig. 5.9(b)). The final output in the sequence corresponds to the denoised image.

5.3.3 Numerical Experiments

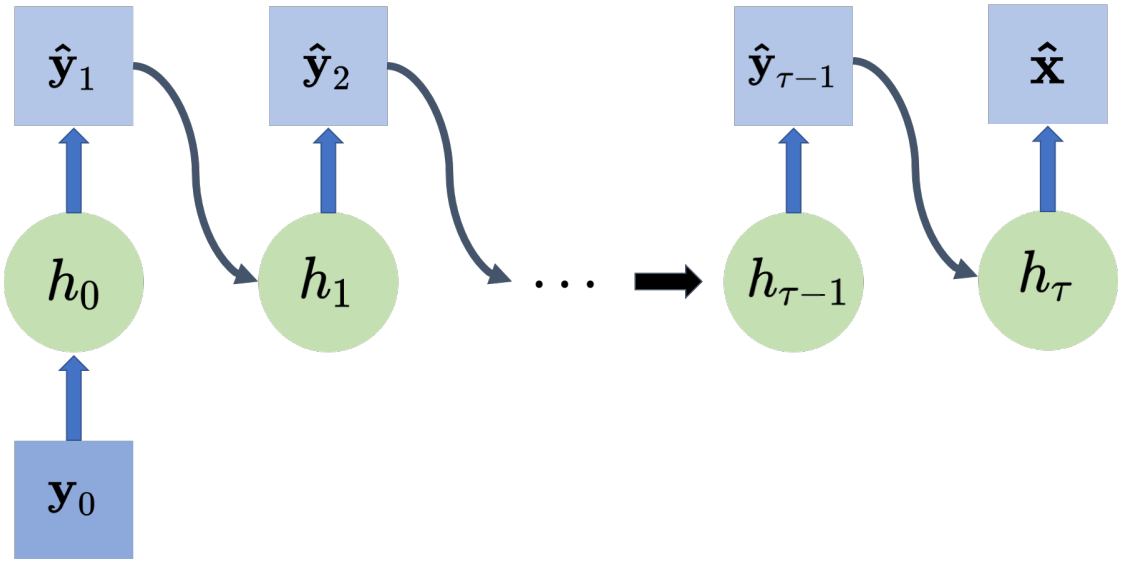
The architectures in Method I (AE) and in Method II (RNN) were all implemented using Pytorch. Training and testing were performed using an NVIDIA RTX 2080Ti. The networks were trained using the ADAM optimizer [28].

Dataset. We evaluate the effectiveness of the proposed methods on a modified version of the CIFAR-10 dataset [134], which we use as a very large comprehensive denoising test set. In particular, for our experiments, we converted the 50,000 training images with a size of 32×32 into single channel grayscale images. Three noisy realizations were created of each image. The noisiest realization contained AWGN with $\sigma_0^2 = 9.0 \times 10^{-3}$ while each subsequent realization was created with $\sigma_1^2 = 6.0 \times 10^{-3}$ and $\sigma_2^2 = 3.0 \times 10^{-3}$ (see Fig. 5.10). The test set is comprised of 10,000 images created in a similar manner to the training set. We note that we only use the noisiest realization ($\sigma_0^2 = 9.0 \times 10^{-3}$) during testing (see Fig. 5.9).

Performance. Both methods were trained for 100 epochs using the MSE in (5.5)



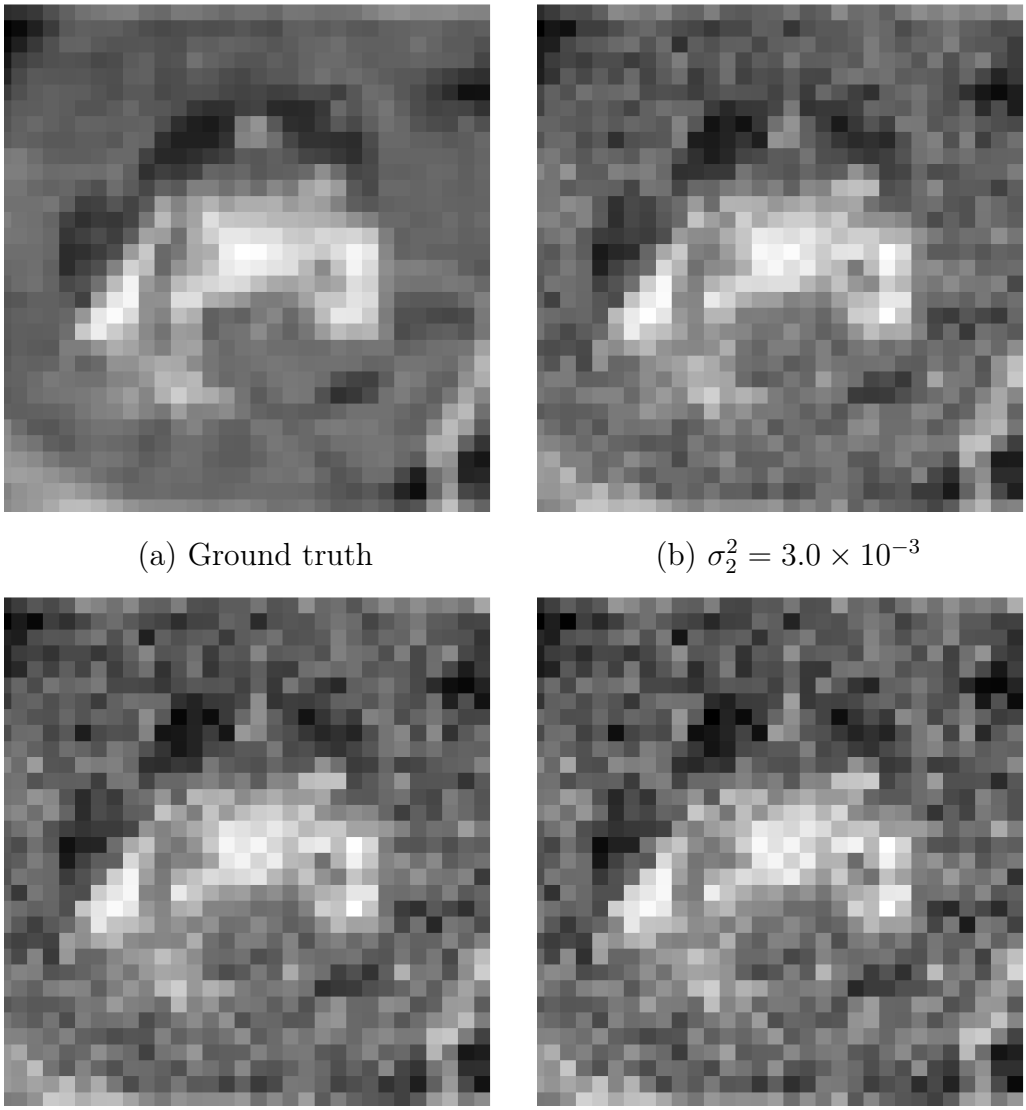
(a) Network configuration during training



(b) Network configuration during testing

Figure 5.9: The unrolled recurrent neural network (RNN) used for denoising. (a) During training, the output of each hidden layer is compared to the target using the mean squared error (MSE). The input at each hidden state is the target of the previous hidden state. (b) During testing, the output of each hidden layer is used as input in the next hidden state.

as a loss function. After reconstructing the test set images using both methods, two metrics were identified to evaluate the quality of the reconstructions. We first report the MSE between the reconstructed image and the original target. We also



(a) Ground truth (b) $\sigma_2^2 = 3.0 \times 10^{-3}$

(c) $\sigma_1^2 = 6.0 \times 10^{-3}$ (d) $\sigma_0^2 = 9.0 \times 10^{-3}$

Figure 5.10: Noisy realizations of an image at various noise variance. (a) Ground truth. (b)-(d) Noisy realizations with increasing noise intensities for increasing values of the variance σ_t^2.

report the structural similarity index (SSIM) given by

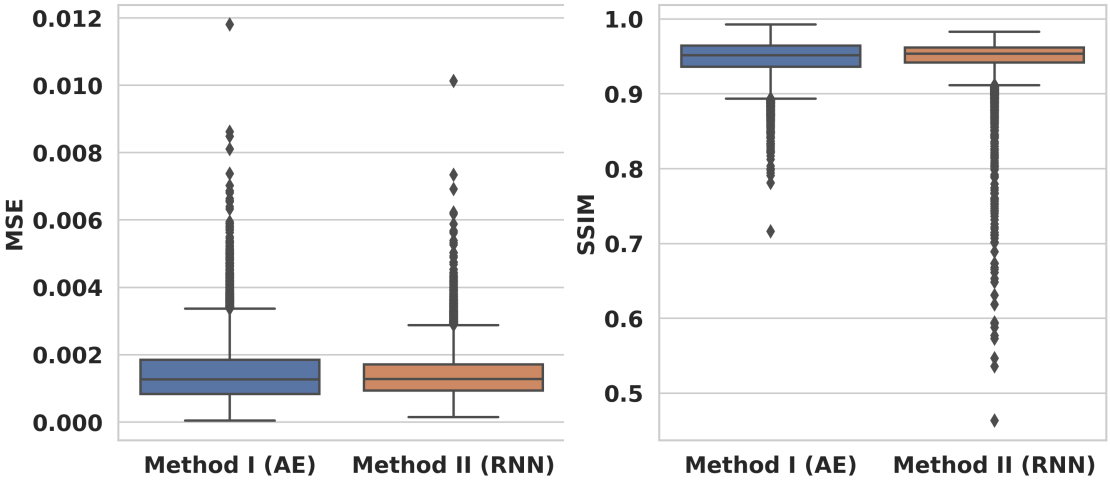
$$\text{SSIM}(\hat{\mathbf{x}}, \mathbf{x}) = \frac{(2\mu_{\hat{\mathbf{x}}}\mu_{\mathbf{x}} + c_1)(2\sigma_{xy} + c_2)}{(\mu_{\hat{\mathbf{x}}}^2 + \mu_{\mathbf{x}}^2 + c_1)(\sigma_{\hat{\mathbf{x}}}^2 + \sigma_{\mathbf{x}}^2 + c_2)}$$

where $\mu_{\hat{\mathbf{x}}}$ and $\mu_{\mathbf{x}}$ are the averages of each input respectively, $\sigma_{\hat{\mathbf{x}}}$ and $\sigma_{\mathbf{x}}$ are their associated variances, and c_1 and c_2 are variables used to stabilize the quotient.

The metric is a way to capture perceptual differences in the reconstruction (see [135] for details).

Complexity. Method I (AE) takes in the image once and denoises it in one forward and one backward propagation. Hence, the upper bound order of complexity is $\mathcal{O}(kn)$, where k is the number of features extracted (equal to the number layers in the network) from the image and n is the number of pixels in the image. On the other hand, our proposed method, Method II (RNN), forward and backward propagates each image in three stages, thus increasing the complexity order to $\mathcal{O}(3kn)$, which is still an $\mathcal{O}(n)$ method, ignoring constants coefficients.

5.3.4 Results



(a) MSE(I) $= 1.446 \times 10^{-3}$
MSE(II) $= 1.394 \times 10^{-3}$

(b) SSIM(I) $= 9.478 \times 10^{-1}$
SSIM(II) $= 9.485 \times 10^{-1}$

Figure 5.11: Performance metrics for Method I (Autoencoder) and Method II (Recurrent Neural Network) for 10,000 test images using the mean squared error (MSE) and the structural similarity index (SSIM). (a) MSE(I) and MSE(II) represent the MSE corresponding to Methods I and II, respectively. (b) SSIM(I) and SSIM(II) represent the SSIM corresponding to Methods I and II, respectively. Note that Method II has lower MSE and higher SSIM values.

The results of our experiments are presented in Figs. 5.11 and 5.12. The average test error of Method I (Autoencoder) is 1.446×10^{-3} while the average test

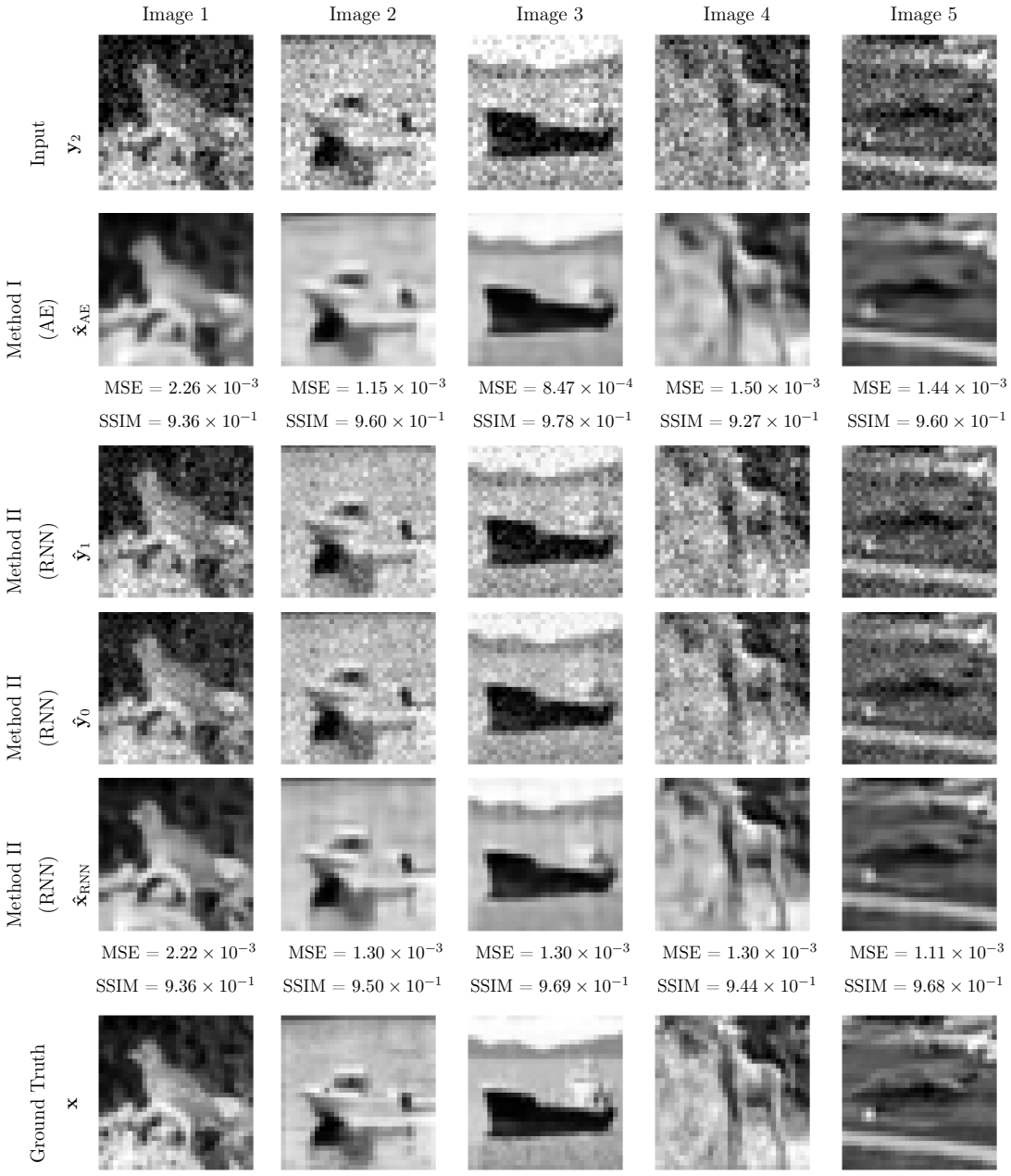


Figure 5.12: Numerical experiments on 5 images from the CIFAR-10 dataset. Row 1: Noisy input images $\mathbf{y}$. Row 2: Reconstructions $\hat{\mathbf{x}}_{\text{AE}}$ using Method I. Rows 3 and 4: Intermediate reconstructions within Method II. Row 5: Final reconstructions $\hat{\mathbf{x}}_{\text{RNN}}$ using Method II. Row 6: Ground truth images $\mathbf{x}$. MSE and SSIM values for both Methods I (AE) and II (RNN) are presented for each image.

error of Method II (Recurrent Neural Network) is lower, with MSE $= 1.394 \times 10^{-3}$. However, one must note that the MSE, by itself, may not be the best compara-

tive measure of reconstruction accuracy. Hence, in addition to the MSE, we also consider the SSIM and compare the performance of both methods. We notice that the SSIM accuracy for Method I (AE) is 9.478×10^{-1} while the SSIM accuracy for Method II (RNN) is higher, with SSIM $= 9.485 \times 10^{-1}$. We illustrate this difference in performance with five sample images in Fig. 5.12. The reconstructions using Method I (AE) are labeled $\mathbf{x}_{\text{AE}}$, and the reconstructions using Method II (RNN) are labeled $\mathbf{x}_{\text{RNN}}$. The input images are labeled $\mathbf{y}_2$. Method I (AE) tends to smooth some of the edges of the primary object of the images. In contrast, Method II (RNN) is able to recover some of the finer details of the image. The images $\hat{y_1}$ and $\hat{y_0}$ are the intermediate reconstructions at $\sigma_1^2 = 6.0 \times 10^{-3}$ and $\sigma_2^2 = 3.0 \times 10^{-3}$, respectively.

5.3.5 Conclusions

In this section we implemented two different deep learning architectures to recover signals under a Gaussian noise model. The first method, Method I (AE), is a convolutional autoencoder which maps the noisy realization directly to the denoised image. The second method, Method II (RNN), reformulates the image denoising problem as a multi-stage denoising process where the variance of the noise is gradually decreased using a recurrent neural network. Numerical experiments show a tendency for the autoencoder to produce smoother reconstructions. In some cases the RNN shows an improvement on the ability to capture detail producing an improved average test error using the MSE and a higher average SSIM.

5.4 Multi-Stage Mixed-Poisson-Gaussian Denoising

The amount of data being collected about our lives is increasing as a result of advances in the technology used to record these digital signals. Although the precision of these recording devices is becoming increasingly sophisticated, they are still susceptible to sources of noise during acquisition. This is especially true

in digital imaging where noise can be caused by sudden changes of light intensity, increase in temperature of the imaging apparatus or electrical fluctuations during the transmission of the signal. Typically this type of noise is modeled as additive-white-Gaussian-noise (AWGN) which is signal-independent [136, 137]. In contrast, for applications in a photon-limited or low-light setting, the measurements at the detector are corrupted by Poisson or one-shot noise and thus modeled using a Poisson model [138]. This type of regime is typically found in medical imaging, including but not limited to MRI reconstruction and microscopy (see e.g.,[139–141]). Denoising under this regime is particularly difficult owing to the fact that the noise is signal-dependent unlike AWGN. Further complicating the signal recovery process is the signal degradation by AWGN [142]. The section organization is as follows - In sec. 5.3.1, we discuss the problem formulation, in sec. 5.3.2, we discuss the proposed approach, in sec. 5.3.3, we discuss the numerical experiments of the proposed approach and in sec. 5.3.4, we discuss the results.

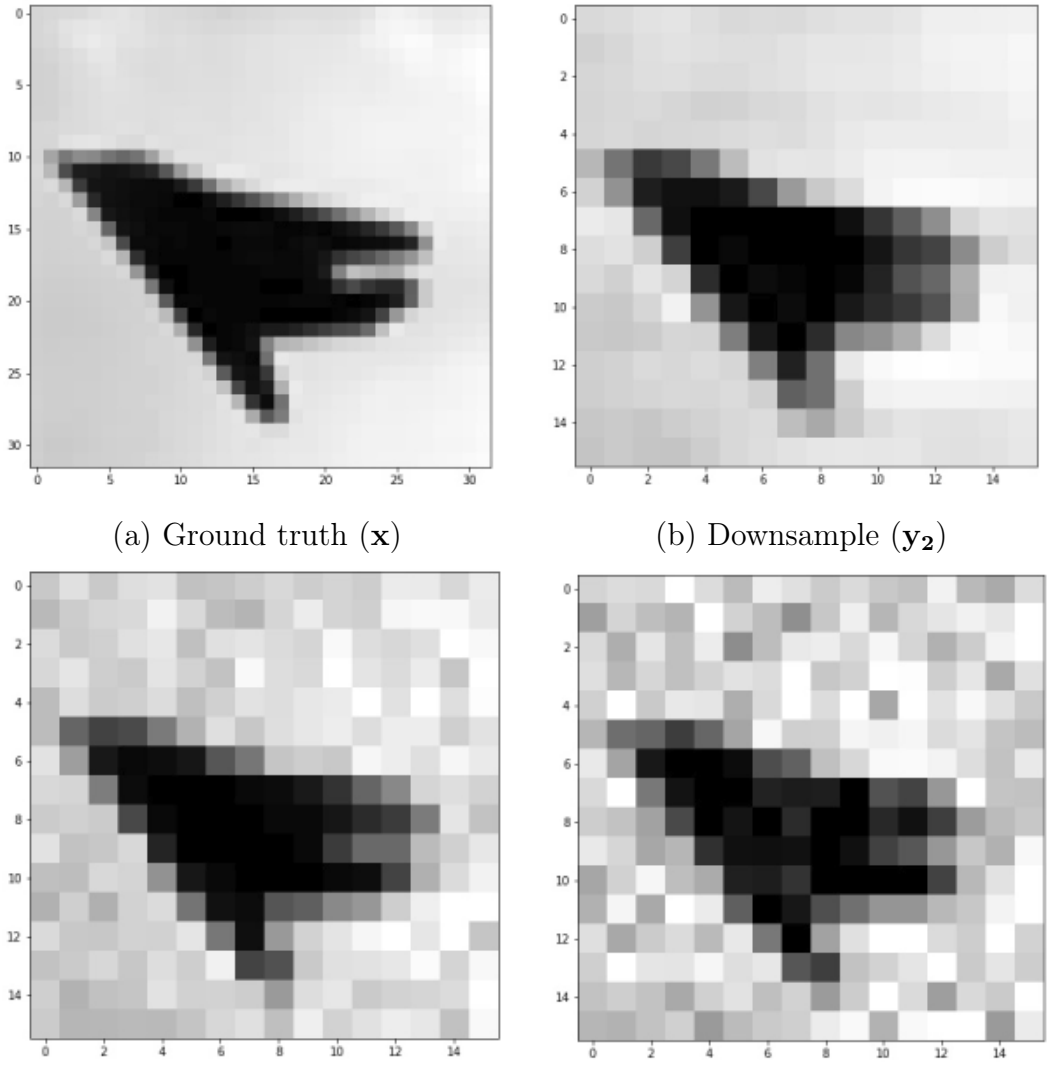
5.4.1 Problem Formulation

The Poisson-Gaussian noisy signal problem is an extension of the Poisson process model given by

$$\mathbf{y} = \mathcal{P}(\mathbf{A}(\mathbf{x})) + \mathbf{g}, \tag{5.8}$$

where $\mathbf{y} \in \mathbb{R}_+^m$ is the noisy, downsampled observation vector, $\mathcal{P}(\cdot)$ denotes the operator which imposes the signal dependent Poisson noise, $\mathbf{x} \in \mathbb{R}_+^m$ is the noiseless signal of interest, $\mathbf{g} \in \mathbb{R}^n$ refers to the vector of AWGN with $\mathbf{g} \sim \mathcal{N}(0, \sigma^2)$ where σ^2 denotes the variance of the Gaussian distribution, and $\mathbf{A}(\cdot)$ is a downsampling operator projecting the signal of interest to the observation space $\mathbb{R}^m$. The goal of this work is to estimate $\mathbf{x}$, denoted $\hat{\mathbf{x}}$, from the observational vector $\mathbf{y}$.

Related work. Traditionally, the problem of removing AWGN from images has been approached using least squares approximations and filters in the frequency domain [112]. Because of the signal dependent nature of Poisson corruption, more sophisticated algorithms use statistical based methods in conjunction with numerical optimization to maximizing the probability of observing the noisy realization



(a) Ground truth ($\mathbf{x}$)

(b) Downsample ($\mathbf{y_2}$)

(c) Poisson noise ($\mathbf{y_1}$)

(d) Mixed noise ($\mathbf{y_0}$)

Figure 5.13: Different realizations of an image at various stages of the observational process. (a) Ground truth. (b) Downsampled realization of (a). (c) Downsampled realization with Poisson noise. (d) Downsampled realization with mixed Poisson and Gaussian noise.

given the true signal or clean image. With the increased interest of deep learning as a result of the success in applications related to computer vision, the field of deep learning as a tool for image processing has grown substantially in the past ten years. Residual neural networks as well as deep convolutional autoencoders have emerged as viable methods for denoising in a variety of applications as well

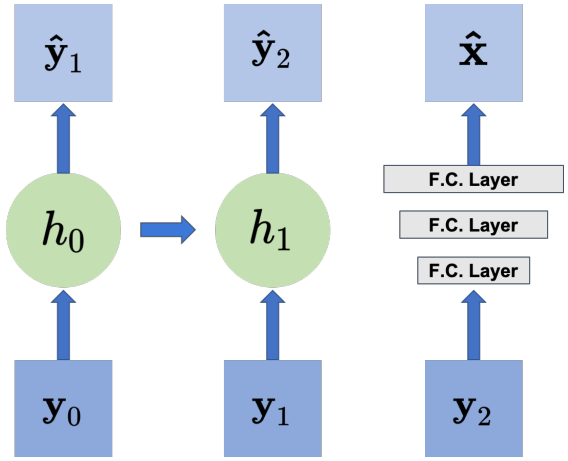
as a variety of types of noise [117, 118, 143]. There has been some work done using RNNs for image denoising (see e.g., [144, 145]) and image restoration (see e.g., [146]). However, to the best of the authors' knowledge, this is the first time they are used to denoise and upsample images with mixed Gaussian-Poisson noise.

In this section, we proposed a recurrent neural network in order to denoise images which have been corrupted by Poisson-Gaussian noise. The novelty of this method is two fold. First, we are applying deep learning techniques to solve the mixed Poisson-Gaussian denoising problem. While other methods have solved each denoising problem separately, we aim to solve both problems within the context of a recurrent neural network. Secondly, while most applications consider a direct mapping from the noisy representation to the noiseless reconstruction, we choose to use a recurrent neural network (RNN). The motivation for using an RNN comes from viewing the denoising of the Poisson-Gaussian realization as a temporal process. The network then attempts to trace back the steps that create the noisy realization in order to arrive at the noiseless reconstruction.

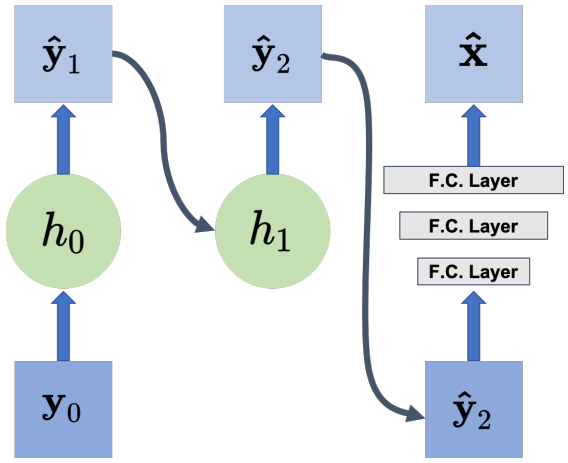
5.4.2 Proposed Approach

In this section we describe the models implemented for image recovery from noisy downsampled realizations. In particular we discuss two different architectures (labeled Methods I and II), both of which take advantage of the denoising properties of autoencoders. The first method directly maps the noisy observation to the clean image, while the second method takes advantage of a specialized dataset which will be discussed in section 4 and reformulates the problem as an RNN.

Method I (Autoencoder). The first method is an existing direct approach using a convolutional autoencoder [123, 124]. This type of architecture has been successful in a variety of applications addressing different types of signal recovery applications [125, 126]. The encoder of the architecture, **E**, consists of eight convolutional layers each followed by the leaky rectified linear unit (Leaky ReLU) [127, 128]. The result is a latent variable representation of the noisy image. The decoder, **D**, then consists of eight convolutional transpose layers to bring back the



(a) Network configuration during training



(b) Network configuration during testing

Figure 5.14: The unrolled recurrent neural network (RNN) used for denoising. (a) During training, the output of each hidden layer is compared to the target using the mean squared error (MSE). The input at each hidden state is the target of the previous hidden state. (b) During testing, the output of each hidden layer is used as input in the next hidden state.

image to the appropriate dimension.The output ($\hat{\mathbf{x}}$) is then compared to the target or clean image ($\mathbf{x}$) using the mean squared error (MSE) loss function given by

$$\text{MSE}(\hat{\mathbf{x}}, \mathbf{x}) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \|\hat{\mathbf{x}}_i - \mathbf{x}_i\|_2^2, \tag{5.9}$$

where $\mathcal{S}$ is the dataset and $|S|$ is its cardinality.

Method II (Recurrent Neural Network). The second method incorporates

the autoencoder of Method I in conjunction with the structure of a recurrent neural network. Recurrent neural networks have been shown to be effective in time dependent applications such as speech recognition, music transcription, scene labeling and a variety of other applications requiring sequence learning [129, 147]. As far as the authors of this work know, this is the first time that this type of architecture is being applied to the problem of denoising. The typical sequence to sequence RNN can be described by the following equations:

$$\begin{aligned} \mathbf{a}_t &= \mathbf{b} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{y}_t, \\ \mathbf{h}_t &= \tanh(\mathbf{a}_t), \\ \mathbf{o}_t &= \mathbf{c} + \mathbf{V}\mathbf{h}_t, \\ \hat{\mathbf{y}}_{t+1} &= \text{softmax}(\mathbf{o}_t), \end{aligned} \tag{5.10}$$

where $\mathbf{b}$ and $\mathbf{c}$ are bias vectors and $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$ are weight matrices that map the input $\mathbf{y}_t$ to the approximation $\hat{\mathbf{y}}_{\mathbf{t+1}}$ using the hidden state $\mathbf{h}_t$ from times $t = 0$ to $t = \tau$.

We modify and adapt the standard formulation (5.6) to a stage-wise denoising approach by treating the noise as a pseudo-temporal component. For a given signal of interest ($\mathbf{x}$) we create a sequence of noisy realizations $\{\mathbf{y_0}, \mathbf{y_1}, \mathbf{y_2}\}$ where $\mathbf{y_0}$ is the observation vector $\mathbf{y}$, $\mathbf{y_1}$ is the realization with Poisson noise and $\mathbf{y_2}$ is the downsampled realization without noise. We train the RNN through a sequence of denoising steps. We use the autoencoder from Method I for denoising at each step, which is modeled by the following equations:

$$\hat{\mathbf{z}}_t = \mathbf{E}_t(\mathbf{y}_t, \mathbf{h}_t^{\mathbf{e}}), \quad \text{and} \quad \hat{\mathbf{y}}_{t+1} = \mathbf{D}_t(\hat{\mathbf{z}}_t, \mathbf{h}_t^{\mathbf{d}}),$$

where $\mathbf{E_t}$ and $\mathbf{D_t}$ are the encoder and decoder from Method I with corresponding weights $\mathbf{h}_t^{\mathbf{e}}$ and $\mathbf{h}_t^{\mathbf{d}}$, respectively, $\hat{\mathbf{z}}_{\mathbf{t}}$ is the latent space representation of the image, and $\hat{\mathbf{y}}_{t+1}$ is the approximation to $\mathbf{y}_{t+1}$. The weights $\mathbf{h_t} = [\mathbf{h}_t^{\mathbf{e}} \quad \mathbf{h}_t^{\mathbf{d}}]$ are then propagated to the next step, and the process is repeated. We illustrate this RNN in Fig. 5.9.

The last noisy realization in the sequence $\mathbf{y}_2$ (the clean downsampled image) is then upsampled to an approximation of the clean image $\hat{\mathbf{x}}$ using fully connected

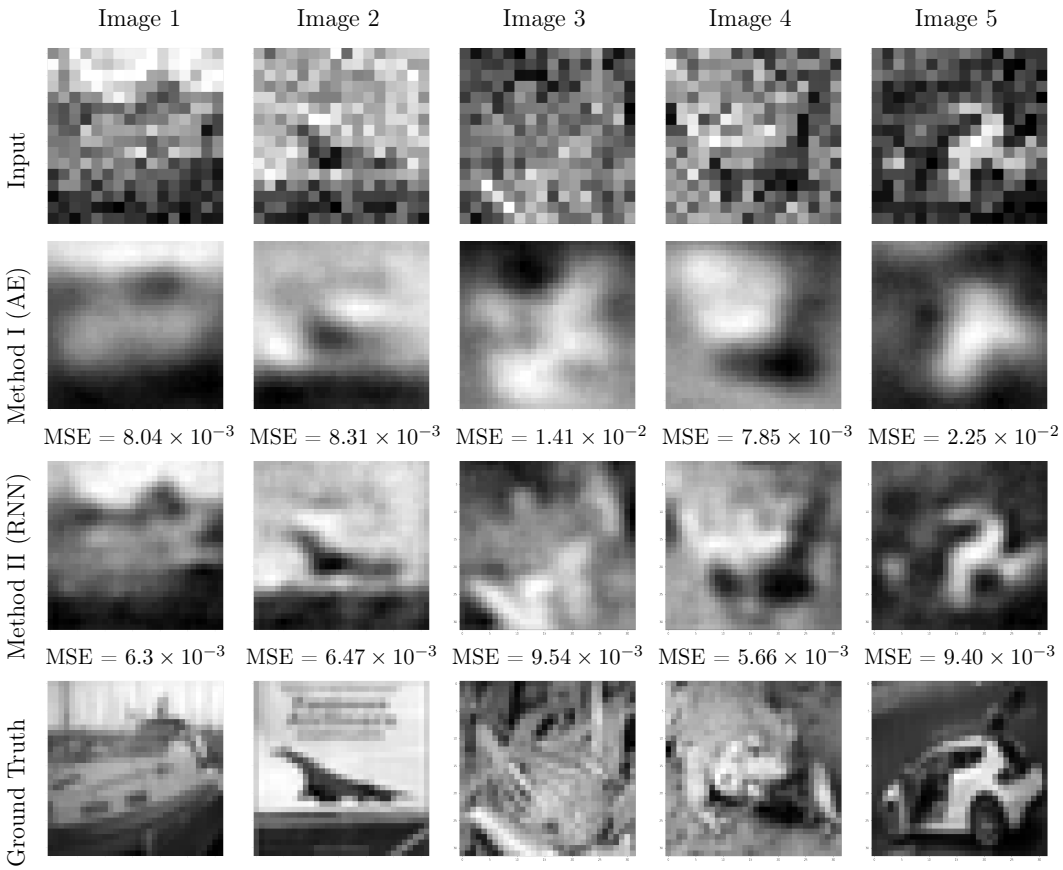


Figure 5.15: Numerical experiments on 5 images from the CIFAR-10 dataset. Row 1: Noisy input images $\mathbf{y}$. Row 2: Reconstructions $\hat{\mathbf{x}}_{\text{AE}}$ using Method I. Row 3: Final reconstructions $\hat{\mathbf{x}}_{\text{RNN}}$ using Method II. Row 4: Ground truth images $\mathbf{x}$. MSE values for both Methods I (AE) and II (RNN) are presented for each image.

layers. In a similar manner to the standard RNN, the approximations are all compared to their target using the Mean-Squared Error (MSE). All MSEs are summed and the loss is used to calculate the gradient across time with respect to the autoencoder weights. During testing, we no longer have access to this series of noisy realizations. In this scenario, we only provide the network with the initial noisy realization $\mathbf{y}_0$. The reconstructions from the previous input is used as input to create the next approximation (see Fig.5.9(b)). Once again, the final output in the sequence is considered the clean downsampled approximation.

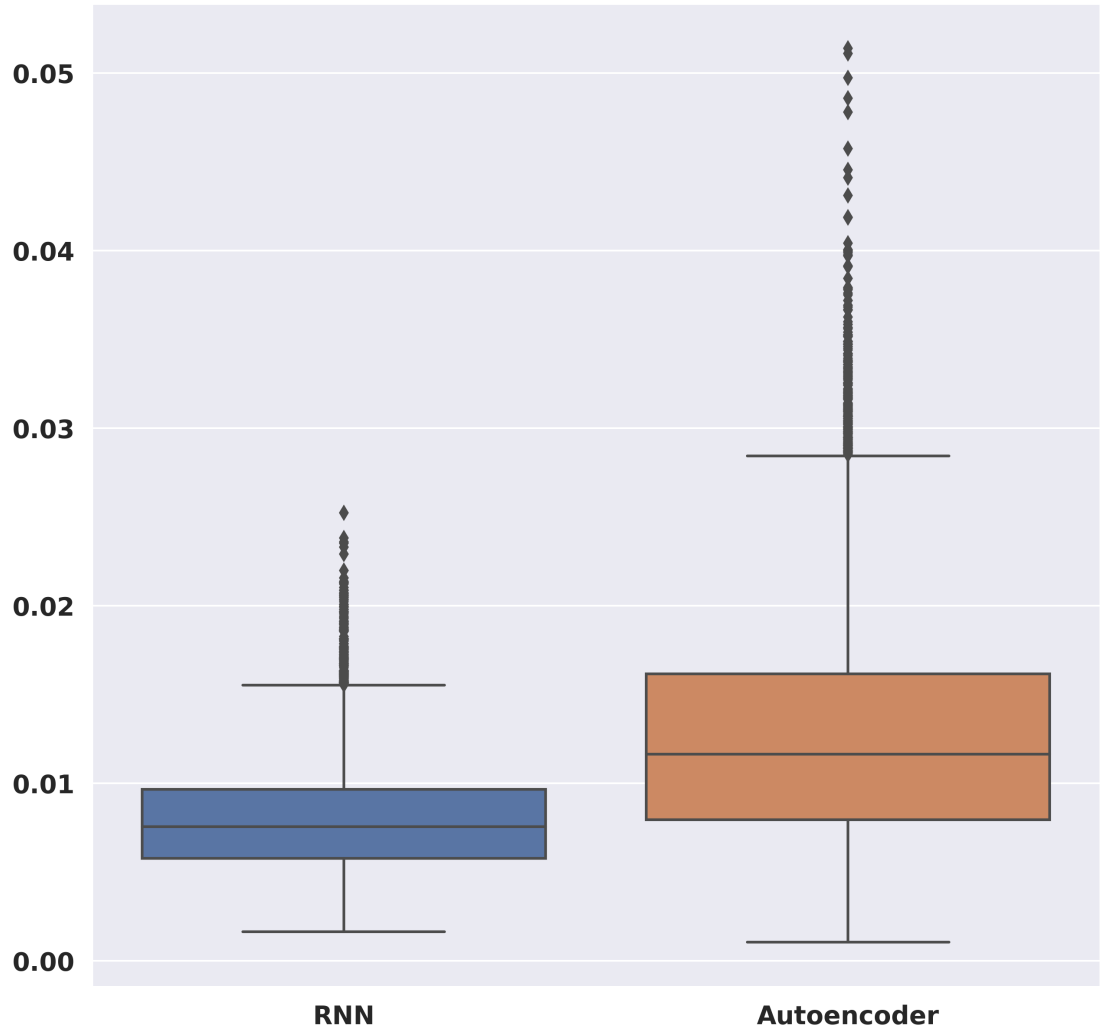


Figure 5.16: Testing mean squared error for the recurrent neural network (RNN) and the autoencoder methods. Note the better performance for the RNN approach both in mean and variance.

5.4.3 Numerical Experiments

The architecture in Method I (AE) and in Method II (RNN) were all implemented using Pytorch. Training and testing were performed using an NVIDIA RTX 2080Ti. The networks were trained using the Adam optimizer [28]. In order to evaluate the effectiveness of Methods I (AE) and II (RNN), we used a modified version of the CIFAR-10 dataset [46]. This modification can be accurately viewed from Fig. 5.13. The explanation of each realization in Fig. 5.13 is given in detail

in Sec. 5.3.4.

The ground truth image is a modified CIFAR-10 32×32 pixel image. We modify the image by representing the RGB image as a grayscale image. This image is then downsampled to a 16×16 space using the convolution operator in Pytorch using average pool operation. Then the Poisson noise is added, followed by the Gaussian noise with a standard deviation of $\sigma = 0.01$.

We reiterate the procedure aforementioned for the reader's convenience and illustrate it in Fig. 5.9.

Training: The images from each realization are fed as an input, and the realization from the next image is used as a comparison for the MSE metric. For example, in Fig. 5.13, the noisy and downsampled image $\mathbf{y_0}$ in Fig. 5.13(d) is fed as an input and compared to $\mathbf{y_1}$ in Fig. 5.13(c) to obtain $\hat{\mathbf{y}}_\mathbf{1}$ as output. Then $\hat{\mathbf{y}}_\mathbf{1}$ is fed as an input in the next cell of the RNN, whose output $\hat{\mathbf{y}}_\mathbf{2}$ is compared to the noiseless downsampled image $\mathbf{y_2}$ in Fig. 5.13(b). Finally, the ouput $\hat{\mathbf{y}}_\mathbf{2}$ is upsampled and compared to the ground truth image $\mathbf{x}$ in Fig. 5.13(a). This results in a final loss function as

$$\mathbf{L} = \|\mathbf{x} - \hat{\mathbf{x}}\|_2 + \|\mathbf{y}_1 - \hat{\mathbf{y}}_1\|_2 + \|\mathbf{y}_2 - \hat{\mathbf{y}}_2\|_2.$$

Testing: A new image $\mathbf{y_0}$ is fed to the first cell, whose output $\hat{\mathbf{y}}_\mathbf{1}$ is directly fed to the next cell to obtain the downsampled but denoised image $\hat{\mathbf{y}}_\mathbf{2}$. This image is then upsampled using fully-connected layers to obtain the denoised upsampled image $\hat{\mathbf{x}}$.

In contrast to the proposed approach, we feed the downsampled image with Poisson and Gaussian noise to the autoencoder and achieve an upsampled image which is directly compared against the ground truth. The reader should note that the upsampling is done before the encoder procedure.

5.4.4 Results

Fig. 5.13 illustrates the various stages of the observation model described in eq. (5.8). Fig. 5.13(a) is the ground truth image. Fig. 5.13(b) is the downsampled clean realization of the ground truth. Fig. 5.13(c) represents the downsampled

images with the Poisson image. Lastly, Fig. 5.13(d) represents the downsampled image with Poisson image and added Gaussian noise.

In Fig. 5.12, we present representative results of our proposed RNN method (Method II) in comparison to Method I, which is based on autoencoders. The first row contains the noisy and downsampled input images for both Method I (AE) and Method II (RNN). The second row of the figure represents the reconstructions using Method I (AE). The third row represents the reconstructions using our proposed method, Method II (RNN). The last row contains the the ground truth images. Note that the independent MSE loss of the images using Method I (AE) are noticeably higher than those of our method. Specifically, the average MSE loss of the Method I (AE) is 1.2×10^{-2}, while the average MSE loss of our method is 7.9×10^{-3}. For more details, see Fig. 5.16. The reader can additionally notice that the structural integrity of the image is stronger in the reconstructions from Method II (RNN) than from Method I (AE).

5.4.5 Conclusion

In this section, we proposed two architectures for image recovery from noisy downsampled realizations. To the best of our knowledge, recurrent neural networks have not been used for image denoising and upsampling. The motivation for its use comes from viewing the denoising of the Poisson-Gaussian realization as a temporal process. The network traces back the steps that created the noisy realization to arrive at the reconstruction. Experimental results show that the proposed method outperforms classical methods in terms of MSE loss and can recover more information from the downsampled realization. The proposed approach is very intuitive in design and simple to implement with comparable space and time complexity as the Autoencoder approach.

Chapter 6

Summary Of Contributions

Through this thesis we have explored various second order techniques in deep learning and machine learning. We will now highlight the main contributions of these approaches in this chapter.

In chapter 2, we proposed a Hessian-Free Trust-Region approach which uses the Hessian information to minimize an approximation of the objective function. The main contribution of this approach was to improve the minimizing iterates without explicitly storing or inverting the Hessian information, but still using the true Hessian information. using fast-exact Hessian-Vector products. We compare the proposed approach to Stochastic Gradient Descent (SGD) and realize - even with the same computational budget, the Trust-region Hessian-Free approach is able to perform better in a range of data limited tasks, where the input dataset is limited to fewer data points.

In chapter 3, we proposed an Adaptive-Regularized Cubics approach, which uses a quasi-Newton approximation. In this approach, even thought we use an approximation of the Hessian, the Limited-Symmetric Rank one (L-SR1) approach is able to capture negative-curvature information, which is absent in other quasi-Newton methods. In addition, due to the special-norm structure, we are able to solve the cubic-regularized subproblem exactly. through rigourous experiment and theoretical proofs, we have shown that the method is able to outperform many state-of-the-art approaches.

In chapter 4, we combined the benefits of both, the quasi-Newton approximation and the exponential moving average moments and momentum of Adam, and propose a quasi-Adam approach. The exponential moving average stabilizes the direction by averaging over all the gradient directions at each iteration. At the same time, the L-BFGS approximates the Hessian matrix, which contains the curvature information. This motivates the iterate from getting stuck at saddle-points. The computational overhead is also bounded since we only use 1 step and change in gradient in the past to approximate the Hessian. Through experiments with different datasets, deep learning models and with a comparable computational budget, we were able to show the method is able to perform better in comparison to Adam.

In addition to good optimization strategies, a well designed deep learning model is just as important. We discuss a variety of these tasks Chapter 5.

In Sec. 5.1, we proposed a novel deep learning architecture to separate two superimposed images. The chapter addresses a signal recovery problem - two signals are multiplexed due to a common measuring instrument, which need to be separated into the original signals. To address this issue, we proposed two deep learning architectures - convoution based neural network model and a transformer based model. Through experimentation, we were able to show that the proposed approach was able to outperfom the convolutional neural network well, with a smaller network footprint than the convolutional network. This demonstrates the networks robust capabilities against the task.

In Sec. 5.2, we proposd a novel adversarial defense technique, to defend against a white-box attack. We used a Radial-Basis Function to cluster the input space and the output space. Given the input-space classes lie in a distribution, the true images get clustered together while the different classes get separated from each other. We detect the adversarial sample if the sample lies in one cluster in the input class, but is classified into a different cluster once it is passed into a neural network. To tackle the computational overhead induced by the clustering methods, the images are downsampled using a Discrete Cosine Transform. Through experimentation, we were able to show that the proposed method was able to significantly outperform an adversarially trained neural network.

In Sec. 5.3, we proposed a recurrent neural network architecture to address the limited photon imaging problem. The main contribution of this approach is to denoise a Gaussian-noisy images in stages instead of direct denoising, which is more commonly adopted in an Autoencoder setting. This method is not only able to denoise the images in stages, but also renders the intermediate stages of the noisy images, which provides an additional depth in information while training the networks. Through experiments, we were able to show the method is robust against various levels of noise.

In Sec. 5.4, we proposed a recurent neural network architecture which is similar to the method proposed in Sec. 5.3. The network addresses a limited-photon

setting for imaging with a low-resolution imaging apparatus. The recurrent neural network addresses the denoising task in stages - it first tackles the limited photon imaging problem, followed by the low-resolution imaging apparatus. Through extensive experimentation, we were able to show that the proposed approach is robust against different noisy and compressed targets and was able to improve upon an Autoencoder approach.

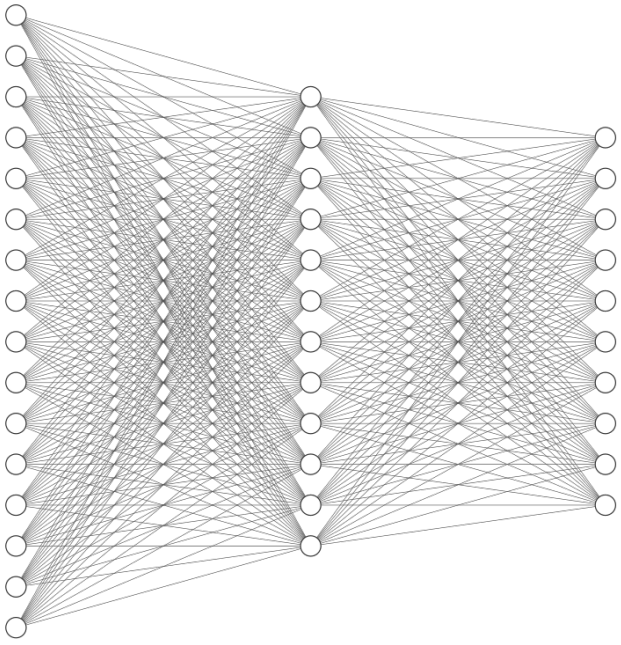


Figure A.1: **Fully Connected Neural Network (FCNN).** Each white circle is considered as a neuron, black lines represent the weights of the neural networks **w**. The left most layer is the input layer, the middle column is the hidden layer and the rightmost layer is the output layer.

Appendix A

Supplemental Material

A.1 Deep learning models

Deep learning encompasses a variety of neural network architectures and models, each tailored for specific tasks and data types. The concept of artificial neural

networks (ANNs) dates back to the 1940s and 1950s, but interest waned in the 1960s. During this period, researcher Stuart Dreyfus (see [148]) explored the idea of designing a these cascaded parameterized architecture, which is now referred to as neural network. These networks had the capability of propagating errors through the layers using computational methods.

A.1.1 Feedforward Neural Networks (FNNs).

Also known as Multi-Layer Perceptrons (MLPs), FNNs are the foundation of deep learning. They consist of an input layer, one or more hidden layers, and an output layer. They are used for tasks like image classification, regression, and more. The network has a cascaded architecture of fully connected hidden layers, which are followed by an activation function. An example of the neural network is presented in Fig. A.1. Given the input dataset in (1.1), the following operations are performed by the network:

$$\mathbf{z}_j = \sum_{i=1}^{d} \mathbf{w}_{ij} \cdot [\mathbf{x}_k]_i + \mathbf{b}_j,$$

where $[\mathbf{x}_j]_i$ is the i^{th} element of the k^{th} observation, $\mathbf{w} \in \mathbb{R}^{l1 \times l2}$ is the weight matrix, represented by the arrows from layer $l1$ going into $l2$, $\mathbf{b} \in \mathbb{R}^{l2}$ is the bias vector for layer $l2$ and $\mathbf{z} \in \mathbb{R}^{l2}$ is the resulting output vector. This operation is then followed by an activation function

$$a_j = \text{activation}(z_j).$$

Popular activation functions include tanh, ReLU, sigmoid etc. The number of hidden layers l can be changed as per the requirements of the task.

MLPs were amongst the first networks in the deep learning space to have done image classification. However, their use became limiting due to dense parameterization schemes and computational constraints. This gave way to a new variant of deep neural networks called Convolutional neural networks.

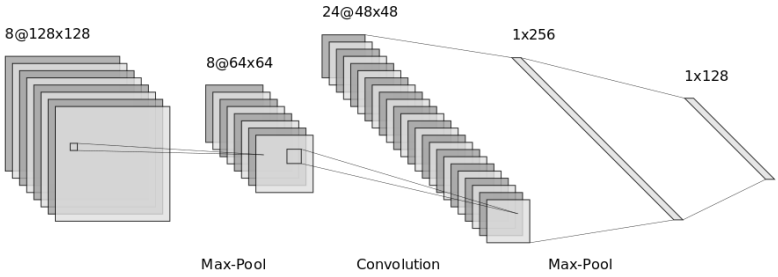


Figure A.2: **Convolutional Neural Network (FCNN).** Each square is an image output after the filter is applied. The box in the middle of the squares is the filter operation being applied.

A.1.2 Convolutional Neural Networks (CNNs).

Designed for image and spatial data, CNNs employ convolutional layers to automatically learn features from data. They excel in tasks like image recognition, object detection, and image segmentation. An example of CNN type architecture is presented in Fig. A.2. Similar to FCNN in terms of construction, the network has a cascaded architecture. However, the spatial information is extracted using the convolutional operators. This operation is given by

$$\begin{bmatrix} \mathbf{x}_{1,1} & \mathbf{x}_{1,2} & \mathbf{x}_{1,3} \\ \mathbf{x}_{2,1} & \mathbf{x}_{2,2} & \mathbf{x}_{2,3} \\ \mathbf{x}_{3,1} & \mathbf{x}_{3,2} & \mathbf{x}_{3,3} \end{bmatrix} * \begin{bmatrix} \mathbf{w}_{1,1} & \mathbf{w}_{1,2} \\ \mathbf{w}_{2,1} & \mathbf{w}_{2,2} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_{1,1} & \mathbf{z}_{1,2} \\ \mathbf{z}_{2,1} & \mathbf{z}_{2,2} \end{bmatrix}.$$

For input $\mathbf{x} \in \mathbb{R}^{w \times h}$, and a weight matrix (also referred to as kernels in the context of convolutional operations) $\mathbf{w} \in \mathbb{R}^{w' \times h'}$, the convolution operator $*$ performs the operation $\mathbf{z}_{i,j} = \sum_m \sum_n \mathbf{x}_{i+m,j+m}.\mathbf{w}_{m,n}$. The double summation over m and n indicates over all positions in the input feature map that the convolution kernel covers, and for each position, multiply the input value by the corresponding kernel

value and accumulate the results to compute the output at position (i, j).

A.1.3 Recurrent Neural Networks (RNNs).

Tailored for sequential data, RNNs have connections that loop back on themselves, allowing them to capture temporal dependencies. They are widely used in natural language processing, speech recognition, and time series analysis. Two most commonly used neural networks are **Long Short-Term Memory (LSTM)** and **Gated Recurrent Unit (GRU) networks**. We will be discussing about LSTM networks below.

Long Short-Term Memory (LSTM) Networks.

A specialized type of RNN, LSTMs address the vanishing gradient problem and are particularly effective for long-range dependencies in sequential data. They find applications in language modeling, speech recognition, and more.

A.1.4 Autoencoders.

Autoencoders are unsupervised models used for dimensionality reduction and feature learning. They consist of an encoder and decoder and are employed in tasks like image denoising, anomaly detection, and data compression.

A.1.5 Transformer Models.

Transformer models, with architectures like BERT and GPT, have revolutionized natural language processing. They use self-attention mechanisms to capture dependencies in text data and are applied in tasks like language understanding, translation, and generation.

A.2 Optimization Techniques and derivation

Gradient descent's history can be traced back to the early 19th century when the French mathematician Augustin-Louis Cauchy first introduced the concept of

gradient. However, it gained widespread recognition in the 20th century, particularly due to its application in numerical analysis, engineering, and machine learning. The popularity of gradient descent in deep learning increased after the advent of **backpropagation** in the context of neural networks.

The key innovation of backpropagation is often attributed to Paul Werbos, who introduced the algorithm in his 1974 Ph.D. dissertation (see [149]). His work laid the foundation for error backpropagation in neural networks. Through backpropagation, users were able to efficiently compute the gradients of the loss function with respect to the network's parameters, enabling the iterative adjustment of weights and biases to minimize the loss.

So why is the gradient information important in minimizing the loss and adjusting the weights ? To answer this, we look at the Taylor series expansion of $\mathcal{L}$ in (1.1). With a slight abuse of notation, the loss function may be rewritten as $\mathcal{L}(\mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{y}}_i; \Theta_i) \equiv \mathcal{L}(\Theta_i)$. For some step $\mathbf{s} \in \mathbb{R}^n$ in the parameter space $\Theta_i \in \mathbb{R}^n$, we can write the Taylor series expansion as

$$\mathcal{L}(\Theta_i + \mathbf{s}) = \mathcal{L}(\Theta_i) + \mathbf{s}^\top \mathbf{g}_i + \frac{1}{2}\mathbf{s}^\top \mathbf{H}_i \mathbf{s} + \mathcal{O}(\xi^3),$$

where $\mathbf{g}_i \in \mathbb{R}^n$ is the first-order derivative (gradient) of $\mathcal{L}$ with respect to Θ $\nabla_\Theta \mathcal{L}$, $\mathbf{H}_i \in \mathbb{R}^{n \times n}$ is the second-order derivative (Hessian) of $\mathcal{L}$ with respect to Θ $\nabla^2_{\Theta\Theta} \mathcal{L}$. The $\mathcal{O}(\xi^3)$ term represents all higher order terms. With some replacement of terms $\nabla^2_{\Theta\Theta} = \nabla^2$ and $\nabla_\Theta = \nabla$, we define Taylor's theorem below.

Theorem A.2.1. *For a continuously differentiable function $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ and some step $\mathbf{s}^n$,*

$$\mathcal{L}(\Theta + \mathbf{s}) = \mathcal{L}(\Theta) + \mathbf{s}^\top \nabla \mathcal{L}(\Theta + t\mathbf{s}),$$

where $t \in [0, 1]$, holds. Moreover, if $\mathcal{L}$ is twice continuously differentiable,

$$\nabla \mathcal{L}(\Theta + \mathbf{s}) = \mathbf{g}^\top \mathbf{s} + \int_0^1 \nabla^2 \mathcal{L}(\Theta + t\mathbf{s}) ds$$

and

$$\mathcal{L}(\Theta + \mathbf{s}) = \mathcal{L}(\Theta) + \mathbf{g}^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top \nabla^2 \mathcal{L}(\Theta + t\mathbf{s})\mathbf{s}, \tag{A.1}$$

where $t \in [0, 1]$.

The iterative strategy for updating the weights of the neural networks, given by

$$\Theta_{i+1} = \Theta_i + \mathbf{s}, \tag{A.2}$$

where $\mathbf{s}$ is the step taken such that $\mathcal{L}(\Theta_i + \mathbf{s}) < \mathcal{L}(\Theta_i)$. In order to do this, we define the first-order and second-order conditions about the behaviour of the function $\mathcal{L}$ at $\Theta + \mathbf{s}$. We make the following mild assumption:

Assumption 3. *We assume $\mathcal{L}$ to be continuous and twice differentiable. This means, $\forall \Theta \in \mathbb{R}^n$, there exists $\mathcal{L}(\Theta) \in \mathbb{R}$, $\nabla\mathcal{L}(\Theta) \in \mathbb{R}^n$ and $\nabla^2\mathcal{L}(\Theta) \in \mathbb{R}^{n\times n}$.*

Theorem A.2.2. *If Θ^* is a local mimizer of $\mathcal{L}$, and if $\mathcal{L}$ is continuously differentiable in an open neighbourhood of Θ^*, then $\nabla\mathcal{L}(\Theta^*) = 0$.*

Theorem A.2.3. *If Θ^* is a local minimizer of $\mathcal{L}$ and assumption 3 holds, then $\nabla\mathcal{L}(\Theta^*) = 0$ and $\nabla^2\mathcal{L}(\Theta^*)$ is positive-semidefinite.*

Theorem A.2.4. *Suppose $\nabla^2\mathcal{L}(\Theta)$ is continuous in the neighborhood of Θ^* and $\nabla\mathcal{L}(\Theta^*) = 0$ and $\nabla^2\mathcal{L}(\Theta^*)$ is positive-definite, then Θ^* is a strict local minimizer of $\mathcal{L}$.*

The proofs of these theorems can be found in [21].

A.2.1 Second-order Methods:

Using theorem A.2.4, if we define $\Theta^* = \Theta + \mathbf{s}$, the expression of $\mathbf{s}$ by applying A.2.3 in the Taylor series expansion given in theorem A.2.1 is given by

$$\mathbf{s} = -[\nabla^2\mathcal{L}(\Theta_i + \mathbf{s})]^{-1}\mathbf{g}_i. \tag{A.3}$$

It can be a daunting process to find $\nabla^2\mathcal{L}(\Theta_i + t\mathbf{s})$, especially when the value of t is not available beforehand. To reduce this difficulty, we substitue $t = 0$ in A.1 and use the qudratic approximation of the function $\mathcal{L}$, which is given by

$$\mathcal{L}(\Theta_i + \mathbf{s}) \approx \mathcal{L}(\Theta_i) + \mathbf{g}_i^\top\mathbf{s} + \frac{1}{2}\mathbf{s}^\top\mathbf{H}_i\mathbf{s}. \tag{A.4}$$

For this approximation, the step $\mathbf{s}$ is given by

$$\mathbf{s} = -[\mathbf{H}_i]^{-1}\mathbf{g}_i. \tag{A.5}$$

This step is most commonly referred to as *Newton's step* or *Newton's direction.* Thus, the iterative step definition of a Newton's method is given by

$$\Theta_{i+1} = \Theta_i - [\mathbf{H}_i]^{-1}\mathbf{g}_i,$$

Based on this step definition in (A.5), we list the potential advantages and disadvantages of using this Hessian information.

Advantages:

- The Newton's method enjoys quadratic convergence. This means, the gradient at the new iterate $\|\nabla\mathcal{L}(\Theta_{i+1})\|_2 \leq 2L\|[\nabla^2\mathcal{L}(\Theta_i)]^{-1}\|\|\nabla\mathcal{L}(\Theta_i)\|$. For more details on this proof, see [21]. Hence, the norms of the gradients converge to 0 quadratically.

- The step exploits curvature information of the function. The curvature information suggests if the function Hessian information is positive-definite or non-positive definite. Hence, the step is able to exploit this curvature information to arrive to the minimizer.

Disadvantages:

- For large-scale non-convex problems like neural-networks, it is computationally difficult to store the Hessian information. Most of the mordern neural networks are of size $n = 1 \times 10^6$ parameters. This would mean, the Hessian matrix would have entries of the order $\mathcal{O}(n^2)$ ($n = 1 \times 10^{12}$). Thus, this quadratic dependence of the Hessian matrix on the model parameters makes storing it intractible.

- The matrix inversion operation can be computationally expensive. The computational complexity of a network with n parameters is of the order $\mathcal{O}(n^3)$. Hence, computing this inverse is also very costly.

- Since $\mathcal{L}$ is highly non-convex and nonlinear, the Hessian can be singular or negative-definite. This means, the inverse of the Hessian may yield highly sub-optimal solutions.

Since the disadvantages outweigh the advantages, researchers resorted to simpler techniques for optimizing neural networks. We will see the foundation for this approach in Sec. A.2.2

A.2.2 First-order Methods:

First-order methods use the gradient information to optimize the objective function (1.1). Also commonly known as Gradient Descent and *Steepest descent*, gradient-based appraches are one of the most common approaches of optimizing neural networks in a deep learning setting. The usage of these methods became widespread after backpropogation was introduced to a deep learning setting (see [150, 151]).

So what is gradient descent ? And how is it derived from (A.4) ?

If we replace the Hessian information with an identity matrix, the update (A.5) is reformed as

$$\Theta_{i+1} = \Theta_i - \eta[\mathbf{I}]^{-1}\mathbf{g}_i,$$

where $\mathbf{I} \in \mathbb{R}^{n\times n}$ is the identity matrix. Thus,

$$\Theta_{i+1} = \Theta_i - \eta\mathbf{g}_i. \tag{A.6}$$

Here η is referred to as the *learning rate* of the optimization scheme.

The update defined in (A.6) is the foundation for most gradient-based optimization techniques. Almost every mordern first-order approach has evolved from (A.6). The gradient descent approach can be viewed as analogous to a line-search approach, where $-\mathbf{g}_i$ yields the direction of steepest descent and η is predetermined.

Now, let's understand the advantages and disadvatages of using this approach.
Advantages:

- $\mathbf{g}_i$ is fairly easy to compute. It is asymptotically equal to one forward-pass through the neural network.
- $\mathbf{g}_i$ is fairly easy to store. It requires only $\mathcal{O}(n)$ memory.
- In practice, first-order approaches always converge to the minimizer **eventually**.

Disadvantages:

- The method suffers slow and linear convergence. Let assumptions 3 hold, and the iterates be defined by (A.6). Given the Hessian matrix at the minimizer $\mathcal{L}(\Theta^*)$ is positive definite, $\lambda_1 \leq \lambda_2 \leq \lambda_3 \cdots \lambda_n$, where λ_j for $j \in [0, n]$ are the eigenvalues of the Hessian matrix $\mathbf{H}_i$, we define

  $$r \in \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}, 1\right).$$

  For sufficiently large i, $\mathcal{L}(\Theta_{i+1}) - \mathcal{L}(\Theta^*) \leq r^2[\mathcal{L}(\Theta_i) - \mathcal{L}(\Theta^*)]$
- The gradient-based approaches are always susceptible to saddle-points. Saddle points are not minimizers (either local or global) of a function. These point are just locations on the contour of the manifold where the norm of gradient $\|\mathbf{g}_i\|$ is zero or approaching zero.

This motivates the question - Is there a way to combine the advantages of a first-order approach and a second-order approach ?

In the next section, we will be discussing Quasi-Newton approaches

Bibliography

[1] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[2] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle. Deep learning for computational biology. *Molecular systems biology*, 12(7):878, 2016.

[3] D.C. Cireşan, A. Giusti, L.M. Gambardella, and J. Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *Intl. Conf. on Medical Image Computing and Computer-Assisted Intervention*, pages 411–418. Springer, 2013.

[4] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[5] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[6] L. Bottou and Y LeCun. Large scale online learning. In *Advances in Neural Information Processing Systems*, pages 217–224, 2004.

[7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[8] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[9] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[10] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[11] M. Zinkevich, M. Weimer, L. Li, and A.J. Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 2595–2603, 2010.

[12] J. Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, volume 27, pages 735–742, 2010.

[13] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

[14] P.J. Werbos. Backpropagation: Past and future. In *Proceedings of the Second International Conference on Neural Network*, volume 1, pages 343–353. IEEE, 1988.

[15] M.F. Møller. Exact calculation of the product of the hessian matrix of feed-forward network error functions and a vector in $O(n)$ time. *DAIMI Report Series*, (432), 1993.

[16] F.J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229, 1987.

[17] L.B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial neural networks: concept learning*, pages 102–111. IEEE Press, 1990.

[18] J. M. Ortega and W. C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*. SIAM, 2000.

[19] N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

[20] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the 21st International Conference on Machine Learning*, page 116. ACM, 2004.

[21] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[22] A.R. Conn, N.I.M. Gould, and P.L. Toint. *Trust-Region Methods*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. ISBN 0-89871-460-5.

[23] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[24] Yann LeCun, Corinna Cortes, Chris Burges, et al. Mnist handwritten digit database, 2010.

[25] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate, 2019.

[26] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[27] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] Nicholas I. M. Gould, Stefano Lucidi, Massimo Roma, and Philippe L. Toint. Exploiting negative curvature directions in linesearch methods for unconstrained optimization. *Optimization Methods and Software*, 14(1-2):75–98, 2000. doi: 10.1080/10556780008805794.

[30] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, Aug 1989. ISSN 1436-4646. doi: 10.1007/BF01589116. URL https://doi.org/10.1007/BF01589116.

[31] Jennifer B. Erway, Joshua D. Griffin, Roummel F. Marcia, and Riadh Omheni. Trust-region algorithms for training responses: machine learning methods using indefinite hessian approximations. *Optimization Methods and Software*, 35:460 – 487, 2020.

[32] H Fayez Khalfan, Richard H Byrd, and Robert B Schnabel. A theoretical and experimental study of the symmetric rank-one update. *SIAM Journal on Optimization*, 3(1):1–24, 1993.

[33] Andrew R. Conn, Nicholas. I. M. Gould, and Philippe. L. Toint. Convergence of quasi-newton matrices generated by the symmetric rank one update. *Mathematical Programming*, 50(1):177–195, Mar 1991. ISSN 1436-4646. doi: 10.1007/BF01594934. URL https://doi.org/10.1007/BF01594934.

[34] Richard. H. Byrd, Jorge. Nocedal, and Robert. B. Schnabel. Representations of quasi-Newton matrices and their use in limited-memory methods. *Math. Program.*, 63:129–156, 1994.

[35] Jennifer B. Erway and Roummel F. Marcia. On efficiently computing the eigenvalues of limited-memory quasi-newton matrices. *SIAM Journal on Matrix Analysis and Applications*, 36(3):1338–1359, 2015. doi: 10.1137/140997737.

[36] Andreas Griewank. The modification of Newtons method for unconstrained optimization by bounding cubic terms. Technical report, Technical report NA/12, 1981.

[37] Coralia Cartis, Nicholas IM Gould, and Philippe L Toint. Adaptive cubic regularisation methods for unconstrained optimization. part i: motivation,

convergence and numerical results. *Mathematical Programming*, 127(2):245–295, 2011.

[38] Yurii Nesterov and Boris T Polyak. Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

[39] Oleg Burdakov, Lujin Gong, Spartak Zikrin, and Ya-xiang Yuan. On efficiently combining limited-memory and trust-region techniques. *Mathematical Programming Computation*, 9(1):101–134, 2017.

[40] Hande Y. Benson and David F. Shanno. Cubic regularization in symmetric rank-1 quasi-newton methods. *Mathematical Programming Computation*, 10 (4):457–486, Dec 2018. ISSN 1867-2957. doi: 10.1007/s12532-018-0136-7. URL `https://doi.org/10.1007/s12532-018-0136-7`.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[42] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, An-

tônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[43] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585 (7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[44] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[45] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[46] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[47] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017.

[48] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330, June 1993.

[49] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization, 2014.

[50] Xiao Wang, Shiqian Ma, Donald Goldfarb, and Wei Liu. Stochastic quasi-newton methods for nonconvex stochastic optimization. *SIAM Journal on Optimization*, 27(2):927–956, 2017.

[51] Buse Melis Ozyildirim and Mariam Kiran. Do optimization methods in deep learning applications matter?, 2020.

[52] Robin M Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley-benchmarking deep learning optimizers. In *International Conference on Machine Learning*, pages 9367–9376. PMLR, 2021.

[53] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *arXiv preprint arXiv:1902.09843*, 2019.

[54] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[55] Donald Goldfarb, Yi Ren, and Achraf Bahamou. Practical quasi-newton methods for training deep neural networks. *Advances in Neural Information Processing Systems*, 33:2386–2396, 2020.

[56] Yurii Evgen'evich Nesterov. A method of solving a convex programming problem with convergence rate o\bigl(k^2\bigr). In *Doklady Akademii Nauk*, volume 269, pages 543–547. Russian Academy of Sciences, 1983.

[57] Aditya Ranganath, Omar DeGuchy, Mukesh Singhal, and Roummel F Marcia. Second-order trust-region optimization for data-limited inference. In *2021 29th European Signal Processing Conference (EUSIPCO)*, pages 2059–2063. IEEE, 2021.

[58] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[59] Albert S Berahas, Majid Jahani, Peter Richtárik, and Martin Takáč. Quasi-newton methods for machine learning: forget the past, just sample. *Optimization Methods and Software*, 37(5):1668–1704, 2022.

[60] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pages 928–936, 2003.

[61] Gene H Golub and Charles F Van Loan. *Matrix computations*. The Johns Hopkins University press, 3 edition, 2013.

[62] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[63] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[64] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[65] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. URL `http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf`.

[66] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL `https://aclanthology.org/J93-2004`.

[67] Xi-Ren Cao and Ruey-wen Liu. General approach to blind source separation. *IEEE Transactions on signal Processing*, 44(3):562–571, 1996.

[68] Ganesh R Naik, Wenwu Wang, et al. Blind source separation. *Berlin: Springer*, 10:978–3, 2014.

[69] Kamran Rahbar and James P Reilly. A frequency domain method for blind source separation of convolutive audio mixtures. *IEEE Transactions on speech and audio processing*, 13(5):832–844, 2005.

[70] Antoine Liutkus, Jean-Louis Durrieu, Laurent Daudet, and Gaël Richard. An overview of informed audio source separation. In *2013 14th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, pages 1–4. IEEE, 2013.

[71] Roummel F. Marcia, Changsoon Kim, Cihat Eldeniz, Jungsang Kim, David J. Brady, and Rebecca M. Willett. Superimposed video disambiguation for increased field of view. *Opt. Express*, 16(21):16352–16363, 2008.

[72] Ines Meganem, Yannick Deville, Shahram Hosseini, Philippe Deliot, and Xavier Briottet. Linear-quadratic blind source separation using nmf to unmix urban hyperspectral images. *IEEE Transactions on Signal Processing*, 62(7): 1822–1833, 2014.

[73] Xianchuan Yu, Dan Hu, and Jindong Xu. *Blind source separation: theory and applications*. John Wiley & Sons, 2013.

[74] Omar DeGuchy, Alex Ho, and Roummel F Marcia. Image disambiguation with deep neural networks. In *Applications of Machine Learning*, volume 11139, pages 68–74. SPIE, 2019.

[75] Aditya Ranganath, Omar DeGuchy, Mukesh Singhal, and Roummel F. Marcia. Multi-stage gaussian noise reduction with recurrent neural networks. In *2021 55th Asilomar Conference on Signals, Systems, and Computers*, pages 135–139, 2021. doi: 10.1109/IEEECONF53345.2021.9723266.

[76] Aditya Ranganath, Omar DeGuchy, Fabian Santiago, Mukesh Singhal, and Roummel Marcia. Recurrent nerual imaging: An evolutionary approach for mixed possion-gaussian image denoising. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 484–489, 2022. doi: 10.1109/ICMLA55696.2022.00078.

[77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[78] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[79] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[80] Dan Zhang and Fangfang Zhou. Self-supervised image denoising for real-world images with context-aware transformer. *IEEE Access*, 11:14340–14349, 2023.

[81] Shaoyan Pan, Tonghe Wang, Richard LJ Qiu, Marian Axente, Chih-Wei Chang, Junbo Peng, Ashish B Patel, Joseph Shelton, Sagar A Patel, Justin Roper, et al. 2d medical image synthesis using transformer-based denoising diffusion probabilistic model. *Physics in Medicine & Biology*, 68(10):105004, 2023.

[82] Roshan M Rao, Jason Liu, Robert Verkuil, Joshua Meier, John Canny, Pieter Abbeel, Tom Sercu, and Alexander Rives. Msa transformer. In *International Conference on Machine Learning*, pages 8844–8856. PMLR, 2021.

[83] Usman Naseem, Imran Razzak, Katarzyna Musial, and Muhammad Imran. Transformer based deep intelligent contextual embedding for twitter sentiment analysis. *Future Generation Computer Systems*, 113:58–69, 2020.

[84] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit

Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[85] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013.

[86] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning, 2016.

[87] Siddhant Bhambri, Sumanyu Muku, Avinash Tulasi, and Arun Balaji Buduru. A survey of black-box adversarial attacks on computer vision models, 2019.

[88] Anish Athalye and Nicholas Carlini. On the robustness of the cvpr 2018 white-box adversarial example defenses. *arXiv preprint arXiv:1804.03286*, 2018.

[89] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[90] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[91] Ruiqi Gao, Tianle Cai, Haochuan Li, Cho-Jui Hsieh, Liwei Wang, and Jason D Lee. Convergence of adversarial training in overparametrized neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[92] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C Duchi, and

Percy S Liang. Unlabeled data improves adversarial robustness. *Advances in Neural Information Processing Systems*, 32, 2019.

[93] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. Efficient defenses against adversarial attacks. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 39–49, 2017.

[94] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. *Advances in Neural Information Processing Systems*, 32, 2019.

[95] Tianyu Pang, Xiao Yang, Yinpeng Dong, Kun Xu, Jun Zhu, and Hang Su. Boosting adversarial training with hypersphere embedding. *Advances in Neural Information Processing Systems*, 33:7779–7792, 2020.

[96] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. *Advances in Neural Information Processing Systems*, 33:2958–2969, 2020.

[97] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization (2017). *arXiv preprint arXiv:1711.01991*, 2018.

[98] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

[99] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 135–147, 2017.

[100] Ajil Jalal, Andrew Ilyas, Constantinos Daskalakis, and Alexandros G Dimakis. The robust manifold defense: Adversarial training using generative models. *arXiv preprint arXiv:1712.09196*, 2017.

[101] Kevin J Liang, Chunyuan Li, Guoyin Wang, and Lawrence Carin. Generative adversarial network training is a continual learning problem. *arXiv preprint arXiv:1811.11083*, 2018.

[102] Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. Adversarial and clean data are not twins. *arXiv preprint arXiv:1704.04960*, 2017.

[103] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[104] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems*, 31, 2018.

[105] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E Houle, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*, 2018.

[106] Jayaram Raghuram, Varun Chandrasekaran, Somesh Jha, and Suman Banerjee. A general framework for detecting anomalous inputs to dnn classifiers. In *International Conference on Machine Learning*, pages 8764–8775. PMLR, 2021.

[107] Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. Discrete cosine transform. *IEEE transactions on Computers*, 100(1):90–93, 1974.

[108] J. Makhoul. A fast cosine transform in one and two dimensions. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):27–34, 1980. doi: 10.1109/TASSP.1980.1163351.

[109] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[110] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press.

[111] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[112] J. Portilla, V. Strela, M. J Wainwright, and E. P. Simoncelli. Image denoising using scale mixtures of gaussians in the wavelet domain. *IEEE Transactions on Image Processing*, 12(11):1338–1351, 2003.

[113] A. M. Wink and J. BTM Roerdink. Denoising functional mr images: a comparison of wavelet denoising and gaussian smoothing. *IEEE Transactions on Medical Imaging*, 23(3):374–387, 2004.

[114] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, 2007.

[115] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Non-local sparse models for image restoration. In *2009 IEEE 12th International Conference on Computer Vision*, pages 2272–2279. IEEE, 2009.

[116] S. Gu, L. Zhang, W. Zuo, and X. Feng. Weighted nuclear norm minimization with application to image denoising. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2862–2869, 2014.

[117] V. Jain and S. Seung. Natural image denoising with convolutional networks. In *Advances in Neural Information Processing Systems*, pages 769–776, 2009.

[118] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

[119] H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2392–2399. IEEE, 2012.

[120] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep image prior. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454, 2018.

[121] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[122] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[123] X. Mao, C. Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In *Advances in Neural Information Processing Systems*, pages 2802–2810, 2016.

[124] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

[125] A. Mousavi and R. G. Baraniuk. Learning to invert: Signal recovery via deep convolutional networks. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2272–2276. IEEE, 2017.

[126] O. DeGuchy, F. Santiago, M. Banuelos, and R. F. Marcia. Deep neural networks for low-resolution photon-limited imaging. In *2019 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3247–3251. IEEE, 2019.

[127] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[128] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[129] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[130] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

[131] P. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *31st International Conference on Machine Learning*, 2014.

[132] S. Lai, Liheng X., K. Liu, and J. Zhao. Recurrent convolutional neural networks for text classification. In *29th AAAI Conference on Artificial Intelligence*, 2015.

[133] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[134] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[135] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. IEEE, 2003.

[136] A. Buades, B. Coll, and J. M. Morel. A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation*, 4(2):490–530, 2005.

[137] P. Cattin. Image restoration: Introduction to signal and image processing. *MIAC, University of Basel. Retrieved*, 11:93, 2013.

[138] D. L. Snyder and M. I. Miller. *Random point processes in time and space*. Springer Science & Business Media, 2012.

[139] A. Jezierska, H. Talbot, C. Chaux, J.-C. Pesquet, and G. Engler. Poisson-Gaussian noise parameter estimation in fluorescence microscopy imaging. In *2012 9th IEEE International Symposium on Biomedical Imaging (ISBI)*, pages 1663–1666. IEEE, 2012.

[140] D.-H. Trinh, M. Luong, J.-M. Rocchisani, C.-D. Pham, N. Linh-Trung, T. Q. Nguyen, et al. An effective example-based learning method for denoising of medical images corrupted by heavy Gaussian noise and Poisson noise. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 823–827. IEEE, 2014.

[141] B. Bajić, J. Lindblad, and N. Sladoje. Blind restoration of images degraded with mixed Poisson-Gaussian noise with application in transmission electron microscopy. In *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, pages 123–127. IEEE, 2016.

[142] F. Luisier, T. Blu, and M. Unser. Image denoising in mixed Poisson–Gaussian noise. *IEEE Transactions on Image Processing*, 20(3):696–708, 2010.

[143] L. Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops*, pages 241–246. IEEE, 2016.

[144] J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/6cdd60ea0045eb7a6ec44c54d29ed402-Paper.pdf`.

[145] R. Rajeev, J. A. Samath, and N. K. Karthikeyan. An intelligent recurrent neural network with long short-term memory (lstm) based batch normalization for medical image denoising. *Journal of Medical Systems*, 43(8): 234, Jun 2019. ISSN 1573-689X. doi: 10.1007/s10916-019-1371-9. URL `https://doi.org/10.1007/s10916-019-1371-9`.

[146] Ding Liu, Bihan Wen, Yuchen Fan, Chen Change Loy, and Thomas S Huang. Non-local recurrent network for image restoration. *Advances in neural information processing systems*, 31, 2018.

[147] A. Graves, A.-R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE, 2013.

[148] Stuart E Dreyfus. Games, strategic behavior, and operations research. *Management science*, 8(4):386–415, 1962.

[149] Paul John Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*, volume 1. John Wiley & Sons, 1994.

[150] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0. URL `https://doi.org/10.1038/323533a0`.

[151] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.