Optimization Algorithms for Machine Learning Designed for Parallel and Distributed Environments

Seyedalireza Yektamaram
Lehigh University

Optimization Algorithms for Machine Learning Designed for Parallel and Distributed Environments

by

Seyedalireza Yektamaram

A Dissertation
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Doctor of Philosophy
in
Industrial and Systems Engineering

Lehigh University
January 2018

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Seyedalireza Yektamaram
Optimization Algorithms for Machine Learning Designed for Parallel and Distributed Environments

Date

Dr. Katya Scheinberg, Dissertation Director, Chair

Accepted Date

Committee Members

Dr. Katya Scheinberg, Committee Chair

Dr. Ted Ralphs

Dr. Martin Takáč

Dr. Joshua D. Griffin

Acknowledgments

I would like to express my gratitude towards many individuals who have touched my life in many aspects through their support, mentorship, teachings and their friendship.

My gratitude goes to my parents Ali and Simin for their constant support, guidance, and selfless sacrifices throughout all different steps of my life.

I'm grateful to my wife Negin, who have been passionately supportive of me, and whose love and support kept me moving forward throughout ups and downs on this path.

My great appreciation to my advisor Dr. Katya Scheinberg, for all of her selfless support during my studies. She is a great advisor and a patient mentor, who through her guidance and encouragements from the beginning of my Ph.D. inspired and will continue to inspire me to develop skills for research and life, and always pushing to reach higher.

I was fortunate to have Dr. Martin Takac, Dr. Ted Ralphs and Dr. Joshua Griffin in my commmitte, whose insights discussions and comments were invaluable to completing this dissertation. This study, would not have been possible without their support. Also I'm thankful to Dr. Wenwen Zhou along with Dr. Griffin, for their time and helpful discussions. I'm thankful to OPTML members including Dr. Takac, Dr. Curtis, Xiaocheng Tang and COR@L lab folks, Julio Goez and Aykut Bulut for their help and discussions.

I'd like to thank Dr. Radhika Kulkarni, Dr. Manoj Chari, Dr. Yan Xu, Dr. Xiangqian Hu at SAS Institute for their support during my internship and career. I'd like to also thank Dr. Nasser Salmasi for his friendship, and guidance towards

higher education.
Last but not least, I'd like to thank my brother Mohammadreza for his constant brotherly support.
A great thanks to all my friends, teachers, and professors at Lehigh, Sharif, Azad Tehran-North Universities, and Tehran highschool whose names I've not included here, and whom I've been blessed to know and interact with during my studies.

Contents

List of Tables

List of Figures

Abstract

This thesis proposes several optimization methods that utilize parallel algorithms for large-scale machine learning problems. The overall theme is network-based machine learning algorithms; in particular, we consider two machine learning models: graphical models and neural networks. Graphical models are methods categorized under unsupervised machine learning, aiming at recovering conditional dependencies among random variables from observed samples of a multivariable distribution. Neural networks, on the other hand, are methods that learn an implicit approximation to underlying true nonlinear functions based on sample data and utilize that information to generalize to validation data. The goal of finding the best methods relies on an optimization problem tasked with training such models. Improvements in current methods of solving the optimization problem for graphical models are obtained by parallelization and the use of a new update and a new step-size selection rule in the coordinate descent algorithms designed for large-scale problems. For training deep neural networks, we consider the second-order optimization algorithms within trust-region-like optimization frameworks. Deep networks are represented using large-scale vectors of weights and are trained based on very large datasets. Hence, obtaining second-order information is very expensive for these networks. In this thesis, we undertake an extensive exploration of algorithms that use a small number of curvature evaluations and are hence faster than other existing methods.

Chapter 1

Introduction

The overwhelming amount of data and its continuous accumulation through online tools has significantly changed the classical approaches to data processing and raised a need for scalable and efficient algorithms with an emphasis on high-performance computing aimed at learning hidden patterns from such data. Topics dealing with such knowledge extraction involve various domains such as computer science, statistics, optimization and more, where collectively development of such tools are referred to as Machine Learning (ML). Fitting machine learning models to a data set by exploiting the hidden patterns involves two major steps: *training* and *validation*. Training involves exploiting sampled observations from some process, and training a model that best fits the data. In most cases of ML problems, this step involves mathematical optimization. Once a model is trained, the validation step verifies the performance of the trained model using a fraction of the data set that has been hidden from the training phase of the model (validation data).

An impetus for the rise of various developments aside from research is the impact of machine learning on businesses. To devise accurate production and marketing strategies, businesses aim to recover and interpret the meaningful patterns from data. This continuous inference of meaningful relationships and patterns from the accumulating data will raise the demand for higher computational power and faster processing times. Despite technological improvements and availability of better technologies for data storage and computing, the need for better algorithms still persists. The result is a high demand for faster algorithms in machine learning. In addition to improvements on computational efficiency, technologies such as distributed computing and GPUs allow classical algorithms to undertake problems of much larger size.

In this thesis, we analyze two prominent problems from ML: *Sparse Inverse Covariance Selection* and *Deep Neural Networks*. We consider methods in optimizing the training step that can undertake large-scale problems. However, first we need to define general framework of the problems, in order to pose the aforementioned problems in a unified context.

In the next section, we present the general machine learning problem definition to get better acquainted with the problems tackled in ML. After defining the general tasks required to fit a machine learning model, we outline the connections to optimization and discuss also how ML problems would undertake optimization methods to find better models. Section 1.2 presents a brief introduction to problems considered in this thesis, by posing them each in the general framework defined in the next section. After formalizing the problem introduction, contributions to each problem is discussed.

1.1 Definitions

Given a dataset of observations, there are different measures or properties for each observation called *features*. For example, in a medical study, each observation may come from different patients and features can be the different measurements obtained by health monitoring devices.

Under *supervised learning* of ML applications, it is generally assumed that there exists an underlying function $\phi : X \to Y$ mapping the features $X = \{x \mid x \in \mathbb{R}^{d_x}\}$ to the observed response or target information $Y = \{y \mid y \in \mathbb{R}^{d_y}\}$ for each observation.

In most practical cases, the closed-form representation of function ϕ may not exist due to various factors such as noise, stochastic behavior, among others. Therefore, a modeling function $f(x)$ is chosen from a hypothesis set $\mathcal{H}_\theta = \{h \mid h : X \to$

$Y\}$, that is assumed to closely approximate the underlying function. The family $\mathcal{H}_\theta$, is parameterized by θ, where a model function $f(x)$ is a realization of specific parameter θ. The task is to find the hypothesis function $h \in \mathcal{H}$ that best approximates the function $\phi(x)$. This task is fulfilled by finding parameters θ, determining the best model in family $\mathcal{H}_\theta$ with respect to certain measure discussed below. For simplicity we drop θ from $\mathcal{H}$ notation as it is encapsulated in the definition of the family.

As mentioned above, each family of hypothesis functions $\mathcal{H}$ has their specific set of parameters θ to be learned; for example, in a simple linear regression model, the hypothesis set involves parameters $\theta = \{(w, b) \in \mathbb{R}^{d_x} \times \mathbb{R}\}$ to approximate $\phi(x)$ using an affine function as $\hat{y} := f(x) = w^T x + b$. Generally the trained function $f(x)$ with the parameters $\bar{\theta}$ is referred to as the ML *model* for learning the true behavior. Based on the true nature of the phenomena involved, a model (from the set of hypothesis functions) should be chosen that best capture the details of underlying function. Note that there is a trade-off between the generality/complexity of the class of hypothesis functions (and hence the difficulty of the problem of choosing the "best" model) and the accuracy of the resulting approximation.

Accuracy performance of these models are measured generally by a *Loss* function represented by $L_{\mathcal{H}}(f, x)$. The loss function measures how the output $\hat{y}$ of model $f(x)$, corresponding to specific θ, closely approximates the true observation y. Therefore, minimizing the loss acts as a proxy for our model to learn mimicking the observed data (for now we are not concerned with issues regarding over-fitting). In the simple regression example above, the sum of squared errors can be considered a loss function in fitting the affine function corresponding to parameters $\theta = (w, b)$.

$$L(f(w, b)) = \sum_i \|y_i - \hat{y}_i\|_2^2 = \sum_i \|y_i - (w^T x_i + b)\|_2^2.$$

As hinted above, this procedure summarizes the tasks under supervised learning problems.

In contrast to supervised learning, the *unsupervised learning* problems are not concerned with a target y_i for observation x_i — their focus is to learn specific patterns from the observed data X. Examples of the types of pattern identification that might be undertaken is the clustering data points into groups according to some measure of similarity or the reduction of noise from a signal; such tasks are generally considered more sophisticated compared to supervised learning tasks. Despite the lack of target information, the concept of loss function still applies in this setting, based on the problem and the learning task. For example, in signal denoising, we may be interested in a model that reduces the noise by projecting a signal x onto another low-dimensional space P and again lifts the projected signal back to original space, retrieving the denoised signal (see Hinton and Salakhutdinov [2006] and Ng [2011]). One measure for the performance of the denoising model is to minimize the loss function that computes the divergence of retrieved signal from original as $L(f, x) = \sum_i \|x_i - \hat{x}_i\|_2$.

The problem of choosing the "best" function from a given set of hypothesis functions for a given machine learning problem is fundamentally an optimization problem in which we want to choose the function minimizing the loss function $(L_{\mathcal{H}}(f, x))$, which serves as the objective function. From optimization point of view, minimizing the loss corresponds to finding a function $f(x)$ with least error. However, $f(x)$ is a realization of the functions in $\mathcal{H}_\theta$ and therefore is parameterized by θ. Therefore, θ is the set of desired decision variables and minimizing the objective function corresponds to finding minimizer solution θ^*. Interpreting this concept in the ML world, we are learning model f by fitting parameters θ^* to mimic the real-world observation more closely.

In this thesis, we therefore approach machine learning problems from an optimization point of view, tackling different strategies to better optimize the loss functions and consequently extract better models.

1.2 Summary of Problems

In this thesis, we develop several efficient algorithms, devised for optimization in large-scale ML applications, with emphasis on parallel and distributed environments. The first two chapters are concerned with an unsupervised learning problem known as sparse inverse covariance selection. The sparse inverse covariance selection problem is a method of finding hidden patterns and dependencies among features. In particular, assuming that the vector of features is distributed according to a multivariate Gaussian distribution, the goal is to identify conditional dependency patterns among the features. The sparse inverse covariance method is usually applied in the cases where the number of features by far exceeds the number of observations; for example in genomic studies, for example, it is often the case is that the number of features (genes) is much higher than the number of observations (patients). To handle this issue, additional structural conditions are imposed on the problem to be able to extract meaningful interpretations. In the inverse covariance matrix case, since conditional dependency patterns are inherently represented using a graph structure, (unknown) sparsity of the graph is the special structure that is assumed or imposed. In this setting our machine learning model for finding the conditional dependencies is f_Σ, set of multivariate Gaussian distributions parameterized by inverse covariance matrix $\theta := \Sigma$. The loss defined for training these models stems from estimation theory by optimizing the function of maximum likelihood estimator defined as

$$\max_{\Sigma} L(f) = \log\det\Sigma - \langle\Sigma, S\rangle - \lambda\|\Sigma\|_1.$$

Finding solution Σ^* maximizing this loss function is the objective of corresponding optimization problem; We will discuss this loss function in detail in chapter 2. Specifically, given sample observations, one seeks to find the underlying sparse graph corresponding to the conditional dependence of random variables. Both chapters 2 and 3 study this problem using different parallel coordinate descent algorithms, suitable for different settings. This is achieved by recovering a sparse matrix that approximates the inverse covariance matrix of the Gaussian distribution. This results in an optimization problem over the space of inverse covariance matrices, with data coming from the sample covariance matrix obtained from the data. As the number of features grows large (as it often does), the matrix size grows large as well and the optimization problem becomes very difficult to solve. In addition, a single computational node is not able to store the whole matrix. Therefore, in Chapters 2 and 3, we propose two distributed variants of coordinate descent optimization methods in order to be able to solve large-scale cases. Chapter 2 examines an existing greedy coordinate descent algorithm known as SINCO (Scheinberg and Rish [2010]). In this method, one element of the symmetric variable matrix is updated at each step, in the way that improves the objective function, maintains sparsity, and preserves positive-definiteness of the matrix. This last condition often results in short steps. Hence, in Chapter 2, we improve upon this method by introducing a novel greedy update, called SINCO2D, which can modify three variables at each step, while still achieving the same goals as the original SINCO. This results in larger steps and faster convergence in practice. We then introduce the parallel/distributed version of our algorithm. There are various ways in which a large matrix can be stored on multiple nodes. In Chapter 2, we discuss two such methods in detail and carefully compare their advantages. We conclude that block-cyclic distribution is the more efficient storage method for the SINCO and SINCO2D methods, see Chapter 2 for detail.

In Chapter 3, we develop a parallel version of a block-coordinate descent algorithm for sparse inverse covariance selection, called *Glasso* originally developed by Friedman et al. [2008]. In this approach a whole row (and the corresponding column) of the variable matrix is updated at each step, while improving the objective function, encouraging sparsity and maintaining the positive-semidefinite property. Each such step is obtained by solving a well-known Lasso problem (Tibshirani [1996]), which is, in turn, solved by a coordinate descent approach. In Chapter 3, we propose a method of distributing the matrix so that each Lasso subproblem can be efficiently solved in parallel, to reduce the overall solution time. Due to this new data distribution, we modify the original coordinate descent for Lasso problem. In addition, we study various step size strategy proposed by Richtárik and Takáč [2016] in the specific application to our setting.

The last two chapters focus on efficient optimization methods for two popular machine learning models, namely deep classifier networks and auto encoders. These models present very challenging, very large-scaled and often ill-conditioned optimization problems. In this setting our ML model is a function described implicitly by a network. The loss function will measure the similarity of network's output to real world observation, based on different measure such as euclidean distance or cross-entropy functions which are explored in chapter 4. Most practical optimization approaches for these models, are different variants of the gradient descent method. Hence, they do not exploit curvatures of the objective function and behave poorly in the presence of ill-conditioning. Study of higher-order algorithms have been limited, since getting second-order curvature information can be costly and therefore algorithms would be competitive only if they keep the number of curvature evaluations to a minimum. However, due to the growing use of parallel and distributed environments, these methods have recently gained popularity. Another difficulty for using second-order information arises from the inherent nonconvexity of the problem, which implies that simple approaches such as Newton method do not apply directly. To handle this, two approaches exist in the literature - convexifying the objective by modifying its Hessian (Dauphin et al. [2014], Martens [2010]) or additional trust-region constraint which restrains computing the Newton direction to a specific ball in the $n-$dimensional Euclidean space (Gould et al. [1999], Steihaug [1983]). In Chapters 4 and 5, we choose to focus on the trust-region strategies but in a nonstandard way. In particular, due to the excessive scale of the problems, solving the trust-region subproblem directly is impossible. Hence, approximate methods that involve only Hessian-vector products become necessary. In both chapters, different variants of the well-known conjugate gradient method (Nocedal and Wright [2006]) are explored to solve the subproblem. In Chapter 4, we study an existing algorithm called Modified Conjugate Gradient (MCG) method as explored by Zhou et al. [2017]. Conjugate gradient method minimizes a quadratic model, by iteratively generating subspaces called Krylov subspaces, while each step requiring information in the form of Hessian-vector products.

Since such computations are very expensive for the considered ML applications, we devise a novel warm-starting strategy for the trust-region subproblem solver in order to minimize the total number of Hessian-vector evaluations. This enhancement adapts a nonconvex conjugate gradient method to scale to problems arising in deep learning. We perform a careful comparison of the proposed method with other existing methods that use curvature information. We show that our strategy of using trust-region methodology outperforms convexifying the objective in the quality of solution it obtains. On the other hand, some less expensive quasi-Newton methods, such as L-BFGS have cheaper iterations, which may prove useful in the context of deep learning (as discussed by Ngiam et al. [2011]). Therefore, we then explore hybrid algorithms for training deep learning networks that use cheap iterations for the initial training and then use the MCG method to improve the solution quality. The resulting hybrid methods decrease the overall number of Hessian-vector products significantly and present an improvement upon the state-of-the-art results.

The last chapter contains development of another variant of a Krylov subspace method, called *eigCGTR* algorithm, devised specifically for nonconvex settings. Algorithms that tackle nonconvex quadratic models often have to store a significant number of vectors in order to capture the Krylov subspace and solve the minimization problem in this reduced subspace. However, in deep learning algorithms the sizes of these vectors are large so that it may not be feasible to store thousands of vectors. To resolve this issue, we combine a trust-region type algorithm with a successive subspace minimization technique (introduced by Hager [2001]), which only requires book-keeping a handful of vectors but exploits the direction of the most negative curvature of the objective function for minimization. Obtaining the negative curvature in exact form is also prohibitive in our setting; hence, we study usage of the existing eigCG algorithm of Stathopoulos and Orginos [2010] which can yield approximate highest negative curvature direction relatively cheaply. Following these approximate directions in the earlier iterations of our trust-region based algorithm, results in significantly better quality solutions compared to other methods that work with convex models such as modified Hessian or L-BFGS.

This introduction summarizes the brief definition of the different ML problems addressed in this thesis, as well as, their relationship to traditional optimization methods by presenting a unifying framework that connects both worlds. Although we briefly presented each problem and the devised algorithms, each section requires a detailed overview of the materials as well as their connection to the literature. Therefore, each following chapter contains its own introduction that goes into more detail of the existing literature and methods developed in the thesis.

Chapter 2

Inverse Covariance Selection: Parallel Coordinate Descent

2.1 Introduction

In this chapter we study a particular setting of unsupervised learning, aimed at discovering meaningful relationships among features specific to the observed data. In particular, we focus on Graphical Models (GM) as a subset of these categories, that model conditional dependencies among multiple random variables that are governed by a multi-variate Gaussian distribution, and visualize such dependencies using graphical representations; these models are one of the prominent tools for pattern discovery among problem features that provide insight into the structure of dependency in the feature space.

One of the challenging problems in the field of Graphical Models is Sparse Recovery as discussed by Sra et al. [2012]. In this area of study, given few samples ($x_i \in \mathbb{R}^p, i = 1, \ldots, n$), the aim is to find a sparse estimator of parameters corresponding to the underlying random variable. When the number of observations is less than the number of features ($n \ll p$), one needs to identify the most significant features. Therefore, researchers often try to minimize a convex loss function penalized with a ℓ_1 norm penalty in order to induce sparsity to the optimal estimator. In this study, we focus on the problem of Sparse Inverse Covariance Selection (SICS).

For this problem given sample observations from a family of random variables — that jointly follow a multivariate Gaussian distribution — the task is to infer marginal conditional dependencies of the variables. This problem corresponds to finding edges in the graph representation of random variables, described in section 2.1. The random variables are observed by the values of the features of the data vectors.

There has been numerous studies of this problem (Banerjee et al. [2008], Friedman et al. [2008], Hsieh et al. [2012]) and many algorithms have been proposed (Hsieh et al. [2011, 2013], Kambadur and Lozano [2013], Scheinberg and Rish [2010], Scheinberg et al. [2010]) from different viewpoints and settings. For a detailed review of this problem and also the discussion of the original method considered in this thesis we refer to the "Optimization Methods for Sparse Inverse Covariance Selection Problem“ chapter by Scheinberg and Ma in the book by Sra et al. [2012].

As the number of features grows, many sophisticated methods involving second-order information may fail to solve the problem in a reasonable time, moreover, the data may not fit in a memory of one computer. Therefore, there is a need for fast parallel/distributed algorithms, that can handle larger amounts of data.

In this study we propose a fast method for solving the SICS problem, taking into account the potentially huge size of the problems at hand. We achieve this goal by utilizing distributed parallel structure to handle large matrices with the help of computing clusters. We also propose a new block-coordinate step, discussed in 2.3.2, which helps the Coordinate Descent (CD) algorithm converge faster in practice.

This chapter first studies the existing coordinate descent algorithm (SINCO); this method involves updates of one coordinate at a time while maintaining feasibility of positive-semidefiniteness of the variable matrix and improving the objective function in a greedy manner. In practice, the convergence of this method can suffer when the same coordinates are updated frequently but by taking small steps, which leads to an increase in the number of iteration. We study the reasons for this slow-down and develop a novel block-coordinate update (SINCO2D) which aims to reduce such effects of small step-sizes. We rigorously prove that solutions of the subproblems in (SINCO2D) satisfy the positive-definiteness constraint and by remaining in the interior of the positive-semidefinite cone boundary, through proving the convexity of the higher dimensional subproblems, along with the number of solutions of the update system which ensures that iterative solutions will remain feasible with respect to the cone.

We also study the algorithm's structure and investigate different methods for parallelizing the algorithms by considering different ways of distributing the matrix and analyzing the resulting communication steps. Two different implementation strategies for the parallelized algorithm are discussed, one based on perfect balancing of coordinates among computational nodes (see for example Kambadur and Lozano [2013]), and the other approach is based on block cyclic distribution, which will be shown to be better suited for the blocked parallel linear algebra operations, which the algorithm requires. Finally, numerical tests are performed on several practical and synthetic datasets to show the efficiency of the proposed method and its parallel version compared to the original algorithm.

In what follows, the algorithms proposed for the SICS problem and its variants are briefly reviewed. Section 2.2 discusses the sequential coordinate descent algorithm and the structure of the parallelized counterpart. In Section 2.3 we first motivate the acceleration scheme by analyzing the iterations in the original CD algorithm and then we introduce the new update steps. Section 2.3.3 contains a summary of the proposed algorithm and details of updates as well as analysis of convergence and pathological cases. We conclude this study by presenting the numerical experiments on real and synthetic datasets and further present *heat map* plots, that illustrate the improved performance of CD algorithms for this problem.

Problem Description and related studies

In graphical models, recovering the underlying graph structure corresponding to conditional dependencies of random variables is of great importance for various practical problems. Graphical models are the intuitive characterizations of conditional independence structures exhibited by random variables. Define an undirected graph $G = (V, E)$ with a set of nodes $V = \{1, \dots, p\}$ connected by undirected edges. Each node represents a random variable, and an existing edge $\{i, j\}$ is an indicator of conditional dependence of random variables i, j.

Gaussian Graphical Models (GGMs) are models over the multivariate Gaussian distribution. This distribution is parameterized by the mean vector (μ) and covariance matrix (Σ) of its marginal random variables. SICS problem seeks to find the matrix variable that is obtained through the maximum likelihood estimator for the inverse covariance matrix, given (under assumption $\mu = 0$) by:

$$\Sigma^{-1} = \arg\max \ \frac{n}{2} \log\det X - \langle S, X \rangle, \tag{2.1}$$

where $S_{p\times p} = \frac{1}{n} \sum_i (x_i - \bar{x})(x_i - \bar{x})^T$ is the sample covariance matrix based on the observations $x_i \in \mathbb{R}^p$, where $(X \succ 0)$ is an implicit constraint, a natural artifact of Gaussian random distribution properties, making the $(\log\det X)$ a natural barrier for the positive-semidefinite cone in iterative methods. The reason making the estimation of the inverse matrix (Σ^{-1}) more desirable rather than the covariance matrix itself, is the main impetus of this problem in ML research. The problem aims to discover relational patterns between p features given n observations. These relations in the data visualization step, correspond to edges of the graph of features. To this end, the zero elements of Σ^{-1} have specific interpretations, corresponding to the independency of the corresponding random variables given the rest; further intrinsic structures of the problem can be easily imposed using constraints of type $X_{ij} = 0$ whenever the random variables (i,j) are conditionally independent. Therefore, the nonzero pattern of this matrix corresponds to the edges of graph G described above.

As an example of practical uses of this problem, Figure 2.1 presents the connectivity pattern of a graph associated with different sections of the brain. Given sample observations of brain activity in different parts, solving the SICS problem yields a sparse inverse covariance matrix that corresponds to the graph's edges and conditional dependencies, which results in the presented connectivity pattern.

As the number of features (p) is more than the number of observations (n) the true minimizer of the above function (S^{-1}) may not exist, as S is a low rank matrix. Therefore, by adding (convex) regularizers we can find estimates that are closest to

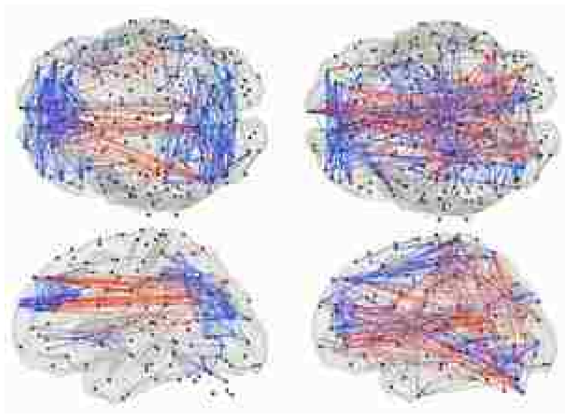


Figure 2.1: Connectivity pattern of different sections of the brain

the true parameter. The most interesting regularizer for this problem is ℓ_1 norm which is a convex surrogate for sparsity inducing l_0 norm. Using this norm we extract significant edges of the graph. Therefore, the problem will be written as

$$\max \frac{n}{2} \log \det X - \langle S, X \rangle - \lambda |X|_1, \tag{2.2}$$

where λ is a regularization parameter balancing the sparsity, and $S := \frac{n}{2}\bar{S}$. If a general balancing parameter of sparsity is used for each element, problem can be slightly modified by

$$\max_{X \succ 0} \frac{n}{2} \log \det X - \langle S, X \rangle - \sum_{i,j} \Gamma_{ij} |X_{ij}|, \tag{2.3}$$

where Γ_{ij} represents the weights for element X_{ij}.

One of earliest methods to solve the above problem, known as the Graphical Lasso, was proposed by Friedman et al. [2008]. The algorithm starts with an initial solution $\bar{\Sigma}^{-1} = S + \lambda I$ and then updates the rows at each iteration. The updates are based on taking the dual of the corresponding subproblem which is a Lasso problem (Banerjee et al. [2008]), that can be solved by a coordinate descent method. This method is further discussed in the third chapter of this document.

Methods with more sophisticated structures, which use first and second-order information also have been devised for this problem. Scheinberg et al. [2010] utilize first-order information within the Alternating Linearization Method (ALM) with $O(n^3)$ iteration complexity by using the spectral decomposition of the current incumbent solution. An algorithm which uses second-order information, known as QUIC, was proposed. Hsieh et al. [2011], propose forming a quadratic approximation model at each iteration and finding the proximal Newton direction by optimizing the model. Despite having various solution methods for this problem, as the problem size grows larger, the efficiency of most approaches reduces significantly. Thus, new ways of tackling these parameter estimation problems and, specifically, SICS problem are needed. An early approach to solving large-scale SICS problems was using divide and conquer method Hsieh et al. [2012]. In this study, Hsieh et al. partition the features into smaller subsets, providing grounds for computing a good starting point in parallel. However, the problem is then reduced to a single processor problem and is solved by QUIC which still suffers from dimensions of the problem.

Distributed parallel techniques can be a practical approach for this situation as they can handle the big matrices in distributed chunks. The first paper to consider the distributed setting for the sparse inverse covariance problem was proposed by Kambadur and Lozano [2013], using a coordinate descent method in a distributed way to achieve results and handle huge matrices. However, the sparsity is handled through the updates by backward sweeps on the selected coordinates and not considered by the ℓ_1 penalty which corroborates having a global optimum solution. Also utilizing parallel computing power by shared memory setting, Hsieh et al. [2013] propose BIGandQUIC which is a parallel counterpart of the QUIC algorithm. However, due to use of shared memory setting it cannot exploit the power of distributed computing grids.

Coordinate Descent (CD) methods have proven to be a useful and practical approach for large-scale machine learning problems (Richtárik and Takáč [2012]). Many researchers study different applications of CD methods and how to approach big data. Also specifically designed acceleration method for an application, potentially can surpass general acceleration schemes of CD methods. In this study, we revisit the idea of greedy coordinate descent (SINCO) by Scheinberg and Rish [2010]

as a basis of our work. The parallelizable structure of the algorithm alleviates extension to distributed setting. We also analyze the sequential algorithm and propose a new step which results in a faster algorithm, along with its parallel counterpart. Other CD methods such as Ginco (Kambadur and Lozano [2013]) could benefit from our acceleration method as well.
Next we discuss the original SINCO algorithm along with steps to parallelize it; further, we shall present the algorithm and discuss convergence.

2.2 Parallel SINCO

In this section, we outline the steps taken to parallelize the algorithm discussed in Scheinberg and Rish [2010]. A similar structure has been discussed in Kambadur and Lozano [2013], however here we consider the ℓ_1 regularized problem which has a finite optimal solution in all cases, unlike the problem considered in Kambadur and Lozano [2013], which only has a finite solution with some probability under specific assumptions on the distribution of the random variables. The other difference is using block cyclic distribution of the matrix, incorporated by ScaLAPACK, used with MPI as the backbone of this parallel algorithm; however, section 2.2.2 outlines our exploration of OpenMP for the parallelism.

2.2.1 SINCO Algorithm

First we present a brief outline of the original algorithm discussed in Scheinberg and Rish [2010]. In each step, the algorithm consists of two phases: *search phase* and *update phase*.
In the search phase (steps 3-5 of Algorithm 1), the method computes potential updates for all coordinates X_{ij} and selects the best coordinate producing the most increase in the objective function. In the update phase, using the X^*_{ij} information the method updates the coordinate in the inverse covariance matrix (steps 6-8); which is a rank two update $X + \alpha_{ij}(e_i e_j^T + e_j e_i^T)$. Next step is to update the maintained estimate of $W = X^{-1}$ using Sherman-Morrison formula for the rank-two update;

Algorithm 1 Outline of SINCO algorithm

```
1: Set $k = 0, X^{(k)} = \mathbf{I}, W^{(k)} = \mathbf{I}$
2: while $f(X^{k+1}) - f(X^k) > \epsilon$ do
3:     for $\forall\ (i,j), 1 \leq i \leq j \leq p$ do
4:         Compute $f_{ij}$ corresponding to updated value of the coordinate $(i,j)$
5:     end for
6:     Choose the best update $\alpha_{ij}$ for updating $(i,j) \leftarrow \arg\max_{(i,j)} f_{ij}$
7:     Update coordinate $X_{ij}^{(k+1)} \leftarrow X_{ij}^{(k)} + \alpha_{ij}(e_i e_j^T + e_j e_i^T)$
8:     Get $W^{(k+1)}$ by updating $W^{(k)}$ by Sherman-Morrison-Woodbury formula
9:     $k \leftarrow k + 1$
10: end while
```

this step is necessary as values of W are extensively used in the search phase of the algorithm. Each subproblem of the search phase will be a one-dimensional problem as

$$f_{ij}(\alpha) = \max \frac{n}{2} \log\det(X+\alpha(e_i e_j^T + e_j e_i^T)) - \langle S, X+\alpha(e_i e_j^T + e_j e_i^T)\rangle - \lambda|X+\alpha(e_i e_j^T + e_j e_i^T)|. \tag{2.4}$$

Updating step will be defined by $\alpha_{ij} = \arg\max_{i,j,\alpha} f_{ij}(\alpha)$. Using the rank-one update formula for the determinant we have:

$$\det(X + \alpha(e_i e_j^T + e_j e_i^T)) = \det(X)(1 + 2\alpha W_{ij} + \alpha^2(W_{ij}^2 - W_{ii}W_{jj})). \tag{2.5}$$

We can write the subproblem (2.4) as:

$$f(\theta) = \max \frac{n}{2} \log(1 + 2\alpha W_{ij} + \alpha^2(W_{ij}^2 - W_{ii}W_{jj})) - 2S_{ij}\alpha - 2\lambda|X_{ij} + \alpha|. \tag{2.6}$$

Solving the above problem — as discussed in Scheinberg and Rish [2010] — yields the optimal solution for the subproblem. The mentioned step is done for every coordinate and the best matrix coordinate $\bar{\theta} := \alpha_{ij}^*$ is chosen for updating. Modifying X, corresponding inverse matrix W should also be updated; this step can be done using Sherman-Morrison formula, resulting in update of W by

$$\bar{W} = W - \bar{\theta}(\kappa_1 W_i W_j^T + \kappa_2 W_i W_i^2 + \kappa_3 W_j W_j^T + \kappa_1 W_j W_i^T). \tag{2.7}$$

Where the coefficients are defined as

$$\begin{aligned}
\kappa_1 &= (-1 + \bar{\theta} W_{ii})/\kappa, \\
\kappa_2 &= (\bar{\theta} W_{jj})/\kappa, \\
\kappa_3 &= (\bar{\theta} W_{ii})/\kappa, \\
\kappa &= \bar{\theta}^2 (W_{ii} W_{jj} - W_{ij}^2) - 1 - 2\bar{\theta} W_{ij},
\end{aligned}$$

and W_i is i^{th} column of the matrix W.
The algorithm is a variant of a greedy coordinate descent algorithm for a convex problem and hence enjoys known convergence guarantees and a sub linear convergence rate. We can see that, since the updates are computed for each coordinate separately, it lends itself to distributed parallel setting. Next, we discuss how the data can be distributed among processors, assuming different distributions and methods for updating the inverse update step.

2.2.2 OpenMP Parallelization

Our initial investigation on the platform for parallelization involved study of how parallelization on levels of threads can help distribution of work load between processors, although the distributed parallelism is inevitable due to the large size of matrices involved in this problem. Implementation in this environment corresponds to the division of workload between different threads using the sections or parallel for directives. The first phase of the algorithm involves a simple search done on all coordinates, so we can just make the for loop parallel. In order to divide the compute load in the coordinate search phase, we need only to use *#pragma parallel for* for the search loop, however each thread although threads are now searching in the shared memory space, in order to save the results (maximum coordinates of each thread) we need to privatize the variables corresponding to coordinates and function value of sections corresponding to thread's territory. The important aspect that needs a specialized treatment is the update of the global best coordinate variables which will maintain the best coordinate information through all threads. To prevent race conditions we need the lines corresponding to this section in a critical statement so that no two threads execute these instructions simultaneously.

```
#pragma omp critical
{
 if (fnew > funmax ) {
    funmax=fnew;
    imax=i;
    jmax=j;
    alphamax=AllSteps(i,j).alpha;
    updatemax=AllSteps(i,j).update;}
}
```

Through this mechanism, since many coordinates may not have the sufficient increase in the objective, we will discard them and just continue the search and therefore have smaller number of threads blocked by the critical section bottleneck.

For the second phase which is updating the matrix with two rank one updates, as all the information is shared, there is no communication involved, however a barrier is needed before the second phase to ensure that all threads have synchronized information about the coordinates staged for updating and also the magnitudes of the updates. We also need to modify the update loop so that threads divide the load on updating the matrix coordinates of the inverse matrix to account for parallel environment. The following modification ensures that threads have privatized indexes *j*,*upIndex* which stores the vectorized index of the matrix coordinate for indices (i, j) and We only need to add the following before the for loop of updating different coordinates.

```
#pragma omp parallel for private(jj,upIndex)
```

For testing this approach we used a problem with 800 features which results in $640,000$ coordinates for the problem to work on. Unfortunately the run-times show that this approach involves congestion and blocking among CPUs which prevents full utilization of the processors compared to MPI work-load distribution.

Therefore, since the use of MPI communication is inevitable in the distributed

Table 2.1: Results for OpenMP with different number of threads

Threads	Runtime
2	197.88
4	371.19
8	1064.72

environment we preferred taking the MPI approach.

2.2.3 Data Distribution for Parallel Algorithm

Two different settings for parallelization of the data is considered for the algorithm. The first setting yields perfect balancing for the partitioned matrix coordinates among different processors. Second setting is based on block cyclic distribution, enabling use of efficient libraries such as ScaLAPACK. Both methods are described below.

Assume that there are τ processors available, $p^2/2$ coordinates need to be distributed evenly among processors. To reduce communication overhead, first $\lfloor p^2/2\tau \rfloor$ coordinates are assigned to first processor and so on until last processor gets the remaining coordinates. This way $\tau - 1$ processors handle the same number of coordinates and the workload of the last processor may have at most $\tau - 1$ more coordinates, which is negligible when $\tau \ll p$.

In order to apply this strategy for each processor k, we need to determine (i_k^s, j_k^s) and (i_k^f, j_k^f), corresponding to the starting and finishing coordinates of every matrix X, W assigned to that processor. We can have an iterative algorithm determining these numbers a priory, to facilitate the book keeping of the coordinates.

This also can be done using contiguous vectors, representing the matrices. However, book-keeping is also needed for the broadcast of columns as well as their updates. With our proposed algorithm the book-keeping of rows and columns are easily done for the coordinates. Assuming we have the (i^s, j^s) for a processor we wish to assign $C := \lfloor p^2/2\tau \rfloor$ coordinates of the symmetric matrix X, by computing (i^s, j^s). Letting $q = \left\lfloor -\frac{(2i^s-1)}{2} + \sqrt{(i-0.5)^2 + 2C} \right\rfloor$ we get an estimate of the number of rows that should be covered by processor k and calculate the q_2 as an indicator of finishing column. Different scenarios like exceeding the diagonal or having negative q_2 could arise for which we can easily adjust the coordinates so that every processor $k < \tau$ have the same number of coordinates and thus balancing the work load.

Algorithm 2 Balanced workload coordinate determination

```
1: Given $p$ coordinates and $n$ processor. Let $(i_1^s, j_1^s) = (1, 1)$.
2: Let $C = \frac{p(p+1)}{2n}$
3: for $k = 1, \ldots, n-1$ do
4:     $q_1 \leftarrow \left\lfloor -\frac{(2i_k^s-1)}{2} + \sqrt{(i_k^s - 0.5)^2 + 2C} \right\rfloor$
5:     $e \leftarrow \binom{i^s+q+1}{2} - \binom{i^s+1}{2}$
6:     $q_2 = C - e + j_k^s$
7:     if $q_2 > i_k^s$ then
8:         $e \leftarrow e + q_1 + i_k^s$, $q_1 \leftarrow q_1 + 1$
9:         $q_2 = C - e + j_k^s$
10:    end if
11:    if $q_2 < 0$ then
12:        $e \leftarrow e - q_1 + i_k^s$, $q_1 \leftarrow q_1 - 1$
13:        $q_2 = C - e + j_k^s$
14:    end if
15:    $i_k^f = i_s^k + q_1$, $j_k^f = q_2$
16:    $i_{k+1}^s = i_f^k + q_1$, $j_s^s = q_2 + 1$
17: end for
18: $i_k^f = p, i_k^f = p$
```

The other approach to data distribution is a *block cyclic* distribution. In order to use ScaLAPACK, and also distribute the workload among the processors, block cyclic distribution has proven useful (Choi et al. [1996]). This setting can be beneficial for communication efficient update of the matrices. Here, the processors are hypothetically divided into a grid. Matrices are divided based on a specific block size b_r, b_c for rows and columns, block by block. We can see that this distribution can lead to nearly balanced workload among processors. Figure 2.2 shows the structure of a 16×16 matrix, distributed among a processor grid consisting of two columns and two rows of processors by block size 4. Both methods of distribution were developed and tested. However, the second setting, allows the use of computationally and communication efficient libraries, hence we chose this setting for our further experiments. Throughout this chapter we refer to the first distribution as perfect balancing and to the second type as block cyclic distribution.

1	2	1	2
3	4	3	4
1	2	1	2
3	4	3	4

Figure 2.2: Block cyclic distribution by block-size $b = 4$ and $\tau = 4$ processors

2.2.4 Parallelization

We now discuss the key steps in the parallel implementation of the algorithm. As mentioned above, the sequential algorithm consists of two main phases: (I) Search Phase and (II) Update Phase.

Through the search phase, the algorithm searches for the best updating coordinate. As the coordinates of the matrix are divided among the processors, directing each processor to search among its designated coordinates (mapping) yields the parallel phase (I), which is demonstrated by figure 2.3.

For the second phase a more detailed procedure should be used. This phase encompasses rank two updates to the covariance matrix W. For the first type of the data distribution one should broadcast the update values (vectors W_i, W_j) and direct the processors to update their part of the inverse accordingly. However, a more efficient implementation could be used for the case of block cyclic distributed matrix using P?SYR2 procedure of ScaLAPACK corresponding to symmetric rank-2 update.

To summarize, during the search step, each processor is responsible for finding the

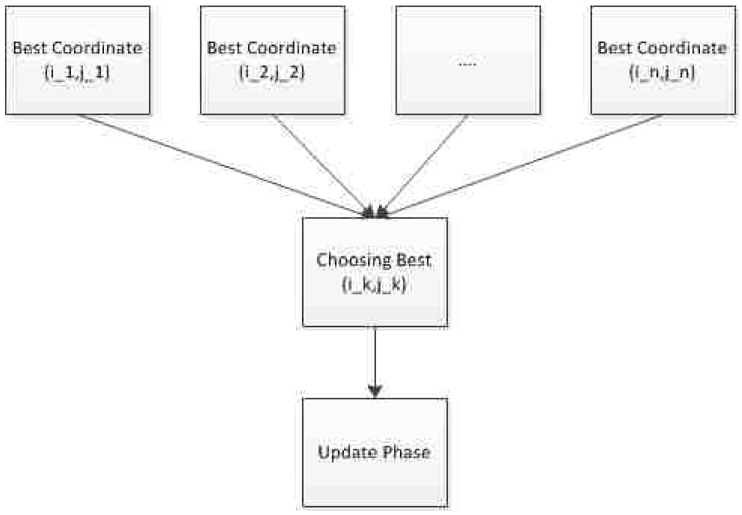

Best Coordinate (i_1,j_1)
Best Coordinate (i_2,j_2)
....
Best Coordinate (i_n,j_n)
Choosing Best (i_k,j_k)
Update Phase

Figure 2.3: Structure of the parallel search algorithm

best coordinate in its partition. Then the results are reduced to select the step with maximum objective value increase. For the update step we utilize P?SYR2 to implement the update formulas in section 2.3.6 in parallel. Next, we give a discussion on the modified algorithm.

2.2.5 Data Structures

As we expect the matrices to be sparse, in order to decrease memory requirement, we implemented our matrices using the CSR sparse format matrix using **Eigen** (Guennebaud et al. [2010]). However, for the dense matrices, simple <**vector**> is sufficient and prevents additional memory problems as it can easily be fed to ScaLAPACK API's.

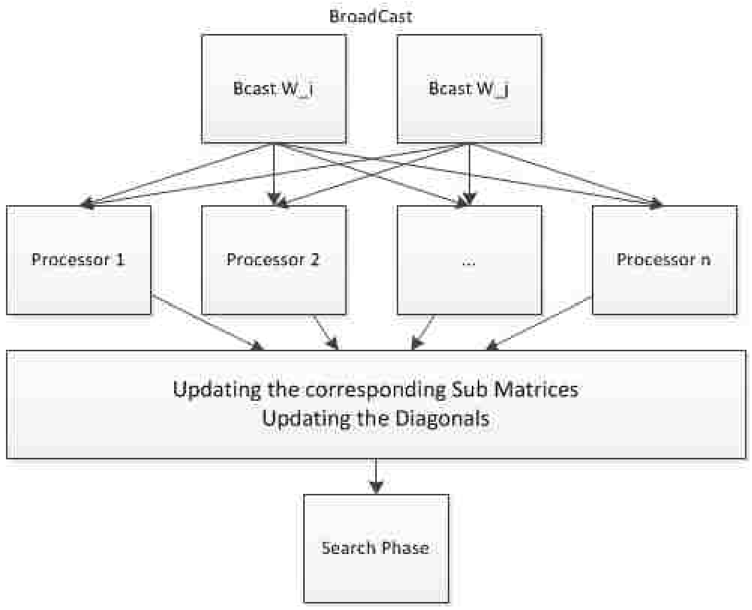


Figure 2.4: Structure of the parallel update algorithm with completely balanced data

2.3 SINCO2D

In this section we briefly outline the improved version of SINCO (Scheinberg and Rish [2010]) which is aimed at reducing the number of updates and thus iterations.

2.3.1 Motivating New Updates

In this section, we briefly analyze and improve SINCO, by aiming to reduce the number of iterations in order to speed up the convergence. To design a faster CD method, we first analyze the original SINCO algorithm; observing the sequence of selected coordinates during SINCO iterations can give insights on how the algorithm works. This analysis reveals that throughout the iterations, special subsequences appear frequently. These sequences are composed of recurring consecutive iterations revolving around a specific coordinate X_{ij} and the corresponding diagonal entries X_{ii}, X_{jj}. When the algorithm selects the off-diagonal coordinate X_{ij} for update, the next iteration involves the corresponding diagonals and consequently selecting the

same coordinate X_{ij} for re-adjustment. These sequences occur frequently throughout the trajectory of SINCO iterations for distinct coordinates X_{ij}.
To illustrate our point, we present the following trend from the consecutive iterations of SINCO algorithm applied to Arabidopsis dataset (Li and Toh [2010]).

$$(1,1) \rightarrow (\mathbf{1,29}) \rightarrow (29,29) \rightarrow (\mathbf{1,29}) \rightarrow (1,1) \rightarrow (\mathbf{1,27}) \rightarrow (27,27) \rightarrow (\mathbf{1,27})$$

The above sequence shows the coordinates selected by SINCO in eight consecutive intermediate iterations. Updating $X_{1,1}$, the bound on coordinate $X_{1,29}$ changes; enabling further adjustment for this coordinate. This sequence, in turn, triggers further possible updates of other diagonal elements. We wish to eliminate such loops in the algorithm.
Let us identify the issue and modify the algorithm accordingly. Considering element X_{ij} needs to be updated at the current iteration; as we have the constraint $X \succ 0$, we know that the matrix will be positive-definite iff all principal components are also positive-definite. Considering the 2×2 sub-matrix $X_{[i,j],[i,j]} \succ 0$ (using Matlab sub-matrix notation) means that

$$(X_{ij} + \theta)^2 < X_{ii}X_{jj} \leq \max(X_{ii}, X_{jj})^2, \tag{2.8}$$

which can be written as $|X_{ij} + \theta| \leq M$, where $M = \max(X_{ii}, X_{jj})$. Consider the case where in the optimal solution X^* we have $|M - X_{ij}| < |X^*_{ij} - X_{ij}|$. Then it is obvious that we need at least three iterations to get to this solution. On the other hand, a simultaneous update for all three coordinates could result in reaching the desired solution in one step.

2.3.2 Augmented Subproblems

Removing the undesired iteration loops –analyzed in previous section– potentially can result in a fewer iterations and therefore faster convergence. As discussed above, an important factor bounding the eigenvalues, depends on the diagonals. Therefore, augmenting the updates by including the diagonals, can prevent the need for iterative updating loops.

The augmented subproblems takes into account the diagonal variables of the corresponding element (i,j), in order to potentially block recurring updates of the element X_{ij}. Namely we modify the old updates ($X^{k+1} = X^k + \theta e_i e_j^T + \theta e_j e_i^T$), with $X^k = X^k + E$ where E accounts for the shift matrix having the form

$$X^{k+1} = X^k + \begin{array}{c|cc|} & i & j \\ \hline & & \\ i & \mu & \theta \\ j & \theta & \lambda \\ & & \\ \hline \end{array},$$

allowing simultaneous update of three desired coordinates. Shown in compact form $X^{k+1} = X^k + uv^T + wy^T$ where $u = \alpha e_j + \mu e_i$, $v = e_i$, $w = \alpha e_i + \lambda e_j$ $y = e_j$; here e_i is the i'th column of identity matrix of proper size. We can get the determinant of the updated matrix by twice invoking rank-1 update of the determinant:

$$\det(X^k + uv^T + wy^T) = \det(X^k)(1 + \mu W_{ii} + 2\theta W_{ij} + \lambda W_{jj} + \theta^2\kappa - \lambda\mu\kappa), \quad (2.9)$$

where $\kappa = (W_{ij}^2 - W_{ii}W_{jj})$. The subproblem to obtain update is described as follows:

$$\underset{\lambda,\mu,\theta}{Max} \quad \frac{n}{2}\log(G_{\lambda,\mu,\theta}) - \langle S_{[i,j]}, E_{[i,j]}\rangle - (\Gamma_{ij} + \Gamma_{ji})|X_{ij} + \theta| - \Gamma_{ii}(X_{ii} + \mu) - \Gamma_{jj}(X_{jj} + \lambda),$$

where

$$G_{\lambda,\mu,\theta} = 1 + \mu W_{ii} + 2\theta W_{ij} + \lambda W_{jj} + \theta^2\kappa - \lambda\mu\kappa. \quad (2.10)$$

As the original subproblem has non-smooth term $(\Gamma_{ij} + \Gamma_{ji})|X_{ij} + \theta|$, first-order condition cannot be directly used. However, in the 1-dimensional search phase, a candidate X_{ij} selected for updating, along with θ_{ij}^* (optimal step-size given from 1-d coordinate descent step, in the original algorithm) may yield a reliable approximation to the sign of the $|X_{ij} + \theta_{ij}|$. The backbone of this reasoning lies in our previous observations; arguing that steps θ^* are small attempts to reach the optimal quantity of X_{ij}^*. Using this information the objective function can be smoothed in this direction. For each subproblem the parameter (a) is the regularization parameter corresponding to the smoothed version of the function. We would define

$a = \mathrm{Sgn}(X_{ij} + \theta^*_{ij})(\Gamma_{ij} + \Gamma_{ji})$; resulting in the subproblem having the smoothed form of

$$\underset{\lambda,\mu,\theta}{Max} f_a := \frac{n}{2}\log(G_{\lambda,\mu,\theta}) - \langle S_{[i,j]}, E_{[i,j]}\rangle - a(X_{ij} + \theta) - \Gamma_{ii}(X_{ii} + \mu) - \Gamma_{jj}(X_{jj} + \lambda),$$

For simplicity of notation, hereafter we use G to denote $G_{\lambda,\mu,\theta}$ and K to denote $\frac{n}{2}$. Since the subproblem is convex in the feasible domain, we exploit first-order conditions in order to solve this problem by a procedure with low computational cost as original SINCO step-size subproblem (up to constant terms). In order to use the first-order condition we require finding points that satisfy $\nabla f_a(\lambda, \mu, \theta) = 0$; the gradient is defined by

$$\nabla f_a = (K\frac{W_{ii} - \lambda\kappa}{G} - S_{ii} - b, K\frac{W_{jj} - \mu\kappa}{G} - S_{jj} - c, K\frac{2W_{ij} + 2\theta\kappa}{G} - 2S_{ij} - a), \quad (2.11)$$

where $b = \Gamma_{ii}$ and $c = \Gamma_{jj}$. By the convexity on the domain, using first-order condition leads to

$$\frac{K}{G}\begin{pmatrix} W_{ii} - \lambda\kappa \\ W_{jj} - \mu\kappa \\ 2W_{ij} + 2\theta\kappa \end{pmatrix} = \begin{pmatrix} S_{ii} + b \\ S_{jj} + c \\ 2S_{ij} + a \end{pmatrix}, \quad (2.12)$$

we see G is a nonlinear term that complicates the condition; its elimination will help obtaining a linear system of equations. Notice G can be eliminated if it is not equal to 0. As long as modifications to the matrix X, namely μ, λ, θ steps, remain feasible i.e. $X^{k+1} \succ 0$ we have $G > 0$. However for this section we only assume $G > 0$ rather than $X^{k+1} \succ 0$, we shall later discuss why the latter holds, in section 2.3.5. Also by definition we have $S_{ii} = \frac{1}{n}\|x_i\|^2 > 0$ where $x_i \in \mathbb{R}^p$ is i'th sample observation. From $\Gamma_{ii} \geq 0$ we have $S_{ii} + \Gamma_{ii} > 0$, thus elimination of G can safely be done. It should be noted that for $2S_{ij} + a = 0$ the solution for θ can be found easily, therefore assuming $2S_{ij} + a \neq 0$. Consequently by dividing the equations in (2.12), we get the following relations:

$$(W_{ii} - \lambda\kappa) = \beta(2W_{ij} + 2\theta\kappa) \quad (2.13)$$

$$(W_{ii} - \lambda\kappa) = \alpha(W_{jj} - \mu\kappa) \quad (2.14)$$

where $\beta := \dfrac{S_{ii}+b}{2S_{ij}+a}, \alpha := \dfrac{S_{ii}+b}{S_{jj}+c}$. Rewriting the above as

$$\begin{pmatrix}\mu \\ \lambda\end{pmatrix} = \frac{1}{\kappa}\begin{pmatrix}-2\frac{\beta}{\alpha}W_{ij}+W_{jj} \\ W_{ii}-2\beta W_{ij}\end{pmatrix} - \begin{pmatrix}2\frac{\beta}{\alpha} \\ 2\beta\end{pmatrix}\theta := b - A\theta, \tag{2.15}$$

gives a parameterized (by θ) equivalent of the system of equations described by (2.13), (2.14). However, satisfying these systems only means that the divided ratios of gradient elements are equal. Using the first-order condition, gradient elements should be equal to 0, here we use gradient of smoothed function f_a with respect to θ. Rewriting the corresponding partial derivative equation ($\frac{\partial f_a}{\partial \theta} = 0$), we have

$$\frac{K(2W_{ij}+2\theta\kappa)}{G} = 2S_{ij}+a. \tag{2.16}$$

By (2.10) and (2.15) we can rewrite $G_{\mu,\lambda,\theta}$ as

$$G_{\mu,\lambda,\theta} \quad = r_1 + r_2\theta + r_3\theta^2, \tag{2.17}$$

where

$$\begin{aligned} r_1 &:= 1 + W_{ii}b_1 + W_{jj}b_2 - \kappa b_1 b_2, \\ r_2 &:= -A_1W_{ii} + 2W_{ij} - A_2W_{jj} + \kappa b_1 A_2 + \kappa b_2 A_1, \\ r_3 &:= \kappa(1 - A_1A_2). \end{aligned}$$

Here A_k, b_k for $k = 1, 2$ correspond to elements of 2d vectors described in (2.15). The quadratic (2.17) shows the determinant changes are parameterized by θ, on the line described by (2.15) where ratios of gradient elements are equal to each other. We can also rewrite G from (2.16) as

$$G = \frac{K(2W_{ij}+2\theta\kappa)}{2s_{ij}+a} = \frac{2KW_{ij}}{2s_{ij}+a} + \frac{2\kappa}{2s_{ij}+a}\theta \quad := \quad t_1 + t_2\theta, \tag{2.18}$$

where $t_1 = \dfrac{2KW_{ij}}{2S_{ij}+a}$, $t_2 = \dfrac{2K\kappa}{2S_{ij}+a}$.

To summarize we characterized determinant update (G) for solutions that satisfy (2.15) ratios because of quadratic equation (2.17), and for solutions that satisfy

$\nabla_\theta f = 0$ from (2.18). Intersecting these two solution sets, will result in our desired subproblem solution, i.e. having

$$r_1 + r_2\theta + r_3\theta^2 \quad = \quad t_1 + t_2, \theta, \tag{2.19}$$

which simplifies to

$$(r_1 - t_1) + (r_2 - t_2)\theta + r_3\theta^2 \quad = \quad 0. \tag{2.20}$$

Therefore, solving the subproblem reduces to solving the above quadratic equation, and by finding optimal θ from above we can determine μ, λ from (2.15). We point out that the computational complexity is constant as in the 1-d coordinate descent.

2.3.3 Outline of the Algorithm

In this section we describe the modified SINCO algorithm (called SINCO2D) which incorporates our new update procedure. The algorithm — similar to the original version — is a (block) coordinate ascent algorithm aimed at recovering the non-zero pattern of the inverse covariance matrix, by updating up to three elements at each iteration. The algorithm is outlined in Algorithm 3. The algorithm resembles original SINCO algorithm (Scheinberg and Rish [2010]) however here a matrix C is introduced to maintain the number of times a coordinate is selected. While the selection count of the coordinate is smaller than some pre-determined threshold (δ) the algorithm proceeds the same as SINCO. However, when a coordinate (i, j) becomes more popular ($C_{ij} > \delta$), it can be beneficial to switch to the 2d updates. This is done by procedure ***2d-Sub***, described in Algorithm 4.

2.3.4 Subproblem Feasibility

As mentioned previously, some special cases may occur and need to be handled with caution. First issue is the solution θ^*_{ij} and how it behaves with regards to the sign(a). For example, when $a = \Gamma_{ij} + \Gamma_{ji}$, it is assumed that $X_{ij} + \theta_{ij}$ will remain positive;

Algorithm 3 SINCO2D

```
 1: Set $k = 0$,$X^{(k)} = \mathbf{I}, W^{(k)} = \mathbf{I}, C = \mathbf{0}$
 2: while $f(X^{k+1}) - f(X^k) > \epsilon$ do
 3:     for Each coordinate$(i, j)$ do
 4:         Find best step-size along coordinate, i.e. $\alpha_{ij}$
 5:         Compute $f_{ij}$ corresponding to function reduction in $ij$'th coordinate
 6:     end for
 7:     Choose best promising coordinate for update
 8:     $(i, j) \leftarrow \arg\max_{(i,j)} f_{ij}$
 9:     $C_{ij} \leftarrow C_{ij} + 1$
10:     if $C_{ij} > \delta$ then
11:         Call 2d-Sub to compute step-sizes $\mu, \lambda, \theta$ and corresponding update
12:     else
13:         Update $X \leftarrow X + \alpha_{ij}(e_i e_j^T + e_j e_i^T)$
14:         Update W by Sherman-Morrison-Woodbury
15:     end if
16: end while
```

Algorithm 4 2d-Sub

```
 1: Input : $\bar{\alpha}_{ij} := \theta_{ij}$                    ▷ $\theta_{ij}$ is the 1d step-size
 2: Let $a :=$ Sgn$(X_{ij} + \bar{\alpha}_{ij})(\Gamma_{ij} + \Gamma_{ji})$
 3: Solve (2.20) for $\theta$ and use(2.15) for $\mu, \lambda$
 4: if Sgn$(X_{ij} + \theta) \neq$ Sgn$(a)$and $G_{\mu,\lambda,\theta} > 0$ then
 5:     Reject 2d-update
 6:     Use 1d-update and $\bar{\alpha}_{ij}$ as step-size
 7:     Update $W$ using Sherman-Morrison-Woodbury formula
 8: else
 9:     $\begin{pmatrix} X_{ii} & X_{ij} \\ X_{ji} & X_{jj} \end{pmatrix} \leftarrow \begin{pmatrix} X_{ii} & X_{ij} \\ X_{ji} & X_{jj} \end{pmatrix} + \begin{pmatrix} \mu & \theta \\ \theta & \lambda \end{pmatrix}$
10:     Update W by (2.24)
11:     $C_{ij} \leftarrow 0$
12: end if
```

therefore, if $X_{ij} + \theta^*$ is negative, we can see that the smoothing scheme has led to a non-optimal solution, vice versa for the case of $a = -(\Gamma_{ij} + \Gamma_{ji})$. This issue arises as a result of discrepancies of non-smooth function and the smoothed version, shown in Figure 2.5. The figure shows the objective function of the subproblem plotted with regards to one parameter (θ). Here, the subproblem assumption is that $X_{ij} + \theta_{ij}$

remains negative; thus we can see that the two functions, after passing to the positive region, begin to diverge.
Therefore, one way to overcome this issue, is to reject the 2d-step, which would leave the current iterate unchanged. However, to achieve improvement after rejecting 2d-step we would update the coordinate with the 1-d step $\bar{\alpha}_{ij}$ and just update the off-diagonal element as original algorithm.

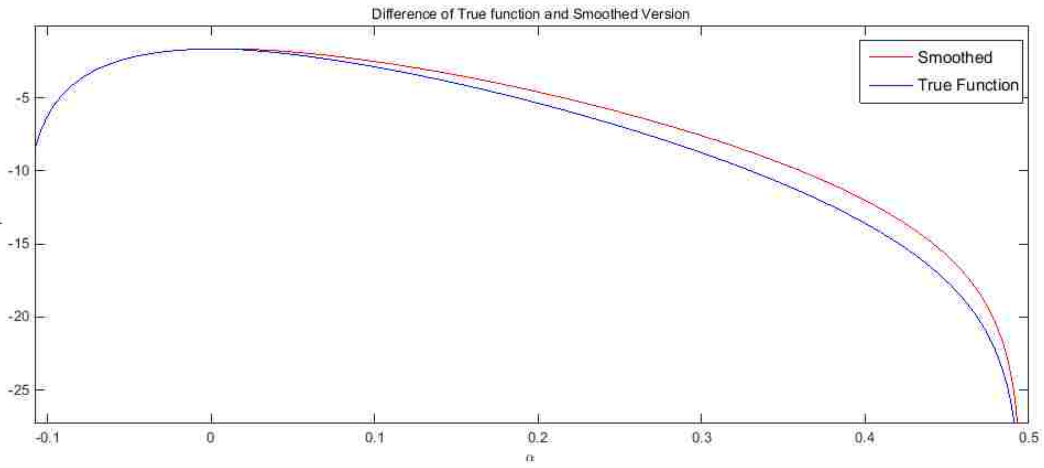


Figure 2.5: Comparison of smoothed function vs true function of the augmented subproblem

Looking at first-order condition system (2.11), assuming $G > 0$ we can rewrite equation (2.12) as

$$K \begin{pmatrix} W_{ii} - \lambda\kappa \\ W_{jj} - \mu\kappa \\ 2W_{ij} + 2\theta\kappa \end{pmatrix} = G_{\mu,\lambda,\theta} \begin{pmatrix} S_{ii} + b \\ S_{jj} + c \\ 2S_{ij} + a \end{pmatrix}. \tag{2.21}$$

The system indicates that as the solution vector $\begin{pmatrix} \mu \\ \lambda \\ \theta \end{pmatrix} \rightarrow \frac{1}{\kappa} \begin{pmatrix} W_{jj} \\ W_{ii} \\ -W_{ij} \end{pmatrix}$, results in $G \rightarrow 0$.
This solution essentially updates the sub-matrix $X_{[i,j],[i,j]}$ by $-(W_{[i,j][i,j]})^{-1}$. This

update satisfies the system described by (2.13),(2.14). Therefore, it is one of the two (possible) roots of equation (2.20), with $G = 0$. Therefore, the quadratic equation always has at least one root. Given this observation next section discuss the feasibility in further detail.

2.3.5 Problem Feasibility

In this section we analyze the updates using the 2d-step, for the original problem. Assuming the rank-2 update matrix ($E = uv^T + wy^T$) is determined based on $\mu^*, \lambda^*, \theta^*$ how can we ensure that the updated matrix is positive-definite. One idea is to check the positive-definiteness of the the updated 2×2 sub-matrix and also the corresponding $p-2 \times p-2$ Schur complement matrix. However, computing this measure of feasibility may not be practical. Therefore, we present another argument by extending the determinant property.

To this end, the only measure of determinant change we have discussed so far is maintaining $G > 0$. However, one can argue that it cannot be a good measure, as it is possible to flip the sign of two eigenvalues simultaneously, keeping the non-negativity of the determinant. For example let A_α as

$$A_\alpha = \begin{pmatrix} 1+\alpha & 0 & 0 & 0 \\ 0 & 1+\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.22}$$

The determinant of the updated matrix is always non-negative $(1+\alpha)^2$, even if $A_\alpha \not\succeq 0$ (e.g. let $\alpha = -2$). Therefore, simply having $G > 0$ will not be a good measure for our purpose. Looking at the determinant as a function of α, vs the function $\log(1+\alpha)^2 - \alpha$ we see that when $\alpha > -1$ we are in the semidefinite cone and also in the domain of $\log\det$, however for $\alpha < -1$ we are still in the domain of the $\log\det$ function but not on the positive-definite cone. As long as we have a measure ensuring cone's boundary is not crossed (similar to logarithmic barrier approaches), the matrices will be positive-definite.

The feasibility argument here is based on our solution scheme for the subproblem. As the main problem is convex and finite, the subproblem will also have a finite optimal solution. The optimum should satisfy the first-order necessary condition which is a nonlinear system of equations. Reducing the system to two simpler conditions, our method aims to satisfy these two conditions. The two conditions include equality of ratios (2.12), and $\nabla_\theta f = 0$. Satisfying the two conditions we find candidate θ^* solutions to satisfy the first-order conditions.
As the procedure is reduced to finding the roots of a quadratic equation, one expects that at most two candidate θ^*s are obtained, one of which corresponding to the optimum solution of the subproblem. In the section 2.3.4 we showed that the other solution to the nonlinear system of equations is $(\mu, \lambda, \theta)^T = \frac{1}{\kappa}(W_{jj}, W_{ii}, -W_{ij})^T$. As this update satisfies the system described by (2.13),(2.14), it is one of the two (possible) roots of equation (2.20), with $G = 0$. Therefore, the quadratic equation always has at least one root; we show that the equation will always have another root and also we show that for this root $G > 0$. This concludes that only one unique point exists that satisfies the necessary condition of the convex problem with $G > 0$.

Remark 1 *For $\rho|X|_1$ regularizer, ($\rho > 0$) , assuming $2s_{ij} + a \neq 0$, quadratic equations* (2.17), (2.20) *are convex.*

Proof. Both quadratics have (r_3) as the coefficient of θ^2, thus we just need to show $r_3 \geq 0$.

$$r_3 := \kappa(1 - A_1 A_2).$$

As $W^k \succ 0$, we have $\kappa < 0$, and therefore, need to show that $A_1 A_2 \geq 1$.
Expanding this term we get

$$A_1 A_2 = (2\frac{\beta}{\alpha})(2\beta) = 4\frac{\beta^2}{\alpha} = 4\frac{(s_{ii}+\rho)^2(s_{jj}+\rho)}{(2s_{ij}+a)^2(s_{ii}+\rho)} = \frac{(s_{ii}+\rho)(s_{jj}+\rho)}{(s_{ij}+(-1)^q\rho)^2},$$

as $q \in \{0, 1\}$. Now let T_1 and T_2 matrices be defined as

$$T_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} T_2 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix},$$

where both matrices are positive-semidefinite. As S is sample covariance matrix we have $S \succeq 0$ and principal sub-matrix of indices i, j will have the same property, i.e. $S_{[i,j],[i,j]} \succeq 0$. Therefore, $S_{[i,j],[i,j]} + \rho T_1 \succeq 0$ and $S_{[i,j],[i,j]} + \rho T_2 \succeq 0$ leading to determinant of both matrices being non-negative:

$$(s_{ii} + \rho)(s_{jj} + \rho) - (s_{ij} \pm \rho)^2 \geq 0$$

Which leads to $A_1 A_2 \geq 1$.

$\square$

In the above $(-1)^q$ is utilized to prove the property for parameter $a = \text{Sgn}(x_{ij} + \theta^*)(2\rho)$ with both signs as we are uncertain which case may happen.

Remark 2 *For $\rho|X|_1$ regularizer, all solutions will have $G \geq 0$ corresponding to line described by* (2.15)*, with exactly one point satisfying $G = 0$.*

Proof. For all solutions (μ, λ, θ) given by (2.15), $G_{\mu,\lambda,\theta}$ is described by (2.17); further, previous remark shows (2.17) is convex; thus we need to show that critical point of this quadratic results in $G = 0$ which is sufficient to have $G > 0$ elsewhere. Apex of any quadratic $a\theta^2 + b\theta + c$ is described by $\theta = -b/2a$. Using (2.15) for definitions of $A = (A_1, A_2)^T$ and $b = (b_1, b_2)^T$, the critical point of (2.17) is:

$$\begin{aligned}
\theta = \frac{-r_2}{2r_3} &= -\frac{-A_1 W_{ii} + 2W_{ij} - A_2 W_{jj} + \kappa b_1 A_2 + \kappa b_2 A_1}{2\kappa(1 - A_1 A_2)} \\
&= -\frac{A_1(\kappa b_2 - W_{ii}) + 2W_{ij} + A_2(\kappa b_1 - W_{jj})}{2\kappa(1 - A_1 A_2)} \\
&= -\frac{A_1(-2\beta W_{ij}) + 2W_{ij} + A_2(-2\frac{\beta}{\alpha} W_{ij})}{2\kappa(1 - A_1 A_2)} \\
&= -\frac{(-2\beta A_1 - 2\frac{\beta}{\alpha} A_2 + 2)W_{ij}}{2\kappa(1 - A_1 A_2)} \\
&= -\frac{(-A_2 A_1 - A_1 A_2 + 2)W_{ij}}{2\kappa(1 - A_1 A_2)} = -\frac{W_{ij}}{\kappa}.
\end{aligned}$$

By arguments in Section 2.3.4, when (2.15) and (2.17) hold, $\theta \to -\frac{W_{ij}}{\kappa}$ results in $G = 0$. Thus the minimum of (2.17) is 0, and by convexity ($r_3 \geq 0$) this implies that for any other points in (2.15), G will be strictly positive. $\square$

The conclusion from above is that in step 3 of algorithm 4, which is acquired by finding the roots of (2.20), will always have two solutions where one will correspond to $G = 0$ (critical point of (2.17)) and the other necessarily will have $G > 0$; unless equations (2.17),(2.20) are identical, i.e. t_1, t_2 from (2.18) are both equal to 0, which is not the case as $\kappa \neq 0$. This ensures that the subproblem always have one point satisfying the necessary condition (2.11) with $G > 0$.
Lastly, we just point out that the SINCO updates are a subset of the new sub problems; i.e. constraints $\mu = \lambda = 0$, in the new subproblem results in achieving the old update; Therefore, the updates at the worst case have the same function reduction of SINCO, however our numerical experiments show that in practice the sub problems are indeed yielding more powerful updates.

2.3.6 Inverse Updates

Since we are using two type of updates for the matrix X, we need to clarify the arguments for updating the inverse matrix W. For updates of the original SINCO we would use (2.7), however for 2d-updates we have

$$\bar{W} = (\bar{X})^{-1} = (X + uv^T + wy^T)^{-1}, \tag{2.23}$$

where $u = \mu e_i + \theta e_j, v = e_i$ and $w = \theta e_i + \lambda e_j, y = e_j$. Invoking Sherman-Morrison formula twice, we would update the inverse matrix $\bar{W}$ by

$$\bar{\mathbf{W}} = \mathbf{X}^{-1} - \frac{a_1}{T}\mathbf{W_i}\mathbf{W_j}^T + (\frac{a_1 a_3}{T} - \frac{\mu}{\kappa_1})\mathbf{W_i}\mathbf{W_i}^T - \frac{a_2}{T}\mathbf{W_j}\mathbf{W_j}^T + (\frac{a_2 a_3}{T} - \frac{\theta}{\kappa_1})\mathbf{W_j}\mathbf{W_i}^T, \tag{2.24}$$

where

$$\begin{aligned} \kappa_1 &= 1 + \mu W_{ii} + \theta W_{ij}, & a_1 &= \theta - \frac{\theta\mu W_{ii}}{\kappa_1} - \frac{\mu\lambda W_{ij}}{\kappa_1}, \\ a_2 &= \lambda - \frac{\theta\lambda W_{ij}}{\kappa_1} - \frac{\theta^2 W_{ii}}{\kappa_1}, & a_3 &= \frac{\mu W_{ij} + \theta W_{jj}}{\kappa_1}, \end{aligned}$$

$$T = 1 + \theta W_{ij} + \lambda W_{jj} - \frac{\mu W_{ii} + \lambda W_{jj}}{\kappa_1}\theta W_{ij} - \frac{\mu\lambda}{\kappa_1}W_{ij}^2 - \frac{\theta^2}{\kappa_1}W_{jj}W_{ii}$$

Superscripts k for W^k elements ($W^k_{\bullet\bullet}$) and columns ($W^k_{\bullet}$) are dropped for better readability. The above update scheme is used whenever ***2d-Sub*** finds a suitable update, i.e. Step 10 of Algorithm 4. Appendix 2.3.2 shows our implementation of this method. For the first type of workload distribution, we would simply broadcast the vectors $W_{i\bullet}, W_{j\bullet}$ and each processor will update their part accordingly. However, in order to utilize ScaLAPACK, we can use the symmetric Rank-2 update, which updates the matrix by

$$W = W + \alpha xy^T + \alpha yx^T. \tag{2.25}$$

We can rotate our updating vectors by

$$r_1 W_i W_j^T + r_2 W_i W_i^T + r_3 W_j W_j^T + r_4 W_j W_i^T = xy^T + yx^T. \tag{2.26}$$

Defining $x = pW_i + qW_j$ and $y = mW_i + nW_j$, we can easily find an analytical solution to the above equation and use the library with the rotated vectors. Also we need to account for a degenerate case where the update can have Rank-1 form.

2.4 Numerical Experiments

In order to examine the performance of the new algorithm both algorithms were implemented in C++. Both synthetic data as well as practical datasets were used to measure the performance of the algorithms. In all of the experimental runs, the stopping criteria of step 2 of algorithm 3 was utilized with relative tolerance of 10^{-6}.

2.4.1 Synthetic Data

Following the setup of Hsieh et al. [2011], Kambadur and Lozano [2013], we investigate the performance of the algorithms on two sets of synthetic data. Two main sparse structures of graphs are examined for the SICS problem: networks corresponding to chain graphs and random sparse graphs.

- Chain Graph: Random data is drawn from a Gaussian distribution where inverse of the covariance matrix is formed by $\Sigma^{-1}_{ii} = 1.25, \Sigma^{-1}_{i,i-1} = -0.5$.

- Random Graphs: The inverse of the covariance matrix is generated first by defining a sparse matrix X with random elements equal to ± 1 and taking $\Sigma^{-1} = I + X^T X$. The sparsity of Σ^{-1} is controlled through non-zeros of X in order to have $2p$ elements to define the empirical covariance matrix.

For each case $n = \frac{p}{2}$ i.i.d observations are taken from the corresponding Gaussian distribution with covariance matrix Σ. Here, we use ρ notation as the penalty weight, which was denoted by λ in (2.2) discussed above; we should point out that in our formulation multiplying (2.2) by $\frac{2}{n}$ eliminates n in the log det term as well as implicit n in S and thus giving: $\rho = \frac{2\lambda}{n}$.
Table 2.2 shows that although QUIC is slightly faster in picking up true positive coordinates, however SINCO2D, eventually picks up the correct coordinates with significantly smaller rate of false-positive error. Moreover, the FP gap intensifies with the problem size growth.

Table 2.2: Results for the chain graph datasets

Size	ρ	**SINCO2D**		**QUIC**	
		TP(%)	**FP(%)**	**TP(%)**	**FP(%)**
100	1	36.24%	0.00%	36.24%	0.00%
	0.5	72.48%	0.47%	76.51%	0.78%
	0.1	100.00%	32.53%	100.00%	37.02%
	0.01	99.33%	78.33%	99.33%	74.44%
500	1	33.38%	0.00%	33.38%	0.00%
	0.5	91.46%	0.00%	96.93%	0.01%
	0.1	100.00%	6.08%	100.00%	14.52%
	0.01	100.00%	54.55%	100.00%	66.43%
1000	1	33.36%	0.00%	33.36%	0.00%
	0.5	94.33%	0.00%	99.53%	0.00%
	0.1	100.00%	0.92%	100.00%	6.63%
	0.01	100.00%	30.44%	100.00%	60.96%
2000	1	33.34%	0.00%	33.34%	0.00%
	0.5	94.00%	0.00%	99.97%	0.00%
	0.1	100.00%	0.01%	100.00%	1.56%
	0.01	100.00%	6.90%	100.00%	54.51%

Table 2.3: Average results for the random synthetic datasets

Size	ρ	**SINCO2D**		**QUIC**	
		TP(%)	**FP(%)**	**TP(%)**	**FP(%)**
100	1	68.49	0.00	68.49	0.00
	0.5	68.49	0.02	68.49	0.02
	0.1	91.78	22.39	91.78	25.01
	0.01	100.00	73.72	98.63	72.62
	0.001	100.00	88.45	98.63	83.05
500	1	66.31	0.00	66.31	0.00
	0.5	66.31	0.00	66.31	0.00
	0.1	93.63	1.96	94.43	4.39
	0.01	100.00	58.92	100.00	61.72
	0.001	100.00	70.17	100.00	80.60
1000	1	66.76	0.00	66.76	0.00
	0.5	66.76	0.00	66.76	0.00
	0.1	93.46	0.10	95.33	0.80
	0.01	100.00	37.82	100.00	54.81
	0.001	100.00	56.37	*	*
2000	1	66.93	0.00	66.93	0.00
	0.5	66.93	0.00	66.93	0.00
	0.1	91.57	0.00	95.31	0.04
	0.01	100.00	12.89	100.00	46.77
	0.001	100.00	39.15	*	*

Table 2.3 shows that for random graphs, as the solutions are getting more dense lower (ρ), both algorithms pick up more true-positive coordinates as well as false-positive coordinates. However, due to the structure of the algorithm as well as its size scalability, when the size grows SINCO2D has significantly lower rate of picking FP points, whereas TP rate can grow to pick all the true coordinates. Missing values in the table correspond to problems not solved in 10 hours.
Therefore, in both cases, although QUIC may find true-positives faster, however the byproduct is to add excessive false-positives as well; whereas SINCO2D eventually finds the true-positive coordinates, by maintaining the number of false-positives to remain as small as possible, by the construction of algorithm to preserve sparsity.

2.4.2 Practical Datasets

In addition to synthetic data, performances are measured by examining real world problems. The true underlying inverse covariance matrix is not known for these problems to provide grounds for measures such as true and false-positives.
Visualizing the performance of the new algorithm vs the old algorithm, we present heat maps of all elements based on their selection frequency in the iterates of SINCO and SINCO2D. To illustrate the iteration comparison of original SINCO and SINCO2D we present the following "heat map" plots. In our experiments from the real data (Lymphoma, Arabidopsis, Leukemia) described in Li and Toh [2010] and also pain data by Rish et al. [2010], we solved the problem using both algorithms. The figures 2.6 and 2.7 show sparsity patterns of two matrices obtained by SINCO and SINCO2D algorithms; each dot represents a non-zero element of the inverse covariance matrix; the frequencies of each element is color coded to show the intensity of multiple selection of the same coordinate.

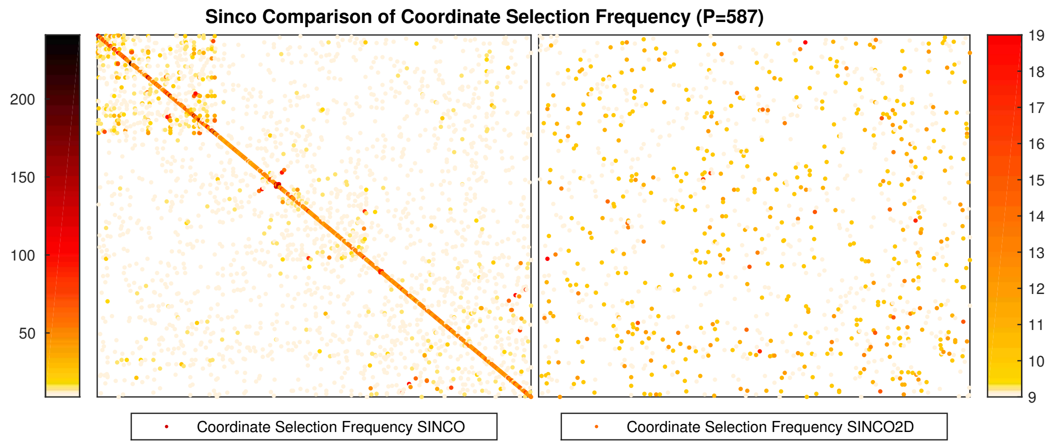


Figure 2.6: Heat map comparison of SINCO and SINCO2D performance on Lymphoma Dataset ($P = 587$): Warmer colors show popularity of coordinates

The left plots in figures 2.6 and 2.7 show the repetition intensity of the selected coordinates in the original SINCO; the color map indicates that the diagonals are among most frequently chosen coordinates (as our discussion in section 2.3.1 suggests). Also there are many single coordinates that engage up to 140 (Arabidopsis) and 230 (Lymphoma) iterations of SINCO. However, the plots on the right show runs with the SINCO2D algorithm. In the figure 2.6, we can see that the repetition is vanished around the diagonals and rarely occurs for the off-diagonal elements as well.

In figure 2.7 ($p = 834$) the colors and frequencies indicate that the heat map is

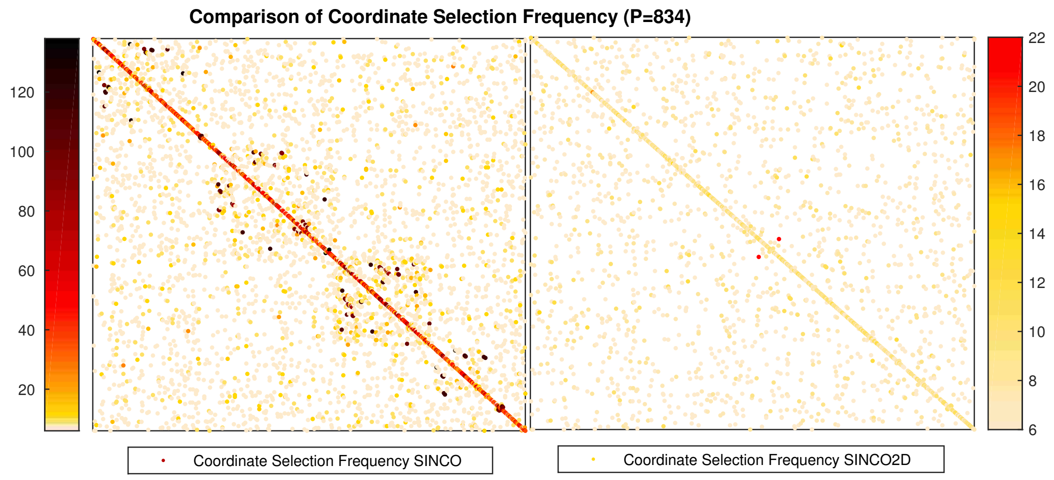
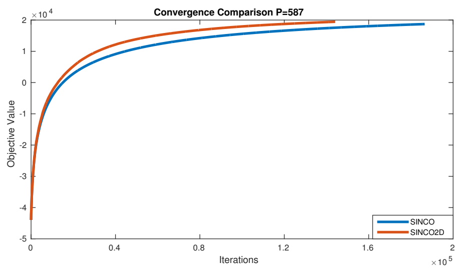


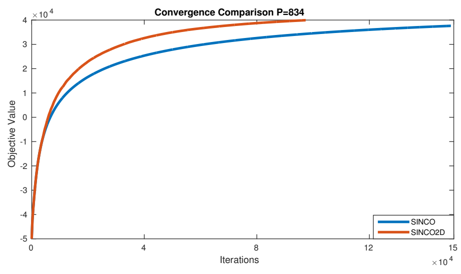
Figure 2.7: Heat map comparison of SINCO and SINCO2D performance on Arabidopsis Dataset ($P = 834$): Warmer colors show popularity of coordinates

"cooled down" around the diagonals and only 1 "hot" coordinate remains, engaging 22 iterations of the algorithm.

Both figures show the intensity reduction of recurrent coordinates is reduced for the selected patterns, which significantly reduces the number of iterations the algorithm makes. The heat maps demonstrate the algorithm taking more mature update steps and preventing unnecessary loops to reach the optimal solution. However, existence of bigger dependency chains, hinders complete removal of recurring coordinates. Further, figure 2.8 shows, eliminating the need for repeated update in coordinates results in faster convergence of the SINCO2D compared to SINCO. Difference performance gap gets intensified as the problem size grows.



(a) Convergence on Lymphoma set $P = 587$



(b) Convergence on Arabidopsis set $P = 834$

Figure 2.8: SINCO2D achieves a faster convergence rate compared to the original counterpart

Table 2.4 compares the results obtained by SINCO2D vs SINCO by measuring the solution times and iteration counts for the mentioned datasets. The results are optimized over blocksize=$\{8, 16, 32\}$ and Ncores=$\{36, 64\}$.

Table 2.4: Run-time and iteration count of SINCO2D vs SINCO

ρ	Size	SINCO2D		SINCO	
		Iteration	Time(s)	Iteration	Time(s)
0.1	587	18201	39.16	20535	43.776
	834	33256	101.923	43275	125.872
	1255	53921	362.302	61563	410.388
0.01	587	159291	536.586	201915	483.712
	834	116799	461.823	172131	698.677
	1255	172274	1476.31	219311	2103.78

The results clearly show that the new algorithm performs better than the old algorithm iteration number; also in almost every case the solution time is improved and gaps get larger with problem size and the density of the matrix. However, as can be seen for a denser solution $p = 587$ the solution time in one instance is higher for the new algorithm which can be assigned to extra constant time computation in smaller sizes, or the rejection of the new steps because of sign discrepancy. The above results are optimized over block size and number of processors, however both quantities have an important role on the solution time.

2.4.3 Sensitivity to Block Size and Parallel Scalability

To illustrate the parallel efficiency of the algorithm we ran the algorithm using 1, 4, 9, and 16 processors. This sequence is chosen as we used a balanced row/column grid topology. However, the grid topology is not restrictive and could be used with different shapes.

Figure 2.9 shows that the actual run-times are slightly better than the expected

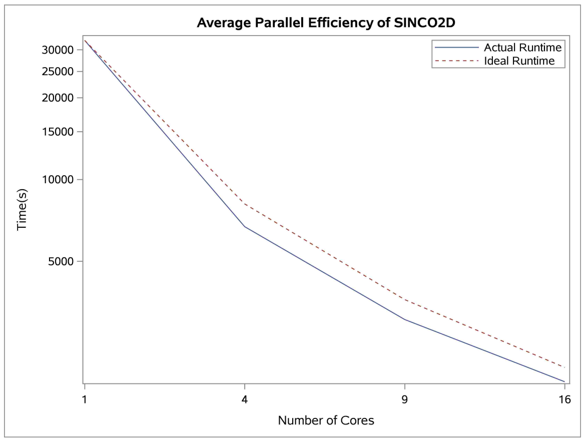


Figure 2.9: Average run-times with different number of cores $p = 1000$

run-time. This can be accounted to maximizing cache hits on each processor and therefore having better speed up factor. The detailed run-time profiles showed that in the ScaLAPACK implemented version the algorithm spends most of its run-time on the search phase. However, for the perfect balancing version the run-times of the search phase and update phase we close to each other. This can be due to the fact that symmetric rank-two updates are optimized in the PBLAS libraries and hence more efficient than the search phase. Two measures can be considered to reduce the time on the search phase. First, one can devise a blocked searching method and utilize BLAS library on each block. Alternatively, one can maintain and reuse information, gathered in the search phase, rather than recomputing it and discarding it at each iteration.

Block cyclic distribution of matrix heavily depends on the local row and column block sizes. This fact in addition of helping with distribution the work load among multiples nodes balanced, it also has an effect on cache utilization locally under each processor. For random graphs of size 500 smaller block size of 8 gives the best

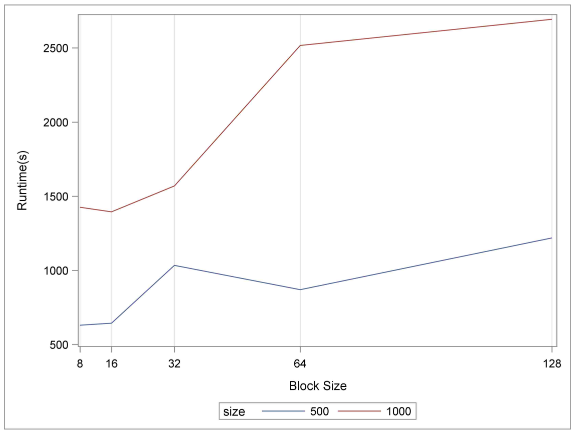


Figure 2.10: The effects of block size on the run-time with 36 processors

results, block size 128 is not efficient in this setting as two rows of processors will be eliminated from the grid. However, as the size grows to 1000, increasing the block size to 16 seems to benefit the speed of the algorithm. Both random tests show that small block sizes benefit the speed of the algorithm with slight difference of run-time for small perturbations.

We also used higher dimensional problem of pain with $p = 5310$. In this problem, as the matrix size gets larger, bigger block sizes of 16 and 36 will be beneficial for balancing cache hits vs efficient workload distribution. This figure also demonstrate the parallel scalability of the algorithm, where from 16 processors to 36, the solution time reduces on average by 45%; however, the scalability from 36 to 64 is much smaller as the gap is a function of problem size.

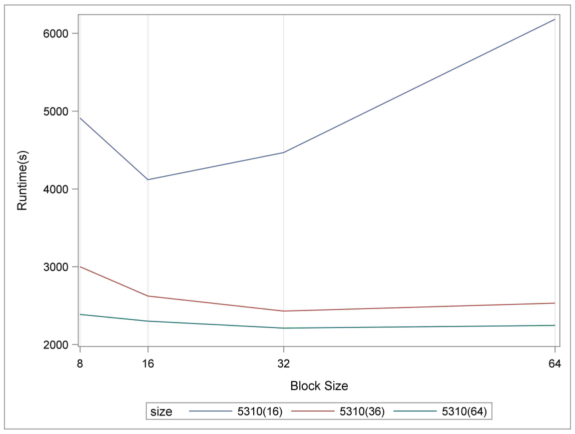


Figure 2.11: The effects of block size on the run-time with $16 - 36 - 64$ processors for pain data $p = 5310$

It is worth mentioning the same pattern on parallel scalability shows between using 16, 36 and 64 processors.

2.5 Conclusion

This study proposes a parallel framework for the SINCO algorithm originally introduced in Scheinberg and Rish [2010]. The main impetus for this algorithm is simplicity of implementation and potential of being massively parallelizable. The new approach for data distribution provides a more efficient implementation of the parallel framework. Further, the algorithm SINCO2D is introduced where the rank-2 updating steps are strengthened by taking into account the structure of semi-definite programs. Numerical examples verify that the new algorithm improves on the convergence rate of SINCO and from practical standpoint, can achieve more accurate solutions compared to other powerful packages such as QUIC.

Chapter 3

Inverse Covariance Selection: Parallel Graphical Lasso

3.1 Introduction

In this chapter, we derive another parallel algorithm for the SICS problem. For this optimization problem, we choose a different coordinate descent algorithm. In the previous chapter, we considered single (SINCO) and augmented version (SINCO2D) coordinate descent algorithms for this problem. However, both cases only consider up to a handful number of individual coordinates. In this chapter, we study the Graphical Lasso (GLasso) – a block coordinate descent algorithm – which updates one row (and one column) of the variable matrix at a time, and which is originally proposed by Banerjee et al. [2008]. The algorithm is further developed by Friedman et al. [2008], where it is proposed to solve the dual of form for the SICS problem

$$\max_{X \succ 0} \log\det X - \langle S, X \rangle - \lambda |X|_1. \tag{3.1}$$

In what follows, first we describe the dual problem derivation and then consider the block coordinate descent GLasso algorithm, developed by Friedman et al. [2008]. The row subproblems considered in GLasso are actually Lasso (Tibshirani [1996]) problems. We study different methods of distributing the data for developing parallel solver based on GLasso. The suitable data distribution that minimizes communication in the parallel setting is chosen and consequently a reformulation of the Lasso problem is proposed to minimize the computational effort with respect to this data distribution.

The reformulated Lasso problem makes it easier to use a parallel method for solving the Lasso problem addressed by Richtárik and Takáč [2013, 2016]. Therefore, at each outer iteration of the algorithm, all subproblems are solved using a parallel Lasso solver. This algorithm becomes essential whenever the size of matrices is too large to fit inside in memory of a single worker.

3.1.1 Analysis of SICS problem

To introduce Lasso approach to SICS problem, we reiterate the formulation and then motivate the use of dual model in derivation of GLasso. Recall that the maximum likelihood estimator, augmented with regularization term, was defined as

$$\max_{X \succ 0} \log\det X - \langle X, S\rangle - \lambda \|X\|_1.$$

As a way to reformulate this problem in a nicer form, we can eliminate the ℓ_1 norm penalty. Since the problem is of maximization form, it can be seen that once an element x_{ij} of matrix X is nonzero, it would contribute $-\lambda|x_ij|$ cost to the objective value. Another way to model such phenomena is through an auxiliary (implicit) variable U.

Consider the inner product $\langle X, U\rangle$; For a given X, we can write another mathematical model, with U as the decision variable, that gathers the contributions of the regularization term.

$$\text{Min } -\langle X, U\rangle,$$

$$\text{s.t}$$

$$-\lambda \le u_{ij} \le \lambda, \ \forall i, j.$$

As a result, for given X, if $x_{ij} < 0$, the optimal corresponding element of U would be $u^*_{ij} = -\lambda$, similarly when $x_{ij} > 0$, then $u^*_{ij} = \lambda$. Therefore the optimal solution of this subproblem will be equal to the regularization term. As a result, the original problem can theoretically be written in a two level optimization problem as:

$$\max_{X \succ 0} \min_{-\lambda \leq u_{ij} \leq \lambda} \log\det X - \langle X, S\rangle - \langle X, U\rangle. \tag{3.2}$$

The bound constraint can be succinctly written as $\|U\|_\infty \leq \lambda$. Next section shows how the concept of dual problem is achieved using this reformulation.

3.1.2 Dual of SICS problem

Banerjee et al. [2008] derived the dual for the SICS problem. By rewriting the problem as

$$\max_{X \succ 0} \min_{\|U\|_\infty \leq \lambda} \log\det X - \langle X, S + U\rangle, \tag{3.3}$$

they show that interchanging the inner and outer problems we find the dual of the SICS problem stated by

$$\min_{\|U\|_\infty \leq \lambda} \max_{X \succ 0} \log\det X - \langle X, S + U\rangle. \tag{3.4}$$

We derive $X = (S + U)^{-1}$ for the solution of inner problem by analytically solving using the first-order condition, resulting in

$$\min_{\|U\|_\infty \leq \lambda} - \log\det(S + U) - p. \tag{3.5}$$

Letting $W = S + U$ we have:

$$\begin{aligned} &\text{Max } \log\det W \\ \text{s.t} \\ &\|W - S\|_\infty \leq \lambda \end{aligned}$$

Friedman et al. [2008] consider a block coordinate descent framework for this problem, which results in the subproblems having a special form. Let us decompose the matrix W by the first row and column: $W = \begin{pmatrix} w_{11} & W_1^T \\ W_1 & W_{\backslash 1 \backslash 1} \end{pmatrix}$ here w_{11} is a scalar, W_1 is a column vector and $W_{\backslash 1\backslash 1}$ is a $p - 1 \times p - 1$ matrix, excluding the first row and column of W; treating $W_{\backslash 1 \backslash 1}$ as a constant and also noting the determinant identity using the Schur complement we have

$$\det W = \det(W_{\backslash 1 \backslash 1})(w_{11} - W_1^T (W_{\backslash 1 \backslash 1})^{-1} W_1). \tag{3.6}$$

We can solve the resulting optimization problem over the first row as follows:

$$\text{Min}_{\beta} \beta^T (W_{\backslash 1 \backslash 1})^{-1} \beta$$

s.t

$$\|\beta - S_1\|_\infty \leq \lambda$$

Friedman et al. [2008] point out that this problem is similar to the dual of a Lasso problem and proposed an algorithm known as Graphical Lasso, which solves the SICS problem by solving Lasso problems at each iteration using coordinate descent.

3.2 Graphical Lasso Algorithm

In order to describe the algorithm first we need to state some properties of the optimal solution. Writing the optimality conditions for the SICS problem (3.1), we get

$$W - S - \lambda\Gamma = 0. \tag{3.7}$$

Sub-gradient equation is written as $\Gamma_{ij} = \text{sgn}(X_{ij})$ whenever $X_{ij} \neq 0$, and $\Gamma_{ij} \in [-1, 1]$ for $X_{ij} = 0$. We also used the fact that $\nabla \log\det X = X^{-1}$ (see Boyd and Vandenberghe [2004]). Since $X \succ 0$, the inequality $X_{ii} > 0$ is ensured for all indices. Therefore $\Gamma_{ii} = 1$ for the diagonal elements, which results in having

$$W_{ii} = S_{ii} + \lambda, \forall i. \tag{3.8}$$

Therefore, the diagonals of the optimal inverse matrix are known. Based on this result, Friedman et al. [2008] propose to start with $W^0 = S + \lambda I$ as the initial solution and proceed by block coordinate descent algorithm updating the columns in a cyclic manner.

Consider the constrained quadratic programming subproblem mentioned above, the dual for this problem is $\min \beta^T W_{\backslash 1 \backslash 1} \beta + S_1^T \beta + \lambda |\beta|_1$. We can achieve this formulation by considering the sub-gradient equation for the first block, which is written as

$$W_1 - S_1 - \lambda \gamma_1 = 0. \tag{3.9}$$

Noting the inverse identity we have: $WX = I$ therefore considering the first column we get $W_1 x_{11} + W_{\backslash 1 \backslash 1} X_1 = 0$ Which yields: $W_1 = -W_{\backslash 1 \backslash 1} \dfrac{X_1}{x_{11}}$, where x_{11} and X_1 are the first diagonal element and the first column of matrix X. For the ease of notation let $\alpha = \dfrac{X_1}{x_{11}}$. Rewriting (3.9) we have:

$$-W_{\backslash 1 \backslash 1} \alpha - S_1 - \lambda \gamma_1 = 0. \tag{3.10}$$

The above is the optimality condition for the following closely related optimization problem,

$$\min_{\beta \in \mathbb{R}^n} \beta^T W_{\backslash 1 \backslash 1} \beta + S_1^T \beta + \lambda |\beta|_1. \tag{3.11}$$

Therefore, we propose to solve the above quadratic problem using a parallel block-coordinate descent algorithm.

3.3 Data Distribution

In order to minimize the communication among the processors we use block column partition of the data. The reason we choose this partition is that when solving the

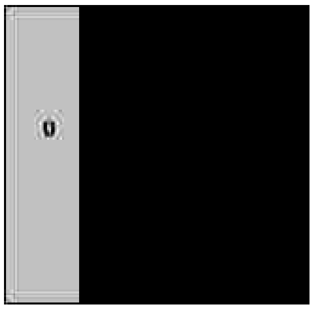


Figure 3.1: Block Column distribution with 4 processors

subproblem (3.11), each processor can only focus on its coordinates locally. In other words, the general Lasso problem defined on the matrix $W_{\backslash 1 \backslash 1}$, is decoupled into smaller subproblems. As we will show in our method no communication is needed between the subproblems until the gradient synchronization is performed every γ iterations. Then, in order to update the gradient, each CPU participates in a reduce step by sending $W_k\bar{\beta}_k$ which corresponds to the local partition of the data.

3.4 Parallel GLasso Algorithm

Our parallel approach to the Lasso subproblems follows the work introduced by Richtárik and Takáč [2013, 2016], however, we address the problem in a different form compared to how it's formed in their definition.
In order to outline the algorithm first we need to define L_λ operator. Consider the (quadratic form of) Lasso problem

$$F(\beta) = \frac{1}{2}\beta^T \bar{W}\beta + \bar{s}^T\beta + \lambda|\beta|_1 \tag{3.12}$$

In order to solve this problem in parallel with τ processors, we solve the following problem in each processor, where subscript p denotes the set of indices of variables assigned to processor p. Each subproblem will be:

$$F(\beta_p) = \frac{\sigma}{2}\beta_p^T \bar{W}_p\beta_p + \bar{s}_p^T\beta_p + \lambda|\beta_p|_1. \tag{3.13}$$

Here σ is a parameter which controls the step sizes taken by each processor. If we set $\sigma = P$ this corresponds to taking small coordinate steps that guarantee decrease of the function value and hence convergence of the algorithm. Smaller values of σ increase the steps and hence improve potential convergence, but may fail to provide safe steps (see Richtárik and Takáč [2013, 2016] for details).
To solve the above problem by coordinate descent method, and update i'th coordinate by δ we have the following subproblem:

$$\frac{1}{2}\bar{W}_{ii}\delta^2 + (s + W\beta)_i\delta + \lambda|\beta_i + \delta|, \tag{3.14}$$

where β is the incumbent solution vector. Representing the above more compactly, for each coordinate we are solving a subproblem of the form: $b\delta^2 + a\delta + \lambda|c + \delta|$. Using optimality conditions we derive the solution to be determined by the following L operator.

$$L_\lambda(a, b, c) = \begin{cases} \frac{-\lambda \text{Sgn}(c - \frac{a}{2b} - \frac{\lambda}{2b}) - a}{2b} & |c - \frac{a}{2b}| > \frac{\lambda}{2b} \\ -c & |c - \frac{a}{2b}| \leq \frac{\lambda}{2b} \end{cases} \tag{3.15}$$

Algorithm 5 Graphical Lasso Algorithm $\bar{W}, \bar{\beta}$

```
1: Let $W = S + \lambda I$
2: while convergence not reached do
3:     for all coordinates $i = 1, \ldots, p$ do
4:         $\beta \leftarrow$ Plasso($\bar{W} = W_{\backslash i \backslash i}, \bar{\beta} = W_{i \backslash i}$)
5:         $W_i \leftarrow W_{ii}\beta$                         ▷ As $W_1 = -W_{\backslash 1 \backslash 1}\alpha$
6:     end for
7: end while
```

Algorithm 6 Plasso: Parallel Lasso solver given $\bar{W}, \bar{\beta}$

```
1: Let $G = \bar{W}\bar{\beta}$
2: while $\|\Delta G\| > \epsilon$ do
3:     for all processors $j = 1, \ldots, \tau$ do
4:         $\Delta G_{(j)} \leftarrow 0$
5:         for $\gamma$ iterations do
6:             Pick coordinate $i \in I_{(j)}$ at random
7:             $\Delta\beta = L_\lambda(G_i + \bar{S}_i + \sigma\Delta G, \frac{\sigma}{2}W_{ii}, \bar{\beta}_i)$
8:             $\Delta G_{(j)} \leftarrow \Delta G_{(j)} + \Delta\beta\bar{W}_{(j)}$
9:         end for
10:    end for
11:    $\Delta G = \sum_j \Delta G_{(j)}$ (MPI_REDUCE)
12:    $G \leftarrow G + \Delta G$
13: end while
```

3.5 Implementation and Experiments

We implemented the above parallel algorithm using ScaLAPACK and C++. Because of our special data distribution, we can use parallel I/O and form the matrices locally on each processor. The important quantity is the parameter σ which plays a crucial role in having a convergent sequence of iterates as well as in the speed of convergence. As mentioned above σ controls local coordinate step-length; thus if it is larger the steps become smaller, so that every processor is making safe but slow progress towards optimum. Richtárik and Takáč [2013] show that, without considering special structures K the number of processors is an upper bound for σ. Therefore, we can safely implement the code with $\sigma = K$.

In order to see the performance of the algorithm we ran experiments using data sets with $p = 587, 690, 834, 1255, 2846$, corresponding to *Lymphoma, Pain1, Arabidopsis, Leukemia, Pain9* data sets. We used $\gamma = 1000$, i.e. we make processors synchronize the gradients every 1000 coordinate updates. We use duality gap tolerance of 10^{-6} for all experiments. We present the results in Tables 3.1 and 3.2, where we list the times, in seconds, it takes for a given algorithm to solve a given problem. The first 5 columns show the results for parallel Graphical Lasso for 5 different values of σ. Table 3.1 shows the results for sparse matrices, controlled by setting $\lambda = 0.5$

Table 3.1: Comparison of solution times (s) $\lambda = 0.5$, $k = 16$ cores

	$\sigma = 3$	$\sigma = 4$	$\sigma = 5$	$\sigma = 10$	$\sigma = 16$	**Glasso**	**QUIC**
587	6.15	8.05	9.96	18.63	29.609	5.34	2.09
690	17.25	23.18	28.45	50.366	84.23	5.57	3.26
834	22.395	30.33	37.87	73.43	126.92	67.76	11.9
1255	52.19	70.10	86.86	189.52	290.28	243.95	42.94
2846	249.865	378.2	334.65	654.82	1053.86	4714.12	235.2

We can see that the parallel algorithm with original the upper-bound for σ can only achieve better results compared to sequential graphical lasso algorithm (Friedman et al. [2008]) for the last problem $p = 2846$. Moreover, when the step-sizes are longer, i.e. $\sigma < k$ we can get comparable results to the sequential algorithm. However, the second-order algorithm QUIC (Hsieh et al. [2011]) achieves better run-times.

In Table 3.2 we show the results on dense matrices by setting $\lambda = 0.1$. For the

Table 3.2: Comparison of solution times (s) $\lambda = 0.1$, $k = 16$ cores

	$\sigma = 3$	$\sigma = 4$	$\sigma = 5$	$\sigma = 10$	$\sigma = 16$	**Glasso**	**QUIC**
587	34.6	47.02	58.67	114.56	180.53	96.16	53.72
690	-	56.83	71.22	139.05	213.95	91.81	77.13
834	61.26	82.49	103.56	204.07	317.76	415.3	221.83
1255	117.598	158.35	199.45	395.25	614.45	1375.06	668.51
2846	-	761.05	926.61	1710.84	2616.70	7748.79	9623.51

safe σ we can see that when the size grows larger the algorithm can outperform QUIC. Also when the σ is reduced for smaller size problems also the solution time is better than Glasso and QUIC. However, for $\sigma = 3$, two of the problems diverged and could not achieve a solution. Therefore, determining σ dynamically for this problem can be good approach to this issue.

3.6 Conclusion and Future Directions

As our implementation and numerical results indicate for large-scale problems with 8 million coordinates the safest σ parameter also produces a considerable reduction time compared to the GLasso algorithm when the solution remains sparse, however the second-order algorithm QUIC is dominant in this case. As the sparsity reduces and the solutions get more dense our coordinate descent method remains dominant in solution even with safest $\sigma = 16$ settings, compared to sequential algorithm as well as second-order algorithm QUIC.

As the experiments show the lower σ parameters can converge in most cases (except $\sigma = 3$) where in some cases the algorithm diverges due to non-safe step lengths for the update of coordinates; therefore, approaches that can reduce the safety parameter σ using dynamic information gather during runtime can significantly.

Chapter 4

Deep Learning: Large-scale Conjugate Gradient Algorithms

In this chapter, we explore the parallel algorithms developed for one of the prominent and useful areas of machine learning, namely deep learning networks. First, we briefly describe the neural networks (NN) and their use. Next, we proceed by introducing different training (optimization) algorithms devised for this problems.

In particular, this chapter introduces and studies several known second-order methods and their applications to large-scale deep learning problems. The more popular of these methods, so far, use convexified second-order approximation (such as Gauss-Newton method) of the nonconvex objective arising in Deep Learning. Here, we argue that exact nonconvex approximations can produce better quality solutions. However, they result in more complex subproblems, which may require a larger number of Hessian-vector products, which are very expensive in the context of deep learning problems. Hence, in this chapter, we focus on a particular method - modified conjugate gradient - aimed specifically at nonconvex problems and we introduce warm-starting strategies for the subproblem and provide analytical proofs of convergence under the resulting new update steps. Our numerical results show that this warm-started variant of modified conjugate gradient algorithm indeed requires much fewer Hessian-vector products compared to cases where warm-start is not used in the algorithm and also compared to convexified counterparts.

4.1 Introduction

With the advent of higher computational power in computing grids and the introduction of GPU computing, the costly task of training a neural network has become more accessible than the early days of its introduction. This computational advantage enables researchers to stack a higher number of layers in order to create wider and more powerful networks mostly known as Deep Neural Networks (DNN).

DNNs play an important role among various tools used for exploring the huge amount of data in different learning problems. They have proven useful in both supervised and unsupervised learning problems. In the supervised learning case, it is assumed that there exists an unknown underlying function governing the data points (x_i, y_i), which maps the inputs/features x_i to the response/label variable (y_i); when such functions have complex structures, simple predictors such as linear or kernel based learners cannot capture the true behavior and may produce inaccurate models; hence, deep neural networks are preferred. In such cases, the task of the network is generally finding implicit representations of the underlying function as a collection of simple mathematical operators, which can serve as classifiers/predictors with a close approximation to the real function. In the unsupervised learning cases networks aim to find patterns in the data or generate compressed representations of the data (Hinton and Salakhutdinov [2006]); such networks are known as auto-encoders.

The task of learning in both cases is the same from the optimization viewpoint, however, the functions being optimized might differ based on the network's task and also the problem structure. Throughout this chapter, we refer to this quantity as loss or more specifically training error. To measure the quality of a trained network, researchers leave a fraction of data to remain unseen by the network; this measure can quantify how the networks predictions generalize beyond the training dataset; we will refer to this quantity as the validation error.

As an example of a complex supervised learning problem, the popular MNIST dataset (LeCun et al. [1998]) consisting the pixel data of 70,000 handwritten digits between 0–9 has drawn much attention from researchers. For the classification task, the aim is to find classifiers for handwriting recognition; this can be done by learning compact representations of such data in order to teach the machines to distinguish between the handwritten scripts (Hinton and Salakhutdinov [2006]). The validation errors of these networks show that they can learn such a sophisticated underlying function accurately. It is shown that both fully connected neural networks and convolutional networks can achieve low misclassification error rate (%0.95) on original the MNIST dataset (LeCun et al. [1998]), quite easily without manipulation of data.

Figure 4.1: Samples images from the MNIST data composed of 28×28 pixels

Performance of neural network and ability to dominate other known methods on the MNIST dataset and also other well-known datasets such as CIFAR and ImageNet, have resulted in the unprecedented popularity of neural networks as an approach to learning in computer vision tasks. However, the extent of deep learning research is not limited to this domain and there are various applications of such networks in Natural Language Processing (NLP) and text analytics (Sutskever et al. [2014], Zhang et al. [2015]), speech recognition (Hinton et al. [2012]) and Reinforcement Learning (RL) (Mnih et al. [2015]) among other fields. Deep neural networks have an advantage by being able to implicitly create complex (classifier) functions may belong to the family of functions not explored by other methods; for example, support vector machine based classifiers, are typically limited to linear, quadratic or other predefined classes of functions belonging to specific function families. This flexibility enables DNNs to serve as an inter-disciplinary tool in the mentioned fields.

Neural networks usually consist of input, hidden and output layers. The hidden segment may include one or multiple layers. Each layer is comprised of multiple neurons; in most cases, the input layer neurons correspond to the input variables (x)

or namely features, and the corresponding labels (y) would define the structure of the output layer; therefore, both quantities are determined by the problem's structure. The number of layers in the hidden segment and also the number of neurons in each of these layers are hyper-parameters that are defined by the user. Each layer is connected to the other layers with a set of weights; therefore, performing a scaling on the outputs of the previous layer before feeding to the next.

To capture more sophisticated families of underlying functions, each neuron applies a nonlinear function to its input known as the activation function. For each neuron, these functions map the input (i.e. the output from neurons of the previous layer) to a scalar value, which is the output of the corresponding neuron. Different choices of the nonlinearity function such as identity, sigmoid, tanh, among others, make this specification a hyper-parameter chosen at the beginning. In order to present how these networks behave and also the role of neurons, nonlinearity functions, and different layers, we first present some notations and definitions.

4.1.1 Definition

To follow notation of Martens and Sutskever [2012], we can denote a neural network as $f(x, w)$, where x is the input to network and w represents the set of weight matrices W_i and bias vectors $b_i, \forall i = 1, \ldots, L$. Assuming that the network only consists of fully connected layers, for any two neurons in consecutive layers a weight w_{ij} carries the output from the previous layer's neuron i to the neuron j in the following layer. Denoting the number of neurons in the i'th layer by n_i, the main equation for propagating information between layers and thus through the network will be

$$s_i = W_i^T a_{i-1} + b_i, \tag{4.1}$$

where s_i shows the input for the i'th layer, and $W_i \in \mathbb{R}^{n_{i-1} \times n_i}$ is the weight matrix carrying the outputs a_{i-1} of previous layer to i'th layer. The vector $b_i \in \mathbb{R}^n$ is the constant bias associated with the layer i. Each layer is associated with a nonlinearity $\phi(s)$ to compute activations

$$a_i = \phi_i(s_i). \tag{4.2}$$

Function ϕ_i often is a nonlinear function corresponding to the i'th layer, in most cases this function aims to mimic certain behaviors of brain's neurons by firing at a certain threshold (ReLU function), or can induce stability to numbers by clipping the s_i back to $(-1, 1)$ range (Hyperbolic tangent). Here, we assume that the function $\phi_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ is a vectorized function applying ϕ to each element of s_i individually; therefore, as hinted above function ϕ_i is usually defined as the vectorized ReLU function or the sigmoid and tanh function. However, there are other activation functions explored in the literature as well showing better results (Xu et al. [2015]).

Using these nonlinearity functions enables the network to learn a close representation of the true function mapping the input x to the outputs y. Once the number of layers L and neurons $n_i, i = 1, \ldots, L$ and also their nonlinearity functions ϕ_i are determined by the user, the task of training the network amounts to fine-tuning the weights $w = \{(W_i, b_i), i = 1, \ldots, L\}$ in order to minimize the difference of network output and the desired observation.

Specifically, the performance of the network, given the training/validation observations, is measured using a loss function as

$$h(w) = \frac{1}{|S|} \sum_{(x,y)\in S} L(y, f(x, w)). \tag{4.3}$$

The loss function is defined as the sum of individual losses of each training observation $(x, y) \in S$. As a result, the gradient of this function is computed by summing the gradients for individual losses

$$\nabla h(w) = \frac{1}{|S|} \sum_{(x,y)\in S} \nabla L(y, f(x, w)). \tag{4.4}$$

Similar behavior governs the second-order information. Based on the intrinsic separability of the loss function, most studies use data parallelsim vs model parallelization which can involve higher communication frequency among worker nodes; as a result, the data is distributed among nodes and communication will occur on the end results of each node. This communication in most cases is regarding quantities such as loss function evaluation, derivatives and higher-order curvature information with respect to each worker's share of data. Therefore, individual scoring of the data points is done on each processor and gathering of data includes reduce calls with sum operator.

The training task involves use of derivative information to provide the optimizer means for reducing the loss with respect to divergence of network output for each training points. Since the function needs to be computed for every point in order to get gradient information, this task can be time consuming. Hence, use of second-order methods may seem computationally prohibitive; however, it is possible that utilizing computational grids as tools for performing distributed computation along worker nodes, one obtains necessary information for optimization using higher-order information such as curvature information.

It is worth mentioning that due to the mentioned considerations, many studies just use the first-order information for training the network. As the loss function is separable for data points, online algorithms such as Stochastic Gradient Descent (SGD) have gained much attention. Also whenever a new observation is made we can easily update the gradient and update the weights in the network by a cheap iteration of SGD.

In order to compute the loss associated with the network, first we need to get $\hat{y} = f(x, \theta)$. This quantity can be calculated by traversing the network from input layer to output layer for the observation x. However, as the variables are matrices (W_i, b_i) $\forall i$, and each layer adds nonlinearity ϕ to its input, computing the closed-form gradient of $f(x, \theta)$ for each observation is not trivial, an algorithm known as back-propagation of Le Cun et al. [1988] described in Algorithm 7 (as outlined by Martens and Sutskever [2012]) helps computing this quantity as systematic routine of applying derivative rules. The algorithm implements simple derivative rules such as the chain rule and summation rule at each layer in order to propagate the error through the network.

Having numerous neurons and layers results in the huge number of weights for the network, making the size of the optimization problem grow significantly. The only practical way to solve the loss minimization problem is through numerical methods using first and second-order information such as SGD and Hessian-Free approaches.

Algorithm 7 Back-Propagation Algorithm

```
1: Let $a_0 = x$
2: for $i = 1, \dots, l$ do
3:     $s_i = W_i^T a_i + b_i$
4:     $a_i = \phi_i(s_i)$
5: end for
6: $\delta_l = \nabla_{a_l} L \odot \phi'(s_l)$
7: for $i = l, \dots, 1$ do
8:     $\frac{\partial L}{\partial W_i} = \delta_i a_{i-1}^T$
9:     $\frac{\partial L}{\partial b_i} = \delta_i$
10:    $\delta_{i-1} = (W_i \delta_i) \odot \phi'(s_{i-1})$
11: end for
```

When training the network, the gradients can be efficiently computed using back-propagation which facilitates the first-order methods. Studies have shown that one also can use the power of second-order information for this problems. In order to use the second-order information, explicit representations of the Hessian matrix are deemed impractical. Also since most numerical algorithms such as Conjugate Gradient (CG), require only Hessian-vector multiplications results, having a method to compute this product would be sufficient for the optimization goal. Therefore, Pearlmutter [1994] derived the well known R-operator in order to extend the back-propagation formulations for computing gradients to compute Hessian-vector multiplications.

4.1.2 Deriving R-Operator for Second-order Information

In order to derive a second-order algorithm for this problem we can use the method introduced in Pearlmutter [1994]. Pearlmutter observed that Hessian-vector (Hv) multiplication can be simply viewed as a directional derivative of gradient with respect to direction (v)as

$$\mathcal{H}v = \lim_{r \to 0} \frac{\nabla_w(w + rv) - \nabla_w(w)}{r} = \frac{\partial}{\partial r} \nabla_w(w + rv) \Big|_{r=0}. \tag{4.5}$$

He defined R-operator to simplify the notation with regards to this quantity as

$$\mathcal{R}_v\{f(w)\} = \frac{\partial}{\partial r} f(w + rv)|_{r=0}. \tag{4.6}$$

As this operator is a derivative, one can show that there are certain rules that it follows:

$$\begin{aligned}
\mathcal{R}_v\{af(w) + bg(w)\} &= a\mathcal{R}_v\{f(w)\} + b\mathcal{R}_v\{g(w)\}, & (4.7)\\
\mathcal{R}_v\{f(w)g(w)\} &= \mathcal{R}_v\{f(w)\}g(w) + \mathcal{R}_v\{g(w)\}f(w), & (4.8)\\
\mathcal{R}_v\{f(g(w))\} &= f'(g(w))\mathcal{R}_v\{g(w)\}, & (4.9)\\
\mathcal{R}_v\{w\} &= v. & (4.10)
\end{aligned}$$

Noting that ($\mathcal{H}v = R_v\{\nabla_w\}$), he showed that Hessian-vector multiplication can be derived by applying the R-operator to the gradient computation method. In other words, we can apply this operator to the back-propagation equations to compute $R_v\{\nabla_w\}$.

Algorithm 8 Hessian-vector product algorithm

```
1:  Given $v$ let $RW_i, Rb_i$ equal to corresponding components of $v$        ▷ (4.10)
2:  $Ra_0 \leftarrow 0$                         ▷ directional derivative with respect to input
3:  for $i = 1, \dots, l$ do
4:      $Rs_i = RW_i^T a_i + W_i^T Ra_i + Rb_i$
5:      $Ra_i = Rs_i \phi_i'(s_i)$
6:  end for
7:  $R\delta_l = (\nabla_{a_l}^2 L) Ra_l \odot \phi'(s_l) + ((W_l \delta_l) \odot \phi''(s_i) \odot s_i)$
8:  for $i = l, \dots, 1$ do
9:      $Hv_{[W_i]} = R\{\frac{\partial L}{\partial W_i}\} = R\delta_i a_{i-1}^T + \delta_i Ra_{i-1}^T$
10:     $Hv_{[b_i]} = R\{\frac{\partial L}{\partial b_i}\} = R\delta_i$
11:     $RDa_i = RW_i \delta_i + W_i R\delta_i$
12:     $R\delta_{i-1} = Rda_i \odot \phi'(s_i) + (W_i \delta_i) \odot \phi''(s_i) \odot Rs_i$
13: end for
```

Algorithm 8 shows the back-propagation method with R-operator applied to the equations (as outlined by Martens and Sutskever [2012]). Applying the above algorithm at each iteration we can get the Hessian-vector multiplication and feed it to the CG algorithm.

4.1.3 Algorithms for Training Neural Networks

Previous section discusses how obtaining first and second-order information regarding the training loss function is managed through use of back-propagation algorithm and its extension to Hessian-vector products. Starting from an initial (advised) random guess x_0, numerical optimization methods heavily rely on such information to improve the initial solution through iterates of sequential updates governed by variants of equation (4.11) and consequently progress towards better solutions which ideally converges to optimal stationary points.

$$x_{k+1} = x_k + s_k \tag{4.11}$$

Devising a good initial guess has great importance for converging to a good trained model; a suitable initial guess will produce a network that can learn by minimizes effects of vanishing/exploding gradients and dead neurons (Sutskever et al. [2013]); otherwise, gradient information would be useless by having minimal effects in the training procedure. Now since the optimization variables (weights/biases of the network) are defined for each layer, and the network output is the result of input data traversing this network, this will result in having multiple nested multiplications of these optimization variables among layers; moreover as the back-propagation algorithm outlines the gradients would also get affected because of their nested dependencies on *delta*; therefore, if neurons are initialized with non-suitable initial values, their interaction can result in relatively small gradients or big gradients. Propagating such quantities through the network by a series of multiplications of small/big numbers this can easily lead to numerically unstable (vanishing/exploding) gradients towards the last layers of the network. Such methods are subject of research into design and architecture of different networks/activations combinations. However, utilizing a good initialization strategy, the randomness effects are minimized in the final trained model compared to different strategies.

The following section discusses different family of optimization algorithms utilizing the first and second-order information to generate a sequence of solutions leading to a well trained model.

Algorithms using first-order information

There are different choices for the optimization algorithms, in this section we summarize some of the methods applied in practice. Same as any other optimization problem, there are multiple choices for minimizing the objective training function. Here, we describe the main approaches utilized in the deep learning community which use first-order information regarding the objective function.

- Steepest-Descent: The motivation for this method is to minimize a first-order Taylor approximation around the iterate x_k to produce the next iterate x_{k+1}. The direction $s_k = -\nabla f(x_k)$ is the steepest descent direction this is the solution of minimizing the following:

$$\min_{\|s_k\| \leq 1} f(x_k + s_k) \simeq \nabla f(x_k)^T s_k + f(x_k).$$

  In practice, this strategy is best used along with a line-search framework to determine how long the step size α_k would be along the direction s_k.

$$x_{k+1} = x_k + \alpha_k s_k.$$

- Stochastic Gradient Descent (SGD): Since the loss function (4.3) is sum of losses for all the training samples, gradient of the total loss is therefore sum of gradients for all sample training points. This means that obtaining the gradient depends on running Algorithm 7 for all sample training points. Arguing against the cost of such computation a family of steepest descent based algorithms were devised that rely on stochastic evaluation of the gradient (4.4) by choosing a batch sample of size b and computing

$$\nabla h(w) \approx \frac{1}{|b|} \sum_{(x,y) \in b} \nabla L(y, f(x,w)). \tag{4.12}$$

  Since the direction obtained is an approximation to the descent direction, it may not actually guarantee descent in the true loss. Most SGD algorithms take a safe predetermined step-size α_k rather than performing a line-search along this direction. This step-size is often referred to as "learning rate" of SGD. As the algorithm is not processing all the data points and also does not perform any line-search step, this algorithm inherently involves cheaper computation per update compared to other methods. However, since the directions s_k are approximations of descent directions, updates are less powerful and therefore the total number of updates would be considerably higher.

- L-BFGS: This optimization approach also relies on the first-order information at each iterate (Nocedal and Wright [2006]). This algorithm minimize a quadratic model around the iterate x_k as:

$$m_k(s_k) = \frac{1}{2} s_k^T B_k s_k + \nabla f(x_k)^T s_k + f(x_k). \tag{4.13}$$

  The curvature matrix B_k is updated at each iteration by the secant equations involving the consecutive gradients, in order to capture the curvature of the true optimization function. Again a line-search is performed on the obtained direction s_k.

Next we would discuss algorithms that use second-order information such as Hessian or Gauss-Newton curvature approximations for B_k instead of L-BFGS approximation.

Second-order Algorithms

In this section we briefly review the algorithms for training the network using second-order information. Similar to traditional methods, at each iterate k, using local information one builds a quadratic model m_k around the incumbent solution x_k and minimizes the model

$$m_k(s) = \frac{1}{2} s_k^T B_k s_k + \nabla f(x_k)^T s_k + f(x_k) \tag{4.14}$$

to find an update direction. Basic method of Newton minimizes the above model where the solution is

$$s_k = -B_k^{-1} \nabla f(x_k) \tag{4.15}$$

for nonsingular B_k. However, as matrix B may be indefinite, the nonconvex model makes the above minimization problem unbounded. Therefore, we need safe guards the model to generate meaningful solutions; these safeguards are also important in convex case because in highly nonlinear settings, the quality of the second-order Taylor approximation model would decline as $\|s_k\|$ grows large. Therefore, the quadratic model may not be a good representer of the true function for the points that are further from current iterate x_k; to overcome this issue, trust-region algorithms have been developed to reflect the extent of optimizer's trust in the current quadratic model in mimicking the true underlying function.

4.2 Trust-region Minimization

To extract meaningful updates from the second-order method, we need to conform the solution to a region of trust where the model is more accurate. Methods safeguarding the quadratic subproblem by enforcing the region constraint are known as trust-region methods. In other words the quadratic model will be minimized in a region where it is trusted to produce a good approximation to the true underlying nonlinear function. Therefore, the optimization problem in the trust-region framework is generally defined as ball constrained region around the current iterate x_k with radius Δ as

$$\begin{array}{ll} \min_{s_k} & m_k(s) : \frac{1}{2} s_k^T B_k s_k + \nabla f(x_k)^T s_k, \\ \text{s.t.} & \|s_k\| \leq \Delta \end{array} \tag{4.16}$$

Starting from an initial guess $\Delta^{(0)}$, parameter Δ will be dynamically updated throughout the iterates k using the feedback we get from ratio of model reduction versus the true reduction in the objective function. This ratio will be monitored as the algorithm progresses; this essentially enables us to expand or shrink the trust-region radius based on accuracy of the model. Algorithm 9 outlines the general framework of trust-region methods.

Algorithm 9 Trust-region Method

```
1: Given $\Delta_0 \leq \bar{\Delta}$ and $\eta \in [0, \frac{1}{4})$
2: for $k = 0, 1, \ldots$ do
3:     Solve problem (4.16) to obtain $s_k$
4:     let $\phi_k = \dfrac{f(x_k) - f(x + s_k)}{m_k(0) - m_k(s_k)}$
5:     if $\phi_k < \frac{1}{4}$ then
6:         $\Delta_{k+1} = \frac{1}{4}\Delta_k$
7:     else
8:         if $\phi_k > \frac{3}{4}$ and $\|s_k\| = \Delta_k$ then
9:             $\Delta_{k+1} = \min(2\Delta_k, \bar{\Delta})$
10:        else
11:            $\Delta_{k+1} = \Delta_k$
12:        end if
13:    end if
14:    if $\phi_k > \eta$ then
15:        $x_{k+1} = x_k + s_k$
16:    else
17:        $x_{k+1} = x_k$
18:    end if
19: end for
```

Algorithm 9 describes the the outer loop of the minimization method with dynamic updates to the radius. Step 3 of the algorithm involves solution of the constrained quadratic model (4.16) which we refer to as subproblem. Clearly the heavy computation of this algorithm involves the solution of quadratic subproblem.

For the construction of new algorithms and also providing more background on the topic of trust-region subproblems, we outline the optimality conditions subproblem described in equation (4.16). The following theorem describes optimality conditions of the model subproblem, using these conditions we can identify solutions providing optimal model reduction.

Theorem 4.2.1. *A vector s^* is a global solution of the trust-region subproblem* (4.16) *if there exists a nonnegative scalar σ^* such that*

$$\begin{aligned} (B+\sigma^* I)s^* &= -g, \quad \sigma^*(\Delta - \|s^*\|) = 0, && (4.17)\\ (B+\sigma^* I) &\succeq 0, \quad \|s^*\| \le \Delta, \quad \sigma^* \ge 0, && (4.18)\end{aligned}$$

are satisfied for the pair (s^*, σ^*) (Nocedal and Wright [2006]).

Here σ^* denotes the corresponding Lagrange multiplier for the trust-region constraint $\|s\| \le \Delta$. In the following we will use an asterisk to denote the exact solution of a problem. These results are well known due to Moré and Sorensen [1983] where they proposed methods to exactly solve the above problem.

There are several methods for solving the trust-region subproblem. First, we discuss the exact method of Moré and Sorensen [1983] as this method provides a deeper look into this problem. We will use this method to exactly solve the small dimensional subproblems in the next chapter. We also provide background on other methods involving cheaper inexact methods for solving these subproblems.

4.2.1 Finding Exact Model Minimizer

Based on the complementarity constraint $\sigma^*(\Delta - \|s\|) = 0$, we can see that either the Lagrange multiplier $\sigma^* = 0$ and therefore $\|s\| \le \Delta$ or $\sigma^* > 0$ which implies that the solution lies on the boundary of the trust-region $\|s\| = \Delta$. Hence, if the multiplier is zero, it is logical to check whether the solution to

$$Bs = -g,$$

satisfies the constraint $\|s\| \le \Delta$. If this is the case and $B \succeq 0$, then we have found the global minimizer of the model based on the optimality conditions. Otherwise, one needs to find $\sigma > 0$, satisfying $B + \sigma I \succeq 0$ so that the boundary solution ($\|s\| = \Delta$) satisfies the first-order condition

$$s_\sigma = -(B+\sigma I)^{-1} g.$$

The above problem can be deemed as a root finding problem on variable σ. We need to elaborate a complication that may arise in this root finding problem. Letting the

$B = V\Lambda V^T$ denote the corresponding spectral decomposition where $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$, we can rewrite the vector s_σ as:

$$s_\sigma = -V(\Lambda + \sigma I)^{-1} V^T g = -\sum_{l=1}^{n} \frac{v_l^T v}{\lambda_j + \sigma} v_j,$$

which enables us to determine the length of this vector as

$$\|s_\sigma\|^2 = \sum_{l=1}^{n} \frac{(v_j^T g)^2}{(\lambda_j + \sigma)^2}.$$

Assuming $v_j^T g \neq 0$ we can see that length of vector s_σ diverges as $\sigma \to -\lambda_j$, however as optimality conditions require $B + \sigma I \succeq 0$, we are interested in the interval $\sigma \in (-\lambda_1, \infty)$. As the above function is monotonically decreasing as σ grows in $\sigma \in (-\lambda_1, \infty)$. If $v_1^T g \neq 0$ then the length would decrease from ∞ to 0, therefore, we are certain that there exists $\sigma \geq -\lambda_1$ where $\|s_\sigma\| = \Delta$, in the case where $v_1^T q = 0$ this condition may not be satisfied easily, this situation is referred to as the "hard case", we later will discuss how solution to this case would be obtained. If we are not in the hard case, the root finding would simply mean finding the σ which satisfies $\|s_\sigma\| = \Delta$. Moré and Sorensen [1983] reformulated the root finding problem to a well behaved function as:

$$\frac{1}{\|s_\sigma\|} - \frac{1}{\Delta} = 0$$

This function behaves nicely compared to the previous counterpart, and therefore is more suitable for applying Newton's root finding approach.

In summary their exact algorithm tries to find solutions to the above root finding problem using the Cholesky factorizations to solve the linear system of equations and ensuring positive-semidefiniteness of the shifted system.

Algorithm 10, with safeguards for $\sigma_i \leq -\lambda_1$ can achieve the true solution of the root finding problem and produce the corresponding s_σ vector as the solution of the optimality conditions. The above loop usually takes 2 or 3 iterations to converge in different cases (Nocedal and Wright [2006]). Note that we assumed $v_1^T g \neq 0$ which ensures that the root finding problem has a solution satisfying $\sigma \in (-\lambda_1, \infty)$,

Algorithm 10 Root finding algorithm

```
1: Given $\sigma_0$
2: for $i = 0, 1, \ldots$ do
3:     Factorize $B + \sigma I = R^T R$
4:     Solve $R^T R s_i = -g$
5:     Solve $R^T q_i = s_i$
6:     Update $\sigma^{i+1} = \sigma^i + (\frac{\|s_i\|}{\|q_i\|})^2(\frac{\|s\| - \Delta}{\Delta})$
7: end for
```

$\|s_\sigma\| = \Delta$. This is the case if at least one of the eigenvectors corresponding to the smallest eigenvalue is not perpendicular to the gradient. However, for the hard case where $v_1^T g = 0$, since the limit $\lim_{\sigma \to -\lambda_1} \|s_\sigma\| = \infty$ does not hold, it is possible that for values $-\lambda_1 \geq \sigma$ there is no possible solution for $\|s_\sigma\| = \Delta$. In this case, the authors show $\sigma = -\lambda_1$ and the solution has the form

$$s_\sigma = \sum_{i:\lambda_i \neq \lambda_1} \frac{v_j^T g}{\lambda_j + \sigma} v_j + \tau \zeta,$$

where ζ is the eigenvector corresponding to eigenvalue λ_1. This setting makes it possible to satisfy the boundary constraint $\|s\| = \Delta$ by finding the correct τ. Therefore, the hard case is much more computationally involved.

4.2.2 Solving Trust-region Subproblem Inexactly

The exact method of solving the trust-region subproblem described above has computational bottlenecks, which renders it impractical for cases with large number of variables as is the case for deep learning problems. Moreover, this method requires full representation of the curvature matrix for the factorization procedure, which is practically infeasible to obtain for the problems we are considering. However the exact method will be useful for small subproblems, as is the case in the next chapter where we form small reduced subproblems that are computationally cheap for solved using the exact method.

There are cheaper methods to solve the trust-region subproblem compared to exact methods. Different strategies can be used to achieve an approximate solution for the subproblem. These methods are computationally cheaper than the exact method which involves Cholesky factorizations on the iterates.

Cauchy Point Method

First method is the Cauchy point mehtod. This method essentially solves the the trust-region subproblem by constraining the solution to live on a 1-dimensional subspace generated by the steepest descent direction $-g_k$.

$$s_k = \arg\min_{\tau \geq 0} m_k(-\tau g_k) \text{ s.t. } \|\tau g_k\| \leq \Delta_k$$

Dogleg Method

This method is more general than the Cauchy point method as it considers minimization on a path generated by steepest descent minimizer direction of model (s^U) and Newton's directions s^N of the unconstrained model. This method is appropriate when model's curvature matrix is positive-definite. These vectors are defined as

$$s^U = -\frac{g^T g}{g^T B g} g, s^P = -B^{-1} g$$

The solution $s_k(\tau)$ will lie on the path generated by these directions subject to trust-region bound.

$$s(\tau) = \begin{cases} \tau\ s^U, & 0 \leq \tau \leq 1 \\ \\ s^U + (\tau - 1)(s^N - s^U), & 1 \leq \tau \leq 2 \end{cases}$$

Two Dimensional Subspace Method

This method is another step in generalizing the dogleg method. The dogleg method restricts the solution to the path generated by s^u followed by s^B, however we can widen the search by considering the full space spanned by these directions, that is $p \in \text{span} g, B^{-1} g$ for subproblem (4.16)

This method is more general than previous methods which can potentially lead to better model minimizers. Building on top of these methods, we can get more sophisticated methods minimizing on subspaces spanned by more than two dimensions. Methods based on Krylov subspaces can generate solutions that more accurate and also more practical in large-scale settings.

Iterative Methods for Subspace Minimization

One of the popular iterative methods for solving the trust-region subproblems are conjugate gradient methods. These methods work on subspaces and at each iteration minimize the model on the incumbent subspace, while adding additional dimension to the subspace for the next iterates. These methods have gained much popularity among deep learning researchers for particular reasons that are inherent to the large size of these problems. The main aspect that makes these algorithms desirable is their ability to optimize the function in a matrix-free framework, i.e. the full representation of the matrix B is not required, and only the multiplication Bv for an arbitrary vector v is needed. Using the R-operator discussed in 4.1.2 we can modify the back-propagation algorithm to find the Hessian-vector multiplication as illustrated in Algorithm 8.

One issue arising in using the traditional CG methods is the assumption of $B \succeq 0$. However, by construction the loss function of the deep learning problems is not necessarily convex and therefore its Hessian is not necessarily positive-semidefinite. Fortunately, there are variants of nonconvex CG methods such as Steihaug [1983], Toint [1981] or GLTR by Gould et al. [1999] for this problem, however there are some considerations with each of these algorithms which should be discussed.

In order to overcome the nonconvexity issue, Martens [2010] proposed using a positive-semidefinite approximation to the Hessian. He proposed using the generalized Gauss-Newton (GN) curvature matrix as an approximation of the Hessian which can be used with a CG solver. Originally, Schraudolph [2002] extended the Gauss-Newton curvature application from least squares problem to neural networks. Similar to Hessian-vector multiplication, GN-vector multiplication can be calculated by back-propagating through the network. Martens also developed pre-conditioners for the CG algorithm which can be easily calculated to speed up convergence of the the CG iterations.

One general issue with this method is that as GN is a positive-semidefinite approximation to the actual Hessian, there is a possibility of converging to saddle points rather than at a local minimum. In other words, as it ignores directions of negative curvature in a nonconvex problem, it is susceptible to yield sub-optimal solutions. We will present numerical examples, as a supporting evidence of this case in section 4.4.3. We also provide explanations on what happens in saddle point and how we can use this information. Mizutani and Dreyfus [2008] note that the negative curvatures of a NN have important information for global minimization. Hence, it could be advantageous to use the exact Hessian, if one can handle directions of negative curvature using a CG solver. It turns out that via use of some classical numerical linear algebra algorithms we can avoid using Gauss-Newton approximation and utilize the Hessian directly. Lanczos method is a classical approach in linear algebra which converts a matrix B to a tridiagonal form T (see the next chapter for further great detail on this method). Dauphin et al. [2014] propose a Lanczos based variant of conjugate gradient to handle nonconvex Hessian for training the neural network. Their method also modifies the Hessian for convexification in a particular way. Basically a reduced Hessian $\bar{B}$ is produced in the space of Lanczos vectors, further to get the search direction the reduced Hessian will be replaced by $|\bar{B}|$ which is the replaces the negative eigenvalues λ_i of the matrix $\bar{B}$ with their modulus $|\lambda_i|$. This algorithm requires a huge storage space and also the reduced space can drop dimensions that encompass components of the model minimizer; we will discuss more deficiencies of the Lanczos based methods further.

The method we consider is a modified conjugate gradient method that doesn't require explicit vector storage and also does not rely on numerically unstable orthogonality of Lanczos vectors. Zhou et al. [2017] developed a modified conjugate gradient (MCG) method which can be applied to the nonconvex case using a trust-region framework. In what follows, we give a brief summary of the MCG method and how we can use it for training the NN.

4.3 Krylov Subspace methods

This section provides necessary background on methods based on iteratively generating subspaces known as Krylov subspace. These methods include conjugate gradient (CG), used in minimizing quadratic forms, Lanczos iterations, and their relationship in minimization of subproblems. First, we present the vanilla conjugate gradient method and Lanczos algorithm, we also need to provide how these two methods are connected and then show a simple unifying algorithm that can be used as either of them.

At each iteration, these methods add a new vector to the incumbent subspace to expand the Krylov subspace by one dimension, their difference aside from their intended usage is their choice of vectors for expanding the subspace.

Most CG based methods rely on extensive use of the Lanczos method within the CG framework for accurately solving the trust-region subproblems. The main interesting property of both is that starting with the same starting vector r_0, they both expand on the same Krylov subspaces through the iterations. The only difference is their basis choice for the same subspace; i.e. the CG algorithm works on a $H-$Conjugate basis of

$$P = \{p_0, p_1, \ldots, p_k\}, \tag{4.19}$$

whereas the Lanczos maintains orthogonal basis vectors

$$Q_k = \{q_0, q_1, \ldots, q_k\}, \tag{4.20}$$

for the Krylov subspace of

$$\mathcal{K}(r_0, B, k) = \{r_0, Br_0, \ldots, B^{k-1}r_0\}. \tag{4.21}$$

Therefore, it is possible to re-utilize the information generated by CG to keep Lanczos informations updated and solve the subproblem past the Steihaug-Toint method. However, as mentioned in the previous chapter, recovering the subproblem solution would require the storage of Lanczos vector or regenerating them. In this section, we review both algorithms as well as their connection and how some solvers utilize both to achieve maximum accuracy in solving the subproblems.

4.3.1 Conjugate Gradient Method

The conjugate gradient method is one of the prominent minimization techniques for unconstrained quadratic functions. Minimizing a function as

$$\min_s \frac{1}{2} s^T B s + g^T s \tag{4.22}$$

Starting from s_0 this method generates a sequence of directions p_i to update iterate s_i which satisfy the following relation:

$$p_i^T B p_j = 0, \forall i \neq j \tag{4.23}$$

The method works by computing the current residual $r_j = Bs_j + g$ and generating the corresponding conjugate direction p_i from this residual along with the optimum step-size α_i and update the iterate as $s_j = s_{j-1} + \alpha_j p_j$. Algorithm 11 shows how given the current residual r_j and iterate s_j CG method calls subroutine **cgUpdate** to generate the conjugate direction and update the iterate. The version of the algorithm shown depicts a slightly modified version of CG update which includes Lanczos book keeping including parameter $\{q, \gamma_j, t_u, t_d\}$. Lanczos method is a closely related linear algebra method for reducing the matrix B into tridiagonal form, this method is discussed in section 4.3.3.

Algorithm 11 Conjugate Gradient Iteration

```
1: function $[r_{j+1}, \alpha_j, p_j, \gamma_j, q, t_u, t_d]$ = cgUpdate($B, r_j, \alpha_{j-1}, p_{j-1}, \gamma_{j-1}$)
2: $\gamma_j = r_j^T r_j$;
3: if $\gamma_{j-1} = 0$ then $\beta = 0$ else $\beta = \gamma_j / \gamma_{j-1}$; end if
4: $p_j = r_j + \beta_j p_{j-1}$;
5: $\alpha_j = \gamma_j / p_j^T B p_j$;
6: $r_{j+1} = r_j - \alpha_j B p_j$;
7: endfunction
```

It is known that CG converges to the minimizer s^* in at most n iterations. Another important property of CG is to obtain minimizer of each subspace spanned by the current conjugate vectors p_j. The following theorem shows an important result of CG methods which is known as expanding subspace minimization. We state this theorem as presented in Nocedal and Wright [2006]

Theorem 4.3.1. *For arbitrary $s_0 \in \mathbb{R}^n$ and sequence s_k generated by conjugate direction algorithm, new residuals are orthogonal to previous conjugate directions that is*

$$r_k^T p_j = 0,\ \forall j = 0, 1, \ldots, k-1 \tag{4.24}$$

Furthermore s_j is the minimizer of $\frac{1}{2}s^T Bs + g^T s$ over the set

$$\{s | s = s_0 + span\{p_0, p_1, \ldots, p_{j-1}\}\} \tag{4.25}$$

Essentially this theorem states that each s_j iterate of CG method, minimizes the function over all previously derived conjugate directions p_j. We will use this theorem heavily on the next chapter.

4.3.2 Trust-region CG Solver: Steihaug-Toint

As one of the popular inexact Newton methods, Steihaug and Toint individually gave rise to a trust-region driver with an inner CG method to solve the nonconvex quadratic model over a trust-region of radius Δ. The method as discussed in Nocedal and Wright [2006] necessarily starts from origin $s_0 = 0$; As the algorithm progresses along the conjugate directions p_i the norm of the incumbents solution s_i increases monotonically, which allows monitoring the feasibility with regards to trust-region ball constraint.

The crucial property of this algorithm is the increasing norm property i.e.

$$0 = \|s_0\| \leq \|s_1\| \leq \ldots \leq s_k \leq s_{k+1} \leq \Delta.$$

This property is essential to the algorithm in order to satisfy the region constraint. Starting from the origin ensures that the increasing norm property holds, this means that the algorithm cannot be warm-started. Another note on this algorithm is that the inner iterations terminates as soon as the direction of nonpositive curvature is encountered (Step 4).

Algorithm 12 Steihaug-Toint CG subproblem solver

```
1: **function**$[s_k]$ = **STCG**$(B_k, g_k, \epsilon_k, \Delta)$
2:   Set $\hat{B}_0 = B_k$, $p_0 = -g_k$, $\hat{s}_0 = 0$, $r_0 = p_0$, isMod = 0, and $i = 0$
3:   While $(\|r_i\| > \epsilon_k)$ do
4:     If $(p_i^T \hat{B} p_i \leq 0)$ Then
5:       Find $\tau$ s.t. $s_k = s_{k-1} + \tau p_i$ minimizes $m_k(s_k)$ and $\|s_k\| = \Delta$.
6:       Return $s_k = s_{k-1} + \tau p_i$
7:     $\alpha_i = r_i^T r_i / p_i^T \hat{B} p_i$
8:     $\hat{s}_{i+1} = \hat{s}_i + \alpha_i p_i$;     $r_{i+1} = r_i + \alpha_i \hat{B} s_i$
9:     If $(\|s_{i+1}\| \geq \Delta)$ Then
10:      Find $\tau$ s.t. $s_k = s_{k-1} + \tau p_i$ minimizes $m_k(s_k)$ and $\|s_k\| = \Delta$.
11:      Return $s_k = s_{k-1} + \tau p_i$
12:    $\beta_{i+1} = r_{i+1}^T r_{i+1} / r_i^T r_i$;     $q_{i+1} = -r_{i+1} + \beta_{i+1} q_i$
13:    $i = i + 1$
14:  End
15:  Set $s_k = \hat{s}_i$
16: **endfunction**
```

4.3.3 Lanczos Iteration Method

As hinted in section 4.3, the Lanczos method produces an orthonormal basis which spans the corresponding Krylov subspace $\mathcal{K}(r_0, B, k)$ with respect to initial vector r_0. However, in the background using the basis Q this method reduces the matrix B of the equation $(Bx = b)$ to a tridiagonal form T as

$$Q_k^T B Q_k^T = T_k, \tag{4.26}$$

where Q_k is defined as in (4.20). The outline of the Lanczos algorithm is as follows:

As Algorithm 13 shows the implementation only requires one matrix-vector product with matrix B per iteration. Through the iterates the matrix T_k contains

$$T_k = \begin{bmatrix} \alpha_0 & \beta_1 & & & & \\ \beta_1 & \alpha_1 & & & & \\ & & & \ddots & & \\ & & \ddots & & \beta_{k-1} & \\ & & & \beta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & & \beta_k & \alpha_k \end{bmatrix} \tag{4.27}$$

Algorithm 13 Non-Restarted Lanczos Algorithm

```
1: Require: $r_0$
2: $\beta_0 = 0$
3: for $i = 1, 2, \ldots$ do
4:     $q_i = r_{i-1}/\|r_{i-1}\|$
5:     $\alpha_i = q_i^T B q_i$
6:     $r_i = Bq_i - \alpha_i q_i - \beta_{i-1} q_{i-1}$
7:     $\beta_i = \|r_i\|$
8: end for
```

It can be shown that when the algorithm converges, T_n stores the same eigen-information as the original matrix B (Golub and Van Loan [2012], Saad [2011]). However, as matrix T_n is tridiagonal this information can be extracted with a considerably cheaper methods in $O(n \log(n))$ (Coakley and Rokhlin [2013]). This structure is the key to enable solvers such as GLTR to use exact method of Moré and Sorensen in a modified subproblem in terms of T_n rather than the original matrix.

4.3.4 Conjugate Gradient and Lanczos Connection

As previously mentioned both CG and Lanczos start from an initial vector and expand the subspace with regards to Krylov property. The choice of starting vector for Lanczos method as a general matrix reduction algorithm is arbitrary, however for the CG method this choice is bound to be parallel to the steepest descent direction of $\nabla(f(x_k))$. Therefore, in algorithm 13 if we set $r_0 = g$ the Krylov subspace generated by Lanczos algorithm will coincide with that of conjugate gradient method (Gould et al. [1999]), this is the key on linking the two algorithms which drives the GLTR algorithm on being able to convert the QP model to simpler form (4.28) by re-using CG information. Algorithm 14 augments the CG method described in algorithm 11 with an additional book-keeping steps to keep track of the tridiagonal matrix generated by corresponding Lanczos method with initial vector set as normalized g.

Algorithm 14 Conjugate Gradient Iteration (with Lanczos book-keeping)

```
1: function $[r_{j+1}, \alpha_j, p_j, \gamma_j, q, t_u, t_d] =$ cgUpdate$(B, r_j, \alpha_{j-1}, p_{j-1}, \gamma_{j-1})$
2: $\gamma_j = r_j^T r_j$;
3: if $\gamma_{j-1} = 0$ then $\beta = 0$ else $\beta = \gamma_j / \gamma_{j-1}$; end if
4: $p_j = r_j + \beta_j p_{j-1}$;
5: $\alpha_j = \gamma_j / p_j^T B p_j$;
6: $r_{j+1} = r_j - \alpha_j B p_j$;
7: Lanczos book-keeping steps:
8: $q = r_j / \sqrt{\gamma_j}$;
9: if $\alpha_{j-1} = 0$ then
10:     $t_d = 1/\alpha_j$ and $t_u = 0$;
11: else
12:     $t_d = 1/\alpha_j + \beta/\alpha_{j-1}$ and $t_u = -\sqrt{\beta}/\alpha_{j-1}$
13: end if
14: endfunction
```

4.3.5 GLTR Algorithm: Integrated CG and Lanczos Solver

If the solution of the QP model lies outside of the trust-region boundary, the Steihaug-Toint algorithm terminates by projecting the solution on the boundary. Gould et al. [1999] proposed a method which continues on the CG iteration once Steihaug-Toint algorithm decides stop further improvements. The Generalized Lanczos Trust-Region (GLTR) algorithm, uses the Lanczos transformation to solve wider prospect of trust-region problems; That is on solving the system

$$(B_k + \sigma_k I)s_k = -g_k,$$

where B_k is not necessarily positive-definite.
As the authors demonstrate, starting the Lanczos algorithm with the initial vector $q_0 = g$, the CG residual information could be used to produce Lanczos vectors as byproduct of CG; the algorithm stores this information along with the CG iterates; therefore, as it hits a direction of negative curvature, it can continue with the Lanczos iterates rather than pursuing CG iterates. Finally the Lanczos vectors stored in the orthonormal matrix Q of size $n \times j$ which spans the Krylov subspace with respect to matrix B and vector g, can yield a basis to express the solution of the QP in that subspace. This is favorable as Lanczos iterations convert the matrix B into a tridiagonal form of T along their progress i.e.

$$Q^T B_k Q = T_k,$$

will have the special form, which enables cheap factorization of T_k, in order to solve the subspace trust-region quickly using an exact solver such as Moré and Sorensen [1983]. Expressing the solution in the Lanczos basis subspace, $s_k = Qw$ problem,

$$\min_{\|w\| \leq \Delta} \bar{g}^T w + \frac{1}{2} w^T T_k w \tag{4.28}$$

where $\bar{g} = Q^T(g_k) = \|g_k\| e_1$, and e_i represents the i^{th} column of identity matrix of proper size. Since now quadratic matrix T_k is tridiagonal, exact method of Moré and Sorensen [1983] can be used to obtain the solution by efficiently factorizing T_k. For practical efficiency the algorithm will progresses as the Steihaug-Toint algorithm initially (first phase), however when the optimal trust-region solution tends lies outside of boundary the algorithm proceeds as Lanczos iterations (second phase) to obtain a subspace trust-region subproblem. This enables the algorithm to obtain interior solutions in a computationally efficient way by pursuing normal CG algorithm, however if the solution is not interior then the power of Lanczos method is utilized. Therefore, either the user needs to save all q_i basis vectors to retrieve the solution from second phase, or needs to save all such vectors in memory, the algorithm can continue first phase without storing q_i for Lanczos basis, however if the algorithm needs to enter the second phase, re-computation of these vectors are needed which essentially doubles the Hessian-vector product or has store all Lanczos vectors; where both are prohibitive in deep learning large-scale problems.

4.3.6 Modified Conjugate Gradient

The algorithm comprises of outer and inner iterations. The outer iteration outlined in Algorithm 15, obtains the direction s_k and performs a line-search sequentially until convergence criteria is met.Omitting lines [3, 6–8] of Algorithm 15, and an arbitrary descent direction s_k, the algorithm resembles a line-search algorithm. Therefore, it preserves the outline of a line-search algorithm. However, the descent direction in this algorithm, is obtained by solving a quadratic subproblem (Step 3).

The inner iteration, described in Algorithm 16, is a modified variant of the conjugate algorithm applied to system $B_k s_k = -g_k$ that controls the size of s. To avoid confusion with other modified CG methods, we will refer to this variant as step-size controlling Modified Conjugate Gradients (MCG).

At each iteration a quadratic model around the iterate is formed (similar to Newton method), and the quadratic model is solved using CG. So far the algorithm resembles a variant of line-search Newton-CG method, or better known as truncated Newton method (Nocedal and Wright [2006]). These methods when reaching a singular or negative curvature direction, stop and yield steepest descent or the sub-optimal CG solution.

On the other hand trust-region methods have been known to handle singularities in the Hessian matrix well, by imposing a radius constraint on the step size. This radius, denoted by λ, is updated based on the ratio of model reduction to the function reduction which is updated by similar idea of trust-region algorithms using steps [6–9]. The inner problem is an implicit trust-region subproblem using the conjugate vectors to solve the system $B_k s_k = -g_k$, defining implicit bounds on the trust-region radius. Whenever the bound for left-most eigenvalue is not on the desired level, the Hessian is shifted by a rank-one update composed of residuals; similar ideas based on use of Lanczos vectors have been explored in Arioli et al. [1993].

Algorithm 15 Outer iteration of MCG with line-search

```
1: Initialization: Choose $x_0$, and a sequence $\{\eta_k\} > 0$ satisfying $\eta_k \to 0$, and set
   $\epsilon > 0$, $c \in (0,1)$, $\lambda_0 > 0$, and $k = 0$;
2: While ($\|g_k\| > \epsilon$) do
3:    $[s_k, \text{isMod}] = \textbf{MCG}(\mathrm{B_k, g_k}, \lambda_k, \eta_k)$
4:    $[\alpha_k, \rho_k] = \textbf{LineSearch}(x_k, s_k, m(s_k), f(x_k), g_k)$
5:    set $x_{k+1} = x_k + \alpha_k s_k$
6:    If ($\alpha < 1.0$), then $\lambda_{k+1} = 2\lambda_k$
7:    Else,
8:       if ($\rho_k > 0.75$) and isMod = 0, then $\lambda_{k+1} = 0.5\lambda_i$
9:    k=k+1
10: End
```

At first glance, it may appear that the sequence of $\{r_i\}$ corresponding to nonzero δ in Hessian modification Step 6:

$$\hat{B} = \hat{B} + \delta r_i r_i^T, \tag{4.29}$$

must be explicitly stored. However, we can leverage the properties of CG, and continue to carry out implicit multiplies with $\hat{B}$ while never storing the additional terms explicitly, this is outlined in section 4.5.1. Instead, the recursive relation $y_i = y_i + \delta r_i(r_i^T p_i)$ is used whenever $y_i = \hat{B} p_i$ is needed. Proof of why this works is provided in Theorem 4.5.1 which ensures that the storage overhead of the propose algorithm is comparable to the original CG algorithm. Additionally, as in CG, only one multiplication per inner iteration is needed (Bp_i). Handling the nonconvexities

Algorithm 16 Inner algorithm to get search direction

```
1:  function[$s_k$, isMod] = MCG($\mathrm{B_k}, \mathrm{g_k}, \lambda_\mathrm{k}, \eta_\mathrm{k}$)
2:    Set $\hat{B}_0 = B_k$, $p_0 = -g_k$, $\hat{s}_0 = 0$, $r_0 = p_0$, isMod = 0, and $i = 0$
3:    While ($\|r_i\| > \eta_k \|g_k\|$) do
4:      If ($p_i^T \hat{B} p_i \leq \lambda_k \|g_k\| \|p_i\|^2$) Then
5:        $\delta = (\lambda_k \|g_k\| \|p_i\|^2 - p_i^T \hat{B} p_i) / r_i^T r_i$
6:        $\hat{B} = \hat{B} + \delta r_i r_i^T$
7:        isMod = 1
8:      End
9:      $\alpha_i = r_i^T r_i / p_i^T \hat{B} p_i$
10:     $\hat{s}_{i+1} = \hat{s}_i + \alpha_i p_i$;     $r_{i+1} = r_i + \alpha_i \hat{B} p_i$
11:     $\beta_{i+1} = r_{i+1}^T r_{i+1} / r_i^T r_i$;     $q_{i+1} = -r_{i+1} + \beta_{i+1} q_i$
12:     $i = i + 1$
13:   End
14:   Set $s_k = \hat{s}_i$
15: endfunction
```

by Hessian modifications, also controling the inertia of Hessian using trust-region based strategy make this algorithm a good choice for optimization of quadratic models that are based on the actual Hessian. Moreover, the algorithm, unlike other nonconvex CG solvers, does not require storage or reproduction of Lanczos vectors; which makes it suitable to the large-scale nonconvex settings of deep learning problems.

Similar to procedure of Arioli et al. [1993], a lower bound is enforced on the modified Hessian in terms of the conjugate vectors, which is of the form:

$$\frac{p_i^T \hat{B} p_i}{p_i^T p_i} \geq \sigma_k. \tag{4.30}$$

The distinction of MCG from existing strategies is to make σ_k proportional to $\|g_k\|$ via the relation $\sigma_k = \lambda_k \|g_k\|$; which in turn makes σ_k approach 0 in the limit, so long as λ_k is bounded. This can be seen in Steps [5–11] of Algorithm 16. Second, the scale term λ_k is used to refine the rate at which σ_k goes to zero according to progress made during the previous iteration of the outer algorithm. This helps tailor the choice of σ_k to the specific problem being solved. Essentially, σ_k used here takes into account the following factors: (i) the resultant effect on the growth of d_j, (ii) the size of the current gradient, and (iii) the quality of the last search direction. Because the residuals of CG are parallel to the Lanczos vectors, it is easy to see that $\delta r_i r_i^T$ used in Step 9 in 16 is equivalent to the modification described in following equation of Arioli et al. [1993]:

$$E_k = \sum_j \gamma_j q_j q_j^T \tag{4.31}$$

where $\gamma_j = \delta \|r_i\|^2$.

Note that λ_k is inversely related to an upper bound on an implicitly defined trust-region. Thus, in Steps 5-11 of Algorithm 15, λ_k is modified in a similar manner to the trust-region radius in a trust-region algorithm. If an adequate model of the objective function is found within the trust-region then the region is expanded; conversely, if the approximation is poor then the region is contracted. That is, when the predicted ratio is good, λ_k is decreased, and conversely, when the predicted ratio is bad, λ_k is subsequently increased. The inner iterations of Algorithm 16 solve the $\hat{B}_k s_k = -g_k$ within a scale term η_k of the current norm of the objective gradient. We later prove that this new algorithm possesses the theoretical strength of a trust-region algorithm and that 16 is actually modifying B_k according to an implicit trust-region. The convergence of the algorithm is at least linear if η_k is bounded away from 0, and superlinear if η_k converges to 0.

As a result of the following inequality

$$\frac{p_i^T \hat{B} p_i}{p_i^T p_i} > \lambda_k \|g_k\|, \tag{4.32}$$

we see that the modified Hessian can approach a singular system only in as much as the current corresponding gradient also approaches zero. This ensures that even if $\|\hat{B}_k^{-1}\|$ approach infinity, the step s_k must still converge to 0 in the limit (which is necessary for fast convergence). Note that 16 can easily be adapted to use a preconditioner if available, as in regular PCG (preconditioned conjugate gradient) methods. To permit the algorithm to be as general as possible, we only require that the modification term δ satisfy the bound

$$\delta \geq (\lambda_k \|g_k\| \|p_i\|^2 - p_i^T \hat{B} p_i)/(r_i^T p_i)^2 \tag{4.33}$$

in step 8 of 16 with equality on the very first iteration. That is, $p_i^T \hat{B} p_i \geq \lambda_k \|g_k\| \|p_i\|^2$ is ensured when $\hat{B} = \hat{B} + \delta r_i r_i^T$. The modification matrix $\delta r_i r_i^T$ may be as large as desired. Zhou et al. [2017] provide a theorem demonstrating that if $\delta = \infty$ whenever indefiniteness is detected and $i > 0$, then MCG will terminate with $s_k = d_i$ and $r_{i+1} = 0$, behaving in a manner very similar to truncated CG. Though we do not recommend such an extreme variant of MCG in practice, we do emphasize that all the convergence properties stated will continue to hold.

4.4 MCG for Deep Learning problems

This section briefly motivates the use of the modified conjugate gradient algorithm by Zhou et al. [2017] for deep learning problems.

4.4.1 Motivation

Martens [2010] explore the use of PSD approximation to the Hessian by using the Gauss-Newton approximation, however in order to use true Hessian information a nonconvex CG solver is required to better exploit the Hessian information. As mentioned in the previous section as we are interested in a matrix-free framework, variants of CG solvers that can handle nonconvexities in the curvature matrix are desired. Two prominent algorithms in this framework include Steihaug-Toint method (Steihaug [1983], Toint [1981]), and GLTR (Gould et al. [1999]). In this section, we give a brief overview of these algorithms and discuss why another variant to handle nonconvexities can be better suited in the context of deep learning.

4.4.2 Deep Learning Application

This section overviews the application of the mentioned algorithms in the context of deep learning. In order to capture complex underlying functions governing the data, it's suggested that deeper and wider networks (Cheng et al. [2016]) should be utilized for training. This in turn vastly increases the dimensions of the underlying

optimization problem.
As mentioned to use actual Hessian information, the inner algorithm needs to be able to handle nonconvexities. Of the two Steihaug-Toint and GLTR methods mentioned above, the former relies heavily on the increasing norm property, which requires dropping the possibility of warm-start solutions as $s_0 = 0$ is necessary. On the other hand, as also Martens and Sutskever [2012] argues, warm-start availability significantly improves the speed of obtaining the solution and reduces number of Hessian-vector products needed for convergence. Ineffectiveness of the Steihaug-Toint algorithm in the deep learning context has also been observed empirically by Wiesler et al. [2013].
In the numerical section we implement the Steihaug-Toint algorithm in oder to compare the results for the second-order algorithms; showing that in fact being able to warm start CG has great advantages.

The GLTR algorithm outlined previously is the next popular choice for the nonconvex quadratic problem. In the deep learning context, the problems are very large-scale compared to traditional optimization problems. This means that the number of variables n would be massively high; although this will not cause an issue normally, however this can create issues when dealing with GLTR algorithm. As mentioned above to be able to solve the quadratic subproblem in the reduced space of Lanczos vector basis, the algorithm relies on maintaining the matrix Q which includes columns with huge number of elements. Therefore, to minimize the quadratic exactly we either need to store all the Lanczos vectors corresponding to matrix Q, in order to be able to get the inverse image of the solution for the tridiagonal system and retrieve the true minimizer of the quadratic. Storing the columns of this matrix, as n is really high, can get infeasible if we are interested in solving the quadratic model exactly.

There are some variants that try to minimize the quadratic in a Krylov space which is restricted to K dimensions, for example Vinyals and Povey [2012] store $K = 20$ columns for the Q matrix.

Therefore, among the algorithms that can work with low memory requirements, we will see that MCG can be more suited in training the network, as it can produce better solutions when direction of negative curvature is occurred and also with limited storage requirements.

4.4.3 Incomeplete Training with GN

Implementing GN-vector computation algorithm and using it instead of Hessian-vector multiplies to feed the CG algorithm, we present cases that can further improve when GN is discarded as the approximation of curvature; thus motivating the use of exact Hessian (HF) instead of approximate curvature matrix GN.

Table 4.1: 2500 Observations - 6 layers each with 10 units

Method	**#Iter**	**Training Loss**	**Validation Loss**	**Training Error(%)**	**Validation Error(%)**
HF	62	0.012283	0.059523	1.07%	15.84%
GN	19	0.018199	0.052841	**2.61%**	15.04%
GN+HF	19+5	0.007109	0.052234	0.69%	13.76%

Table 4.1 initially shows three training information corresponding to exact Hessian method, GN method and also hybrid of both. In the first two rows, the algorithm starts from the same initial solution and we can see that exact Hessian yields a better training error, where as the Gauss-Newton method gives a better validation error. However, as GN ignores the negative curvature directions, the achieved solution potentially may not satisfy second-order necessary condition. This patterin is indeed the case, in our motivating example. We can view this, by using HF method starting from last iterate of GN's solution. Thus we can see that by taking into account the exact curvature information, we can further proceed to optimize the solution and get better results in terms of both the training and validation errors. One possible guideline for using the hybrid method can be the number of training misclassification points. To shed further light on this issue, first, we write the terms corresponding to Hessian and Gauss-Newton Matrix as stated in Martens and

Sutskever [2012].

$$H = \frac{1}{|S|} \sum_{(x,y)\in S} J_f \mathcal{H}_L J_f + \sum_{i=1}^{m} [\nabla L]_i \mathcal{H}_{[f(x,w)]_i}, \tag{4.34}$$

$$G = \frac{1}{|S|} \sum_{(x,y)\in S} J_f \mathcal{H}_L J_f \tag{4.35}$$

Here ∇L and $\mathcal{H}_L$ are gradient and Hessian of the loss function and J_f is the Jacobian of the implicit function $f(x,w)$. Therefore, the two curvature information matrices differ only in the the second term. As we fit more training data $\nabla L_{(x,y)} \to 0$. However, when there are misclassification, these two matrices can differ. Therefore, the gradient of the loss can be a good measure to a promising hybrid method, in order to remove the possibility of getting saddle-point solutions.

The above example shows that although GN yields good results, it can also stop on a saddle point, as it ignores the negative directions. Therefore, the hybrid method could be useful in which we use the power of MCG and exact Hessian in order to hedge against having saddle point solutions.

4.4.4 Numerical Experiments

We used MNIST dataset in order to evaluate the performance of each of the above algorithms. The processed data has 784 input variables corresponding to pixel information and has 10 outputs that correspond to 0–9 digits. MNIST data has 70,000 observations, where 60,000 are used for training and the remaining 10,000 are used for validating the NN results.

4.4.5 CUTEr Problems

To demonstrate the robustness of the new line-search approach on a wide range of problem types we compare the implementations of MCG with the Steihaug-Toint trust-region method on the unconstrained CUTEr test problems Bongartz et al. [1993], Gould et al. [2003] at SAS. We applied Algorithm 15 to all SAS translations of the unconstrained CUTEr test problems maintained by the SAS/OR testing team resulting in a total of 220 problems. For efficiency, a diagonal pre-conditioner is used initially and transitioned to an approximate LDL factorization of the Hessian matrix when suitable near a solution. Warm-starting is used when applicable to improve performance.

Table 4.2: Results on SAS CUTEr set

	Steihaug-Toint			
MCG	**ERROR**	**LIMIT**	**SOLVED**	Total
ERROR	1	0	0	1
LIMIT	0	0	2	2
SOLVED	0	11	206	217
Total	1	11	208	220

Table 4.2 shows that the MCG algorithm is more stable solving 9 more instances compared to Steihaug-Toint and reaching the iteration limit on 2 instances. The failed instance correspond to "s214.mod" from AMPL instances; where the origin is the optimal solution with values of 0. Both algorithms get close to this amount however both fail the line-search as the curvature grows towards the optimal solution.

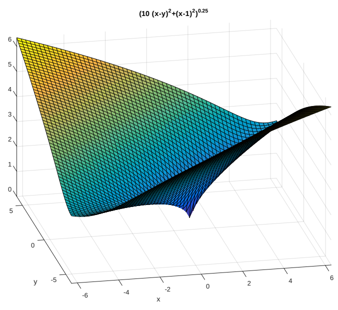


Figure 4.2: Problem S214.mod

Because of the large number of problems we use performance profiles recommended by Dolan and Moré [2002]. Figure 4.3 and Figure 4.4 provides performance profiles comparing iterations and function evaluations used by Steihaug-Toint and MCG respectively. We emphasize that past articles have struggled to show marginal improvement over Steihaug-Toint on the unconstrained CUTEr test set Erway and Gill [2009], Erway et al. [2009], Gould et al. [1999]. Both plots show the advantage of MCG in terms of both iterations and evaluations. The benefits of saving in terms of function evaluations is highlighted in the next section in the context of deep learning problems, where both function evaluation and Hessian-vector multiplies are expensive.

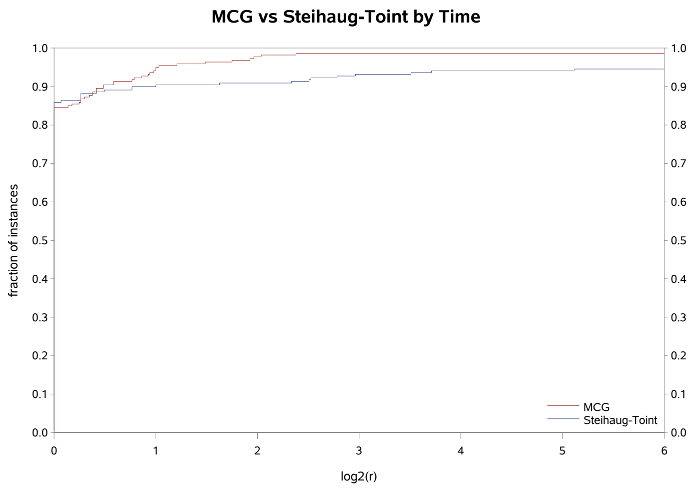


Figure 4.3: Performance profile comparing iterations on the CUTEr test set.

4.4.6 Hot-starting with Computationally Cheaper Algorithms

Although proving useful, the second-order algorithms tend to have heavy iterations. We can distinguish the heavy comparison by counting the number of multiplies against the curvature matrix (Hessian or Gauss-Newton).

In order to see how we can reduce the cost of heavy iterations, we try ideas on

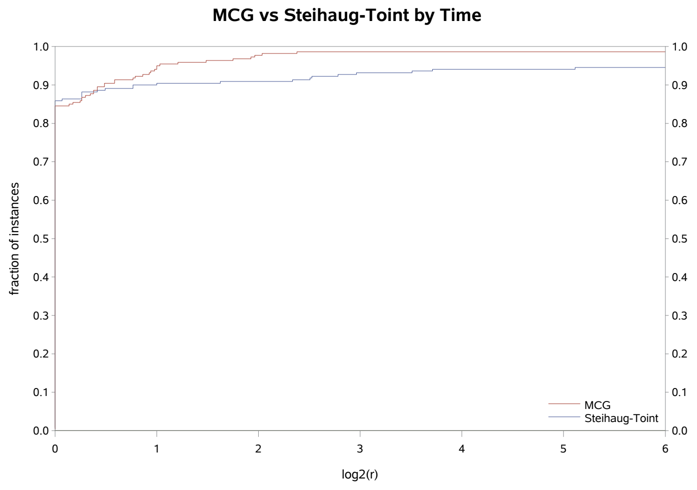


Figure 4.4: Performance profile comparing number of function evaluations on CUTEr test set.

how to hot-start the second-order algorithm using a better initial point compared to random which can save time on training the model.

L-BFGS: As at each iteration we are evaluating the gradient by performing the back-propagation algorithm, application of algorithms such as L-BFGS comes to mind. The winning property of L-BFGS is that since computing curvature matrix-vector product is much more expensive than gradient evaluation, having an approximate curvature matrix based on innate secant updates of L-BFGS we can get a good solution in faster runtime. Therefore, in addition to the combination of HF-MCG/GN-MCG (Hessian-Free Methods), we incorporate L-BFGS solver to minimize the loss function in our numerical examples. For each of the solvers mentioned above we can have regularizers such as ℓ_1 and ℓ_2 added as penalty to the loss function.

Although L-BFGS algorithm cannot generalize well to the data, however as it drops obtaining direct second-order information, the iterations are reliably much faster than the rest of the algorithms. Therefore, using different strategies we can try to make matrix-free algorithms work less towards the optimum solution.

We ran the experiments on two different networks with 300 and also 350-100 hidden neurons. From the Tables 4.3,4.4 we can see that both methods HF and GN produce comparable validation errors; corresponding regularizers (if any) are enclosed as subscripts. We can see that exact Hessian (HF) tends to converge faster than approximation of Gauss-Newton. Also we can see that (ℓ_1) regularized counterparts tend to generate comparable results.

In our experiments we used L-BFGS to train the NN. We can see that it quickly minimizes the training error although cannot generalize comparable to second-order algorithms. Starting MCG from the initial point generated by L-BFGS, we can try to train the network further by extracting information on the curvature of the function. We used both HF and GN to train the network further, as with early-stopping criteria the network cannot improve by more than 4 iterations (Table 4.3) and 6 iterations (Table 4.4). This means that after few number of iterations the solution is trapped in a local minimum region where further optimization, over-fits the network, giving rise to higher validation error. Therefore, hot-starting MCG from a naive L-BFGS without regularization cannot yield results as good as native MCG run.
One observation is that since L-BFGS is the can rapidly reduce the training error and thus more prone to over-fitting, we can use different norms in order to regularize the weights which is a technique used to prevent over-fitting, also used in classifiers such as SVM and regression. Therefore, we used a variant of smoothed ℓ_1 and ℓ_2 regularizers for L-BFGS. We can see that accounting for the actual and approximate curvature using HF and GN respectively, good solutions with lowest validation error are achieved. Therefore, by using such permutation of solvers we can easily train a huge network, and use less number of costly second-order evaluations. However, it seems that to our goals smoothed ℓ_1 norm, performs better than ℓ_2 norm, yielding errors comparable or better than the complete second-order algorithm.
Similar results are attained in larger networks. In Table 4.4, we see that the results with ℓ_1 regularized LBFGS are comparable to complete second-order methods with slightly higher misclassification.

Table 4.3: MNIST results with one layer and 300 neurons

Neurons	Methods	Iter	Train	Valid	T-Mis	V-Mis
300	HF	38	0.000316	0.00583	0	0.0173
300	GN	58	0.0003	0.005686	0	0.0176
300	HF$_{\ell_1}$	200	0.00675	0.00632	0.0025	0.017
300	GN$_{\ell_1}$	200	0.00698	0.00676	0.0032	0.0188
300	L-BFGS	86	0.001777	0.010096	0.0042	0.0286
300	L-BFGS$\rightarrow$HF	4	0.0003	0.009657	0.0001	0.0258
300	L-BFGS$\rightarrow$GN	4	0.000301	0.00967	0.0001	0.0252
300	L-BFGS$_{\ell_1}$	216	0.004918	0.005972	0	0.0196
300	L-BFGS$_{\ell_1}$ $\rightarrow$HF	8	0.0002	0.005689	0	0.0183
300	L-BFGS$_{\ell_1}$ $\rightarrow$GN	4	0.000283	**0.005681**	0	0.0187
300	L-BFGS$_{\ell_2}$	87	0.00362	0.008152	0.0045	0.0248
300	L-BFGS$_{\ell_2}$ $\rightarrow$HF	5	0.000455	0.007863	0.0001	0.0237
300	L-BFGS$_{\ell_2}$ $\rightarrow$GN	4	0.000505	0.007848	0.0001	0.0235

Table 4.4: MNIST results with two layers and 350-100 neurons

Neurons	Methods	Iter	Train	Valid	T-Mis	V-Mis
350-100	HF	57	0.000559	0.005714	0.0001	0.0178
350-100	GN	65	0.000448	**0.00554**	0	0.0168
350-100	HF$_{\ell_1}$	300	0.00641	0.00595	0.0016	0.016
350-100	GN$_{\ell_1}$	300	0.0063	0.00617	0.0016	0.0173
350-100	L-BFGS	88	0.002688	0.009622	0.0069	0.0284
350-100	L-BFGS$\rightarrow$HF	5	0.000571	0.008975	0.0002	0.0256
350-100	L-BFGS$\rightarrow$GN	6	0.000421	0.009013	0.0003	0.0256
350-100	L-BFGS$_{\ell_1}$	111	0.00957	0.006656	0.0054	0.0217
350-100	L-BFGS$_{\ell_1}$ $\rightarrow$HF	8	0.000377	0.005686	0.0001	0.019
350-100	L-BFGS$_{\ell_1}$ $\rightarrow$GN	12	0.000226	0.005713	0	0.0185
350-100	L-BFGS$_{\ell_2}$	91	0.003905	0.008311	0.0066	0.0233
350-100	L-BFGS$_{\ell_2}$ $\rightarrow$HF	4	0.000555	0.007675	0.0002	0.0226
350-100	L-BFGS$_{\ell_2}$ $\rightarrow$GN	5	0.000446	0.007652	0.0001	0.0217

Therefore, we see that our hybrid proposed method can get comparable validation error compared to pure second-order algorithms, with much less computational cost. Therefore, it is easier in the context of hyper-parameter optimization to adjust the parameters and get better solutions. For example running the LBFGS-ℓ_1 algorithm with updated weight of regularizer we get better results summarized in Table 4.5.

We can see that our method could get validation error and misclassification count

Table 4.5: Hyper-Optimized problem ℓ_1 weight=2

Neurons	Method	Iter	Train	Valid	T-Mis	V-Mis
350-100	L-BFGS$_{\ell_1}$	191	0.009845	0.005783	0.0015	0.0182
350-100	L-BFGS$_{\ell_1}$ $\rightarrow$HF	12	0.000275	0.005427	0	0.0163
350-100	L-BFGS$_{\ell_1}$ $\rightarrow$GN	15	0.000172	**0.005393**	0	**0.0157**

that is better than all results achieved in Table 4.4.

4.5 MCG for Deep Learning

The context of deep learning problems, require specific customization of algorithms to make it suitable for such problems. In this section, we show two major properties of the proposed method which are desirable in this context. The first part includes the memory-less property of our Hessian updates, in other words, unlike other complex algorithms such as GLTR the MCG method does not require to store the modifications to the Hessian. The second property is the ability of warm-starts. Due to their design, the nonconvex Steihaug-Toint and therefore GLTR cannot be warm started as their crucial increasing norm property would not be valid otherwise. We extend the MCG algorithm and propose new criteria for warm-starting the algorithm. Numerical results show the benefits of warm-start on the computation of Hessian-vector products.

4.5.1 Memory-less property

As mentioned in section 4.3.5, GLTR (Gould et al. [1999]) tackles the nonconvex problem by incorporating the power of Lanczos Iterations. This method reduces the original QP to a subproblem with tridiagonal matrix and therefore would be able to solve the inner trust-region problem using the exact solver by Moré and Sorensen [1983]. Basically the inner QP is of the form

$$\begin{aligned} \min m_k(s) \quad &= \frac{1}{2} s_k^T B_k s_k + \nabla f(x_k)^T s_k \\ &\|s_k\| \leq \Delta \end{aligned}$$

Where as if there is a transformation matrix Q that reduces the B_k to a tridiagonal form T_k as $T_k = Q_k^T B_k Q_k$, by change the variables in the form $x = Q_k d_k$ we get the equivalent problem as (4.28). Which can be easily solved using the exact algorithm of Moré and Sorensen [1983], because of special structure of T_k. Lanczos based methods such as Dauphin et al. [2014], Gould et al. [1999], Vinyals and Povey [2012], need to store the transformation matrix Q to be able to retrieve the solution from the reduced space (w^*) and project back to original problem to get s^*, using the linear system $s = Qw$.

The power of this method relies on being able to store the matrix of all Lanczos vectors that led to producing the solution for the tridiagonal subproblem. However, in the deep learning context, a moderate network induces the optimization problem to have at least hundred thousands of variables. Storing the Lanczos vectors of this size can get impractical, especially if the number of inner iterations gets bigger. Another remedy is being able to regenerate the Lanczos vectors which effectively duplicates the number of multiplications by the Hessian. In this section, we show that the MCG algorithm does not require to storing the rank-1 modifications to the Hessian.

Theorem 4.5.1 provides safe removing the requirement of storing a matrix for MCG as discussed in Zhou et al. [2017].

Theorem 4.5.1. *No residual vectors $\{r_k,\ k < i\}$ are stored to calculate $\hat{B}p_{i+1}$ in Algorithm 16.*

Proof. Because of the recursive relationship $p_{i+1} = -r_{i+1} + \beta_{i+1}p_i$, we note that calculation of $\hat{B}p_{i+1}$ only involves $\hat{B}r_{i+1}$. However, the main CG property as mentioned in (Theorem 5.3) Nocedal and Wright [2006] asserts that the residuals remain orthogonal throughout the iterations i.e.

$$r_{i+1}^T r_k = 0, \forall k < i+1. \tag{4.36}$$

Now assume that at some iteration j a direction of non-negative curvature is occurred which means that the Hessian estimate was updated by a rank-1 update of $\delta_j r_j r_j^T$. Therefore, we have the relation

$$\hat{B}r_{i+1} = (B + \delta r_k r_k^T)r_{i+1} = Br_{i+1}. \tag{4.37}$$

Second equality follows from (4.36), and this thus the theorem follows. □

This interesting result removes the requirement of storing a matrix for MCG in Zhou et al. [2017]. Having stated this, we noticed that the low-rank modification matrix can be stored to improve the stability. Combining Theorem 4.5.1 with the observation that the numerical stability can be improved with the stored matrix, we believe that storing only the last several modifications (for example, say the last 5 modifications) and continuous updating this matrix with the fixed size as the iteration continues, can yield better results.

Theorem 4.5.1, leverages the benefit of using MCG compared to Lanczos variant algorithms, which essentially require to store the normalized residuals q_i or regenerate them in order to translate back the solution from the subspace to the original space. Our result therefore indicates that Algorithm 16 is just a simple addition to original CG which makes it easy to implement and also does not require complex transformation; moreover, the storage requirements can be the same as CG or Steihaug-Toint, which are feasible in the large-scale problems in deep learning context.
In the next section we discuss another property suitable for deep learning problems on being able to warm-start the algorithm.

4.5.2 Warm-start MCG

The experiments show that inner CG algorithm spends lots of iterations on solving the QP. Martens [2010] emphasize on having warm starting strategy for the algorithm in order to reduce the time it takes to solve the inner QP. Passing the previous direction which likely will capture the negative curvature as the trust-region may have restricted the step size. A similar kind of strategy is used in the first-order algorithms by incorporating the momentum in the SGD updates. This is especially important in the context of second-order algorithms, as warm starting can save the time spent on the CG algorithm. In the context of nonconvex CG methods like Steihaug-Toint, it's assumed to start from the origin, however, we need to be able to feed the previous solution p_{k-1} as the starting solution of CG.

In this section we provide necessary guaranties to be able to warm-start the MCG algorithm from an arbitrary given direction, which as discussed in Martens [2010] can be the previous direction of the CG solve. This in turn will improve the speed of the algorithm for practical implementation.

As the method presented by Zhou et al. [2017] does not support a nonzero input vector s^-, in Algorithm 17 we present new convergence theory in the following section proving identical properties. Algorithm 17 resembles the main MCG algorithm, however direction d^- is used to warm-start MCG; The corresponding residual needs to updated as step 2 captures this requirement. Step 23 exits the algorithm by inducing the final step found to incorporate the warm starting vector s^-.

To ensure convergence, any initialization strategy for s^- may be used as long as it satisfies

$$\|s^-\| \leq \frac{1}{\lambda} \tag{4.38}$$

$$\begin{aligned} m_k(s^-) &\leq 0 && (4.39) \\ \|Bs^- + g_k\| &\leq \|g_k\| && (4.40) \end{aligned}$$

Trivially this is satisfied by $s^- = 0$, but other initializations choices are readily formed by finding roots of a one-dimensional quadratic subproblem. It is hoped

Algorithm 17 Warm-start MCG Algorithm

```
1: function $[d_k, \text{isMod}] = \textbf{MCG}(\mathrm{B}, \mathrm{g_k}, \lambda_\mathrm{k}, \mathrm{cgtol}, \mathrm{s}^-)$
2:   $p_0 = -(Bs^- + g_k)$;
3:   $y_0 =$ MatrixMult$(B, p_0)$;                    ▷ $y_0 = Bp_0$
4:   Set $\hat{d}_0 = 0$, $r_0 = p_0$, isMod $= 0$, and $i = 0$;
5:    while($\|r_i\| >$ cgtol and $i \le i_{\max}$) do
6:     if $(p_i^T y_i \le \lambda_k \|g_k\| \|p_i\|^2)$ then          ▷ $y_i = \hat{B}p_i$
7:         Set $\delta_{\text{low}} = (\lambda_k \|g_k\| \|p_i\|^2 - p_i^T y_i)/(r_i^T p_i)^2$;
8:          if  $i = 0$   then
9:            $\delta = \delta_{\text{low}}$;
10:         else
11:           choose   $\delta \ge \delta_{\text{low}}$;
12:         end
13:        $y_i = y_i + \delta r_i (r_i^T p_i)$;
14:         isMod $= 1$;
15:     end
16:    $\alpha_i = r_i^T r_i / p_i^T y_i$;
17:    $\hat{d}_{i+1} = \hat{d}_i + \alpha_i p_i$;          $r_{i+1} = r_i + \alpha_i y_i$;
18:    $\beta_{i+1} = r_{i+1}^T r_{i+1} / r_i^T r_i$;     $p_{i+1} = -r_{i+1} + \beta_{i+1} p_i$;
19:    $v_{i+1} =$ MatrixMult$(B, r_{i+1})$;       ▷ $v_{i+1} = Br_{i+1}$
20:    $y_{i+1} = -v_{i+1} + \beta_{i+1} y_i$;
21:    $i = i + 1$;
22:   end
23:   Set $d_k = s^- + \hat{d}_i$
24: endfunction
```

that this warm-start strategy will reduce the total number of required matrix multiplications.

It is important to point out that the Algorithm 17 is an MCG algorithm on the shifted system. That is, for a given s^-, it solves

$$\begin{array}{ll} \min_{\hat{s}\in\mathbb{R}^n} & \hat{m}(\hat{s}) = \hat{s}^T(g_k + \hat{B}s^-) + \frac{1}{2}\hat{s}^T\hat{B}\hat{s} \\ \text{s.t.} & \hat{s} \in \operatorname{span}(p_0, \ldots, p_{i-1}). \end{array} \tag{4.41}$$

The following lemma is important to ensure monotonic decrease of the quadratic model; essentially a guarantee that the predicted reduction does not elbow up after a certain number of matrix multiplies.

Lemma 4.5.2. *Suppose that Algorithm 16 is applied to the system $(Bs = -g)$ with the parameter λ_k used to decide δ. Then the following properties hold at each iteration:*

$$\hat{m}(\hat{s}_i + s^-) \quad \leq \quad \hat{m}(\hat{s}_{i-1} + s^-) \tag{4.42}$$

Further $\hat{s}_i + s^-$ solves the subspace subproblem

$$\begin{array}{ll} \min_{\hat{s}\in\mathbb{R}^n} & \hat{m}(\hat{d}) = {g_k}^T s + \frac{1}{2} s^T \hat{B} s \\ \text{subject} & s = \hat{s} + s^-, \hat{s} \in \operatorname{span}(p_0, \ldots, p_{i-1}). \end{array} \tag{4.43}$$

Proof. The lemma follows because of Lemma 3.2 in Zhou et al. [2017] on problem (4.41) and the definition of $\hat{m}(\hat{s})$. □

Lemma 4.5.3 implies Algorithm 15 implicitly defines a controllable trust-region. Further it show that the size of the radius is proportional to the inverse of λ_k used in Algorithm 15 and 16.

Lemma 4.5.3. *Let d_k denote the search direction obtained by algorithm 16. Then*

$$\|d_k\| \quad \leq \quad \frac{n}{\lambda_k} \tag{4.44}$$

$$m(0) - m(d_k) \quad \geq \quad \frac{\|g_k\|}{2} \min\left(\frac{1}{\lambda_k}, \frac{\|g_k\|}{\|B_k\|}\right) \tag{4.45}$$

$$d_k^T g_k \quad < \quad 0 \tag{4.46}$$

Proof. This lemma can be shown in the same manner as in the proof of Theorem 3.3, Lemma 3.4 and 3.5 in Zhou et al. [2017] because of equations (4.38) to (4.40). □

Theorem 4.5.4 shows that Algorithm 16 will naturally reduce to unmodified linear CG after a finite number of iterations. Thus faster convergence rate can be achieved as the algorithm automatically reverts to unmodified truncated Newton's method satisfying the step-size requirements of Dembo and Steihaug [1983] by design.

Theorem 4.5.4. *Suppose that x^* is an accumulation point of $\{x_k\}$ where x_k is obtained from Algorithm 15. If the second-order sufficient conditions hold at x^* and $f(x)$ is twice Lipschitz continuous in an open neighborhood of x^*, then the following properties hold:*

- *λ_k in Algorithm 15 is bounded, and there exists an integer K such that following holds for all $k > K$,*

$$\frac{p_i^T B p_i}{p_i^T p_i} > \lambda_k \|g_k\| \tag{4.47}$$

- *The main sequence $\{x_k\}$ converges at least linearly to x^*, and super-linearly if $\lambda_k \to 0$.*

- *The actual to predicted reduction ratio ρ_k converges to 1.*

Proof. Leverage Lemma 4.5.2 and Lemma 4.5.3, this can be shown as in Zhou et al. [2017]. □

4.6 Deep Learning Experiments

4.6.1 Comparison of the Algorithms

To compare the second-order method we implemented Steihaug-Toint and experimented on three networks. The experiments involve early stopping criteria; therefore, at each iteration the validation set is scored and the best solution is selected throughout the trajectory of the algorithm after 20 non-improving iterations. Table 4.6 shows the results with regards to number of mis-classifications with the best trained network on the validation set. Results of Table 4.6 show how Hessian information may yield slightly better solutions compared to Gauss-Newton; however the significant distinction is how LBFGS overfits the data and cannot generalize as well as the second-order methods.

Table 4.6: MCG (HF-GN) vs Steihaug Algorithm on MNIST

Network	**MCG(GN)**	**MCG(HF)**	**Steihaug**	**LBFGS**
350-100	2.23%	1.68%	1.68%	2.75%
350-350	1.72%	1.67%	1.60%	2.70%
500-150	1.69 %	1.63%	1.70%	2.78%

4.6.2 Warm-start Effects on Hessian-vector Multiplication

In this section we show how the warm-start strategy helps reducing the computation time. To We also offer a comparison between the proposed method and the Steihaug-Toint algorithm using the HF operator. Table 4.7 outlines the results of our comparison, here #Hv shows the number of matrix multiplies of each algorithm until converging to a well-trained network; however, as discussed in detail by Martens and Sutskever [2012], the Steihaug-Toint approach is in general not appropriate for the unique needs of deep-learning solvers. We can see that, as was the case with other second-order algorithms, Steihaug-Toint appears to reliably outperform the L-BFGS method. This further confirms Martens [2010] observation on the benefits of using second-order information in the deep learning context. However, as suggested by Martens and Sutskever [2012], the cost of applying Steihuag-Toint is in this case much greater; this can be observed by considering the total number of matrix multiplies denoted in columns two and five. Note that for Hessian-free approaches in deep learning, matrix multiplication quickly becomes the underlying computational bottleneck and it is thus important to keep this quantity as low as possible.

Table 4.7: Number of Hessian-vector products (#Hv) and misclassification percentage for MCG (HF) and Steihaug-Toint(HF) algorithms on MNIST

Algorithm	MCG		Steihaug-Toint	
Network	# Hv	V(%)	# Hv	V(%)
350-100	3,592	1.68	22,809	1.68
350-350	3,093	1.67	22,472	1.6
500-150	3,254	1.63	15,005	1.7

4.7 Discussion

The proposed MCG approach provides a new flexible Hessian-free solver that is ideally suited for the unique needs of challenging high-dimensional deep learning problems. Current deep learning approaches are unable to solve the underlying

Newton equations (even when positive-definite) to user-defined levels of accuracy; either because the Hessian is never used, or the system is restricted to a necessarily small dimensional Lanczos subspace. We have modified and applied a line-search approach free from these limitations that has trust-region strength convergence theory and is effective for both Hessian and Generalized Gauss-Newton (GN) operators. This method uses the exact Hessian matrix, and it is therefore hoped that it inherits all the nice convergence properties from the Newton method. The warm started expansion and modification has further removed some restrictions of the original method. Numerical results demonstrate the effectiveness of this approach.

Chapter 5

Deep Learning: Successive Subspace Methods

5.1 Introduction

In this chapter, we recognize another set of closely related algorithms that are used for solving the trust-region subproblems, namely Successive Subspace Methods (SSM). SSM, introduced by Hager [2001], relaxes the requirements corresponding to sequences of incumbent subspaces while showing same convergence results. This chapter first introduces SSM and its variations, then motivates its application in the large-scale context of deep learning problems and the need for suitable extensions of the algorithm. Lastly, the chapter concludes showing the benefits in comparison to traditional methods in deep learning problems.

The highlight of this chapter is the development of a new algorithm as a hybrid variant of SSM and conjugate gradient methods. This algorithm generates update directions that can take into account the eigen directions corresponding to smallest eigenvalues of the curvature matrix and utilize them in small sized subproblems that are devised to handle projected vectors in small subspaces. Our numerical results show benefits of extracting the eigen information, compared to other algorithms that do not use eigenvector information in the nonconvex setting and also some convex counterparts.

5.2 Background

Successive Subspace Methods (SSM) tackle difficulties that are innate to Lanczos method. Paige [1972] showed that in practice the orthogonality of basis vectors q_i defined from Lanczos iterates (4.20) are lost quickly due to numerical issues in floating point arithmetic, which also causes loss of tridiagonal structure in the matrix T_k. One way to tackle this issue is to re-orthogonalize vectors to generate a repaired basis for the columns of Q; however, as Hager [2001] mentions when the dimension of the problem or the number of stored Lanczos vectors are large, the overhead of maintaining the orthogonality becomes significant; moreover, for this solution one needs to store the Lanczos vectors which may require significant amount of memory that can be prohibitive in the context of large-scale deep learning problems. This re-orthogonalization also ensures obtain higher accuracy of the solution to model minimizer, which in turns can lead to less number of trust-region solves. Hager [2001] showed that the overhead of storing numerous Lanczos vectors is not needed when minimizing a quadratic on a trust-region boundary. Hager's analysis shows that only storing and minimizing in a subspace composed of 3 vectors is essential to reach the solution.

5.2.1 SSM Algorithm and SSM-A

The Successive Subspace Methods also aim to solve the constrained quadratic subproblem which arises in the trust-region framework. The original method introduced by Hager [2001] tackles the equality constrained version of the problem i.e.

$$\begin{array}{ll} \min_{s\in\mathbb{R}^n} & m_k(s) : \dfrac{1}{2}s_k^T H_k s_k + g_k^T s_k, \\ \text{s.t.} & \|s_k\| = \Delta. \end{array} \tag{5.1}$$

The author considers the equality constrained version of the quadratic program, this is justified by indicating that interior solutions (with respect to the ball constraint), can be obtained easily by a normal CG approach applied to the QP objective function; the same reasoning and strategy has been applied in the GLTR algorithm. The SSM algorithm aims to solve a sequence of subproblems in the reduced subspace $\mathcal{S}_j$ in order to generate a sequence of intermediate solutions converging towards the true optimum solution of the subproblem. The Krylov subspace methods such as CG, start minimization from small subspace $\mathcal{S}_0$, and form subproblem (5.2) for $\mathcal{S}_j$ as

$$\begin{aligned} \min_{s_j \in \mathcal{S}_k} m_j(d) \quad &= \frac{1}{2} s_j^T H_k s_j + g_k^T s_j \\ &\|s_j\| = \Delta. \end{aligned} \tag{5.2}$$

Krylov subspace methods then expand the solution s_j^* to higher dimensional $\mathcal{S}_k, k > j$ subspaces as they progress, while maintaining the nested property of these sequence of subspaces

$$\mathcal{S}_0 \subset \mathcal{S}_1 \subset \ldots \subset \mathcal{S}_k. \tag{5.3}$$

Also if the iterates progress to completion $\mathcal{S}_k$ will converge to spanning the full space $\mathbb{R}^n$.

Hager's approach relaxes the nested subspace property and therefore keeps the incumbent subspace dimension much lower than the original problem dimension. The subspace $\mathcal{S}_k$ involves a 4 dimensional sub-space spanned by the following vectors:

- Previous iterate(s_{j-1}): Ensuring that the value of the model objective is only decreased in the consecutive iterations.
- Updated gradient($As_{j-1} + g$): This direction ensures existence of a descent direction in the subspace, when the first-order condition is not satisfied.
- Left most eigenvector (v_1) : Will ensure not being trapped in saddle point solutions.
- SQP direction (s_{SQP}): Acceleration vector.

The SQP direction is obtained by applying the Newton's method to the SQP linear system:

$$\begin{aligned} (H + \sigma_k I)s_{\text{SQP}} + s_k \nu &= -g - (H + \sigma_k I)s_k, & (5.4) \\ s_k^T s_{\text{SQP}} &= 0, & (5.5) \end{aligned}$$

where ν is the update to the current estimate of the Lagrangian of the boundary constraint. Although later Hager and Park [2005] showed that only the first three vectors are essential for the SSM to be globally convergent (Theorem 1 of Hager and Park [2005]), it was stated adding the SQP direction helps providing faster locally quadratic convergence Hager [2001]. Here we state the important results of global SSM convergence due to Hager and Park [2005]:

Theorem 5.2.1. *If in each step of SSM, the subspace $\mathcal{S}_k$ contains vectors $\{-g - Hs_k, s_k\}$, and v_1 a left most eigenvector of matrix H, then SSM algorithm converges to global solution of* (5.1)*.*

In the deep learning context, the first issue to point is that obtaining vector v_1 can be a prohibitive computation, especially as there is no explicit representation of the matrix H available and also the dimension of the problem is really high. The second issue is that providing the acceleration vector in the actual implementation involves solving the SQP linear system, therefore each subspace $\mathcal{S}_k$ includes a vector which is generated using solvers such as MINRES (Paige et al. [2014]), to solve the linear system (5.4). We can immediately see obtaining the SQP direction using a black-box approach can be wasteful only to obtain one vector and not use information involved in solving the system. For these two reasons, we motivate the use of more efficient approaches to this problem.

One approach to tackle the mentioned drawbacks of SSM is the annulus method (Gratton [2012]) which hereby we denote by SSM-A; our approach in this chapter extends the SSM-A algorithm accompanied by numerical results showing gained speed up. SSM-A algorithm incorporates SSM strategy while maintaining the original CG iterations for solving the subproblem. To tackle the expensive eigenvector computation the algorithms maintains its estimate of $\hat{v}_k$ and as it progresses through the non-nested subspaces, dynamically updates the estimate.

Algorithm 18 SSM-A: Annulus based SSM

```
Require: $H$, $g$, and $\delta$
Require: $\epsilon \in (0,1)$, $\tau > 1$.
Require: Initial guesses for $v^*$ and $s^*$: $v_0$, $s_0$.
 1: Initializations: $j = 0$, $\hat{\sigma}_j = 0$, $\hat{s}_0 = 0$.
 2: $r_0 = -(g + Hs_0)$; $\gamma_0 = 0$; $p_0 = 0$;
 3: for $k = 1, \dots, 2n$ do
 4:     $[r_k, \alpha_k, p_k, \gamma_j] = \mathbf{cgUpdate}(H + \hat{\sigma}_j I, r_{k-1}, p_{k-1}, \gamma_{k-1})$;
 5:     $\hat{s}_k = \hat{s}_{k-1} + \alpha_k p_k$;
 6:     $\mathcal{A}_k = \{\hat{s}_k, r_{k-1}, r_k\}$;
 7:     $\mathcal{S}_k = \{s_{k-1}, v_{k-1}\} \cup \mathcal{A}_k$
 8:     Set $W = \text{basis}(\mathcal{S}_k)$;
 9:     $[s_k, \sigma_k, v_k, \lambda_k] = \mathbf{ssmUpdate}(H, g, W, \delta)$;
10:     if $p_k^T(H + \hat{\sigma}_j I)p_k \leq 0$ or $\Big| \|s_{k-1}\| - \|\hat{s}_{k-1}\| \Big| \geq \epsilon\delta$ then    ▷ Restart CG
11:         $j = j + 1$;
12:         $\hat{\sigma}_j = \sigma_k$; $\hat{s}_k = s_k$;
13:         $r_{k+1} = -(H + \hat{\sigma}_j I)\hat{s}_k - g$, $\gamma_{k+1} = 0$,
14:     end if
15:
16:     if $\|r_k\| \leq \epsilon \|g\|$ then
17:         break    ▷ Approximate solution found
18:     end if
19: end for
```

SSM-A as outlined in Algorithm 18, maintains two sequences of iterates $\{s_k, \hat{s}_k\}$ which produce approximate solutions for the subproblem (5.1), in addition, it approximates and updates the Lagrangian multiplier ($\hat{\sigma}$) corresponding to the boundary constraint. SSM-A incorporates a modified version of subspace solve which is outlined in Algorithm 19. This step finds an approximate solution to QP subproblem, by restricting it to subspace $\mathcal{S}_k$. This is easily done by assuming span W of $\mathcal{S}_k$ and the corresponding change of basis $s_k = Wu$ in (5.1), which yields problem (5.6). Since W has few number of columns (5 columns in this case), the subspace QP problem is a low dimensional problem, that is $(W^T HW)$ is a 5×5 matrix. Solving this subproblem due to its small dimensions can be achieved using external solvers. For our implementation we used the exact method of Moré and Sorensen [1983]; this method is similar to what GLTR uses to solve the reduced QP using tridiagonal matrix T as this exact solver needs to factorize this matrix; however, in our case the matrix is not necessarily in the tridiagonal format but since the dimension is low we can use this method without significant overhead. Furthermore finding the exact Eigen information is also possible using the exact solvers, for this purpose we used *eigs* subroutine of Matlab in our implementation.

Algorithm 19 SSM Update

```
1: **function**$[s, \sigma, v, \lambda] = $ **ssmUpdate**$(H, g, W, \Delta)$
2: Determine $(z^*, \eta^*)$ as the minimum eigenpair of $(W^T HW)z = \eta(W^T W)z$.
3: Determine $(u^*, \xi^*)$ by minimizing (5.2) in span of $W$:
```

$$\begin{array}{ll} \min u & u^T(W^Tg) + \frac{1}{2}u^T(W^THW)u, \\ \text{subject} & \|Wu\|_2 \le \Delta_k \end{array} \tag{5.6}$$

```
4: Set $v = Wz^*$, and $\lambda = \eta^*$.
5: Set $s = Wu^*$, and $\sigma = \xi^*$
6: **endfunction**
```

Initially as $\sigma_0 = 0$, SSM-A algorithm proceeds to solve the non-shifted system $Hs = -g$, meanwhile updating the SSM solution, the SSM at each iterate is constrained to the subspace spanned by vectors

$$W = \{r_{k-1}, r_k, s_k, v_{k-1}, \hat{s}_{k-1}\}, \tag{5.7}$$

which includes the previous and current residuals (r_{k-1}, r_k) with respect to CG iterates, the previous CG solution (s_k), an approximation of the left-most eigenvector v_{k-1}, and also the previous SSM iterate. As mentioned above, Theorem 5.2.1 states that convergence is ensured when current iterate, current residual and left-most eigenvector are provided, therefore as v_k asymptotically converges to v^* the convergence can be guaranteed. The algorithm starts by maintaining CG and SSM iterates simultaneously and updating their corresponding solutions with $\sigma = 0$ shift; it also updates the dual (shift) variable while updating $\sigma, \hat{v}$; whenever the CG solution encounters the trust-region boundary (for example in presence of negative curvatures), the algorithm detects that $\sigma = 0$ is no longer valid and therefore the shift parameter should be updated in order for CG to be able to produce the solution that satisfies the KKT optimality conditions (4.17),(4.18). This in effect requires a restarting scheme for the infeasible CG solution (s_k) to satisfy the trust-region constraint; the restart will use the updated SSM solution ($\hat{s}_k$) that lies on the boundary and update σ_k based on the approximation obtained through SSM as well as updating the CG residual to reflect the new gradient at the SSM point (Steps $11 - 14$ respectively). On the next iteration as (restarted) CG solution already lies on the boundary it is likely to proceed outside of the feasible area of trust-region. Using annulus rings, we allow CG solution to move past the region up to a certain threshold which allows producing new information for updating the SSM solution. The same logic is utilized in the GLTR implementation (Gould et al. [1999]), allowing Lanczos update iterates proceeding past the Steihaug-Toint point which effectively can improve the solution on the trust-region boundary. However, as mentioned we allow the discrepancy of CG and SSM solutions up to a specific threshold ($\tau\Delta$); here the condition $\|s_k - \hat{s}_k\| \geq \tau\Delta$ ensures that the CG solution conforms to the SSM solution as soon as it leaves the annulus rings.

As mentioned above, providing the left-most eigenvector is one of the important necessities of the general SSM algorithm, the SSM-A algorithm tries to approximate this quantity as it progresses through iterates, however, this can stall the convergence. On the other hand, next section shows developments that enabled CG solvers to obtain more information regarding eigenvalues by adding a small overhead of book keeping.

5.2.2 EigCG Algorithm

Stathopoulos and Orginos [2010] introduced a useful book keeping strategy that extends the unconstrained CG algorithm to produce the n_{ev} desired eigenvectors along with their eigenvalues. The original motivation for extracting this information was to solve systems of the form $Ax = b_i$ for different number of right-hand side vectors b_i. The EigCG algorithm starts by solving the first system using a CG method and gains eigen information as a byproduct of simple book keeping during iterations. The eigen information are then exploited in acquiring the solution for multiple systems ($Ax = b_i$) with different right-hand sides by using deflation techniques (Saad et al. [2000]).

However, for our usage, we exploit the generated eigen information for solving the QP subproblem more efficiently. The EigCG algorithm utilizes a methodology that benefits from the close relationship of the CG and Lanczos methods. That is concurrent to updating the CG iterates, information is reused to form the Lanczos tridiagonal matrix T. Step 6 of Algorithm 20 outlines how Lanczos information are updated as CG iterations progress (Step 4).

Algorithm 20 EigCG Algorithm (functional form)

```
Require: $s_0$, $H$, $\tau$
Require: $m$, $n_{ev}$ with $2n_{ev} < m$
 1: Initializations: $r_0 = -g - Hs_0$; $\gamma_{-1} = 0$; $p_{-1} = 0$; $\alpha_{j-1} = 1$;
 2: $T_0 = []$, $Q_0 = []$;
 3: for $j = 0, \dots,$maxitr do
 4:     $[r_{j+1}, \alpha_j, p_j, \gamma_j, q, t_u, t_d] =$ cgUpdate$(H, r_j, \alpha_{j-1}, p_{j-1}, \gamma_{j-1})$
 5:     $s_{j+1} = s_j + \alpha_j p_j$;
 6:     $[T_{j+1}, Q_{j+1}] =$ EigCGUpdate$(T_j, Q_j, q, t_u, t_d, m, n_{ev})$
 7:     if $\|r_{j+1}\| \leq \tau \|r_0\|$ then                ▷ Desired accuracy achieved
 8:         break;
 9:     end if
10: end for
```

It is well known that eigenvalues of matrix T, can be used as close approximations to the eigenvalues of A in the space spanned by Lanczos vectors. For stability of the algorithm and also to keep the overhead of eigenvalue extraction of T, the authors keep the search space from practical point of view, this means that the number of active Lanczos vectors kept, would be small. In other words, a buffer window of size m for storing the Lanczos information, where m is bigger than twice the number of desired eigenvalues ($m > 2n_{ev}$). Vectors are added to this buffering window through the CG iterates and as the window is filled with m columns, the algorithm restarts the Lanczos book keeping to keep the dimensions small before going to details we outline the algorithm.

Algorithm 21 EigCG Update

```
1: function$[T, Q, v_{\text{best}}]$ = EigCGUpdate$(T, Q, q_{new}, t_u, t_d, m, n_{ev})$
2: $[k, k] = \text{size}(T)$;
3: if k = 0 then
4:     $T = t_d$
5: else
6:     $T_{k+1,k+1} = t_d$ and $T_{k,k+1} = T_{k+1,k} = t_u$
7: end if
8: if k = m then
9:     Let $T$ and $\hat{T}$ denote $m$ and $m$-1 principal sub-matrices of T respectively;
10:    Let $Y$ store eigenvectors of $n_{ev}$ smallest eigenvalues of $T$;
11:    Let $\hat{Y}$ store eigenvectors of $n_{ev}$ smallest eigenvalues of $\hat{T}$;
12:    Let $W$ denotes an orthonormal basis for $\left(Y, \begin{bmatrix}\hat{Y}\\0\end{bmatrix}\right) \in \mathbb{R}^{m\times 2n_{ev}}$
13:    Let $E$ denote diagonal matrix of $2n_{ev}$ eigenvalues of $W^TTW$;
14:    Let $Z$ denote corresponding $2n_{ev}$ eigenvectors of $W^TTW$;
15:    Set $Q \leftarrow QWZ$; now $Q \in \mathbb{R}^{n\times 2n_{ev}}$
16:    Set $T = \begin{pmatrix} E & Q^THq_{\text{new}} \\ q_{\text{new}}^THQ & T_{k+1,k+1}\end{pmatrix}$
17: end if
18: Let $v_{\text{best}}$ denote vector $V$ with smallest Rayleigh Quotient.
19: Set $Q \leftarrow [Q, q_{\text{new}}]$
20: endfunction
```

Algorithm 21 outlines the book keeping incorporated in EigCG. The algorithm starts as regular CG applied to the system $Ax = b$ and gradually grows the square matrix T by augmenting with a diagonal $T_{k+1,k+1} = t_d$ and an off-diagonal element $T_{k,k+1}, T_{k+1,k} = t_u$ (steps 3-7); therefore the size of T increases by one at each iteration. When the size of T reaches $m \times m$, the restarting procedure shrinks the size of T to $2n_{ev} \times 2n_{ev}$ by extracting and condensing the necessary information into a smaller matrix.

The algorithm essentially incorporates a variation of the thick restart method for Lanczos basis that is used in the TRLAN package (Wu and Simon [2000]). TRLAN works similarly by extracting the eigen information of matrix T by solving $TY = YM$, and storing n_{ev} desired eigenvectors in the matrix Y. At the core of Lanczos method, the recurrence

$$AQ_k = Q_kT_k + \beta_k q_{k+1} e_k^T, \tag{5.8}$$

is the key that plays crucial role in the Lanczos method definition. Recurrence (5.8), is the main update rule for the matrix Q_k, where it stores the residual $(AQ_k - Q_kT_k)/\beta_k$ as the $k+1^{\text{th}}$ column of Q. On the m^{th} iterate, restarting scheme replaces T_m by $\hat{T} := Y^T T_m Y$ where Y includes the n_{ev} desired eigenvectors. The restart, in turn, makes $\hat{T}$ to be a diagonal matrix with the corresponding eigenvalues as its elements. This means that basically the Lanczos vectors Q_m are being replaced by the n_{ev} Ritz vectors $\hat{Q}_m = Q_mY$. Therefore as Lanczos iterates require maintaining orthogonality of the columns of Q, the new residual vector q_{m+1} should be re-orthogonalized against all n_{ev} vectors. The re-orthogonalization involves computing a vector of coefficients for the new Lanczos vector, stored in T, which keeps the above recurrence valid. Therefore the diagonal $\hat{T}$ matrix needs augmentation by a vector of coefficients $\hat{Q}_m H q_{m+1}$; this change makes matrix $\hat{T}$ loose its diagonal form to an arrowhead matrix form. As Wu and Simon [2000] demonstrate, with these changes, the recurrence (5.8) remains valid for the subsequent iterations involving q_{m+i}. Thus, the algorithm proceeds as a normal unrestarted Lanczos method by adding one diagonal $\hat{T}_{m+i,m+i}$ and an off-diagonal $\hat{T}_{m+i-1,m+i} = \hat{T}_{m+i,m+i-1}$ to the matrix $\hat{T}$.

The restarting strategy can be derived with any orthonormal basis of the k dimensional Krylov subspace. Therefore, Stathopoulos and Orginos [2010] enrich the search space not only by n_{ev} current Ritz vectors, but also n_{ev} vectors from the previous step $m-1$. However, since Y_{m-1} vectors belong to a smaller subspace of $\mathbb{R}^{m-1}$, the authors propose lifting the vectors to the $\mathbb{R}^m$ subspace by augmenting them with an additional 0 for the last dimension. The choice of these vectors is inspired by locally optimal conjugate gradient method of D'yakonov [1983]. Now since the new vectors $\left(Y_m, \begin{bmatrix} \hat{Y}_{m-1} \\ 0 \end{bmatrix}\right)$ do not necessarily form an orthonormal basis, step 12 of Algorithm 21 needs to extract an orthonormal basis W of their span to proceed with the thick restart.

After restarting, the matrix Q is replaced by the new vectors QWZ that estimate the desired eigenvectors (step 15). Moreover, the new T_m matrix will have arrowhead form outlined in equation (5.9)(step 16), this will add some overhead when computing the Ritz information of matrix T as the desired tridiagonal form is lost.

Throughout the first m-1 iterations, matrix T is identical to the original Lanczos tridiagonal matrix. This can be seen in steps 3-7, where the matrix T is augmented by the new Lanczos tridiagonal elements t_d and t_u. Similarly, the matrix Q, which is updated in step 19, contains the Lanczos vectors computed during the CG update as the normalized residuals (see step 8 in Algorithm 14).

When T becomes an $m \times m$ matrix, an internal restart occurs that reduces the dimension of T to $(2n_{ev}+1) \times (2n_{ev}+1)$, implying Q and T will never be large than $n \times m$ and $m \times m$ respectively. To better describe this part of the algorithm, we denote the first $m-1$ columns of the matrix Q by $\hat{Q}$. At this point, the following properties remain true

$$Q^T H Q = T \text{ and } \hat{Q} H \hat{Q} = \hat{T},$$

where $\hat{T}$ denotes the $m-1$ principal submatrix of T. We next compute the n_{ev} smallest eigenvalues of T and $\hat{T}$ and store their corresponding eigenvectors in matrix Y and $\hat{Y}$ respectively. As hinted before, QY and $\hat{Q}\hat{Y}$ could be thought of as two consecutive estimates for approximating the n_{ev} smallest eigenvalues of H.

The next computation denotes the fundamental step of the algorithm, proof of its surprising effectiveness still pending; the algorithm simply forms the new (restarted) Q matrix using columns that form an orthonormal basis for the

$$\text{span}(QY, \hat{Q}\hat{Y}),$$

with the additional property that $Q^T HQ$ forming a diagonal matrix. Thus, a mechanism is needed for forming an orthonormal basis matrix for $(QY, \hat{Q}\hat{Y})$. Performing a Gram-Schmidt procedure on $(QY, \hat{Q}\hat{Y})$ is a simple yet less cost-effective method for computing the new basis. However, it is desired to work in the reduced $\mathcal{O}(m)$ dimension. Following relation

$$(QY, \hat{Q}\hat{Y}) = QA, \text{ where } A \overset{\Delta}{\underset{\text{def}}{=}} \left(Y, \begin{bmatrix} \hat{Y} \\ 0 \end{bmatrix}\right)$$

shows that rather than orthogonalzing matrix $(QY, \hat{Q}\hat{Y})$ with the size of $n \times 2n_{ev}$, we can instead orthogonalize the much smaller matrix A with the size of$m \times 2n_{ev}$, generating the othornormal basis matrix W in Step 12 of Algorithm 21. Matrix W then satisfies following properties

$$\text{span}(W) = \text{span}\left(Y, \begin{bmatrix} \hat{Y} \\ 0 \end{bmatrix}\right) \text{ and } W^T W = I.$$

At this point the matrix Q could have been replaced by the projection QW; however, the product $(QW)^T H(QW)$ results in a dense matrix, whereas a sparse (preferably diagonal) form is required to for the restarted T matrix. Thus, forming the eigen decomposition of $(QW)^T H(QW)$

$$ZEZ^T = (QW)^T H(QW) = W^T T W,$$

we have E as a diagonal matrix of eigenvalues and Z as the matrix of corresponding eigenvectors. Therefore a diagonal matrix can be extracted as

$$E = (QWZ)^T H(QWZ).$$

This ultimately enables the redefinition of $Q \leftarrow QWZ$ in Step 15 of Algorithm 21. The new redefined matrix Q now satisfies

$$Q^T HQ = E \text{ and } Q^T Q = I.$$

Finally we augment matrix Q with the current Lanczos vector in Step 19, which implies that Q satisfies

$$Q^T HQ = \begin{pmatrix} E & Q^T H q_{\text{new}} \\ q_{\text{new}}^T HQ & T_{k+1,k+1} \end{pmatrix}, \tag{5.9}$$

which justifies the corresponding update to T in Step 17 of Algorithm 21 in addition to the orthogonal property of $Q^TQ = I$. On exit T will have the classical arrow-head nonzero structure reminiscent of other restarted Lanczos algorithms.

This conclude our description of Algorithm 21 based on Stathopoulos and Orginos [2010]. Remarkably, without modification of the CG residuals r_{j+1} after such updates, significant numerical results demonstrate that the first n_{ev} vectors of Q_{j+1} converge to the first n_{ev} eigenvectors corresponding to the n_{ev} smallest eigenvectors of H with convergence rates comparable in accuracy to unrestarted (stabilized and hence significantly more costly) Lanczos methods (Stathopoulos and Orginos [2010]).

5.3 EigCG-TR

5.3.1 Motivation

In order to motivate our extension to the SSM-A algorithm, first the importance of the left-most eigenvector v_1 of the Hessian matrix is emphasize. Section 5.2.1 briefly hinted on the importance of obtaining v_1 for ensuring the global convergence of the SSM method. The importance of this vector is two fold for the model minimizer of the trust-region as the following theorem depicts:

Theorem 5.3.1. *Suppose (s^*, σ^*) denotes a solution to*

$$\begin{aligned} \min m_k(s) &= \frac{1}{2} s_k^T H_k s_k + \nabla f(x_k)^T s_k \\ & \|s_k\| \leq \Delta. \end{aligned}$$

Given any constant $C > \lambda_1$, we can decompose the solution as

$$s^* = v + r, \ \textit{with } \|r\| \leq \frac{\|g\|}{C - \lambda_1} \textit{ and } \|s^*\| = \|v\| + \|r\|,$$

where $v \in \operatorname{span}\{v_i : \lambda_i < C\}$ and $r \in \operatorname{span}\{v_i : \lambda_i \geq C\}$. Further, whenever H is not positive-definite,

$$\lim_{\Delta \to \infty} \frac{\|s^* - v\|}{\|s^*\|} = 0.$$

Proof. From equations (4.17) and (4.18), we know that if $\sigma^* > -\lambda_1$,

$$s^* = \sum_{i=1}^{n} \left(\frac{g^T v_i}{\sigma^* + \lambda_i} \right) v_i$$

And if $\sigma^* = -\lambda_1$, then

$$s^* = \tau v_1 + \sum_{\lambda_1 < \lambda_i} \left(\frac{g^T v_i}{\sigma^* + \lambda_i} \right) v_i$$

Thus in either case we can partition the solution as

$$s^* = v + \left(\sum_{\lambda_i \geq C} \frac{g^T v_i}{\sigma^* + \lambda_i} v_i \right).$$

where $v \in \text{span}\{v_i : \lambda_i < C\}$. And we may define $r = s^* - v$. Then

$$\|r\|^2 = \sum_{\lambda_i \geq C} \left(\frac{g^T v_i}{\sigma^* + \lambda_i} \right)^2 \leq \sum_{\lambda_i \geq C} \left(\frac{g^T v_i}{\lambda_i - \lambda_1} \right)^2 \leq \sum_{i=1}^{n} \left(\frac{g^T v_i}{C - \lambda_1} \right)^2,$$

which implies $\|r\| \leq \|g\|/(C - \lambda_1)$.

For the second part, since H is not positive-definite we have $\sigma^* > 0$ due to optimality condition of $(H + \sigma^* I) \succeq 0$. The complementarity condition $\sigma^*(\Delta - \|s^*\|) = 0$ implies $\|s\| \to \infty$ as $\Delta \to \infty$. Substituting $\|r\|$ from above we can rewrite the limit as:

$$\lim_{\Delta \to \infty} \frac{\|s^* - v\|}{\|s^*\|} = \lim_{\|s^*\| \to \infty} \frac{\|r\|}{\|s^*\|} \leq \lim_{\|s^*\| \to \infty} \frac{\|g\|}{\|s^*\|(C - \lambda_1)} = 0.$$

□

The above theorem implies that in the nonconvex case as the trust-region radius grows $\|v\|$ component of the $\|s^*\|$ plays a more important role in the solution compared to $\|r\|$. As shown above, $v \in \text{span}\{v_i : \lambda_i < C\}$ means that v component of s^* is comprised of eigenvectors corresponding to the smallest eigenvalue. Therefore having a more accurate approximation of the eigenvectors can speed up solving the QP.

5.3.2 EigCG-TR Algorithm

In this section we introduce the new algorithm EigCG-TR, which is a variant of SSM-A algorithm; As the previous section motivates the importance of left-most eigenvectors, we extend SSM-A to allow better utilization of existing information in order to exploit the problem structure and provide faster convergence. Algorithm 22 outlines the proposed EigCG-TR algorithm.

As mentioned in section 5.2.1, rather than "out-sourcing" the solution of the linear SQP system to MINRES, SSM-A involves an internal CG solver for producing the acceleration vector. However, this transition does not exploit the full potential of the CG solver. Since the CG algorithm readily produces the Lanczos information, we can easily use the existing information to extract eigen information regarding the curvature matrix.

On an extreme end, exploiting the existing information to the extent of storing the full Lanczos information will result in an algorithm similar to GLTR which in addition dynamically updates the eigenvalue and eigenvector approximations based on information on the tridiagonal matrix T.

However, on one hand storing entire basis of Lanczos vectors is not practical in the deep learning context. On the other hand, Stathopoulos and Orginos [2010] have shown that the eigenvalue information can be extracted using far less number of Lanczos vectors by gradually solving small $m \times m$ eigenvalue decomposition problems. More precisely, the algorithm integrates the **EigCGUpdate** step, which essentially adds a procedure similar to the thick restart Lanczos book keeping (Wu and Simon [2000]) to SSM-A algorithm.

As the inner SSM algorithm requires an estimate of the eigenvector corresponding to smallest eigenvalue, we can provide EigCG book keeping updates added to CG to maintain a running estimate of this quantity. Accommodating the EigCG updates into SSM-A, the inner SSM algorithm readily gets much better estimates for the left-most eigenvector which will make the progress faster.

The following demonstrates the progress of estimates through SSM iterations. While for $\sigma_j = 0$, $s = \hat{s}_k$ can be computed directly by applying the CG algorithm

Algorithm 22 EigCG-TR

```
Require: $H$, $g$, and $\delta$
Require: $\epsilon \in (0,1)$, $\tau > 1$.
Require: $m$, $n_{ev}$ with $2n_{ev} < m$
Require: Initial guesses for $v^*$ and $s^*$: $v_0$, $s_0$.
 1: Initializations: $j = 0$, $\hat{\sigma}_j = 0$, $\hat{s}_0 = 0$.
 2: $r_0 = -(g + Hs_0)$; $\gamma_0 = 0$; $p_0 = 0$;
 3: for $k = 1, \ldots, 2n$ do
 4:     $[r_k, \alpha_k, p_k, \gamma_j, q, t_u, t_d] =$ cgUpdate$(H + \hat{\sigma}_j I, r_{k-1}, \alpha_{k-1}, p_{k-1}, \gamma_{k-1})$;
 5:     $\hat{s}_k = \hat{s}_{k-1} + \alpha_k p_k$;
 6:     $[T_{k+1}, Q_{k+1}, \hat{v}_k] =$ EigCGUpdate$(T_k, Q_k, q, t_u, t_d, m, n_{ev})$
 7:
 8:     $\mathcal{A}_k = \{\hat{s}_k, \hat{v}_k, r_{k-1}, r_k\}$;
 9:     $\mathcal{S}_k = \{s_{k-1}, v_{k-1}\} \cup \mathcal{A}_k$
10:     Set $W = \text{basis}(\mathcal{S}_k)$;
11:     $[s_k, \sigma_k, v_k, \lambda_k] =$ ssmUpdate$(H, g, W, \delta)$;
12:     if $p_k^T (H + \hat{\sigma}_j I) p_k \leq 0$ or $\Big| \|s_{k-1}\| - \|\hat{s}_{k-1}\| \Big| \geq \epsilon\delta$ then    ▷ Restart CG
13:         $j = j + 1$;
14:         $W = [Q_{k+1}, s_k]$;
15:         $[s_k, \sigma_k, v_k, \lambda_k] =$ ssmUpdate$(H, g, W, \delta)$;
16:         $\hat{\sigma}_j = \sigma_k$; $\hat{s}_k = s_k$;
17:         $r_{k+1} = -(H + \hat{\sigma}_j I)\hat{s}_k - g$, $\gamma_{k+1} = 0$, $\alpha_{k+1} = 0$.
18:     end if
19:
20:     if $\|r_k\| \leq \epsilon \|g\|$ then
21:         break    ▷ Approximate solution found
22:     end if
23:     $\Delta Q = Q(s_{k-1}) - Q(s_k)$    ▷ Note that $\Delta Q \geq 0$
24:     if $Q(s_k) \leq -\max(\tau \Delta Q, \epsilon\delta)$ then
25:         break;    ▷ Q-stall termination, asymptotically inactive
26:     end if
27: end for
```

to $Hs = -g$.

Lemma 5.3.2. *Let s^* be the optimal solution to the trust-region subproblem in subspace $\mathcal{S}_k$ and let λ^* be the minimum eigenvalue of H. For any subspace sequence $\mathcal{S}_k$ defined in Algorithm 22 we have*

1. $m_k(s^*) \leq m_k(s_k) \leq m_k(s_{k-1})$,

2. $\lambda^* \leq \lambda_k \leq \lambda_{k-1}$,

3. $s_k^T g \leq 0$,

where $m_k(s)$ denotes the objective function of QP. For $k \geq 1$, $m_k(s_k) \leq m_k(s_c)$, where s_c denotes the corresponding Cauchy point solution.

Proof. The definition of Algorithm 19 requires that $\|s_j\| \leq \Delta$ and $\|v_j\| = 1$ for all iterates j. By construction, W_k in Algorithm 22 satisfies $W_k e_1 = s_{k-1}$ and $W_k e_2 = v_{k-1}$. Hence for right-hand inequality of (1) we have

$$m_k(s_k) = \min_{\|W_k u\| \leq \Delta} m_k(W_k u) \leq m_k(W_k e_1) = m_k(s_{k-1}),$$

similarly for (2)

$$\lambda_k = \min_{\|W_k z\|=1} z W_k^T H W_k z \leq e_2 W_k^T H W_k e_2 = \lambda_{k-1}.$$

Moreover, by definition $m_k(s^*)$ and λ^* are the respective lower bounds on $m_k(s_k)$ and λ_k for all vectors in $\mathbb{R}^n$ and thus left-hand inequalities for (1) and (2) hold true. For part (3) assume $s_k^T g > 0$ which results in $m_k(-s_k) < m_k(s_k)$; however, as $-s_k \in \mathcal{S}_k$, it contradicts optimality of subproblem step 5 in Algorithm 19.

The final assertion ($\psi(s_k) \leq \psi(s_c)$ for $k \geq 1$) follows by observing that the Cauchy point is obtained by solving subproblem (5.6) with $W = \{g\}$. □

The following theorem ensures SSM-A algorithm 22 is globally convergent as long as v_k converges to v^*.

Theorem 5.3.3. *If the global trust-region solution is interior, then Algorithm 22 reduces to CG on $Hs = -g$, so the algorithm is globally convergent. Otherwise, Algorithm 22 converges to a point satisfying*

$$(1 - \tau_1)\Delta \leq \|\hat{s}\| \leq (1 + \tau_1)\Delta, \tag{5.10}$$

$$\|(H + \hat{\sigma}_j I)\hat{s} + g\| \leq \tau_2 \|g\|, \tag{5.11}$$

whenever v_k converges to v^*.

Proof. If the solution lies within the interior of trust-region boundary, H will be positive-definite and stepsize $\alpha_k > 0$ for every iteration k. Furthermore, r_k and r_{k+1} (which are simply multiples of the Lanczos vectors q_k and q_{k+1}, respectively) are contained in the subspace $\mathcal{S}_k$ at each iteration. Now since CG is not being restarted, we can use expanding subspace property of CG. Theorem 4.3.1, states that during the unmodified CG solve $\hat{s}_k$ is the minimizer of the quadratic function on the set

$$\{s_k = s_0 + \text{span}\{p_0, p_1, \ldots, p_{k-1}\}\} \tag{5.12}$$

this essentially means that the SSM solution s_k, starting at s_{k-1} cannot further improve the CG solution on the subspace spanned by r_k, r_{k-1}. Moreover, the additional eigenvector estimate (in SSM compared to CG) is restricted to the same space and thus is not adding any more dimensions to the subspace. Therefore we can conclude that $\hat{s}_k = s_k$ at every iteration, thus Step 12 of the algorithm is not entered, therefore $\hat{\sigma}_j = 0$ for all j, and $\hat{s}_k$ simply denotes the classical CG iterates.

If the solution lies on the boundary, two cases could be identified. The first considers the case where σ_j is modified only a finite number of times. This implies that there exists an iteration K such that for all $k > K$, $\hat{s}_k$ satisfies (5.10). This means that after iteration K the standard CG method (without restarts) is being applied to the system $(H + \hat{\sigma}_K I)s = -g$. Therefore by global CG properties of finite convergence, we are assured that $\|(H + \hat{\sigma}_K I)\hat{s} + g\|$ converges to 0; thus, as a result condition (5.11) holds for $j = K$.

The second case arises when $\hat{\sigma}_j$ is modified infinitely often. In this case, $\mathcal{S}_k$ contains the vectors s_k and $r_k = (H + \hat{\sigma}_j I)s_k + g$ infinitely often. By noting that $r_k - \sigma_j s_k = Hs_k + g$, also observing that by definition the SSM subspaces include s_k and r_k; therefore, subspaces implicitly include a corresponding quadratic gradient at s_k, that is $\nabla m(s_k) = Hs_k + g$. Thus, whenever $v_k \to v^*$, as v_k is contained in the subspace $\mathcal{S}_k$, global convergence results of Hager and Park [2005] applies Theorem 5.2.1 to the subsequence of subspaces $\mathcal{S}_k$ and conclude that Algorithm 22 converges. $\square$

Theorem 5.3.3 ensures convergence as long as the left-most eigenvector is obtained asymptotically. However Hager [2001], initially generates a set of Lanczos vectors q and finds a good estimate on this subspace and then improves the approximation through the SSM subspaces $\mathcal{S}_k$ as SSM algorithm proceeds. The second strategy can be observed in the SSM-A algorithm (and subsequently in EigCG-TR) by improving the approximation through the sequence of subspaces. The addition of the EigCG strategy and adding Lanczos book keeping can be thought as an accelerating strategy which exploits eigCG algorithm by maintaining thick restart basis through the iterates.

5.4 Deep Learning Application

The impetus of introducing the EigCG-TR method is to have better solvers that can handle nonconvexities arising in the quadratic model of second-order methods. Section 4.1.3 of the previous chapter discusses the importance of CG based solvers that continue improving the solution when a direction of nonpositive curvature occurs as well as another important feature which is the accuracy of the model minimizers when nonconvexity arises.

Increased accuracy of these models is obtained by further improvements towards the true model minimizer of the trust-region QP once the trust-region boundary is encountered. Algorithm 12 demonstrates an approximate solution strategy used in the truncated CG setting. The algorithm projects the solution onto the trust-region boundary in two cases. First, when a singular or negative curvature direction is encountered (step 5); second, case when solution lies outside of trust-region boundary regardless of negative curvature (step 10).

Therefore, due to this truncation, the output of the algorithm may not be accurate approximation to the true model minimizer. The bound mentioned by Martens [2010] for the difference of the Steihaug-Toint solution vs the global true model minimizer upper bounded by $\frac{1}{2}m_k(s^*)$, only applies on the convex QP function. The bound initially introduced by Yuan [2000] shows that when truncated CG method is applied to a convex function the solution produces at least half the reduction of the true global minimizer.

$$m_k(s_{\mathrm{TR}}) \leq \frac{1}{2} m_k(s^*) \tag{5.13}$$

This although may seem like a reasonable compromise, does not hold on the non-convex case; due to nature of deep learning problems, this case can occur if the true Hessian is used instead of Gauss-Newton approximation. As a simple example of bound not holding for the nonconvex case, consider

$$\begin{array}{ll} \min m(s)_{s\in\mathbb{R}^2} & s^T \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \frac{1}{2} s^T \begin{bmatrix} -10^6 & 0 \\ 0 & 10^6 \end{bmatrix} s \\ \text{subject} & \|s\|_2 \leq 1, \end{array}$$

Since $\bar{s} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is a feasible solution, we have that $m(s^*) < -\frac{10^6}{2} = m(\bar{s})$. However, as $g^T B g = 0$, the Steihaug-Toint algorithm would exit immediately, with

$$s_{ST} = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \quad \Rightarrow \quad m(s_{ST}) = -2/\sqrt{2} > \frac{1}{2} m(s^*).$$

As Gould et al. [1999] showed in their experiments, for the nonconvex QP model, progressing past the Steihaug-Toint point shows better improvement on the global trust-region model solution. Basically, this means that having more accurate model minimizers, CG can produce better directions for model minimizer by initially producing more accurate solutions; this, in turn, can lead to faster convergence towards the goal of DNN training. Therefore, the number of Hessian-vector products although increased initially can be reduced when considered in overall. The numerical results show that compared to regular CG solver the new algorithm can achieve better training loss in significantly less number of Hessian-vector products, due to the better model minimizers. Moreover, using the eigenvector information can help further reduce the number of Hessian-vector products.

5.5 Numerical Results

In order to show the benefits of EigCG-TR algorithm, we implemented both SSM-A and EigCG-TR in Matlab and used it as the inner model solver for trust-region subproblem. Also to benchmark the results, the standard Levenberg-Marquardt method of Martens [2010] was used. In the deep learning context, Hessian-vector products are computationally heavy bottlenecks for the second-order solvers; therefore, comparisons in addition to the loss measure, also consider the number of total Hessian-vector product, used by each algorithm.

For computational tests, the tolerance of $\eta = 10^{-3}$ for the residual stopping criteria of truncated Newton $\|r_k\| \leq \eta\|r_0\|$ was chosen; moreover, the stalling criteria was also used in all algorithms for termination. Also, CG-backtracking strategy of Martens [2010] was applied to improve the quality of the update direction found through the CG iterations.

5.5.1 MNIST Dataset

For the EigCG-TR, $n_{ev} = 1$ was used as the number of desired eigenvectors to be generated by **EigCGUpdate** step 6 of Algorithm 20, also for the memory window size, $m = 3$ was chosen. The MNIST dataset was used in two different settings of shallow and deep networks, containing 60,000 samples for training and 10,000 for validating the trained model.

Shallow Network

For this network we used two hidden layers of sizes $\{400, 150\}$ for classification, giving rise to network architecture of $\{784, 400, 150, 10\}$; although this is a shallow network, it consists of 375,660 variables for the optimization problem. Since classification task is desired, the softmax/cross-entropy canonical pair was used for the output layer and loss function respectively. The sigmoid function was chosen for the nonlinearity of the hidden layers.

Four different settings of algorithms were tested for 500 iterations with monitoring measures such as training loss, validation loss, and misclassification rate. Second-order algorithms heavily rely on multiplications with curvature matrix as this procedure is the most computationally involving step of the algorithm. Therefore, as our original motivation was to maximally exploit the available information, the number of calls to this function was counted for all algorithms and monitored at each iteration.

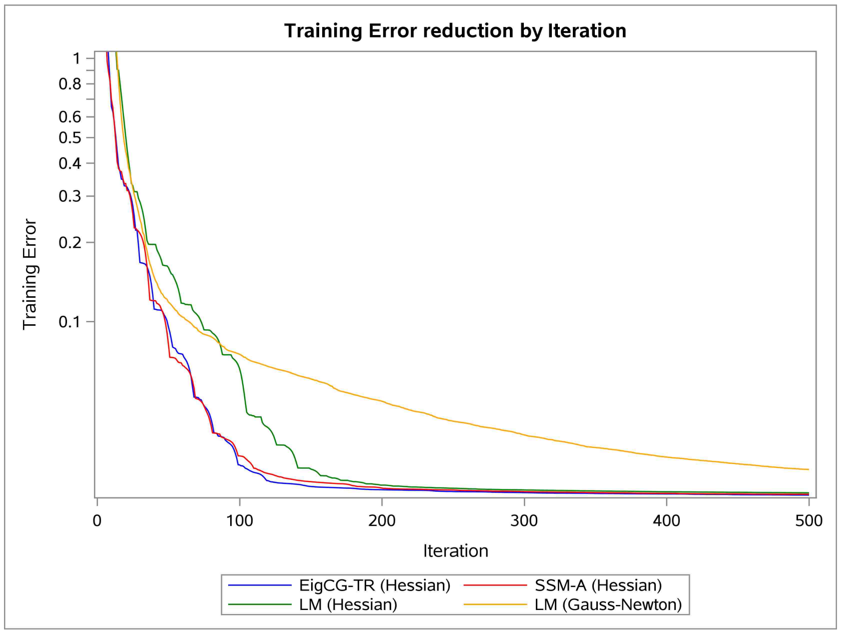


Figure 5.1: Training loss on each iteration of EigCG-TR, SSM-A, Levenberg-Marquardt (Hessian and Gauss-Newton)

Figure 5.1 shows that algorithms involving Hessian are more successful in the rapid reduction of the training loss compared to LM (Gauss-Newton), as they utilize a more accurate QP model. Both EigCG-TR and SSM-A algorithms outperform the LM (Hessian) method by minimizing the training value in a fewer number of iterations. However, from this plot, their benefits are not clear. We use the same information, but replace the horizontal axis to reflect the number of curvature matrix vector products, which will translate to number of multiplications of vectors by

Hessian or Gauss-Newton in the corresponding cases.

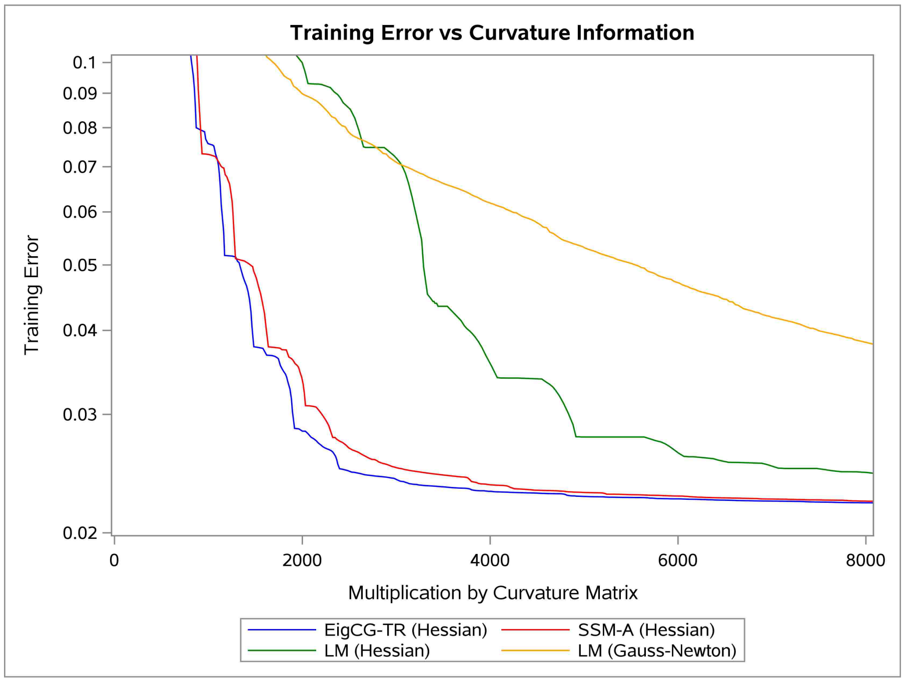


Figure 5.2: Training loss per Matrix-vector product of EigCG-TR, SSM-A, Levenberg-Marquardt (Hessian and Gauss-Newton)

Figure 5.2 demonstrates that both EigCG-TR and SSM-A use a much smaller number of curvature information compared to the LM methods. Moreover, utilizing the eigenvector information in the CG solve procedure has resulted in a faster convergence for the QP model and therefore reducing the total number of CG iterations which in turn results in the gap between EigCG-TR and SSM-A.

Although, reducing training loss is the goal of the optimization method, our ultimate desire is to minimize the validation loss function to improve the generalization of the trained network on the unforeseen data. Plotting the validation error information shows the same trend for these algorithms.

The figure 5.3 shows the multiplications counts required for reducing the validation loss by 0.01 unit for each algorithm. We can see that both SSM based methods can achieve better results in fewer number of multiplications.

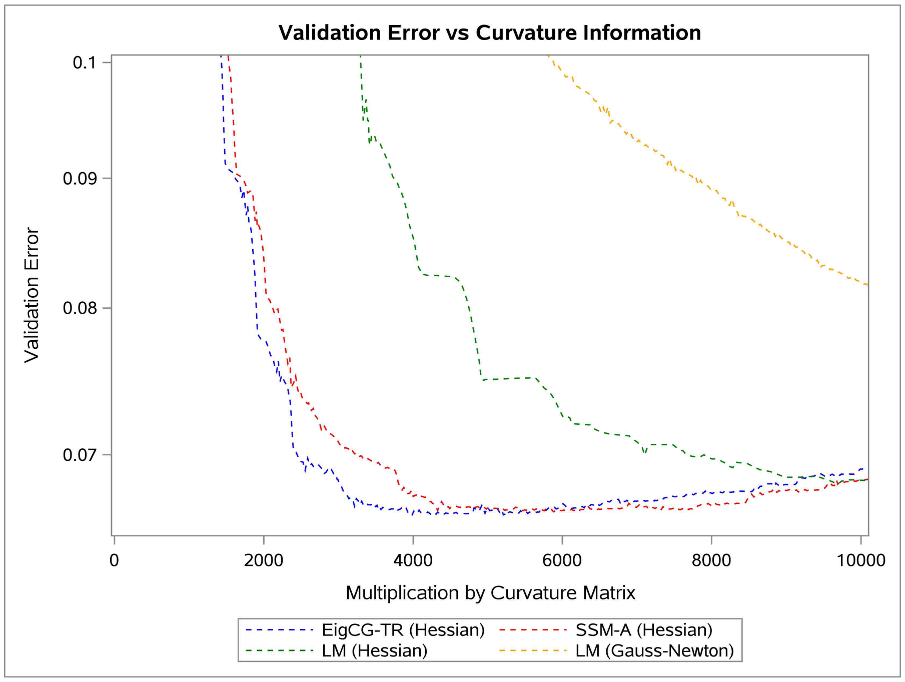


Figure 5.3: Validation loss on each iteration of EigCG-TR, SSM-A, Levenberg-Marquardt (Hessian and Gauss-Newton)

To avoid over-fitting the training data often early stopping criteria is utilized and therefore the algorithms suspend further optimization if the validation error is not improved after a specific number of iterations. Table 5.1 summarizes the results by showing the last iteration for which the validation error seized to improve.

Table 5.1: Shallow Network: Performance summary of the training algorithms

Method	Multiplication	Train Loss	Valid Loss	Error(%)
EigCG-TR	3961	0.023063	0.06623	1.46
SSM-A	5929	0.022686	0.066459	1.50
LM(Hessian)	>20000	0.022438	0.065208	1.56
LM(GN)	13700	0.027412	0.074495	1.68

Table 5.1 shows that by taking small steps and slow convergence, the LM(Hessian) method can reduce the validation loss further than other algorithms; however, this is

obtained with a huge cost of having more than 20,000 multiplications with the Hessian matrix, although the obtained misclassification error for SSM based variants is more interesting. The table shows that both EigCG-TR and SSM-A algorithms are achieving the same accuracy by respectively consuming far less number of curvature information than both LM methods.

Deep Network

For the deep setting network, we used the architecture used by Martens [2010], based on the networks initially introduced by Hinton and Salakhutdinov [2006]; The network has 9 layers which correspond to sizes of

$$784, 1000, 500, 250, 30, 250, 500, 1000, 10.$$

The nonlinearity of the hidden layers is setup using the sigmoid function, except for the middle layer which involves 30 hidden units for which the linear activation was chosen. The main impetus of this networks comes from autoencoder architecture to allow for noise reduction in the handwritings.

Figure 5.4 shows the training loss per each iteration of the Hessian-free methods. The optimization setting for this deep network was a total limit of 500 outer iterations, we also put a budget on the number of multiplications to be 18500 total passes or Hessian evaluation. The figure 5.4 shows that Hessian based methods again outperform the GN method. Where EigCG-TR is slightly reducing the error at a faster rate compared to SSM-A, however, both SSM based methods are outperforming the Levenberg-Marquardt method.

We can see that when plotting the error reduction by the number of curvature information products as figure 5.5 demonstrates, the gap among the performance of algorithms gets more clear. As figure shows, the SSM based algorithms consume a fewer number of matrix vector products to get to lower accuracy.
Figure 5.6 shows that the validation loss reduction nearly mimics training loss reduction; therefore, SSM based approaches reduce the error at a faster rate compared to Levenberg-Marquardt methods. Although the EigCG-TR gets the best performance, however, we can see that towards the 460th iteration the error drops to a

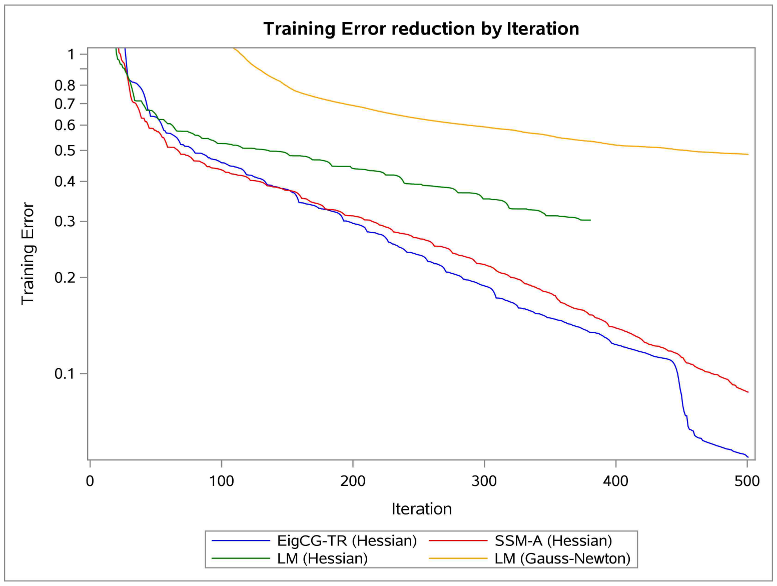


Figure 5.4: Training loss on each iteration of EigCG-TR, SSM-A, Levenberg-Marquardt (Hessian and Gauss-Newton)

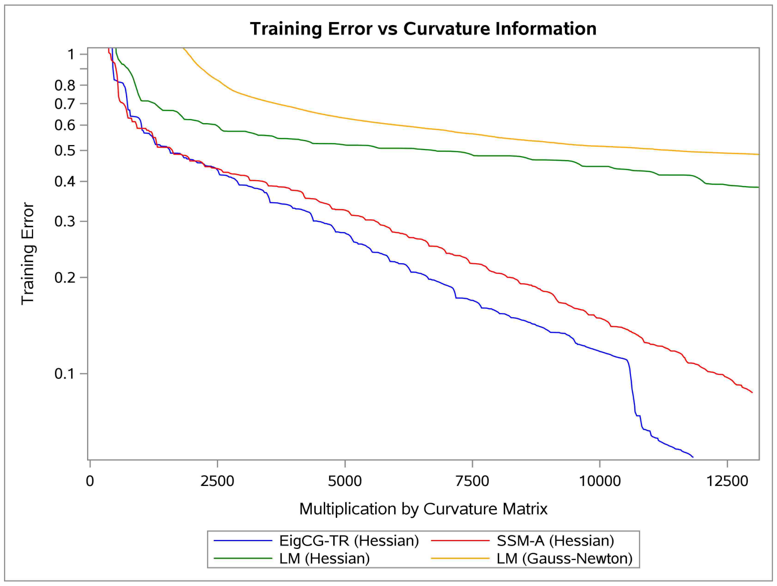


Figure 5.5: Training loss per Matrix-vector product of EigCG-TR, SSM-A, Levenberg-Marquardt (Hessian and Gauss-Newton)

minimum and afterward, it starts some over-fitting on the training data-points.

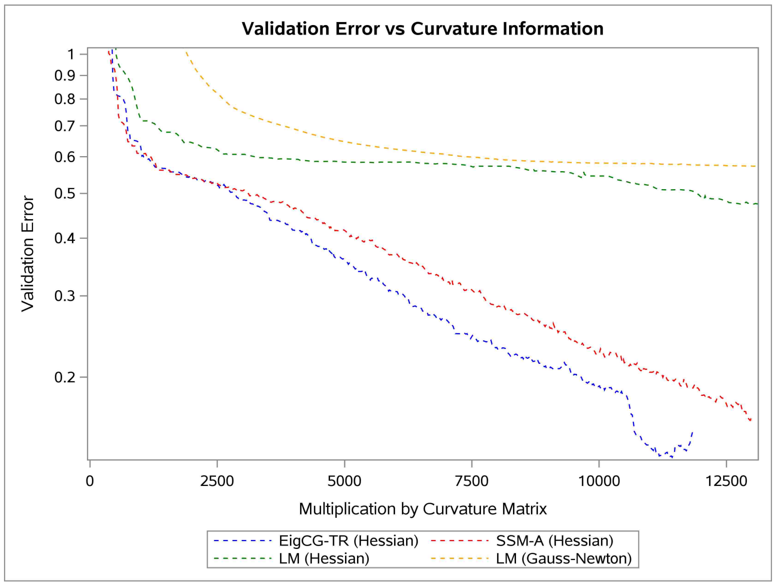


Figure 5.6: Validation loss on each iteration of EigCG-TR, SSM-A, Levenberg-Marquardt (Hessian and Gauss-Newton)

Table 5.2 summarizes the results obtained with the mentioned algorithms. EigCG-TR produces the lowest training and validation error with using the fewest multiplications by curvature matrix for this dataset. The SSM-A algorithm is slightly worse, however both algorithms train networks with 1.94% misclassification rates.

Table 5.2: Deep Network: Performance summary of the training algorithms

Method	**Multiplication**	**Train Loss**	**Valid Loss**	**Error (%)**
EigCG-TR	11437	0.058705	0.13404	1.94
SSM-A	12971	0.087587	0.16115	1.94
LM(Hessian)	18145	0.30273	0.38756	2.11
LM(GN)	13253	0.48407	0.57031	2.26

5.5.2 Curves Dataset

Further examining the new algorithm, we used the curves dataset from Hinton and Salakhutdinov [2006]. This dataset contains synthetic data of curve image information which were generated randomly by choosing three random points in a two-dimensional plane. We used the same deep network architecture (as described in Hinton and Salakhutdinov [2006]) which includes 13 layers with

$$\{784 - 400 - 200 - 100 - 50 - 25 - 6\}, \{25 - 50 - 100 - 200 - 400 - 784\}$$

architecture, where the encoder and decoder layers are segregated by braces. All hidden layers use sigmoid activation except the last encoder layer which consists of 6 linear neurons. The motivation for using 6 neurons on the encoder layer is that originally given the 3 random pounts, these curves only relied on 6 quantities; therefore, the encoder segment of the network aims to implicitly project the image pixel data into the true intrinsic dimension of the problem. Moreover, the decoder layer will aim to regenerate the pixel data points from this 6-dimensional projection. Figure 5.7 shows 25 random data points generated by visualizing the pixel data.

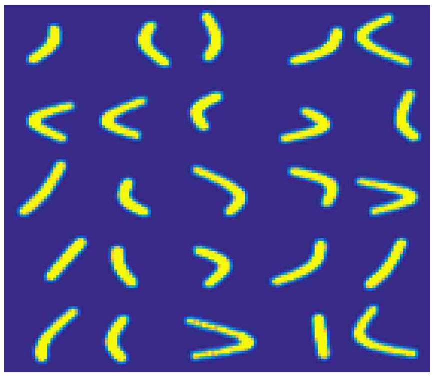


Figure 5.7: Visualized curves image dataset

Since training autoencoders are generally considered as a harder task compared to classification, using the same solvers as the MNIST classification case, the n_{ev} was increased to monitor 3 vectors, similarly the buffer window m was increased to

15 vectors for the EigCG-TR solver. The same setting as previous test was chosen for runs, however, the initial testing showed that utilizing the Gauss-Newton matrix produces a more suitable quadratic model compared to full Hessian. Figure 5.8 shows how each algorithm performs by measuring the reduction in the training loss per consumed information from curvature. We can see that the Gauss-Newton fed to Levenberg-Marquardt (LM) method is converging faster than the methods that use Hessian (including LM). Furthermore, SSM-A and EigCG-TR closely converge to error level of LM(GN) where as LM with Hessian struggles to reduce the training loss as fast as others.

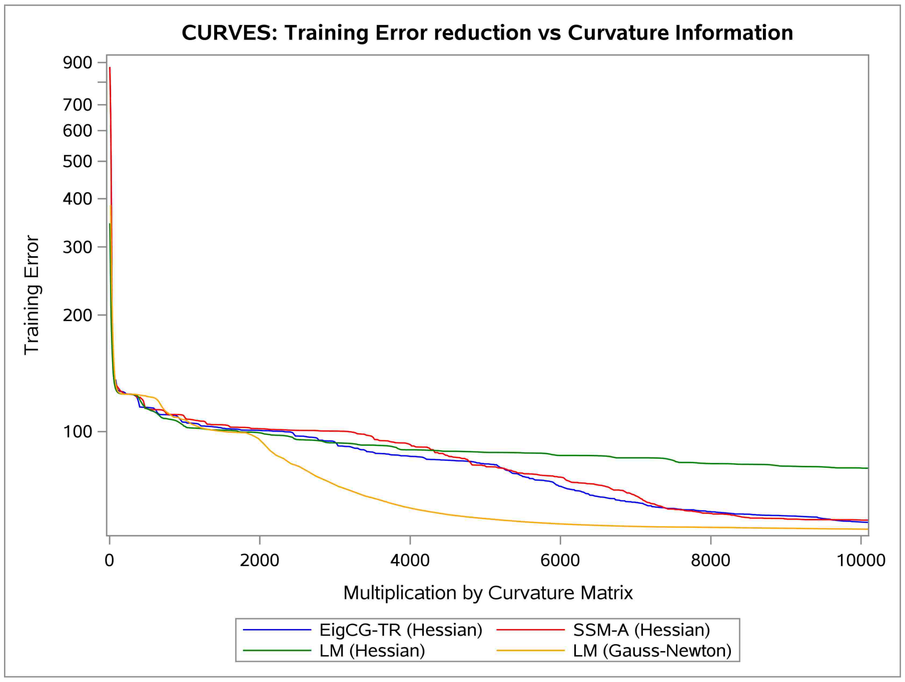


Figure 5.8: Training loss per Matrix-vector product of EigCG-TR, SSM-A, Levenberg-Marquardt (Hessian and Gauss-Newton)

The same behavior is occurring on the validation error as depicted in figure 5.9. The minimization trajectory suggests that for the autoencoder task on curves dataset, Gauss-Newton approximation can form a more suitable model for the optimization.

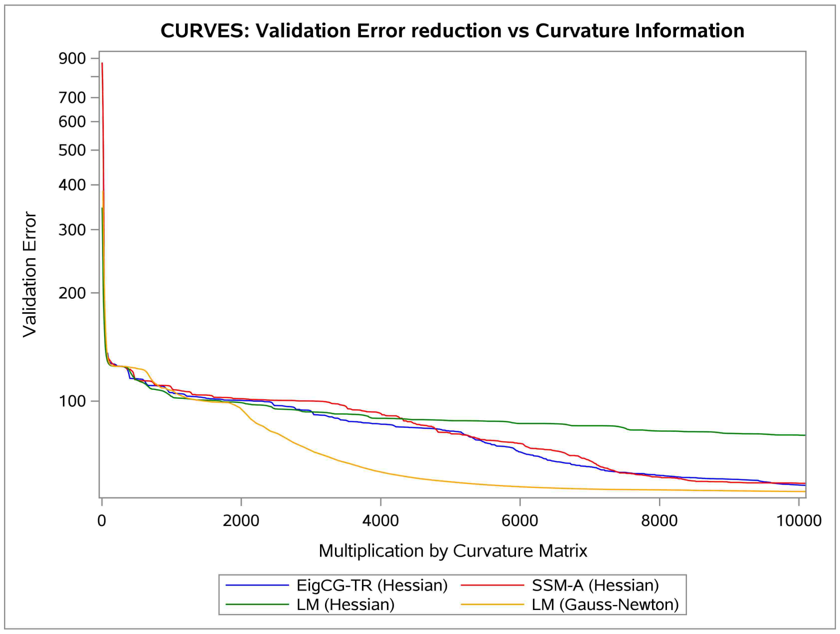


Figure 5.9: Validation error on each iteration of EigCG-TR, SSM-A, Levenberg-Marquardt (Hessian and Gauss-Newton)

Therefore, the curvature information for SSM based solvers was switched from Hessian to Gauss-Newton. Figure 5.10 shows that maintaining Gauss-Newton as the second-order estimate is beneficial for optimizing the training error. We can see that in this new setting, both SSM-A and EigCG-TR exploit the information much better than Levenber-Marquardt, with EigCG-TR slightly converging faster compared to SSM-A.

Same as previous runs, Figure 5.11 shows that validation error is also reduced same as training error and both SSM based methods are performing less number of matrix multiplications to converge to same error.

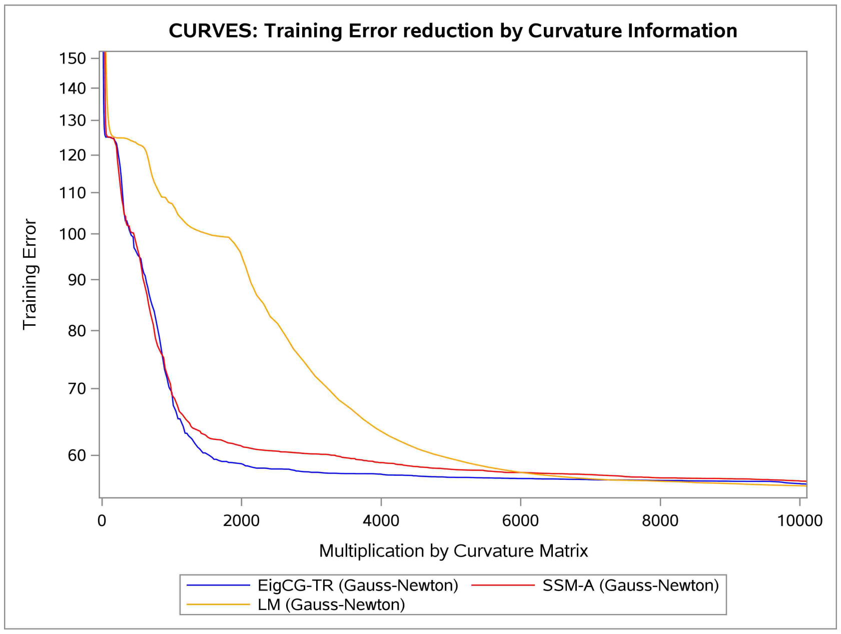


Figure 5.10: Training loss per Matrix-vector product of EigCG-TR, SSM-A, Levenberg-Marquardt using Gauss-Newton

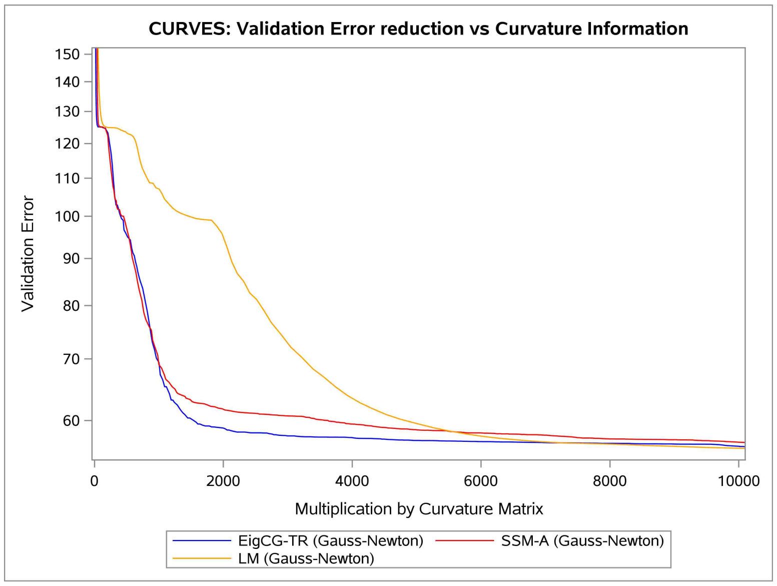


Figure 5.11: Validation error on each iteration of EigCG-TR, SSM-A, Levenberg-Marquardt using Gauss-Newton

5.6 Discussion

One of the computationally involving procedures of second-order methods for training deep neural networks is the multiplication with the curvature information We proposed an extended version of SSM-A which improves the convergence compared to the original algorithm by using the available information to maintain estimates of the eigen pair information. The algorithm provides a seamless solver to be utilized in both convex and nonconvex QP models; providing the user with the choice to use both information as suited to the problem. As numerical results show, there are cases for which the nonconvex Hessian is more suitable for a faster reduction of the training and validation loss, whereas on the other hand for some cases Gauss-Newton matrix provides models that facilitate easier convergence. The proposed EigCG-TR method supports both modules and moreover utilizing the eigen pair information it can reduce the number of vector multiplication by curvature matrix.

Appendices

Appendix A

Sinco2d Subroutines

A.1 Parallelization of First phase

This phase includes searching through the coordinates and finding the most suitable coordinate which would greatly improve the loss. Therefore each processor would search around its portion of coordinates in order to find the desired coordinate. Once the coordinate is found a the **coordinate_reduce** would be called to propagate the information globaly.

```
while (!stop && iter<iterthreshold) {
    iter++;
    funmax=f;
    alphamax=0;
    twoDupd=0;
    selprocmax=myrank_mpi;
    /*---------search phase-----------*/
    for (int i=1; i<=mp; i++) {
        for (int j=1; j<=nq; j++) {
        //only searching half of the matrix
        sat=IDSat[(j-1)*mp+(i-1)];
        sut=IDSut[(j-1)*mp+(i-1)];
        if (sut==sat) {
```

```
 alpha=0;
KW=2*K*W[(j-1)*mp+(i-1)];
 double ccp, ccpp;
 ccp=0;
 ccpp=Cpp.coeffRef((i-1), (j-1));
 if (ccpp<tol) {
         ccp=Cp.coeffRef((i-1), (j-1));
 }
 if ((2*Gp[(j-1)*mp+(i-1)]+KW>tol) && (ccpp<=tol)){
     upd=1;
     alpha=findposstep(K, W[(j-1)*mp+(i-1)],
     DgW[sat-1], DgW[sut-1]  , A[(j-1)*mp+(i-1)],
     lmbda*S[(j-1)*mp+(i-1)],lmbda*S[(j-1)*mp+(i-1)]
      , upd);
 }
 else if ((2*Gpp[(j-1)*mp+(i-1)]-KW>tol) && (ccp<=tol)){
     upd=-1;
     alpha=findposstep(K, W[(j-1)*mp+(i-1)],
     DgW[sat-1], DgW[sut-1], A[(j-1)*mp+(i-1)],
     lmbda*S[(j-1)*mp+(i-1)], lmbda*S[(j-1)*mp+(i-1)],
     upd);
 }
 else if ((2*Gp[(j-1)*mp+(i-1)]+KW<-tol) && (ccp>tol)){
     upd=1;
     alpha=findnegstep(K, (sat==sut), W[(j-1)*mp+(i-1)],
     DgW[sat-1], DgW[sut-1], A[(j-1)*mp+(i-1)],
      lmbda*S[(j-1)*mp+(i-1)], lmbda*S[(j-1)*mp+(i-1)]
      , ccp, ccpp, upd);
 }
 else if ((2*Gpp[(j-1)*mp+(i-1)]-KW<-tol) && (ccpp>tol)){
     upd=-1;
     alpha=findnegstep(K, (sat==sut), W[(j-1)*mp+(i-1)],
            DgW[sat-1], DgW[sut-1], A[(j-1)*mp+(i-1)],
            lmbda*S[(j-1)*mp+(i-1)], lmbda*S[(j-1)*mp+(i-1)],
             ccp, ccpp, upd);
        }
        if (fabs(alpha)>tol) {
                fchange=funvalue_upddiag(2*alpha,
                K, DgW[sat-1], A[(j-1)*mp+(i-1)],
                lmbda*S[(j-1)*mp+(i-1)], upd);
        }
        else {
                 fchange=0;
        }

        fnew=f+fchange;

        if (fnew>funmax) {
                 //select this coordinate
                 funmax=fnew;
                 imax=sat;
                 jmax=sut;
                 //alphamax=AllSteps(i,j).alpha;
                 alphamax=alpha;
                 updmax=upd;
                 wijmax=W[(j-1)*mp+(i-1)];
                 //updmax=AllSteps(i,j).upd;
        }
}

if (sut>sat) {
   alpha=0;
  KW=2*K*W[(j-1)*mp+(i-1)];
   double ccp, ccpp;
   ccp=0;
```

```
ccpp=Cpp.coeffRef((i-1), (j-1));
if (ccpp<tol) {
       ccp=Cp.coeffRef((i-1), (j-1));
}
if ((2*Gp[(j-1)*mp+(i-1)]+KW>tol) && (ccpp<=tol)) {
       upd=1;
       alpha=findposstep(K, W[(j-1)*mp+(i-1)], DgW[sat-1],
       DgW[sut-1], A[(j-1)*mp+(i-1)],
       lmbda*S[(j-1)*mp+(i-1)], lmbda*S[(j-1)*mp+(i-1)],
       upd);
}
else if ((2*Gpp[(j-1)*mp+(i-1)]-KW>tol) && (ccp<=tol)) {
    upd=-1;
    alpha=findposstep(K, W[(j-1)*mp+(i-1)],
    DgW[sat-1],DgW[sut-1], A[(j-1)*mp+(i-1)]
    lmbda*S[(j-1)*mp+(i-1)],lmbda*S[(j-1)*mp+(i-1)]
    , upd);
}
else if ((2*Gp[(j-1)*mp+(i-1)]+KW<-tol) && (ccp>tol)) {
    upd=1;
    alpha=findnegstep(K, (sat==sut), W[(j-1)*mp+(i-1)],
    DgW[sat-1], DgW[sut-1], A[(j-1)*mp+(i-1)], l
    ambda*S[(j-1)*mp+(i-1)], lmbda*S[(j-1)*mp+(i-1)],
    ccp, ccpp, upd);
}
else if ((2*Gpp[(j-1)*mp+(i-1)]-KW<-tol) && (ccpp>tol)) {
    upd=-1;
    alpha=findnegstep(K, (sat==sut), W[(j-1)*mp+(i-1)],
     DgW[sat-1], DgW[sut-1], A[(j-1)*mp+(i-1)],
     lmbda*S[(j-1)*mp+(i-1)], lmbda*S[(j-1)*mp+(i-1)],
     ccp, ccpp, upd);
}
if (fabs(alpha)>tol) {
        fchange=funvalue_upd(alpha, K, W[(j-1)*mp+(i-1)],
        DgW[sat-1], DgW[sut-1], A[(j-1)*mp+(i-1)],
        lmbda*S[(j-1)*mp+(i-1)], lmbda*S[(j-1)*mp+(i-1)],
         upd);
        }
        else {
            fchange=0;
        }
        fnew=f+fchange;
        if (fnew>funmax) {
        //select this coordinate
            funmax=fnew;
            imax=sat;
            jmax=sut;
            alphamax=alpha;
            updmax=upd;
            wijmax=W[(j-1)*mp+(i-1)];
          }
        }
    }
}
  //REDUCE to the max coordinate on all processors
    MPI_Barrier(MPI_COMM_WORLD);
    coordinate_reduce(nprocs_mpi, mpiroot, imax, jmax,
                funmax,alphamax, updmax, wijmax, selprocmax);
```

A.2 Finding 1d-stepsize

These routines as discussed in Scheinberg and Rish Scheinberg and Rish [2010] determine the stepsize along one dimensional upds; these procedures are called frequently during the search phase.

```
double findposstep(double K, double Wij, double Wii, double Wjj,
double Aij, double Sij, double Sji, int upd) {
/* This routine computes the optimal length of
 a step in the direction e_ie_j^T+e_je_i^T for
  the positive component of C */
double aux1, aux2, aux3, aux4, aux5;
double a, b, c, D, alpha, alpha1, alpha2;

if (upd==1) {
        aux1=2*(K*Wij-Aij)-Sji-Sij;
        aux2=(Wii*Wjj-Wij*Wij);
        aux3=2*Wij;
        aux4=-2*K*(Wii*Wjj+Wij*Wij);
        aux5=2*K*Wij*aux2;
}
else {
        aux1=2*(-K*Wij+Aij)-Sij-Sji;
        aux2=(-Wii*Wjj+Wij*Wij);
        aux3=2*Wij;
        aux4=2*K*(Wii*Wjj+Wij*Wij);
        aux5=-2*K*Wij*aux2;
}
a=aux2*aux1-aux5;
b=-aux3*aux1-aux4;
c=-upd*aux1;
if (fabs(a)>zerotol) {
        D=b*b-4*a*c;
        if (D<0) {
                printf("Negative␣discriminant,␣\n");
        }
        double sqrD=sqrt(D);
        double inv2a=1/(2*a);
        if (a>0) {
                alpha1=(-b-sqrD)*inv2a;
```

```
                        alpha2=(-b+sqrD)*inv2a;
                }
                else {
                        alpha1=(-b+sqrD)*inv2a;
                        alpha2=(-b-sqrD)*inv2a;
                }

                if (alpha1>=0) { alpha=alpha1; }
                else {
                        if (alpha2>=0) { alpha=alpha2; }
                        else {
                                printf("unbound direction!,\n");
                        }
                }
        }
        else if (-c/b>0)
        {
                alpha=-c/b;
        }
        else {
                printf("unbound direction!,\n");
        }
        return alpha;
}

double findnegstep(double K, bool diag, double Wij,
 double Wii, double Wjj, double Aij, double Sij, double Sji,
  double Cpij, double Cppij, int upd) {
        /* This routine computes the optimal length
         of a step in the direction e_ie_j^T+e_je_i^T
         for the negative component of C */
        double aux1, aux2, aux3, aux4, aux5;
        double a, b, c, D, maxstep, alpha, alpha1, alpha2;
```

```
if (upd==1) {
        aux1=2*(K*Wij-Aij)-Sji-Sij;
        aux2=(Wii*Wjj-Wij*Wij);
        aux3=2*Wij;
        aux4=-2*K*(Wii*Wjj+Wij*Wij);
        aux5=2*K*Wij*aux2;
        maxstep=Cpij;
}
else {
        aux1=2*(-K*Wij+Aij)-Sij-Sji;
        aux2=(-Wii*Wjj+Wij*Wij);
        aux3=2*Wij;
        aux4=2*K*(Wii*Wjj+Wij*Wij);
        aux5=-2*K*Wij*aux2;
        maxstep=Cppij;
}
if (diag) {
        maxstep=maxstep/2;
}
a=aux2*aux1-aux5;
b=-aux3*aux1-aux4;
c=-upd*aux1;
if (fabs(a)>zerotol) {
        D=b*b-4*a*c;
        if (D<0) {
                printf("negative discriminant,\n");
        }
   double sqrD=sqrt(D);
alpha1=fmin(((-b-sqrD)/(2*a)), ((-b+sqrD)/(2*a)));
alpha2=fmax(((-b-sqrD)/(2*a)), ((-b+sqrD)/(2*a)));
if (alpha2<0) {
        alpha=fmax(alpha2, -maxstep);
}
```

```
          else if (alpha1<0) {
               alpha=fmax(alpha1, -maxstep);
          }
          else {
               alpha=-maxstep;
          }
          }
          else if (-c/b<0) {
                    alpha=fmax(-c/b, -maxstep);
          }
          else {
                    alpha=-maxstep;
          }
          return alpha;
}
```

A.3 2d-sub

The core of our algorithmic contribution was the development of the 2d-sub routine. Basically as a coordinate (i, j) gains popularity, instead of regular SINCO upds, we compute a more powerful upd step which encompass the corresponding diagonal elements. Algorithm 2.3.2 describes a pseudo code of how this can be achieved. Here we present a commented demonstration of practical implementation of 2d-sub.

```
//If the coordinate is off-diagonal and popular
if ((twoDupd==1) && (imax !=jmax)) {
//broadcast the off-diagonal element
//        corresponding to chosen coordinate
   if (myrank_mpi==selprocmax)
        AIJ=A[(locj)*mp+(loci)];

   MPI_Bcast(&AIJ, 1, MPI_DOUBLE, selprocmax, MPI_COMM_WORLD);
```

```
char nu=' ', allchr='A';
//Obtain elements (i,i) and (j,j)
//from the distributed matrix
PDELGET(&allchr, &nu, &AJJ, A, &imax, &imax, descA);
PDELGET(&allchr, &nu, &AII, A, &jmax, &jmax, descA);
//compute the quadratic equation
//to find values of 2d upd
wjj=DgW[imax-1];
wii=DgW[jmax-1];
aa=updmax*(2*SIJ);
alphaa=(AII+SII)/(AJJ+SJJ);
beta=(AII+SII)/(2*AIJ+aa);
kappa=wijmax*wijmax-wii*wjj;
b1=((-2*wijmax*beta/alphaa)+wjj)/(kappa);
b2=(wii-2*beta*wijmax)/(kappa);
AA1=2*beta/alphaa;
AA2=2*beta;
r1=1+wii*b1+wjj*b2-kappa*b1*b2;
r2=-AA1*wii+2*wijmax-AA2*wjj+kappa*(b1*AA2+b2*AA1);
r3=kappa-kappa*AA1*AA2;
t1=K*2*wijmax/(2*AIJ+aa);
t2=2*K*kappa/(2*AIJ+aa);
//finding degenerate root
if (abs(r1-t1)>1e-7) {
        del=(r2-t2)*(r2-t2)-4*(r1-t1)*r3;
        theta1=(t2-r2+sqrt(del))/(2*(r3));
        theta2=(t2-r2-sqrt(del))/(2*(r3));
        x3=theta1;
        x1=b1-AA1*x3;
        x2=b2-AA2*x3;
        G=1+x1*wii+2*x3*wijmax+x2*wjj
        +kappa*x3*x3-x1*x2*kappa;
```

```
          //corresponding to remark 2 proof
          if (abs(G)<1e-8)
                  x3=theta2;

    }
    else {
          theta1=(t2-r2)/r3;
          x3=theta1;
    }
    x1=b1-AA1*x3;
    x2=b2-AA2*x3;
}
//Corresponding to Step 4 of algorithm 2d-sub
//Checking whether upd agrees with the sign
if ((sgn((ccp-ccpp+x3))*sgn(aa)>0) && (twoDupd==1)
     && (imax !=jmax)) {
     //here the step is accepted
     //finding the indexes corresponding to maxrow and maxcol
     int imaxrow=INDXG2P(&imax, &nb, &one, &zero, &nprow);
     int imaxcol=INDXG2P(&imax, &nb, &one, &zero, &npcol);
     //updating element (i,i)
     if ((myrow==imaxrow) && (mycol==imaxcol)) {
          double lloci,llocj;
          lloci=INDXG2L(&imax, &nb, &myrow, &zero, &nprow)-1;
          llocj=INDXG2L(&imax, &nb, &mycol, &zero, &npcol)-1;
          Cp.coeffRef(lloci, llocj) +=x2;
     }
     imaxrow=INDXG2P(&jmax, &nb, &one, &zero, &nprow);
     imaxcol=INDXG2P(&jmax, &nb, &one, &zero, &npcol);
     //updating element (j,j)
     if ((myrow==imaxrow) && (mycol==imaxcol)) {
          double lloci,llocj;
          lloci=INDXG2L(&jmax, &nb, &myrow, &zero, &nprow)-1;
```

```
        llocj=INDXG2L(&jmax, &nb, &mycol, &zero, &npcol)-1;
        Cp.coeffRef(lloci, llocj) +=x1;
    }
//updating the off diagonal and resetting the counter
if (myrank_mpi==selprocmax) {
        double XX=ccp-ccpp+x3;
        Cp.coeffRef(loci, locj)=XX<0 ? 0 : XX;
        Cpp.coeffRef(loci, locj)=XX<0 ? -XX : 0;
        Count.coeffRef(loci, locj)=0;
}
//forming formulas for updating the inverse
G=1+x1*wii+2*x3*wijmax+x2*wjj+kappa*x3*x3-x1*x2*kappa;
fchange=f;
 f +=K*log(G)-x1*AII-x2*AJJ-2*x3*AIJ
 -SII*(x1)-SJJ*(x2)-aa*(x3);
 double a, b, c, d, wij;
 double k1, a1, a2, a3, T;

 wij=wijmax;
 theta=x3;
 mu=x1;
 lmbdaU=x2;
 k1=1+theta*wij+mu*wii;
 a1=theta-((mu*theta*wii)/k1)-((mu*lmbdaU*wij)/k1);
 a2=lmbdaU-((theta*lmbdaU*wij)/k1)-((wii*theta*theta)/k1);
 a3=(mu*wij+theta*wjj)/k1;
  T=1+theta*wij+lmbdaU*wjj-(mu*theta/k1)*wij*wii
  -(mu*lmbdaU/k1)*wij*wij
 -(theta*theta/k1)*wjj*wii-(theta*lmbdaU/k1)*wjj*wij;

  a=(-a1/T);
  b=(a1*a3/T)-(mu/k1);
  c=(-a2/T);
```

```
d=(a2*a3/T)-(theta/k1);
double feab=a*a-b*c;
double n1, q;
//computing the coeeficients for
//the symmetric rank two upd
 if (feab>tol) {
    n1=(a-sqrt(feab))/b;
    q=a-b*n1*(0.5);
    pdgeadd_(&transN,&p,&one,&n1,U,&one,&one,
                    descU,&one,V,&one,&one,descV);
    coef=b/2;
    q -=coef*n1;
    pdgeadd_(&transN,&p,&one,&coef,V,&one,&one,
                  descV,&q,U,&one,&one,descU);
    coef=1;
    pdsyr2_(&uplo,&p,&coef,U,&one,&one,descU,&one,V,
         &one,&one,descV,&one,W,&one,&one,descW);
    if (nq_rhs !=0) {
           vdMul(mp,U,V,UU);
    }
    coef *=2;
    pdscal_(&p,&coef,UU,&one,&one,descUU,&one);
    PDGEMR2D(&p,&one,UU,&one,&one,descUU,Diags,
                    &one, &one, descDiags, &ictxt);
    if (mpiroot) {
           Cdgebs2d(ictxt,"All","␣",p,1,Diags,1);
    }
    else {
           Cdgebr2d(ictxt,"All","␣",p,1,Diags,1,0,0);
    }
    cblas_daxpy(p, 1, Diags, 1, DgW, 1);
    MPI_Barrier(MPI_COMM_WORLD);
}
```

```
    else {
        //rank two upd reduces to rank one upd
        double m=1;
        double al=-(a*a-b*c)/b;
        pdsyr_(&uplo,&p,&al,U,&one,&one,descUU,
                     &one,W,&one,&one,descW);
        coef=a/b;
        pdgeadd_(&transN,&p,&one,&coef,U,&one,&one,
                          descU,&one,V,&one,&one,descV);
        pdsyr_(&uplo,&p,&b,V,&one,&one,descV,&one,
                     W,&one,&one,descW);
        if (nq_rhs !=0) {
                vdSqr(mp,U,UU);
                vdSqr(mp,V,VV);
        }
        coef=b;
        pdgeadd_(&transN,&p,&one,&al,UU,&one,&one,
                          descUU,&b,VV,&one,&one,descVV);
        PDGEMR2D(&p,&one,VV,&one,&one,descVV,
                          Diags,&one,&one,descDiags,&ictxt);
        //updating the diagonal elements in seperate array
        if (mpiroot) {
                Cdgebs2d(ictxt,"All"," ",p,1,Diags,1);
        }
        else {
                Cdgebr2d(ictxt,"All"," ",p,1,Diags,1,0,0);
        }
        cblas_daxpy(p, 1, Diags, 1, DgW, 1);
        MPI_Barrier(MPI_COMM_WORLD);
    }
}
```

Appendix B

Parallel Graphical Lasso

B.1 Main GLasso driver loop

```
// This is a cyclic block coordinate descent method
//This way the whole matrix W is used in the
//LASSO sub-problem and the corresponding beta=0
        while((!stop) && (iter<Maxiter)){
        blkidx=(iter % p)+1;
           bool lasstop=false;
        int t=((blkidx-1)-(((blkidx-1)% nb)))/nb;
        bool ihaveidx=(myrank_mpi==(t % nprocs_mpi));
        iter++;
        int localidx=-1;
        //Get initial Solution for Beta
        double nnn=0;
        if(ihaveidx){
                //generating theta
                localidx=indxg2l_(&blkidx,&nb,&mycol
                            ,&zero,&npcol);
                nnn=1/(Q[(localidx-1)*p+blkidx-1]);
        }
```

```
MPI_Bcast(&nnn,1,MPI_DOUBLE,
        int(t% nprocs_mpi),MPI_COMM_WORLD);
nnn=0;
cblas_dcopy(nq,&Q[blkidx-1],p,&localBeta[0],one);
cblas_dscal (nq, nnn, &localBeta[0], one);
std::vector<double>oldbeta(nq,0);
cblas_dcopy(nq,&Q[blkidx-1],p,&oldbeta[0],one);
cblas_dscal (nq, nnn, &oldbeta[0], one);
vector<double>G(nq,0);
// int localidx=-1;
if(ihaveidx){
        localBeta[localidx-1]=0;
}
Uplo='L';
//This essentially gets the current
// gradient before calling the lasso
pdsymv_(&Uplo,&p,&one,&W[0],&one,&one,descW,
        &localBeta[0],&one,&one,descBeta,&one,&zero,
        &G[0],&one,&one,descG,&one);
//reducing the gradient from pxp to p-1xp-1
//to make suitable for subproblem
if(ihaveidx){
        G[localidx-1]=0;
}
//CALL LASSO Solver
```

B.2 Inner Lasso Solver

This procedure solve the inner lasso problem in a distributed setting and then propagates the results by synchronizing the information through processors.

```
//This is the local Lasso solver
//ihaveidx shows that whether processor
//contains the row which corresponds to
```

```
    for (int it=0; ((it<lassoit)&&(!lasstop)); it++){
    //Setting change in the gradient to zero
    dlaset_(&transN,&p,&one,&zero,&zero,&deltaG[0],&p);
    if(ihaveidx){
    for (int h=0; h<comm_it; h++){
            // local coordinate descent
            int idx=rand()/(0.0+RAND_MAX)*nq;
            int indx=idx+1;
            indx=indxl2g_(&indx,&nb,&mycol,&zero,&npcol);
            indx--;
            while(indx==blkidx-1){
             idx=rand()/(0.0+RAND_MAX)*nq;
             indx=idx+1;
             indx=indxl2g_(&indx,&nb,&mycol,&zero,&npcol);
             indx--;
    }
double prmA,prmB,prmC,deltaStar;
    prmA=(G[idx]+4*deltaG[indx])+S[idx*p+(blkidx-1)];
    prmB=0.5*sigma*W[idx*p+indx];
    prmC =localBeta[idx];
    deltaStar=softthreshold(prmA,prmB,prmC,lmbda);
    localBeta[idx]+=deltaStar;
    cblas_daxpy(p,deltaStar, &W[idx*p],one,&deltaG[0],one);
     }
    }
    else{
    for (unsigned int h=0; h<comm_it; h++){
    // local coordinate descent
    int idx=rand()/(0.0+RAND_MAX)*nq;
    int indx=idx+1;
    indx=indxl2g_(&indx,&nb,&mycol,&zero,&npcol);
    indx--;
    double prmA,prmB,prmC,deltaStar;
    prmA=( G[idx]+4*deltaG[indx])+S[idx*p+(blkidx -1)];
    prmB=0.5*sigma*W[idx*p+indx];
    prmC =localBeta[idx];
    deltaStar=softthreshold(prmA,prmB,prmC,lmbda);
    localBeta[idx]+=deltaStar;
    cblas_daxpy(p,deltaStar, &W[idx*p],one,&deltaG[0],one);

      }
    }
    char align='␣';
    MPI_Barrier(MPI_COMM_WORLD);
    // reduce all
    Cdgsum2d(ictxt,&Allchr,&align,one
            ,p,&deltaG[0],one,zero,zero);
    //G=G+deltag
    pdgeadd_(&transN,&one,&p,&one,&deltaG[0],&one,
    &one,descvec_root,&one,&G[0],&one,&one,descG);
    gradientnorm=cblas_dnrm2(p,&deltaG[0],one);
    MPI_Bcast(&gradientnorm,1,MPI_DOUBLE,
                  0,MPI_COMM_WORLD);
    if(gradientnorm<gradtol)
            {
                    lasstop=true;
            }
    }
```

Bibliography

M. Arioli, T. F. Chan, I.S. Duff, N.I.M. Gould, and J.K. Reid. Computing a search direction for large-scale linearly-constrained nonlinear optimization calculations. Technical report ral, Rutherford Appleton Laboratory, Chilton, England, 1993.

Onureena Banerjee, Laurent El Ghaoui, and Alexandre d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *The Journal of Machine Learning Research*, 9:485–516, 2008.

I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and unconstrained testing environment. Report 93/10, Département de Mathématique, Facultés Universitaires de Namur, 1993.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10. ACM, 2016.

Jaeyoung Choi, James Demmel, Inderjiit Dhillon, Jack Dongarra, Susan Ostrouchov, Antoine Petitet, Ken Stanley, David Walker, and R Clinton Whaley. Scalapack: A portable linear algebra library for distributed memory computersdesign issues and performance. In *Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science*, pages 95–106. Springer, 1996.

Ed S Coakley and Vladimir Rokhlin. A fast divide-and-conquer algorithm for computing the spectra of real symmetric tridiagonal matrices. *Applied and Computational Harmonic Analysis*, 34(3):379–414, 2013.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pages 2933–2941, 2014.

Ron S Dembo and Trond Steihaug. Truncated-newton algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26(2):190–212, 1983.

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2, Ser. A):201–213, 2002. ISSN 0025-5610.

Evgenii Georgievich D'yakonov. Iteration methods in eigenvalue problems. *Mathematical Notes*, 34(6):945–953, 1983.

Jennifer B Erway and Philip E Gill. A subspace minimization method for the trust-region step. *SIAM Journal on Optimization*, 20(3):1439–1461, 2009.

Jennifer B Erway, Philip E Gill, and Joshua D Griffin. Iterative methods for finding a trust-region step. *SIAM Journal on Optimization*, 20(2):1110–1131, 2009.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Trans. Math. Softw.*, 29(4):353–372, Dec 2003. URL `http://doi.acm.org/10.1145/962437.962438`.

Nicholas IM Gould, Stefano Lucidi, Massimo Roma, and Philippe L Toint. Solving the trust-region subproblem using the lanczos method. *SIAM Journal on Optimization*, 9(2):504–525, 1999.

Melanie Gratton. *Algorithms for Trust-Region Subproblems with Linear Inequality Constraints*. PhD thesis, University of North Carolina at Chapel Hill, 2012.

Gaël Guennebaud, Benoît Jacob, et al. Eigen v3, 2010.

William Hager and Soonchul Park. Global convergence of ssm for minimizing a quadratic over a sphere. *Mathematics of Computation*, 74(251):1413–1423, 2005.

William W Hager. Minimizing a quadratic over a sphere. *SIAM Journal on Optimization*, 12(1):188–208, 2001.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29 (6):82–97, 2012.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Cho-Jui Hsieh, Inderjit S Dhillon, Pradeep K Ravikumar, and Mátyás A Sustik. Sparse inverse covariance matrix estimation using quadratic approximation. In *Advances in Neural Information Processing Systems*, pages 2330–2338, 2011.

Cho-Jui Hsieh, Arindam Banerjee, Inderjit S Dhillon, and Pradeep K Ravikumar. A divide-and-conquer method for sparse inverse covariance estimation. In *Advances in Neural Information Processing Systems*, pages 2330–2338, 2012.

Cho-Jui Hsieh, Mátyás A Sustik, Inderjit S Dhillon, Pradeep K Ravikumar, and Russell Poldrack. Big & quic: Sparse inverse covariance estimation for a million variables. In *Advances in Neural Information Processing Systems*, pages 3165–3173, 2013.

Prabhanjan Kambadur and Aurelie Lozano. A parallel, block greedy method for sparse inverse covariance estimation for ultra-high dimensions. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, pages 351–359, 2013.

Yann Le Cun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lu Li and Kim-Chuan Toh. An inexact interior point method for l 1-regularized sparse covariance selection. *Mathematical Programming Computation*, 2(3-4):291–315, 2010.

James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742, 2010.

James Martens and Ilya Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural Networks: Tricks of the Trade*, pages 479–535. Springer, 2012.

Eiji Mizutani and Stuart E Dreyfus. Second-order stagewise backpropagation for hessian-matrix analyses and investigation of negative curvature. *Neural Networks*, 21(2):193–203, 2008.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Jorge J Moré and Danny C Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4(3):553–572, 1983.

Andrew Ng. Sparse autoencoder. *CS294A Lecture notes 72.2011*, 2011.

Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 265–272, 2011.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

CC Paige, MA Saunders, Sou-Cheng Choi, Dominique Orban, Umberto Emanuele Villa, and Danielle Maddix. Minres: Sparse symmetric equations, 2014.

Christopher C Paige. Computational variants of the lanczos method for the eigenproblem. *IMA Journal of Applied Mathematics*, 10(3):373–381, 1972.

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

Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, pages 1–52, 2012.

Peter Richtárik and Martin Takáč. Distributed coordinate descent method for learning with big data. *arXiv preprint arXiv:1310.2059*, 2013.

Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1-2):433–484, 2016.

Irina Rish, Guillermo A Cecchi, Marwan N Baliki, and A Vania Apkarian. Sparse regression models of pain perception. In *Brain Informatics*, pages 212–223. Springer, 2010.

Yousef Saad. *Numerical Methods for Large Eigenvalue Problems: Revised Edition*, volume 66. Siam, 2011.

Yousef Saad, Manshung Yeung, Jocelyne Erhel, and Frédéric Guyomarc'h. A deflated version of the conjugate gradient algorithm. *SIAM Journal on Scientific Computing*, 21(5):1909–1926, 2000.

Katya Scheinberg and Irina Rish. Learning sparse gaussian markov networks using a greedy coordinate ascent approach. In *Machine learning and knowledge discovery in databases*, pages 196–212. Springer, 2010.

Katya Scheinberg, Shiqian Ma, and Donald Goldfarb. Sparse inverse covariance selection via alternating linearization methods. In *Advances in neural information processing systems*, pages 2101–2109, 2010.

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

Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for machine learning*. Mit Press, 2012.

Andreas Stathopoulos and Konstantinos Orginos. Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics. *SIAM Journal on Scientific Computing*, 32(1): 439–462, 2010.

Trond Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1139–1147, 2013.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

Philippe L Toint. Towards an efficient sparsity exploiting newton method for minimization. *Sparse matrices and their uses*, pages 57–88, 1981.

Oriol Vinyals and Daniel Povey. Krylov subspace descent for deep learning. In *AISTATS*, pages 1261–1268, 2012.

Simon Wiesler, Jinyu Li, and Jian Xue. Investigations on hessian-free optimization for cross-entropy training of deep neural networks. In *Interspeech*, January 2013.

Kesheng Wu and Horst Simon. Thick-restart lanczos method for large symmetric eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(2): 602–616, 2000.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Yaxiang Yuan. On the truncated conjugate gradient method. *Mathematical Programming*, 87(3):561–573, 2000.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

W Zhou, IG Akrotirianakis, S Yektamaram, and JD Griffin. A matrix-free line-search algorithm for nonconvex optimization. *Optimization Methods and Software*, pages 1–24, 2017.

Biography

Seyedalireza Yektamaram is a Ph.D. candidate in the Industrial and Systems Engineering (ISE) Department of Lehigh University. His interests include large scale nonlinear optimization methods for machine learning applications. He has recieved his Industrial Engneering B.S. degree from Azad Tehran-North university of Iran, and his Industrial Engineering M.S. degree from Sharif University of Technology of Iran. Seyedalireza will join SAS Institute as an Operations Research Specialist.