Faster Inductive Training for Convex Two-Layer Models

by

Chandrasekhara Ganesh Jagadeesan
B.Tech., SRM University, 2016

Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2019

Chicago, Illinois

Defense Committee:
Xinhua Zhang, Chair and Advisor
Piotr Gmytrasiewicz
Xiaorui Sun

To my mom and dad.

ACKNOWLEDGMENTS

Foremost, I would like to express my profound gratitude to my thesis advisor Professor Xinhua Zhang for his constant guidance, support, and time. As my teacher and mentor, he has taught me more than I could mention here. I am also indebted to Vignesh Ganapathiraman for all the hours spent discussing implementations and fixing bugs.

I am grateful to all of my course instructors and all the professors with whom I have had the pleasure of working during my Master's degree. Special thanks to Professor Piotr Gmytrasiewicz and Professor Xiaorui Sun for agreeing to serve on the defense committee.

I would also like to thank everyone at the Electronic Visualization Laboratory and specifically my coworkers for their comradery. Research would have been infinitely harder without their constant motivation.

I extend my perpetual gratitude to my Uncle, Aunt, and my cousin Sharad, for their love and unconditional reliability. I also thank my flatmates Aashu and Pratik, who have always treated me like family. Preserving my sanity would have been a challenge without them.

My parents are my ultimate role-models who have always showered me with love and guidance through all my pursuits. Being a world apart has only served to bring us closer, and my love and gratitude for them knows no bounds.

GJ

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CP	Completely Positive matrices
KKT	Karush-Kuhn-Tucker
GCG	Generalized Conditional Gradient
PSD	Positive Semi-definite
ReLU	Rectified Linear Unit
SDP	Semi-definite Programming
SVD	Singular Value Decomposition

SUMMARY

Deep learning is currently one of the most effective approaches in machine learning with applications in image processing, computer vision, and natural language processing. The key technique underpinning its success is the automated learning of latent representation in data using neural networks that employ parametric hidden variables. However, these parameters are typically subject to a non-convex optimization, making the global optimum hard to find.

Inductive learning frameworks that guarantee global optimality have been recently developed for two-layer conditional models with a learning strategy based on parametric transfer functions. However, they require optimization over large kernel matrices, hence are slow in training and cannot be scaled to big datasets. In this thesis, we propose a novel optimization strategy that iteratively and greedily expands the subspace of kernels, interlaced with network parameter optimization in the low-rank subspace. The resulting approach significantly speeds up training by 10 to 100 times, while maintaining optimality and accuracy. This allows convex neural networks to be scaled to 10,000 examples for the first time.

CHAPTER 1

INTRODUCTION

The success of neural networks in various application domains such as computer vision, natural language processing, and game playing is often attributed to the automated learning of latent representations. This automated synthesis of abstract and semantic features from data often results in strong estimations of the predictive relations between the observation regardless of the possible complexity in the correlation.

Despite the vast successes in applications, it remains that these latent models are hard to train due to the non-linear coupling of the model parameters. This in general leads to unconstrained convex optimization strategies being applied to a non-convex optimization problem, thus making it hard to guarantee a globally optimal solution.

This issue of global trainability has been researched extensively and substantial progress has been made towards reformulating the latent models in achieving tractable global solutions. An inductive learning framework that facilitated convex relaxations based on completely positive cones was proposed by Ganapathiraman et al. [1] that provided an efficient learning strategy for two-layer conditional models and guaranteed global optimality.

Although this framework significantly scaled up the size of solvable problems by convex neural networks, the optimization over large kernel matrices still bottle-necked the computation time, preventing the model from scaling to real world datasets. The goal of this thesis, therefore, is to develop a better optimization strategy that achieves greater scaling while maintaining the guarantees of tractable global optimality.

1.1 Motivation

Let $\Phi = (\phi_1, \phi_2, ..)$ be the hidden node values of the training examples where ϕ_i is for training example x_i. The convex relaxation scheme in [1] is based on the Fenchel dualization of the activation function, which acts on the corresponding dual variables $\Lambda = (\lambda_1, \lambda_2, ..)$. The key step of SDP relaxation is to re-write the dual variable Λ in terms of Φ, as

$$\Lambda = \Phi A. \tag{1.1}$$

This allows Φ to be multiplied with Φ' (see details in Chapter 3), so that SDP relaxation can be carried out by replacing $\Phi'\Phi$ with a positive semi-definite (PSD) matrix T:

$$T = \Phi'\Phi \tag{1.2}$$

while shifting the key computational cost to A, whose size is $t \times t$, *i.e.*,, the square of number of training examples. This hampers the scalability of the model.

In our work, we take the key insight that within the SDP solver in [1], the solution T gradually expands its rank/subspace by greedily adding new bases. The efficiency of such greedy algorithms, indeed for general greedy optimization solvers such as GCG [2], lies in the so-called local optimization (step 6 of Algorithm 1), which in our case first recovers a Φ from the current low-rank solution of T, and then optimizes the loss over Φ as a *non-convex* problem. As GCG [2] showed, this local optimization substantially accelerates the overall convex optimization on T.

This insight motivates us to reconsider the optimization strategy: now that this step of local optimization has forgone the convexity in solving Φ, could we dispense with the optimization over A and directly optimize over the original Λ in Equation 1.1?

This leads to a significant reduction in the size of the optimization variable: from $t \times t$ (for A) to $t \times h$ (for Φ). Furthermore, it turns out that the objective can be reformulated accordingly in terms of Λ, thereby enjoying the performance boost offered by the reduced size of the problem.

1.2 Key Contribution

The main contributions of this work are listed below.

Optimization over low-rank matrices: We propose a strategy for solving the objective by decoupling the local optimization step from nested high-rank variable optimizations, which were the cause of significant bottle-necks in [1]. This limits the context of local optimizations to low-rank variables.

Closed-form solutions for large matrix: The reformulation of the objective function to optimize Λ instead of A (as in [1]) allows for closed form solutions for A. This, combined with the previous point, offers a significant speed up to inductive training for the conditional two-layer models.

Scaling to 10,000 examples: The new optimization strategy proposed in this work allows for the model to scale up to 10,000 examples for the first time. This is discussed in Chapter 6.

1.3 Outline of the Thesis

We begin in Chapter 2 by setting up an introduction to machine learning, the classification task, neural networks and then convex optimizations. These form the cumulative background required to read this thesis. In Chapter 2, we also describe the various datasets that were used in training and evaluation of the models described in this work.

Chapter 3 sets up conditional two-layer models which is the bedrock for this work, and Chapter 4 describes the optimization strategy currently employed by these models. While these chapters directly follow the works of Ganapathiraman et al. [1], we make moderate modifications to focus on the developments that are relevant to our proposed optimization algorithm.

The new optimization strategy and the updated algorithms are presented in Chapter 5 with a discussion of changes to the objective function and the computational complexity of the new optimization. Chapter 6 proceeds to describe the results of evaluation by comparing our optimization with the existing optimization. Finally, we conclude in Chapter 7 with a summary and the possible directions for this work.

CHAPTER 2

BACKGROUND

This chapter provides an introductory explanation to the various concepts that provide a unifying base for the work in this thesis. Section 2.1 defines machine learning and explains its types. Section 2.2 assumes prior knowledge of a perceptron and describes feed forward neural networks. Finally, section 2.3 introduces convex optimization methods. We follow the definitions and notations of Goodfellow et al. [3] for sections 2.1 and 2.2, and the works of Boyd and Vandenberghe [4] for section 2.3. Figure 1 has been sourced from the works of Nielsen [5].

2.1 Machine Learning

Mitchell [6] defined machine learning as, "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Tasks: Machine learning enables us to tackle tasks that are hard to solve with programs with rules and fixed conditions that are written and designed by humans. These tasks are typically described in terms of how a system processes a collection of quantitatively measured features. We typically represent this collection of features or **examples** as a vector $\boldsymbol{x} \in \mathbb{R}^n$ where each entry x_i of the vector is a feature. This collection of examples is called a **training set**.

Over time, machine learning has been successfully used towards various tasks such as regression, classification, anomaly detection, machine translation. In this thesis, we focus our attention on the classification task.

Performance: A performance measure helps us evaluate the abilities of a machine learning algorithm. Typically, this performance measure is specific to the task being performed by the system. For the classification task, we often measure the **accuracy** of the model. We evaluate the quality of the algorithm on previously unseen data or a **test set**.

Experience: The experience is defined based on the type of data the system is allowed to see during the learning process. This broadly classifies machine learning algorithms into three categories -

- **Inductive or supervised learning**: When the training data includes the corresponding output for every example.
- **Transductive or semi-supervised learning**: When the training data does not assume the presence of outputs for every example.
- **Deductive or unsupervised learning**: When the training data does not include the corresponding output for every example.

In this thesis, our classification task follows an inductive learning framework.

The goal of a machine learning algorithm is to enable a system to infer or *learn* the relation between observations based on given data without the need of explicitly providing said relation. This learning typically takes place by the process of *training* where the algorithm produces outputs and evaluates its performance. The metric by which the algorithm self-evaluates its performance during training is called a loss function or the objective function.

2.1.1 The Classification Task

The automated categorization of a new observation to a given set of k categories or labels based on previous data and their categorical labels is called classification. This task is typically performed by estimating a mapping function $f := \mathbb{R}^n \rightarrow \{1, .., k\}$ such that $y = f(\boldsymbol{x})$ assigns the numerical code of the correct category for an input described by the vector $\boldsymbol{x}$. An example of a classification task is object classification, where the input is the image of some object and the categories are the names of the objects in the image.

2.2 Feed-forward Neural Networks

Feed-forward neural networks are a collection of perceptrons or artificial neurons that are sequentially layered. Figure 1 describes a typical feed-forward neural network as a computational graph. The idea is that each layer characterizes a latent representation of the input data $\boldsymbol{x} \in \mathbb{R}^n$ using a non-linear function $f : \mathbb{R}^h \rightarrow \mathbb{R}^h$ which converts the linear transformation $W\boldsymbol{x}$ into $\phi = f(W\boldsymbol{x})$. Here $W \in \mathbb{R}^{h \times n}$ is the hidden layer weights. Thus, the output from a feed forward neural network of *2* layers would be

$$\hat{\boldsymbol{y}} = f_2(W_2 f_1(W_1 \boldsymbol{x})) \tag{2.1}$$

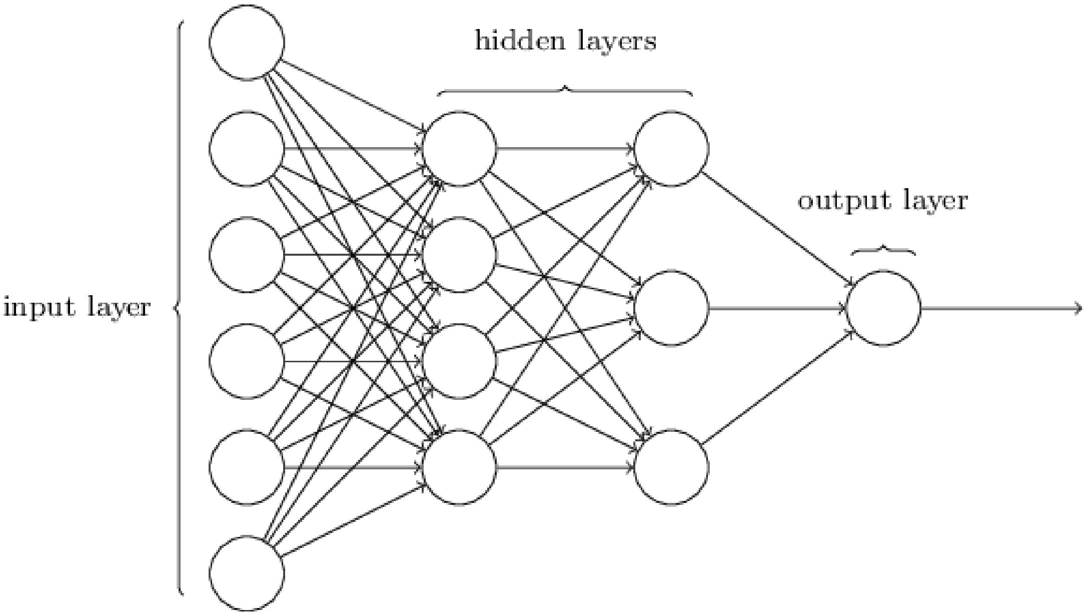


Figure 1: The computational graph of a feed-forward neural network

where f_i and W_i denote the transfer function and weights respectively for each hidden layer. There are several popular transfer functions that are applied to hidden layers. A common setting is to apply ReLU $f(x) = x^+$ to all the hidden layers and follow that with the softmax function $\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_j}}$ to the output layer to produce $\hat{\boldsymbol{y}}$. In the setting of the typical multi-class classification task with C classes, the output $\hat{\boldsymbol{y}}$ is measured against the given label $\boldsymbol{y} \in \mathbb{R}^m$

(which encodes a class $c \in C$ with the canonical vector $\boldsymbol{e}_c$) using an objective function or a loss function $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})$. The most commonly used loss function is the logistic loss

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\hat{\boldsymbol{y}}'\boldsymbol{y} + \log \sum_{c} \exp(\hat{y}_c) \tag{2.2}$$

Training a neural network model is equivalent to *optimizing* the loss function or minimizing the objective value.

2.3 Convex Optimization

This chapter serves as an introduction to convex optimization methods. We first go over the definition and the standard form of a convex problem, followed by some important functions that make convex optimizations applicable to problems that are non-convex. We then describe some common solution strategies by providing an overview of the oracles, but for this work, we treat the solvers as black-boxes.

2.3.1 Definition and Formulation

A constrained minimization problem can be written as

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq 0, \quad \forall i = 1, .., m \\ & h_i(x) = 0, \quad \forall i = 1, .., p, \end{aligned} \tag{2.3}$$

where $f_0(x) : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*, $f_i(x) : \mathbb{R}^n \to \mathbb{R}$ are the *inequality constraints*, and $h_i(x) : \mathbb{R}^n \to \mathbb{R}$ are the *equality constraints*. We call a problem a convex problem if $f_i, \forall i = 0, .., m$, are convex and $h_i, \forall i = 1, .., p$, are affine. Convex problems are a class of optimization problems in which the objective function is a convex function and the feasible set is a convex set.

2.3.2 Convex Conjugate

The *convex conjugate* $f^* : \mathbb{R}^n \to \mathbb{R}$ of a function $f \colon \mathbb{R}^n \to \mathbb{R}$ is defined as the supremum

$$f^*(y) = \sup_{x \in \mathbf{dom}\, f} \big(y^\mathsf{T} x - f(x) \big), \tag{2.4}$$

This is known as the *Legendre-Fenchel transform* of f. The Fenchel *bi-conjugate* f^{**} yields a *convex envelope*, the largest closed convex underapproximation of f [4]. The *epigraph* of a convex function f can be defined as

$$\mathbf{epi}(f) = \Big\{ (x, d) \in \mathbb{R}^{n+1} \mid f(x) \le d \Big\} \tag{2.5}$$

For all convex functions with closed *epigraph* and non-empty domains in finite dimensions, the bi-conjugate $f^{**} = f$.

2.3.3 Lagrange Duality

An important function in constrained optimization is the *Lagrangian*

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \tag{2.6}$$

where $(\lambda, \nu) \in \mathbb{R}^m \times \mathbb{R}^p$ are called the Lagrange multipliers or the *dual variables.* The dual function is defined as

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu) \tag{2.7}$$

The (Lagrange) *dual problem* is then given as

$$\begin{aligned} &\text{maximize} && g(\lambda, \nu) \\ &\text{subject to} && \lambda_i \geq 0, \quad \forall i = 1, .., m \end{aligned} \tag{2.8}$$

For a convex problem, necessary and sufficient conditions for primal and dual optimality for differentiable objective and constraint functions are given by Karush-Kuhn-Tucker (KKT) conditions [7]

$$\begin{cases} f_i(x^*) \le 0, \quad \forall i = 1, .., m \\ h_i(x^*) = 0, \quad \forall i = 1, .., p \\ \lambda_i^* \ge 0, \quad \forall i = 1, .., m \\ \lambda_i^* f_i(x^*) = 0, \quad \forall i = 1, .., m \\ \nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(x^*) = 0 \end{cases} \tag{2.9}$$

2.3.4 Solving Convex Optimizations

Typically, the harder task is the formulation of problems in the standard convex optimization form:

$$\begin{aligned} &\text{maximize} && f(x) \\ &\text{subject to} && x \in \mathcal{Q} \end{aligned} \tag{2.10}$$

where $f(x) = f_0(x)$ and

$$\mathcal{Q} = \{x \,|\, f_i(x) \le 0, \forall i = 1, .., m; h_i(x) \le 0, \forall i = 1, .., p\} \tag{2.11}$$

The methods for solving the problems as formed in Equation 2.10 are characterized by the type of structural information that can be evaluated.

Zero-order oracles evaluate only $f(x)$. Exhaustive search in combinatorial optimization employ these oracles.

First-order oracles evaluate $f(x)$ and $\nabla f(x)$. The classic steepest descent, conjugate gradient, quasi-Newton techniques employ a first-order oracle.

Second-order oracle evaluate $f(x)$, $\nabla f(x)$, and $\nabla^2 f(x)$. Newtons method and interior-point methods employ second-order oracles.

In application, first-order oracles are the most commonly used solvers even though second-order oracles arrive at the optima faster. This is attributed to the fact that computation of a Hessian does not scale well to large number of examples because of the computational complexity.

2.4 Datasets

This section details the different datasets used for training and evaluation in this work.

2.4.1 MNIST

The MNIST [8] dataset is an extremely popular dataset consisting of hand-written digits from 0-9 in different variations. There are totally (784×60000) training and (784×10000) testing image samples.

2.4.2 Letter

The Letter [9] dataset is made of (16×20000) image samples of vowel letters A-E and non-vowel letters B-F made available by UCI [10].

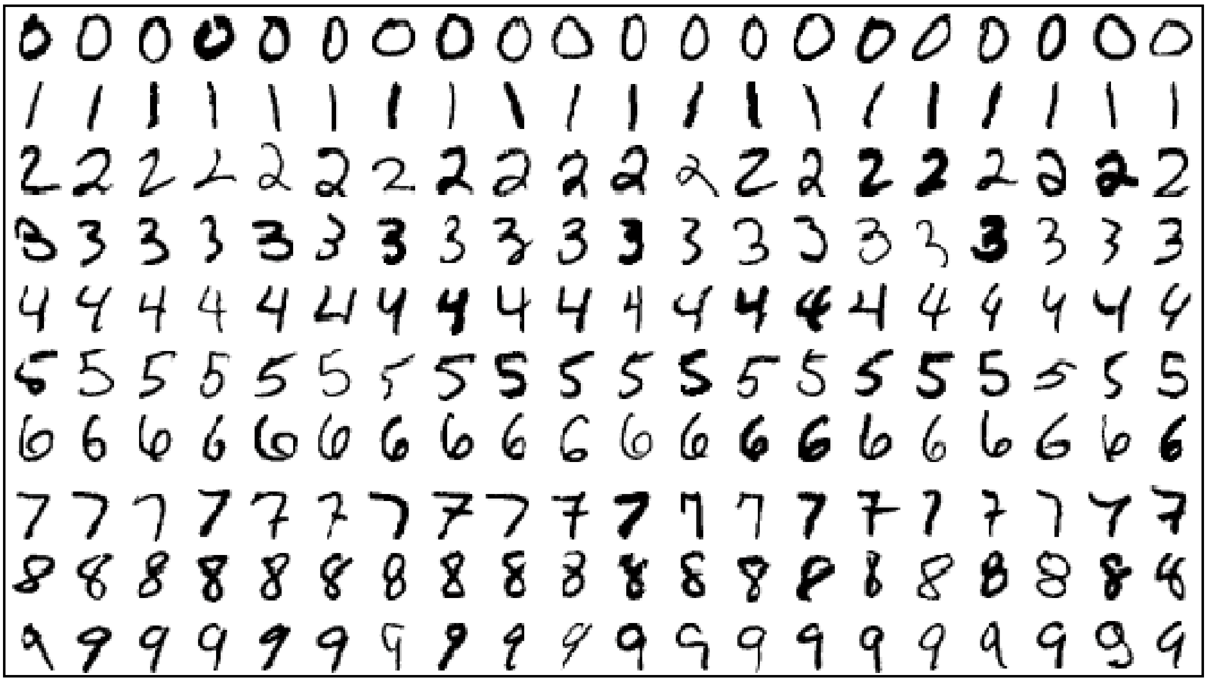


Figure 2: Samples from the MNIST dataset

2.4.3 CIFAR-SM

The CIFAR-SM [11] dataset is a subsampled dataset from the CIFAR-10 [12] dataset. It contains (256×1526) image samples of bicycles, motorcycles, lawn-mowers, and tanks with the red channel features preprocessed by averaging pixels.

2.4.4 USPS

USPS [13] is another popular hand-written text dataset containing (256×9298) images that have been sampled from zip-codes and state names in grey-scale.

2.4.5 COIL

COIL was collected by the Center for Research on Intelligent Systems at the Department of Computer Science, Columbia University. The database contains color images of various objects. This dataset was used in a real-time object recognition system whereby a system sensor could identify the object and display its angular pose. We use a sampled version of this database for binary classification, containing (241×1500) sample images.

2.4.6 G241N

G241N [14] by Chapelle is a commonly used synthesized benchmark dataset for various convex optimization binary classification tasks.. It contains (241×1500) randomly sampled data points.

CHAPTER 3

CONDITIONAL TWO-LAYER MODELING

Chapter 3 as well as Chapter 4 describe the works of Ganapathiraman et al [1] towards convex two-layer modeling and the optimizing these models. This work forms the underlying base for the algorithms derived in this thesis. As previously mentioned, the contents of Chapter 3 and Chapter 4 follow [1] directly with moderate modifications made to focus on the developments that are relevant to our proposed optimization strategy (detailed in Chapter 5).

3.1 Choosing a Loss Function

As explained in 2.2, a major source of non-convexity in neural networks is the non-linear mapping or the transfer function $\phi = f(W\boldsymbol{x})$. To cope with this, the approach is to construct a loss that would satisfy three conditions [1]:

- *Unique Recovery*: $\arg\min_{\phi} L(\phi, \mathbf{z}) = f(\mathbf{z})$ for all $\mathbf{z}$ with the arg min attained uniquely.
- *Joint Recovery*: L is jointly convex over ϕ and $\mathbf{z}$. This is required if a convex deep model is jointly built by directly using L to connect the input and output of adjacent layers.
- *Grounding*: $\min_{\phi} L(\phi, \mathbf{z}) = 0$ for all $\mathbf{z}$, so that there is no bias towards any value of $\mathbf{z}$.

The authors of [1] theorize:

Theorem 1 *There exists a loss L that satisfies all three conditions if, and only if, $\boldsymbol{f}$ is affine.*

They note that the matching loss [15] not only meets the first and third conditions, but also satisfies a weakened version of convexity by assuming that the transfer function is the gradient of a *strictly convex* function $F: \mathbf{f} = \nabla F$, with $F: \mathbb{R}^{\mathrm{h}} \to \mathbb{R}$.

Given that $\mathbf{f}$ is elementwise, constituent f is continuous and strictly increasing. This means that the inverse of $\mathbf{f}$ exists, $\mathbf{f}^{-1} = \nabla F^{*}$, where F^{*} is the *Fenchel conjugate* of F. When the transfer function is not strictly increasing in some direction like ReLU, which is non-increasing on the negative half line, approximations can be used such as the leaky-ReLU $f_{\mathrm{r}}(\mathbf{z}) = \max(\epsilon \mathbf{z}, \mathbf{z})$ for infinitesimally small $\epsilon > 0$.

In the case that $\mathbf{f}$ is not elementwise, this assumption of F implies:

1. $\mathbf{f}$ is strictly increasing in the vector sense: $(\mathbf{x} - \mathbf{y})'(\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})) > 0$.
2. The Jacobian of $\mathbf{f}$ is symmetric (as the Hessian of F): $J\mathbf{f} = (J\mathbf{f})'$, provided $\mathbf{f}$ is differentiable.

Under these assumptions, the authors finally adopt the following loss function based on Bregman divergence:

$$L(\phi, \mathbf{z}) = D_{F^{*}}(\phi, \mathbf{f}(\mathbf{z})) = F^{*}(\phi) + F(\mathbf{z}) - \phi'\mathbf{z}, \tag{3.1}$$

where $D_{F^{*}}$ is the Bregman divergence induced by F^{*}. Here, L satisfies the conditions of recovery and grounding, and is *not* jointly convex because of the bilinear term $\phi'\mathbf{z}$, while both F and F^{*} are convex. This decoupling of the convex and non-convex parts from the transfer function enables the convex reformulation.

3.2 Convex Modeling

Supposing there are t training pairs $\left\{(\mathbf{x}_j, \mathbf{y}_j)\right\}_{j=1}^{\mathrm{t}}$, stacked in two matrices $X = (\mathbf{x}_1, \dots, \mathbf{x}_t) \in \mathbb{R}^{n\times t}$ and $Y = (\mathbf{y}_1, \dots, \mathbf{y}_t) \in \mathbb{R}^{m\times t}$, with the corresponding set of latent layer outputs stacked into $\Phi = (\phi_1, \dots, \phi_t) \in \mathbb{R}^{h\times t}$, the regularized risk minimization objective can be written as:

$$\begin{aligned}&\min_{W,\Phi,U,\mathbf{b}} \sum_{j=1}^{t} D_{F^*}(\phi_j, \mathbf{f}(W\mathbf{x}_{\mathrm{j}})) + \ell(U'\phi_j + \mathbf{b}, \mathbf{y}_j) + \frac{\|W\|^2 + \|U\|^2}{2} \\ &= \min_{W,\Phi,U,\mathbf{b}} \sum_{j=1}^{t} \left\{F^*(\phi_j) - \phi_j' W\mathbf{x}_j + F(W\mathbf{x}_j) + \ell(U'\phi_j + \mathbf{b})\right\} + \frac{1}{2}\|W\|^2 + \frac{1}{2}\|U\|^2,\end{aligned} \tag{3.2}$$

where $\ell(U'\phi_j + \mathbf{b}) := \ell(U'\phi + \mathbf{b}, \mathbf{y}_j)$. The regularizations are introduced via Frobenius norms, and the assumed $\mathsf{dom}\,\ell_j$ is the entire space. For convenience, vector-input functions on matrices are written representing the sum of the function values applied to each column, *e.g.* $F^*(\Phi) = \sum_j F^*(\phi_j)$. The objective can now be rewritten compactly as

$$\min_{\Phi,W,U,\mathbf{b}} F^*(\Phi) - \mathsf{tr}(\Phi' WX) + F(WX) + \ell(U'\Phi + \mathbf{b}\mathbf{1}') + \frac{1}{2}\|W\|^2 + \frac{1}{2}\|U\|^2 \tag{3.3}$$

This is bi-convex in $(\Phi, \mathbf{b})$ and (W, U), *i.e.* when one group is fixed, it is convex in the other. In order to derive a jointly convex reformulation, the authors note that

$$\ell(U'\phi + \mathbf{b}\mathbf{1}') = \max_R \left\{ \operatorname{tr}(R'(U'\Phi + \mathbf{b}\mathbf{1}')) - \ell^*(R) \right\},$$

where ℓ^* is the Fenchel conjugate of ℓ, and $R \in \mathbb{R}^{m \times t}$. They further note that

- For binary hinge loss, $\ell^*(r) = yr$ over $r \in [\min\{0, -y\}, \max\{0, -y\}]$, and otherwise ∞.
- For multi-class hinge loss, $\ell^*(\mathbf{r}) = \mathbf{y}'\mathbf{r}$ if $\mathbf{r} + \mathbf{y} \in \Delta^m := \left\{\mathbf{x} \in \mathbb{R}^m_+ : \mathbf{1}'\mathbf{x} = \mathbf{1}\right\}$, and ∞ otherwise.
- For multi-class logistic loss, $\ell^*(\mathbf{r}) = \sum_i (r_i + y_i) \log(r_i + y_i)$ if $\mathbf{r} + \mathbf{y} \in \Delta^m$, and ∞ otherwise. Similarly, $F(WX) = \max_\Lambda \{\operatorname{tr}(\Lambda' WX) - F^*(\Lambda)\}$.

With this, Equation 3.2 can be rewritten as

$$\begin{aligned}
& \min_{W,U,\mathbf{b},\Phi} \max_{R,\Lambda} F^*(\Phi) - \operatorname{tr}(\Phi' WX) + \operatorname{tr}(\Lambda' WX) - F^*(\Lambda) \\
& \qquad - \operatorname{tr}(R'(U'\Phi + \mathbf{b}\mathbf{1}')) - \ell^*(R) + \frac{\|W\|^2 + \|U\|^2}{2} \\
= {} & \min_{\Phi} \max_{R,\Lambda} \min_{W,U,\mathbf{b}} F^*(\Phi) - \operatorname{tr}(\Phi' WX) + \operatorname{tr}(\Lambda' WX) - F^*(\Lambda) \\
& \qquad - \operatorname{tr}(R'(U'\Phi + \mathbf{b}\mathbf{1}')) - \ell^*(R) + \frac{\|W\|^2 + \|U\|^2}{2} \\
= {} & \min_{\Phi} \max_{R\mathbf{1}=\mathbf{0},\Lambda} F^*(\Phi) - \frac{1}{2}\|(\Phi - \Lambda)X'\|^2 - \frac{1}{2}\|\Phi R'\| - F^*(\Lambda) - \ell^*(R)
\end{aligned} \tag{3.4}$$

A few things are happening here -

1. The optimal W and U for the last equality are $W = (\Phi + \Lambda)X'$ and $U = -\Phi R'$.
2. The first equality swaps $\min_{W,U,\mathbf{b}}$ with $\max_{R,\Lambda}$.

The authors note that this strong duality is non-trivial because Sion's minimax lemma [16] requires that the domain of (W, U) be compact, which is *not* assumed here. They then theorize:

Theorem 2 *For any W, U, $\boldsymbol{b}$, denote* $\mathcal{L}(\Phi, R) = F^*(\Phi) - \mathrm{tr}(\Phi' WX) + \mathrm{tr}(R'(U'\Phi + \boldsymbol{b1'})) - \ell^*(R)$. *Then,*

$$\min_{\Phi} \max_{R} \mathcal{L}(\Phi, R) = \max_{R} \min_{\Phi} \mathcal{L}(\Phi, R)$$

Based on this, the convex relaxation for Equation 3.4 can be derived using ReLU $F_r(Z) = \frac{1}{2}\|[Z]_+\|^2$ and its Fenchel dual $F_r{}^*(\Phi) = \frac{1}{2}\|\Phi\|^2$ for $\Phi \geq \mathbf{0}$(element-wise) and $+\infty$ otherwise. Equation 3.4 is specialized into

$$\min_{\Phi \geq \mathbf{0}} \max_{R\mathbf{1}=\mathbf{0}, \Lambda \geq \mathbf{0}} \frac{1}{2}\|\Phi\|^2 - \frac{1}{2}\|(\Phi - \Lambda)X'\|^2 - \frac{1}{2}\|\Phi R'\|^2 - \frac{1}{2}\|\Lambda\|^2 - \ell^*(R) \tag{3.5}$$

Here, both Φ and Λ are constrained to the positive orthant and hence are both sized $h \times t$. Typically, $t \gg h$, their ranks are h, and their column spaces have full rank. As a result, the authors perform a change of variable as

$$\Lambda = \Phi A, \tag{3.6}$$

where $A \in \mathbb{R}_+^{t\times t}$ and is not necessarily symmetric. Using this, Equation 3.5 can be rewritten as

$$\begin{aligned}\min_{\Phi\geq\mathbf{0}}\ \max_{R\mathbf{1}=\mathbf{0},\Lambda\geq\mathbf{0}}\ &\frac{1}{2}||\Phi||^2 - \frac{1}{2}\mathrm{tr}(\Phi'\Phi(I-A)X'X(I-A')) \\ &-\frac{1}{2}\mathrm{tr}(\Phi'\Phi R'R) - \frac{1}{2}\mathrm{tr}(\Phi'\Phi AA') - \ell^*(R)\end{aligned} \tag{3.7}$$

While this is still not convex, all occurrence of Φ are of the form $\Phi'\Phi$, thus making it possible to optimize over $\Phi'\Phi$ directly. Denoting $T := \Phi'\Phi \in \mathbb{R}^{t\times t}$, Equation 3.7 is rewritten as:

$$\begin{aligned}\min_{T\in\mathcal{T}_h}\ \max_{R\mathbf{1}=\mathbf{0},A\geq\mathbf{0}}\ &\frac{1}{2}\mathrm{tr}(T) - \frac{1}{2}\mathrm{tr}(T(I-A)X'X(I-A')) \\ &-\frac{1}{2}\mathrm{tr}(TR'R) - \frac{1}{2}\mathrm{tr}(TAA') - \ell^*(R),\end{aligned} \tag{3.8}$$

where $\mathcal{T}_h := \left\{\Phi'\Phi : \Phi \in \mathcal{R}_+^{h\times t}\right\} \subseteq \left\{T \in \mathcal{R}_+^{t\times t} : T \succeq \mathbf{0}\right\}$. There are a few things to note here:

1. $T \succeq \mathbf{0}$ means that T is positive semi-definite (PSD).

2. R and A are decoupled making inner optimization efficient, and given T, the maximization over R and A is concave because $T \succeq \mathbf{0}$.
3. The objective is convex in T, because maximization of linear terms is convex.

The authors address the challenge of the non-convexity of $\mathcal{T}_h$ by pointing out that the set $\mathcal{T}_h$ is a cone. If the fixed value of h is relaxed, then T_∞ is the *completely positive* (CP) matrix cone [17] and $\mathcal{T}_\infty$ is the convex hull or the tightest possible relaxation of $\mathcal{T}_h$ for any h. Using this, the final objective becomes

$$\begin{aligned} \min_{T \in \mathcal{T}} \max_{R\mathbf{1}=\mathbf{0}, A \geq \mathbf{0}} \frac{1}{2}\mathrm{tr}(T) - \frac{1}{2}\mathrm{tr}(T(I-A)X'X(I-A')) \\ - \frac{1}{2}\mathrm{tr}(TR'R) - \frac{1}{2}\mathrm{tr}(TAA') - \ell^*(R) \end{aligned} \tag{3.9}$$

The authors note that this technique can be extended beyond ReLU, to even non-elementwise transfer functions and hence provide a general framework for creating convex relaxations for two layer neural networks.

They further note that while Equation 3.9 is convex, the set $\mathcal{T}$ lacks a compact characterization in terms of linear or quadratic, PSD, or second-order conic constraints. Optimization over CP matrices is known hard [17], and projection to $\mathcal{T}$ is NP-hard [18]. Therefore, a special optimization framework has been proposed which is discussed in the next section.

CHAPTER 4

INDUCTIVE TRAINING

Due to the hardness of optimization over CP matrices, Ganapathiraman et al. [1] propose a framework for optimization that employs conditional gradient methods [19][20] that are free of projections through the Generalized Conditional Gradient algorithm (GCG) [2]. Since GCG operates on gauge regularized objectives, Equation 3.9 has to be rewritten.

Given a convex bounded set C containing the origin, the guage function induced by C evaluated at T can be defined as $\gamma_C(T) := \min\{\gamma \geq 0 : \gamma X = T, X \in C\}$. If no (γX) meets the condition, then $\gamma_C\,(T) := \infty$. Equation 3.9 has to be recast into this framework in order to deal with the unbound $\mathcal{T}$, which can be done by using the trace norm.

$$\begin{aligned} \mathcal{S} := {} & \mathcal{T} \cap \{T : \mathrm{tr}(T \leq 1\} \\ = {} & \text{conv}\, \mathcal{T}_1 \cap \{T : \mathrm{tr}(T) \leq 1\} \\ = {} & \text{conv}\left\{\mathbf{x}\mathbf{x}' : \mathbf{x} \in \mathbb{R}^t_+, \|\mathbf{x}\| \leq 1\right\}. \end{aligned} \tag{4.1}$$

The authors present the following lemma and state that the domain of the gauge implicitly enforces constraint on T.

Lemma 1 *$\mathcal{S}$ is convex, bounded, and closed. In addition,*

$$\gamma_{\mathcal{S}}(T) = \begin{cases} \mathrm{tr}(T) & T \in \mathcal{T} \\ +\infty & \textit{otherwise} \end{cases} \tag{4.2}$$

This allows Equation 3.9 to be equivalently rewritten as

$$\begin{aligned} \min_{T} J(T) := & \ \frac{1}{2}\gamma_{\mathcal{S}}(T) + g(T) \quad \text{where} \\ g(T) := & \max_{R\mathbf{1}=\mathbf{0}, A \geq \mathbf{0}} -\frac{1}{2}\mathrm{tr}(T(I-A)X'X(I-A')) \\ & -\frac{1}{2}\mathrm{tr}(TR'R) - \frac{1}{2}\mathrm{tr}(TAA') - \ell^*(R) \end{aligned} \tag{4.3}$$

which falls into the framework of GCG, as shown in Algorithm 1 [2] [20]. GCG iteratively seeks the steepest descent extreme point T^{new} (called the basis) of the set $\mathcal{S}$ with respect to the objective gradient (steps 3-4). It directly optimizes the underlying factor Φ after finding the conic combination with the existing solution.

Algorithm 1: General GCG algorithm

```
Randomly sample $\Phi_1 \in [0,1]^t$, and set $T_1 = \Phi_1'\Phi_1$
**while** $k = 1, 2, \ldots$ **do**
    Find $\nabla g(T_k)$ with $T_k = \Phi_k'\Phi_k$ by solving the inner maximization problem in $g(T_k)$ of Equation 4.3
    **Polar operator**: find a new basis via $T^{\text{new}} = \arg\max_{T \in S} < T, -\nabla g(T_k) >$.
    Compute the optimal combination weight $(\alpha, \beta) := \arg\min_{\alpha \geq 0, \beta \geq 0} J(\alpha T_k + \beta T^{\text{new}})$.
    **Locally** optimize $T$: $\Phi_{k+1} = \arg\min_{\Phi \geq 0} J(\Phi'\Phi)$ with $\Phi$ initialized by the value corresponding to $\Phi'\Phi = \alpha T_k + \beta T^{\text{new}}$.
**Return** $T_{k+1}$
```

4.1 Polar Operator

For the computational strategies for the operations in GCG, the authors bring our attention to the polar operator of $\mathcal{S}$, which tries to solve the following optimization in Equation 4.1:

$$\max_{T \in \mathcal{S}} \operatorname{tr}(G'T) \Longleftrightarrow \max_{\mathbf{x} \in \mathbb{R}_+^t, ||\mathbf{x}|| \leq 1} \operatorname{tr}(\mathbf{x}'G\mathbf{x}) \tag{4.4}$$

This optimization is NP-hard and are usually solved by semi-definite relaxations (SDP). For this optimization, the authors show that this bound can be tightened under the condition that $G \succeq \mathbf{0}$. This is presented as the theorem :

Theorem 3 *Assuming $G \succeq \mathbf{0}$, a $\frac{1}{4}$-approximate solution to Equation 4.4 can be found in $O(t^2)$ time.*

Here, a multiplicative α-approximate solution can be defined as:

Definition 1 *Let* $\alpha \in [0,1]$ *and assume an optimization problem* $\max_{\boldsymbol{x}\in\chi} f(\boldsymbol{x})$ *has non-negative optimal value. A solution* $\boldsymbol{x}^* \in \chi$ *is called* α*-apprpximate if* $f(\boldsymbol{x}^*) \geq \alpha \max_{\boldsymbol{x}\in\chi} f(\boldsymbol{x}) \geq 0$*. Similarly, the condition becomes* $0 \leq f(\boldsymbol{x}^* \leq \frac{1}{\alpha}\min_{\boldsymbol{x}\in\chi} f(\boldsymbol{x}))$ *for minimization problems.*

While this requirement of G being positive semi-definite is not satisfied in general, it happens to be fulfilled for this particular problem via Equation 4.3. The gradient of g is negative semi-definite, and can be calculated as

$$-\frac{1}{2}(I-A)X'X(I-A') - \frac{1}{2}R'R - \frac{1}{2}AA', \tag{4.5}$$

where the R and A are the optimal solution to the inner maximizations.

4.2 Optimization using equivalent min-min objective

For the local optimization, g has to be converted from a *min-max* problem to a *min-min* problem so that it can be solved by alternating, *i.e.*, optimizing one variable by fixing the other variables, as done in Algorithm 2.

This can be done by

$$\begin{aligned} g(\Phi'\Phi) = {} & \max_{R\mathbf{1}=\mathbf{0}} \left\{ -\frac{1}{2}\|\Phi R'\|^2 - \ell^*(R) \right\} + \max_{A\geq\mathbf{0}} \left\{ -\frac{1}{2}\|\Phi(I-A)X'\|^2 - \frac{1}{2}\|\Phi A\|^2 \right\} \\ = {} & \max_{R} \min_{\mathbf{b}} \left\{ \mathbf{b}'R\mathbf{1} - \ell^*(R) - \max_{U} - \operatorname{tr}(U'\Phi R') - \frac{\|U\|^2}{2} \right\} \\ & + \max_{A} \min_{M\geq\mathbf{0}} \left\{ -\frac{\|\Phi(I-A)X'\|^2}{2} - \frac{\|\Phi A\|^2}{2} + \operatorname{tr}(M'A) \right\} \\ = {} & \min_{U,\mathbf{b}} \left\{ \ell(U'\Phi + \mathbf{b}\mathbf{1}') + \frac{1}{2}\|U\|^2 \right\} + \min_{M\geq\mathbf{0}} h(M,\Phi), \end{aligned} \tag{4.6}$$

$$\text{where} \quad h(M,\Phi) := \max_{A} \left\{ -\frac{1}{2}\|\Phi(I-A)X'\|^2 - \frac{1}{2}\|\Phi A\|^2 + \operatorname{tr}(M'A) \right\} \tag{4.7}$$

Using this, the local optimization $\min_{\Phi\geq\mathbf{0}} J(\Phi'\Phi) = \min_{\Phi\geq\mathbf{0}} \frac{1}{2}\|\Phi\|^2 + g(\Phi'\Phi)$ can now be solved by alternating between $(U,\mathbf{b})$, M, and Φ. This is shown in Algorithm 3.

The optimization over M and Φ is tricky because h requires the nested optimization over A. Since h is quadratic in A, a *closed-form* solution for A can be calculated using the Woodbury formula [21] and Nesterov smoothing [22]. This is done by the introduction of a small strongly convex regularizer ($\mu > 0$) on A in the definition of $h(M,\Phi)$:

$$h_\mu(M,\Phi) := \max_{A} \left\{ -\frac{1}{2}\|\Phi(I-A)X'\|^2 - \frac{1}{2}\|\Phi A\|^2 + \operatorname{tr}(M'A) - \frac{\mu}{2}\operatorname{tr}(A(X'X+I)A') \right\}, \tag{4.8}$$

The optimal A can be calculated by setting its gradient to zero:

$$A = (\Phi'\Phi = \mu I)^{-1}(M + \Phi'\Phi X'X)(X'X + I)^{-1} \tag{4.9}$$

for efficiency, A can be computed using the Woodbury formula as:

$$\begin{aligned} \mu A &= (M + \Phi'\Phi X'X)(X'X + I)^{-1} \\ &\quad - \Phi'(\mu I + \Phi\Phi')^{-1}\Phi(M + \Phi'\Phi X'X)(X'X + I)^{-1}. \end{aligned} \tag{4.10}$$

Now, optimization over $(U, \mathbf{b})$ is the standard supervised learning, and optimization over M and Φ can be done using LBGFS-B with A computed using Equation 4.10.

Algorithm 2: Solving Equation 3.9 for T by the GCG algorithm

```
Randomly sample $\Phi_1 \in [0, 1]^t$, and set $T_1 = \Phi_1'\Phi_1$
while $k = 1, 2, \ldots$ do
    if $k = 1$ then
        $(U_k, \mathbf{b}_k)$ = optimal $U$ and $\mathbf{b}$ in Equation 4.6 for $\Phi_1$.
        $M_k$ = optimal $M$ in Equation 4.6 for $\Phi_1$
    Recover the optimal $R : R_k = \nabla\ell(U_k'\Phi_k + \mathbf{b}_k\mathbf{1}')$.
    Recover the optimal $A$ by Equation 4.10.
    Compute the gradient $G_k$ of $g_\mu$ at $T_k = \Phi_k'\Phi_k$ via Equation 4.5, with $R$ and $A$
      served by $R_k$ and $A_k$, respectively.
    Compute a new basis $\mathbf{x}_k$ by approximately solving $\arg\max_{\mathbf{x}\in\mathbb{R}^t_+, \|\mathbf{x}\|\leq 1} \mathbf{x}'(-G_k)\mathbf{x}$
      (Theorem 3).
    Line search: $(\alpha, \beta) := \arg\min_{\alpha\geq 0, \beta\geq 0} J(\alpha T_k + \beta\mathbf{x}_k\mathbf{x}_k')$.
    Set $\Phi_{\text{tmp}} = (\sqrt{\alpha}\Phi_k', \sqrt{\beta}\mathbf{x}_k)'$.
    Local search: $(\Phi_{k+1}, U_{k+1}, \mathbf{b}_k, M_{k+1}) :=$ Local_Opt$(\Phi_{\text{tmp}}, U_k, \mathbf{b}_k, M_k)$ using
      Algorithm 3.
Return $T_{k+1}$.
```

Algorithm 3: Local optimization used by GCG

```
Require $(\Phi_{\text{tmp}}, U_k, \mathbf{b}_k, M_k)$ from the current step.
Initialize: $\Phi = \Phi_{\text{tmp}}, U = U_k, \mathbf{b} = \mathbf{b}_k, M = M_k$.
for $t = 1, 2, \ldots$ do                                          // till change is small
    $(U, \mathbf{b}) = \arg\min_{U,\mathbf{b}} \left\{\ell(U'\Phi + \mathbf{b}\mathbf{1}') + \frac{1}{2}\|U\|^2\right\}$.
    $M = \arg\min_{M\geq\mathbf{0}} h(M, \Phi)$.
    $\Phi = \arg\min_{\Phi\geq\mathbf{0}} \left\{\ell(U'\Phi + \mathbf{b}\mathbf{1}') + h(M, \Phi)\right\}$.
Return $(\Phi, U, \mathbf{b}, M)$
```

CHAPTER 5

FASTER INDUCTIVE TRAINING

A major bottle-neck for local optimization in Algorithm 3 is the nested optimization of A in M and Φ. According to Ganapathiraman et al. [1], the time complexity for local optimization is $O(t^3)$ and if $n \ll t$, then $O(nt^2)$ (recall that $A \in \mathbb{R}^{t\times t}$ and $X \in \mathbb{R}^{n\times t}$). Yet, A is required for optimizing Equation 3.9 and therefore, our key contribution in this thesis is the decoupling of the optimization of A from the local optimization step, making alternating between the variables efficient. This new strategy will remove the need for nested optimizations.

This chapter details an updated framework with a reformulated objective function as described in Section 5.1, followed by Section 5.2 that sketches the proposed algorithms for optimizing Equation 3.9 using GCG and the local optimization of the variables.

5.1 Reformulating the Objective

In Algorithm 2, the optimization of Φ is *non-convex*, and makes uses of a local-optimization strategy that is described in Algorithm 3. A was introduced for the reformulation of the objective for the convenient optimization of Φ as $T := \Phi'\Phi$ in Equation 3.7. This motivates us to explore the optimization of Λ directly, and eventually computing A from the optimal Λ. Recall from Equation 3.6 that

$$\Lambda = \Phi A.$$

By using this in Equation 4.6, we can reformulate the min-min objective, while maintaining its solvability via alternating, as:

$$
\begin{aligned}
g(\Phi'\Phi) = \; & \max_{R\mathbf{1}=\mathbf{0}} \left\{ -\frac{1}{2}||\Phi R'||^2 - \ell^*(R) \right\} + \max_{\Lambda \geq \mathbf{0}} \left\{ \frac{1}{2}||(\Phi - \Lambda)X'||^2 - \frac{1}{2}||\Lambda||^2 \right\} \\
= \; & \max_{R} \min_{\mathbf{b}} \left\{ \mathbf{b}'R\mathbf{1} - \ell^*(R) - \max_{U} - \mathrm{tr}(U'\Phi R') - \frac{||U||^2}{2} \right\} \\
& + \max_{\Lambda} \min_{N \geq \mathbf{0}} \left\{ -\frac{||(\Phi - \Lambda)X'||^2}{2} - \frac{||\Lambda||^2}{2} + \mathrm{tr}(N'\Lambda) \right\} \\
= \; & \min_{U,\mathbf{b}} \left\{ \ell(U'\Phi + \mathbf{b}\mathbf{1}') + \frac{1}{2}||U||^2 \right\} + \min_{N \geq \mathbf{0}} q\left(N, \Phi\right),
\end{aligned}
\tag{5.1}
$$

$$
\text{where} \quad q\left(N, \Phi\right) := \max_{\Lambda} \left\{ -\frac{1}{2}||(\Phi - \Lambda)X'||^2 - \frac{1}{2}||\Lambda||^2 + \mathrm{tr}(N'\Lambda) \right\} \tag{5.2}
$$

This reformulation is significant because as we will show next, $q\left(N, \Phi\right)$ does not lead to a nested optimization of A, allowing us to optimize A outside of the local optimization. Recall that optimization of A within the local optimization step for $h\left(M, \Phi\right)$ (Equation 4.7) was the cause of the bottle-neck.

5.2 Optimization

We begin our optimizations by first targeting Λ. Fortunately, by taking the derivative of Equation 5.2 and setting it to 0, we arrive as a closed form solution for Λ.

$$\begin{aligned} \frac{\delta}{\delta\Phi} q\left(N,\Phi\right) &= 0 \\ \Rightarrow\quad (\Phi X' - \Lambda X')X - \Lambda + N &= 0 \\ \Rightarrow\quad \Lambda(X'X + I) &= \Phi X'X + N \end{aligned} \tag{5.3}$$

$$\Lambda = (\Phi X'X + N)(X'X + I)^{-1} \tag{5.4}$$

We can substitute this value of Λ back in Equation 5.2 and we get

$$\begin{aligned} q\left(N,\Phi\right) = \quad & -\frac{1}{2}\|(\Phi - (\Phi X'X + N)(X'X + I)^{-1})X'\|^2 \\ & -\frac{1}{2}\|(\Phi X'X + N)(X'X + I)^{-1}\|^2 \\ & + \mathrm{tr}(\mathrm{N}'(\Phi X'X + N)(X'X + I)^{-1}) \end{aligned} \tag{5.5}$$

Using Equation 5.5, we can now obtain the optimal N by computing the derivative of $q\left(N,\Phi\right)$ with respect to N and optimizing using LBFGS-B solver.

$$\begin{aligned} \frac{\delta}{\delta N} q\left(N,\Phi\right) = \quad & (\Phi - (\Phi X'X + N)(X'X + I)^{-1})X'X(X'X + I)^{-1} \\ & - (\Phi * X'X + N)(X'X + I)^{-1}(X'X + I)^{-1} \\ & + (\Phi X'X + N)(X'X + I)^{-1} + N(X'X + I)^{-1} \end{aligned} \tag{5.6}$$

Similarly, the derivative of Equation 5.5, combined with the gradient of $\ell(U'\Phi + \mathbf{b1}')$, can be used to optimize Φ.

$$\begin{aligned}\frac{\delta}{\delta\Phi} q\,(N, \Phi) = \quad & -t + t\,(X'X + I)^{-1} X'X \\ & -(\Phi X'X + N)(X'X + I)^{-1}(X'X + I)^{-1} X'X \\ & + \Phi + N(X'X + I)^{-1} X'X\end{aligned} \tag{5.7}$$

$$\text{where, } t = (\Phi - (\Phi X'X + N)(X'X + I)^{-1})X'X. \tag{5.8}$$

Using the optimal Φ and Λ, optimal A can be recovered as

$$A = \text{lsqminnorm}(\Phi, \Lambda), \tag{5.9}$$

where the lsqminnorm function computes a unique minimal norm least square solution for the system of linear equations in Equation 3.6.

Finally, we can now run local optimization $\min_{\Phi\geq\mathbf{0}} J(\Phi'\Phi) = \min_{\Phi\geq\mathbf{0}} \frac{1}{2}||\Phi||^2 + g\,(\Phi'\Phi)$ by alternating between $(U, \mathbf{b})$, N, and Φ. Algorithm 4 shows the faster inductive training updates for Equation 3.9, and Algorithm 5 details the new local optimization used by Algorithm 4. This local optimization strategy now focuses on the optimization of low-rank variables while computing closed-form solution for Λ. The optimization for A is now taken out of the local-optimization step as in Algorithm 3 (nested within the optimization of $h(M, \Phi)$) and is separately computed using significantly faster techniques in step 8 of Algorithm 4.

Algorithm 4: Solving Equation 3.9 for T by the GCG algorithm

```
1  Randomly sample $\Phi_1 \in [0,1]^t$, and set $T_1 = \Phi_1'\Phi_1$
2  while $k = 1, 2, \ldots$ do
3      Compute $\Lambda$ using Equation 5.4
4      if $k = 1$ then
5          $(U_k, \mathbf{b}_k)$ = optimal $U$ and $\mathbf{b}$ in Equation 4.6 for $\Phi_1$.
6          $N_k$ = optimal $N$ in Equation 5.5
7      Recover the optimal $R : R_k = \nabla\ell(U_k'\Phi_k + \mathbf{b}_k\mathbf{1}')$.
8      Compute the optimal $A$: lsqminnorm($\Phi$, $\Lambda$)
9      Compute the gradient $G_k$ of $g_\mu$ at $T_k = \Phi_k'\Phi_k$ via Equation 4.5, with $R$ and $A$
        served by $R_k$ and $A_k$, respectively.
10     Compute a new basis $\mathbf{x}_k$ by approximately solving $\arg\max_{\mathbf{x}\in\mathbb{R}^t_+, \|\mathbf{x}\|\le 1} \mathbf{x}'(-G_k)\mathbf{x}$
        (Theorem 3).
11     Line search: $(\alpha, \beta) := \arg\min_{\alpha\ge 0, \beta\ge 0} J(\alpha T_k + \beta \mathbf{x}_k\mathbf{x}_k')$.
12     Set $\Phi_{\text{tmp}} = (\sqrt{\alpha}\Phi_k', \sqrt{\beta}\mathbf{x}_k)'$.
13     Local search: $(\Phi_{k+1}, U_{k+1}, \mathbf{b}_k, N_{k+1}) :=$ Local_Opt($\Phi_{\text{tmp}}, U_k, \mathbf{b}_k, N_k$) using
        Algorithm 5.
14 Return $T_{k+1}$.
```

5.3 Computational Complexity

The time complexity for the variable optimizations assuming LBGFS-B solver is used is of the order $O(t^2)$. In Algorithm 3, the challenge was optimizing A, which made the complexity $O(t^3)$. In Algorithm 4 however, A is solved by calculating the minimal norm least squares

Algorithm 5: Local optimization used by GCG

```
Require $(\Phi_{\text{tmp}}, U_k, \mathbf{b}_k, N_k)$ from the current step.
Initialize: $\Phi = \Phi_{\text{tmp}}, U = U_k, \mathbf{b} = \mathbf{b}_k, N = N_k$.
for $t = 1, 2, \ldots$ do                                   // till change is small
    $(U, \mathbf{b}) = \arg\min_{U,\mathbf{b}} \left\{\ell(U'\Phi + \mathbf{b}\mathbf{1}') + \frac{1}{2}\|U\|^2\right\}$.
    $N = \arg\min_{N \geq \mathbf{0}} q(N, \Phi)$.
    $\Phi = \arg\min_{\Phi \geq \mathbf{0}} \left\{\ell(U'\Phi + \mathbf{b}\mathbf{1}') + q(N, \Phi)\right\}$.
Return $(\Phi, U, \mathbf{b}, N)$
```

solution of a system of linear equations. Typically this involves the calculation of the Moore-Penrose pseudo-inverse [23]. The time complexity of computing the pseudo-inverse of a matrix of size $h \times t$ is of the order $O(t \times h \times \min(t, h))$. This is a significant reduction in the computation time as compared to Algorithm 3.

CHAPTER 6

EXPERIMENTS

6.1 Experimental setup

We evaluate the proposed faster inductive two-layer model (F-CVX) based on Algorithm 4 by comparing it with the original inductive two-layer model (CVX-IN) [1] based on two metrics - mean classification error and running time for local optimization step. To perform this evaluation, we trained both models over a some benchmark datasets for "real world" binary classification tasks. All experiments were conducted on a 16GB RAM Intel Core-i7 quad-core CPU.

6.2 Convergence to global optima

A key advantage of a convex neural network is the guarantee of the convergence to the global optima. Since our model optimizes over the same objective function as [1], we hypothesised that:

1. Both models (F-CVX and CVX-IN) should converge to the same objective value.
2. Our model (F-CVX) should converge much faster.

We evaluate this by training both models on 1000 samples of the MNIST dataset [8] till the reduction in the objective value plateaus, and then compare the time taken for both models to perform the local optimization step per iteration. We can clearly see from Figure 3 that not

	COIL	**G241N**	**Letter**	**MNIST**	**CIFAR-SM**	**USPS**
CVX-IN	41	28	22	22	27	13
F-CVX	40	25	7	21	22	12

TABLE I: Mean errors of models on 100 samples from each dataset.

only do both models converge to similar values in roughly the same number of iterations for this dataset, but also that our model reaches this optimal value significantly faster.

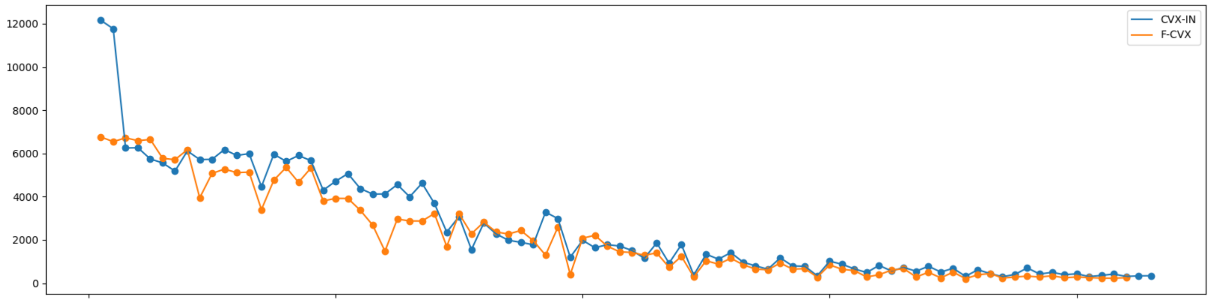
Figure 4 shows the time taken for the objective optimization in F-CVX model to plateau and the corresponding value for the non-converged CVX-IN model (note that CVX-IN model will plateau at similar values after a longer time of training) using 1000 samples of MNIST and Letter, and 750 samples of the remaining datasets. This serves as another validation for our claim that the objective is being optimized faster using Algorithm 4.

6.3 Comparison on smaller subsets

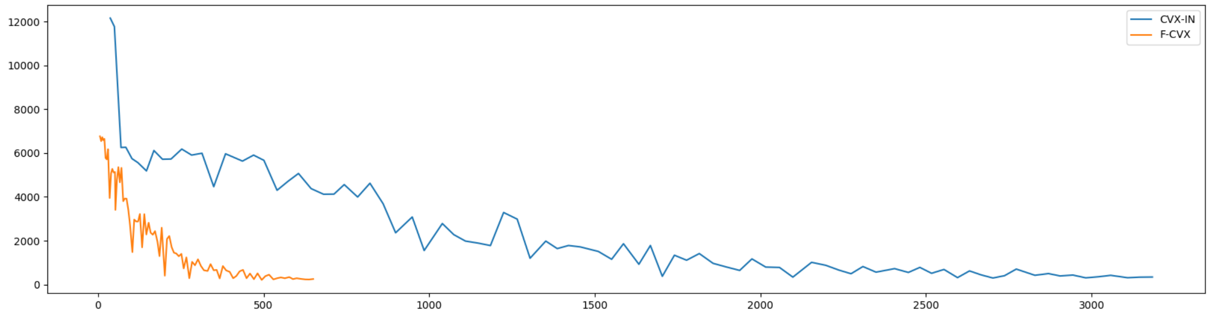
We compare the accuracies of the F-CVX and CVX-IN models, first, by using 100 training and testing samples of six "real world" classification datasets - COIL [24], G241N [14], Letter [10], MNIST [8], CIFAR-SM [11], and USPS [13]. We can see from Table I and Figure 5 that while both models achieve similar mean testing errors, the optimization time for F-CVX is lower than that of CVX-IN.

6.4 Comparison on larger subsets

We then trained both models on 1000 training and testing samples each of Letter and MNIST, and 750 training and testing samples each of COIL, G241N, Nested and USPS datasets.

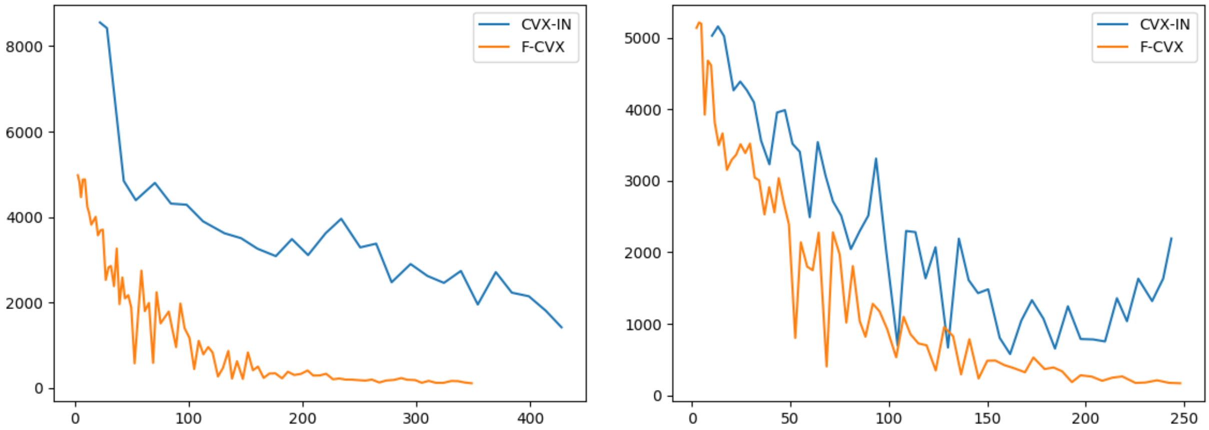


(a) Objective value at each iteration

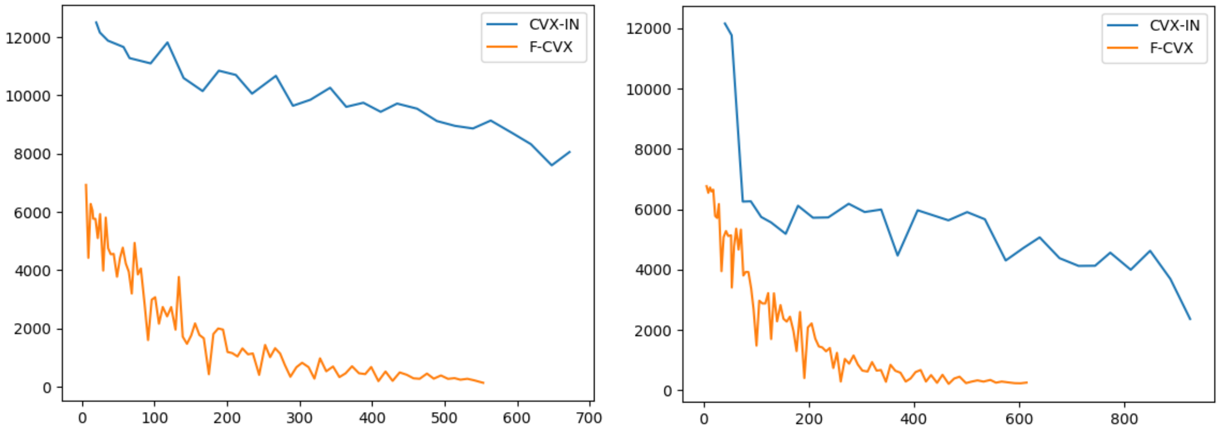


(b) Objective value vs. time (in seconds)

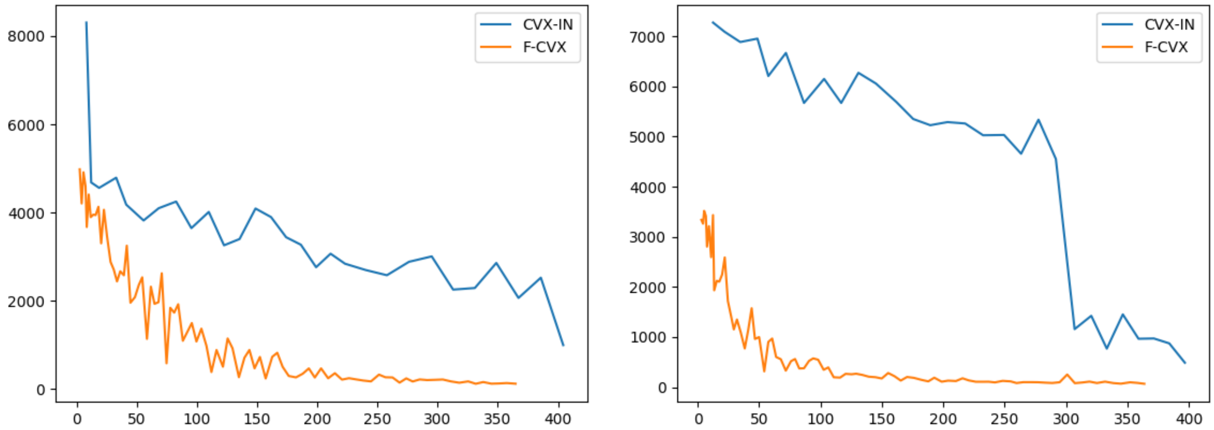
Figure 3: Comparing the number of steps and the total time taken for both models to converge to the global optima while training on 1000 samples of MNIST dataset. (a) Plots the objective on the X-axis against the iterations on Y-axis. (b) Plots the objective on X-axis against the local optimization time on Y-axis.



(a) COIL (b) G241N

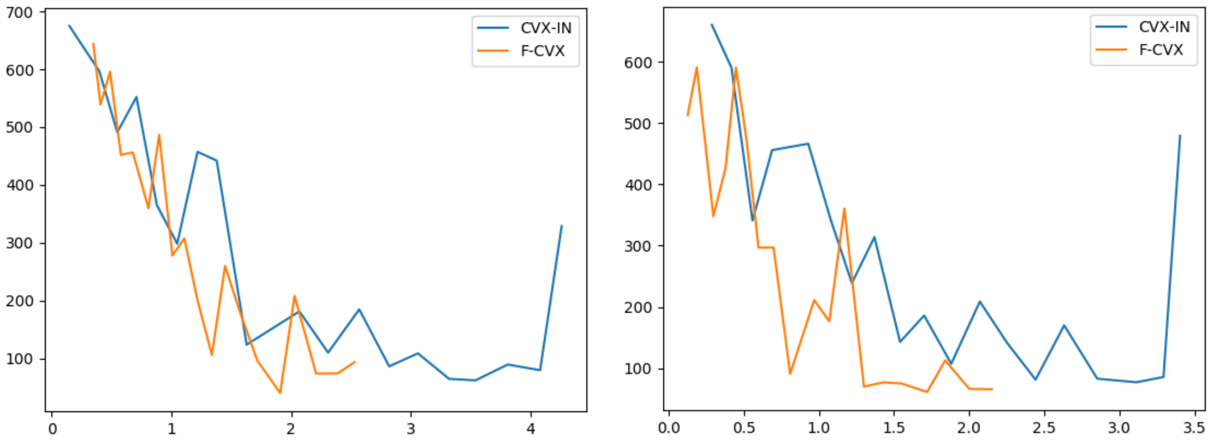


(c) Letter (d) MNIST

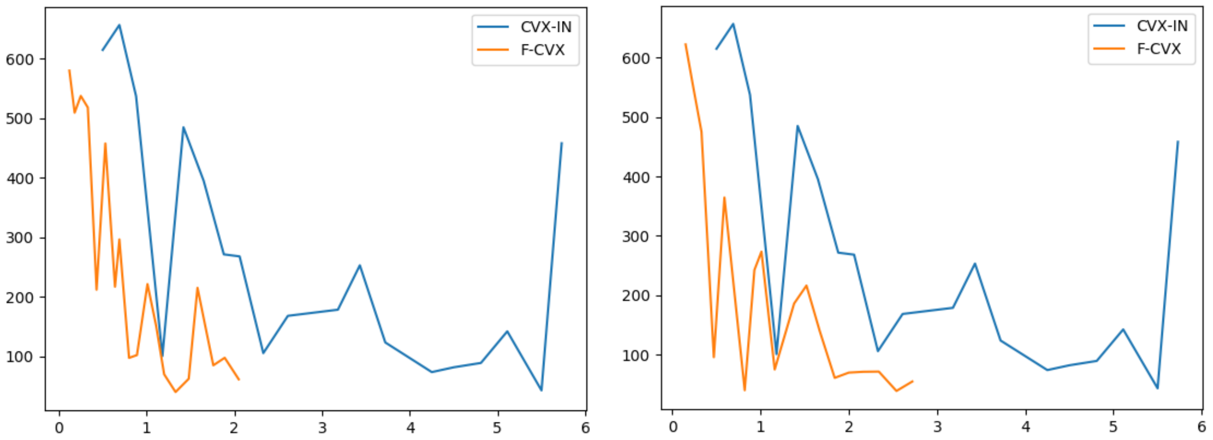


(e) CIFAR-SM (f) USPS

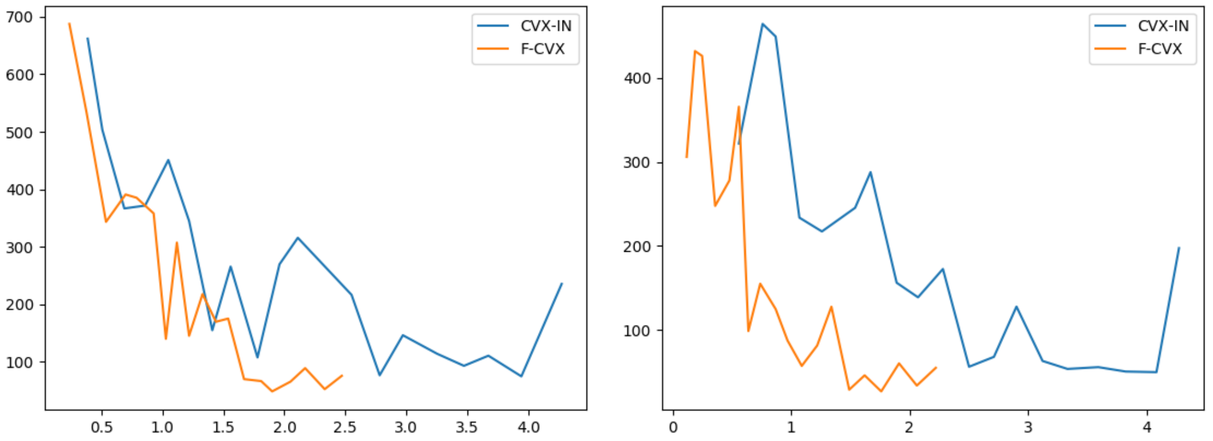
Figure 4: Time taken for optimization per 1000 samples of Letter and MNIST datasets, and 750 samples each of the rest. Y-axis shows the objective value and X-axis plots time (in seconds).



(a) COIL (b) G241N



(c) Letter (d) MNIST



(e) CIFAR-SM (f) USPS

Figure 5: Time taken for optimization per 100 samples of each dataset. X-axis shows the objective value and Y-axis plots time (in seconds) for 20 iterations of training for each model.

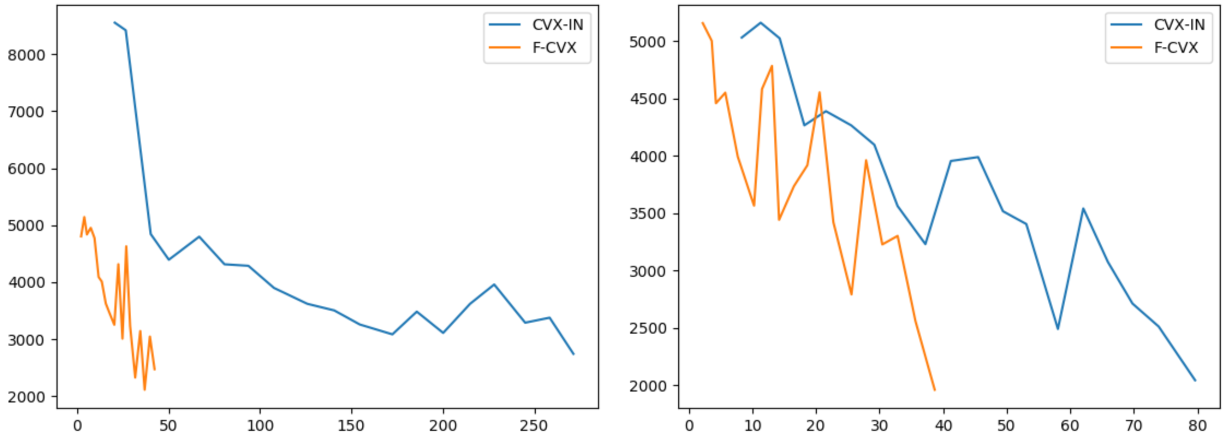
	COIL	**G241N**	**Letter**	**MNIST**	**CIFAR-SM**	**USPS**
CVX-IN	14.267	15.6	4.8	9.2	23.867	9.2
F-CVX	14.133	14.533	6.3	8.667	22.0	9.067

TABLE II: Mean errors of models on 1000 samples from Letter and MNIST, and 750 samples of the remaining datasets.

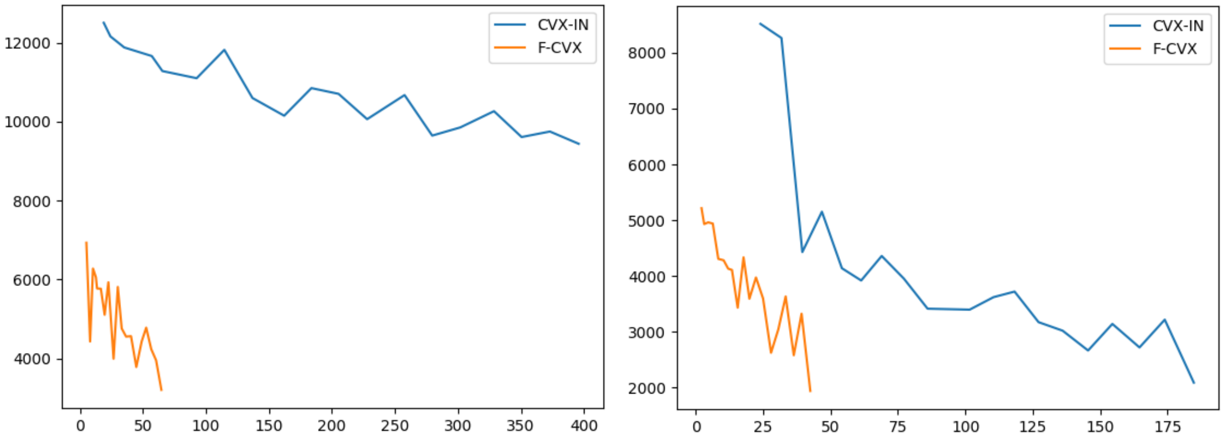
Once again, we see in Table II that both models achieve a similar test accuracy. Figure 6 shows F-CVX completing the 20 iterations significantly faster than the CVX-IN model, as also demonstrated in Section 6.2. Figure 7 displays the testing accuracies over time for both models and shows that the F-CVX model reaches the optimal accuracies achieved by these models much faster than the CVX-IN model.

6.5 Scaling to larger datasets

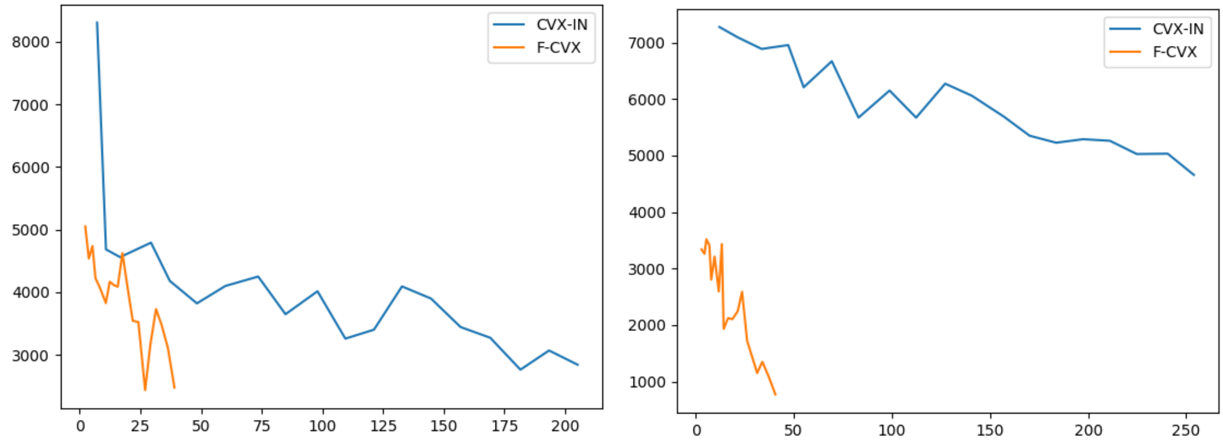
A key merit of the F-CVX model is that for the first time, it is now possible to train on 10,000 samples. We evaluated this by using 10,000 training and testing samples from the MNIST dataset and achieved 7.2% mean classification error. 10 iterations of local optimization under this setting took 2 hours and 51 minutes. Figure 8 shows the training and testing accuracies over 35 training iterations of the F-CVX model on 10,000 training and 10,000 testing samples of the MNIST dataset. The total optimization time for performing 35 iterations of training using the mentioned hardware is 9 hours and 47 minutes. To the best of our knowledge, no previous convex neural network model has been able to achieve this.



(a) COIL (b) G241N

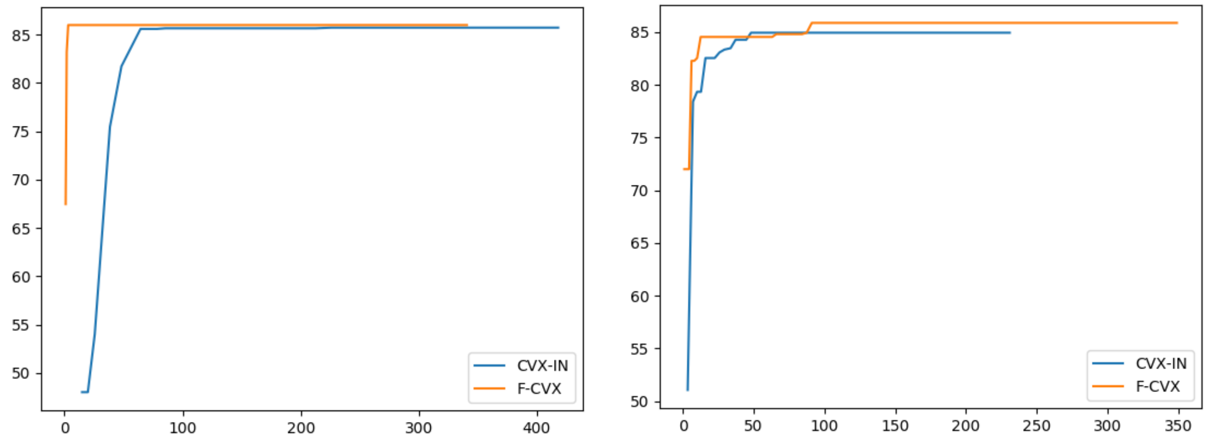


(c) Letter (d) MNIST

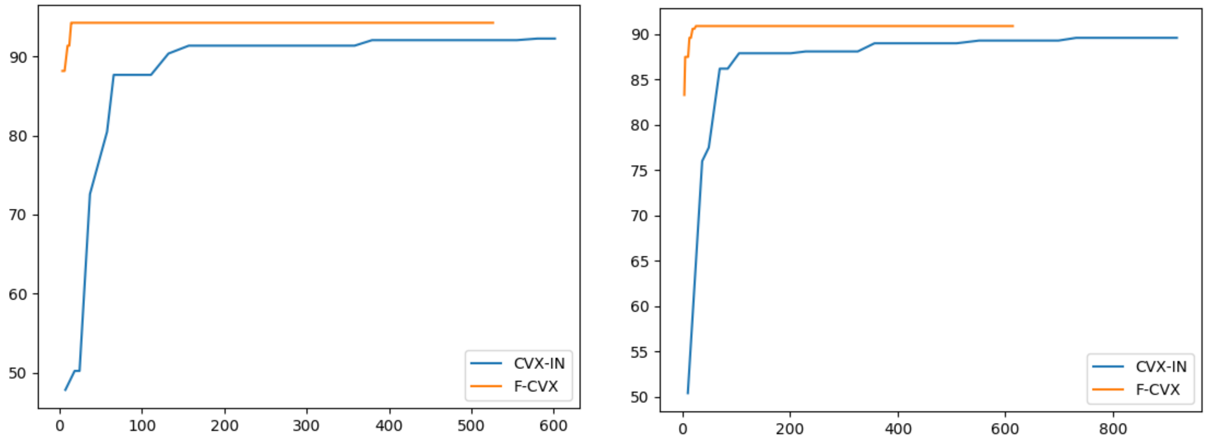


(e) CIFAR-SM (f) USPS

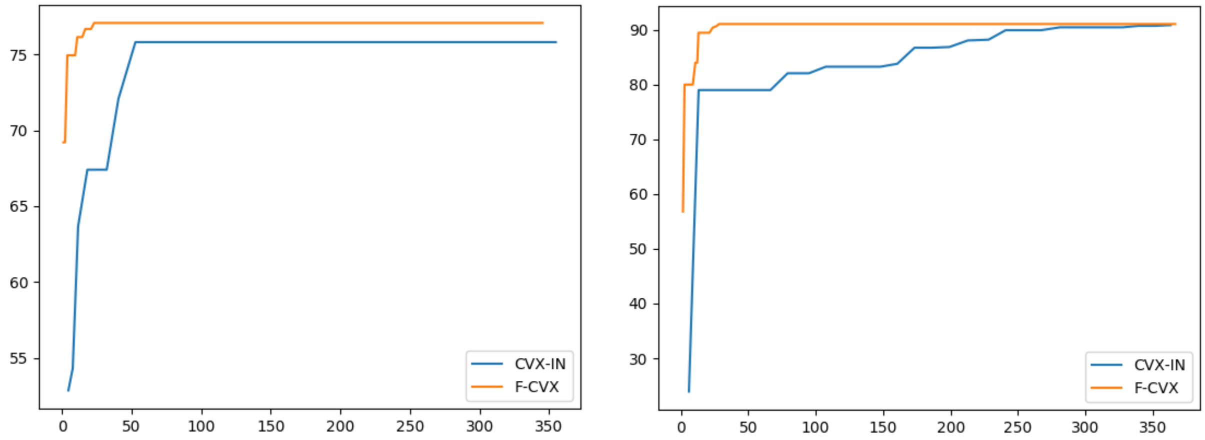
Figure 6: Time taken for optimization per 1000 samples of Letter and MNIST datasets, and 750 samples of the remaining. X-axis plots the time (in seconds) and Y-axis plots objective value over 20 iterations of training for each model.



(a) COIL (b) G241N



(c) Letter (d) MNIST



(e) CIFAR-SM (f) USPS

Figure 7: Plots the testing accuracies over time for 85 training iterations of the F-CVX model and 30 training iterations for CVX-IN model using 1000 samples of MNIST and Letter datasets, and 750 samples of the rest. Y-axis plots the accuracy and X-axis plots the time (in seconds).

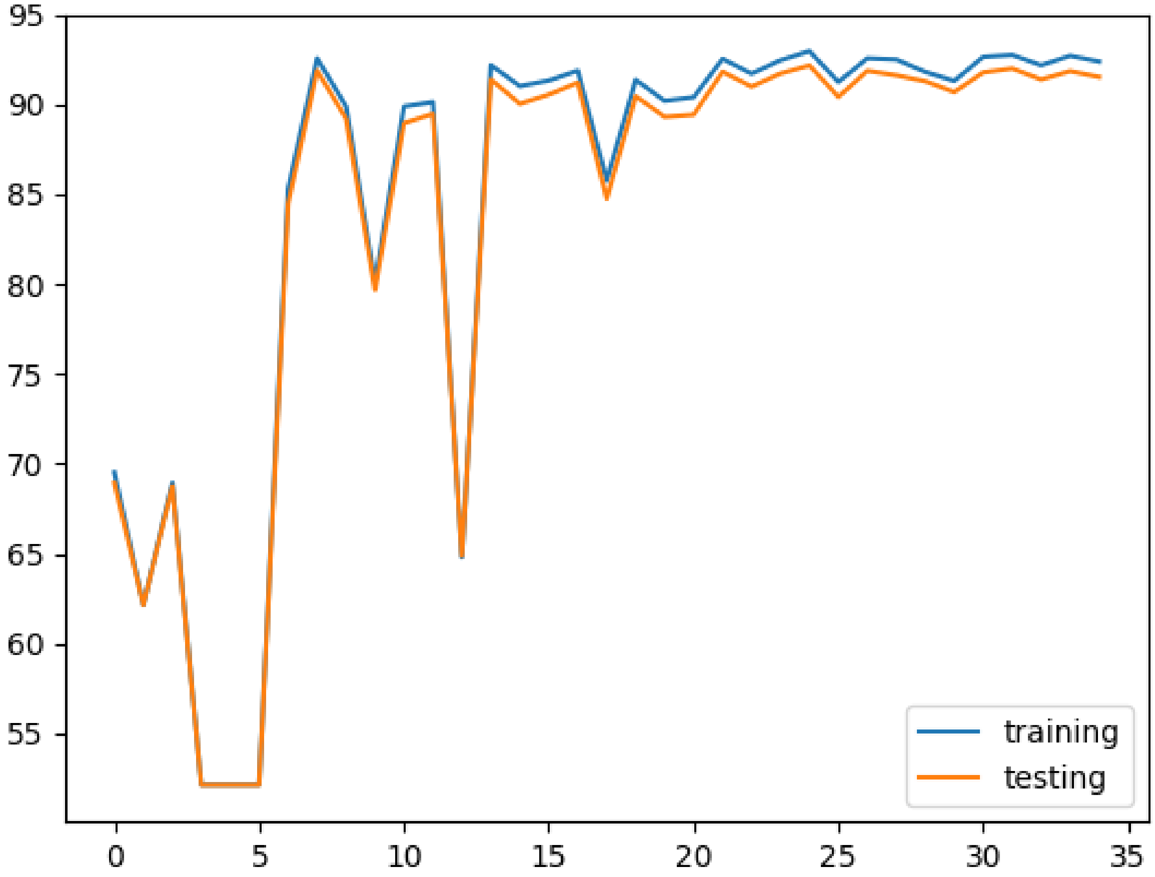


Figure 8: Training and Testing accuracies over 35 training iterations of F-CVX model on 10,000 training samples of MNIST dataset.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this thesis, we introduced convex two-layer modeling with the problems involved in the choice of loss functions and the use of parametric Bregman transfer. Key steps of local-optimization in previous works was involving nested optimizations of a large matrices, inevitably causing a huge bottle-neck in terms of optimization time while also preventing the models from scaling to larger datasets.

We extended the idea from the previous works, by using the same objective functions but modifying them just enough to allow local-optimization to be focused on non-nested low rank matrices and closed-form strategies for the large matrix. This provides a significant boost to the optimization time and allows the model to scale to 10,000 samples for the first time. To the best of our knowledge, no known convex neural network model has achieved the same scale.

Although our empirical results show promise, some work towards establishing theoretical guarantees would be interesting. Further, extension of this model to leverage performance boosts offered by high performance clusters and parallel processing would be fruitful.

CITED LITERATURE

1. V. Ganapathiraman, Z. Shi, X. Zhang, and Y. Yu. Inductive two-layer modeling with parametric Bregman transfer. *In* J. Dy and A. Krause, eds., *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 1636–1645. PMLR, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. URL `http://proceedings.mlr.press/v80/ganapathiraman18a.html`.

2. X. Zhang, D. Schuurmans, and Y. liang Yu. Accelerated training for matrix-norm regularization: A boosting approach. *In* F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., *Advances in Neural Information Processing Systems 25*, pp. 2906–2914. Curran Associates, Inc., 2012.

3. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618, 9780262035613.

4. S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.

5. M. A. Nielsen. Neural networks and deep learning, 2018. URL `http://neuralnetworksanddeeplearning.com/`.

6. T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edn., 1997. ISBN 0070428077, 9780070428072.

7. H. W. Kuhn and A. W. Tucker. Nonlinear programming. *In Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492. University of California Press, Berkeley, Calif., 1951. URL `https://projecteuclid.org/euclid.bsmsp/1200500249`.

8. Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

9. P. W. Frey and D. J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6(2):161–182, Mar 1991. ISSN 1573-0565. doi:10.1007/BF00114162. URL `https://doi.org/10.1007/BF00114162`.

10. D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

11. O. Aslan, H. Cheng, X. Zhang, and D. Schuurmans. Convex two-layer modeling. *In* C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds., *Advances in Neural Information Processing Systems 26*, pp. 2985–2993. Curran Associates, Inc., 2013. URL `http://papers.nips.cc/paper/4867-convex-two-layer-modeling.pdf`.

12. A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/ kriz/cifar.html`.

13. J. J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, May 1994. ISSN 0162-8828. doi:10.1109/34.291440.

14. O. Chapelle, B. Schölkopf, and A. Zien, eds. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006. URL `http://www.kyb.tuebingen.mpg.de/ssl-book`.

15. P. Auer, M. Herbster, and M. K. Warmuth. Exponentially many local minima for single neurons. *In NIPS*. 1995.

16. M. Sion. On general minimax theorems. *Pac. J. Math.*, 8:171–176, 1958. ISSN 0030-8730.

17. A. Berman and N. Shaked-Monderer. *Completely Positive Matrices*. World Scientific Publishing Company Pte Limited, 2003. ISBN 9789812795212. URL `https://books.google.com/books?id=AllToPtYGGgC`.

18. P. J. C. Dickinson and L. Gijben. On the computational complexity of membership problems for the completely positive cone and its dual. *Computational Optimization and Applications*, 57(2):403–415, Mar 2014. ISSN 1573-2894. doi:10.1007/s10589-013-9594-z. URL `https://doi.org/10.1007/s10589-013-9594-z`.

19. M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. *In* S. Dasgupta and D. McAllester, eds., *Proceedings of the 30th International Conference on Machine Learning*, vol. 28 of *Proceedings of Machine Learning Research*, pp. 427–435. PMLR, Atlanta, Georgia, USA, 17–19 Jun 2013. URL `http://proceedings.mlr.press/v28/jaggi13.html`.

20. Z. Harchaoui, A. Juditsky, and A. Nemirovski. Conditional gradient algorithms for norm-regularized smooth convex optimization. *Math. Program.*, 152(1-2):75–112, Aug 2015. ISSN 0025-5610. doi:10.1007/s10107-014-0778-9. URL `http://dx.doi.org/10.1007/s10107-014-0778-9`.

21. M. A. Woodbury. *Inverting modified matrices.* Statistical Research Group, Princeton University, Princeton, NJ, 1950.

22. Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, May 2005. ISSN 1436-4646. doi:10.1007/s10107-004-0552-5. URL `https://doi.org/10.1007/s10107-004-0552-5`.

23. J. Gower. Generalized inverse matrices. *Technometrics*, 14:806–807, 04 2012. doi: 10.1080/00401706.1972.10488972.

24. S. A. Nene, S. K. Nayar, and H. Murase. Columbia university image library (coil-20). 1996. URL `http://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php`.

VITA

Chandrasekhara Ganesh Jagadeesan

cjagad2@uic.edu

Education

Master of Science in Computer Science Aug 2017 - July 2019
University of Illinois at Chicago GPA: 3.83/4.00

Bachelor of Technology in Computer Science and Engineering Aug 2012 - May 2016
SRM University, Chennai

Experience

Graduate Research Assistant, *UIC EVL* Jan 2019 - Present
SMART: Developing the front-end for a data driven visual analysis web application for head and neck cancer treatment.

Research Assistant, *UIC AHS* Oct 2018 - Dec 2018
Mis-information Detection: Researched techniques to detect the spread of mis-information regarding E-Cigarettes and HPV Vaccines using Deep learning and NLP Techniques.
Webscraping and Lexicon Matching: Wrote scripts for automated web crawling, information mining, and lexicon matching using python

Research Engineer, *Neubay Inc.* Oct 2018 - Dec 2018
Developed and tested different deep learning architectures that could represent uncertainty for benchmark datasets as well as data from Neubay customers.

Skills

Technical: Python, Javascript, React.js, D3.js, SQL, C++, Git

Other: Client interviewing, Requirements analysis, Data abstraction, Data modeling, Visualization design, Visual story telling, Technical writing, Machine learning, NLP, Data Science, ETL, Data analysis