Deep Learning-Aided Unimodular Quadratic Programming: Initialization and Provable Guarantees

BY

AMRUTHA VARSHINI RAMESH
B.E. Electrical Engineering, Sathyabama University, India, 2011

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2021

Chicago, Illinois

To Vignesh and Adhwaith

ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor and mentor Dr. Mojtaba Soltanalian, whose expertise was invaluable in formulating my research problem and solutions. I thank him for providing a great deal of support and opportunity to further my research, throughout the Masters program, especially during the pandemic.

I thank Dr. Rashid Ansari and Dr. Amit Trivedi for accepting to be a part of my thesis committee.

I thank my husband, Dr. Vignesh Ganapathiraman for his wise counsel, sympathetic ear and being there for me always. I could not have completed this thesis without his unconditional support.

I thank all my family members especially my parents and in-laws. I have always been able to count on them during trying times and they have never failed me. I thank them for the trust they have on me.

I thank my team members and friends for all the stimulating discussions and fun conversations.

Finally, I would like to thank my one year old son Adhwaith Vignesh, for cooperating with my odd schedules while conducting my research, and cheering me up during difficult times.

AR

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER PAGE

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

UQP	Unimodular Quadratic Program
DNN	Deep Neural Network
DUN	Deep Unfolding Network
WSN	Wireless Sensor Network
FC	Fusion Center
SNR	Signal-to-Noise Ratio
ACMA	Analytic Constant Modulus Algorithm
ADMM	Alternating Direction Method of Multipliers
PMLI	Power Method Like Iterations
SDR	Semi Definite Relaxation
SDP	Semi Definite Program
MERIT	Monotonically Error-Bound Improving Technique
ML	Machine learning
SGD	Stochastic Gradient Descent

NOTATIONS

Bold lowercase letters are used to denote the vectors and bold uppercase letters for matrices. The following mathematical notations are used throughout this thesis:

$\lvert x\rvert$	the absolute value of a scalar x
$[x]$	the integral part of a real scalar x, *i.e.*, the greatest integer $\leq x$
$\{x\}$	the fractional part of a real scalar x, *i.e.*, $\{x\} = x - [x]$
$\boldsymbol{x}_m(k)$	the k^{th} element of vector $\boldsymbol{x}_m$
$[\boldsymbol{X}]_{i,j}$	the $(i,j)^{\text{th}}$ element of matrix $\boldsymbol{X}$
$\lVert\boldsymbol{x}\rVert_p$	the l_p-norm of $\boldsymbol{x}$, defined as $(\sum_k \lvert\boldsymbol{x}(k)\rvert^p)^{\frac{1}{p}}$
$\lVert\boldsymbol{x}\rVert$	the l_2-norm of $\boldsymbol{x}$
$\boldsymbol{x} \circledast \boldsymbol{y}$	denotes convolution of $\boldsymbol{x}$ and $\boldsymbol{y}$
$< \boldsymbol{x}, \boldsymbol{y} >$	denotes dot product of $\boldsymbol{x}$ and $\boldsymbol{y}$
$\lVert\boldsymbol{X}\rVert_F$	the Frobenius norm of matrix $\boldsymbol{X}$ defined as $\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} \lvert[\boldsymbol{X}]_{i,j}\rvert^2}$
$\boldsymbol{X}^*$	the complex conjugate of the matrix $\boldsymbol{X}$
$\boldsymbol{X}^T$	the transpose of the matrix $\boldsymbol{X}$
$\boldsymbol{X}^H$	the complex conjugate transpose of the matrix $\boldsymbol{X}$
$\boldsymbol{X}^\dagger$	the Moore-Penrose pseudoinverse of the matrix $\boldsymbol{X}$
$\mathrm{Tr}\,(\boldsymbol{X})$	the trace of matrix $\boldsymbol{X}$

NOTATIONS (Continued)

$\mathrm{vec}\left(\boldsymbol{X}\right)$	the vector obtained by column-wise stacking of matrix $\boldsymbol{X}$
$\mathrm{diag}\left(\boldsymbol{X}\right)$	denotes a vector formed by diagonal entries of the matrix $\boldsymbol{X}$
$\mathrm{Diag}\left(\boldsymbol{x}\right)$	denotes a diagonal matrix formed by the entries of the vector $\boldsymbol{x}$
$\arg\left(\boldsymbol{X}\right)$	the phase angle (in radians) of $\boldsymbol{X}$
$\mathrm{cov}\left(\boldsymbol{X}\right)$	the covariance matrix of $\boldsymbol{X}$
$\Re(\boldsymbol{X})$	the real part of $\boldsymbol{X}$
$\Im(\boldsymbol{X})$	the imaginary part of $\boldsymbol{X}$
$\sigma_n(\boldsymbol{X})$	the n^{th} maximal eigenvalue of $\boldsymbol{X}$
$\lambda_n(\boldsymbol{X})$	the n^{th} maximal singular value of $\boldsymbol{X}$
$\boldsymbol{X} \otimes \boldsymbol{Y}$	the Kronecker product of two matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$
$\boldsymbol{X} \odot \boldsymbol{Y}$	the Hadamard product of two matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$
$\boldsymbol{X} \succ \boldsymbol{Y}$	$\boldsymbol{X} - \boldsymbol{Y}$ is positive definite
$\boldsymbol{X} \succeq \boldsymbol{Y}$	$\boldsymbol{X} - \boldsymbol{Y}$ is positive semidefinite
$\boldsymbol{I}_n$	the identity matrix of dimension n
$\mathbf{1}_n$	the all-one vector of size $n \times 1$
$\mathbf{0}_n$	the all-zero vector of size $n \times 1$
$\boldsymbol{O}$	the matrix with all elements as zero
$\boldsymbol{e}_n$	the n^{th} standard basis of $\mathbb{C}^n$ or n^{th} column of an identity matrix

NOTATIONS (Continued)

$\mathbb{R}$	the set of real numbers
$\mathbb{R}_+$	the sets of real non-negative numbers
$\mathbb{C}$	the set of complex numbers
$\mathbb{N}$	the set of natural numbers
$\mathbb{Z}$	the set of integers
$\mathbb{B}_N^M$	the set of binary vectors with size M and N non-zero elements, $N \leq M$
$\mathcal{S}^M$	the set of all real symmetric matrices of size $M \times M$
$\mathbf{F}_n$	the n dimensional discrete Fourier transform matrix
$\mathbb{E}\left\{\cdot\right\}$	the mathematical expectation of a random variable
$\mathbb{P}\mathrm{r}\left\{\cdot\right\}$	denotes the probability of a random event
$\operatorname{sign}(\cdot)$	the element-wise signum operator
$\operatorname{csign}(\cdot)$	the element-wise complex signum operator as $\operatorname{sign}(\Re\{\cdot\}) + j\operatorname{sign}(\Im\{\cdot\})$
$\mathcal{N}(\cdot,\cdot)$	the normal distribution with mean, and covariance as first and second arguments, respectively
$\ln a$	natural logarithm of a, equivalent to $\log_e a$
j	the imaginary unit *i.e.*, $j = \sqrt{-1}$
$\oplus$	Minkowski's sum of two sets.

SUMMARY

Provably optimal algorithms that are also computationally efficient are becoming increasingly critical for several practical signal processing applications. Though such algorithms are very much in demand, there are only a few prior works that proffer both. This is usually the case because, strong theoretical guarantees come at the cost of increased computational complexity. However, in recent times, owing to the surge in technological advancements and increasing demand for large-scale and real-time signal processing systems, there has been a growing interest in developing faster and reliable algorithms, to cater to this demand.

In this thesis, we study an important optimization problem called "Unimodular Quadratic Program" (UQP) that has shown its presence in prominent applications such as wireless communication, active sensing, etc. UQP is an NP-hard constrained optimization problem and prior works that have proposed approximate solutions have generally suffered from the speed versus reliability trade-off. With the aim of improving the computational efficiency of existing UQP solutions and equipped with the highly scalable deep learning framework as a backbone, we propose two novel solvers for UQP. Our first solution is a black-box computational approach, which we call *Deep-PMLI*, where the deep learning model learns to predict a solution to a given UQP based on already seen example UQPs. Deep-PMLI is an attractive solver for applications that require low-cost solutions but do not require strong guarantees. In our second solution, *Deep-INIT*, we propose a novel data-driven strategy to speed-up an existing solver for UQP with guarantted performance. Deep-INIT, apart from achieving a significant speed-up over the underlying UQP solver, also preserves its guarantees.

CHAPTER 1

INTRODUCTION TO UNIMODULAR QUADRATIC PROGRAM

1.1 Introduction

In recent years, signal processing technologies are becoming more and more commonplace. This surge in usage demands low-cost problem solving algorithms underneath, without compromising on the quality of the solution.

Recently, deep neural network (DNN) (or deep learning) models have been used in a variety of tasks that involve downstream predictions of some nature, including classification, regression, metric learning, representation learning and reinforcement learning. DNN models are generally computationally very efficient and perform well given large amount of training data to learn from. However, deep learning models are usually difficult to interpret, analyse and thus do not provide any theoretical guarantees in its original form. But owing to its high computational efficiency, several signal processing applications have adopted deep learning models.

Recently several works have tried to analyze deep learning models theoretically, under restricted settings. Simultaneously, there have also been works that have made use of deep learning models purely as a computational workhorse, without providing a rigorous theoretical treatment to characterize the quality of its solutions. Finally, hybrid models have been proposed that leverage the computational benefits of deep learning, but are also able to provide theoretical guarantees.

In this work, we study an important optimization problem called Unimodular Quadratic Program (UQP) that has several applications in signal processing. In this thesis, we propose a novel hybrid solver called Deep-INIT, that makes use of deep learning to automatically learn "good" initializations for a theoretically principled underlying solver called MERIT. We also propose a fast black-box solver called Deep-PMLI, which does not provide any guarantees, but is able to enjoy the scalability benefits of being a pure deep learning-based solver.

1.2 Unimodular Quadratic Program

Unimodular signal design plays an important role in improving the overall performance of several signal processing applications such as active sensing, wireless communications, phase retrieval etc. Specifically, unimodular codes are shown to help maximize signal-to-noise ratio (SNR) of active sensing systems while maintaining an optimal (*i.e.*, unity) peak-to-average-power ratio (PAR). Unimodular signals also occur naturally in the problem of phase retrieval, where the aim is to recover a signal $\mathbf{x} \in \mathbb{C}^n$ from the observed magnitude of the same signal $|\mathbf{Ax}|$, $\mathbf{A} := [\mathbf{a}_1, ..., \mathbf{a}_m] \in \mathbb{C}^{m \times n}$ and each $\{\mathbf{a}_j\}_{j=1}^m \in \mathbb{C}^n$. As $| < \mathbf{a}_j, \mathbf{x} > | = | < \mathbf{a}_j, \mathbf{cx} > |$ for any unimodular vector $\mathbf{c}$, recovering signal $\mathbf{x}$ upto a unimodular constant is the best outcome phase retrieval can achieve.

Such a wide usage of unimodular signals leads to several applications of the so-called Unimodular Quadratic Program (UQP). An UQP is often formulated as a maximization of a quadratic form over a set of complex unimodular vectors, more precisely,

$$\max_{\mathbf{s} \in \Omega^n} \mathbf{s}^H \mathbf{R} \boldsymbol{s}, \tag{1.1}$$

where $\mathbf{R} \in \mathbb{C}^{\mathrm{n}\times\mathrm{n}}$ is a given Hermitian matrix, and $\mathbf{s}$ is a complex unimodular vector, with each element lying on the unit circle $\Omega = \{\mathbf{s} : \ |\mathbf{s}| = 1\}$.

Earlier works such as [1,2] show multiple problems from wireless communications and active sensing which can be formulated as UQPs. For instance, in beamforming for WSNs, the computation of phase-only coefficients in phase-shift-and-forward WSNs turns into an UQP [2, 3]. Consider a distributed network where N single antenna sensors observe a signal y_n independently. Any sensor $n \in N$ observes

$$y_n = \theta + u_n, \tag{1.2}$$

where θ is an unknown deterministic complex-valued parameter observed by the sensors, u_n is a complex-valued gaussian noise variable with variance σ_n^2. On each y_n, a unimodular beamforming weight w_n is applied before transmitting it to the Fusion Center (FC) of N antennas. Assuming coherent multiple access protocol [4], FC receives the signal:

$$\mathbf{y} = \mathbf{H}\mathbf{w}\theta + \mathbf{H}\mathbf{D}\mathbf{u} + \mathbf{v}, \tag{1.3}$$

where, $\mathbf{y} = [y_1, \ldots, y_N], \mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_N]$ and each $\mathbf{h}_n \in \mathbb{C}^{N\times 1}$ is the channel vector between the n^{th} sensor and FC, $\mathbf{D}$ is a diagonal matrix of weights $\mathbf{w} = [w_1, \ldots, w_N], \mathbf{u} = [u_1, \ldots, u_N], \mathbf{v} \in \mathbb{C}^{N\times 1}$ is the gaussian noise vector at FC with covariance matrix $\sigma_v^2\mathbf{I}_N$. Here, $\mathbf{I}_N$ is an $N \times N$ identity matrix. $\mathbf{HDu} + \mathbf{v}$ is the overall noise term and it is a gaussian with covariance matrix $\mathbf{HUH}^H + \sigma_v^2\mathbf{I}_N$, where $\mathbf{U} = \mathbb{E}[\mathbf{uu}^H]$ is the diagonal matrix of $[\sigma_{u,1}^2, ...\sigma_{u,n}^2]$. Assuming $\mathbf{H}, \mathbf{U}$ and

σ_v^2 are known, the maximum likelihood (ML) estimate of the parameter θ and its variance [5] are given as,

$$\widehat{\theta}_{ML} = \frac{\mathbf{w}^H \mathbf{H}^H (\mathbf{H}\mathbf{U}\mathbf{H}^H + \sigma_v^2 \mathbf{I}_N)^{-1} \mathbf{y}}{\mathbf{w}^H \mathbf{H}^H (\mathbf{H}\mathbf{U}\mathbf{H}^H + \sigma_v^2 \mathbf{I}_N)^{-1} \mathbf{H}\mathbf{w}} \tag{1.4}$$

$$Var(\widehat{\theta}_{ML}) = [\mathbf{w}^H \mathbf{H}^H (\mathbf{H}\mathbf{U}\mathbf{H}^H + \sigma_v^2 \mathbf{I}_N)^{-1} \mathbf{H}\mathbf{w}]^{-1}. \tag{1.5}$$

Now the beamforming weights $\mathbf{w}$ need to be designed in order to minimize $Var(\widehat{\theta}_{ML})$. This can be formulated as the following optimization problem which becomes an UQP:

$$\max_{\mathbf{w}} \frac{1}{2} \mathbf{w}^H \mathbf{Q} \mathbf{w}, \tag{1.6}$$

$$\text{s.t.} \quad |w_i| = 1, i = 1, \dots, N, \tag{1.7}$$

where $\mathbf{Q}$ is $\mathbf{H}^H (\mathbf{H}\mathbf{U}\mathbf{H}^H + \sigma_v^2 \mathbf{I}_N)^{-1} \mathbf{H}$.

Another notable area that uses UQPs, is in the design of radar codes aiming to improve radar detection performance. Works [1, 6, 7] show that designing radar codes to optimize SNR or the ratio of power at the target location to that of the interferences, becomes an UQP problem. Let us consider a monostatic radar that transmits a linearly encoded burst of pulses. The observed backscattered N-dimensional column vector $\mathbf{v}$ can be expressed as,

$$\mathbf{v} = \alpha(\mathbf{c} \odot \mathbf{p} + \mathbf{w}),$$

where α is a parameter representing both channel propagation and target backscattering effects, $\mathbf{c}$ is a N-dimensional unimodular vector containing the code elements, $\mathbf{p} = [1, e^{j2\pi\nu_d}, ..., e^{j2\pi(N-1)\nu_d}]^T$ is the temporal steering vector; where ν_d is the normalized doppler frequency, $\mathbf{w}$ is a complex-valued vector of noise samples and is assumed to be a zero-mean circular gaussian vector. The positive definite covariance matrix $\mathbf{M} = \mathbb{E}[\mathbf{w}\mathbf{w}^H]$ is known. The SNR is given by [8] as

$$\begin{aligned} \text{SNR} &= |\alpha|^2(\mathbf{c} \odot \mathbf{p}^H)\mathbf{M}^{-1}(\mathbf{c} \odot \mathbf{p}) \\ &= |\alpha|^2\mathbf{c}^H(\mathbf{M}^{-1} \odot (\mathbf{p}\mathbf{p}^H)^*)\mathbf{c} \\ &= |\alpha|^2\mathbf{c}^H\mathbf{R}\mathbf{c}, \end{aligned} \tag{1.8}$$

where, $\mathbf{R} = \mathbf{M}^{-1} \odot (\mathbf{p}\mathbf{p}^H)^*$. Therefore, the problem of designing $\mathbf{c}$ in order to maximize SNR becomes an UQP.

As we discussed in paragraph 1.2, phase retrieval is another significant area where UQP occurs naturally [9–13]. Mathematically, phase retrieval can be formulated as

$$\begin{aligned} &\text{find} \quad \mathbf{x} \\ &\text{s.t.} \quad |\mathbf{A}\mathbf{x} = b|, \end{aligned} \tag{1.9}$$

where $\mathbf{x} \in \mathbb{C}^n$ is the signal to be recovered and $b \in \mathbb{R}$ is the observed magnitude of the signal $\mathbf{x}$. In the noiseless case, (Equation 1.9) can be written as,

$$\min_{\mathbf{x},|\mathbf{u}_i|=1} ||\mathbf{A}\mathbf{x} - \text{Diag}(b)\mathbf{u}||_2^2, \tag{1.10}$$

when $\mathbf{A}\mathbf{x} = \text{Diag}(b)\mathbf{u}$, where $\mathbf{u} \in \mathbb{C}^n$. In (Equation 1.10) we optimize over both $\mathbf{x}$ and $\mathbf{u}$. Here, $\mathbf{x}$ admits a closed-form solution $\mathbf{x} = \mathbf{A}^H\text{Diag}(b)\mathbf{u}$. Now (Equation 1.10) can be reduced to,

$$\begin{aligned} &\min_{|\mathbf{u}_i|=1} \quad ||\mathbf{A}\mathbf{A}^{\mathbf{H}}\text{Diag}(\mathbf{b})\mathbf{u} - \text{Diag}(b)\mathbf{u}||_2^2 \\ = &\min_{|\mathbf{u}_i|=1} \quad ||(\mathbf{A}\mathbf{A}^{\mathbf{H}} - \mathbf{I})\text{Diag}(\mathbf{b})\mathbf{u}||_2^2. \end{aligned} \tag{1.11}$$

By assigning $\widetilde{M} = (\mathbf{A}\mathbf{A}^H - \mathbf{I})^H\ (\mathbf{A}\mathbf{A}^H - \mathbf{I})$, (Equation 1.11) becomes,

$$\min_{|\mathbf{u}_i|=1} \quad \mathbf{u}^H\text{Diag}(b^T)\widetilde{\mathbf{M}}\text{Diag}(b)\mathbf{u}.$$

Finally, phase retrieval becomes an UQP by

$$\begin{aligned} \min \quad & \mathbf{u}^H\mathbf{M}\mathbf{u} \\ \text{s.t.} \quad & |\mathbf{u}_i| = 1, i = 1, \ldots, n, \end{aligned}$$

where $\mathbf{M} = \text{Diag}(b)(\mathbf{I} - \mathbf{A}\mathbf{A}^H)\text{Diag}(b)$ is a positive semi-definite Hermition matrix.

1.3 Speed vs Accuracy

In recent times, there have been a plethora of practical signal processing applications, whose mathematical machinery incorporates sub-problems that transform itself as UQPs, including the above mentioned examples. Most often, these applications demand both high-speed and accuracy at the same. For example, consider civilian radar systems such as autonomous vehicles, which are expected to operate in a highly dynamic environment. These systems face intricate interactions with complex information that are natural to the real world [14]. Yet, the high (obstacle) detection quality of such radar systems is of paramount importance, as they directly impact the safety of the vehicles [14]. In (Equation 1.8), we showed the importance of solving UQP to achieve high SNR. This directly means that, accuracy of the UQP solution in transmit waveform design of such radar systems directly impact the safety of autonomous vehicles. At the same time, these radar systems must also detect the obstacles fast, as certain time sensitive modules like AEB (automatic electronic braking) rely on the obstacle detection of these radar systems. In a nutshell, there are practical applications of UQP that demand the underlying solvers to be both accurate and fast at the same time. We will soon see that this is a trade-off that is challenging to deal with.

1.4 Contributions of this thesis

The key contributions of this thesis work are presented in four chapters. The first and last chapters serve as the introduction and conclusion respectively.

1. In the **second chapter**, we rigorously survey existing prior model-based methods for approximately solving UQPs
2. In the **third chapter**, we review some fundamental ideas behind an important black-box computational model called deep learning. In that, we discuss the strengths and limitations of deep learning and end the section by introducing hybrid computational models that combine model-based algorithms and deep learning for well-known signal processing applications.
3. We present our first solver called *Deep-PMLI* - a black-box solver for UQP, in **chapter 4**.
4. In **chapter 5**, we introduce our novel hybrid model for UQPs called *Deep-INIT* that uses the power of deep learning to automatically find better initializations for a well-known model based solver for UQP.

CHAPTER 2

PRIOR WORKS ON UQP

2.1 Introduction:

It is a well known fact that UQPs are NP-hard in general [15]. The NP-hardness of the problem mainly arises due to the unimodular constraint and can be proved by a reduction from a well-known NP-complete matrix partitioning problem. In effort to provide both provably optimal and computationally efficient solvers, recently several approximation methods have been studied for solving UQPs. Some of these methods carefully construct reformulations of the UQP optimization problem (Equation 1.1) in such a way that the resulting approximate solutions are *guaranteed to be close* to the true UQP solution. Other methods focus on cost efficiency by proposing computational approaches that are faster, but do not provide guarantees. In this chapter, we will review some of these existing solvers that approximate solutions to UQP.

2.2 Semidefinite relaxation (SDR)

One of the popular methods for approximating UQP solutions is semidefinite relaxation (e.g. [7,15–24]). To begin with, the canonical UQP optimization problem (Equation 1.1) can be rewritten as:

$$\begin{aligned} \max_{\mathbf{S}} \quad & \operatorname{Tr}(\mathbf{RS})\,; \\ \text{s.t.} \quad & \mathbf{S} = \boldsymbol{s}\boldsymbol{s}^H,\ \boldsymbol{s} \in \Omega^n. \end{aligned} \tag{2.1}$$

Note that the above reformulation is possible since $\boldsymbol{s}^H \mathbf{R} \boldsymbol{s} = \operatorname{Tr}\left(\boldsymbol{s}^H \mathbf{R} \boldsymbol{s}\right) = \operatorname{Tr}\left(\mathbf{R} \boldsymbol{s} \boldsymbol{s}^H\right)$. The arising problem is non-convex due to the rank constraint on the matrix variable $\mathbf{S}$. Relaxing the rank constraint gives us the following semidefinite program (SDP):

$$\begin{aligned} &\max_{\mathbf{S}} \operatorname{Tr}\left(\mathbf{RS}\right) \\ &\text{s.t. } [\mathbf{S}]_{k,k} = 1,\ k = 1 \ldots N; \quad \mathbf{S} \succeq \mathbf{0} \end{aligned} \tag{2.2}$$

Due to this relaxation, the optimization process can return higher rank solutions, which may make the optimal solution elusive. Fortunately, interior point methods can be used to solve the SDP formulation of UQP in polynomial time, thanks to induced convexity [21, 25, 26]. After solving the above SDP, the approximate UQP solutions can be recovered using several approximation techniques [7,15,22], which are shown to have a sub-optimality guarantee of $\pi/4$, where sub-optimality is measured by

$$\gamma = \frac{v_{SDR}}{v_{opt}}. \tag{2.3}$$

γ is the sub-optimality coefficient, v_{SDR} and v_{opt} are objective values at the obtained solution from SDR and at the optimal value. However, the computational complexity and the large memory requirements of interior point methods restrict the scalability of the SDR method.

2.3 Power method like iterations (PMLI)

The authors of [1, 27] proposed an iterative algorithm called *power method like iterations* (PMLI) to approximate the solution to UQP. With every progressing iteration until convergence to a local optimum, PMLI aims to increase the objective value of (Equation 1.1) by solving the following nearest-vector problem [14].

$$\begin{aligned} \min_{\mathbf{s}^{(n+1)}} \quad & ||\mathbf{s}^{(n+1)} - \mathbf{R}\mathbf{s}^{(n)}||_2 \\ \text{s.t.} \quad & |s_k^{(n+1)}| = 1 \quad \forall k. \end{aligned} \tag{2.4}$$

An analytical solution for the above (Equation 2.4) is provided by [1, 27], where

$$\mathbf{s}^{(n+1)} = e^{j \arg(\mathbf{R}\mathbf{s}^{(n)})}, \tag{2.5}$$

where $\mathbf{s}^{(0)}$ is initialized with any unimodular vector. Here, $\mathbf{s}^{(i)}$ is the value of $\mathbf{s}$ at iteration i.

In practice, PMLI is generally faster than other popular UQP solvers. The cost efficiency of PMLI comes from the fact that the analytical solution, as in (Equation 2.5), to the optimization problem in (Equation 2.4), clearly encodes the unimodularity constraint of $\mathbf{s}$ directly in computation, unlike other methods that rely on slower and high-cost methods like projection or restriction of search space. But the quality of solutions that this iterative algorithm converges to, is primarily dependent on the initialization of $\boldsymbol{s}^{(0)}$ [1]. That is, for two different initializations of $\boldsymbol{s}^{(0)}$, $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, PMLI is likely to converge to different local solutions and it is possible that one solution is better than the other. In general, it is possible that there exists initializations that can lead PMLI to global optima, but, as the characterization of such initializations are unknown, no guarantees can be provided about the solution of PMLI. This makes PMLI an attractive low-cost, but less reliable approach to solve UQPs.

2.4 MERIT

The work [1] provided an iterative solution to UQPs based on an algorithm called "MERIT: A Monotonically Error-Bound Improving Technique for Mathematical Optimization", leading to tighter sub-optimality guarantees than SDR. MERIT aims to successively approximate (and get closer to) the original UQP problem instance and its global optimum of interest via constructing a sequence of problem instances whose corresponding global optima are readily known. Notably, the authors of [1] were able show that the MERIT solution for UQP enjoys data-dependent sub-optimality guarantees, which in most cases are better than the $\pi/4$ guarantee provided by SDR.

The central idea of MERIT is as follows. Let $\mathcal{P}(v,x)$ be an optimization problem *structure* with *given* (a.k.a. *problem instance*) and *optimization* variables partitioned as (v,x). For example:

$$\begin{aligned} \mathbf{X} = \arg\max\ \mathrm{tr}(\mathbf{RX}) \\ \text{s.t. } \mathrm{tr}(\mathbf{QX}) \leq t \end{aligned} \quad \xrightarrow{\text{variable partitioning}} \quad \begin{aligned} \mathbf{R}, \mathbf{Q}, t \to v \\ \mathbf{X} \to x \end{aligned}$$

Now suppose $\mathcal{P}(v,x)$ is a *difficult* (possibly NP-hard) optimization problem for which an approximate solution is being sought; and that the following viable conditions are met:

- A sequence $v_0, v_1, v_2, \cdots$ of v can be constructed such that (i) the associated global optima of their associated sub-problems, viz. $x_k = \arg\max_x \mathcal{P}(v_k, x)$, are known for any v_k, and that (ii) the *distance* $\mathcal{D}(v, v_k)$ between v and v_k is decreasing with k.

- A suboptimality guarantee γ_k of the obtained solutions $\{x_k\}$ can be efficiently computed using the known distance between v and $\{v_k\}$; i.e., there exists an easily computable

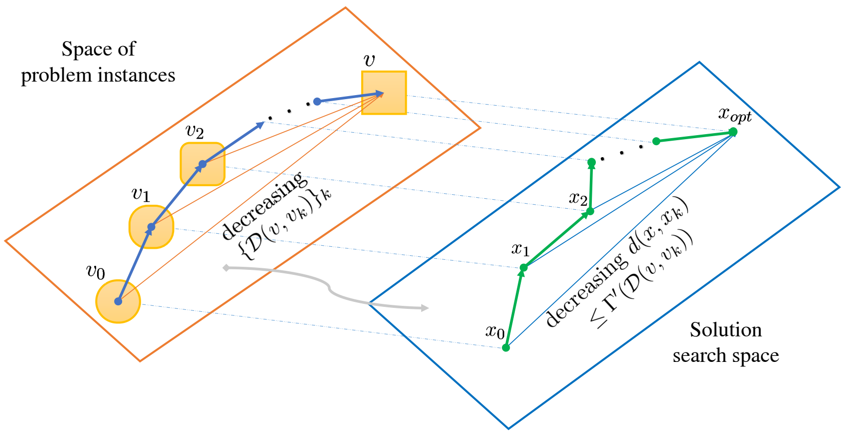


Figure 1: MERIT simultaneously approaching the problem of interest and its global optimum. The guarantees are produced by acquiring proximity information from the space of problem instances.

function $\Gamma(.)$ such that $\gamma_k = \Gamma(\mathcal{D}(v, v_k))$. This condition may also be interpreted as establishing an upper bound on the distance $d(x_{opt}, x_k)$ between $\{x_k\}$ and the global optimum of the problem denoted as x_{opt}, namely $d(x_{opt}, x_k) \leq \Gamma'(\mathcal{D}(v, v_k))$.

As a result, one can derive *computational/data-aided suboptimality guarantees* along with the approximate solutions $\{x_k\}$ that might:

- Outperform the existing analytically derived suboptimality guarantees, or
- Be the only reliable class of suboptimality guarantees in cases where no non-trivial *a priori* known guarantees are available for the given problem.

An intuitive illustration of the proposed idea is presented in Fig. Figure 1. Next, we will discuss in more detail how [1] applied the MERIT framework to UQPs.

2.4.1 MERIT for UQPs

As described in the previous section, careful construction of sub-problems for which the global optima are known is a key step in the MERIT framework. The sub-problem design is enabled by the following equivalence characterization in the input matrix $\mathbf{R}$ as observed by [1]. In this section, we will briefly describe this characterization and end with the details of the sub-problem construction and a full summary of the MERIT algorithm for UQP.

2.4.1.1 Equivalence characterization for UQP problem instances

The work [1] first derives an explicit characterization of the input matrix $\mathbf{R}$ with given UQP solutions. Let $\mathcal{K}(\mathbf{s})$ represent the set of all matrices $\mathbf{R}$ for which a unimodular vector $\mathbf{s}$ is the solution to the UQP. It turns out that $\mathcal{K}(\mathbf{s})$ is a convex cone and can be characterized by the following mapping. If $\mathbf{s}_1, \mathbf{s}_2$ are two unimodular vectors and $\mathbf{R} \in \mathcal{K}(\mathbf{s}_1)$, then it holds that

$$\mathbf{R} \in \mathcal{K}(\mathbf{s}_1) \Leftrightarrow \mathbf{R} \odot \mathbf{s}_0\mathbf{s}_0^H \in \mathcal{K}(\mathbf{s}_2), \text{ where } \mathbf{s}_0 = \mathbf{s}_1^* \odot \mathbf{s}_2 \tag{2.6}$$

In particular the above mapping Equation 2.6 implies

$$\mathbf{R} \in \mathcal{K}(\mathbf{1}) \Leftrightarrow \mathbf{R} \odot \mathbf{s}\mathbf{s}^H \in \mathcal{K}(\mathbf{s}) \tag{2.7}$$

for any unimodular vector $\mathbf{s}$. Additionally, [1] proposes a computational recipe to approximate $\mathcal{K}(\mathbf{s})$ by the Minowski sum of two particular convex cones. For the sake of completeness, we present the corresponding theorem from [1] below.

Theorem 1. *For any given* $\boldsymbol{s} = (e^{j\phi_i}, \ldots, e^{j\Phi_n} \in \Omega^n$*, let* $\{\mathbf{B}_{k,l}\}$ *be a set of matrices defined as*

$$\mathbf{B}_{k,l} = (\mathbf{e}_k\mathbf{e}_l^H + \mathbf{e}_l\mathbf{e}_k^H) \odot (\boldsymbol{s}\boldsymbol{s}^H)$$

and $V_s = \{\mathbf{B}_{k,l} : 1 \leq k \leq k \leq n\} \cup \{-\mathbf{I}_n\}$*. Let* $\mathcal{C}(V_s)$ *represent the convex cone associated with the basis matrices in* $\mathbf{V_s}$*. Also let* C_s *represent the convex cone of matrices with* $\boldsymbol{s}$ *being their dominant eigenvector. Then for any* $\mathbf{R} \in \mathcal{K}(s)$*, there exists* $\alpha_0 \geq 0$ *such that for all* $\alpha \geq \alpha_0$*,*

$$\mathbf{R} + \alpha\boldsymbol{s}\boldsymbol{s}^H \in \mathcal{C}(V_{\boldsymbol{s}}) \oplus C_{\boldsymbol{s}}$$

For any $\tilde{\mathbf{s}} \in L$, where L represents set of all local optima and saddle points of UQP and $arg(\tilde{\mathbf{s}}) = arg(\mathbf{R}\tilde{\mathbf{s}})$, then $\tilde{\mathbf{s}}$ is referred to as an hyperpoint of UQP. As a direct consequence of Theorem 1, [1] shows that if $\mathbf{s}$ is a hyper point of the UQP corresponding to $\mathbf{R}$ then the following equality holds

$$\mathbf{R} + \alpha_0\mathbf{s}\mathbf{s}^H = (\mathbf{Q}_1 + \mathbf{P}_1) \odot (\mathbf{s}\mathbf{s}^H), \tag{2.8}$$

where $\mathbf{Q}_1 \in \mathcal{C}_1$ and $\mathbf{P}_1 \in \mathcal{C}(V_1)$.

It can be easily verified that $\mathbf{s}$ is the UQP solution for all matrices in $\mathcal{C}_{\mathbf{s}}$ and $\mathcal{C}(V_{\mathbf{s}})$. Furthermore, the vector $\mathbf{s}$ that satisfies (Equation 2.8) is guaranteed to be the global optimum of the UQP associated with $\mathbf{R}$ when $\alpha_0 = 0$, and a local optima when $\alpha_0 > 0$.

When $\alpha_0 = 0$, we get the following optimization problem:

$$\min_{\mathbf{s}\in\Omega^n,\mathbf{Q}_1\in\mathcal{C}_1,\mathbf{P}_1\in\mathcal{C}(V1)} ||\mathbf{R} - \underbrace{(\mathbf{Q}_1 + \mathbf{P}_1) \odot \mathbf{s}\mathbf{s}^H}_{\mathbf{R}_\mathbf{s}}||_F. \tag{2.9}$$

In the case of $\alpha_0 > 0$, [1] then shows that the local optimum of (Equation 2.8) can be found iteratively by defining $\mathbf{R}' = \mathbf{R} + \alpha_0\mathbf{s}\mathbf{s}^H$ for increasing values of α_0 (whose best value is found using a bisection algorithm) and minimizing the objective (Equation 2.9) while using $\mathbf{R}'$ instead of $\mathbf{R}$

2.4.1.2 Sub-problem construction

The MERIT algorithm, as given in [1] solves (Equation 2.9) by first performing the optimization processes with respect to $\mathbf{Q}_1$ and $\mathbf{P}_1$, each of which is convex. The optimization with respect to $\mathbf{s}$ can be done by using power method-like iterations. Note that, (Equation 2.9) ensures that the constructed sub-problems always get closer to original problem as characterized by (Equation 2.8).

2.5 Other notable methods

In this section we would like to briefly review some other notable UQP solvers, for the interested reader. [28] proposed three heuristic methods - dominant eigen vector matching, greedy strategy and row-swap greedy strategy to approximate the UQP solution that can be solved in polynomial-time with respect to the problem size. In the dominant eigen vector matching method, the authors of [28] observed that the objection function

$$\max_{|s_i|=1 \quad i=1,\dots,N} \quad \mathbf{s}^H\mathbf{R}\mathbf{s},$$

where $\mathbf{R} \in \mathbb{C}^{N\times N}$ is a given Hermition matrix, is maximum for any vector $\mathbf{s}$ that has $|t(N)|^2 = N$ and $|t(i)| = 0 \ \forall i < N$. Here, $t(i)$ is the i^{th} element of $\mathbf{U}^H\mathbf{s}$, $\mathbf{U}$ being a unitary matrix with eigen vectors of $\mathbf{R}$ as its columns. Therefore, the authors of [28] propose an algorithm to pick a unimodular vector $\mathbf{s}$ that maximizes $t(N)$. In their other two greedy algorithms, [28] maximizes a partitioned representation of the objective function and optimizes for the unimodular sequence element-wise. Under a restriction of string-submodularity of the objective function for a given $\mathbf{R}$, they are able to provide a performance guarantee.

The authors of [2] proposed an ADMM (alternating direction method of multipliers) based solution for UQP and empirically showed that their proposed approach performs almost identical to the SDR-based approach with much fewer computations. They do not provide any optimality guarantees of their solution.

Another approach to solve UQP is an ACMA (analytic constant modulus algorithm) [29] based technique [2, 3]. ACMA-based solutions may initially appear as an attractive low-cost approach. However apart from not providing guarantees, they can also perform poorly, especially with growing size of $\mathbf{R}$.

2.6 Cost vs reliability trade-off

Considering the wide spectrum of different approaches to solve UQPs, the ones that provide strong optimality guarantees and tighter bounds tend to have high computational cost than the ones that provide little or no theoretical results. For instance, the computational complexity of SDR based methods are generally high - $O(N^7)$ floating point operations [2]. This is because the original problem dimensions are squared to cast into SDR form. On the other hand, low-cost solutions such as ACMA provide approximate solutions to UQPs in $O(lN^2)$ time, where l is the related subspace dimension parameter [2, 3]. ACMA-based approaches [3] are computationally efficient because of the considerably simpler closed-forms on which the methods are built on. But the approximation quality of ACMA is generally poor and no guarantees on the solutions are provided.

But as discussed in Chapter 1, there is a strong need for algorithms to be fast and computationally efficient while being more reliable. One way to achieve this goal is to speed-up existing methods that provide strong guarantees, while also preserving or tightening the guarantees wherever possible.

CHAPTER 3

BLACK-BOX COMPUTATIONAL SOLUTIONS FOR COMMON SIGNAL PROCESSING PROBLEMS

3.1 Introduction

In earlier chapters, we highlighted the fact that there exists practical applications of UQP where accuracy (theoretical correctness guarantees) and speed are imperative. Akin to this, there are several other signal processing applications where this trade-off is quintessential. Researchers have recently turned to black-box computational approaches based on deep neural networks (sometimes called deep learning) to approximate existing computational solutions to these problems, either in parts or whole.

In this section, we will layout the basics of deep learning-based modeling and motivate how they can be used as a computational tool for speeding up expensive estimation problems. We will also discuss some fundamental limitations of deep learning.

Machine learning (ML) systems are typically designed to automatically learn patterns in the input, which gives them the ability to predict outcomes from previously "unseen" data. Specifically, given training data $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n);\ \mathbf{x}_i \in \mathcal{X};\ \mathbf{y}_i \in \mathcal{Y}\}$, where $\mathcal{X}$ is the set of input training points and $\mathcal{Y}$ is the set of labels, an ML model learns a function $f : \mathcal{X} \rightarrow \mathcal{Y}$, that captures the relationship between $\mathbf{x}_i$ and y_i. For example, in the image classification task, $\mathcal{X}$ could be a set of images and $\mathcal{Y}$ could be a set of object names and the ML algorithm learns to classify an object in an image, having seen many examples of similar images. Such ML models that learn in the presence of class labels are called **supervised learning models**. Commonly, supervised learning models learn the function f by formulating the learning task as a mathematical optimization problem, that minimizes a **loss** function over the input samples, via the so-called **Empirical Risk Minimization (ERM) principle** [30]. Here, given an input $\mathbf{x}\in\mathcal{X}$ and output $y\in\mathcal{Y}$, drawn from an unknown joint probability distribution D, the goal is to "learn" a **hypothesis (predictor)** $\mathbf{h} : \mathcal{X}\to\mathcal{Y}$ from a hypothesis space $\mathcal{H}$, that minimizes the following *population risk*.

$$R(\mathbf{h}) = \mathbb{E}_{(\mathbf{x},y)\sim D}\ell(\mathbf{h}(\mathbf{x}), y), \tag{3.1}$$

where $\ell(\mathbf{h}(\mathbf{x}), y)$ is a chosen loss function that measures the amount of deviation between the output predicted by the hypothesis $\mathbf{h}$ and the true output label y. The best hypothesis is the one that minimizes the above risk.

$$\mathbf{h}^* = \text{argmin}_{\mathbf{h}\in\mathcal{H}} R(\mathbf{h}). \tag{3.2}$$

(Equation 3.1) represents the so-called *population risk* and (Equation 3.2) means that we need to find a hypothesis that minimizes the expected risk over all $(\mathbf{x}, y)$ drawn from D. Unfortunately, this is an impractical optimization problem, since we do not know the distribution D apriori.

Therefore, it is common to solve the empirical version of the above problem, where we minimize the loss function ℓ over the given n training samples $S{\sim}D^n$ and $|S| = n$ as,

$$\hat{\mathbf{h}}^* = \mathrm{argmin}_{\mathbf{h}\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{h}(\mathbf{x}), y_i). \tag{3.3}$$

Finding $\hat{\mathbf{h}}^*$ amounts to finding (approximately) the $\mathbf{h} \in \mathcal{H}$, which is often parameterized by a vector of parameters $\mathbf{w}$, that minimizes the training loss or *training error*. In other words, we want a hypothesis that makes the minimum amount of mistakes in predicting the output $\mathbf{h}(\mathbf{x})$. Mathematically, this can be achieved by merely memorizing the patterns between $\mathbf{x}$ and y in the given input training data. Usually, it is more interesting and useful in practice, to find a hypothesis that makes fewer mistakes on previous unseen "test input", often referred to as the *test error*. Conceptually, reducing the propensity of the ML model to memorize patterns in the training data, leads to better generalization performance (reduction in generalization error). This is commonly achieved by imposing restrictions on the class of functions from which the hypothesis $\mathbf{h}$ can be chosen. One way to create this restriction is to bound the norm of the parameters $\mathbf{w}$ of the hypothesis $\mathbf{h}$. Such a restriction expresses the intention that we are only interested in the restricted class $\mathcal{H} = \{\mathbf{h}_{\mathbf{w}} : |\mathbf{w}| \leq \lambda; \lambda \geq 0\}$. The new risk minimization, often called *structural risk minimization (SRM)* [30] can be written as,

$$\hat{R}(\mathbf{h}) = \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{h}(\mathbf{x}_i), y) + \lambda_i|\mathbf{w}_i|.$$

Depending on the initial hypothesis class $\mathcal{H}$, the loss function ℓ and the task at hand, there are multiple ways to restrict $\mathcal{H}$. While, we want our hypothesis class to be as small as possible in order to get good generalization performance, we also want $\mathcal{H}$ to be sufficiently rich, so that the "best" hypothesis $\mathbf{h} \in \mathcal{H}$ is able to capture non-trivial patterns in the input. Usually it's hard to come up with a general theory on how to choose such a $\mathcal{H}$.

3.2 Optimization for training machine learning models

In the previous section, we introduced some general principles on how to model a predictor for a supervised ML problem. We will now discuss algorithms, that will help us learn a predictor, based on these learning principles (ERM and SRM).

Given a set of a predictors $\{f_{\mathbf{w}}\}$, with parameters $\mathbf{w}$ from a function class $\mathcal{F}$, we are interested in learning the parameters $\mathbf{w}$ that minimizes the empirical risk (Equation 3.3) [1]. Assuming $f_{\mathbf{w}}$ is differentiable, then the risk (Equation 3.3) is differentiable. When the function f and its gradients $\nabla_{\mathbf{w}} f$ can be efficiently computed, a popular optimization algorithm called *Gradient Descent* can be employed. This algorithm iteratively updates the parameters of the model, by taking a "step" opposite to the direction of steepest increase (gradient), in each iteration. The update rule of Gradient Descent is given by,

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla_{\mathbf{w}} f(\mathbf{w}_t),$$

[1]Please note the change of notation from the previous section.

where, $\nabla_{\mathbf{w}} f(\mathbf{x})$ is the gradient of the function f at $\mathbf{x}$ and η is a hyper parameter called *learning rate*, which can be tuned. If f is convex, then the convergence rate of gradient descent is bounded by,

$$|f(\mathbf{w}_t) - f(\mathbf{w}^\star)| \leq \frac{|\mathbf{w}_0 - \mathbf{w}^\star|^2}{2t\eta},$$

where t is the number of iterations.

Note that the gradient of f is calculated at the input $\mathbf{x}$, which is the entire dataset. When the input dataset is huge, its often time consuming / impractical (in some cases) to compute the *full gradient*. A slightly modified version of gradient descent, called the *stochastic gradient descent* (SGD) [31] computes the gradient of f at a point $\mathbf{x}_i$, randomly chosen from the input data. Another version called the *mini-batch stochastic gradient descent*, computes the gradient of a batch of input data. In recent years, with massive amounts of training data being used by ML models, SGD has become one of the go-to algorithms for ML practitioners. Since the advent of gradient descent and stochastic gradient descent, numerous other gradient based algorithms have been proposed. For instance, momementum based methods [32] use past gradients to explore directions, other than the direction opposite to the gradient, that minimize the objective function more consistently. The update rule for momentum is given as,

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_{\mathbf{w}} f(\mathbf{w}) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - v_t, \end{aligned} \tag{3.4}$$

where v_t is called the velocity vector.

All the methods mentioned above use gradient information to compute the weight updates in every iteration. They are sometimes also called "first-order" methods. In the same regard, there are methods that use the second order derivative (hessian) in their weight update equations. These are analogously called "second-order" optimization methods. Some examples of second order method include Newton method, LBFGS etc. Most second order methods also use gradient information, in addition to the hessian in their optimization routine.

3.2.1 Characterizing solutions of optimization

The training procedure, using any of the gradient based methods, stops when the optimization algorithm reaches the so-called *critical point*. Critical points of a function can be characterized by a vanishing gradient *i.e.* $f'(\mathbf{x}) = 0$. The question is whether the reached critical point corresponds to the best solution to our optimization problem. A critical point of a function could further represent one of the following points in a function.

Local Minima. A local minima is a point where the objective function $f(\mathbf{x})$ is lower (similarly higher for maximization problems) than all its immediate neighbors.

Global Minima. A global minima is a point where the objective function is the lowest *i.e.* $f(\mathbf{x}) \leq f(\mathbf{x}'),\ \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}; \mathbf{x} \neq \mathbf{x}'$.

Saddle points. Saddle points are critical points where the function has both a local minimum and a local maximum. Note that since saddle points are critical points, the gradient $f'(\mathbf{x})$ will be zero at a saddle point. Therefore for gradient based optimization algorithms such as SGD, saddle points are indistinguishable from local minimas.

In general, it is always desirable to find **global solutions** (global optimal point) to an optimization problem (also true in optimization for machine learning). However, it is computationally hard (NP-complete) to find a global minimum of even a “well-behaved” function. Therefore all we can hope to find is a local solution to the problem. Indeed, most gradient based methods converge to a local minima, (nowadays despite the presence of saddle points in the objective). However, local solutions are almost always inferior to global solutions to the problem.

Interestingly, if the optimization problem is known to be **convex**, then it is guaranteed (in theory) that the local optima returned by the algorithm is indeed the globally optimal. **Non-convex** methods on the other hand have multiple local minimums. However, obtaining global solutions to non-convex optimization problems are hard in general.

3.3 Poor training guarantees and model interpretability

In the previous few sections we saw how a typical deep learning model is trained. By characterizing the solutions to the optimization problem associated with deep learning training, we noted that the solutions are only guaranteed to be locally optimal. Also, as deep learning training is hard to analyze from the perspective of optimality (given a local optima, it is hard to computationally verify its quality), the patterns that these models learn are generally hard to interpret. Specifically, it is hard to ascertain mathematically what the learned $\mathbf{w}$ represents with respect to the original learning problem. This view is crucial in understanding when and where the black-box computational solutions provided by deep learning models are useful.

Because of the hardness associated with interpretability of deep learning models, practitioners often use deep learning as a black-box computational tool for estimating quantities of interest. In contrast, most of the algorithms introduced for approximating UQPs in the previous Section 2 are designed specifically for UQPs and incorporate aspects of the structure of the UQP problem directly in its problem specification or in its solution. For the rest of the thesis, we call the algorithms reviewed in Section 2 as **model-based methods** to contrast them from black-box deep learning methods.

3.4 Overfitting in deep learning training

Recall how deep learning models typically operate in two phases - training and testing. Just to recap: during (supervised) training, deep learning models are shown labelled training data and the weights of the model are optimized from the data using optimization algorithms such as SGD. After training, at the testing or *inference* phase, the model is asked to make predictions on unlabelled and typically unseen test data[1]. Overfitting occurs when a model is trained "well" on the labelled training data but performs poorly on test data. All illustration of the overfitting phenomenon is show in Figure 2. Here as training progresses, the training loss goes down and approaches 0. This is indicative of the model being well-trained on the training data. Initially, the model's predictions on the unseen test data are also quite accurate, as suggested by the decreasing test loss. After a few training iterations the training error continues to decreases,

[1]by unseen data we mean data samples that were not used during training

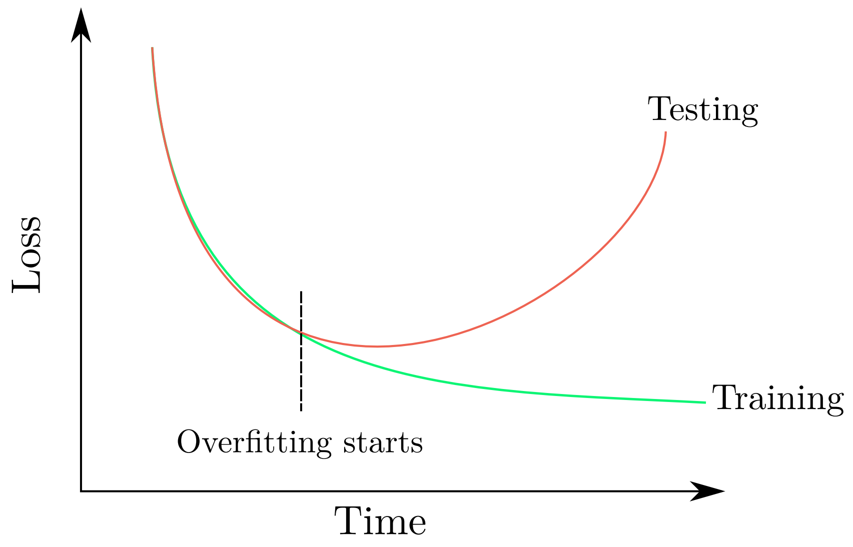


Figure 2: Illustration of the overfitting phenomenon in deep learning

however the testing loss begins to increase. This inflection point (marked by a dotted straight line parallel to the y-axis) is where overfitting starts to occur.

The phenomenon of overfitting and its effects on test loss or corresponding test-time prediction accuracy is even more pronounced when the training and test distributions differ significantly. Going back to the image classification example, such a phenomenon might occur when we train a deep learning model on a training data containing only cat images and then using the model to make predictions on a test set of dog images. When the model in question is trained to overfit on cat images, it is likely to perform poorly on the task of predicting dog images. This problem is sometimes called the *data shift* or *covariate shift* problem.

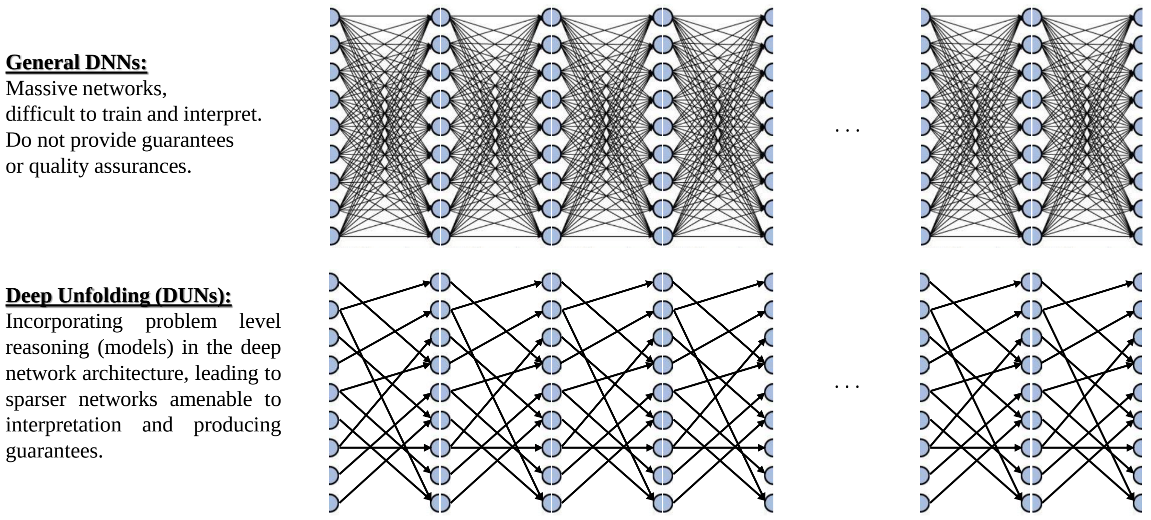
3.5 Speed vs. accuracy trade-off for black-box models

As noted in an earlier section in the chapter, several practical signal processing applications have recently adopted deep learning based learning models to develop approximate solutions to some of the hard computational problems. An attractive benefit of deep learning models is indeed its speed. Even a sufficiently “complex” deep learning model after training, can make predictions much faster than some of the model-based solutions for the same problem, in some cases by a significant order of magnitude. However, because of the black-box nature of these models, in addition to their fundamental limitations such as lack of interpretability and over-fitting, they are generally devoid of the theoretical advantages that model-based methods often provide. Also, deep learning models typically require large volumes of *labeled training data* in order to work well. This requirement is exacerbated by the inherent overfitting problem in deep networks. Having said that, the speed and scalability benefits of deep learning models make it an attractive computational solution. With the invention of modern computational hardware, the scalability advantages of deep learning have only been reinforced and well optimized for large scale computational problems. Adopting deep learning based solvers and computational oracles have thus been a fervent venture of researchers in several scientific disciplines.

The question is then, how do we get the best of accurate model-based solutions and scalable deep learning based models? In the next section, we will explore hybrid models that do exactly that. Then based on those ideas, we will begin to detail a new hybrid model for approximately solving the UQP problem.

3.6 Hybrid computational models using deep unfolding

Aside from developing new model-based methods that are faster, the scientific community has been recently looking at data-driven approaches to simply speedup existing model-based methods, based on the deep unfolding framework among others [33]. The key idea behind deep unfolding is to “unfold” the iterations of an inference algorithm and restructure them as layers of a deep neural network. The parameters of the deep network are then learned using training data in a suitably designed supervised learning framework. Note that this is different from using conventional deep neural networks as black-box estimators (see Figure 3 that contrasts a conventional feed-forward neural network with that of a deep-unfolding based network). The layers of a deep unfolding based DNN (abbreviated as DUN) closely follow the underlying model-based algorithm, thereby preserving the rich problem structure information that the model unveils. Several model-based methods have benefited from such a learning-based estimation; see e.g ., [34–37]. Researchers have exhibited that the resulting deep architectures are able to achieve a performance similar to their model-based counterparts in far fewer number of iterations (layers) after the training stage.



General DNNs:
Massive networks,
difficult to train and interpret.
Do not provide guarantees
or quality assurances.

Deep Unfolding (DUNs):
Incorporating problem level reasoning (models) in the deep network architecture, leading to sparser networks amenable to interpretation and producing guarantees.

Figure 3: Conventional DNNs vs DUNs. Encoding problem-specific information inside the layers of a DUN minimizes the number of required trainable parameters compared to a conventional DNN.

CHAPTER 4

PROPOSED METHOD - DEEP-PMLI

As a first step towards developing a computationally efficient algorithm for finding approximate UQP solutions, we developed a black-box solver based on deep neural networks that predicts a unimodular vector $\mathbf{s}$ for a given UQP. We call this proposed model **Deep-PMLI**.

In this chapter, we first discuss some of the challenges in developing a deep learning based solver for UQPs. We then address the challenges and then discuss in detail, the model architecture and training details. Through our detailed experiments we showcase the benefits and shortcomings of using a black-box solver for UQPs.

4.1 Challenges in developing deep learning model for UQP

While developing a deep learning based solver for UQP, we were immediately confronted with a few fundamental challenges that made the task less trivial.

1. **No benchmark datasets.** Deep learning-based solvers require vast amounts of labeled training data. Deep learning-based solvers have not been attempted for UQPs before this work, to the best of our knowledge. As a result, there aren't any existing benchmark datasets for UQP. As another novel contribution to this work, we have developed a principled dataset generation algorithm in Section 4.2 to train our model.

2. **Enforcing unimodularity constraint.** As UQPs are fundamentally a constrained optimization problem, with an unimodularity constraint on the optimization variable $\mathbf{s}$, we

had to figure out a way to enforce the unimodularity constraint within the deep learning architecture. To overcome this difficulty, we have introduced a new activation function in Section 4.3 that embeds the constraint within the model.

4.2 Data generation for training deep learning models for UQPs

As we mentioned in the previous section, deep learning-based solvers for optimizing UQPs have not been explored prior to this work and therefore labelled data for training deep learning models isn't readily available. Note that the observed input for a given UQP is the positive semi-definite matrix $\mathbf{R}$ and the output is a unimodular $\mathbf{s}$ that maximizes the UQP objective. So a labelled dataset should consist of a collection of $\mathbf{R}, \mathbf{s}$ pairs.

One way to get around this roadblock is to randomly generate $\mathbf{R}$ and use existing UQP solvers to obtain a unimodular $\mathbf{s}$. Unfortunately, due to the non-convexity of the UQP objective, the $\mathbf{s}$ produced by these solvers cannot be guaranteed to be globally optimal. Certain solvers are able to provide global optimality guarantees for specialized versions of the general UQP problem. However deep learning-based solvers that are trained on data generated based on a specialized version of UQP may not generalize well to other, more general UQP settings. In this section, we explain a new technique for generating training data for UQPs based on a characterization of global optima for UQP.

4.2.1 Data generation using globally optimal characterizations of some UQPs.

Some unimodular quadratic forms have analytical characterizations of its global optima and can be computed quite efficiently. The following theorem from [1] characterizes the global optima of such $\mathbf{R}$ matrices.

Theorem 2. *Let* $\mathbf{R}$ *be a Hermitian matrix with eigenvalue decomposition* $\mathbf{R} = \mathbf{U\Sigma U^H}$*. Suppose* $\mathbf{\Sigma}$ *is of the form*

$$\mathbf{\Sigma} = \boldsymbol{Diag}([\underbrace{\sigma_1 \dots \sigma_1}_{m\ times} \dots \sigma_{n-m+1}]^T) = \sigma_1 > \sigma_2 \geq \dots \sigma_{n-m+1},$$

and let $\mathbf{U_m}$ *be the matrix made from the first* m *columns of* $\mathbf{U}$*. Now suppose* $\tilde{\mathbf{s}} \in \Omega^n$ *lies in the linear space spanned by the columns of* $\mathbf{U_m}$*, i.e. there exists a vector* $\alpha \in \mathbb{C}^m$ *such that,* $\tilde{\mathbf{s}} = \mathbf{U_m}\alpha$*, Then* $\tilde{\mathbf{s}}$ *is the global optimizer of UQP.*

We refer the reader to [1] for the proof of Theorem 2. Similar to [1] we assume that the input $\mathbf{R}$ matrix is positive semi-definite. If $\mathbf{R}$ is not positive semi-definite, then we can perform a procedure called *diagonal loading* ($\mathbf{R} \leftarrow \mathbf{R} + \lambda I$), where $\lambda > -\sigma(\mathbf{R})$ to make it positive semi-definite. It can be shown that this procedure does not change the UQP.

Algorithm 4.1 Method 1: Data generation

```
1: Input: $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ random diagonal matrix with positive entries
2: Randomly generate $\mathbf{s}_i \in \Omega^n,\ i \in \{1 \dots n\}$
3: Define $\mathbf{S}$ as a matrix generated by stacking $\mathbf{s}_i$'s.
4: Compute $\mathbf{U} = \mathrm{QR}(\mathbf{S})$.          ▷ QR($\mathbf{X}$) performs QR decomposition of X.
5: Construct $\mathbf{R} \leftarrow \mathbf{U\Sigma U}^H$, $\mathbf{R} \leftarrow \mathbf{RR}^H$.          ▷ To ensure $R$ is Hermitian.
6: Return $(\mathbf{R}, \mathbf{s}_1)$.
```

We also propose another algorithm for data generation. This algorithm is motivated by (Equation 2.6) and the proof is provided in [1]

Algorithm 4.2 Method 2: Data generation

```
1: Input: ($\mathbf{R}_1 \in \mathbb{R}^{n\times n}$ , $\mathbf{s}_1 \in \Omega^n$) pair obtained from Algorithm 4.1
2: Randomly generate $\mathbf{s}_2 \in \Omega^n$
3: Compute $\mathbf{s}_0 = \mathbf{s}_1^* \odot \mathbf{s}_2$
4: Compute $\mathbf{R}_2 = \mathbf{R}_1 \odot \mathbf{s}_0\mathbf{s}_0^H$
5: Return ($\mathbf{R}_2, \mathbf{s}_2$).
```

4.3 Φ_{phase}- A UQP-tailored Unimodular Activation Function

Recall that the problem at hand, (Equation 1.1) is a constrained optimization problem with hard unimodularity constraint on the optimization variable $\mathbf{s}$. As we briefly described in Chapter 3, deep learning models, in its original form, does not enforce constraints on its learnable parameters. Also it is not immediately obvious as to how to incorporate the unimodularity constraint in deep learning. We have absorbed this limitation by introducing a new non-linear activation function, Φ_{phase} that takes in any vector $\mathbf{x}$ and gives a unimodular vector $\mathbf{s}$. The basis for this is type of non-linearity emerges from power-method like iterations PMLI, shown in (Equation 2.5). As described earlier, [1] showed that by successively applying PMLI, one can effectively search for a local optimum of the UQP. Motivated by this, we define the following parameterized nonlinear activation function Φ_{phase} as follows:

$$\Phi_{\text{phase}}(\mathbf{s}; (\mathbf{W}, \mathbf{R})) = e^{j \arg(\mathbf{WRs})}, \tag{4.1}$$

where, $\mathbf{W}$ is a learnable matrix parameter inside the neural network and $\mathbf{R}$ is the given input. An interesting property of Φ_{phase} is that the output of the activation function is inherently unimodular.

4.4 Our proposed method: Deep-PMLI

With the new nonlinear activation formulated as in (Equation 4.1), we are now ready to define our black-box solver for UQP. This following model, which we call *Deep-PMLI* repeatedly applies the Φ_{phase} activation function in (Equation 4.1) on the input, similar to the power method-like iterations described earlier in (Equation 2.5). The only difference between the iterations of the PMLI and Deep-PMLI is the fact that Φ_{phase} is parameterized by learnable weight matrices. To fully specify the architecture of Deep-PMLI, we are still to define a suitable loss function for supervising the training. One possible loss function is to directly compare the distance between the predicted unimodular vector $\hat{\mathbf{s}}$ and the true $\mathbf{s}$ from the generated training data. However it can be shown that the solution to a given UQP is not necessarily unique. This can be observed as a corollary of Theorem 2. Therefore, we considered the following loss function that compares the UQP objectives of $\mathbf{s}$ and $\hat{\mathbf{s}}$. Let $f = \mathbf{s}^H\mathbf{R}\mathbf{s}$ and $\hat{f} = \hat{\mathbf{s}}^H\mathbf{R}\hat{\mathbf{s}}$ be the

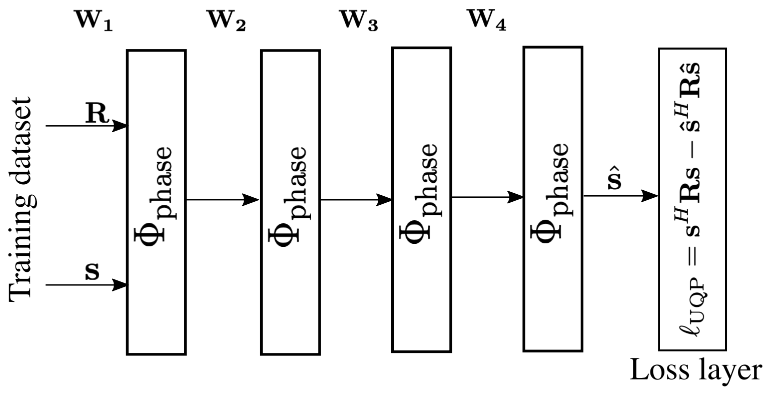


Figure 4: This figure illustrates the architecture of the proposed Deep-PMLI model with four Φ_{phase} layers. Here, $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4]$ are the weight parameters to be learned.

UQP objectives corresponding to $\mathbf{s}$ and $\hat{\mathbf{s}}$. The proposed loss functions is basically the squared loss of their difference.

$$\ell_{\text{UQP}} = (f - \hat{f})^2. \tag{4.2}$$

Note that ℓ_{UQP} will penalize the prediction for producing a f that is larger than $\hat{f}$. To deal with this problem one can consider other loss functions; the hinge loss of the difference between the objectives, for instance, that will only penalize f if it is smaller than $\hat{f}$. However, as we will soon see, the training data $\{\mathbf{R}_i, \mathbf{s}_i\}$ for our experiments are based on an analytical solution to UQP. This means that f can be guaranteed to be less than or equal to $\hat{f}$. Moreover, we found in our experiments that ℓ_{UQP} trains slightly better than the other loss functions.

4.4.1 Training and inference of Deep-PMLI

Recall that the UQP problem is defined in the complex domain; $\mathbf{s}$ and $\mathbf{R}$ are both complex. Therefore it is desirable to construct our deep network as a complex-valued function. Popular deep learning frameworks such as PyTorch [38] and Tensorflow [39] do not support complex valued inputs and layers out-of-the box as of writing of this thesis. This means that we could not leverage the autodiff capabilities offered by these packages. To this end, we performed an unconstrained minimization of ℓ_{UQP} (Equation 4.2) using `fminunc` [1].

During training, our hypothesis is that the model learns sufficient patterns of $\mathbf{R}$ and $\mathbf{s}$ from the labelled traiing data. The number of layers needed by the Deep-PMLI solver for a given training dataset, correspondingly the number of trainable parameters of the solver, are hyperparemeters and tuned manually.

During inference, test data is passed through the Deep-PMLI model using the learned weights $\mathbf{W} = [\mathbf{W}_1, \ldots, \mathbf{W}_n]$. The model predicts $\hat{\mathbf{s}}$ for a given $\mathbf{R}$. We would like to highlight that, though the Deep-PMLI solver draws motivation from PMLI (Equation 2.5), the theory of PMLI cannot be applied directly to Deep-PMLI. Specifically, the authors of [1] show local convergence guarantees for PMLI. However the fact that Deep-PMLI's layers are additionally parameterized (recall that this is the key difference between the two solvers), means that the convergence theory of PMLI no longer applies to Deep-PMLI.

[1]https://www.mathworks.com/help/optim/ug/fminunc.HTML

4.4.2 Metrics

To compare the performance of our model with that of the baselines, we used the metric "sub-optimality ratio" , λ, where, for a given matrix $\mathbf{R}$, Deep-PMLI model's prediction $\hat{\mathbf{s}}$ and baseline model's prediction $\mathbf{s}_{base}$,

$$\lambda = \frac{\hat{\mathbf{s}}^H \mathbf{R} \hat{\mathbf{s}}}{\mathbf{s}_{base}^H \mathbf{R} \mathbf{s}_{base}} \tag{4.3}$$

Here, if λ is greater than 1, it means that the Deep-PMLI model is performing better than the baseline model and vice versa.

Likewise, the solver's performance can also be compared with gold truth data using a similar metric, "sub-optimality gap", γ:

$$\gamma = \frac{\hat{\mathbf{s}}^H \mathbf{R} \hat{\mathbf{s}}}{\mathbf{s}^H \mathbf{R} \mathbf{s}} \tag{4.4}$$

4.5 Experiments and Results

To test the performance of our Deep-PMLI model, we conducted the following experiments. We trained the model with 5000 data points of dimensionality $d = 8$ generated by the data generators 4.1 and 4.2. First, we tested the performance of the model by predicting unimodular outputs for 50 test points with same $d = 8$ generated by the same data generators 4.1 and 4.2. We compared the performances by using "sub-optimality gap". In our second experiment, we tested the performance of the model by predicting unimodular outputs of randomly generated positive semi-definite Hermition matrices set of 50 test points and compared it against the baseline predictions of PMLI and MERIT given by as given [1]. We used the metric, "sub-optimality ratio", λ.

Number of Φ_{phase} layers	Mean γ
1	0.9615
2	0.98
3	0.96

TABLE I: PERFORMANCE OF DEEP-PMLI ON TEST DATASET WITH SAME DISTRIBUTION AS TRAINING DATASET.

Mean Number of Power iterations	Number of Φ_{phase} layers	Mean λ_{PMLI}	λ_{MERIT}
37.52	1	0.81	0.81
37.52	2	0.86	0.87
37.52	3	0.80	0.80

TABLE II: PERFORMANCE OF DEEP-PMLI ON RANDOM TEST DATASET.

4.6 Discussion

We see that, while predicting on the test data that lies on the same distribution as that of the training dataset, Deep-PMLI predicts almost as good as the gold truth with only 2 layers. But when we try to predict on random data, it performs poorly. Our hypothesis is that this behavior is due to the deficit in diversity of the data distribution, the model overfits to the training data distribution which results in poor generalization.

In the future, when more diverse benchmark dataset becomes available for UQP, Deep-PMLI could perform well on random data, which would make it very attractive for applications that require low-cost solutions to UQP.

Another limitation to this model is that, one cannot provide any theoretical guarantees for this model. Even when big data for UQP becomes available for the model to be trained , one cannot theoretically prove the generalization ability of this model, which makes Deep-PMLI less reliable for applications that require reliability.

This strong need for computational models that are both faster and more reliable motivated us to pursue further to develop "*Deep-INIT*", which is a deep learning based solution, that carefully parameterizes only parts of an existing algorithm without disturbing the problem structure, so that the guarantees are preserved, while significant speed up is achieved. We will discuss more about Deep-INIT in the next chapter.

CHAPTER 5

PROPOSED METHOD: DEEP-INIT

In this chapter, we discuss the architecture and training methods of our novel DNN solver for UQP named *Deep-INIT*.

5.1 MERIT iterations

Recall that the MERIT algorithm for UQP operates in two stages. For ease of explanation, we explicitly call the stages as *Stage 1* and *Stage 2*.

1. **Stage 1 [$\alpha_0 > 0$].** In this stage the algorithm determines a suitable value for the multiplier α_0 in (Equation 2.8).

2. **Stage 2 [$\alpha_0 = 0$].** For a chosen α_0 from Stage 1, the algorithm optimizes equation (Equation 2.9) and finds the optimal $(\boldsymbol{s}, \mathbf{Q}_1, \mathbf{P}_1)$ to compute $\mathbf{R}_{\boldsymbol{s}}$. Please see Algorithm 5.1 and [1] more details.

The MERIT algorithm continuously applies Stage 1 and Stage 2 in succession, until the necessary stopping criteria is met. The algorithm terminates when the estimated $||\mathbf{R}' - \mathbf{R}||_F \leq \epsilon; \mathbf{R}' = \mathbf{Q}_1^* + \mathbf{P}_1^*$. Now, note that Stage 2 is in itself another iterative procedure, within the iterates of Stage 1. Each iteration in Stage 2, which we call as a *MERIT iteration*, creates a sub-problem to equation (Equation 1.1) and finds the global optimum for the sub-problem. Specifically, it does the following:

- Find $\mathbf{R}_1 = \mathbf{P}_1 + \mathbf{Q}_1$, by optimally solving for $\mathbf{P}_1$ and $\mathbf{Q}_1$

- Find the global UQP associated with $\mathbf{R}_1$, by employing power method-like iterations.

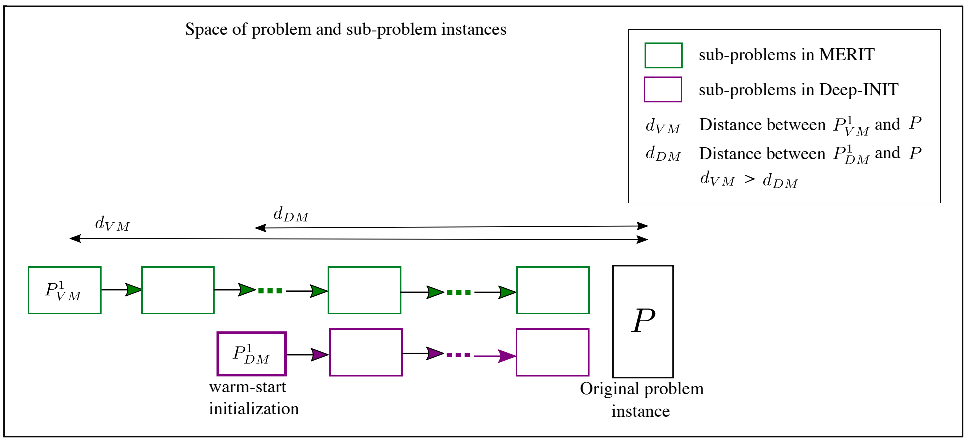
5.1.1 "Warm-starting" MERIT for faster convergence

A key observation we make here is that the number of iterations required by MERIT to approximately solve UQP (solution to (Equation 2.9)) depends directly on the proximity of the initial sub-problem to the original problem instance. Recall that, in each iteration, a new sub-problem is created by MERIT in such a way that it is closer to the original problem instance than the sub-problem constructed in the previous iteration. Therefore, one could potentially speed up MERIT by starting with a sub-problem that is as close as possible to the original problem instance. Specifically, the question we ask ourselves is, if we can find initializations to $\mathbf{P}_1, \mathbf{Q}_1$ and $\mathbf{s}$ in such a way that the resulting sub-problem is closer to the original UQP problem instance.

This work approaches this question by parameterizing the initial values of $\mathbf{s}$ and $\mathbf{Q}_1$ using parameters $\theta = [\mathbf{W}_1, \mathbf{W}_2]$ and learning these parameters using a deep network. Given a new UQP instance, the learned θ can be used to "warm-start" the optimization problem (Equation 2.9), by setting $\mathbf{Q}_1 = \mathbf{W}_1\mathbf{Q}_1^{rand}, \mathbf{s} = e^{j\arg(\mathbf{W}_2\mathbf{s}_1^{rand})}, \mathbf{P}_1 = |\boldsymbol{R}| \odot \boldsymbol{R}_{c+}$, where $\boldsymbol{R}_c = \cos(\arg(\boldsymbol{R}) - \arg(\mathbf{s}\mathbf{s}^H))$. Our hypothesis is that the sub-problem constructed using the "warm-started" parameters is closer to the original problem instance, than a random initialization. In the numerical experiments that we conducted, we observe that this is indeed the case.

Initialization methods for non-convex optimization problems have been extensively studied in the past, aiming to provide faster convergence and/or achieving "better" local optima or global optimum. [40] and its variants provided an initialization procedure for an alternative minimization algorithm for the phase retrieval problem, where they provided provable statistical guarantees under restricted settings like resampling requirement in every iteration. [41, 42] considered the alternating minimization problem for the sparse coding problem, where they characterised the "basin of attraction", that is, when the algorithm is initialized in the "basin of attraction" leads to the true solution. They also provided optimality guarantees under assumptions on the problem set up. [43] proposed to solve an unstructured random quadratic system in linear time through a novel initialization and descent procedure. They performed regularization on a spectral initialization procedure that provided them with a tighter initial guess. [44] proposed an initialization strategy for Non-negative matrix factorization problem and showed achieve faster convergence and less error.

In this work, we take inspiration from the above mentioned works on initialization methods for non-convex optimization. However, our motivations and end objectives are fundamentally different from the methods mentioned above. The key motivation of all of the above mentioned works is to obtain algorithms that lead to favorable theoretical analysis of convergence or optimality. To that end the authors of the above mentioned works often make strong modeling and data assumptions. The aim of our paper is to develop a data-driven initialization scheme that enables us to speed-up existing model based solutions to UQP. We make no further assumptions above the problem statement / data than what is already made by the underlying model-based solution, on which we build our solution. To the best of our knowledge, our work is one of the first few that proposes a data-driven initialization scheme to accelerate a model-based algorithm. Although the methods introduced in this work is tailor-made for UQPs, the



Space of problem and sub-problem instances
sub-problems in MERIT
sub-problems in Deep-INIT
d_{VM} Distance between P^1_{VM} and P
d_{DM} Distance between P^1_{DM} and P
$d_{VM} > d_{DM}$
d_{VM}
d_{DM}
P^1_{VM}
P^1_{DM}
P
warm-start initialization
Original problem instance

Figure 5: The initial sub-problem constructed by Deep-INIT is "closer" to the original problem instance than the initial sub-problem constructed by MERIT. In effect, Deep-INIT achieves local optimum in far lesser number of iterations than MERIT. The "closeness" between a sub-problem and the original problem instance is measured using d as in (Equation 5.7)

.

techniques proposed in this work can serve as a general recipe for any problem where proper initialization could potentially speed-up convergence. We now outline the new deep model and training details.

5.1.2 Training details

Let $g(\mathbf{P}_1, \mathbf{Q}_1, \mathbf{s})$ denote the objective function of problem (Equation 2.9).

$$g(\mathbf{R}) = ||\mathbf{R} - \underbrace{(\mathbf{Q}_1 + \mathbf{P}_1) \odot \mathbf{s}\mathbf{s}^H}_{\mathbf{R_s}}||_F \tag{5.1}$$

Let $\mathbf{Q}_1^0$ and $\mathbf{s}^0$ denote the initial values of $\mathbf{Q}_1$ and $\mathbf{s}$ respectively. Specifically, we can define:

$$\begin{aligned} \mathbf{Q}_1^0 &= \mathbf{W}_1 \mathbf{Q}_1^{\text{rand}} \\ \mathbf{s}^0 &= \exp(j \arg(\mathbf{W}_2 \mathbf{s}_1^{\text{rand}})) \end{aligned} \tag{5.2}$$

where, $\mathbf{Q}_1^{\text{rand}}$ is a random initial value of $\mathbf{Q}_1$ in $\mathcal{C}(\mathbf{V}_1)$ and $\mathbf{s}_1^{\text{rand}}$ is a random unimodular vector. Then in principle, the "warm-starting" of MERIT can be specified by the following objective

$$\begin{aligned} &\min_{\theta} \min_{\mathbf{P}_1, \mathbf{Q}_1, \mathbf{s}} \; g(\mathbf{R}) \\ \text{s.t. } & \mathbf{Q}_1^0 = \mathbf{W}_1 \mathbf{Q}_1^{\text{rand}} \\ & \mathbf{s}^0 = \exp(j \arg(\mathbf{W}_2 \mathbf{s}_1^{\text{rand}})) \end{aligned}$$

This however is a computationally expensive learning objective as each forward pass requires us to evaluate the full MERIT algorithm. Instead, in our experiments, we adopt a truncated version of g as our forward pass (or prediction function). Specifically let $f_\theta(\mathbf{R}; k)$ denote the new prediction function of our deep model, which is obtained by running a modified version of g, exactly k times. k is a hyper parameter which is tuned separately. Let ℓ denote the loss function employed by the model during training. Then our overall training objective can be specified as:

$$\min_{\theta} \min_{\mathbf{P}_1, \mathbf{Q}_1, \mathbf{s}} \; \ell(f_\theta(\mathbf{R}; k)). \tag{5.3}$$

5.1.3 Defining the prediction function f.

We chose the prediction function f to resemble a truncated MERIT solver, in that it partially (for k iterations) runs the optimization over $\mathbf{P}_1$ and $\mathbf{Q}_1$ with warm-start initialization on $\mathbf{Q}_1$ and $\mathbf{s}$. It then runs the local optimization over $\mathbf{s}$ for t iterations, where t is a tunable hyperparameter.

5.1.4 Loss function

The loss function ℓ that supervises the training of Deep-INIT plays a key role in learning accurate initializations for UQP. In our experiments we define ℓ to be a convex combination of two losses ℓ_1 and ℓ_2. Motivated by the original UQP objective (Equation 2.9), we define ℓ_1 and ℓ_2 as follows,

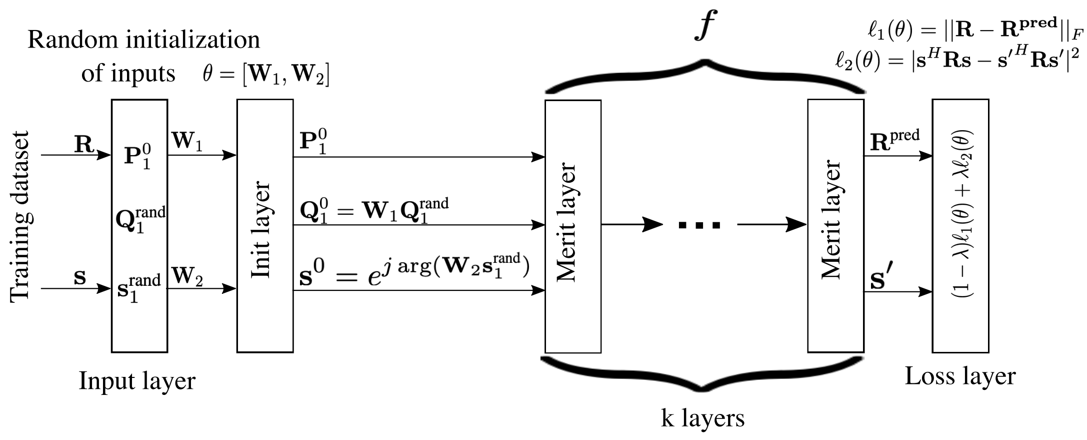
$$\ell_1(\theta) = ||\mathbf{R} - f(\mathbf{R})||_F; \quad \ell_2(\theta) = |\Psi(\mathbf{s}) - \Psi(\mathbf{s}')|^2 \tag{5.4}$$

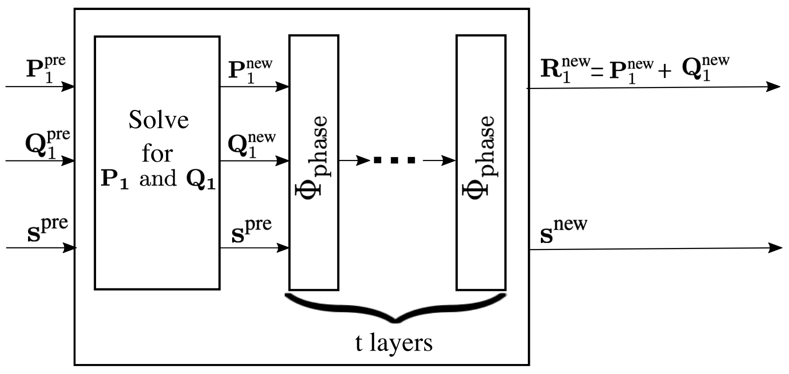
where $\Psi(\mathbf{x}) = \mathbf{x}^H\mathbf{R}\mathbf{x}$ (the original UQP objective). ℓ_1 promotes the Deep-INIT to learn initializations that encourage the initial sub-problems (enabled by the learned initializations) to be close to the original UQP problem instance. Whereas ℓ_2 ensures that the learned solutions ultimately lead to a good UQP solution. The overall loss function is a mixup of ℓ_1 and ℓ_2, with mixing parameter λ.

$$\ell(\theta) = (1 - \lambda)\ell_1(\theta) + \lambda\ell_2(\theta) \tag{5.5}$$

5.1.5 Inference

At test time, we are given with a new $\mathbf{R}_{\text{test}}$ matrix and we are required to predict the UQP solution corresponding to $\mathbf{R}_{\text{test}}$. In order to recover the optimal $\boldsymbol{s}$, we run the full MERIT algorithm where $\mathbf{Q}_1$ and $\boldsymbol{s}$ are initialized using learned weights (Equation 5.2). The expectation is

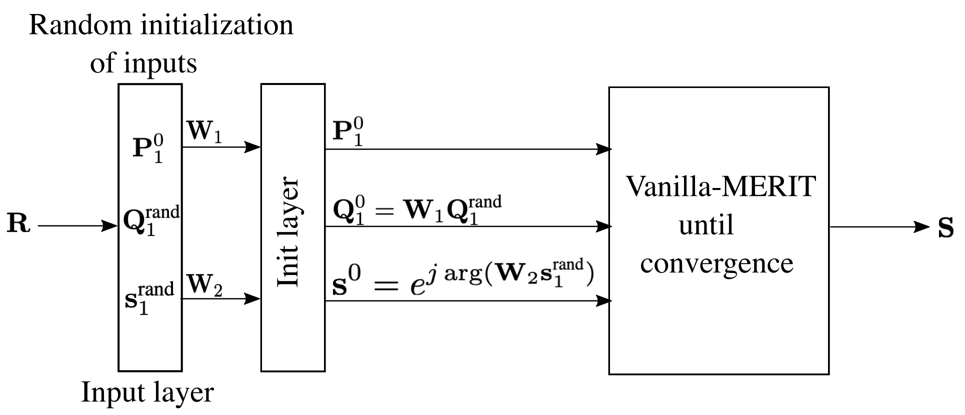

Random initialization of inputs $\theta = [\mathbf{W}_1, \mathbf{W}_2]$
f
$\ell_1(\theta) = ||\mathbf{R} - \mathbf{R}^{\mathbf{pred}}||_F$;
$\ell_2(\theta) = |\mathbf{s}^H\mathbf{R}\mathbf{s} - \mathbf{s}'^H\mathbf{R}\mathbf{s}'|^2$
Training dataset
$\mathbf{R}$
$\mathbf{s}$
$\mathbf{P}_1^0$
$\mathbf{Q}_1^{\text{rand}}$
$\mathbf{s}_1^{\text{rand}}$
$\mathbf{W}_1$
$\mathbf{W}_2$
Init layer
$\mathbf{P}_1^0$
$\mathbf{Q}_1^0 = \mathbf{W}_1\mathbf{Q}_1^{\text{rand}}$
$\mathbf{s}^0 = e^{j \arg(\mathbf{W}_2\mathbf{s}_1^{\text{rand}})}$
Merit layer
Merit layer
$\mathbf{R}^{\text{pred}}$
$\mathbf{s}'$
$(1-\lambda)\ell_1(\theta) + \lambda\ell_2(\theta)$
Input layer
k layers
Loss layer


$\mathbf{P}_1^{\text{pre}}$
$\mathbf{Q}_1^{\text{pre}}$
$\mathbf{s}^{\text{pre}}$
Solve for $\mathbf{P_1}$ and $\mathbf{Q_1}$
$\mathbf{P}_1^{\text{new}}$
$\mathbf{Q}_1^{\text{new}}$
$\mathbf{s}^{\text{pre}}$
Φphase
Φphase
$\mathbf{R}_1^{\text{new}} = \mathbf{P}_1^{\text{new}} + \mathbf{Q}_1^{\text{new}}$
$\mathbf{s}^{\text{new}}$
t layers

MERIT LAYER

Figure 6: An illustration of the training architecture of the proposed Deep-INIT solver. The **top** figure shows the high-level architecture of Deep-INIT training. The model is composed of several MERIT layers, which corresponds to a single iteration of the Deep-INIT algorithm. The image at the **bottom** gives more details about the MERIT layer. Here Φ_{phase} denotes one iteration of $\mathbf{s}$ optimization of MERIT.

that the learned weights possess sufficient patterns about the sub-problem structure of MERIT, helping us to "jump ahead" in the iterative sequence of MERIT. Indeed, this is what we observe in our experiments.



Random initialization of inputs
$\mathbf{R}$
$\mathbf{P}_1^0$
$\mathbf{Q}_1^{\text{rand}}$
$\mathbf{s}_1^{\text{rand}}$
$\mathbf{W}_1$
$\mathbf{W}_2$
Input layer
Init layer
$\mathbf{P}_1^0$
$\mathbf{Q}_1^0 = \mathbf{W}_1\mathbf{Q}_1^{\text{rand}}$
$\mathbf{s}^0 = e^{j\,\arg(\mathbf{W}_2\mathbf{s}_1^{\text{rand}})}$
Vanilla-MERIT until convergence
$\mathbf{S}$

Figure 7: The above figure illustrates the inference stage of the proposed Deep-INIT solver. The initial variables $\mathbf{P}_1^0, \mathbf{Q}_1^0$ and $\mathbf{s}^0$ of the MERIT algorithm are initialized using the weights W_1 and W_2 learnt from the Deep-INIT training, enabling faster convergence of MERIT algorithm.

5.2 Experiments

In this section we will validate our hypothesis that the weights learned through a Deep-INIT model is able to significantly reduce the number of iterations. We would like to note here that we do not establish a rigorous theoretical basis for the speed-up benefits provided by Deep-INIT. However, we empirically validate our intuition that Deep-INIT helps by-pass the first m iterations of the MERIT algorithm by virtue of smart initialization, as shown in Figure 5.

To that end, in this section, we will show several experiments that corroborate our hypothesis and intuition as stated above using synthetic UQP data.

5.2.1 Experimental setup

We conduct two numerical experiments to verify the effectiveness of our proposed Deep-INIT model and training procedure for approximating UQPs. The first experiment *Experiment 1* is

Algorithm 5.1 MERIT for UQP

```
The case of $\alpha_0 = 0$
1: Initialize the variables $\mathbf{Q}_1$ and $\mathbf{P}_1$ with $I$. Let $\mathbf{s}$ be a random vector in $\Omega^n$
2: Perform the diagonal loading of input $\mathbf{R}$.
3: Obtain the minimum of (Equation 2.9) with respect to $\mathbf{Q}_1$.
4: Obtain the minimum of (Equation 2.9) with respect to $\mathbf{P}_1$.
5: Minimize (Equation 2.9) with respect to $\mathbf{s}$.
6: Go to step 3 until a stop criterion is satisfied , e.g $||\mathbf{R} - (\mathbf{Q_1} + \mathbf{P_1}) \odot (\mathbf{ss}^H)||_F \leq \epsilon_0$
The case of $\alpha_0 > 0$
7: Initialize the variables ($\mathbf{Q}_1$,$\mathbf{P}_1$, $\mathbf{s}$) using the results obtained by the optimization of (Equation 2.9) as in case $\alpha_0 = 0$
8: Set $\delta$ (the step size for increasing $\alpha_0$ in each iteration). Let $\delta_0$ be the minimal $\delta$ to be considered and $\alpha_0 = 0$
9: Let $\alpha_0^{pre} = \alpha_0$, $\alpha_0^{new} = \alpha_0 + \delta$ and $\mathbf{R}^{'} = \mathbf{R} + \alpha_0^{new}\mathbf{ss}^H$
10: Solve
        $$\min_{\mathbf{s}\in\Omega^n, \mathbf{Q}_1\in\mathcal{C}_1, \mathbf{P}_1\in\mathcal{C}(V1)} ||\mathbf{R}^{'} - \underbrace{(\mathbf{Q}_1 + \mathbf{P}_1) \odot \mathbf{ss}^H}_{\mathbf{R}_{\boldsymbol{s}}}||_F$$
    using steps 3 to 6.
11: If $||\mathbf{R}^{'} - (\mathbf{Q}_1 + \mathbf{P}_1) \odot \mathbf{ss}^H||_F \leq \epsilon_0$ do:
      • If $\delta \geq \delta_0$, let $\delta \leftarrow \delta/2$ and initialize (10) with the previously obtained variables $(\boldsymbol{s}, \mathbf{Q}_1, \mathbf{P}_1)$ for $\alpha_0 = \alpha_0^{pre}$. Goto step 9.
      • If $\delta < \delta_0$ Stop
12: Else let $\alpha_0 = \alpha_0^{new}$ and goto step 9
```

used to validate the speedup benefits of Deep-INIT. *Experiment 2* tests our hypothesis that the warm-start initializations that Deep-INIT provides, creates a sub-problem that is closer to the original problem instance than random initialization.

Metrics. For each experiment, we measure the following two metrics:

Algorithm 5.2 Deep-INIT for UQP

One training iteration of Deep-INIT

```
1: Initialize: $\mathbf{Q}_1^0 = W_1\mathbf{Q}_1^{\text{rand}}$ , $\mathbf{s}^0 = \exp(j\arg(W_2\mathbf{s}_1^{\text{rand}}))$ and $\mathbf{P}_1^0 = |\mathbf{R}| \odot \boldsymbol{R}_{c+}$, where $\mathbf{R}_c = \cos(\arg(\mathbf{R}) - \arg(\mathbf{s}\mathbf{s}^H))$, $i = 1$
2: Perform the diagonal loading of input $\mathbf{R}$.
3: while $i < k$ do
4:     $i \leftarrow i + 1$
5:     Obtain the minimum of (Equation 2.9) with respect to $\mathbf{Q}_1^0$.
6:     Obtain the minimum of (Equation 2.9) with respect to $\mathbf{P}_1^0$.
7:     Minimize (Equation 2.9) with respect to $\mathbf{s}_0$ only for $t$ iterations.
8: end while
9: Compute the loss function $\ell$ as in (Equation 5.5) and update the weights $(W_1, W_2)$
```

Inference

```
10: Initialize the variables $(\mathbf{Q_1}, \mathbf{P_1}, \mathbf{s})$ as $(\mathbf{Q_1^0}, \mathbf{P_1^0}, \mathbf{s}^0)$ where $(W_1, W_2)$ are the learned weights from the training.
11: Follow steps 8 to 12 from MERIT algorithm 5.1
```

1. *Percentage reduction in iterations* (η): As the name suggests, η measures the reduction in number of iterations of the underlying MERIT procedure after using Deep-INIT. Let n_v and n_d denote the number of iterations taken by MERIT and Deep-INIT to converge to a locally optimal solution for a given UQP problem instance. Then η is defined as:

$$\eta = \frac{n_v - n_d}{n_v} \tag{5.6}$$

If Deep-INIT doesn't reduce the number of iterations compared to MERIT, then η will be 0. On the extreme end, if Deep-INIT converges in the first iteration, then η will be very close to 1.

2. *Wall-clock time (t)*: We use the `MATLAB` command `cputime`[1] to measure the time taken (in seconds) by each of MERIT and Deep-INIT.

Experiment 1. We first train our Deep-INIT model with data generated using Algorithm 4.1 and Algorithm 4.2. For inference, we generated random positive definite input matrices. We train and test the model on four different datasets with varied dimensionality d. Specifically we generated 500 data points for training and 50 data points for inference with $d \in \{8, 16, 24, 32\}$, creating a total of four variations. For each variation, we ran the training and testing 10 different times and recorded the mean and standard deviation of our metrics.

Experiment 2. For this experiment, we fix $d = 8$ and generate 10 different training (500 points) and test (50 points) sets. For each set, we first train the Deep-INIT algorithm and obtain the warm-start initializations $(Q_1, \boldsymbol{s})$. Using these learned parameters we run 1 truncated MERIT iteration (function f, defined the previous section). We run another truncated MERIT iteration with random initializations (no warm-start). In both the cases, we measure the approximation strength, measured as the difference between the estimated and true $\mathbf{R}$. If the initial sub-problem is closer to the original problem instance, then we might expect that the $\mathbf{R}$ is well approximated after 1 truncated MERIT iteration. We measure appoximation strength (δ) as:

$$\delta(\mathbf{R}, \mathbf{R}') = ||\mathbf{R} - \mathbf{R}'||_F \tag{5.7}$$

[1]https://www.mathworks.com/help/matlab/ref/cputime.html

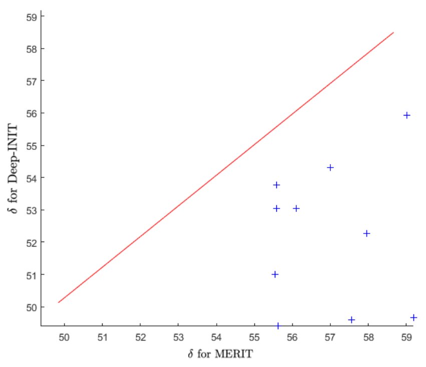
5.2.2 Discussion

The results of Experiment 1 are given in Table III. What we observe is that across all the variations, Deep-INIT (MERIT with learned parameters used for warm-start) achieves a significant speed-up over the MERIT, but without sacrificing estimation quality.

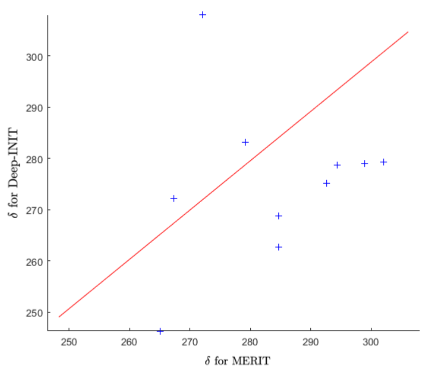
The scatter plot in Figure 8 may be interpreted as follows. The $y-$axis represents the δ values of Deep-INIT and $x-$axis denotes the δ values of MERIT. If δ for Deep-INIT is smaller overall than MERIT, then we expect to see more points below the diagonal reference line on the scatter plot. This is indeed what we observe from the plots.

d	η (%)	$\gamma_{\text{Deep-INIT}}$	γ_{MERIT}	$t_{\text{Deep-INIT}}$	t_{MERIT}
8	15±6	0.989	0.987	0.33±0.04	0.43
16	19±5	0.954	0.949	17.7±2.64	22.62
24	14±6	0.93	0.924	51.54±4.4	54.8
32	18±7	0.9	0.9	149.18±6	195.47

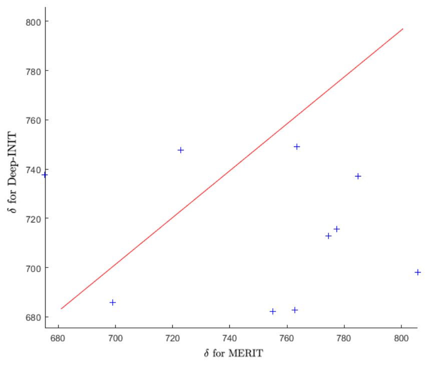
TABLE III: PERFORMANCE OF DEEP-INIT MODEL'S WARM-START MEASURED BY PERCENTAGE REDUCTION IN NUMBER OF ITERATIONS AND WALL-CLOCK TIME. HERE d IS THE DIMENSION OF EACH SAMPLE POINT.



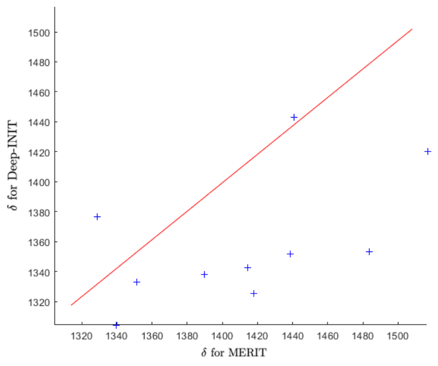
(a) $N = 8$



(b) $N = 16$



(c) $N = 24$



(d) $N = 32$

Figure 8: Scatter plot of approximation gap of vanilla MERIT vs Deep-INIT δ measured for $N = \{8, 16, 24, 32\}$ after running 1 iteration of truncated MERIT. Here Deep-INIT is initialized with the learned weights using warm-start initialization

5.3 Analysis

Sub-optimality analysis. The authors of [1] show a data-dependent sub-optimality analysis of MERIT using the sub-optimality gap γ,

$$\gamma = \frac{\mathbf{s}^H \mathbf{R} \mathbf{s}}{\mathbf{s}^H \mathbf{R}_{\mathbf{s}} \mathbf{s}} = 1 - \frac{\alpha_0 n^2}{\mathbf{s}^H \mathbf{R}_{\mathbf{s}} \mathbf{s}}$$

where, n is the length of $\mathbf{s}$.

A key feature of the proposed Deep-INIT algorithm is that it is able to preserve the optimality guarantees provided by the MERIT algorithm [1] during inference. This is because at inference time, Deep-INIT does not alter the main MERIT algorithm and achieves the speed-up merely by smart initialization of the optimization variables of the original MERIT algorithm. In future, we envisage that by carefully studying and reparameterizing the inner layers of the MERIT algorithm, we could obtain tighter optimality guarantees without sacrificing model performance.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

In this work, we developed two deep learning-based solvers for solving the NP-hard optimization problem of Unimodular Quadratic Programming (UQP). First, we proposed a black-box solver Deep-PMLI, that predicts a unimodular output $\mathbf{s}$, for a given UQP, after learning patterns from already seen examples. Our second proposed algorithm, Deep-INIT, automatically learns right initializations for the parameters of a model-based solver for UQP, called MERIT. With the learned initializations, Deep-INIT provably converges to a locally optimal solution for UQP, in significantly fewer number of iterations than MERIT. As an added contribution, we also proposed a novel data generation algorithm for obtaining training data to learn our Deep-PMLI and Deep-INIT solvers.

6.1 Future Works

In our view, there are two directions where this idea can be extended and used. One is to extend our proposed models for UQP to produce better UQP solutions or tighter guarantees. The second is to use our Deep-INIT model on other problems to achieve speed-up.

One way to improve UQP solutions is by improving Deep-INIT through more involved reparameterization of the MERIT solver and refined analysis of the solution. Specifically, in our current solution, the initializations of the MERIT solver are learned. In future, other parts of the underlying MERIT solver, such as the sub-problems can be parameterized and learned from data. Furthermore, such a parameterization will demand involved analysis, but has a potential to provide tighter optimality guarantees.

Another future direction is to improve our Deep-PMLI solver by enhancing its generalization capabilities. Recall from the numerical experiment section that, Deep-PMLI has the tendency to overfit to the training data distribution and as a result saw a drop in the test performance. In future, by making use of recent technological advancements in machine learning like “domain adaptation”, to align training and test data-distributions close to each other, Deep-PMLI can be made to generalize better on unseen test data.

Finally, we would like to highlight that the Deep-INIT is a general recipe that could be potentially used to initialize other non-convex optimization problems, in general. This means that one could potentially use Deep-INIT on algorithms other than MERIT for other signal processing problems and reap its theoretical and scalability benefits.

CITED LITERATURE

1. Soltanalian, M. and Stoica, P.: Designing unimodular codes via quadratic optimization. IEEE Transactions on Signal Processing, 62(5):1221–1234, 2014.

2. Tsinos, C. G. and Ottersten, B.: An efficient algorithm for unit-modulus quadratic programs with application in beamforming for wireless sensor networks. IEEE Signal Processing Letters, 25(2):169–173, 2017.

3. Jiang, F., Chen, J., and Swindlehurst, A. L.: Estimation in phase-shift and forward wireless sensor networks. IEEE transactions on signal processing, 61(15):3840–3851, 2013.

4. Leong, A. S. and Dey, S.: On scaling laws of diversity schemes in decentralized estimation. IEEE transactions on information theory, 57(7):4740–4759, 2011.

5. Kay, S. M.: Fundamentals of statistical signal processing. Prentice Hall PTR, 1993.

6. Li, H., Chen, C., and Zhu, X.: Unimodular waveform design for mimo radar transmit beamforming. In MATEC Web of Conferences, volume 208, page 02003. EDP Sciences, 2018.

7. De Maio, A., Huang, Y., Piezzo, M., Zhang, S., and Farina, A.: Design of optimized radar codes with a peak to average power ratio constraint. IEEE Transactions on Signal Processing, 59(6):2683–2697, 2011.

8. De Maio, A. and Farina, A.: Code selection for radar performance optimization. In 2007 International Waveform Diversity and Design Conference, pages 219–223. IEEE, 2007.

9. Wang, Y. and Xu, Z.: Phase retrieval for sparse signals. Applied and Computational Harmonic Analysis, 37(3):531–544, 2014.

10. Wang, G., Zhang, L., Giannakis, G. B., Akçakaya, M., and Chen, J.: Sparse phase retrieval via truncated amplitude flow. IEEE Transactions on Signal Processing, 66(2):479–491, 2017.

11. Gao, B., Sun, Q., Wang, Y., and Xu, Z.: Phase retrieval from the magnitudes of affine linear measurements. Advances in Applied Mathematics, 93:121–141, 2018.

12. Waldspurger, I., d'Aspremont, A., and Mallat, S.: Phase recovery, maxcut and complex semidefinite programming. Mathematical Programming, 149(1-2):47–81, 2015.

13. Tranter, J.: Fast unit-modulus least squares with applications in beamforming and phase retrieval. 2016.

14. Khobahi, S., Bose, A., and Soltanalian, M.: Deep radar waveform design for efficient automotive radar sensing. In 2020 IEEE 11th Sensor Array and Multichannel Signal Processing Workshop (SAM), pages 1–5. IEEE, 2020.

15. Zhang, S. and Huang, Y.: Complex quadratic optimization and semidefinite programming. SIAM Journal on Optimization, 16(3):871–890, 2006.

16. Jaldén, J., Martin, C., and Ottersten, B.: Semidefinite programming for detection in linear systems-optimality conditions and space-time decoding. In 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03)., volume 4, pages IV–9. IEEE, 2003.

17. Kyrillidis, A. T. and Karystinos, G. N.: Rank-deficient quadratic-form maximization over m-phase alphabet: Polynomial-complexity solvability and algorithmic developments. In 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 3856–3859. IEEE, 2011.

18. Verdú, S.: Computational complexity of optimum multiuser detection. Algorithmica, 4(1-4):303–312, 1989.

19. Ma, W.-K., Vo, B.-N., Davidson, T. N., and Ching, P.-C.: Blind ml detection of orthogonal space-time block codes: Efficient high-performance implementations. IEEE Transactions on Signal Processing, 54(2):738–751, 2006.

20. Cui, T. and Tellambura, C.: Joint channel estimation and data detection for ofdm systems via sphere decoding. In IEEE Global Telecommunications Conference, 2004. GLOBECOM'04., volume 6, pages 3656–3660. IEEE, 2004.

21. Boyd, S., Boyd, S. P., and Vandenberghe, L.: Convex optimization. Cambridge university press, 2004.

22. Goemans, M. X. and Williamson, D. P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM), 42(6):1115–1145, 1995.

23. Luo, Z.-Q., Ma, W.-K., So, A. M.-C., Ye, Y., and Zhang, S.: Semidefinite relaxation of quadratic optimization problems. IEEE Signal Processing Magazine, 27(3):20–34, 2010.

24. So, A. M.-C., Zhang, J., and Ye, Y.: On approximating complex quadratic optimization problems via semidefinite programming relaxations. Mathematical Programming, 110(1):93–110, 2007.

25. Alizadeh, F.: Interior point methods in semidefinite programming with applications to combinatorial optimization. SIAM journal on Optimization, 5(1):13–51, 1995.

26. Nesterov, Y. and Nemirovskii, A.: Interior-point polynomial algorithms in convex programming, volume 13. Siam, 1994.

27. Soltanalian, M., Tang, B., Li, J., and Stoica, P.: Joint design of the receive filter and transmit sequence for active sensing. IEEE Signal Processing Letters, 20(5):423–426, 2013.

28. Ragi, S., Chong, E. K., and Mittelmann, H. D.: Polynomial-time methods to solve unimodular quadratic programs with performance guarantees. IEEE Transactions on Aerospace and Electronic Systems, 55(5):2118–2127, 2018.

29. Van Der Veen, A.-J. and Paulraj, A.: An analytical constant modulus algorithm. IEEE Transactions on Signal Processing, 44(5):1136–1155, 1996.

30. Vapnik, V.: The nature of statistical learning theory. Springer science & business media, 2013.

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

32. Nesterov, Y.: A method for unconstrained convex minimization problem with the rate of convergence o (1/k^ 2). In Doklady an ussr, volume 269, pages 543–547, 1983.

33. Hershey, J. R., Roux, J. L., and Weninger, F.: Deep unfolding: Model-based inspiration of novel deep architectures. arXiv preprint arXiv:1409.2574, 2014.

34. Belanger, D. and McCallum, A.: Structured prediction energy networks. In International Conference on Machine Learning, pages 983–992, 2016.

35. Borgerding, M., Schniter, P., and Rangan, S.: Amp-inspired deep networks for sparse linear inverse problems. IEEE Transactions on Signal Processing, 65(16):4293–4308, 2017.

36. Khobahi, S., Naimipour, N., Soltanalian, M., and Eldar, Y. C.: Deep signal recovery with one-bit quantization. In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 2987–2991. IEEE, 2019.

37. Khobahi, S. and Soltanalian, M.: Model-aware deep architectures for one-bit compressive variational autoencoding. arXiv preprint arXiv:1911.12410, 2019.

38. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems, pages 8024–8035, 2019.

39. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), pages 265–283, 2016.

40. Netrapalli, P., Jain, P., and Sanghavi, S.: Phase retrieval using alternating minimization. IEEE Transactions on Signal Processing, 63(18):4814–4826, 2015.

41. Agarwal, A., Anandkumar, A., Jain, P., Netrapalli, P., and Tandon, R.: Learning sparsely used overcomplete dictionaries. In Conference on Learning Theory, pages 123–137, 2014.

42. Agarwal, A., Anandkumar, A., Jain, P., and Netrapalli, P.: Learning sparsely used overcomplete dictionaries via alternating minimization. SIAM Journal on Optimization, 26(4):2775–2799, 2016.

43. Chen, Y. and Candes, E.: Solving random quadratic systems of equations is nearly as easy as solving linear systems. In Advances in Neural Information Processing Systems, pages 739–747, 2015.

44. Gong, L. and Nandi, A. K.: An enhanced initialization method for non-negative matrix factorization. In 2013 IEEE International Workshop on Machine Learning for Signal Processing (MLSP), pages 1–6. IEEE, 2013.

VITA

AMRUTHA VARSHINI RAMESH

EDUCATION	B.E. Electrical and Electronics Engineering, Sathyabama University, Chennai, India.
	M.S. Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, IL, USA.
RESEARCH EXPERIENCE	**Research Assitant**, University of Illinois at Chicago, Chicago, IL, USA, Summer 2020. Advisor: Prof. Mojtaba Soltanalian
INDUSTRY EXPERIENCE	**Software Engineer**, Agaraz Data Inc., Chicago, IL Jan 2017 - Nov 2017
	Modeling Engineer, Fiat Chrysler Automobiles, Chennai, India, Aug 2011 - Jul 2015
RESEARCH PROJECTS	Efficient and guaranteed methods for non-convex optimization, Chicago, IL, USA, Jan 2018 - Dec 2020
	Time-aware recommendation systems, Chicago, IL, USA, Jan 2018 - Dec 2020