DUALITY BETWEEN DEEP LEARNING AND ALGORITHM DESIGN

A Dissertation
Presented to
The Academic Faculty

By

Xinshi Chen

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Machine Learning
School of Mathematics

Georgia Institute of Technology

May 2022

DUALITY BETWEEN DEEP LEARNING AND ALGORITHM DESIGN

Thesis committee:

Dr. Le Song
Department of Machine Learning
Mohamed bin Zayed University of Artificial Intelligence

Dr. Xiuwei Zhang
School of Computational Science & Engineering
Georgia Institute of Technology

Dr. Kolchinskii Vladimir
School of Mathematics
Georgia Institute of Technology

Dr. Molei Tao
School of Mathematics
Georgia Institute of Technology

Dr. Guanghui Lan
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date approved: April 18, 2022

To all of the people that light up my life.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Le Song, for his guidance, support, and encouragement over the past five years. I still remember how little I knew about machine learning researches when I first came to Georgia Tech. Le's encouragement and patience have greatly helped me in moving to the right track. From our countless research meetings I learned how to conduct machine learning researches. I am grateful to Le for dedicating so much time and effort to shaping the direction of the works presented in this thesis. This thesis would not exist without his guidance and support.

I would also like to thank Vladimir Koltchinskii for advising and supporting my Ph.D. study throughout last year. My journey at Georgia Tech will not go so smoothly without his support. My appreciation is further extended to my thesis committee members, Guanghui Lan, Xiuwei Zhang, and Molei Tao, for spending their precious time on the examination of my thesis research.

I have had the pleasure of collaborating with many excellent colleagues and friends. Thanks to all of my close collaborators: Binghong Chen, Hanjun Dai, Caleb Ellington, Yu Li, Hui Li, Harsh Shrivastava, Haoran Sun, Yuan Yang, Yan Zhu, Yuyu Zhang, Yufei Zhang, etc. Thanks to professors who have contributed to projects appearing in this thesis: Srinivas Aluru, Xin Gao, Guanghui Lan, Han Liu, Christoph Reisinger, Eric Xing. Without the contribution from my collaborators, the projects in this thesis can never be accomplished. Furthermore, I would like to thank some professors who have provided very valuable suggestions to my research: Yongxin Chen, Yonina Eldar, Quanquan Gu, Vladimir Koltchinskii, Molei Tao, Santosh Vempala, Haomin Zhou, Xiuwei Zhang. Discussions with them can always help to refresh my thoughts. Looking back, my mentors earlier in my life stimulated my interests in science and opened my eyes. Here I express my special thanks to Eric Chung, Raymond Chan, and Kwai Wong.

My research and growth in machine learning also benefit from my internships and visits

at Ant Group in 2018, Facebook in 2020, and MBZUAI in 2021. Thank Akbobek, Muhan, Peter, Romain, Shaohua, Wenliang, Yan, and Zhiqiang, for their helps and funny chats during my internships and visits. I further take this opportunity to thank the financial support for my Ph.D. study offered by Google PhD Fellowship.

My Ph.D. journey can never become colorful without the friends I met at Georgia Tech. Thank Jiachen, Karan, Kuan, Mengyang, Qingru, Tianzhe, Weiyang, and Yinghao, for entertaining conversations about life, research, health, and everything else. Despite the distance, I am also thankful to my old pals, especially Jiahao, Que, Shuang, Simon, Tony, Yuliang, and Zihao. Over the past five years and more, they have been a continual source of fun, energy, support, and distractions from this thesis.

Finally, I want to thank my family members for their everlasting love, support, and encouragement over the years. My gratitude and love to them can never be expressed in words.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis introduces "Duality Between Deep Learning And Algorithm Design". Deep learning is a data-driven method, whereas conventional algorithm design is a knowledge-driven method. Based on their connections and complementary features, this thesis develops new methods to combine the merits of both and, in turn, improve both. Specifically:

- **Algorithm inspired deep learning model.** Despite the unprecedented performance of deep learning in many computer vision and natural language processing problems, the development of deep neural networks is hindered by their black-box nature, i.e., a lack of interpretability and the need for very large training sets. To eliminate these issues, this thesis introduces the use of algorithms as *modeling priors* to integrate specialized knowledge of domain experts into deep learning models. From both the *empirical and theoretical* perspective, this thesis explains how such algorithm inspired deep learning models can achieve improved interpretability and sample efficiency.

- **Deep learning based algorithm design.** In conventional algorithm design, domain experts will first develop a model to describe the mechanism behind it and then establish a mathematical algorithm to find the solution. Notwithstanding its interpretability, this model-based method is inferior in terms of its limited effective range and accuracy. This is mostly due to the simplifying assumptions of the models which often deviate from real-world problems. To address these issues, this thesis investigates the potential of *deep learning based methods* for discovering data-driven algorithms that adapt better to the interested problem distribution. This thesis explains how to design data-driven components to replace some fixed procedures in traditional algorithms, how to optimize these components, and how these data-driven components can *improve* the accuracy and efficiency of traditional algorithms, from both the *empirical and theoretical* perspectives.

Keywords: Deep learning, Optimization algorithms, Learning to optimize, Meta learning, Structured prediction, Sparse statistical recovery, Graph neural network, Bioinformatics applications.

CHAPTER 1

INTRODUCTION

1.1 Motivation

This thesis is motivated by the challenges in two seemingly disparate research topics: deep learning and algorithm design.

Deep learning has led to significant improvements in data-rich applications such as speech recognition, machine translation, and image recognition. Given input-output pairs, deep learning can efficiently optimize the weights in a neural network to improve its performance on a training dataset. However, in order to achieve acceptable generalization performance, a deep learning model (i.e., neural network) must be trained on a massive amount of labeled data. The process of labeling a large number of examples and then learning

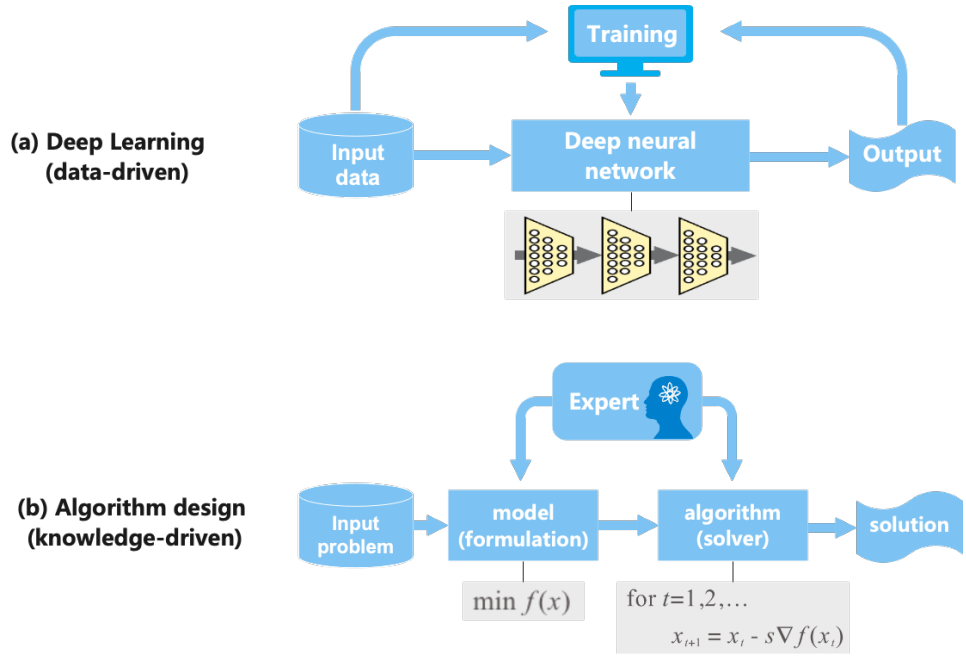


Figure 1.1: Different processes of **(a)** deep learning and **(b)** algorithm design.

Table 1.1: Complimentary features of deep learning and algorithm design.

features	Deep Learning (data-driven)		Algorithm Design (knowledge-based)	
sample complexity	✗	data-hungry	✓	almost zero training data
interpretability	✗	not interpretable	✓	interpretable
flexibility	✓	universal approximator	✗	limited to human knowledge
adaptation	✓	adapted to training data	✗	incompatible assumptions
automation	✓	automatically learned	✗	labor-intensive

from them are costly and time-consuming, especially in domains where the labeling work requires well-trained experts, such as healthcare, computational biology, and finance. In addition, the methods are typically not interpretable. Many methods directly apply general network architectures to different problems and learn certain underlying mappings, such as classification and regression functions, completely through end-to-end training. It is therefore hard to discover what is learned inside the networks by examining the high dimensional network parameters, and what the roles are of individual parameters. Adoptions of deep learning to new areas are hindered by its hunger for data and lack of interpretability.

Algorithms are step-by-step instructions designed by experts for computers to solve a problem. To solve a problem, domain experts first develop a model (formulation) to describe the mechanism and then establish a mathematical algorithm to find the solution. This process has three obvious drawbacks. First, the model developed by human experts usually involves simplified assumptions (e.g., convexity) so that it is easier to understand and to perform mathematical derivations. However, those assumptions lead to simplified models that deviate from real-world problems which are usually intrinsically hard, complex, and large-scale. Second, a human-designed algorithm is understandable by human, but it is also restricted to human knowledge. A better algorithm may exist, but finding it may be challenging and non-intuitive to human. As a result of the incompatible assumptions and limitations to human knowledge, the performances of hand-designed algorithms on real-world problems are still far from satisfactory. Finally, this algorithm design process requires the

experts to perform extensive research and experiment with many trial-and-errors, which is labor-intensive.

The aforementioned challenges in deep learning and algorithm design are summarized in Table 1.1. Seemingly unrelated, they are closely connected in the sense that the features of these two methods are complementary - the advantages of deep learning precisely correspond to the disadvantages of algorithm design, and vice versa. Motivated by their connections and complimentary features as listed in Table 1.1, the researches in this thesis focus on developing new methods to combine their advantages to improve both of them and providing theoretical insights fro understanding the proposed methods.

1.2 A Duality View

In order to combine the benefits of deep learning and algorithm design, this thesis proposes a duality view to connect neural networks with algorithms: *On the one hand, a deep learning model (neural network) can be viewed as a neural algorithm. On the other hand, an algorithm can be viewed as a hand-crafted deep learning model.* The duality view of deep learning and algorithm design offers the foundation for bridging these two research domains.

1.2.1 Correspondence in terms of their definitions

The first part of the duality view establishes how neural networks and algorithms, in terms of their definitions, can be converted to each other.

1. *A deep learning model (neural network) can be viewed as a neural algorithm* which takes an input, and carries out step-by-step neural processing to arrive at an answer. Each layer in the neural network can be viewed as one algorithm step, and thus a T-layer neural network can be viewed as a T-step algorithm. Unlike pre-defined update steps in the algorithm, the operations in a neural network are learned from data. After trained on enough problem instances, the learned neural network can be used as a neural algorithm to solve

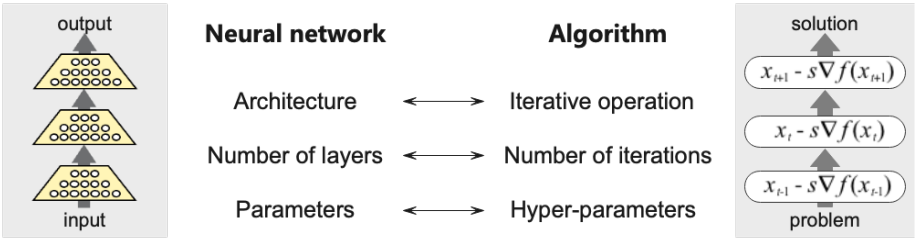


Figure 1.2: Duality view: correspondence between neural networks and algorithms.

new problems.

2. *An algorithm can be viewed as a hand-crafted deep learning model*. Each iteration of an algorithm can be viewed as one layer in the neural network. The problem to be solved can be considered as the neural network's input, and the solution as its output. For instance, for a linear optimization algorithm, the input can be a set of matrices and vectors that describe the optimization objective and constraints, and the output is the estimated optimal solution returned from the algorithm. Unlike conventional neural networks that contain a massive amount of parameters, algorithms usually only contain a few number of hyperparameters such as the step size and the regularization parameters which can be obtained by line-search or grid-search.

Table 1.2: Correspondence between a neural network and an optimization algorithm in terms of their definitions and terminology.

DL terminology	Feed-forward network	Gradient descent algorithm	Algorithm terminology
t-th layer	$\boldsymbol{x}_t = \sigma\left(W\boldsymbol{x}_{t-1} + \boldsymbol{b}\right)$	$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} - s \cdot \nabla f(\boldsymbol{x}_{t-1})$	t-th iteration
number of layers	T (static)	$T_{f,s}$ (dynamic)	number of iterations
parameters	$W, \boldsymbol{b}$	s	hyperparameters

Therefore, a neural network can be thought of as a special algorithm, and an algorithm can be thought of as a special neural network. Their correspondences are shown in Figure 1.2 and demonstrated in Table 1.2.

1.2.2 Correspondence in terms of their properties

Apart from establishing the correspondence between neural networks and algorithms in their definitions, making connections between their properties is probably more important for understanding their behavior and performance. Therefore, the second part of the duality view discusses how neural networks and algorithms, in terms of their properties, can be converted to each other. Table 1.3 summarizes the correspondences.

Deep learning community cares the most about the representation and generalization ability of neural networks. Let's start with the representation ability. Given a neural architecture, its representation ability is characterized by the **approximation error**, which measures the error between the target function and the best possible model defined by the neural architecture. It is well-known that feed-forward networks are universal approximators [1], meaning that they can approximate any continuous functions. What's more important than the approximation error itself is the **approximation efficiency**, meaning how fast the approximation error can decrease when increasing the width and depth of the neural network [2, 3]. Now we are ready to ask, *which property of algorithms is similar to the approximation efficiency of neural networks?* We say the answer is **convergence rate**. The convergence rate of an optimization algorithm expresses how fast the optimization error decreases as the number of iterations grows. If we assume the target function is the oracle solver which maps any function to its minimizer, then the approximation efficiency of an iterative algorithm is exactly characterized by the convergence rate.

Another most important property of neural networks is their generalization ability. Depending on the analysis framework, characterizing the generalization ability resorts to an-

Table 1.3: Correspondence in terms of properties.

Meaning	Neural network property	Algorithm property
decrease rate of approximation error	approximation efficiency	convergence rate
Lipschitzity w.r.t. input	robustness	stability
Lipschitzity w.r.t. parameters	sharpness	sensitivity

alyzing the robustness [4, 5] and sharpness [6, 7] in many works. **Robustness** measures the vulnerability of neural networks to *small perturbations around the input*, which is the Lipschitz constant of the network with respect to the input. **Sharpness** is also referred to "parameter perturbation error". It measures the robustness to *small perturbations on the parameters*, which is the Lipschitz constant of the network with respect to its parameters. It has been used for deriving generalization bounds of neural networks, both in the Rademacher complexity framework [8] and PAC-Bayes framework [9].

Which properties of algorithms are similar to the robustness and sharpness of neural networks? We claim that the **stability** of an algorithm is similar to the robustness, and the **sensitivity** of an algorithm is similar to sharpness. Stability of an algorithm is defined as its robustness to small perturbations in the optimization objective, and sensitivity is defined as its robustness to small perturbations in the step size. When we think of an optimization algorithm as a function that maps the optimization objective to the approximation minimizer and its step size as the (hyper)parameter, the correspondence between these properties is obvious.

Table 1.3 summarizes how the algorithm properties are related to the approximation efficiency, robustness, and sharpness of neural networks. Such correspondences are very useful for understanding the behaviors of algorithms when they are used as layers in deep learning models. For many optimization algorithms, their properties including the convergence rate and stability are well-studied. As an example, Table 1.4 summarizes the algorithmic properties of gradient descent (GD) and Nesterov's accelerated gradient (NAG) method including their convergence rate, stability, and sensitivity. As a result, if we use

Table 1.4: Comparison of algorithmic properties between GD and NAG. Variable t indicates the number of iterations, and s indicates the step size. Details can be bound in Section 4.4.

Algorithm	Convergence rate	Stability	Sensitivity
GD	$\mathcal{O}\left((1-s\mu)^t\right)$	$\mathcal{O}\left(1-(1-s\mu)^t\right)$	$\mathcal{O}\left(t(1-c_0\mu)^{t-1}\right)$
NAG	$\mathcal{O}\left(t(1-\sqrt{s\mu})^t\right)$	$\mathcal{O}\left(1-(1-\sqrt{s\mu})^t\right)$	$\mathcal{O}\left(t^3(1-\sqrt{c_0\mu})^t\right)$

GD steps to define algorithm layers in a deep learning model, we can immediately know its robustness, sharpness, etc, based on the stability and sensitivity of GD. Therefore, the techniques and results in algorithm analysis are powerful tools for us to study the representation and generalization ability of hybrid deep learning models containing algorithm layers.

1.2.3 Summary

The duality view connects deep learning models with algorithms from the perspective of their definitions and properties. This viewpoint guides both the methods and analyses presented in this thesis. Based on the duality view, the remainder of this thesis will discuss how we develop concrete methods to combine deep learning and algorithm design to improve both, as well as how we provide theoretical insight for proposed methods.

1.3 Part I: Algorithm Inspired Deep Learning Models

The first main part in this thesis (Part I) presents how the techniques in expert-designed algorithms inspire the development and improvement of deep learning models.

In fact, many well-known neural networks are inspired by hand-designed algorithms. For example, the concepts of kernel and convolution in convolutional neural networks (CNN) were widely used techniques in traditional image processing algorithms. Graph neural networks (GNN) certainly have a similar structure to the Weisfeiler-Lehman (WL) test [10], a classic algorithm designed to determine whether two graphs are isomorphic by iterative aggregations. Many recent works on GNNs still largely borrowed techniques from exiting graph algorithms to improve or to analyze the representation power of GNNs.

Deep learning extends far beyond the reach of computer vision tasks and graph problems. Rather than using fully-connected layers, the community is increasingly interested in domain-specific modeling priors, especially for small-data tasks. These priors integrate specialized knowledge that we have as humans into the model and ideally can be used as

differentiable modules inside a deep learning model. To be more specific, Part I presents 'algorithm inspired deep learning models' from two aspects stated below.

1.3.1 Algorithm layers in deep learning models

Algorithms are powerful modeling tools. As pointed out in the duality view (Section 1.2), each iteration of an algorithm can be viewed as one layer in the neural network. Therefore, an iterative algorithm can be unrolled, truncated, and then used as a specialized module in deep learning models. We call the layers defined by algorithm steps **algorithm layers**. We call the layers defined by highly parameterized and non-interpretable operations (i.e., black-box) **neural layers**. Examples of neural layers include feed-forward layers, convolutional layers, transformer layers, etc. A deep learning model that contains both algorithm layers and neural layers is called a **hybrid model**. If the algorithm layers are differentiable or their gradients can be estimated, the overall hybrid model can be trained end-to-end in the same way as other conventional neural networks.

What motivates the use of algorithm layers in deep learning are many real-world applications that require perception and reasoning to work together to solve a problem. Neural layers play the role of perception to understand and represent the complex information in the inputs, whereas algorithm layers play the role of reasoning by executing prescribed actions to derive outputs that encode specific prior knowledge or satisfy certain constraints. Therefore, there has been a surge of interest in developing hybrid models that contain both neural and algorithm layers in order to tackle more sophisticated learning tasks.

Empirically, this thesis presents the advantage of such hybrid models through an application in computational biology (Chapter 3). Theoretically, this thesis provides rigorous understanding of such hybrid models by analyzing their approximation and generalization abilities (Chapter 4). The analysis has largely used proof techniques for analyzing algorithmic properties from the optimization literature (as introduced in the duality view in Section 1.2).

1.3.2 Dynamic deep learning models inspired by algorithms

Algorithm techniques have inspired the design of many neural networks. To further explores this aspect, this thesis proposes deep learning models with dynamic depth in Chapter 5. This model is inspired by the fact that traditional algorithms can run for a different number of iterations given a different input problem, determined by certain stopping criteria. Nonetheless, most deep learning models, including those algorithm-inspired models, run for a predetermined number of layers for any input. It is natural to wonder whether the depth of a deep architecture should be different for different input instances, similar to the number of iterations in algorithms. This motivates the research in Chapter 5, in which a deep learning model and a variational stopping policy are learned together to sequentially determine the optimal number of layers for each input instance. Intuitively, such a dynamic model can avoid "over-thinking" or compute less for operations converged already. Experimentally, Chapter 5 shows that such a dynamic deep learning model can achieve improved performance on a diverse set of tasks, including learning sparse recovery, few-shot meta learning, and computer vision tasks.

1.4 Part II: Deep Learning Based Algorithm Design

Part II of this thesis discusses how deep learning methods can be used to improve the design of algorithms. This is indeed a broad question which has become an active research topic in recent years. Part II's main focus is on designing learning-based algorithms for solving **statistical optimization** problems.

The conventional design process of a statistical optimization algorithm consists of two steps: **(1) Objective**: formulate an optimization objective to reflect the model assumptions; **(2) Solver**: derive an iterative algorithm (i.e., a solver) to approximate the minimizer of the optimization objective established in Step (1).

The objective and the solver will contribute to the **statistical error** and **optimization**

error respectively. The statistical error is the difference between the minimizer of the objective and the true parameters (i.e., the estimation target). It occurs as a result of potential model misspecifications and unavoidable Bayes error. The optimization error is the difference between the minimizer and the approximation given by the iterative algorithm. The optimization error can be reduced at the expense of a higher computational cost. Therefore, reducing the optimization error is, to some extent, equivalent to reducing the **computational cost** of the solver.

The statistical error, on the other hand, is more difficult to reduce without the access to more observations (i.e., data). In the literature of high-dimensional statistic, *multi-task learning* is proposed to leverage the data in multiple related tasks to help improve the statistical performance of all the tasks. In most traditional algorithms, this is realized by constructing a joint objective for multiple estimation tasks. The joint objective typically encodes the similarities among different tasks by adding some group norms or other regularization terms. However, in many practical problems, we only know that multiple tasks are related, without knowing how they are similar to each other quantitatively. Manually constructing the joint objective may not best reflect the actual similarity.

To go one step farther than multi-task learning, this thesis focuses on using deep learning methods to automatically learn to leverage the similarity between different estimations problems and discover better performing algorithms for solving the target distribution of problems. The following characteristics illustrate intuitively why deep learning can be advantageous in designing statistical optimization algorithms:

1. *Flexibility.* Both the objective and solver can be augmented with neural components. These neural components can provide flexibility for the objective to express more sophisticated, hidden, or non-intuitive models, and for the solver to extract useful patterns from data. Such improved representation power opens up more possibilities for finding a better-fitting objective and a faster solver for domain-specific problems.

2. *Adaptation.* The training of the neural components fills the gap between the known

rules and real-world observations by optimizing the performance on problem instances in the training dataset. As a results, the learned objective and solver can adapt better to the target distribution of problems.

3. *End-to-end.* Unlike traditional algorithms, which design the objective and solver separately, deep learning can handle these two components jointly through end-to-end training. As a result, the learned algorithm can balance statistical and optimization errors under given computing constraints, and result in a higher ultimate accuracy.

In order to verify the benefit of deep learning in algorithm design, this thesis presents both empirical and theoretical results in Chapter 6 and Chapter 7.

1.5 Organization

The remainder of this thesis is structured as follows.

Chapter 2 summarizes two lines of existing works. One is concerned with deep learning models inspired by algorithms (Section 2.1). The other is about designing data-driven algorithms based on deep learning (Section 2.2). Chapter 2 also delves into the challenges of these two research directions that are less explored in existing works.

Part I presents the first portion of research contributions in this thesis. This part shows, both empirically and theoretically, how to use algorithms as modeling priors to improve the performance of deep learning models. More specifically:

- Through an application in computational biology, Chapter 3 demonstrates the advantages of using algorithm layers in deep learning models.
- Chapter 4 provides theoretical understanding of such hybrid deep architectures with algorithm layers, by analyzing their approximation and generalization abilities.
- Chapter 5 presents how to design a dynamic deep learning model inspired by the stopping criteria of algorithms and achieve improved performance on various tasks.

Part II presents the second portion of research contributions in this thesis. This part of

the research shows how data-driven algorithms are learned using deep learning techniques. More specifically:

- To learn algorithms for estimating the precision matrix of Gaussian graphical models, Chapter 6 introduces a deep learning model (alternatively, a parameterized algorithm) which is designed based on unrolling an alternating minimization algorithm.
- To provide theoretical guarantees for learning based algorithms, Chapter 7 presents `PLISA` (`P`rovable `L`earning-based `I`terative `S`parse recovery `A`lgorithm). It is a provable learning based algorithm for sparse recovery, whose achievable accuracy and generalization ability are characterized by our theoretical analysis.

CHAPTER 2

LITERATURE REVIEW

This chapter summarizes two lines of existing works. Section 2.1 summarizes researches that used traditional algorithms as modeling priors to design the architecture of deep learning models. Section 2.2 summarizes researches that employ deep learning techniques to automatically learn data-driven algorithms. It is worth noting that these two lines of works are inherently highly related, though stated in two separate sections in this chapter.

2.1 Algorithm Inspired Deep Learning Models

In machine learning, algorithms play the role of optimizing the model during the training process. In recent years, their secondary role as modeling priors in deep learning models has received a lot of attention. These algorithm inspired components integrate domain-specific knowledge into the deep learning model and ideally can be used as differentiable modules in the architecture.

2.1.1 Existing works

Depending on the specific tasks, different algorithms are used in different deep learning models. The scope of this thesis does not allow for a full discussion of all such models. The following are a few examples.

- Differentiable beam search [11] is used as a module in neural sequence models.
- Dynamic programming algorithms are turned into differentiable operators and used as a layer in the neural network [12, 13].
- Differentiable maximum satisfiability (MAXSAT) solver [14] are integrated into the deep learning model to learn the logical structure of challenging problems.

• Differentiable power iterations for eigendecomposition are used in deep learning models for applications including PCA denoising [15] and graph matching [16].

• Optimal transport algorithms are used in deep learning for learning to sort [17].

• Optimization algorithms are used as deep priors in many works for a wide range of applications [18, 19, 20, 21, 22, 23] including generic classification, structured prediction [24, 25, 26], denoising [27], graph matching [28], and for stabilizing the convergence generative adversarial networks [29], etc.

• Many studies have been done in order to compute the gradients of various algorithms [12, 14, 15, 21, 22].

These recent advances highlighted the benefits of using algorithm layers in deep learning models, most notably improved data efficiency and model interpretability. Section 3.2, Section 4.2, and Section 5.2 in Part I mention a few other related works.

2.1.2 Less explored topics

In contrast to the plethora of empirical studies on algorithm inspired deep learning models, the theoretical underpinning of such architectures remains largely unexplored. Many important questions have yet to be answered theoretically. For instance,

• What is the benefit of using algorithm layers compared to conventional deep architectures such as recurrent neural networks (RNN)?

• How exactly will the algorithm layers affect the data efficiency (or generalization ability) of the deep learning model?

• For different algorithms which can solve the same task, what are their differences when used as algorithm layers in deep models?

• How will the algorithmic properties such as the convergence rate and stability of an algorithm affect the learning behavior of deep architectures containing layers defined by the algorithm?

Chapter 4 in this thesis will provide some theoretical insights for answering these questions.

Furthermore, determining the number of algorithm steps (or layers) used in the architecture can sometimes be difficult. Unlike traditional algorithms, which decide the number of algorithm steps based on specific stopping criteria, the number of layers in deep architecture is often prefixed. Using too many layers may be inefficient, whereas using too few layers may be insufficient.

Chapter 5 in this thesis is primarily concerned with addressing the aforementioned challenges.

2.2 Deep Learning Based Algorithm Design

The fact that similar problems may need to be solved repeatedly in practice encourages the use of machine learning methods to design data-driven algorithms automatically. Learning to learn, or learning to optimize, has become an emerging approach that learns an algorithm for solving a specific distribution of problems. In the learning to learn setting, an algorithm is viewed as a function mapping, which takes the problem to be solved as the input and outputs the solution. Based on this view, one can construct a deep learning model to represent a parameterized algorithm, and learn the parameters by optimizing its performance on a set of training problem instances via gradient-based methods. Intuitively, the learned algorithm may be better adapted to the problem distribution of interest as a result of the use of training problems and gradient-based parameter search, and so has a higher accuracy and efficiency when solving these problems.

2.2.1 Existing works

Learning to learn, or learning to optimize, has been applied to various domains.

For combinatorial optimization problems like mixed integer programming, [30] learns branching rules on branching tree by self improvement imitation learning; [31] uses a graph convolution network to mimic strong branching on constraint graphs; and [32] shows how to use reinforcement learning to select the cutting plane.

The concept of learning to learn is also applied to online learning. [33, 34, 35, 36, 37] demonstrate how, both practically and theoretically, applying machine learning approaches to predict future input might improve the performance of online learning algorithms.

A substantial body of works has focused on learning continuous optimization algorithms. Deep learning models proposed for continuous optimization algorithms fall into two main categories: (i) *black-boxed architectures* such as LSTM [38], which attempt to learn totally new update rules; and (ii) *algorithm-unrolling based architectures*, which are designed based on the structure of existing traditional algorithms. In further detail, an algorithm-unrolling-based design means that each step of a traditional algorithm becomes a neural network layer whose parameters may be learnt from data. Certain operations in the algorithm step can be more flexible than the traditional one, but the deep learning model retains the overall structure of the algorithm steps.

Algorithm unrolling is a favorable design choice in many works [39, 40, 41, 42, 43, 44] because (i) it allows us to better employ traditional optimization techniques to design and understand the neural algorithm; and that (ii) it often results in a neural algorithm with fewer parameters. Therefore, the models studied in this thesis mainly fall into this category.

A well-known example is LISTA [45] which interprets the classic algorithm ISTA [46] as layers of neural networks and has since then been an active research topic [47, 48, 49, 50, 51]. Many other algorithms have also been used as the basis of deep architectures. Examples include ADMM [39, 43], approximate message passing [40, 52], Frank-Wolfe algorithm [53, 54], PDE/ODE solvers [55, 56], etc. We will refer the audience to [57, 58] for a more comprehensive summary of related works in this area.

2.2.2 Less explored topics

The most investigated topic in this area is learning algorithms for solving compressed sensing (or sparse coding) problems with a fixed design matrix. However, other more sophisticated algorithm learning problems, especially theoretical studies, have received less

attention.

Furthermore, existing theoretical efforts primarily focus on analyzing the convergence rate achievable by the neural algorithm [48, 49, 59, 50], but the generalization error bound has received less attention so far. A considerable body of work merely claims intuitively that algorithm unrolling architectures can generalize well due to their minimal number of parameters. The second direction to be investigated is how to establish a rigorous and specialized theoretical argument for the generalization ability of algorithm unrolling based architectures.

The second part of this thesis (Part II) is primarily concerned with addressing the aforementioned unexplored directions.

Part I

Algorithm Inspired Deep Learning Model

CHAPTER 3

EMPIRICAL STUDY: RNA SECONDARY STRUCTURE PREDICTION BY LEARNING UNROLLED ALGORITHMS

This Chapter presents an empirical study on using algorithm layers in the deep architecture. A deep learning model for RNA secondary structure prediction is proposed and called E2Efold (end-to-end fold). The key idea underneath E2Efold is to use an unrolled algorithm for solving a constrained optimization problem in the deep architecture to integrate some RNA base-pairing constraints discovered by biology experts. Experiments have shown that a design like this allows E2Efold to predict structures considerably better while being efficient.

The research in this chapter was previously presented at the ICLR 2020 conference [26].

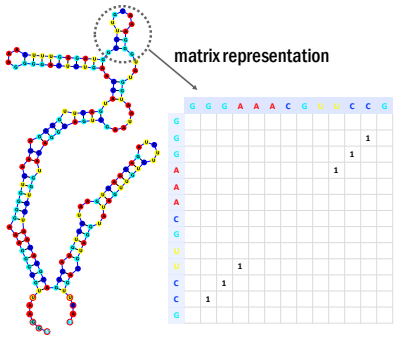
3.1 Introduction

Ribonucleic acid (RNA) is a molecule playing essential roles in numerous cellular processes and regulating expression of genes [60]. It consists of an ordered sequence of nucleotides, with each nucleotide containing one of four bases: *Adenine (A)*, *Guanine (G)*, *Cytosine (C)* and *Uracile (U)*. This sequence of bases can be represented as

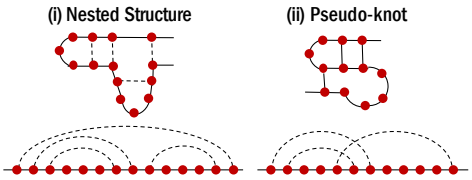
$$\boldsymbol{x} := (x_1, \ldots, x_L) \text{ where } x_i \in \{A, G, C, U\},$$

which is known as the *primary structure* of RNA. The bases can bond with one another to form a set of base-pairs, which defines the *secondary structure*. A secondary structure can be represented by a matrix A^* where $A^*_{ij} = 1$ if the i, j-th bases are paired (Fig 3.1a).

Discovering the secondary structure of RNA is important for understanding functions



(a) Graph and matrix representations of RNA secondary structure.



(b) Nested and non-nested structures.

of RNA. Because experimental assays are expensive, computational prediction of RNA secondary structure has become an important task in RNA research.

Research on computational prediction of RNA secondary structure from knowledge of primary structure has been carried out for decades. Most existing methods assume the secondary structure is a result of energy minimization, i.e., $A^* = \arg\min_A E_{\boldsymbol{x}}(A)$. The energy function is either estimated by physics-based thermodynamic experiments [61, 62] or learned from data [63]. These approaches are faced with a common problem that the search space of *all valid secondary structures* is exponentially-large with respect to the length L of the sequence. To make the minimization tractable, it is often assumed the base-pairing has a *nested* structure (Fig 3.1b left), and the energy function factorizes pairwisely. With this assumption, dynamic programming (DP) based algorithms can iteratively find the optimal structure for subsequences and thus consider an enormous number of structures in time $\mathcal{O}(L^3)$. However, they restrict the search space to *nested structures*, which excludes some valid yet biologically important RNA secondary structures that contain '*pseudoknots*', i.e., elements with at least two non-nested base-pairs (Fig 3.1b right). Pseudoknots make up roughly 1.4% of base-pairs [64], and are overrepresented in functionally important regions [65, 66]. Furthermore, pseudoknots are present in around 40% of the RNAs. They also assist folding into 3D structures [67] and thus should not be ignored. To predict RNA structures with pseudoknots, energy-based methods need to run more computationally intensive algorithms to decode the structures.

In summary, in the presence of more complex structured output (i.e., pseudoknots), it is challenging for energy-based approaches to simultaneously take into account the complex constraints while being efficient. Therefore, we adopt a different viewpoint by assuming that the secondary structure is the output of a deep learning model, i.e., $A^* = \mathcal{F}_\theta(\boldsymbol{x})$, and propose to learn the parameters θ from data in an end-to-end fashion. It avoids the second minimization step needed in energy function based approach, and does not require the output structure to be nested. Furthermore, the feed-forward model can be fitted by directly optimizing the loss that one is interested in.

Despite the above advantages of using a feed-forward model, the architecture design is challenging. To be more concrete, in the RNA case, $\mathcal{F}_\theta$ is difficult to design for the following reasons:

(i) RNA secondary structure needs to obey certain *hard constraints* (see details in Section 3.3), which means certain kinds of pairings cannot occur at all [68]. Ideally, the output of $\mathcal{F}_\theta$ needs to satisfy these constraints.

(ii) The number of RNA data points is limited, so we cannot expect that a naive fully connected network can learn the predictive information and constraints directly from data. Thus, inductive biases need to be encoded into the network architecture.

(iii) One may take a two-step approach, where a post-processing step can be carried out to enforce the constraints when $\mathcal{F}_\theta$ predicts an invalid structure. However, in this design, the deep network trained in the first stage is unaware of the post-processing stage, making less effective use of the potential prior knowledge encoded in the constraints.

Faced with these challenges, we propose a hybrid deep learning model called **E2Efold** in this chapter. It contains both neural layers and algorithm layers, called *Deep Score Module* and *Algorithm Module* respectively in this chapter. *Deep Score Module* is a transformer-based deep model which encodes sequence information useful for structure prediction. *Algorithm Module* is the second part of E2Efold, which gradually enforces the constraints

and restrict the output space. It is designed based on an unrolled algorithm for solving a constrained optimization. These two networks are coupled together and learned jointly in an end-to-end fashion. Therefore, we call this model E2Efold.

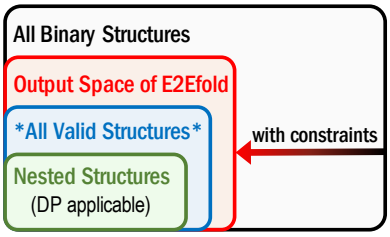


Figure 3.2: Output space of E2Efold.

By using an iterative algorithm for solving constrained optimization to design the Algorithm Module, the output space of E2Efold is constrained (illustrated in Fig 3.2), which makes it easier to learn a good model in the case of limited data and also reduces the overfitting issue. Yet, the constraints encoded in E2Efold are flexible enough such that pseudoknots are not excluded. In summary, E2Efold strikes a nice balance between model biases for learning and expressiveness for valid RNA structures.

We conduct extensive experiments to compare E2Efold with state-of-the-art (SOTA) methods on several RNA benchmark datasets, showing superior performance of E2Efold including:

- being able to predict valid RNA secondary structures including pseudoknots;
- running as efficient as the fastest algorithm in terms of inference time;
- producing structures that are visually close to the true structure;
- better than previous SOTA in terms of F1 score, precision and recall.

Although in this chapter we focus on RNA secondary structure prediction, which presents an important and concrete problem where E2Efold leads to significant improvements, our method is generic and can be applied to other problems where constraints need to be enforced or prior knowledge is provided. We imagine that our design idea of learning unrolled algorithm to enforce constraints can also be transferred to problems such as protein folding and natural language understanding problems (e.g., building correspondence structure

between different parts in a document).

3.2 Related Work

Learning-based RNA folding methods such as ContraFold [63] and ContextFold [69] have been proposed for energy parameters estimation due to the increasing availability of known RNA structures, resulting in higher prediction accuracies, but these methods still rely on the above DP-based algorithms for energy minimization. A recent deep learning model, CDPfold [70], applied convolutional neural networks to predict base-pairings, but it adopts the dot-bracket representation for RNA secondary structure, which can not represent pseudoknotted structures. Moreover, it requires a DP-based post-processing step whose computational complexity is prohibitive for sequences longer than a few hundreds.

Learning with differentiable algorithm layers is a useful idea that has sparked a series of works (See Chapter 2). Some models are also applied to structured prediction problems [25, 71, 72], but they did not consider the challenging RNA secondary structure problem or discuss how to properly incorporating constraints into the architecture. OptNet [27] integrates constraints by differentiating KKT conditions, but it has cubic complexity in the number of variables and constraints, which is prohibitive for the RNA case.

3.3 Hard Constraints in RNA Secondary Structure

In the RNA secondary structure prediction problem, the input is the ordered sequence of bases $\boldsymbol{x} = (x_1, \ldots, x_L)$ and the output is the RNA secondary structure represented by a matrix $A^* \in \{0, 1\}^{L \times L}$. Hard constraints on the forming of an RNA secondary structure dictate that certain kinds of pairings cannot occur [68]. Formally, these constraints are listed in Table 3.1.

(i) and (ii) prevent pairing of certain base-pairs based on their types and relative locations. Incorporating these two constraints can help the model exclude lots of illegal pairs. (iii) is a global constraint among the entries of A^*.

Table 3.1: Constraints of RNA secondary structures

(i) Only 3 combinations of nucleotides, $\mathcal{B} := \{AU, UA\} \cup \{GC, CG\} \cup \{GU, UG\}$, can form base-pairs.	$\forall i, j$, if $x_i x_j \notin \mathcal{B}$, then $A_{ij} = 0$.
(ii) No sharp loops are allowed.	$\forall \lvert i - j \rvert < 4$, $A_{ij} = 0$.
(iii) There is no overlap of pairs, i.e., it is a matching.	$\forall i, \sum_{j=1}^{L} A_{ij} \leq 1$.

The space of all valid secondary structures contains all *symmetric* matrices $A \in \{0, 1\}^{L \times L}$ that satisfy the above three constraints. This space is much smaller than the space of all binary matrices $\{0, 1\}^{L \times L}$. Therefore, if we could incorporate these constraints in our deep model, the reduced output space could help us train a better predictive model with less training data. We do this by using an unrolled algorithm as the inductive bias to design deep architecture.

3.4 E2Efold: Deep Learning Model With Algorithm Layers

In the literature on feed-forward networks for structured prediction, most models are designed using traditional deep learning architectures. However, for RNA secondary structure prediction, directly using these architectures does not work well due to the limited amount of RNA data points and the hard constraints on forming an RNA secondary structure. These challenges motivate the design of our E2Efold deep model, which combines a *Deep Score Module* with a *Algorithm Module* based on an unrolled algorithm for solving a constrained optimization problem.

3.4.1 Neural layers: deep score module

The first part of E2Efold is a *Deep Score Module* $U_\theta(\boldsymbol{x})$ whose output is an $L \times L$ symmetric matrix. Each entry of this matrix, i.e., $U_\theta(\boldsymbol{x})_{ij}$, indicates the score of nucleotides x_i and x_j being paired. The $\boldsymbol{x}$ input to the network here is the $L \times 4$ dimensional one-hot embedding. The specific architecture of U_θ is shown in Fig 3.3. It mainly consists of

- a position embedding matrix $\boldsymbol{P}$ which distinguishes $\{x_i\}_{i=1}^{L}$ by their positions: $\boldsymbol{P}_i =$

$\text{MLP}\big(\psi_1(i), \ldots, \psi_\ell(i), \psi_{\ell+1}(i/L), \ldots, \psi_n(i/L)\big)$, where $\{\psi_j\}$ is a set of n feature maps such as $\sin(\cdot)$, $\text{sigmoid}(\cdot)$, etc, and $\text{MLP}(\cdot)$ denotes multi-layer perceptions.

- a stack of Transformer Encoders [73] which encode the sequence information and the global dependency between nucleotides;
- a 2D Convolution layers [74] for outputting the pairwise scores.

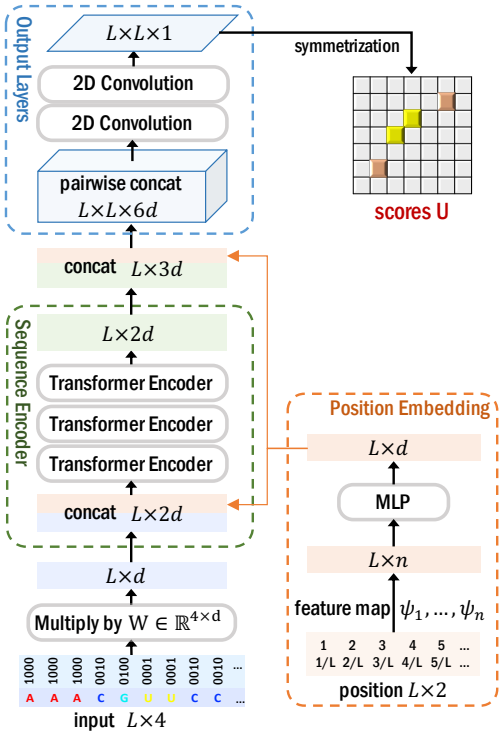


Figure 3.3: Architecture of the *Deep Score Module* (neural layers).

With the representation power of neural networks, the hope is that we can learn an informative U_θ such that higher scoring entries in $U_\theta(\boldsymbol{x})$ correspond well to actual paired bases in RNA structure. Once the score matrix $U_\theta(\boldsymbol{x})$ is computed, a naive approach to use it is to choose an offset term $s \in \mathbb{R}$ (e.g., $s = 0$) and let $A_{ij} = 1$ if $U_\theta(\boldsymbol{x})_{ij} > s$. However, such entry-wise independent predictions of A_{ij} may result in a matrix A that violates the constraints for a valid RNA secondary structure. Therefore, a post-processing step is needed to make sure the predicted A is valid. This step could be carried out separately after U_θ is learned. But such decoupling of base-pair scoring and post-processing for constraints may lead to sub-optimal results, where the errors in these two stages can not be considered

together and tuned together. Instead, we will introduce a Algorithm Module which can be trained end-to-end together with U_θ to enforce the constraints.

3.4.2 Algorithm layers: unrolled constrained optimization solver

The second part of E2Efold is a *Algorithm Module* (i.e., algorithm layers) Alg_ϕ which is an unrolled and parameterized algorithm for solving a constrained optimization problem. We first present how we formulate the constrained optimization and the algorithm for solving it.

Formulation of constrained optimization. Given the scores predicted by $U_\theta(\boldsymbol{x})$, we define the total score $\frac{1}{2}\sum_{i,j}(U_\theta(\boldsymbol{x})_{ij} - s)A_{ij}$ as the objective to maximize, where s is an offset term. Clearly, without structure constraints, the optimal solution is to take $A_{ij} = 1$ when $U_\theta(\boldsymbol{x})_{ij} > s$. Intuitively, the objective measures the covariation between the entries in the scoring matrix and the A matrix. With constraints, the exact maximization becomes intractable. To make it tractable, we consider a convex relaxation of this discrete optimization to a continuous one by allowing $A_{ij} \in [0, 1]$. Consequently, the solution space that we consider to optimize over is

$$\mathcal{A}(\boldsymbol{x}) := \left\{A \in [0,1]^{L\times L} \mid A \text{ is symmetric and satisfies constraints (i)-(iii) in Section 3.3}\right\}.$$

To further simplify the search space, we define a nonlinear transformation $\mathcal{T}$ on $\mathbb{R}^{L\times L}$ as $\mathcal{T}(\hat{A}) := \frac{1}{2}\big(\hat{A}\circ\hat{A} + (\hat{A}\circ\hat{A})^\top\big)\circ M(\boldsymbol{x})$, where $\circ$ denotes element-wise multiplication. Matrix M is defined as $M(\boldsymbol{x})_{ij} := 1$ if $x_i x_j \in \mathcal{B}$ and also $|i-j| \geq 4$, and $M(\boldsymbol{x})_{ij} := 0$ otherwise. From this definition we can see that $M(\boldsymbol{x})$ encodes both constraint (i) and (ii). With transformation $\mathcal{T}$, the resulting matrix is non-negative, symmetric, and satisfies constraint (i) and (ii). Hence, by defining $A := \mathcal{T}(\hat{A})$, the solution space is simplified as $\mathcal{A}(\boldsymbol{x}) = \{A = \mathcal{T}(\hat{A}) \mid \hat{A} \in \mathbb{R}^{L\times L}, A\mathbf{1} \leq \mathbf{1}\}$.

Finally, we introduce a ℓ_1 penalty term $\|\hat{A}\|_1 := \sum_{i,j}|\hat{A}_{ij}|$ to make A sparse and formu-

late the post-processing step as: ($\langle \cdot, \cdot \rangle$ denotes matrix inner product, i.e., sum of entry-wise multiplication)

$$\max_{\hat{A} \in \mathbb{R}^{L \times L}} \frac{1}{2} \left\langle U_\theta(\boldsymbol{x}) - s, A := \mathcal{T}(\hat{A}) \right\rangle - \rho \|\hat{A}\|_1 \quad \text{s.t. } A\mathbf{1} \leq \mathbf{1} \tag{3.1}$$

The advantages of this formulation are that the variables $\hat{A}_{ij}$ are free variables in $\mathbb{R}$ and there are only L inequality constraints $A\mathbf{1} \leq \mathbf{1}$. This system of linear inequalities can be replaced by a set of nonlinear equalities $\text{relu}(A\mathbf{1} - \mathbf{1}) = \mathbf{0}$ so that the constrained problem can be easily transformed into an unconstrained problem by introducing a Lagrange multiplier $\boldsymbol{\lambda} \in \mathbb{R}_+^L$:

$$\min_{\boldsymbol{\lambda} \geq \mathbf{0}} \max_{\hat{A} \in \mathbb{R}^{L \times L}} \underbrace{\frac{1}{2} \langle U_\theta(\boldsymbol{x}) - s, A \rangle - \langle \boldsymbol{\lambda}, \text{relu}(A\mathbf{1} - \mathbf{1}) \rangle}_{f} - \rho \|\hat{A}\|_1. \tag{3.2}$$

Algorithm for solving it. We use a primal-dual method for solving Eq. 3.2. In each iteration, $\hat{A}$ and $\boldsymbol{\lambda}$ are updated alternatively by:

$$\text{(primal) gradient step:} \quad \dot{A}_{t+1} \leftarrow \hat{A}_t + \alpha \cdot \gamma_\alpha^t \cdot \hat{A}_t \circ M(\boldsymbol{x}) \circ \Big(\partial f / \partial A_t + (\partial f / \partial A_t)^\top \Big), \tag{3.3}$$

$$\text{where} \begin{cases} \partial f / \partial A_t = \frac{1}{2}(U_\theta(\boldsymbol{x}) - s) - (\boldsymbol{\lambda} \circ \text{sign}(A_t\mathbf{1} - \mathbf{1}))\, \mathbf{1}^\top, \\ \text{sign}(c) := 1 \text{ when } c > 0 \text{ and } 0 \text{ otherwise}, \end{cases} \tag{3.4}$$

$$\text{(primal) soft threshold:} \quad \hat{A}_{t+1} \leftarrow \text{relu}(|\dot{A}_{t+1}| - \rho \cdot \alpha \cdot \gamma_\alpha^t), \quad A_{t+1} \leftarrow \mathcal{T}(\hat{A}_{t+1}), \tag{3.5}$$

$$\text{(dual) gradient step:} \quad \boldsymbol{\lambda}_{t+1} \leftarrow \boldsymbol{\lambda}_{t+1} + \beta \cdot \gamma_\beta^t \cdot \text{relu}(A_{t+1}\mathbf{1} - \mathbf{1}), \tag{3.6}$$

where α, β are step sizes and $\gamma_\alpha, \gamma_\beta$ are decaying coefficients. When it converges at T, an approximate solution $Round\big(A_T = \mathcal{T}(\hat{A}_T)\big)$ is obtained.

Algorithm Layers. The *Algorithm Module*, denoted by Alg_ϕ, is designed based on the above algorithm. The specific computation graph of Alg_ϕ is given in Algorithm 1, whose main component is a recurrent cell which we call AlgCell_ϕ. The computation graph is almost the same as the iterative update from Eq. 3.3 to Eq. 3.6, except for several modifications:

- *(learnable hyperparameters)* The hyperparameters including step sizes α, β, decaying rate $\gamma_\alpha, \gamma_\beta$, sparsity coefficient ρ and the offset term s are treated as learnable parameters in ϕ, so that there is no need to tune the hyperparameters by hand but automatically learn them from data instead.
- *(fixed # iterations)* Instead of running the iterative updates until convergence, AlgCell_ϕ is applied recursively for T iterations where T is a manually fixed number. This is why in Fig 3.2 the output space of E2Efold is slightly larger than the true solution space.
- *(smoothed sign function)* Resulted from the gradient of $\text{relu}(\cdot)$, the update step in Eq. 3.4 contains a $\text{sign}(\cdot)$ function. However, to push gradient through Alg_ϕ, we require a differentiable update step. Therefore, we use a smoothed sign function defined as $\text{softsign}(c) := 1/(1 + \exp(-kc))$, where k is a temperature.
- *(clip $\hat{A}$)* An additional step, $\hat{A} \leftarrow \min(\hat{A}, 1)$, is included to make the output A_t at each iteration stay in the range $[0, 1]^{L \times L}$.

Algorithm 1: $\text{Alg}_\phi(U, M)$

Parameters $\phi := \{w, s, \alpha, \beta, \gamma_\alpha, \gamma_\beta, \rho\}$

$U \leftarrow \text{softsign}(U - s) \circ U$

$\hat{A}_0 \leftarrow \text{softsign}(U - s) \circ \text{sigmoid}(U)$

$A_0 \leftarrow \mathcal{T}(\hat{A}_0)$

$\boldsymbol{\lambda}_0 \leftarrow w \cdot \text{relu}(A_0\mathbf{1} - \mathbf{1})$

For $t = 0, \ldots, T-1$ **do**

  $\boldsymbol{\lambda}_{t+1}, A_{t+1}, \hat{A}_{t+1}$

  $\leftarrow \text{AlgCell}_\phi(U, M, \boldsymbol{\lambda}_t, A_t, \hat{A}_t, t)$

return $\{A_t\}_{t=1}^T$

Algorithm 2: AlgCell_ϕ

Function $\texttt{AlgCell}_\phi(U, M, \boldsymbol{\lambda}, A, \hat{A}, t)$ **:**

  $G \leftarrow \frac{1}{2}U - (\boldsymbol{\lambda} \circ \text{softsign}(A\mathbf{1} - \mathbf{1}))\,\mathbf{1}^\top$

  $\dot{A} \leftarrow \hat{A} + \alpha \cdot {\gamma_\alpha}^t \cdot \hat{A} \circ M \circ (G + G^\top)$

  $\hat{A} \leftarrow \text{relu}(|\dot{A}| - \rho \cdot \alpha \cdot {\gamma_\alpha}^t)$

  $\hat{A} \leftarrow 1 - \text{relu}(1 - \hat{A})$ [i.e.,$\min(\hat{A}, 1)$]

  $A \leftarrow \mathcal{T}(\hat{A})$

  $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \beta \cdot {\gamma_\beta}^t \cdot \text{relu}(A\mathbf{1} - \mathbf{1})$

  return $\boldsymbol{\lambda}, A, \hat{A}$

With these modifications, the Algorithm Module Alg_ϕ is a tuning-free and differentiable unrolled algorithm with meaningful intermediate outputs. Combining it with the Deep Score Module, the final deep model is

$$\textbf{E2Efold}: \quad \{A_t\}_{t=1}^T = \overbrace{\text{Alg}_\phi(\underbrace{U_\theta(\boldsymbol{x})}_{\textit{Deep Score Module}}, M(\boldsymbol{x}))}^{\textit{Algorithm Module}}. \tag{3.7}$$

3.5 Training Algorithm

Given a dataset $\mathcal{D}$ containing examples of input-output pairs $(\boldsymbol{x}, A^*)$, the training procedure of E2Efold is similar to standard gradient-based supervised learning. However, for RNA secondary structure prediction problems, commonly used metrics for evaluating predictive performances are F1 score, precision and recall, which are non-differentiable.

Differentiable F1 Loss. To directly optimize these metrics, we mimic true positive (TP), false positive (FP), true negative (TN) and false negative (FN) by defining continuous functions on $[0, 1]^{L \times L}$:

$$\text{TP} = \langle A, A^* \rangle,\ \text{FP} = \langle A, 1 - A^* \rangle,\ \text{FN} = \langle 1 - A, A^* \rangle,\ \text{TN} = \langle 1 - A, 1 - A^* \rangle.$$

Since $\text{F1} = 2\text{TP}/(2\text{TP} + \text{FP} + \text{FN})$, we define a loss function to mimic the negative of F1 score as:

$$\mathcal{L}_{-\text{F1}}(A, A^*) := -2\langle A, A^*\rangle / \left(2\langle A, A^*\rangle + \langle A, 1 - A^*\rangle + \langle 1 - A, A^*\rangle\right). \tag{3.8}$$

Assuming that $\sum_{ij} A^*_{ij} \neq 0$, this loss is well-defined and differentiable on $[0, 1]^{L\times L}$.

It is notable that this F1 loss takes advantages over other differentiable losses including ℓ_2 and cross-entropy losses, because there are much more negative samples (i.e. $A_{ij} = 0$) than positive samples (i.e. $A_{ij} = 1$).

Overall Loss Function. As noted earlier, E2Efold outputs a matrix $A_t \in [0, 1]^{L\times L}$ in each iteration. This allows us to add auxiliary losses to regularize the intermediate results, guiding it to learn parameters which can generate a smooth solution trajectory. More specifically, we use an objective that depends on the entire trajectory of optimization:

$$\min_{\theta,\phi} \frac{1}{|\mathcal{D}|} \sum_{(x,A^*)\in\mathcal{D}} \frac{1}{T} \sum_{t=1}^{T} \gamma^{T-t} \mathcal{L}_{-\text{F1}}(A_t, A^*), \tag{3.9}$$

where $\{A_t\}_{t=1}^T = \text{PP}_\phi(U_\theta(\boldsymbol{x}), M(\boldsymbol{x}))$ and $\gamma \leq 1$ is a discounting factor. Empirically, we find it very useful to pre-train U_θ using logistic regression loss. Also, it is helpful to add this additional loss to Eq. 3.9 as a regularization.

3.6 Experiments

The full experimental results can be found in [26][1]. Here we only report the result of ablation study which explicitly reveals the effectiveness of the Algorithm Module in E2Efold.

To exam whether integrating the two modules by pushing gradient through the Algorithm Module is useful, we conduct an ablation study (Table 3.2). We test the performance of training of Deep Score Network U_θ alone. After U_θ is learned, we apply the algorithm

[1] the codes for reproducing the experimental results are released at https://github.com/ml4bio/e2efold.

Table 3.2: Ablation study (RNAStralign test set)

Method	Prec	Rec	F1	Prec(S)	Rec(S)	F1(S)
E2Efold	**0.866**	**0.788**	**0.821**	**0.880**	**0.798**	**0.833**
U_θ+PP	0.755	0.712	0.721	0.782	0.737	0.752

steps for solving the augmented Lagrangian to post-process the outputs of U_θ (thus the notation "$U_\theta+$PP" in Table 3.2). Although "$U_\theta+$PP" performs decently well, with constraints incorporated into training, E2Efold still has significant advantages over it.

CHAPTER 4

THEORETICAL STUDY: UNDERSTANDING DEEP ARCHITECTURE WITH ALGORITHM LAYERS

This chapter discusses some theoretical findings related to hybrid deep architectures that include both neural module and algorithm module.

Similar to the model introduced in Chapter 3, there has been a recent surge of interest in combining deep learning models with algorithm layers in order to handle more sophisticated learning tasks. In many cases, a reasoning task can be solved by an iterative algorithm. This algorithm is often unrolled, and used as a specialized layer in the deep architecture, which can be trained end-to-end with other neural components (Fig. 4.1).

Designing hybrid models that contain both neural layers (i.e., neural module) and algorithm layers (i.e., algorithm module) has led to many empirical successes (see Chapter 2 for a brief review). However, the theoretical foundation of such architectures, especially the interplay between algorithm layers and other neural layers, remains largely unexplored. In this chapter, we take an initial step towards an understanding of such hybrid deep architectures by showing that properties of the algorithm layers, such as convergence, stability and sensitivity, are intimately related to the approximation and generalization abilities of the end-to-end model. Furthermore, our analysis closely matches our experimental findings under a variety of conditions, suggesting that our theory can provide useful guidelines for designing deep architectures with algorithm layers.

The research in this chapter was previously presented at the NeuRIPs 2020 conference in [75].

4.1 Introduction

4.1.1 Motivation and challenges

Many real-world applications necessitate the collaboration of perception and reasoning in order to solve a problem. Perception is the ability to understand and represent inputs, whereas reasoning is the ability to follow prespecified steps and derive answers that meet certain constraints. To tackle such sophisticated learning tasks, recently, there has been a surge of interests in combining deep perception models with algorithm layers.

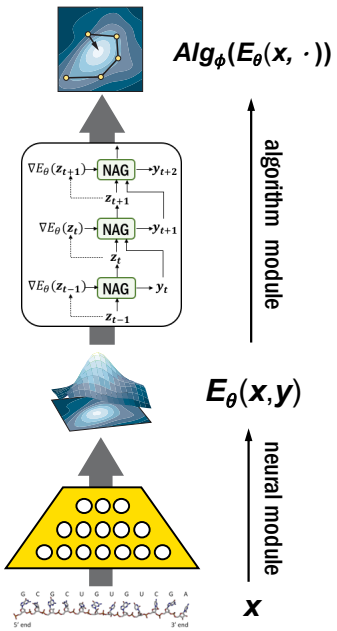


Figure 4.1: Hybrid deep architecture.

Typically, a **algorithm module** (i.e., algorithm layers) is stacked on top of a **neural module** (i.e., neural layers), and treated as an additional layer of the overall deep architecture; then all the parameters in the architecture are optimized end-to-end with loss gradients (Fig 4.1). Very often these algorithm modules can be implemented as unrolled *iterative algorithms*, which can solve more sophisticated tasks with carefully designed and interpretable operations. See Chapter 2 for a brief review of deep learning models that use differentiable algorithm layers in the architecture for various learning tasks.

While these previous works have demonstrated the effectiveness of combining deep learning with algorithm layers, the theoretical underpinning of such hybrid deep architectures remains largely unexplored. For instance, what is the benefit of using a algorithm module based on unrolled algorithms compared to generic architectures such as recurrent neural networks (RNN)? How exactly will the algorithm module affect the generalization ability of the deep architecture? For different algorithms which can solve the same task, what are their differences when used as algorithm modules in deep models? Despite the rich literature on rigorous analysis of algorithmic properties, there is a paucity of work leveraging these analyses to formally study the learning behavior of deep architectures containing algorithm layers. This motivates us to ask the crucial and timely question of

> *How will the algorithmic properties of algorithm layers affect the learning behavior of deep architectures containing such layers?*

In this chapter, we provide a first step towards an answer to this question by analyzing the approximation and generalization abilities of such hybrid deep architectures. To the best our knowledge, such an analysis has not been done before and faces several difficulties:

1. The analysis of certain algorithmic properties such as convergence can be complex by itself;
2. Models based on highly structured iterative algorithms have rarely been analyzed before;
3. The bound needs to be sharp enough to match empirical observations. In this new setting, the complexities of the algorithm's analysis and generalization analysis are intertwined together, making the analysis even more challenging.

4.1.2 Summary of results

We find that standard Rademacher complexity analysis, widely used for neural networks [8, 76, 77], is insufficient for explaining the behavior of these hybrid architectures. Thus we

turn to a more refined local Rademacher complexity analysis [78, 79], which yields the following results:

• **Relation to algorithmic properties.** algorithmic properties such as convergence, stability and sensitivity all play important roles in the generalization ability of the hybrid architecture. Generally speaking, an algorithm module that is faster converging, more stable and less sensitive will be able to better approximate the joint perception and reasoning task, while at the same time generalize better.

• **Which algorithm?** There is a tradeoff that a faster converging algorithm has to be less stable [80]. Therefore, depending on the precise setting, the best choice of algorithm layer may be different. Our theorem reveals that when the neural module is over- or under-parameterized, stability of the algorithm module can be more important than its convergence; but when the neural module has an 'about-right' parameterization, a faster converging algorithm layer may give a better generalization.

• **What depth?** With deeper algorithm layers, the representation ability gets better, but the generalization becomes worse if the neural module is over/under-parameterized. Only when it has 'about-right' complexity, deeper algorithm layers can induce both better representation and generalization.

• **What if RNN?** It has been shown that RNN (or graph neural networks, GNN) can represent reasoning and iterative algorithms [38, 77]. On the example of RNN we demonstrate in Sec 4.6.4 that these generic algorithm modules can also be analyzed under our framework, revealing that RNN layers induce better representation but worse generalization compared to traditional algorithm layers.

• **Experiments.** We conduct empirical studies to validate our theory and show that it matches well with experimental observations under various conditions. These results suggest that our theory can provide useful practical guidelines for designing deep architectures with algorithm layers.

Contributions and limitations. To the best of our knowledge, this is the first result

to quantitatively characterize the effects of algorithmic properties on the learning behavior of hybrid deep architectures with algorithm layers, showing that algorithm biases can help reduce sample complexity of such architectures. Our result also reveals a subtle and previously unknown interplay between algorithm convergence, stability and sensitivity when affecting model generalization, and thus provides design principles for deep architectures with algorithm layers. To simplify the analysis, our initial study is limited to a setting where the algorithm module is an unconstrained optimization algorithm and the neural module outputs a quadratic energy function. However, our analysis framework can be extended to more complicated cases and the insights can be expected to apply beyond our current setting.

4.2 Related Theoretical Works

Our analysis borrows proof techniques for analyzing algorithmic properties from the optimization literature [80, 81] and for bounding Rademacher complexity from the statistical learning literature [8, 78, 79, 82, 83], but our focus and results are new. More precisely, the 'leave-one-out' stability of optimization algorithms have been used to derive generalization bounds [84, 85, 86, 80, 87, 88]. However, all existing analyses are in the context where the optimization algorithms are used to train and select the model, while our analysis is based on a fundamentally different viewpoint where the algorithm itself is unrolled and integrated as a layer in the deep model. Also, existing works on the generalization of deep learning mainly focus on generic neural architectures such as feed-forward neural networks, RNN, GNN, etc [8, 76, 77]. The complexity of models based on highly structured iterative algorithms and the relation to algorithmic properties have not been investigated. Furthermore, we are not aware of any previous use of local Rademacher complexity analysis for deep learning models.

4.3 Setting: Optimization Algorithm Layers in Deep Learning

In many applications, reasoning can be accomplished by solving an optimization problem defined by a neural perceptual module. For instance, a visual SUDOKU puzzle can be solved using a neural module to perceive the digits followed by a quadratic optimization module to maximize a logic satisfiability objective [14]. The RNA folding problem can be tackled by a neural energy model to capture pairwise relations between RNA bases and a constrained optimization module to minimize the energy, with additional pairing constraints, to obtain a folding [26]. In a broader context, MAML [89, 90] also has a neural module for joint initialization and a reasoning module that performs optimization steps for task-specific adaptation. Other examples include [25, 24, 91, 27, 21, 28, 22, 48, 23, 92, 20, 93, 19, 18, 94]. More specifically, perception and reasoning can be jointly formulated in the form

$$\boldsymbol{y}(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}\in\mathcal{Y}} \; E_\theta(\boldsymbol{x}, \boldsymbol{y}), \tag{4.1}$$

where $\boldsymbol{x}$ is an input, and $E_\theta(\boldsymbol{x}, \boldsymbol{y})$ is a neural energy function with parameters θ, which specifies the type of information needed for performing reasoning, and together with constraints $\mathcal{Y}$ on the output $\boldsymbol{y}$, specifies the style of reasoning. Very often, the optimizer can be approximated by iterative algorithms, so the mapping in Eq. 4.1 can be approximated by the following end-to-end hybrid model

$$f_{\phi,\theta}(\boldsymbol{x}) := \texttt{Alg}_\phi^k \left(E_\theta(\boldsymbol{x}, \cdot)\right) \;\; : \;\; \mathcal{X} \mapsto \mathcal{Y}. \tag{4.2}$$

$\texttt{Alg}_\phi^k$ is the algorithm module with parameters ϕ. Given a neural energy, it performs k-step iterative updates to produce the output (Fig 4.1). When k is finite, $\texttt{Alg}_\phi^k$ corresponds to approximate optimizer. As an initial attempt to analyze deep architectures with algorithm layers, we will restrict our analysis to a simple case where $E_\theta(\boldsymbol{x}, \boldsymbol{y})$ is quadratic in $\boldsymbol{y}$. A

reason is that the analysis of advanced algorithms such as Nesterov accelerated gradients will become very complex for general cases. Similar problems occur in [80] which also restricts the proof to quadratic objectives. Specifically:

Problem setting: Consider a hybrid architecture where the neural module is an energy function of the form $E_\theta((\boldsymbol{x}, \boldsymbol{b}), \boldsymbol{y}) = \frac{1}{2}\boldsymbol{y}^\top Q_\theta(\boldsymbol{x})\boldsymbol{y} + \boldsymbol{b}^\top \boldsymbol{y}$, with Q_θ a neural network that maps $\boldsymbol{x}$ to a matrix. Each energy can be uniquely represented by $(Q_\theta(\boldsymbol{x}), \boldsymbol{b})$, so we can write the overall architecture as

$$f_{\phi,\theta}(\boldsymbol{x}, \boldsymbol{b}) := \texttt{Alg}_\phi^k(Q_\theta(\boldsymbol{x}), \boldsymbol{b}). \tag{4.3}$$

Assume we are given a set of n i.i.d. samples $S_n = \{((\boldsymbol{x}_1, \boldsymbol{b}_1), \boldsymbol{y}_1^*), \cdots, ((\boldsymbol{x}_n, \boldsymbol{b}_n), \boldsymbol{y}_n^*)\}$, where the labels $\boldsymbol{y}^*$ are given by the *exact minimizer* $\texttt{Opt}$ of the corresponding Q^*, i.e.,

$$\boldsymbol{y}^* = \texttt{Opt}(Q^*(\boldsymbol{x}), \boldsymbol{b}). \tag{4.4}$$

Then the learning problem is to find the best model $f_{\phi,\theta}$ from the space $\mathcal{F} := \{f_{\phi,\theta} : (\phi, \theta) \in \Phi \times \Theta\}$ by minimizing the empirical loss function

$$\min_{f_{\phi,\theta} \in \mathcal{F}} \ \frac{1}{n}\sum_{i=1}^{n} \ell_{\phi,\theta}(\boldsymbol{x}_i, \boldsymbol{b}_i), \tag{4.5}$$

where $\ell_{\phi,\theta}(\boldsymbol{x}, \boldsymbol{b}) := \|\texttt{Alg}_\phi^k(Q_\theta(\boldsymbol{x}), \boldsymbol{b}) - \texttt{Opt}(Q^*(\boldsymbol{x}), \boldsymbol{b})\|_2$. Furthermore, we assume:

- We have $\mathcal{Y} = \mathbb{R}^d$, and both Q_θ and Q^* map $\mathcal{X}$ to $\mathcal{S}_{\mu,L}^{d\times d}$, the space of symmetric positive definite (SPD) matrices with $\mu, L > 0$ as its smallest and largest singular values, respectively. Thus the induced energy function E_θ will be μ-strongly convex and L-smooth, and the output of $\texttt{Opt}$ is unique.
- The input $(\boldsymbol{x}, \boldsymbol{b})$ is a pair of random variables where $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^m$ and $\boldsymbol{b} \in \mathcal{B} \subseteq \mathbb{R}^d$. Assume $\boldsymbol{b}$ satisfies $\mathbb{E}[\boldsymbol{b}\boldsymbol{b}^\top] = \sigma_b^2 I$. Assume $\boldsymbol{x}$ and $\boldsymbol{b}$ are independent, and their joint distribution follows a probability measure P. Assume samples in S_n are drawn i.i.d.

from P.

- Assume $\mathcal{B}$ is bounded, and let $M = \sup_{(Q,\boldsymbol{b})\in\mathcal{S}_{\mu,L}^{d\times d}\times\mathcal{B}} \|\texttt{Opt}(Q,\boldsymbol{b})\|_2$.

Though this setting does not encompass the full complexity of hybrid deep architectures, it already reveals interesting connections between algorithmic properties of the algorithm module and the learning behaviors of hybrid architectures.

4.4 Properties of Algorithms

In Section 1.2.2, we have briefly introduced some algorithmic properties and their connections to the approximation efficiency, robustness, and sharpness of neural networks. In this section, we will delve into more details to formally define the algorithmic properties of $\texttt{Alg}_\phi^k$, under the problem setting presented in Sec 4.3. After that, we compare the corresponding properties of gradient descent, $\texttt{GD}_\phi^k$, and Nesterov's accelerated gradients, $\texttt{NAG}_\phi^k$, as concrete examples.

4.4.1 Definitions of Algorithmic Properties

(I) The **convergence rate** of an algorithm expresses how fast the optimization error decreases as k grows. Formally, we say $\texttt{Alg}_\phi^k$ has a convergence rate $\boldsymbol{Cvg}(k,\phi)$ if for any $Q \in \mathcal{S}_{\mu,L}^{d\times d}, \boldsymbol{b}\in\mathcal{B}$,

$$\|\texttt{Alg}_\phi^k(Q,\boldsymbol{b}) - \texttt{Opt}(Q,\boldsymbol{b})\|_2 \le \boldsymbol{Cvg}(k,\phi)\|\texttt{Alg}_\phi^0(Q,\boldsymbol{b}) - \texttt{Opt}(Q,\boldsymbol{b})\|_2. \tag{4.6}$$

(II) Stability of an algorithm characterizes its robustness to small *perturbations in the optimization objective*, which corresponds to the perturbation of Q and $\boldsymbol{b}$ in the quadratic case. For the purpose of this chapter, we say an algorithm $\texttt{Alg}_\phi^k$ is $\boldsymbol{Stab}(k,\phi)$-stable if for any $Q, Q' \in \mathcal{S}_{\mu,L}^{d\times d}$ and $\boldsymbol{b}, \boldsymbol{b}' \in \mathcal{B}$,

$$\|\texttt{Alg}_\phi^k(Q,\boldsymbol{b}) - \texttt{Alg}_\phi^k(Q',\boldsymbol{b}')\|_2 \le \boldsymbol{Stab}(k,\phi)\|Q-Q'\|_2 + \boldsymbol{Stab}(k,\phi)\|\boldsymbol{b}-\boldsymbol{b}'\|_2, \tag{4.7}$$

where $\|Q - Q'\|_2$ is the spectral norm of the matrix $Q - Q'$.

(III) Sensitivity characterizes the robustness to small *perturbations in the algorithm parameters* ϕ. We say the sensitivity of $\texttt{Alg}_\phi^k$ is $\mathit{Sens}(k)$ if it holds for all $Q \in \mathcal{S}_{\mu,L}^{d\times d}, \boldsymbol{b} \in \mathcal{B}$, and $\phi, \phi' \in \Phi$ that

$$\|\texttt{Alg}_\phi^k(Q, \boldsymbol{b}) - \texttt{Alg}_{\phi'}^k(Q, \boldsymbol{b})\|_2 \leq \mathit{Sens}(k)\|\phi - \phi'\|_2. \tag{4.8}$$

This concept is referred in the deep learning community to "parameter perturbation error" or "sharpness" [6, 7, 5]. It has been used for deriving generalization bounds of neural networks, both in the Rademacher complexity framework [8] and PAC-Bayes framework [9].

(IV) The **stable region** is the range Φ of the parameters ϕ where the algorithm output will remain bounded as k grows to infinity, i.e., numerically stable. Only when the algorithms operate in the stable region, the corresponding $\mathit{Cvg}(k, \phi)$, $\mathit{Stab}(k, \phi)$ and $\mathit{Sens}(k)$ will remain finite for all k. It is usually very difficult to identity the exact stable region, but a sufficient range can be provided.

4.4.2 Algorithmic Properties of GD and NAG

Now we will compare the above four algorithmic properties for gradient descent and Nesterov's accelerated gradient method, both of which can be used to solve the quadratic optimization in our problem setting. First, the algorithm update steps are summarized bellow:

$$\texttt{GD}_\phi : \boldsymbol{y}_{k+1} \leftarrow \boldsymbol{y}_k - \phi(Q\boldsymbol{y}_k + \boldsymbol{b}) \quad \texttt{NAG}_\phi : \begin{cases} \boldsymbol{y}_{k+1} \leftarrow \boldsymbol{z}_k - \phi(Q\boldsymbol{z}_k + \boldsymbol{b}) \\ \boldsymbol{z}_{k+1} \leftarrow \boldsymbol{y}_{k+1} + \frac{1-\sqrt{\mu\phi}}{1+\sqrt{\mu\phi}}(\boldsymbol{y}_{k+1} - \boldsymbol{y}_k) \end{cases} \tag{4.9}$$

where the hyperparameter ϕ corresponds to the step size. The initializations $\boldsymbol{y}_0, \boldsymbol{z}_0$ are set to zero vectors throughout this chapter. Denote the results of k-step update, $\boldsymbol{y}_k$, of GD and NAG by $\texttt{GD}_\phi^k(Q, \boldsymbol{b})$ and $\texttt{NAG}_\phi^k(Q, \boldsymbol{b})$, respectively. Then their algorithmic properties are summarized in Table 4.1.

Table 4.1: Comparison of algorithmic properties between GD and NAG. For simplicity, only the order in k is presented.

Alg	*Cvg*(k,ϕ)	*Stab*(k,ϕ)	*Sens*(k)	Stable region Φ
$\texttt{GD}_\phi^k$	$\mathcal{O}\left((1-\phi\mu)^k\right)$	$\mathcal{O}\left(1-(1-\phi\mu)^k\right)$	$\mathcal{O}\left(k(1-c_0\mu)^{k-1}\right)$	$[c_0, \frac{2}{\mu+L}]$
$\texttt{NAG}_\phi^k$	$\mathcal{O}\left(k(1-\sqrt{\phi\mu})^k\right)$	$\mathcal{O}\left(1-(1-\sqrt{\phi\mu})^k\right)$	$\mathcal{O}\left(k^3(1-\sqrt{c_0\mu})^k\right)$	$[c_0, \frac{4}{\mu+3L}]$

Table 4.1 shows: *(i) Convergence*: NAG converges faster than GD, especially when μ is very small, which is a well-known result. *(ii) Stability*: However, as k grows, NAG is less stable than GD for a fixed k, in contrast to their convergence behaviors. This is pointed out in [80], which proves that a faster converging algorithm has to be less stable. *(iii) Sensitivity*: The sensitivity behaves similar to the convergence, where NAG is less sensitive to step-size perturbation than GD. Also, the sensitivity of both algorithms gets smaller as k grows larger. *(iv): Stable region*: Since $\mu < L$, the stable region of GD is larger than that of NAG. It means a larger step size is allowable for GD that will not lead to exploding outputs even if k is large. Note that all the other algorithmic properties are based on the assumption that ϕ is in the stable region Φ. Furthermore, as k goes to infinity, the space $\{\texttt{Alg}_\phi^k : \phi \in \Phi\}$ will finally shrink to a single function, which is the exact minimizer.

Our purpose of comparing the algorithmic properties of GD and NAG is to show in a later section their difference when used as a algorithm module in deep architectures.

4.5 Approximation Ability

How will the algorithmic properties affect the approximation ability of deep architecture with algorithm layers? Given a model space $\mathcal{F} := \{\texttt{Alg}_\phi^k\left(Q_\theta(\cdot),\cdot\right) : \phi \in \Phi, \theta \in \Theta\}$, we are interested in its approximation ability to functions of the form $\texttt{Opt}\left(Q^*(\boldsymbol{x}), \boldsymbol{b}\right)$. More specifically, we define the loss

$$\ell_{\phi,\theta}(\boldsymbol{x},\boldsymbol{b}) := \|\texttt{Alg}_\phi^k\left(Q_\theta(\boldsymbol{x}),\boldsymbol{b}\right) - \texttt{Opt}(Q^*(\boldsymbol{x}),\boldsymbol{b})\|_2, \tag{4.10}$$

and measure the approximation ability by $\inf_{\phi\in\Phi,\theta\in\Theta}\sup_{Q^*\in\mathcal{Q}^*} P\ell_{\phi,\theta}$, where $\mathcal{Q}^* := \{\mathcal{X} \mapsto \mathcal{S}^{d\times d}_{\mu,L}\}$ and $P\ell_{\phi,\theta} = \mathbb{E}_{\boldsymbol{x},\boldsymbol{b}}[\ell_{\phi,\theta}(\boldsymbol{x},\boldsymbol{b})]$. Intuitively, using a faster converging algorithm, the model $\texttt{Alg}^k_\phi$ could represent the reasoning-task structure, $\texttt{Opt}$, better and improve the overall approximation ability. Indeed we can prove the following lemma confirming this intuition.

Lemma 4.5.1. (Faster Convergence $\Rightarrow$ Better Approximation Ability). *Assume the problem setting in Sec 4.3. The approximation ability can be bounded by two terms:*

$$\inf_{\phi\in\Phi,\theta\in\Theta}\sup_{Q^*\in\mathcal{Q}^*} P\ell_{\phi,\theta} \le \sigma_b\mu^{-2} \underbrace{\inf_{\theta\in\Theta}\sup_{Q^*\in\mathcal{Q}^*} P\|Q_\theta - Q^*\|_F}_{\textit{approximation ability of the neural module}} + M \underbrace{\inf_{\phi\in\Phi} \mathit{Cvg}(k,\phi)}_{\textit{best convergence}}. \tag{4.11}$$

With Lemma 4.5.1, we conclude that: A faster converging algorithm can define a model with better approximation ability. For example, for a fixed k and Q_θ, NAG converges faster than GD, so $\texttt{NAG}^k_\phi$ can approximate $\texttt{Opt}$ more accurately than $\texttt{GD}^k_\phi$, which is experimentally validated in Sec 4.7.

Similarly, we can also reverse the reasoning, and ask the question that, given two hydrid architectures with the same approximation error, which architecture has a smaller error in representing the energy function Q^*? We show that this error is also intimately related to the convergence of the algorithm.

Lemma 4.5.2. (Faster Convergence $\Rightarrow$ Better Representation of Q^*). *Assume the problem setting in Sec 4.3. Then $\forall\phi\in\Phi,\theta\in\Theta,Q^*\in\mathcal{Q}^*$ it holds true that*

$$P\|Q_\theta - Q^*\|_F^2 \le \sigma_b^{-2}L^4(\sqrt{P\ell^2_{\phi,\theta}} + M\cdot \mathit{Cvg}(k,\phi))^2. \tag{4.12}$$

Lemma 4.5.2 highlights the benefit of using an algorithmic layer that aligns with the reasoning-task structure. Here the task structure is represented by $\texttt{Opt}$, the minimizer, and convergence measures how well $\texttt{Alg}^k_\phi$ is aligned with $\texttt{Opt}$. Lemma 4.5.2 essentially indicates that *if the structure of a algorithm module can better align with the task structure,*

then it can better constrain the search space of the underlying neural module Q_θ, making it easier to learn, and further lead to better sample complexity, which we will explain more in the next section.

As a concrete example for Lemma 4.5.2, if $\texttt{GD}_\phi^k\left(Q_\theta, \cdot\right)$ and $\texttt{NAG}_\phi^k\left(Q_\theta, \cdot\right)$ achieve the **same** accuracy for approximating $\texttt{Opt}\left(Q^*, \cdot\right)$, then the neural module Q_θ in $\texttt{NAG}_\phi^k\left(Q_\theta, \cdot\right)$ will have a **better** accuracy for approximating Q^* than Q_θ in $\texttt{GD}_\phi^k\left(Q_\theta, \cdot\right)$. In other words, a faster converging algorithm imposes more constraints on the energy function Q_θ, making it approach Q^* faster.

4.6 Generalization Ability

How will algorithmic properties affect the generalization ability of deep architectures with algorithm layers? We theoretically showed that the generalization bound is determined by both the algorithmic properties and the complexity of the neural module. Moreover, it induces interesting implications - when the neural module is over- or under- parameterized, the generalization bound is dominated by algorithm stability; but when the neural module has an about-right parameterization, the bound is dominated by the product of algorithm stability and convergence.

More specifically, we will analyze *generalization gap* between the expected loss and empirical loss,

$$P\ell_{\phi,\theta} = \mathbb{E}_{\boldsymbol{x},\boldsymbol{b}}\ell_{\phi,\theta}(\boldsymbol{x},\boldsymbol{b}) \text{ and } P_n\ell_{\phi,\theta} = \tfrac{1}{n}\textstyle\sum_{i=1}^n \ell_{\phi,\theta}(\boldsymbol{x}_i,\boldsymbol{b}_i), \text{ respectively,} \tag{4.13}$$

where P_n is the empirical probability measure induced by the samples S_n. Let $\ell_{\mathcal{F}} := \{\ell_{\phi,\theta} : \phi \in \Phi, \theta \in \Theta\}$ be the function space of losses of the models. The generalization gap, $P\ell_{\phi,\theta} - P_n\ell_{\phi,\theta}$, can be bounded by the Rademacher complexity, $\mathbb{E}R_n\ell_{\mathcal{F}}$, which is defined as the expectation of the empirical Rademacher complexity, $R_n\ell_{\mathcal{F}} := \mathbb{E}_{\boldsymbol{\sigma}} \sup_{\phi\in\Phi,\theta\in\Theta} \frac{1}{n}$ $\sum_{i=1}^n \sigma_i\ell_{\phi,\theta}(\boldsymbol{x}_i,\boldsymbol{b}_i)$, where $\{\sigma_i\}_{i=1}^n$ are n independent Rademacher random variables uni-

formly distributed over $\{\pm 1\}$. Generalization bounds derived from Rademacher complexity have been studied in many works [95, 96, 97, 98].

However, deriving the Rademacher complexity of $\ell_{\mathcal{F}}$ is highly nontrivial in our case, and we are not aware of prior bounds for deep learning models with algorithm layers. Aiming at bridging the relation between algorithmic properties and generalization ability that can explain experimental observations, we find that standard Rademacher complexity analysis is insufficient. The shortcoming of the standard Rademacher complexity is that it provides *global* estimates of the complexity of the model space, which ignores the fact that the training process will likely pick models with small errors. Taking this factor into account, we resort to more refined analysis using *local* Rademacher complexity [8, 78, 79].

4.6.1 Main Result

The local Rademacher complexity of $\ell_{\mathcal{F}}$ at level r is defined as

$$\mathbb{E}R_n\ell_{\mathcal{F}}^{loc}(r) \text{ where } \ell_{\mathcal{F}}^{loc}(r) := \{\ell_{\phi,\theta} : \phi \in \Phi, \theta \in \Theta, P\ell_{\phi,\theta}^2 \leq r\}. \tag{4.14}$$

This notion is less general than the one defined in [78, 79] but is sufficient for our purpose. Here we also define a loss function space $\ell_{\mathcal{Q}} := \{\|Q_\theta - Q^*\|_F : \theta \in \Theta\}$ for the neural module Q_θ, and introduce its local Rademacher complexity $\mathbb{E}R_n\ell_{\mathcal{Q}}^{loc}(r_q)$, where $\ell_{\mathcal{Q}}^{loc}(r_q) = \left\{\|Q_\theta - Q^*\|_F \in \ell_{\mathcal{Q}} : P\|Q_\theta - Q^*\|_F^2 \leq r_q\right\}$. With these definitions, we can show that the local Rademacher complexity of the hybrid architecture is explicitly related to all considered algorithmic properties, namely convergence, stability and sensitivity, and there is an intricate trade-off.

Theorem 4.6.1. *Assume the problem setting in Sec 4.3. Then we have for any $t > 0$ that*

$$\mathbb{E}R_n\ell_{\mathcal{F}}^{loc}(r) \leq \sqrt{2}dn^{-\frac{1}{2}}\textit{Stab}(k)\left(\sqrt{(\textit{Cvg}(k)M+\sqrt{r})^2C_1(n)+C_2(n,t)}+C_3(n,t)+4\right) \tag{4.15}$$

$$+\textit{Sens}(k)B_\Phi, \tag{4.16}$$

where $\textit{Stab}(k) = \sup_\phi \textit{Stab}(k,\phi)$ *and* $\textit{Cvg}(k) = \sup_\phi \textit{Cvg}(k,\phi)$ *are worst-case stability and convergence,* $B_\Phi = \frac{1}{2}\sup_{\phi,\phi'\in\Phi}\|\phi-\phi'\|_2$, $C_1(n) = \mathcal{O}(\log N(n))$, $C_3(n,t) = \mathcal{O}(\frac{\log N(n)}{\sqrt{n}} + \frac{\sqrt{\log N(n)}}{e^t})$, $C_2(n,t) = \mathcal{O}(\frac{t\log N(n)}{n} + (C_3(n,t)+1)\frac{\log N(n)}{\sqrt{n}})$, *and* $N(n) = \mathcal{N}(\frac{1}{\sqrt{n}}, \ell_\mathcal{Q}, L_\infty)$ *is the covering number of* $\ell_\mathcal{Q}$ *with radius* $\frac{1}{\sqrt{n}}$ *and* L_∞ *norm.*

4.6.2 Implications

Trade-offs between convergence, stability and sensitivity. Generally speaking, the convergence rate $\textit{Cvg}(k)$ and sensitivity $\textit{Sens}(k)$ have similar behavior, but $\textit{Stab}(k)$ behaves opposite to them; see illustrations in Fig 4.2. Therefore, the way these three quantities interact in Theorem 4.6.1 suggests that in different regimes one may see different generalization behavior. More specially, depending on the parameterization of Q_θ, the coefficients C_1, C_2, and C_3 in Eq. 4.15 may have different scale, making the local Rademacher complexity bound dominated by different algorithmic properties. Since the coefficients C_i are monotonely increasing in the covering number of $\ell_\mathcal{Q}$, we expect that:

(i) When Q_θ is **over-parameterized**, the covering number of $\ell_\mathcal{Q}$ becomes large, as do the three coefficients. Large C_i will reduce the effect of $\textit{Cvg}(k)$ and make Eq. 4.15 dominated by $\textit{Stab}(k)$;

(ii) When Q_θ is **under-parameterized**, the three coefficients get small, but they still reduce the effect of $\textit{Cvg}(k)$ given the constant 4 in Eq. 4.15, again making it dominated by $\textit{Stab}(k)$;

(iii) When the parametrization of Q_θ is **about-right**, we can expect $\textit{Cvg}(k)$ to play

a critical role in Eq. 4.15, which will then behave similar to the product $Stab(k)Cvg(k)$, as illustrated schematically in Fig 4.2. We experimentally validate these implications in Sec 4.7.

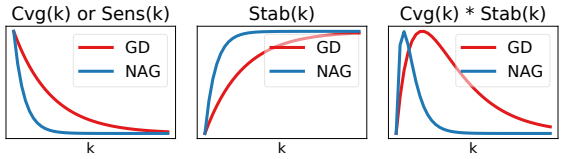


Figure 4.2: Overall trend of algorithmic properties.

Trade-off of the depth. Combining the above implications with the approximation ability analysis in Sec 4.5, we can see that in the above-mentioned cases **(i)** and **(ii)**, deeper algorithm layers will lead to better approximation accuracy but worse generalization. Only in the ideal case **(iii)**, a deeper algorithm module can induce both better representation and generalization abilities. This result provides practical guidelines for some recently proposed infinite-depth models [99, 100].

4.6.3 Comparison to Standard Rademacher complexity Analysis

If we consider the standard Rademacher complexity and directly bound it by the covering number of $\ell_{\mathcal{F}}$ via Dudley's entropy integral in the way some existing generalization bounds of deep learning are derived [8, 76, 77], we will get the following upper bound for the covering number, where $Cvg(k)$ does not play a role:

$$\mathcal{N}(\epsilon, \ell_{\mathcal{F}}, L_2(P_n)) \leq \mathcal{N}(\epsilon/\left(2Stab(k)\right), \mathcal{Q}, L_2(P_n)) \cdot \mathcal{N}(\epsilon/\left(2Sens(k)\right), \Phi, \|\cdot\|_2). \quad (4.17)$$

Since Φ only contains the hyperparameters in the algorithm and $\mathcal{Q} := \{Q_\theta, \theta \in \Theta\}$ is often highly expressive, typically stability will dominate this bound. Or, consider the case when algorithm layers are fixed so Φ only contains one element. Then this covering number is determined by stability, which infers that $\texttt{NAG}_1^k(Q_\theta, \cdot)$ has a **larger** Rademacher complexity than $\texttt{GD}_1^k(Q_\theta, \cdot)$ since it is less stable. However, in the local Rademacher complexity bound

in Theorem 4.6.1, even if $\mathit{Sens}(k)$ in Eq. 4.16 is ignored, there is still a trade-off between convergence and stability which implies $\texttt{NAG}_1^k(Q_\theta, \cdot)$ can have a **smaller** local Rademacher complexity than $\texttt{GD}_1^k(Q_\theta, \cdot)$, leading to a different conclusion. Our experiments show the local Rademacher complexity bound is better for explaining the actual observations.

4.6.4 Pros and Cons of RNN as a Reasoning Layer

It has been shown that RNN (or GNN) can represent reasoning and iterative algorithms over structures [38, 77]. Can our analysis framework also be used to understand RNN (or GNN)? How will its behavior compare with more interpretable algorithm layers such as $\texttt{GD}_\phi^k$ and $\texttt{NAG}_\phi^k$? In the case of RNN, the algorithm update steps in each iteration are given by an RNN cell

$$\boldsymbol{y}_{k+1} \leftarrow \texttt{RNNcell}\left(Q, \boldsymbol{b}, \boldsymbol{y}_k\right) := V\sigma\left(W^L\sigma\left(W^{L-1}\cdots W^2\sigma\left(W_1^1\boldsymbol{y}_t + W_2^1\boldsymbol{g}_t\right)\right)\right). \quad (4.18)$$

where the activation function $\sigma = \text{ReLU}$ takes $\boldsymbol{y}_k$ and the gradient $\boldsymbol{g}_t = Q\boldsymbol{y}_t + \boldsymbol{b}$ as inputs. Then a recurrent neural network $\texttt{RNN}_\phi^k$ having k unrolled RNN cells can be viewed as a neural algorithm.

Table 4.2: Properties of $\texttt{RNN}_\phi^k$.

Stable region Φ	$c_\phi < 1$
$\mathit{Stab}(k, \phi)$	$\mathcal{O}(1 - c_\phi^k)$
$\mathit{Sens}(k)$	$\mathcal{O}(1 - (\inf_\phi c_\phi)^k)$
$\min_\phi \mathit{Cvg}(k, \phi)$	$\mathcal{O}(\rho^k)$ with $\rho < 1$

The algorithmic properties of $\texttt{RNN}_\phi^k$ are summarized in Table 4.2. Assume $\phi = \{V, W_1^1, W_2^1, W^{2:L}\}$ is in a stable region with $c_\phi := \sup_Q \|V\|_2\|W_1^1 + W_2^1 Q\|_2 \prod_{l=2}^L \|W^l\|_2 < 1$, so that the operations in $\texttt{RNNcell}$ are strictly contractive, i.e., $\|\boldsymbol{y}_{k+1} - \boldsymbol{y}_k\|_2 < \|\boldsymbol{y}_k - \boldsymbol{y}_{k-1}\|_2$. In this case, the stability and sensitivity of $\texttt{RNN}_\phi^k$ are guaranteed to be bounded.

However, the fundamental disadvantage of RNN is its lack of worst-case guarantee for convergence. In general the outputs of $\texttt{RNN}_\phi^k$ may not converge to the minimizer $\texttt{Opt}$,

meaning that its worst-case convergence rate can be much larger than 1. This will lead to worse generalization bound according to our theory compared to $\texttt{GD}_\phi^k$ and $\texttt{NAG}_\phi^k$.

The advantage of RNN is its expressiveness, especially given the universal approximation ability of MLP in the RNNcell. One can show that $\texttt{RNN}_\phi^k$ can express $\texttt{GD}_\phi^k$ or $\texttt{NAG}_\phi^k$ with suitable choices of ϕ. Therefore, its best-case convergence can be as small as $\mathcal{O}(\rho^k)$ for some $\rho < 1$. When the needed types of reasoning is unknown or beyond what existing algorithms are capable of, RNN has the potential to learn new reasoning types given sufficient data.

4.7 Experimental Validation

Our experiments aim to validate our theoretical prediction with computational simulations, rather than obtaining state-of-the-art results. Implementations in Python are released[1].

The experiments follow the problem setting in Sec 4.3. We sample 10000 pairs of $(\boldsymbol{x}, \boldsymbol{b})$ uniformly as overall dataset. During training, n samples are randomly drawn from these 10000 data points as the training set. Each $Q^*(\boldsymbol{x})$ is produced by a rotation matrix and a vector of eigenvalues parameterized by a randomly fixed 2-layer dense neural network with hidden dimension 3. Then the labels are generated according to $\boldsymbol{y} = \texttt{Opt}(Q^*(\boldsymbol{x}), \boldsymbol{b})$. We train the model $\texttt{Alg}_\phi^k(Q_\theta, \cdot)$ on S_n using the loss in Eq. 4.10. Here, Q_θ has the same overall architecture as Q^* but the hidden dimension could vary. Note that in all figures, each k corresponds to an **independently trained model** with k iterations in the algorithm layer, instead of the sequential outputs of a single model. Each model is trained by ADAM and SGD with learning rate grid-searched from [1e-2,5e-3,1e-3,5e-4,1e-4], and only the best result is reported. Furthermore, error bars are produced by 20 independent instantiations of the experiments.

Approximation ability. To validate Lemma 4.5.1, we compare $\texttt{GD}_\phi^k(Q_\theta, \cdot)$ and $\texttt{NAG}_\phi^k(Q_\theta, \cdot)$ in terms of approximation accuracy. For various hidden sizes of Q_θ, the results are similar,

[1] https://github.com/xinshi-chen/Deep-Architecture-With-Reasoning-Layer

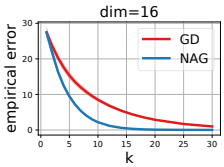


Figure 4.3: Training error.

so we report one representative case in Fig 4.3. The approximation accuracy aligns with the convergence of the algorithms, showing that faster converging algorithm can induce better approximation ability.

Faster convergence⇒better Q_θ. We report the error of the neural module Q_θ in Fig 4.4. Note that $\texttt{Alg}_\phi^k(Q_\theta, \cdot)$ is trained end-to-end, without supervision on Q_θ. In Fig 4.4, the error of Q_θ decreases as k grows, in a rate similar to algorithm convergence. This validates the implication of Lemma 4.5.2 that, when $\texttt{Alg}_\phi^k$ is closer to $\texttt{Opt}$, it can help the underlying neural module Q_θ to get closer to Q^*.

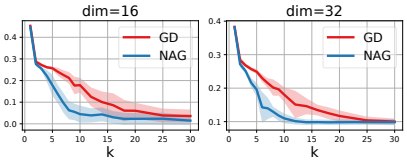


Figure 4.4: $P\|Q_\theta - Q^*\|_F^2$

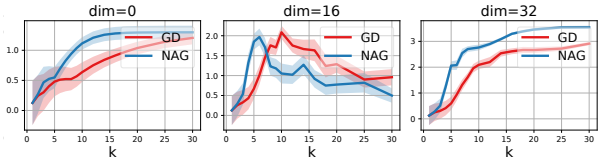


Figure 4.5: Generalization gap

Generalization gap. In Fig 4.5, we report the generalization gaps, with hidden sizes of Q_θ being 0, 16, and 32, which corresponds to the three cases **(ii)**, **(iii)**, and **(i)** discussed under Theorem 4.6.1, respectively. Comparing Fig 4.5 to Fig 4.2, we can see that the experimental results match very well with the theoretical implications.

CHAPTER 5

DYNAMIC DEEP LEARNING MODEL INSPIRED BY ALGORITHMS: LEARNING TO STOP WHILE LEARNING TO PREDICT

There is a recent surge of interest in designing deep architectures based on the update steps in traditional algorithms, or learning neural networks to improve and replace traditional algorithms. While traditional algorithms have certain stopping criteria for outputting results at different iterations, many algorithm-inspired deep models are restricted to a "fixed-depth" for all inputs. Similar to algorithms, the optimal depth of a deep architecture may be different for different input instances, either to avoid "over-thinking", or because we want to compute less for operations converged already. In this chapter, we tackle this varying depth problem using a steerable architecture, where a feed-forward deep model and a variational stopping policy are learned together to sequentially determine the optimal number of layers for each input instance. Training such architecture is very challenging. We provide a variational Bayes perspective and design a novel and effective training procedure which decomposes the task into an oracle model learning stage and an imitation stage. Experimentally, we show that the learned deep model along with the stopping policy improves the performances on a diverse set of tasks, including learning sparse recovery, few-shot meta learning, and computer vision tasks.

The research in this chapter was previously presented at the ICML 2020 conference in [101].

5.1 Introduction

Recently, researchers are increasingly interested in the connections between deep learning models and traditional algorithms: deep learning models are viewed as parameterized algorithms that operate on each input instance iteratively, and traditional algorithms are used

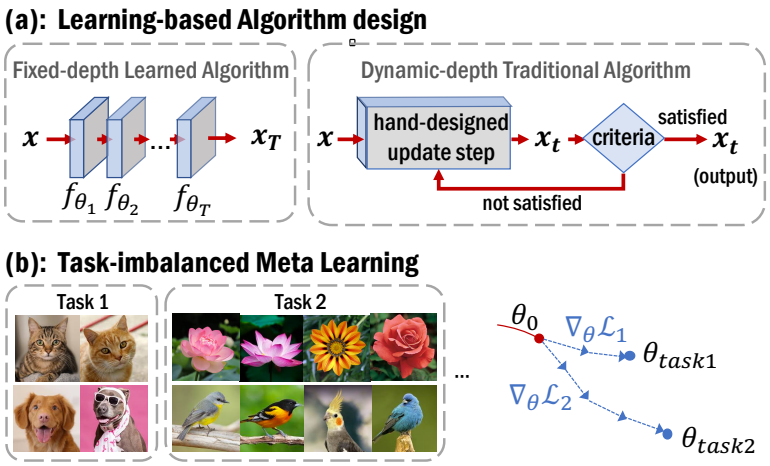


Figure 5.1: Motivation for learning to stop.

as templates for designing deep learning architectures. While an important concept in traditional algorithms is the stopping criteria for outputting the result, which can be either a *convergence* condition or an *early stopping* rule, such stopping criteria has been more or less ignored in algorithm-inspired deep learning models. A "fixed-depth" deep model is used to operate on all problem instances (Fig. 5.1 (a)). Intuitively, for deep learning models, the optimal depth (or the optimal number of steps to operate on an input) can also be different for different input instances, either because we want to compute less for operations converged already, or we want to generalize better by avoiding "over-thinking". Such motivation aligns well with both the cognitive science literature [102] and many examples below:

- In learning to optimize [38, 103], neural networks are used as the optimizer to minimize some loss function. Depending on the initialization and the objective function, an optimizer should converge in different number of steps;
- In learning to solve statistical inverse problems such as compressed sensing [48, 49], inverse covariance estimation [43], and image denoising [55], deep models are learned to directly predict the recovery results. In traditional algorithms, problem-dependent early stopping rules are widely used to achieve regularization for a variance-bias trade-off. Deep learning models for solving such problems maybe also achieve a better recovery

accuracy by allowing instance-specific computation steps;

- In meta learning, MAML [89] used an unrolled and parametrized algorithm to adapt a common parameter to a new task. However, depending on the *similarity* of the new task to the old tasks, or, in a more realistic *task-imbalanced* setting where different tasks have different numbers of data points (Fig. 5.1 (b)), a task-specific number of adaptation steps is more favorable to avoid under or over adaption.

To address the varying depth problem, we propose to learn a steerable architecture, where a shared feed-forward model for normal prediction and an additional stopping policy are learned together to sequentially determine the optimal number of layers for each input instance. In our framework, the model consists of (see Fig. 5.2)

- A **feed-forward or recurrent mapping** $\mathcal{F}_\theta$, which transforms the input $\boldsymbol{x}$ to generate a path of features (or states) $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T$; and
- A **stopping policy** $\pi_\phi : (\boldsymbol{x}, \boldsymbol{x}_t) \mapsto \pi_t \in [0, 1]$, which sequentially observes the states and then determines the probability of stopping the computation of $\mathcal{F}_\theta$ at layer t.

These two components allow us to sequentially **predict** the next targeted state while at the same time determining when to **stop**. In this chapter, we propose a single objective function for learning both θ and ϕ, and we interpret it from the perspective of variational Bayes, where the stopping time t is viewed as a latent variable conditioned on the input $\boldsymbol{x}$. With this interpretation, learning θ corresponds to maximizing the marginal likelihood, and learning ϕ corresponds to the inference step for the latent variable, where a variational distribution $q_\phi(t)$ is optimized to approximate the posterior. A natural algorithm for solving this problem could be the Expectation-Maximization (EM) algorithm, which can be very hard to train and inefficient.

How to learn θ and ϕ effectively and efficiently? We propose a principled and effective training procedure, where we decompose the task into an oracle model learning stage and an imitation learning stage (Fig. 5.3). More specifically,

- During the oracle model learning stage, we utilize a closed-form oracle stopping distri-

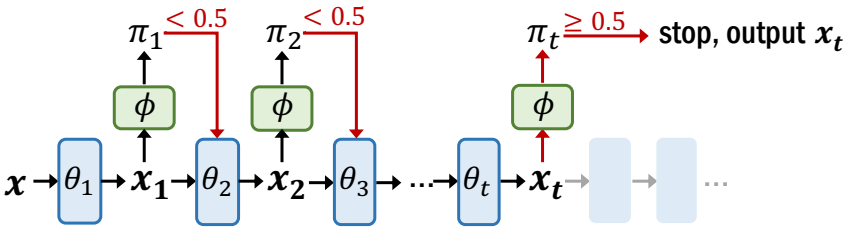


Figure 5.2: Two-component model: learning to predict (*blue*) while learning to stopping (*green*).

bution $q^*|\theta$ which can leverage label information not available at testing time.

- In the imitation learning stage, we use a sequential policy π_ϕ to mimic the behavior of the oracle policy obtained in the first stage. The sequential policy does not have access to the label so that it can be used during testing phase.

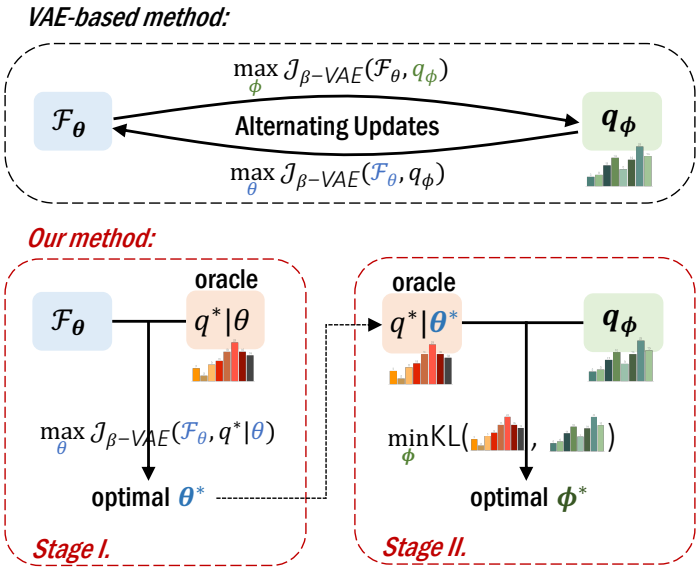


Figure 5.3: Two-stage training framework.

This procedure provides us a very good initial predictive model and a stopping policy. We can either directly use these learned models, or plug them back to the variational EM framework and reiterate to further optimize both together.

Our proposed learning to stop method is a generic framework that can be applied to a diverse range of applications. To summarize, our contribution in this chapter includes:

1. a variational Bayes perspective to understand the proposed model for learning both the

predictive model and the stopping policy together;

2. a principled and efficient algorithm for jointly learning the predictive model and the stopping policy; and the relation of this algorithm to reinforcement learning;
3. promising experiments on various tasks including learning to solve sparse recovery problems, task-imbalanced few-shot meta learning, and computer vision tasks, where we demonstrate the effectiveness of our method in terms of both the prediction accuracy and inference efficiency.

5.2 Related Works

Unrolled algorithm. A line of recent works unfold and truncate iterative algorithms to design neural architectures. These algorithm-based deep models can be used to automatically learn a better algorithm from data. This idea has been demonstrated in different problems including sparse signal recovery [45, 104, 40, 105, 59, 48, 49], sparse inverse covariance estimation [43], sequential Bayesian inference [56], parameter learning in graphical models [106], non-negative matrix factorization [107], etc. Unrolled algorithm based deep module has also be used for structured prediction [24, 25, 26]. Before the training phase, all these works need to assign a fixed number of iterations that is used for every input instance regardless of their varying difficulty level. Our proposed method is orthogonal and complementary to all these works, by taking the variety of the input instances into account via adaptive stopping time.

Meta learning. Optimization-based meta learning techniques are widely applied for solving challenging few-shot learning problems [108, 89, 109]. Several recent advances proposed task-adaptive meta-learning models which incorporate task-specific parameters [110, 111, 112] or task-dependent metric scaling [113]. In parallel with these task-adaptive methods, we propose a task-specific number of adaptation steps and demonstrate the effectiveness of this simple modification under the task-imbalanced scenarios.

Other adaptive-depth deep models. In image recognition, 'early exits' is proposed

mainly aimed at improving the computation efficiency during the inference phase [114, 115, 116], but these methods are based on specific architectures. [117] proposed to avoiding "over-thinking" by early stopping. However, the same as all the other 'early exits' models, some heuristic policies are adopted to choose the output layer by confidence scores of internal classifiers. Also, their algorithms for training the feed-forward model $\mathcal{F}_\theta$ do not take into account the effect of the stopping policy.

Optimal stopping. In optimal control literature, optimal stopping is a problem of choosing a time to take a given action based on sequentially observed random variables in order to maximize an expected payoff [118]. When a policy for controlling the evolution of random variables (corresponds to the output of $\mathcal{F}_\theta$) is also involved, it is called a "mixed control" problem, which is highly related to our work. Existing works in this area find the optimal controls by solving the Hamilton-Jacobi-Bellman (HJB) equation, which is theoretically grounded [119, 120, 121]. However, they focus on stochastic differential equation based model and the proposed algorithms suffer from the curse of dimensionality problem. [122] use DL to learn the optimal stopping policy, but the learning of θ is not considered. Besides, [122] use reinforcement learning (RL) to solve the problem. In Section 5.4, we will discuss how our variational inference formulation is related to RL.

5.3 Problem Formulation

In this section, we will introduce how we model the stopping policy together with the predictive deep model, define the joint optimization objective, and interpret this framework from a variational Bayes perspective.

5.3.1 Dynamic model

The predictive model, $\mathcal{F}_\theta$, is a typical T-layer deep model that generates a path of embeddings $(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T)$ through:

$$\textbf{Predictive model:} \quad \boldsymbol{x}_t = f_{\theta_t}(\boldsymbol{x}_{t-1}), \text{ for } t = 1, \cdots, T \tag{5.1}$$

where the initial $\boldsymbol{x}_0$ is determined by the input $\boldsymbol{x}$. We denote it by $\mathcal{F}_\theta = \{f_{\theta_1}, \cdots, f_{\theta_T}\}$ where $\theta \in \Theta$ are the parameters. Standard supervised learning methods learn θ by optimizing an objective estimated on the final state $\boldsymbol{x}_T$. In our model, the operations in Eq. 5.1 can be stopped earlier, and for different input instance $\boldsymbol{x}$, the stopping time t can be different.

Our stopping policy, π_ϕ, determines whether to stop at t-th step after observing the input $\boldsymbol{x}$ and its first t states $\boldsymbol{x}_{1:t}$ transformed by $\mathcal{F}_\theta$. If we assume the Markov property, then π_ϕ only needs to observe the most recent state $\boldsymbol{x}_t$. In this chapter, we only input $\boldsymbol{x}$ and $\boldsymbol{x}_t$ to π_ϕ at each step t, but it is trivial to generalize it to $\pi_\phi(\boldsymbol{x}, \boldsymbol{x}_{1:t})$. More precisely, π_ϕ is defined as a randomized policy as follows:

$$\textbf{Stopping policy:} \quad \pi_t = \pi_\phi(\boldsymbol{x}, \boldsymbol{x}_t), \text{ for } t = 1, \cdots, T-1 \tag{5.2}$$

where $\pi_t \in [0, 1]$ is the probability of stopping. We abuse the notation π to both represent the parametrized policy and also the probability mass.

This stopping policy sequentially makes a decision whenever a new state $\boldsymbol{x}_t$ is observed. Conditioned on the states observed until step t, whether to stop before t is independent on states after t. Therefore, once it decides to stop at t, the remaining computations can be saved, which is a favorable property when the inference time is a concern, or for some optimal stopping problems such as option trading where getting back to earlier states is not allowed.

5.3.2 From sequential policy to stop time distribution

The stopping policy π_ϕ makes sequential actions based on the observations, where $\pi_t := \pi_\phi(\boldsymbol{x}, \boldsymbol{x}_t)$ is the probability of stopping when $\boldsymbol{x}_t$ is observed. These sequential actions $\pi_1, \cdots, \pi_{T-1}$ jointly determines the random time t at which the stop occurs. Induced by π_ϕ, the probability mass function of the stop time t, denoted as q_ϕ, can be computed by

$$\textbf{Variational stop time distribution:} \begin{cases} q_\phi(t) = \pi_t \prod_{\tau=1}^{t-1}(1-\pi_\tau) & \text{if } t < T, \\ q_\phi(T) = \prod_{\tau=1}^{T-1}(1-\pi_\tau) & \text{else.} \end{cases} \tag{5.3}$$

In this equation, the product $\prod_{\tau=1}^{t-1}(1-\pi_\tau)$ indicates the probability of 'not stopped before t', which is the survival probability. Multiply this survival probability with π_t, we have the stop time distribution $q_\phi(t)$. For the last time step T, the stop probability $q_\phi(T)$ simply equals to the survival probability at T, which means if the process is 'not stopped before T', then it must stop at T.

Note that we only use π_ϕ in our model to sequentially determine whether to stop. However, we use the induced probability mass q_ϕ to help design the training objective and also the algorithm.

5.3.3 Optimization objective

Note that the stop time t is a discrete random variable with distribution determined by $q_\phi(t)$. Given the observed label $\boldsymbol{y}$ of an input $\boldsymbol{x}$, the loss of the predictive model stopped at position t can computed as $\ell(\boldsymbol{y}, \boldsymbol{x}_t; \theta)$ where $\ell(\cdot)$ is a loss function. Taking into account all possible stopping positions, we will be interested in the loss in expectation over t,

$$\mathcal{L}(\theta, q_\phi; \boldsymbol{x}, \boldsymbol{y}) := \mathbb{E}_{t\sim q_\phi} \ell(\boldsymbol{y}, \boldsymbol{x}_t; \theta) - \beta H(q_\phi), \tag{5.4}$$

where $H(q_\phi) := -\sum_t q_\phi(t) \log q_\phi(t)$ is an entropy regularization and β is the regularization coefficient. Given a data set $\mathcal{D} = \{(\boldsymbol{x}, \boldsymbol{y})\}$, the parameters of the predictive model and the stopping policy can be estimated by

$$\min_{\theta,\phi} \quad \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \mathcal{L}(\theta, q_\phi; \boldsymbol{x}, \boldsymbol{y}). \tag{5.5}$$

To better interpret the model and objective, in the following, we will make a connection from the perspective of variational Bayes, and how the objective function defined in Eq. 5.4 is equivalent to the β-VAE objective.

5.3.4 Variational Bayes perspective

In the Bayes' framework, a probabilistic model typically consists of prior, likelihood function and posterior of the latent variable. We find the correspondence between our model and a probabilistic model as follows (also see Table 5.1)

- we view the adaptive stopping time t as a *latent variable* which is unobserved;
- The conditional *prior* $p(t|\boldsymbol{x})$ of t is a uniform distribution over all the layers in this chapter. However, if one wants to reduce the computation cost and penalize the stopping decisions at deeper layers, a prior with smaller probability on deeper layers can be defined to regularize the results;
- The *likelihood* function $p_\theta(\boldsymbol{y}|t, \boldsymbol{x})$ of the observed label $\boldsymbol{y}$ is controlled by θ, since $\mathcal{F}_\theta$ determines the states $\boldsymbol{x}_t$;
- The *posterior* distribution over the stopping time t can be computed by Bayes' rule $p_\theta(t|\boldsymbol{y}, \boldsymbol{x}) \propto p_\theta(\boldsymbol{y}|t, \boldsymbol{x})p(t|\boldsymbol{x})$, but it requires the observation of the label $\boldsymbol{y}$, which is infeasible during testing phase.

In this probabilistic model, we need to learn θ to better fit the observed data and learn a variational distribution q_ϕ over t that only takes $\boldsymbol{x}$ and the transformed internal states as inputs to approximate the true posterior.

Table 5.1: Corresponds between our model and Bayes' model.

stop time t	latent variable
label $\boldsymbol{y}$	observation
loss $\ell(\boldsymbol{y}, \boldsymbol{x}_t; \theta)$	likelihood $p_\theta(\boldsymbol{y}\|t, \boldsymbol{x})$
stop time distribution q_ϕ	posterior $p_\theta(t\|\boldsymbol{y}, \boldsymbol{x})$
regularization	prior $p(t\|\boldsymbol{x})$

More specifically, the parameters in the likelihood function and the variational posterior can be optimized using the variational autoencoder (VAE) framework [123]. Here we consider a generalized version called β-VAE [124], and obtain the optimization objective for data point $(\boldsymbol{x}, \boldsymbol{y})$

$$\mathcal{J}_{\beta\text{-VAE}}(\theta, q_\phi; \boldsymbol{x}, \boldsymbol{y}) := \mathbb{E}_{q_\phi} \log p_\theta(\boldsymbol{y}|t,\boldsymbol{x}) - \beta \mathrm{KL}(q_\phi(t)||p(t|\boldsymbol{x})), \tag{5.6}$$

where $\mathrm{KL}(\cdot||\cdot)$ is the KL divergence. When $\beta = 1$, it becomes the original VAE objective, i.e., the evidence lower bound (ELBO). Now we are ready to present the equivalence relation between the β-VAE objective and the loss defined in Eq. 5.4. See [101] for the proof.

Lemma 5.3.1. *Assume (i) the loss function ℓ in Eq. 5.4 is defined as the negative log-likelihood (NLL), i.e., $\ell(\boldsymbol{y}, \boldsymbol{x}_t; \theta) := -\log p_\theta(\boldsymbol{y}|t, \boldsymbol{x})$; and (ii) the prior $p(t|\boldsymbol{x})$ is a uniform distribution over t. Then **minimizing** the loss $\mathcal{L}$ in Eq. 5.4 is equivalent to **maximizing** the β-VAE objective $\mathcal{J}_{\beta\text{-VAE}}$ in Eq. 5.6.*

For classification problems, the cross-entropy loss is aligned with NLL. For regression problems with mean squared error (MSE) loss, we can define the likelihood as $p_\theta(\boldsymbol{y}|t, \boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{x}_t, I)$. Then the NLL of this Gaussian distribution is $-\log p_\theta(\boldsymbol{y}|t, \boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{x}_t\|_2^2+C$, which is equivalent to MSE loss. More generally, we can always define $p_\theta(\boldsymbol{y}|t, \boldsymbol{x}) \propto \exp(-\ell(\boldsymbol{y}, \boldsymbol{x}_t; \theta))$.

This VAE view allows us to design a two-step procedure to effectively learn θ and ϕ in the predictive model and stopping policy, which is presented in the next section.

5.4 Effective Training Algorithm

VAE-based methods perform optimization steps over θ (M step for learning) and ϕ (E step for inference) alternatively until convergence, which has two limitations in our case:

i. The alternating training can be slow to converge and requires tuning the training scheduling;

ii. The inference step for learning q_ϕ may have the mode collapse problem, which in this case means q_ϕ only captures the time step t with highest averaged frequency.

To overcome these limitations, we design a training procedure followed by an optional fine-tuning stage using the variational lower bound in Eq. 5.6. More specifically,

Stage I. Find the optimal θ by maximizing the conditional mariginal likelihood when the stop time distribution follows an oracle distribution q^*_θ.

Stage II. Fix the optimal θ learned in Stage I, and only learn the distribution q_ϕ to mimic the oracle by minimizing the KL divergence between q_ϕ and q^*_θ.

Stage III. (Optional) Fine-tune θ and ϕ jointly towards the joint objective in Eq. 5.6.

The overall algorithm steps are summarized in Algorithm 3. In the following sections, we will focus on the derivation of the first two training steps. Then we will discuss several methods to further improve the memory and computation efficiency for training.

Oracle Stop Time Distribution. We first give the definition of the oracle stop time distribution q^*_θ. For each fixed θ, we can find a closed-form solution for the optimal q^*_θ that optimizes the joint objective.

$$q^*_\theta(\cdot|\boldsymbol{y}, \boldsymbol{x}) := \arg\max_{\boldsymbol{q}\in\Delta^{T-1}} \mathcal{J}_{\beta\text{-VAE}}(\theta, \boldsymbol{q}; \boldsymbol{x}, \boldsymbol{y})$$

Alternatively, $q^*_\theta(\cdot|\boldsymbol{y}, \boldsymbol{x}) = \arg\min_{q\in\Delta^{T-1}} \mathcal{L}(\theta, \boldsymbol{q}; \boldsymbol{x}, \boldsymbol{y})$. Under the mild assumptions in

Lemma 5.3.1, these two optimizations lead to the same optimal oracle distribution.

$$\textbf{Oracle stop time distribution:} q_\theta^*(t|\boldsymbol{y},\boldsymbol{x}) = \frac{p_\theta(\boldsymbol{y}|t,\boldsymbol{x})^{\frac{1}{\beta}}}{\sum_{t=1}^{T} p_\theta(\boldsymbol{y}|t,\boldsymbol{x})^{\frac{1}{\beta}}} \tag{5.7}$$

$$= \frac{\exp(-\frac{1}{\beta}\ell(\boldsymbol{y},\boldsymbol{x}_t;\theta))}{\sum_{t=1}^{T}\exp(-\frac{1}{\beta}\ell(\boldsymbol{y},\boldsymbol{x}_t;\theta))} \tag{5.8}$$

This closed-form solution makes it clear that the oracle picks a step t according to the smallest loss or largest likelihood with an exploration coefficient β.

Remark: When $\beta = 1$, q_θ^* is the same as the posterior distribution $p_\theta(t|\boldsymbol{y},\boldsymbol{x}) \propto p_\theta(\boldsymbol{y}|t,\boldsymbol{x})p(t|\boldsymbol{x})$.

Note that there are no new parameters in the oracle distribution. Instead, it depends on the parameters θ in the predictive model. Overall, the oracle q_θ^* is a function of θ, t, $\boldsymbol{y}$ and $\boldsymbol{x}$ that has a closed-form. Next, we will introduce how we use this oracle in the first two training stages.

5.4.1 Stage I: Predictive model learning

In Stage I, we optimize the parameters θ in the predictive model by taking into account the oracle stop distribution q_θ^* . This step corresponds to the M step for learning θ, by maximizing the marginal likelihood. The difference with the normal M step is that here q_ϕ is replaced by the oracle q_θ^* that gives the optimal stopping distribution so that the marginal likelihood is independent on ϕ. More precisely, stage I finds the optimum of:

$$\max_\theta \frac{1}{|\mathcal{D}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \mathcal{J}_{\beta-\text{VAE}}(\theta, q_\theta^*; \boldsymbol{x},\boldsymbol{y}), \tag{5.9}$$

where the β-VAE objective here is $\mathcal{J}_{\beta-\text{VAE}}(\theta, q_\theta^*; \boldsymbol{x},\boldsymbol{y}) = \sum_{t=1}^{T} q_\theta^*(t|\boldsymbol{y},\boldsymbol{x}) \log p_\theta(\boldsymbol{y}|t,\boldsymbol{x}) - \beta\text{KL}(q_\theta^*(t)||p(t|\boldsymbol{x}))$.

Remark. For experiments that require higher memory costs (e.g., MAML), we prefer to drop the entropy term, $\beta\text{KL}(q_\theta^*(t)||p(t|\boldsymbol{x}))$, in the objective, so that stochastic sampling can

Algorithm 3: Overall Algorithm

Randomly initialized θ and ϕ.
For $itr = 1$ *to #iterations* **do** ▷ Stage I.
  Sample a batch of data points $\mathcal{B} \sim \mathcal{D}$.
  Take an optimization step to update θ towards the marginal likelihood function defined in Eq. 5.9.
For $itr = 1$ *to #iterations* **do** ▷ Stage II.
  Sample a batch of data points $\mathcal{B} \sim \mathcal{D}$.
  Take an optimization step to update ϕ towards the reverse KL divergence defined in Eq. 5.10.
For $itr = 1$ *to #iterations* **do** ▷ Optional Step
  Sample a batch of data points $\mathcal{B} \sim \mathcal{D}$.
  Update both θ and ϕ towards β-VAE objective in Eq. 5.6.
return θ, ϕ

be applicable to reduce the memory cost. Since we can adjust β in the oracle q^* to control the concentration level of the distribution, dropping the entropy term in the objective in stage I does not affect much the performance.

Since q^*_θ has a differentiable closed-form expression in terms of $\theta, \boldsymbol{x}, \boldsymbol{y}$ and t, the gradient can also propagate through q^*_θ, which is also different from the normal M step.

To summarize, in Stage I., we learn the predictive model parameter θ, by assuming that the stop time always follows the best stopping distribution that depends on θ. In this case, the learning of θ has already taken into account the effect of the data-specific stop time.

However, we note that the oracle q^*_θ is not in the form of sequential actions as in Eq. 5.2 and it requires the access to the true label $\boldsymbol{y}$, so it can not be used for testing. However, it plays an important role in obtaining a sequential policy which will be explained next.

5.4.2 Stage II: Imitation with sequential policy

In Stage II, we learn the sequential policy π_ϕ that can best mimic the oracle distribution q^*_θ, where θ is fixed to be the optimal θ learned in Stage I. The way of doing so is to minimize the divergence between the oracle q^*_θ and the variational stop time distribution q_ϕ induced by π_ϕ (Eq. 5.3). There are various variational divergence minimization approaches that we

can use [125]. For example, a widely used objective for variational inference is the **reverse KL divergence**:

$$\mathrm{KL}(q_\phi||q^*_\theta) = \sum_{t=1}^{T} -q_\phi(t) \log q^*_\theta(t|\boldsymbol{y}, \boldsymbol{x}) - H(q_\phi).$$

Remark. We write $q_\phi(t)$ instead of $q_\phi(t|\boldsymbol{x}_{1:T}, \boldsymbol{x})$ for notation simplicity, but q_ϕ is dependent on $\boldsymbol{x}$ and $\boldsymbol{x}_{1:T}$ (Eq. 5.3).

If we rewrite q_ϕ using $\pi_1, \cdots, \pi_{T-1}$ as defined in Eq. 5.3, we can find that minimizing the reverse KL is equivalent to finding the optimal policy π_ϕ in a reinforcement learning (RL) environment, where the state is $\boldsymbol{x}_t$, action $a_t \sim \pi_t := \pi_\phi(\boldsymbol{x}, \boldsymbol{x}_t)$ is a stop/continue decision, the state transition is determined by θ and a_t, and the reward is defined as

$$r(\boldsymbol{x}_t, a_t; \boldsymbol{y}) := \begin{cases} -\beta \ell(\boldsymbol{y}, \boldsymbol{x}_t; \theta) & \text{if } a_t = 0 \text{ (i.e. stop)} \\ 0 & \text{if } a_t = 1 \text{ (i.e. continue)} \end{cases}$$

where $\ell(\boldsymbol{y}, \boldsymbol{x}_t; \theta) = -\log p_\theta(\boldsymbol{y}|t, \boldsymbol{x})$. Details and derivation are given in Appendix A.2 in [101] which shows minimizing $\mathrm{KL}(q_\phi||q^*_\theta)$ is equivalent to solving the following **maximum-entropy RL**:

$$\max_\phi \mathbb{E}_{\pi_\phi} \sum_{t=1}^{T} \left[r(\boldsymbol{x}_t, a_t; \boldsymbol{y}) + H(\pi_t) \right].$$

In some related literature, optimal stopping problem is often formulated as an RL problem [122]. Above we bridge the connection between our variational inference formulation and the RL-based optimal stopping literature.

Although reverse KL divergence is a widely used objective, it suffers from the mode collapse issue, which in our case may lead to a distribution q_ϕ that captures only a common stopping time t for all $\boldsymbol{x}$ that on average performs the best, instead of a more spread-out

stopping time. Therefore, we consider the **forward KL divergence**:

$$\mathrm{KL}(q_\theta^*||q_\phi) = -\sum_{t=1}^{T} q_\theta^*(t|\boldsymbol{y}, \boldsymbol{x}) \log q_\phi(t) - H(q_\theta^*), \tag{5.10}$$

which is equivalent to the **cross-entropy** loss, since the term $H(q_\theta^*)$ can be ignored as θ is fixed in this step. Experimentally, we find forward KL leads to a better performance.

The Optional Fine Tuning Stage. It is easy to see that our two-stage training procedure also has an EM flavor. However, with the oracle q_θ^* incorporated, the training of θ has already taken into account the effect of the optimal stopping distribution. Therefore, we can save a lot of alternation steps. After the two-stage training, we can fine-tune θ and ϕ jointly towards the β-VAE objective. Experimentally, we find this additional stage does not improve much the performance trained after the first two stages.

5.5 Experiments

We conduct experiments on (i) learning-based algorithm for sparse recovery, (ii) few-shot meta learning, and (iii) image denoising. The comparison is in an ablation study fashion to better examine whether the stopping policy can improve the performances given the same architecture for the predictive model, and whether our training algorithm is more effective compared to the alternating EM algorithm. In the end, we also discuss our exploration of the image recognition task. Pytorch implementation of the experiments is released at https://github.com/xinshi-chen/l2stop.

5.5.1 Learning to optimize: sparse recovery

We consider a sparse recovery task which aims at recovering $\boldsymbol{x}^* \in \mathbb{R}^n$ from its noisy linear measurements $\boldsymbol{b} = A\boldsymbol{x}^* + \boldsymbol{\epsilon}$, where $A \in \mathbb{R}^{m\times n}$, $\boldsymbol{\epsilon} \in \mathbb{R}^m$ is Gaussian white noise, and $m \ll n$. A popular approach is to model the problem as the LASSO formulation $\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{b} - A\boldsymbol{x}\|_2^2 + \rho\|\boldsymbol{x}\|_1$ and solves it using iterative methods such as the ISTA [126]

and FISTA [127] algorithms. We choose the most popular model named Learned ISTA (LISTA) as the baseline and also as our predictive model. LISTA is a T-layer network with update steps:

$$\boldsymbol{x}_t = \eta_{\lambda_t}(W_t^1 \boldsymbol{b} + W_t^2 \boldsymbol{x}_{t-1}), \quad t = 1, \cdots, T, \tag{5.11}$$

where $\theta = \{(\lambda_t, W_t^1, W_t^2)\}_{t=1}^T$ are leanable parameters.

Experiment setting. We follow [48] to generate the samples. The signal-to-noise ratio (SNR) for each sample is uniformly sampled from 20, 30, and 40. The training loss for LISTA is $\sum_{t=1}^T \gamma^{T-t} \|\boldsymbol{x}_t - \boldsymbol{x}^*\|_2^2$ where $\gamma \leq 1$. It is commonly used for algorithm-based deep learning, so that there is a supervision signal for every layer. For ISTA and FISTA, we use the training set to tune the hyperparameters by grid search.

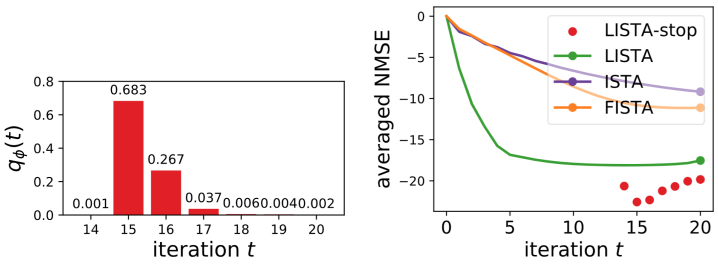
Table 5.2: Recovery performances of different algorithms/models.

SNR	mixed	20	30	40
FISTA ($T = 100$)	-18.96	-16.75	-20.46	-20.97
ISTA ($T = 100$)	-14.66	-13.99	-14.99	-15.07
ISTA ($T = 20$)	-9.17	-9.12	-9.24	-9.16
FISTA ($T = 20$)	-11.12	-10.98	-11.19	-11.19
LISTA ($T = 20$)	-17.58	-16.52	-18.16	-18.29
LISTA-stop ($T \leq 20$)	**-22.41**	**-20.29**	**-23.90**	**-24.21**

Recovery performance. (Table 5.2) We report the NMSE (in dB) results for each model/algorithm evaluated on 1000 fixed test samples per SNR level. It is revealed in Table 5.2 that learning-based methods have better recovery performances, especially for the more difficult tasks (i.e. when SNR is 20). Compared to LISTA, our proposed adaptive-stopping method (LISTA-stop) significantly improve recovery performance. Also, LISTA-stop with ≤ 20 iterations performs better than ISTA and FISTA with 100 iterations, which indicates a better convergence.

Stopping distribution. The stop time distribution $q_\phi(t)$ induced by π_ϕ can be computed via Eq. 5.3. We report in Fig. 5.4 the stopping distribution averaged over the test samples,

from which we can see that with a high probability LISTA-stop terminates the process before arriving at 20-th iteration.



(a) stop time distribution (b) convergence

Figure 5.4: *Left*: Stop time distribution averaged over the test set. *Right*: Convergence of different algorithms. For LISTA-stop, the NMSE weighted by the stopping distribution q_ϕ is plotted. In the first 13 iterations $q_\phi(t) = 0$, so no red dots are plotted.

Convergence comparison. Fig. 5.4 shows the change of NMSE as the number of iterations increases. Since LISTA-stop outputs the results at different iteration steps, it is not meaningful to draw a unified convergence curve. Therefore, we plot the NMSE weighted by the stopping distribution q_ϕ, i.e., $10\log_{10}\big(\frac{\sum_{i=1}^N q_\phi(t|i)\|\boldsymbol{x}_t-\boldsymbol{x}^{*,i}\|_2^2}{\sum_{i=1}^N q_\phi(t|i)}\big/\big(\frac{\sum_{i=1}^N \|\boldsymbol{x}^{*,i}\|_2^2}{N}\big)$, using the red dots. We observe that for LISTA-stop the expected NMSE increases as the number of iterations increase, this might indicate that the later stopped problems are more difficult to solve. Besides, at 15th iteration, the NMSE in Fig. 5.4 (b) is the smallest, while the averaged stop probability mass $q_\phi(15)$ in Fig. 5.4 (a) is the highest.

Table 5.3: Different algorithms for training LISTA-stop.

SNR	mixed	20	30	40
AEVB algorithm	-21.92	-19.92	-23.27	-23.58
Stage I. + II.	**-22.41**	**-20.29**	**-23.90**	**-24.21**
Stage I.+II.+III.	**-22.78**	**-20.59**	**-24.29**	**-24.73**

Ablation study on training algorithms. To show the effectiveness of our two-stage training, in Table 5.3, we compare the results with the auto-encoding variational Bayes (AEVB) algorithm [128] that jointly optimizes $\mathcal{F}_\theta$ and q_ϕ. We observe that the distribution q_ϕ in AEVB gradually becomes concentrated on one layer and does not get rid of this

Table 5.4: Few-shot classification in vanilla meta learning setting [89] where all tasks have the same number of data points.

	Omniglot 5-way		Omniglot 20-way		MiniImagenet 5-way	
	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
MAML	$98.7 \pm 0.4\%$	$99.1 \pm 0.1\%$	$95.8 \pm 0.3\%$	$98.9 \pm 0.2\%$	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$
MAML-stop	$\mathbf{99.62 \pm 0.22\%}$	$\mathbf{99.68 \pm 0.12\%}$	$\mathbf{96.05 \pm 0.35\%}$	$\mathbf{98.94 \pm 0.10\ \%}$	$\mathbf{49.56 \pm 0.82\%}$	$\mathbf{63.41 \pm 0.80\%}$

local minimum, making its final result not as good as the results of our two-stage training. Moreover, it is revealed that Stage III does not improve much of the performance of the two-stage training, which also in turn shows the effectiveness of the oracle-based two-stage training.

5.5.2 Task-imbalanced meta learning

In this section, we perform meta learning experiments in the few-short learning domain [108].

Experiment setting. We follow the setting in MAML [89] for the few-shot learning tasks. Each task is an N-way classification that contains meta-{train, valid, test} sets. On top of it, the macro dataset with multiple tasks is split into train, valid and test sets. We consider the more realistic *task-imbalanced* setting proposed by [112]. Unlike the standard setting where the meta-train of each task contains k-shots for each class, here we vary the number of observation to perform k_1- k_2-shot learning where $k_1 < k_2$ are the minimum/maximum number of observations per class, respectively. Build on top of MAML, we denote our variant as MAML-stop which learns how many adaptation gradient descent steps are needed for each task. Intuitively, the tasks with less training data would prefer fewer steps of gradient-update to prevent overfitting. As we mainly focus on the effect of learning to stop, the neural architecture and other hyperparameters are largely the same as MAML.

Dataset. We use the benchmark datasets Omniglot [129] and MiniImagenet [108]. Omniglot consists of 20 instances of 1623 characters from 50 different alphabets, while MiniImagenet involves 64 training classes, 12 validation classes, and 24 test classes. We use exactly the same data split as [89]. To construct the imbalanced tasks, we perform 20-way

1-5 shot classification on Omniglot and 5-way 1-10 shot classification on MiniImagenet. The number of observations per class in each meta-test set is 1 and 5 for Omniglot and MiniImagenet, respectively. For evaluation, we construct 600 tasks from the held-out test set for each setting.

Table 5.5: Task-imbalanced few-shot image classification.

	Omniglot 20-way, 1-5 shot	MiniImagenet 5-way, 1-10 shot
MAML	$97.96 \pm 0.3\%$	$57.20 \pm 1.1\%$
MAML-stop	$\mathbf{98.45 \pm 0.2\%}$	$\mathbf{60.67 \pm 1.0\%}$

Results. Table 5.5 summarizes the accuracy and the 95% confidence interval on the held-out tasks for each dataset. The maximum number of adaptation gradient descent steps is 10 for both MAML and MAML-stop. We can see the optimal stopping variant of MAML outperforms the vanilla MAML consistently. For a more difficult task on MiniImagenet where the imbalance issue is more severe, the accuracy improvement is 3.5%. For completeness, we include the performance on vanilla meta learning setting where all tasks have the same number of observations in Table 5.4. MAML-stop still achieves comparable or better performance.

5.5.3 Image denoising

In this section, we perform the image denoising experiments.

Dataset. The models are trained on BSD500 (400 images) [130], validated on BSD12, and tested on BSD68 [131]. We follow the standard setting in [55, 132, 133] to add Gaussian noise to the images with a random noise level $\sigma \leq 55$ during training and validation phases.

Experiment setting. We compare with two DL models, DnCNN [133] and UNLNet$_5$ [132], and two traditional methods, BM3D [134] and WNNM [135]. Since DnCNN is one of the most widely-used models for image denoising, we use it as our predictive model.

All deep models including ours are considered in the *blind* Gaussian denoising setting, which means the noise-level is not given to the model, while BM3D and WNNM require the noise-level to be known.

Table 5.6: PSNA performance comparison. The sign * indicates that noise levels 65 and 75 do not appear in the training set.

σ	DnCNN-stop	DnCNN	UNLNet$_5$	BM3D	WNNM
35	**27.61**	27.60	27.50	26.81	27.36
45	**26.59**	26.56	26.48	25.97	26.31
55	**25.79**	25.71	25.64	25.21	25.50
*65	**23.56**	22.19	-	24.60	**24.92**
*75	**18.62**	17.90	-	24.08	**24.39**

Results. The performance is evaluated by the mean peak signal-to-noise ratio (PSNR). Table 5.6 shows that DnCNN-stop performs better than the original DnCNN. Especially, for images with noise levels 65 and 75 which are **unseen** during training phase, DnCNN-stop generalizes significantly better than DnCNN alone. Since there is no released code for UNLNet$_5$, its performances are copied from the paper [132], where results are not reported for $\sigma = 65$ and 75. For traditional methods BM3D and WNNM, the test is in the noise-specific setting. That is, the noise level is given to both BM3D and WNNM, so the comparison is not completely fair to learning based methods in blind denoising setting.

5.5.4 Image recognition

We explore the potential of our idea for improving the recognition performances on Tiny-ImageNet, using VGG16 [136] as the predictive model. With 14 internal classifiers, after Stage I training, if the oracle q_θ^* is used to determine the stop time t, the accuracy of VGG16 can be improved to 83.26%. Similar observation is provided in SDN [117], but their loss $\sum_t w_t \ell_t$ depends on very careful hand-tuning on the weight w_t for each layer, while we directly take an expectation using the oracle, which is more principled and leads to higher accuracy (Table 5.7). However, it reveals to be very hard to mimic the behavior of the

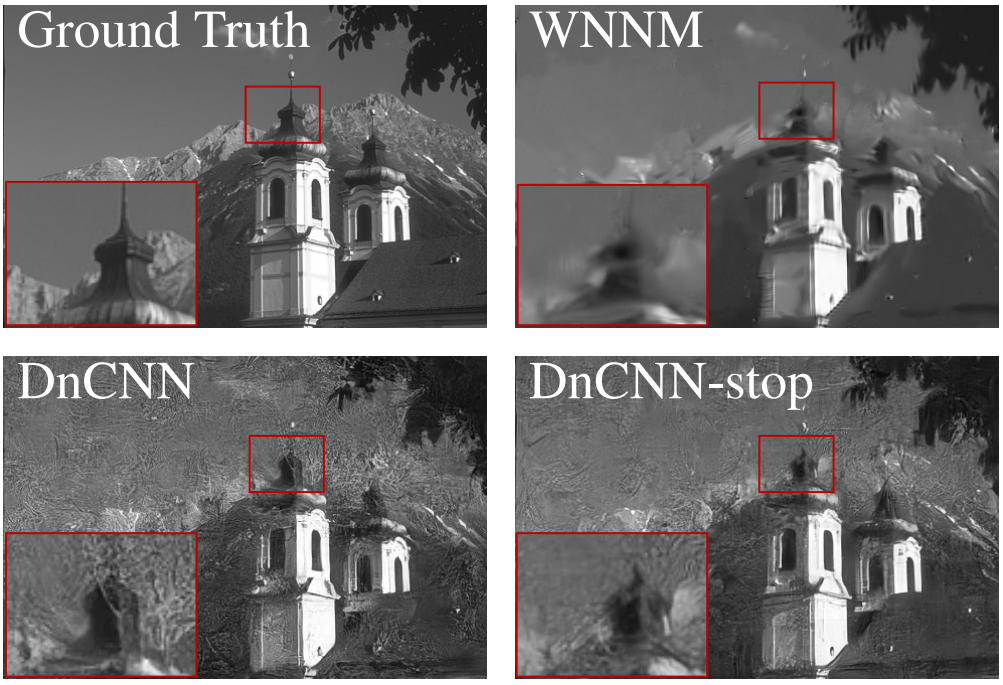


Figure 5.5: Denoising results of an image with noise level 65.

orcale q_θ^* by π_ϕ in Stage II, either due to the need of a better parametrization for π_ϕ or more sophisticated reasons. Our learned π_ϕ leads to similar accuracy as the heuristic policy in SDN, which becomes the bottleneck in our exploration. However, based on the large performance gap between the oracle and the original VGG16, our result still provides a potential direction for breaking the performance bottleneck of DL on image recognition.

Table 5.7: Image recognition with oracle stop distribution.

VGG16	SDN training	Our Stage I. training
58.60%	77.78% (best layer)	83.26% (best layer)

Part II

Deep Learning Based Algorithm Design

CHAPTER 6

LEARNING TO ESTIMATE SPARSE PRECISION MATRIX

Recovering sparse conditional independence graphs from data is a fundamental problem in machine learning with wide applications. A popular formulation of the problem is an ℓ_1 regularized maximum likelihood estimation. Many optimization algorithms (such as the Graphical Lasso Algorithm [137]) have been designed to solve this formulation to recover the graph structure.

This chapter presents a deep learning method, called GLAD (Graph recovery Learning Algorithm by Data-driven training), to learn an data-driven algorithm for recovering the the sparse precision matrix from empirical covariance.

It is challenging to design the architecture for this task, since the symmetric positive definiteness (SPD) and sparsity of the matrix are not easy to enforce in a deep learning model. In this chapter, we will explain how we use an Alternating Minimization (AM) algorithm as the inductive bias to design the architecture of GLAD.

The research in this chapter was previously presented at the ICLR 2020 conference in [43].

6.1 Introduction

Very often a family of optimization problems needs to be solved again and again, similar in structures but different in data. A data-driven algorithm may be able to leverage this distribution of problem instances, and learn an algorithm which performs better than traditional convex formulation. In the case of sparse graph recovery problems, the precision matrix may also need to be estimated again and again, where the underlying graphs are different but have similar degree distribution, the magnitude of the precision matrix entries, etc. For instance, gene regulatory networks may be rewiring depending on the time and conditions,

and we want to estimate them from gene expression data.

Given a task (e.g. an optimization problem), an algorithm will solve it and provide a solution. Thus we can view an algorithm as a function mapping, where the input is the task-specific information (i.e. the sample covariance matrix in our case) and the output is the solution (i.e. the estimated precision matrix in our case).

However, it is very challenging to design a data-driven algorithm for precision matrix estimation. First, the input and output of the problem may be large. A neural network parameterization of direct mapping from the input covariance matrix to the output precision matrix may require as many parameters as the square of the number of dimensions. Second, there are many structure constraints in the output. The resulting precision matrix needs to be positive definite and sparse, which is not easy to enforce by a simple deep learning architecture. Third, direct mapping may result in a model with lots of parameters, and hence may require lots of data to learn. Thus a data-driven algorithm needs to be designed carefully to achieve a better bias-variance trade-off and satisfy the output constraints.

In this chapter, we propose a deep learning model 'GLAD' with following attributes:

- Uses an unrolled Alternating Minimization (AM) algorithm as an inductive bias.
- The regularization and the square penalty terms are parameterized as entry-wise functions of intermediate solutions, allowing GLAD to learn to perform entry-wise regularization update.
- Furthermore, this data-driven algorithm is trained with a collection of problem instances in a supervised fashion, by directly comparing the algorithm outputs to the ground truth graphs.

In our experiments, we show that the AM architecture provides very good inductive bias, allowing the model to learn very effective sparse graph recovery algorithm with a small amount of training data. In all cases, the learned algorithm can recover sparse graph structures with much fewer data points from a new problem, and it also works well in recovering gene regulatory networks based on realistic gene expression data generators.

6.2 Sparse Graph Recovery Problem and Convex Formulation

Given m observations of a d-dimensional multivariate Gaussian random variable $X = [X_1, \ldots, X_d]^\top$, the sparse graph recovery problem aims to estimate its covariance matrix Σ^* and precision matrix $\Theta^* = (\Sigma^*)^{-1}$. The ij-th component of Θ^* is zero if and only if X_i and X_j are conditionally independent given the other variables $\{X_k\}_{k \neq i,j}$. Therefore, it is popular to impose an ℓ_1 regularization for the estimation of Θ^* to increase its sparsity and lead to easily interpretable models. Following [138], the problem is formulated as the ℓ_1-regularized maximum likelihood estimation

$$\widehat{\Theta} = \arg\min_{\Theta \in \mathcal{S}^d_{++}} \quad -\log(\det \Theta) + \operatorname{tr}(\widehat{\Sigma}\Theta) + \rho \left\|\Theta\right\|_{1,\text{off}}, \tag{6.1}$$

where $\widehat{\Sigma}$ is the empirical covariance matrix based on m samples, $\mathcal{S}^d_{++}$ is the space of $d \times d$ symmetric positive definite matrices (SPD), and $\left\|\Theta\right\|_{1,\text{off}} = \sum_{i \neq j} |\Theta_{ij}|$ is the off-diagonal ℓ_1 regularizer with regularization parameter ρ. The sparse precision matrix estimation problem in Eq. 6.1 is a convex optimization problem which can be solved by many algorithms. A few canonical and advanced examples include the G-ISTA algorithm [139], ADMM [140], BCD [137], etc.

6.3 Learning Data-Driven Algorithm for Precision Matrix Estimation

In the remainder of this chapter, we will present a data-driven method to learn an algorithm for precision matrix estimation, and we call the resulting algorithm GLAD. We ask the question of

Given a family of precision matrices, is it possible to improve recovery results for sparse graphs by learning a data-driven algorithm?

Formally, suppose we are given n precision matrices $\{\Theta^{*(i)}\}_{i=1}^n$ from a family $\mathcal{G}$ of graphs and m samples $\{\boldsymbol{x}^{(i,j)}\}_{j=1}^m$ associated with each $\Theta^{*(i)}$. These samples can be used

to form n sample covariance matrices $\{\widehat{\Sigma}^{(i)}\}_{i=1}^{n}$. We are interested in learning an algorithm for precision matrix estimation by solving a supervised learning problem,

$$\min_{f} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\text{GLAD}_f(\widehat{\Sigma}^{(i)}), \Theta^{*(i)}),$$

where f is a set of parameters in $\text{GLAD}(\cdot)$ and the output of $\text{GLAD}_f(\widehat{\Sigma}^{(i)})$ is expected to be a good estimation of $\Theta^{*(i)}$ in terms of an interested evaluation metric $\mathcal{L}$.

6.3.1 Architecture

As mentioned earlier in Section 6.1, it is a challenging task to design a good parameterization of GLAD_f for this graph recovery problem. There is a list of desiderata for this model:

- Small model size;
- Interpretable architecture;
- Meaningful intermediate outputs; and
- Semi positive definiteness of the output.

To take into account the above desiderata, our GLAD model is designed based on a reformulation of the original optimization problem in Eq. 6.1 with a squared penalty term, and an alternating minimization (AM) algorithm for solving it. More specifically, we consider a modified optimization with a quadratic penalty parameter λ:

$$\widehat{\Theta}_\lambda, \widehat{Z}_\lambda := \arg\min_{\Theta, Z \in \mathcal{S}^d_{++}} -\log(\det \Theta) + \text{tr}(\widehat{\Sigma}\Theta) + \rho \|Z\|_1 + \tfrac{1}{2}\lambda \|Z - \Theta\|_F^2 \tag{6.2}$$

and the alternating minimization (AM) method for solving it:

$$\Theta^{\text{AM}}_{k+1} \leftarrow \tfrac{1}{2}\big(-Y + \sqrt{Y^\top Y + \tfrac{4}{\lambda} I}\big), \text{ where } Y = \tfrac{1}{\lambda}\widehat{\Sigma} - Z^{\text{AM}}_k; \tag{6.3}$$

$$Z^{\text{AM}}_{k+1} \leftarrow \eta_{\rho/\lambda}(\Theta^{\text{AM}}_{k+1}), \tag{6.4}$$

Algorithm 4: GLAD

Function `GLADcell` $(\widehat{\Sigma}, \Theta, Z, \lambda)$ **:**

$\lambda \leftarrow \Lambda_{nn}(\|Z - \Theta\|_F^2, \lambda)$

$Y \leftarrow \lambda^{-1}\widehat{\Sigma} - Z$

$\Theta \leftarrow \frac{1}{2}\big(-Y + \sqrt{Y^\top Y + \frac{4}{\lambda}I}\big)$

For *all* i, j **do**

$\rho_{ij} = \rho_{nn}(\Theta_{ij}, \widehat{\Sigma}_{ij}, Z_{ij})$

$Z_{ij} \leftarrow \eta_{\rho_{ij}}(\Theta_{ij})$

return Θ, Z, λ

Function `GLAD` $(\widehat{\Sigma})$ **:**

$\Theta_0 \leftarrow (\widehat{\Sigma} + tI)^{-1}$, $\lambda_0 \leftarrow 1$

For $k = 0$ *to* $K - 1$ **do**

$\Theta_{k+1}, Z_{k+1}, \lambda_{k+1}$

$\leftarrow$`GLADcell` $(\widehat{\Sigma}, \Theta_k, Z_k, \lambda_k)$

return Θ_K, Z_K

where $\eta_{\rho/\lambda}(\theta) := \text{sign}(\theta)\max(|\theta| - \rho/\lambda, 0)$. We replace the penalty constants (ρ, λ) by problem dependent neural networks, ρ_{nn} and Λ_{nn}. These neural networks are minimalist in terms of the number of parameters as the input dimensions are mere $\{3, 2\}$ for $\{\rho_{nn}, \Lambda_{nn}\}$ and outputs a single value. Algorithm 4 summarizes the update equations for our unrolled AM based model, GLAD. Except for the parameters in ρ_{nn} and Λ_{nn}, the constant t for initialization is also a learnable scalar parameter. This unrolled algorithm with neural network augmentation can be viewed as a highly structured recurrent architecture.

There are many traditional algorithms for solving graph recovery problems. We choose AM as our basis because: First, empirically, we tried models built upon other algorithms including G-ISTA, ADMM, etc, but AM-based model gives consistently better performances. Second, and more importantly, the AM-based architecture has a nice property of maintaining Θ_{k+1} as a SPD matrix throughout the iterations as long as $\lambda_k < \infty$. Third, as we prove later in Section 6.4, the AM algorithm has linear convergence rate, allowing us to use a fixed small number of iterations and still achieve small error margins.

6.3.2 Training algorithm

To learn the parameters in GLAD architecture, we will directly optimize the recovery objective function rather than using log-determinant objective. A nice property of our deep learning architecture is that each iteration of our model will output a valid precision matrix estimation. This allows us to add auxiliary losses to regularize the intermediate results of our GLAD architecture, guiding it to learn parameters which can generate a smooth solution trajectory.

Specifically, we will use Frobenius norm in our experiments, and design an objective which has some resemblance to the discounted cumulative reward in reinforcement learning:

$$\min_{f} \ \text{loss}_f := \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\gamma^{K-k}\left\|\Theta_k^{(i)} - \Theta^*\right\|_F^2, \tag{6.5}$$

where $(\Theta_k^{(i)}, Z_k^{(i)}, \lambda_k^{(i)}) = \texttt{GLADcell}_f(\widehat{\Sigma}^{(i)}, \Theta_{k-1}^{(i)}, Z_{k-1}^{(i)}, \lambda_{k-1}^{(i)})$ is the output of the recurrent unit `GLADcell` at k-th iteration, K is number of unrolled iterations, and $\gamma \leq 1$ is a discounting factor. We will use stochastic gradient descent algorithm to train the parameters f in the `GLADcell`.

6.4 Theoretical Analysis

Since GLAD architecture is obtained by augmenting an unrolled optimization algorithm by learnable components, the question is what kind of guarantees can be provided for such learned algorithm, and whether learning can bring benefits to the recovery of the precision matrix. In this section, we will first analyze the statistical guarantee of running the AM algorithm in Eq. 6.3 and Eq. 6.4 for k steps with a fixed quadratic penalty parameter λ, and then interpret its implication for the learned algorithm. First, we need some standard assumptions about the true model from the literature [141]:

Assumption 6.4.1. *Let the set* $S = \{(i,j) : \Theta^*_{ij} \neq 0, i \neq j\}$. *Then card*$(S) \leq s$.

Assumption 6.4.2. $\Lambda_{\min}(\Sigma^*) \geq \epsilon_1 > 0$ *(or equivalently* $\Lambda_{\max}(\Theta^*) \leq 1/\epsilon_1$*),* $\Lambda_{\max}(\Sigma^*) \leq \epsilon_2$ *and an upper bound on* $\|\widehat{\Sigma}\|_2 \leq c_{\widehat{\Sigma}}$.

The assumption 6.4.2 guarantees that Θ^* exists. Assumption 6.4.1 just upper bounds the sparsity of Θ^* and does not stipulate anything in particular about s. These assumptions characterize the fundamental limitation of the sparse graph recovery problem, beyond which recovery is not possible. Under these assumptions, we prove the linear convergence of AM algorithm.

Theorem 6.4.1. *Under the assumptions 6.4.1 & 6.4.2, if* $\rho \asymp \sqrt{\frac{\log d}{m}}$*, where* ρ *is the* l_1 *penalty,* d *is the dimension of problem and* m *is the number of samples, the Alternate Minimization algorithm has linear convergence rate for optimization objective defined in (6.2). The* k^{th} *iteration of the AM algorithm satisfies,*

$$\left\|\Theta_k^{AM} - \Theta^*\right\|_F \leq C_\lambda \left\|\Theta_{k-1}^{AM} - \widehat{\Theta}_\lambda\right\|_F + \mathcal{O}_{\mathbb{P}}\left(\sqrt{\frac{(\log d)/m}{\min(\frac{1}{(d+s)}, \frac{\lambda}{d^2})}}\right), \tag{6.6}$$

where $0 < C_\lambda < 1$ *is a constant depending on* λ.

From the theorem, one can see that by optimizing the quadratic penalty parameter λ, one can adjust the C_λ in the bound. We observe that at each stage k, an optimal penalty parameter λ_k can be chosen depending on the most updated value C_λ. An adaptive sequence of penalty parameters $(\lambda_1, \ldots, \lambda_K)$ should achieve a better error bound compared to a fixed λ. Since C_λ is a very complicated function of λ, the optimal λ_k is hard to choose manually.

Besides, the linear convergence guarantee in this theorem is based on the sparse regularity parameter $\rho \asymp \sqrt{\frac{\log d}{m}}$. However, choosing a good ρ value in practice is tedious task as shown in our experiments.

In summary, the implications of this theorem are:

• An adaptive sequence $(\lambda_1, \ldots, \lambda_K)$ should lead to an algorithm with better convergence than a fixed λ, but the sequence may not be easy to choose manually.

• Both ρ and the optimal λ_k depend on the corresponding error $\|\Theta^{\text{AM}} - \widehat{\Theta}_\lambda\|_F$, which make these parameters hard to prescribe manually.

• Since, the AM algorithm has a fast linear convergence rate, we can run it for a fixed number of iterations K and still converge with a reasonable error margin.

6.5 Experiments

In this section, we report several experiments to compare GLAD with traditional algorithms and other data-driven algorithms. The results validate the list of desiderata mentioned previously. Especially, it shows the potential of pushing the boundary of traditional graph recovery algorithms by utilizing data. Python implementation is available[1].

Evaluation metric. We use normalized mean square error (NMSE) and probability of success (PS) to evaluate the algorithm performance. NMSE is $10\log_{10}(\mathbb{E}\,\|\Theta^p - \Theta^*\|_F^2 / \mathbb{E}\,\|\Theta^*\|_F^2)$ and PS is the probability of correct signed edge-set recovery, i.e.,

$$\mathbb{P}\left[\text{sign}(\Theta^p_{ij}) = \text{sign}(\Theta^*_{ij}), \forall (i,j) \in \mathbf{E}(\Theta^*)\right],$$

where $\mathbf{E}(\Theta^*)$ is the true edge set.

Notation. In all reported results, D stands for dimension d of the random variable, M stands for sample size and N stands for the number of graphs (precision matrices) that is used for training.

6.5.1 Benefit of data-driven gradient-based algorithm

Inconsistent optimization objective. Traditional algorithms are typically designed to optimize the ℓ_1-penalized log likelihood. Since it is a convex optimization, convergence to optimal solution is usually guaranteed. However, this optimization objective is different from the true error. Taking ADMM as an example, it is revealed in Figure 6.1 that, although

[1] https://github.com/Harshs27/GLAD

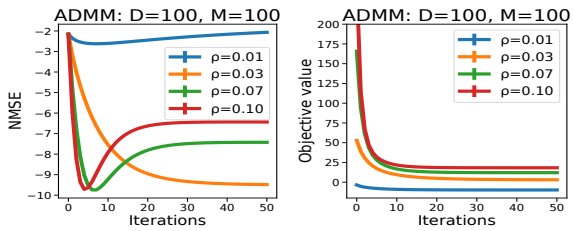


Figure 6.1: Convergence of ADMM in terms of NMSE and optimization objective.

the optimization objective always converges, errors of recovering true precision matrices measured by NMSE have very different behaviors given different regularity parameter ρ, which indicates the necessity of directly optimizing NMSE and hyperparameter tuning.

6.5.2 Recovery probability

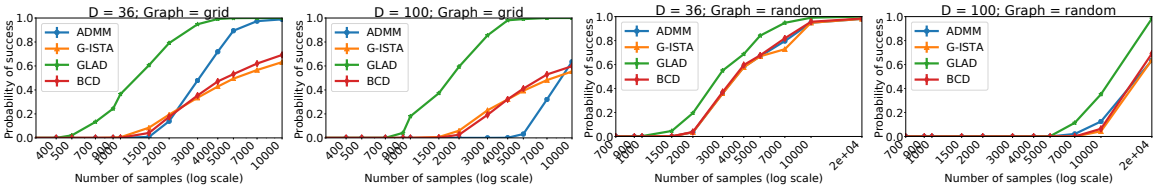


Figure 6.2: Sample complexity for model selection consistency.

As analyzed by [142], the recovery guarantee (such as in terms of Frobenius norm) of the ℓ_1 regularized log-determinant optimization significantly depends on the sample size and other conditions. Our GLAD directly optimizes the recovery objective based on data, and it has the potential of pushing the sample complexity limit. We experimented with this and found the results positive.

We follow [142] to conduct experiments on GRID graphs, which satisfy the conditions required in [142]. Furthermore, we conduct a more challenging task of recovering restricted but randomly constructed graphs. The probability of success (PS) is non-zero only if the algorithm recovers all the edges with correct signs, plotted in Figure 6.2. GLAD consistently outperforms traditional methods in terms of sample complexity as it recovers the true edges with considerably fewer number of samples.

6.5.3 Gene regulation data

The SynTReN [143] is a synthetic gene expression data generator specifically designed for analyzing the sparse graph recovery algorithms. It models different types of biological interactions and produces biologically plausible synthetic gene expression data. Figure 6.3 shows that GLAD performs favourably for structure recovery in terms of NMSE on the gene expression data. As the governing equations of the underlying distribution of the SynTReN are unknown, these experiments also emphasize the ability of GLAD to handle non-Gaussian data.

Figure 6.4 visualizes the edge-recovery performance of GLAD models trained on a sub-network of true Ecoli bacteria data. We denote, TPR: True Positive Rate, FPR: False Positive Rate, FDR: False Discovery Rate. The number of simulated training/validation graphs were set to 20/20. One batch of M samples were taken per graph. Although, GLAD was trained on graphs with $D = 25$, it was able to robustly recover a higher dimensional graph $D = 43$ structure.

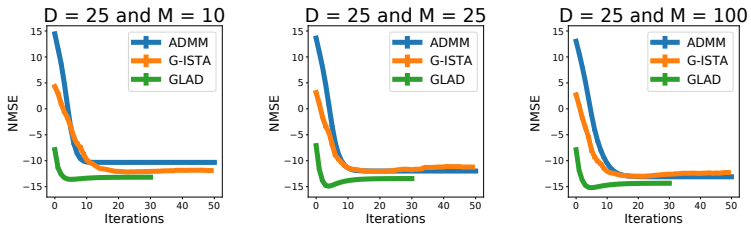


Figure 6.3: Performance on the SynTReN generated gene expression data with graph as Erdos-renyi having sparsity $p = 0.05$.

(a) True graph (b) **M=10**, fdr=0.613, tpr=0.913, fpr=0.114 (c) **M=100**, fdr=0.236, tpr=0.986, fpr=0.024

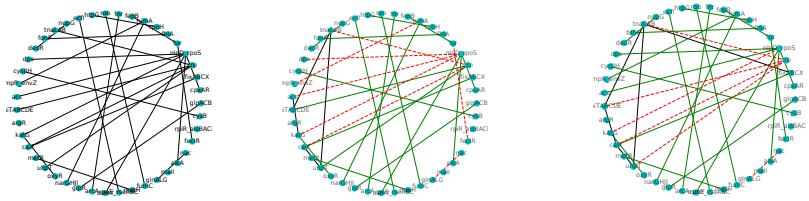


Figure 6.4: Recovered graph structures for a sub-network of the *E. coli* consisting of 43 genes and 30 interactions with increasing samples. Increasing the samples reduces the fdr by discovering more true edges.

CHAPTER 7

PROVABLE LEARNING-BASED ALGORITHM FOR SPARSE RECOVERY

Recovering sparse parameters from observational data is a fundamental problem in machine learning with wide applications. Many classic algorithms can solve this problem with theoretical guarantees, but their performances rely on choosing the correct hyperparameters. Besides, hand-designed algorithms do not fully exploit the particular problem distribution of interest. In this chapter, we propose a deep learning method for algorithm learning called PLISA (Provable Learning-based Iterative Sparse recovery Algorithm). PLISA is designed by unrolling a classic path-following algorithm for sparse recovery, with some components being more flexible and learnable. We theoretically show the improved recovery accuracy achievable by PLISA. Furthermore, we analyze the empirical Rademacher complexity of PLISA to characterize its generalization ability to solve new problems outside the training set. This chapter contains novel theoretical contributions to the area of learning-based algorithms in the sense that (i) PLISA is generically applicable to a broad class of sparse estimation problems, (ii) generalization analysis has received less attention so far, and (iii) our analysis makes novel connections between the generalization ability and algorithmic properties such as stability and convergence of the unrolled algorithm, which leads to a tighter bound that can explain the empirical observations. The techniques could potentially be applied to analyze other learning-based algorithms in the literature.

The research in this chapter was previously presented at the ICLR 2022 conference in [144].

7.1 Introduction

The problem of recovering a sparse vector $\boldsymbol{\beta}^*$ from finite observations $Z_{1:n} \sim (\mathbb{P}_{\boldsymbol{\beta}^*})^n$ is fundamental in machine learning, covering a broad family of problems including compressed sensing, sparse regression analysis, graphical model estimation, etc. It has also found applications in various domains. For example, in magnetic resonance imaging, sparse signals need to be reconstructed from measurements taken by a scanner. In computational biology, estimating a sparse graph structure from gene expression data is important for understanding gene regulatory networks.

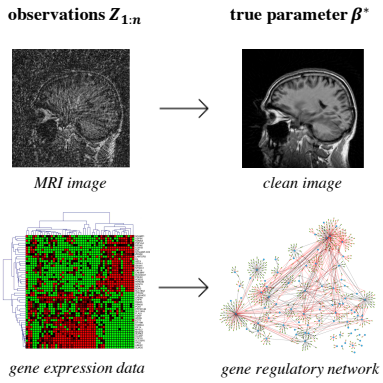


Figure 7.1: Sparse recovery problems.

Various classic algorithms are available for solving sparse recovery problems. Many of them come with theoretical guarantees for the recovery accuracy. However, the theoretical performance often relies on choosing the correct hyperparameters, such as regularization parameters and the learning rate, which may depend on unknown constants. Furthermore, in practice, similar problems may need to be solved repeatedly, but it is hard for classic algorithms to fully utilize this information.

To alleviate these limitations, we consider the approach of learning-to-learn and propose a neural algorithm, called `PLISA` (`P`rovable `L`earning-based `I`terative `S`parse recovery `A`lgorithm). `PLISA` is a deep learning model that takes the observations $Z_{1:n}$ as the input and outputs an estimation for $\boldsymbol{\beta}^*$. To make use of classic techniques developed by domain

experts, we design the architecture of PLISA by *unrolling and modifying a classic path-following algorithm* proposed by [145]. To benefit from learning, some components in this classic algorithm are made more flexible with careful design and treated as learnable parameters in PLISA. These parameters can be learned by optimizing the performances on a set of training problems. The learned PLISA can then be used for solving other problems in the target distribution.

With the algorithm design problem converted to a deep learning problem, we ask the two fundamental questions in learning theory:

1. **Capacity:** What's the recovery accuracy achievable by PLISA? Can the flexible components in PLISA lead to an algorithm which effectively improves the recovery performance?
2. **Generalization:** How well can the learned PLISA solve new problems outside the training set? Is the generalization behavior related to the algorithmic properties of PLISA?

Aiming at supplying rigorous answers to these questions, we conduct theoretical analysis for PLISA to provide guarantees for its representation and generalization ability. The results and the techniques in our analysis can distinguish our work from existing studies on algorithm learning. We summarize our novel contributions into the following three aspects.

1. Theoretical understanding. In contrast to the plethora of empirical studies on algorithm learning, there have been relatively few studies devoted to the theoretical understanding. Existing theoretical efforts primarily focus on analyzing the convergence rate achievable by the neural algorithm [48, 49, 59, 50], but the generalization error bound has received less attention so far. A substantial body of works only argue intuitively that algorithm unrolling architectures can generalize well because they contain a small number of parameters. In comparison, we provide theoretical guarantees for both the capacity and the generalization ability of PLISA, which are more solid arguments.

2. General setting. The problem setting in this chapter is new and more challenging.

Existing works mainly focus on a specific problem. For example, the compressed sensing problem with a fixed design matrix is the mostly investigated one. PLISA, however, is generic and is applicable to various sparse recovery problems as long as they satisfy certain conditions in Assumption 7.4.1.

3. Novel connection. The algorithmic structure in PLISA can make it behaves differently from conventional neural networks. Therefore, we largely utilize the analysis techniques in classic algorithms to derive its generalization bound. By combining the analysis tools of deep learning theory and optimization algorithms, our result reveals a novel connection between the generalization ability of PLISA and the algorithmic properties including the convergence rate and stability of the unrolled algorithm. Benefit from this connection, our generalization bound is tight in the sense that it matches the interesting behavior of PLISA observed in experiments - the generalization gap could decrease in the number of layers, which is rarely observed in conventional neural networks.

7.2 Related Work

Learning-to-learn has become an active research direction in recent years [146, 147, 20, 148, 21, 149, 22]. Many works share the idea of unrolling or differentiating through algorithms to design the architecture [39, 40, 41, 42, 43, 26, 44, 150, 151, 152]. A well-known example of learning-based algorithm is LISTA [45] which interprets ISTA [46] as layers of neural networks and has been an active research topic [59, 47, 48, 49, 50, 51].

However, the generalization of algorithm learning has received less attention. The only exceptions are several works. However, [75] and [153] only consider learning to optimize quadratic losses. [154, 155] do not connect the generalization analysis with algorithmic properties to provide tighter bounds as in our work. Unlike our work that analyzes the Lipschitz continuity (Lemma 7.5.1), the work of [156, 157] studied the generalization of learning-based algorithms with a focus on scenarios when the Lipschitz continuity is unavailable. We will refer the audience to [57, 58] for a more comprehensive summary of

related works.

7.3 PLISA: Learning To Solve Sparse Estimation Problems

A sparse estimation problem is to recover $\boldsymbol{\beta}^*$ from finite observations $Z_{1:n}$ sampled from $\mathbb{P}_{\boldsymbol{\beta}^*}$. As a concrete example, in a sparse linear regression problem, n observations $\{Z_i = (\boldsymbol{x}_i, y_i)\}_{i=1}^n$ are sampled from a linear model $y = \boldsymbol{x}^\top \boldsymbol{\beta}^* + \epsilon$, and an algorithm needs to estimate $\boldsymbol{\beta}^*$ from n observations. Classic algorithms recover $\boldsymbol{\beta}^*$ by minimizing a regularized empirical loss:

$$\widehat{\boldsymbol{\beta}}_\lambda \in \arg\min L_n(Z_{1:n}, \boldsymbol{\beta}) + P(\lambda, \boldsymbol{\beta}), \tag{7.1}$$

where L_n is an empirical loss that measures the "fit" between the parameter $\boldsymbol{\beta}$ and observations $Z_{1:n}$, and $P(\lambda, \boldsymbol{\beta})$ is a sparsity regularization with coefficient λ. When L_n is the least square loss and $P(\lambda, \boldsymbol{\beta})$ is $\lambda\|\boldsymbol{\beta}\|_1$, the optimization is known as LASSO and can be solved by the well-known algorithm ISTA [46]. Based on the idea of **algorithm unrolling**, [45] proposed LISTA, a neural algorithm that interprets ISTA as layers of neural networks. It has been demonstrated that LISTA outperforms ISTA thanks to its learnable components. Since then, designing neural algorithms by unrolling ISTA has become an active research topic. However, existing works mostly focus on the compressed sensing problem with a fixed design matrix only.

To enable for more general applicability, we design the architecture of PLISA by unrolling a classic path-following algorithm called **APF** [145] instead of ISTA. APF is applicable to nonconvex losses and nonconvex penalty functions, covering a considerably larger range of objectives than LASSO. Designing the architecture based on APF allows PLISA to be applicable to a broader class of problems such as nonlinear sparse regression, graphical model estimation, etc. Furthermore, employing nonconvexity can potentially lead to better statistical properties [158, 159, 160], for which we will explain more in Section 7.4.

In the following, we will introduce APF and the architecture of our proposed PLISA. After that, we will describe how to optimize the parameters in PLISA under the learning-to-learn setting.

7.3.1 A brief introduction to APF

We briefly introduce the classic algorithm APF [145], and its details are presented in Algorithm 3 in Appendix I in [144]. The key idea of path-following algorithms is creating **a sequence of T many sub-objectives** to gradually approach the target objective that is supposed to be more difficult to solve. More specifically, APF approximates the local minimizers of a sequence of sub-objectives:

$$\boldsymbol{\beta}_t \approx \widehat{\boldsymbol{\beta}}_{\lambda_t} \in \arg\min_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}) + P(\lambda_t, \boldsymbol{\beta}), \quad \text{for } t = 1, \cdots, T, \tag{7.2}$$

where $\lambda_1 > \lambda_2 > \cdots > \lambda_T$ is a decreasing sequence of regularization parameters. The last parameter λ_T is the target regularization parameter. As a result, APF contains T blocks, and each block contains an iterative algorithm that minimizes one sub-objective in Eq. 7.2. The output of the $(t-1)$-th block, denoted by $\boldsymbol{\beta}_{t-1}$, is used as the initialization of the t-th block, i.e., $\widetilde{\boldsymbol{\beta}}_t^0 = \boldsymbol{\beta}_{t-1}$. Then the t-th block minimizes the t-th sub-objective by the **modified proximal gradient** algorithm:

$$\text{for } k = 1, \cdots, K, \quad \widetilde{\boldsymbol{\beta}}_t^k \leftarrow \mathcal{T}_{\alpha\cdot\lambda_t}\left(\widetilde{\boldsymbol{\beta}}_t^{k-1} - \alpha\big(\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \widetilde{\boldsymbol{\beta}}_t^{k-1}) + \nabla_{\boldsymbol{\beta}} Q(\lambda_t, \widetilde{\boldsymbol{\beta}}_t^{k-1})\big)\right). \tag{7.3}$$

$$\text{output of } t\text{-th block:} \quad \boldsymbol{\beta}_t = \widetilde{\boldsymbol{\beta}}_t^K. \tag{7.4}$$

The notation $\mathcal{T}_\delta(\boldsymbol{\beta}) := \text{sign}(\beta)\max\{|\boldsymbol{\beta}| - \delta, 0\}$ is the soft-thresholding function, and the function Q is the concave component of P defined as $Q(\lambda, \boldsymbol{\beta}) := P(\lambda, \boldsymbol{\beta}) - \lambda\|\boldsymbol{\beta}\|_1$. The number of steps K in each block is determined by certain stopping criteria. It can be seem that the update steps in each block is similar to the ISTA algorithm, but it is modified in

order to incorporate nonconvexity.

7.3.2 Architecture of PLISA

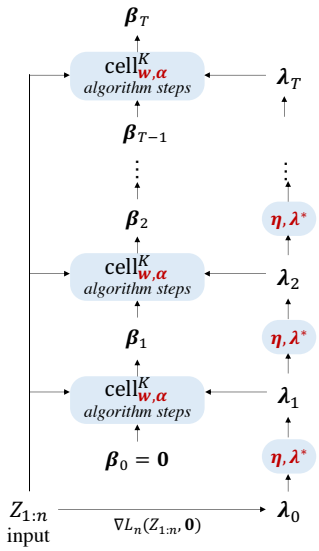


Figure 7.2: Architecture.

The architecture of PLISA is designed by unrolling the APF algorithm, and augmenting some learnable parameters θ. Therefore, the architecture of PLISA$_\theta$ contains T blocks and each block contains K layers defined by the K-step algorithm in Eq. 7.3 (See Figure 7.2). Note that in PLISA$_\theta$, both K and T are pre-defined. The architecture of PLISA$_\theta$ is **different** from APF as summarized below:

1. **Element-wise and learnable regularization parameters.** Most classic algorithms including APF employ a uniform regularization parameter λ_t across all entries of $\boldsymbol{\beta}$, but PLISA$_\theta$ uses a d-dimensional vector $\boldsymbol{\lambda}_t = [\lambda_{t,1}, \cdots, \lambda_{t,d}]^\top$ to enforce different levels of sparsity to different entries in $\boldsymbol{\beta}$. Furthermore, the regularization parameters in PLISA$_\theta$ are learnable, which will be optimized during the training.
2. **Learnable penalty function:** Classic algorithms use a pre-defined sparse penalty function P, but PLISA$_\theta$ parameterizes it as a combination of q different penalty functions and learns the weights of each penalty. In other words, PLISA$_\theta$ can learn to select a

specific combination of the penalty functions from training data.

3. **Learnable step size:** The step sizes in APF are selected by line-search but they are learnable in $\texttt{PLISA}_\theta$. Experimentally, we find the learned step sizes lead to a much faster algorithm.

In later sections of this chapter, we will show how such differences can make $\texttt{PLISA}_\theta$ perform better than APF both empirically and theoretically.

Algorithm 5 and 6 present the mathematical details of the architecture, follow which we explain some notations and definitions. Red-colored symbols indicate learnable parameters in $\texttt{PLISA}_\theta$.

Algorithm 5: $\texttt{PLISA}_\theta$ architecture

#blocks: T, *#layers per block:* K

Parameters: $\theta = \{\boldsymbol{\eta}, \boldsymbol{\lambda}^*, \boldsymbol{w}, \alpha\}$

Input: samples $Z_{1:n}$

$\boldsymbol{\beta}_0 \leftarrow \mathbf{0}, \quad \boldsymbol{\lambda}_0 \leftarrow \nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \mathbf{0})$

For $t = 1, \ldots, T$ **do**

$\quad \boldsymbol{\lambda}_t \leftarrow \max\{\sigma(\boldsymbol{\eta}) \circ \boldsymbol{\lambda}_{t-1}, \boldsymbol{\lambda}^*\}$

$\quad \boldsymbol{\beta}_t \leftarrow \texttt{Block}^K_{\boldsymbol{w},\alpha}(Z_{1:n}, \boldsymbol{\beta}_{t-1}, \boldsymbol{\lambda}_t)$

return $\boldsymbol{\beta}_T$

Algorithm 6: Layers in each block $\texttt{Block}^K_{\boldsymbol{w},\alpha}$

Input: $Z_{1:n}, \boldsymbol{\beta}_{t-1}, \boldsymbol{\lambda}_t$

$\widetilde{\boldsymbol{\beta}}_t^0 \leftarrow \boldsymbol{\beta}_{t-1}$

For $k = 1, \ldots, K$ **do**

$\quad \boldsymbol{g}_t^k \leftarrow \nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \widetilde{\boldsymbol{\beta}}_t^{k-1}) + \nabla_{\boldsymbol{\beta}} Q_{\boldsymbol{w}}(\boldsymbol{\lambda}_t, \widetilde{\boldsymbol{\beta}}_t^{k-1})$

$\quad \widetilde{\boldsymbol{\beta}}_t^k \leftarrow \mathcal{T}_{\alpha \cdot \boldsymbol{\lambda}_t}\left(\widetilde{\boldsymbol{\beta}}_t^{k-1} - \alpha \cdot \boldsymbol{g}_t^k\right)$

return $\boldsymbol{\beta}_t = \widetilde{\boldsymbol{\beta}}_t^K$

Regularzation parameters. In $\texttt{PLISA}_\theta$, the element-wise regularization parameters

are initialized by a vector $\boldsymbol{\lambda}_0 := \nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \mathbf{0})$, and then updated sequentially by

$$\boldsymbol{\lambda}_t \leftarrow \max\left\{\sigma(\boldsymbol{\eta}) \circ \boldsymbol{\lambda}_{t-1}, \boldsymbol{\lambda}^*\right\}, \tag{7.5}$$

where $\sigma(\cdot)$, $\circ$, and $\max\{\cdot,\cdot\}$ are element-wise sigmoid function, multiplication, and maximization. $\{\boldsymbol{\eta}, \boldsymbol{\lambda}^*\}$ are both d-dimensional learnable parameters. Eq. 7.5 creates a sequence $\boldsymbol{\lambda}_1, \cdots, \boldsymbol{\lambda}_T$ through the decrease ratio $\sigma(\boldsymbol{\eta})$, until they reach the target regularization parameters $\boldsymbol{\lambda}^*$.

Penalty function. $\texttt{PLISA}_\theta$ parameterizes the penalty function as follows,

$$P_{\boldsymbol{w}}(\boldsymbol{\lambda}, \boldsymbol{\beta}) = \textstyle\sum_{i=1}^q \widetilde{w}_i \cdot P^{(i)}(\boldsymbol{\lambda}, \boldsymbol{\beta}), \quad \text{where } \widetilde{w}_i = \frac{\exp(w_i)}{\sum_{i'=1}^q \exp(w_{i'})}. \tag{7.6}$$

In other words, $P_{\boldsymbol{w}}$ is a learnable convex combination of q penalty functions $(P^{(1)}, \cdots, P^{(q)})$. The weights of these functions are determined by learnable parameters $\boldsymbol{w} = [w_1, \cdots, w_q]$. In this chapter, we focus on learning the combination of three well-known penalty functions:

$$\begin{aligned} P^{(1)}(\boldsymbol{\lambda}, \boldsymbol{\beta}) &= \|\boldsymbol{\lambda} \circ \boldsymbol{\beta}\|_1, \\ P^{(2)}(\boldsymbol{\lambda}, \boldsymbol{\beta}) &= \sum_{j=1}^d \mathrm{MCP}(\lambda_j, \beta_j), \\ P^{(3)}(\boldsymbol{\lambda}, \boldsymbol{\beta}) &= \sum_{j=1}^d \mathrm{SCAD}(\lambda_j, \beta_j), \end{aligned}$$

where $P^{(1)}$ is convex, and MCP [161] and SCAD [158] are *nonconvex penalties* whose analytical forms are given in Appendix B in [144]. One can include any other penalty functions as long as they satisfy a set of conditions specified in Appendix B in [144]. $Q_{\boldsymbol{w}}(\boldsymbol{\lambda}, \boldsymbol{\beta}) := P_{\boldsymbol{w}}(\boldsymbol{\lambda}, \boldsymbol{\beta}) - \|\boldsymbol{\lambda} \circ \boldsymbol{\beta}\|_1$ represents the concave component of $P_{\boldsymbol{w}}$. The analytical form of $\nabla_{\boldsymbol{\beta}} Q_{\boldsymbol{w}}(\boldsymbol{\lambda}_t, \boldsymbol{\beta})$ are in Appendix B in [144].

7.3.3 Learning-to-learn setting

Now we describe how to train the parameters θ in $\texttt{PLISA}_\theta$ under the learning-to-learn setting.

Training set. Similar to other works in this domain, we assume the access to m problems from the target problem-space $\mathcal{P}$, and use them as the training set:

$$\mathcal{D}_m = \{(Z^{(1)}_{1:n_1}, \boldsymbol{\beta}^{*(1)}), \cdots, (Z^{(m)}_{1:n_m}, \boldsymbol{\beta}^{*(m)})\} \quad \text{with} \quad (Z^{(i)}_{1:n_i}, \boldsymbol{\beta}^{*(i)}) \in \mathcal{P}.$$

Here each estimation problem is represented by a pair of observations and the corresponding true parameter to be recovered. A different problem i can contain a different number n_i of observations.

Training loss. Since the intermediate outputs $\boldsymbol{\beta}_t(Z_{1:n}; \theta)$ of $\texttt{PLISA}_\theta$ are also estimates of $\boldsymbol{\beta}^*$, a common design of the training loss is the weighted sum of the intermediate estimation errors [58]. More specifically, we employ the following training loss:

$$\mathcal{L}^{\gamma}_{train}(\mathcal{D}_m; \theta) := \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T} \gamma^{T-t} \left\| \boldsymbol{\beta}_t(Z^{(i)}_{1:n_i}; \theta) - \boldsymbol{\beta}^{*(i)} \right\|_2^2, \tag{7.7}$$

where $\gamma < 1$ is a discounting factor If $\gamma = 0$ then the loss is only estimated at the last layer.

Generalization error. The ultimate goal of algorithm learning is to minimize the estimation error on expectation over all problems in the target problem distribution:

$$\mathcal{L}_{gen}(\mathbb{P}(\mathcal{P}); \theta) := \mathbb{E}_{(Z_{1:n}, \boldsymbol{\beta}^*) \sim \mathbb{P}(\mathcal{P})} \left\| \boldsymbol{\beta}_T(Z_{1:n}; \theta) - \boldsymbol{\beta}^* \right\|_2^2, \tag{7.8}$$

where $\mathbb{P}(\mathcal{P})$ is a distribution in the target problem-space $\mathcal{P}$. Let $\theta^* \in \arg\min \mathcal{L}^{\gamma=0}_{train}(\mathcal{D}_m; \theta)$ be a minimizer of the training loss. It is well-known that the generalization error can be

bounded by:

$$\mathcal{L}_{gen}(\mathbb{P}(\mathcal{P});\theta^*) \leq \underbrace{\mathcal{L}_{gen}(\mathbb{P}(\mathcal{P});\theta^*) - \mathcal{L}_{train}^{\gamma=0}(\mathcal{D}_m;\theta^*)}_{\text{generalization gap: Theorem 7.5.1}} + \underbrace{\mathcal{L}_{train}^{\gamma=0}(\mathcal{D}_m;\theta^*)}_{\text{training error: Theorem 7.4.1}} . \tag{7.9}$$

We will theoretically characterize these two terms in Theorem 7.4.1 and Theorem 7.5.1.

7.4 Capacity of `PLISA`

Can $\texttt{PLISA}_\theta$ achieve a small training error without using too many layers? How can designs of element-wise regularization and learnable penalty functions help $\texttt{PLISA}_\theta$ to achieve a smaller training error compared to classic algorithms? We answer this question theorectically in this section.

7.4.1 Problem space assumption

Before stating the theorem, we follow the notations in [145, 160] to describe some classic assumptions on the estimation problems.

Assumption 7.4.1 (Problem Space)**.** *Let $s^*, \tilde{s}$ be positive integers and ρ_-, ρ_+ be positive constants such that $\tilde{s} > (121(\rho_+/\rho_-) + 144(\rho_+/\rho_-)^2)s^*$. Assume for every estimation problem $(Z_{1:n}, \boldsymbol{\beta}^*)$ in the space $\mathcal{P}$, the following conditions are satisfied.*

(a) $\|\boldsymbol{\beta}^*\|_0 \leq s^*$ *and* $\|\boldsymbol{\beta}^*\|_\infty \leq B_1$*;*

(b) For any nonzero $\boldsymbol{v} \in \mathbb{R}^d$ *with sparsity* $\|\boldsymbol{v}\|_0 \leq s^* + 2\tilde{s}$*, it holds* $\frac{\boldsymbol{v}^\top \nabla^2_{\boldsymbol{\beta}} L_n(Z_{1:n},\boldsymbol{\beta})\boldsymbol{v}}{\|\boldsymbol{v}\|_2^2} \in [\rho_-, \rho_+]$*;*

(c) $8\,|[\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*)]_j| \leq |[\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \mathbf{0})]_j| \leq B_2, \forall j = 1, \cdots, d.$

Condition (a) assumes $\boldsymbol{\beta}^*$ is s^*-sparse and B_1-bounded. Condition (b) is commonly referred to as 'sparse eigenvalue condition' [162, 145], which is weaker than the well-known restricted isometry property (RIP) in compressed sensing [163]. Note that the class of functions satisfying conditions of this type is much larger than the class of convex losses.

In the special case when $L_n(Z_{1:n}, \boldsymbol{\beta})$ is strongly convex in $\boldsymbol{\beta}$, condition (b) holds with $\tilde{s} \to \infty$. The last condition bounds the gradient of the empirical loss L_n at the true parameter $\boldsymbol{\beta}^*$ and $\mathbf{0}$.

7.4.2 First main result: capacity

Let $\boldsymbol{\beta}_t(Z_{1:n}; \theta)$ be the output of the t-th block in $\texttt{PLISA}_\theta$. Let $\boldsymbol{x} \vee a$ denote entry-wise maximal value $\max\{\boldsymbol{x}, a\}$. Let $(\boldsymbol{x})_S$ denote the sub-vector of $\boldsymbol{x}$ with entries indexed by the set S.

Theorem 7.4.1 (Capacity)**.** *Assume the problem space $\mathcal{P}$ satisfies Assumption 7.4.1 and $\mathcal{D}_m \subseteq \mathcal{P}$. Let T be the number of blocks in $\texttt{PLISA}_\theta$ and let K be the number of layers in each block. For any $\varepsilon > 0$, there exists a set of parameters $\theta = \{\boldsymbol{\eta}, \boldsymbol{\lambda}^*, \boldsymbol{w}, \alpha\}$ such that the estimation error of every problem $(Z_{1:n}, \boldsymbol{\beta}^*) \in \mathcal{D}_m$ is bounded as follows, $\forall T > t_0$,*

$$\|\boldsymbol{\beta}_T(Z_{1:n}; \theta) - \boldsymbol{\beta}^*\|_2 \leq \varepsilon^{-1} c_\theta s^* \exp(-C_\theta K(T - t_0)) \qquad \textit{optimization error} \tag{7.10}$$

$$+ c'_\theta \kappa_m \| \left(\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*) \vee \varepsilon\right)_{S^*} \|_2, \qquad \textit{statistical error} \tag{7.11}$$

where $S^* := \mathrm{supp}(\boldsymbol{\beta}^*)$ is the support indices of $\boldsymbol{\beta}^*$, c_θ, c'_θ, and C_θ are some positive values depending on the chosen θ, and κ_m is a condition number which reveals the similarity of the problems in $\mathcal{D}_m$. Note that K and t_0 are required to be larger than certain values, but we will elaborate in Appendix E in [144] that the required lower bounds are small. See Appendix E in [144] for the proof of this theorem.

This estimation error can be interpreted as a combination of the optimization error (in Eq. 7.10) and the statistical error (in Eq. 7.11). The optimization error decreases linearly in both K and T. The statistical erroroccurs because of the randomness in $Z_{1:n}$. The gradient at the true parameter $\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*)$ characterizes how well the finite samples $Z_{1:n}$ can represent the distribution $\mathbb{P}_{\boldsymbol{\beta}^*}$.

A direct consequence of Theorem 7.4.1 is that the training error can be small without using too many layers and blocks in $\texttt{PLISA}_\theta$. We will also elaborate on how the entry-wise regularization and learnable penalty function can effectively reduce the training error in the following.

(i) Impact of entry-wise regularization. Restricting the regularization to be uniform across entries will lead to an error bound that replaces the statistical error $\| \left(\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*) \vee \varepsilon\right)_{S^*} \|_2$ in Eq. 7.11 by $\sqrt{s^*}\left(\|\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*)\|_\infty \vee \varepsilon\right)$. To understand how the former has improved the latter, we can consider the sparse linear regression problem. If the design matrix is normalized such that $\max_{1\le j\le d} \|([\boldsymbol{x}_1]_j, \cdots, [\boldsymbol{x}_n]_j)\|_2 \le \sqrt{n}$, then $\| \left(\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*) \vee \varepsilon\right)_{S^*} \|_2 \le C\sqrt{s^*/n}$ with high probability. In comparison, $\sqrt{s^*}\|\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*)\|_\infty \le C\sqrt{s^* \log d/n}$ with high probability is a slower statistical rate due to the term $\log d$.

(ii) Impact of learnable penalty function. To explain the the benefit of using learnable penalty function, we give a more refined bound for the statistical error in Eq. 7.11 in the following lemma.

Lemma 7.4.1 (Refined bound)**.** *Assume the same conditions and parameters θ in Theorem 7.4.1. Assume $T \to \infty$ so that the optimization error can be ignored. For simplicity, assume $\widetilde{w_3} = 0$ and only consider the weights $\widetilde{w_1}$ and $\widetilde{w_2}$ for ℓ_1 penalty and MCP. Then for every problem $(Z_{1:n}, \boldsymbol{\beta}^*) \in \mathcal{D}_m$:*

$$\|\boldsymbol{\beta}_\infty(Z_{1:n}; \theta) - \boldsymbol{\beta}^*\|_2 \le \tfrac{1+8(1+\widetilde{w_2})\kappa_m}{\rho_- - \widetilde{w_2}/b} \|(\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*) \vee \varepsilon)_{S_1^*}\|_2 \quad \left(S_1^*\text{: Small } \left|\beta_j^*\right|\text{'s}\right) \tag{7.12}$$

$$+ \tfrac{1+8(1-\widetilde{w_2})\kappa_m}{\rho_- - \widetilde{w_2}/b} \|(\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*) \vee \varepsilon)_{S_2^*}\|_2 \quad \left(S_2^*\text{: Large } \left|\beta_j^*\right|\text{'s}\right), \tag{7.13}$$

where $b > 1$ is a hyperparameter in MCP*, and the index sets S_1^* and S_2^* are defined as $S_1^* := \{j \in S^* : \left|\beta_j^*\right| \le b\lambda_j^*\}$ and $S_2^* := \{j \in S^* : \left|\beta_j^*\right| > b\lambda_j^*\}$. See Appendix E in [144] for the proof.*

This refined bound reveals the benefit of learning the penalty function because:

1. According to Lemma 7.4.1, the optimal penalty function is *problem-dependent*. For example, if $(8(b\rho_- +1)\kappa_m +1)\|(\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*) \vee \varepsilon)_{S_1^*}\|_2 > (8(b\rho_- -1)\kappa_m -1)\|(\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*) \vee \varepsilon)_{S_2^*}\|_2$, choosing $\widetilde{w_2} = 0$ can induce a smaller error bound. Otherwise, $\widetilde{w_2} = 1$ is better. Therefore, learning is a more suitable way of choosing the penalty function.
2. The convergence speed C_θ in Eq. 7.10 is also affected by the weights, monotonely decreasing in $\widetilde{w_2}$. Through gradient-based training, we can automatically find the optimal combination of penalty functions to strike a nice balance between the statistical error and convergence speed.

7.5 Generalization Analysis

How well can the learned $\texttt{PLISA}_\theta$ solve new problems outside the training set? In this section, we conduct the generalization analysis in a novel way to focus on answering the questions:

How is the *generalization bound* of $\texttt{PLISA}_\theta$ related to its *algorithmic properties*?

And how is it *different from* conventional neural networks?

7.5.1 Second main result: generalization bound

To analyze the generalization properties of neural networks, many works have adopted the analysis framework of [98] to bound the Rademacher complexity via Dudley's integral [8, 76, 77, 155]. A key step in this analysis framework is deriving the robustness of the training loss to the small perturbation in model parameters θ. Since we can view $\texttt{PLISA}_\theta$ as an iterative algorithm, we borrow the analysis tools of classic optimization algorithms to derive its robustness in θ. The following lemma states this key intermediate result, which connects the Lipschitz constant to algorithmic properties of $\texttt{PLISA}_\theta$.

Lemma 7.5.1 (Robustness to θ)**.** *Assume $\mathcal{P}$ satisfies Assumption 7.4.1 and $\mathcal{D}_m \sim \mathbb{P}(\mathcal{P})^m$. Assume $\texttt{PLISA}_\theta$ contains $T > t_0$ blocks and K layers. Consider a parameter space Θ in which the parameters satisfy (i) $\alpha \in [\alpha_{\min}, \frac{1}{\rho_+}]$, (ii) $\eta_j \in [\sigma^{-1}(0.9), \eta_{\max}]$, (iii) $\widetilde{w_2}\frac{1}{b} + \widetilde{w_3}\frac{1}{a-1} \le \xi_{\max} < \rho_-$, and (iv) $\lambda_j^* \in [8\sup_{(Z_{1:n},\boldsymbol{\beta}^*)\in\mathcal{D}_m} |[\nabla_{\boldsymbol{\beta}} L_n(Z_{1:n}, \boldsymbol{\beta}^*)]_j| \vee \varepsilon, \lambda_{\max}]$ with some positive constants $\alpha_{\min}$, $\eta_{\max}$, $\xi_{\max}$, and $\lambda_{\max}$. Then for any $\theta = \{\boldsymbol{\eta}, \boldsymbol{\lambda}^*, \boldsymbol{w}, \alpha\}$ and $\theta' = \{\boldsymbol{\eta}', \boldsymbol{\lambda}^{*\prime}, \boldsymbol{w}', \alpha'\}$ in Θ, and for any recovery problem $(Z_{1:n}, \boldsymbol{\beta}^*) \in \mathcal{D}_m$, the following inequality holds,*

$$\|\boldsymbol{\beta}_T(Z_{1:n};\theta) - \boldsymbol{\beta}_T(Z_{1:n};\theta')\|_2 \le c_1 K(T-t_0)\sqrt{s^*}\,|\alpha-\alpha'|\underbrace{\exp(-C_\Theta K(T-t_0))}_{\text{convergence rate}} \tag{7.14}$$

$$+\Big(c_2\|\boldsymbol{\eta}-\boldsymbol{\eta}'\|_2 + c_3\|\boldsymbol{\lambda}^*-\boldsymbol{\lambda}^{*\prime}\|_2 + c_4\sqrt{d}\|\boldsymbol{w}-\boldsymbol{w}'\|_2\Big)\underbrace{(1-\exp(-C_\Theta K T))}_{\text{stability rate}}, \tag{7.15}$$

where c_1, c_2, c_3, c_4 and C_Θ are some positive constants. Note that similar to Theorem 7.4.1, K and t_0 are required to be larger than certain small values. See Appendix F.1 in [144] for the proof.

Convergence rate & step size perturbation. In Eq. 7.14, the Lipschitz constant in the step size α scales at the same rate as the convergence rate of $\texttt{PLISA}_\theta$, decreasing exponentially in T and K (See Fig. 7.3 for a visualization). To understand this, consider when both step sizes α and α' are within the convergence region (i.e., $(0, \rho_+^{-1}]$). After infinitely many steps, their induced outputs will both converge to the same optimal point. This intuitively explains why the output perturbation caused by α-perturbation has the same decrease rate as the optimization error.

Stability rate & regularization perturbation. In the literature of optimization, stability of an algorithm expresses its robustness to small perturbation in the optimization objective. This is clearly related to the robustness of $\texttt{PLISA}_\theta$ to the perturbation in $\boldsymbol{\eta}, \boldsymbol{\lambda}^*, \boldsymbol{w}$, because these parameters jointly determine the regularization $P_{\boldsymbol{w}}(\boldsymbol{\lambda}_t, \boldsymbol{\beta})$, which is a part

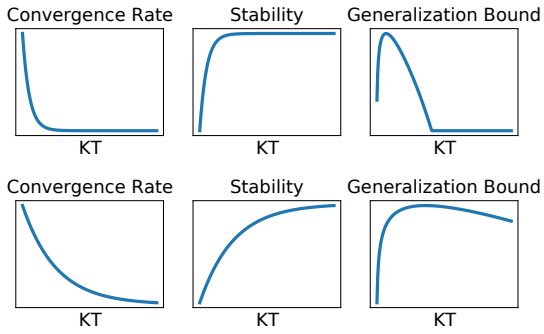


Figure 7.3: Visualization of convergence, stability, and generalization bound in Theorem 7.5.1. The two sets of visualizations are obtained by choosing different speeds C_Θ in the convergence rate and stability.

of the optimization objective. Therefore, we exploit the analysis techniques for algorithmic stability to derive the robustness in $(\boldsymbol{\eta}, \boldsymbol{\lambda}^*, \boldsymbol{w})$-perturbation and obtain the Lipschitz constant in Eq. 7.15, which is bounded but increasing in T and K (See Fig. 7.3 for a visualization).

Based on the key result in Lemma 7.5.1, we can apply Dudley's integral to measure the empirical Rademachar complexity which immediately yields the following generalization bound.

Theorem 7.5.1 (Generalization gap)**.** *Assume the assumptions in Lemma 7.5.1. For any $\epsilon > 0$, with probability at least $1 - \epsilon$, the generalization gap is bounded by*

$$\mathcal{L}_{gen}(\mathbb{P}(\mathcal{P}); \theta) - \mathcal{L}^{\gamma=0}_{train}(\mathcal{D}_m; \theta) \leq c_1 \sqrt{m^{-1} \log(4\epsilon^{-1})} + \tag{7.16}$$

$$\sqrt{c_2 m^{-1} \log\left(\sqrt{m} K T \underbrace{\exp(-C_\Theta K (T - t_0))}_{\text{convergence rate}} \vee 1\right) + c_3 d m^{-1} \log\left(\sqrt{m} \underbrace{(1 - \exp(-C_\Theta K T))}_{\text{stability}}\right)},$$

where c_1, c_2, c_3, C_Θ are constants independent of d, m, K and T.

Fig. 7.3 visualizes how the generalization bound in Theorem 7.5.1 grows when KT increases. The two sets of plots look slightly different by picking different constants C_Θ. We have also tried varying the values of c_2, c_3, d, m in Theorem 7.5.1. Overall, they lead to the two types of behaviors in Fig. 7.3.

An important observation in Theorem 7.5.1 and Figure 7.3 is that the generalization gap could *decrease in the number of layers*, and we will see in Section 7.7 that this matches the empirical observations. It also distinguishes algorithm-unrolling based architectures from conventional neural networks, whose generalization gaps rarely decrease in the number of layers.

Remark. The above generalization results are conducted on a constrained parameter space (as described in Lemma 7.5.1) so that we can utilize the algorithmic properties of $\texttt{PLISA}_\theta$. We focus on this space because the analysis contains more interesting and new ingredients. For parameters outside this space, the analysis procedure is similar to other conventional recurrent networks. Since the bound in Theorem 7.5.1 has matched the empirical observations, it is reasonable to believe that after training, the learned parameters are likely to be in this 'nice' constrained space.

7.6 Extension To Unsupervised Learning-to-learn Setting

Real-world datasets may not contain the ground-truth parameters $\boldsymbol{\beta}^*$, but only contain the samples from each task, $\mathcal{D}_m^U = \{Z_{1:n}^{(1)}, \cdots, Z_{1:n}^{(m)}\}$. In this setting, we can construct an unsupervised training loss to minimize the empirical loss function L_n (e.g., the likelihood function) on the samples.

$$\textbf{Unsupervised training loss:} \quad \mathcal{L}_{train}^U(\mathcal{D}_m^U;\theta) := \frac{1}{m}\sum_{i=1}^{m} L_{n_2}\big(Z_{1:n_2}^{(i)}, \boldsymbol{\beta}_T(Z_{1:n_1}^{(i)};\theta)\big) \quad (7.17)$$

In this loss, both $Z_{1:n_1}^{(i)}$ and $Z_{1:n_2}^{(i)}$ are subsets of $Z_{1:n}^{(i)}$. The samples $Z_{1:n_1}^{(i)}$ are used as the input to $\texttt{PLISA}_\theta$ and the samples $Z_{1:n_2}^{(i)}$ are used for evaluating the output $\boldsymbol{\beta}_T$ from $\texttt{PLISA}_\theta$.

Let $\theta_U^* \in \arg\min \mathcal{L}_{train}^U(\mathcal{D}_m^U;\theta)$ be a minimizer to this unsupervised loss. **Theoreti-**

cally, to bound the generalization error of θ_U^*, we can show that

$$\mathcal{L}_{gen}(\mathbb{P}(\mathcal{P});\theta_U^*) \leq \underbrace{\frac{C}{m}\sum_{i=1}^{m}\left(L_{n_2}\left(Z_{1:n_2}^{(i)}, \boldsymbol{\beta}_T(Z_{1:n_1}^{(i)};\theta_U^*)\right) - L_{n_2}\left(Z_{1:n_2}^{(i)}, \boldsymbol{\beta}^{*(i)}\right)\right)}_{\text{unsupervised training error}} \tag{7.18}$$

$$+ \underbrace{\mathcal{L}_{gen}(\mathbb{P}(\mathcal{P});\theta_U^*) - \mathcal{L}_{train}^{\gamma=0}(\mathcal{D}_m;\theta_U^*)}_{\text{generalization gap: Theorem 7.5.1}} + \underbrace{\frac{C}{m}\sum_{i=1}^{m}\|\nabla_{\boldsymbol{\beta}} L_{n_2}(Z_{1:n_2}^{(i)}, \boldsymbol{\beta}^{*(i)})\|_2^2}_{\text{statistical error}}. \tag{7.19}$$

Compared to Eq. 7.9, this upper bound contains an additional statistical error, which appears because of the gap between the unsupervised loss and the true error that we aim to optimize. Clearly, in this upper bound, the generalization gap can be bounded by Theorem 7.5.1. Furthermore, the unsupervised training error in Eq. 7.18 can be bounded by combining Theorem 7.4.1 with the following in equality.

$$L_{n_2}\left(Z_{1:n_2}, \boldsymbol{\beta}_T(Z_{1:n_1};\theta_U^*)\right) - L_{n_2}\left(Z_{1:n_2}, \boldsymbol{\beta}^*\right) \tag{7.20}$$

$$\leq L_{n_2}\left(Z_{1:n_2}, \boldsymbol{\beta}_T(Z_{1:n_1};\theta^*)\right) - L_{n_2}\left(Z_{1:n_2}, \boldsymbol{\beta}^*\right)$$

$$\leq \|\nabla_{\boldsymbol{\beta}} L_{n_2}(Z_{1:n_2}, \boldsymbol{\beta}^*)\|_2 \underbrace{\|\boldsymbol{\beta}_T(Z_{1:n_1};\theta^*) - \boldsymbol{\beta}^*\|_2}_{\text{bounded by Theorem 7.4.1}} + \frac{\rho_+}{2}\underbrace{\|\boldsymbol{\beta}_T(Z_{1:n_1};\theta^*) - \boldsymbol{\beta}^*\|_2^2}_{\text{bounded by Theorem 7.4.1}}. \tag{7.21}$$

7.7 Experiments

7.7.1 Synthetic experiments

In synthetic datasets, we consider sparse linear regression problems and sparse precision matrix estimation problems for Gaussian graphical models. Specifically, we recover target vectors $\boldsymbol{\beta}^* \in \mathbb{R}^d$, where $d = \{256, 1024\}$ in SLR, and estimate precision matrices $\Theta^* \in \mathbb{R}^{d\times d}$, where $d = \{50, 100\}$ in SPE.

Sparse Linear Regression (SLR)

In this experiment, we compare `PLISA` with several baselines and also verify the theorems.

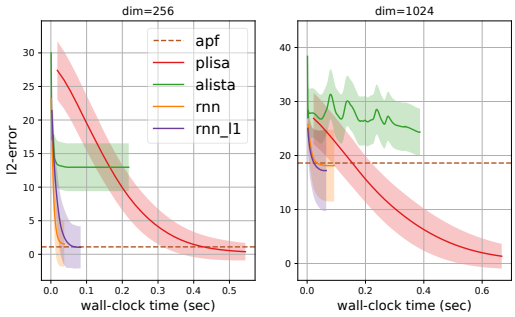


Figure 7.4: Convergence of recovery error. Since APF takes a long time to converge, its curve are outside the range of these plots. We use a dash-line to represent the final ℓ_2 error it achieves.

Performance comparison. We consider baselines including APF [145], ALISTA [49], RNN [38], and RNN-ℓ_1. APF is the path-following algorithm that is used as the basis of our architecture. ALISTA is a representative of algorithm unrolling based architectures, which is an advanced variant of LISTA. We have tried the vanilla LISTA, but it performs worse than ALISTA on our tasks so it is not reported. RNN refers to the LSTM-based model in [38]. Besides, we add a soft-thresholding operator to this model to enforce sparsity, and include this variant as a baseline, called RNN-ℓ_1. Except for APF, all methods are trained on the same set of training problems and selected by the validation problems. For APF, we perform grid-search to choose its hyperparameters, which is also selected by the validation set.

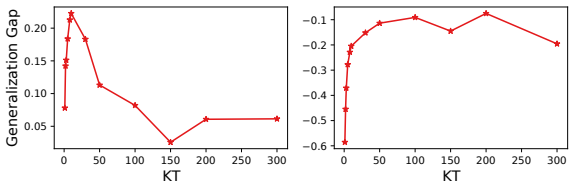


Figure 7.5: Generalization gap of PLISA with varying KT, for two different experimental settings.

Fig. 7.4 shows the convergence of $\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2$ for problems in the test set. The x-axis indicates the wall-clock time. In terms of the final recovery accuracy, PLISA outperforms all baseline methods. In the more difficult setting (i.e, $d = 1024$), its advantage is obvious. Although PLISA is slightly slower than other deep learning based models due to the com-

putations of MCP and SCAD, PLISA achieves a better accuracy and it has been converging much faster than the classic algorithm APF. APF is very slow mainly due to the use of line-search for selecting step sizes.

Generalization gap. We are interested in the generalization behavior of PLISA. As this experiment is conducted for theoretical interest, we do not use the validation set to select the model. We vary the number of layers (K) and blocks (T) in PLISA to create a set of models with different depths. For each depth, we train the model with 2000 training problems, and then test it on a separate set of 100 problems to approximate the generalization gap. In Fig. 7.5, we observe the interesting behavior of the generalization gap, where the left one increases in KT at the beginning and then decrease to a constant, and the right one increases fast and then decrease very slowly. This surprisingly matches the two different visualizations in Fig. 7.3 of the predicted generalization gap given by Theorem 7.5.1.

Sparse Precision Matrix Estimation (SPE)

We compare PLISA with APF, GLASSO [137], GISTA [164], and GGM [165] on sparse precision estimation tasks in Gaussian graphical models. GLASSO estimates the precision matrix by block-coordinate decent methods. GISTA is a proximal gradient method for precision matrix estimation. GGM utilizes convolutional neural network to estimate the precision matrix.

Table 7.1: Recovery error in SPE. The reported time is the average wall-clock time for solving each instance in seconds.

Sizes	$p = 50$		$p = 100$	
Methods	$\|\Theta - \Theta^*\|_F^2$	Time	$\|\Theta - \Theta^*\|_F^2$	Time
PLISA	119.47 ± 12.23	0.117	142.70 ± 13.38	0.132
GLASSO	169.63 ± 17.99	1.66	237.95 ± 27.49	3.12
GISTA	186.96 ± 25.48	53.47	373.66 ± 41.72	36.02
APF	269.51 ± 32.28	46.02	485.94 ± 60.33	86.82
GGM	194.26 ± 10.73	0.007	445.00 ± 58.89	0.008

Table 7.1 reports the Frobenius error $\|\Theta - \Theta^*\|_F^2$ between the estimation Θ and the

true precision matrix Θ^*, averaged over 100 test problems. PLISA achieves consistent improvements. Classic algorithm are slower because they perform line-search. GLASSO is faster than other classic algorithm because we use the sklearn package [166] in which the implementations are optimized.

Discussion on Training-Testing Time

Test time. Fig. 7.4 and Table 7.1 shows the wall-clock time for solving test problems. Overall, classic algorithms are slower because they need to perform line-search. It is noteworthy that learning-based methods can solve a batch of problems parallelly but most classic algorithms cannot. To allow more advantages for classic algorithms, the test problems are solved without batching in all methods.

Train time. As metioned earlier, we perform grid-search to select the hyperparameters in classic algorithms using validation sets. The training time comparison is summarized in Table 7.2 and Table 7.3. We can see that training time is not a bottleneck of this problem. Moreover, In SPE, classic algorithms even require a longer training time than learning-based methods.

Table 7.2: Training time for SLR (minutes)

Sizes	$d = 256$	$d = 1024$
PLISA	393	462
ALISTA	176	271
RNN	96	99
RNN_{ℓ_1}	101	106
APF	214	426

Table 7.3: Training time for SPE (minutes)

Sizes	$d = 50$	$d = 100$
PLISA	35	39
GGM	14	43
GISTA	176	116
APF	316	331
GLASSO	42	57

7.7.2 Unsupervised learning on real-world datasets

We conduct experiments of unsupervised algorithm learning on 3 datasets: **Gene** expression dataset [167], **Parkinsons** patient dataset [168], and **School** exam score dataset [169].

The goal of the algorithm is to estimation the sparse linear regression parameters $\boldsymbol{\beta}^*$ for

Table 7.4: Recovery accuracy on real-world datasets.

	PLISA	APF	RNN	RNN-ℓ_1	ALISTA
Gene	1.177	1.289	1.639	1.349	1.289
Parkinsons	11.63	11.86	11.91	13.05	34.843
School	296.6	367.9	561.5	310.3	884.2

each problem. Recovery accuracy is estimated by the least-square error on a set of held-out samples for each problem. Table 7.4 shows the effectiveness of PLISA. Note that these real-world datasets may not satisfy the assumptions in this chapter. This set of experiments are conducted only to demonstrate the robustness of the proposed method.

CHAPTER 8
CONCLUSION

In this thesis, we presented a novel duality view between deep learning models and algorithms. This duality view allows us to convert a neural network to an algorithm and convert an algorithm to the layers of a neural network. Furthermore, we have established connections between the learning properties of neural networks and the algorithmic properties of optimization algorithms. This duality view of their properties allow us to integrate the analysis tools of both domains to better understand the proposed method theoretically.

Based on the duality view, this thesis has presented multiple approaches to either use deep learning methods to help design algorithms or use algorithms to help design deep architectures.

Chapter 3 has presented a hybrid deep learning model, E2Efold, for RNA secondary structure prediction. This hybrid model contains algorithm layers which can incorporate hard constraints into the architecture. Experiments have demonstrated the benefits of this hybrid architecture. In the future, we believe the idea of unrolling constrained optimization solver can be useful for other structured prediction problems.

Chapter 4 has presented the theoretical analysis for the representation and generalization ability of hybrid deep learning models with algorithm layers. The theorem in this chapter reveals an intriguing relation between algorithmic properties of the algorithm layers and the approximation and generalization of the overall hybrid model. The current analysis is limited due to the simplified problem setting. Future efforts could be devoted to generalizing the results to more complex instances with fewer assumptions.

Chaper 5 has introduced a dynamic deep learning model with input-specific depth, which was inspired by the stopping criteria in classic algorithms. Experiments have demonstrated the effectiveness of this model in a wide range of applications. In the future, it will

be interesting to see whether other aspects of algorithms can be incorporated into deep learning models to further boost performance.

Chapter 6 has proposed to learn a neural algorithm, called GLAD, for solving the precision matrix estimation problem. The architecture of GLAD is designed based on unrolling an optimization algorithm. This chapter has empirically demonstrated that the learned GLAD outperforms classic algorithms in terms of sample complexity. Future efforts could be directed on the development of theory in order to fully understand this neural algorithm.

Chapter 7 has proposed PLISA for learning to solve sparse parameter recovery problems. It has also presented the theoretical guarantees for the capacity and generalization ability of PLISA. The analysis techniques could be used to derive guarantees for other algorithm-unrolling based architectures. In future work, the model PLISA could potentially be improved by using a more flexible penalty function, and the analysis for unsupervised settings could be improved by more careful derivations.

In summary, this thesis takes an initial step toward the empirical and theoretical exploration of the combination of deep learning and algorithm design. Many interesting research questions remain to be explored as mentioned above.

REFERENCES

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[2] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli, "Exponential expressivity in deep neural networks through transient chaos," *Advances in neural information processing systems*, vol. 29, 2016.

[3] Q. Wang *et al.*, "Exponential convergence of the deep neural network approximation for analytic functions," *arXiv preprint arXiv:1807.00297*, 2018.

[4] H. Xu and S. Mannor, "Robustness and generalization," *Machine learning*, vol. 86, no. 3, pp. 391–423, 2012.

[5] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro, "Exploring generalization in deep learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 5947–5956.

[6] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016.

[7] K. Kawaguchi, L. P. Kaelbling, and Y. Bengio, "Generalization in deep learning," *arXiv preprint arXiv:1710.05468*, 2017.

[8] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky, "Spectrally-normalized margin bounds for neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 6240–6249.

[9] B. Neyshabur, S. Bhojanapalli, and N. Srebro, "A PAC-bayesian approach to spectrally-normalized margin bounds for neural networks," in *International Conference on Learning Representations*, 2018.

[10] A. Leman and B. Weisfeiler, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsiya*, vol. 2, no. 9, pp. 12–16, 1968.

[11] K. Goyal, G. Neubig, C. Dyer, and T. Berg-Kirkpatrick, "A continuous relaxation of beam search for end-to-end training of neural sequence models," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[12] A. Mensch and M. Blondel, "Differentiable dynamic programming for structured prediction and attention," in *35th International Conference on Machine Learning*, vol. 80, 2018.

[13] H. Peng, S. Thomson, and N. A. Smith, "Backpropagating through structured argmax using a spigot," *arXiv preprint arXiv:1805.04658*, 2018.

[14] P.-W. Wang, P. Donti, B. Wilder, and Z. Kolter, "Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver," in *International Conference on Machine Learning*, 2019, pp. 6545–6554.

[15] W. Wang, Z. Dang, Y. Hu, P. Fua, and M. Salzmann, "Backpropagation-friendly eigendecomposition," in *Advances in Neural Information Processing Systems*, 2019, pp. 3156–3164.

[16] A. Zanfir and C. Sminchisescu, "Deep learning of graph matching," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2684–2693.

[17] M. Cuturi, O. Teboul, and J.-P. Vert, "Differentiable sorting using optimal transport: The sinkhorn cdf and quantile operator," *arXiv preprint arXiv:1905.11885*, 2019.

[18] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," *arXiv preprint arXiv:1806.04910*, 2018.

[19] A. Shaban, C.-A. Cheng, N. Hatch, and B. Boots, "Truncated back-propagation for bilevel optimization," in *The 22nd International Conference on Artificial Intelligence and Statistics*, PMLR, 2019, pp. 1723–1732.

[20] V. Niculae, A. Martins, M. Blondel, and C. Cardie, "Sparsemap: Differentiable sparse structured inference," in *International Conference on Machine Learning*, 2018, pp. 3799–3808.

[21] M. V. Pogančić, A. Paulus, V. Musil, G. Martius, and M. Rolinek, "Differentiation of blackbox combinatorial solvers," in *International Conference on Learning Representations*, 2019.

[22] Q. Berthet, M. Blondel, O. Teboul, M. Cuturi, J.-P. Vert, and F. Bach, "Learning with differentiable perturbed optimizers," *arXiv preprint arXiv:2002.08676*, 2020.

[23] A. Ferber, B. Wilder, B. Dilkina, and M. Tambe, "Mipaal: Mixed integer program as a layer.," in *AAAI*, 2020, pp. 1504–1511.

[24] D. Belanger, B. Yang, and A. McCallum, "End-to-end learning for structured prediction energy networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 429–439.

[25] J. Ingraham, A. Riesselman, C. Sander, and D. Marks, "Learning protein structure with a differentiable simulator," in *International Conference on Learning Representations*, 2019.

[26] X. Chen, Y. Li, R. Umarov, X. Gao, and L. Song, "Rna secondary structure prediction by learning unrolled algorithms," *arXiv preprint arXiv:2002.05810*, 2020.

[27] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 136–145.

[28] M. Rolinek, P. Swoboda, D. Zietlow, A. Paulus, V. Musil, and G. Martius, "Deep graph matching via blackbox differentiation of combinatorial solvers," in *European Conference on Computer Vision*, Springer, 2020, pp. 407–424.

[29] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, *Unrolled generative adversarial networks*, 2016. arXiv: 1611.02163 [cs.LG].

[30] J. Song, R. Lanka, A. Zhao, Y. Yue, and M. Ono, "Learning to search via retrospective imitation," *arXiv preprint arXiv:1804.00846*, 2018.

[31] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 554–15 566.

[32] Y. Tang, S. Agrawal, and Y. Faenza, "Reinforcement learning for integer programming: Learning to cut," *arXiv preprint arXiv:1906.04859*, 2019.

[33] C.-Y. Hsu, P. Indyk, D. Katabi, and A. Vakilian, "Learning-based frequency estimation algorithms," 2018.

[34] M. Purohit, Z. Svitkina, and R. Kumar, "Improving online algorithms via ml predictions," in *Advances in Neural Information Processing Systems*, 2018, pp. 9661–9670.

[35] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, ACM, 2019, pp. 270–288.

[36] H. Kim, Y. Jiang, S. Kannan, S. Oh, and P. Viswanath, "Deepcode: Feedback codes via deep learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 9436–9446.

[37] S. Gollapudi and D. Panigrahi, "Online algorithms for rent-or-buy with expert advice," in *International Conference on Machine Learning*, 2019, pp. 2319–2327.

[38] M. Andrychowicz *et al.*, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, 2016, pp. 3981–3989.

[39] Y. Yang, J. Sun, H. Li, and Z. Xu, "Admm-net: A deep learning approach for compressive sensing mri. corr," *arXiv preprint arXiv:1705.06869*, 2017.

[40] M. Borgerding, P. Schniter, and S. Rangan, "Amp-inspired deep networks for sparse linear inverse problems," *IEEE Transactions on Signal Processing*, vol. 65, no. 16, pp. 4293–4308, 2017.

[41] M.-C. Corbineau, C. Bertocchi, E. Chouzenoux, M. Prato, and J.-C. Pesquet, "Learned image deblurring by unfolding a proximal interior point algorithm," in *2019 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2019, pp. 4664–4668.

[42] X. Xie, J. Wu, G. Liu, Z. Zhong, and Z. Lin, "Differentiable linearized admm," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6902–6911.

[43] H. Shrivastava *et al.*, "GLAD: Learning sparse graph recovery," in *International Conference on Learning Representations*, 2020.

[44] K. Wei, A. Aviles-Rivero, J. Liang, Y. Fu, C.-B. Schönlieb, and H. Huang, "Tuning-free plug-and-play proximal algorithm for inverse imaging problems," in *International Conference on Machine Learning*, PMLR, 2020, pp. 10 158–10 169.

[45] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Omnipress, 2010, pp. 399–406.

[46] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, vol. 57, no. 11, pp. 1413–1457, 2004.

[47] U. S. Kamilov and H. Mansour, "Learning optimal nonlinearities for iterative thresholding algorithms," *IEEE Signal Processing Letters*, vol. 23, no. 5, pp. 747–751, 2016.

[48] X. Chen, J. Liu, Z. Wang, and W. Yin, "Theoretical linear convergence of unfolded ista and its practical weights and thresholds," in *Advances in Neural Information Processing Systems*, 2018, pp. 9061–9071.

[49] J. Liu, X. Chen, Z. Wang, and W. Yin, "ALISTA: Analytic weights are as good as learned weights in LISTA," in *International Conference on Learning Representations*, 2019.

[50] K. Wu, Y. Guo, Z. Li, and C. Zhang, "Sparse coding with gated learned ista," in *International Conference on Learning Representations*, 2020.

[51] D. Kim and D. Park, "Element-wise adaptive thresholds for learned iterative shrinkage thresholding algorithms," *IEEE Access*, vol. 8, pp. 45 874–45 886, 2020.

[52] J. Ma and L. Ping, "Orthogonal amp," *IEEE Access*, vol. 5, pp. 2020–2033, 2017.

[53] D. Liu, K. Sun, Z. Wang, R. Liu, and Z.-J. Zha, "Frank-wolfe network: An interpretable deep structure for non-sparse coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 9, pp. 3068–3080, 2019.

[54] R. Pauwels, E. Tsiligianni, and N. Deligiannis, "Hcgm-net: A deep unfolding network for financial index tracking," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2021, pp. 3910–3914.

[55] X. Zhang, Y. Lu, J. Liu, and B. Dong, "Dynamically unfolding recurrent restorer: A moving endpoint control method for image restoration," in *International Conference on Learning Representations*, 2019.

[56] X. Chen, H. Dai, and L. Song, "Particle flow bayes' rule," in *International Conference on Machine Learning*, 2019, pp. 1022–1031.

[57] N. Shlezinger, J. Whang, Y. C. Eldar, and A. G. Dimakis, "Model-based deep learning," *arXiv preprint arXiv:2012.08405*, 2020.

[58] T. Chen *et al.*, "Learning to optimize: A primer and a benchmark," *arXiv preprint arXiv:2103.12828*, 2021.

[59] J. Zhang and B. Ghanem, "Ista-net: Interpretable optimization-inspired deep network for image compressive sensing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1828–1837.

[60] F. Crick, "Central dogma of molecular biology," *Nature*, vol. 227, no. 5258, p. 561, 1970.

[61] S. Bellaousov, J. S. Reuter, M. G. Seetin, and D. H. Mathews, "RNAstructure: Web servers for RNA secondary structure prediction and analysis," *Nucleic acids research*, vol. 41, no. W1, W471–W474, 2013.

[62] N. Markham and M. Zuker, *Unafold: Software for nucleic acid folding and hybridization in: Keith jm, editor.(ed.) bioinformatics methods in molecular biology, vol. 453*, 2008.

[63] C. B. Do, D. A. Woods, and S. Batzoglou, "Contrafold: RNA secondary structure prediction without physics-based models," *Bioinformatics*, vol. 22, no. 14, e90–e98, 2006.

[64] D. H. Mathews and D. H. Turner, "Prediction of RNA secondary structure by free energy minimization," *Current opinion in structural biology*, vol. 16, no. 3, pp. 270–278, 2006.

[65] C. E. Hajdin, S. Bellaousov, W. Huggins, C. W. Leonard, D. H. Mathews, and K. M. Weeks, "Accurate shape-directed RNA secondary structure modeling, including pseudoknots," *Proceedings of the National Academy of Sciences*, vol. 110, no. 14, pp. 5498–5503, 2013.

[66] D. W. Staple and S. E. Butcher, "Pseudoknots: RNA structures with diverse functions," *PLoS biology*, vol. 3, no. 6, e213, 2005.

[67] P. Fechter, J. Rudinger-Thirion, C. Florentz, and R. Giege, "Novel features in the tRNA-like world of plant viral RNAs," *Cellular and Molecular Life Sciences CMLS*, vol. 58, no. 11, pp. 1547–1561, 2001.

[68] E. W. Steeg, "Neural networks, adaptive optimization, and RNA secondary structure prediction," *Artificial intelligence and molecular biology*, pp. 121–160, 1993.

[69] S. Zakov, Y. Goldberg, M. Elhadad, and M. Ziv-Ukelson, "Rich parameterization improves RNA structure prediction," *Journal of Computational Biology*, vol. 18, no. 11, pp. 1525–1542, 2011.

[70] H. Zhang *et al.*, "A new method of RNA secondary structure prediction based on convolutional neural network and dynamic programming," *Frontiers in genetics*, vol. 10, 2019.

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

[72] V. K. Pillutla, V. Roulet, S. M. Kakade, and Z. Harchaoui, "A smoother way to train structured prediction models," in *Advances in Neural Information Processing Systems*, 2018, pp. 4766–4778.

[73] A. Vaswani *et al.*, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[74] S. Wang, S. Sun, Z. Li, R. Zhang, and J. Xu, "Accurate de novo prediction of protein contact map by ultra-deep learning model," *PLoS computational biology*, vol. 13, no. 1, e1005324, 2017.

[75] X. Chen, Y. Zhang, C. Reisinger, and L. Song, "Understanding deep architecture with reasoning layer," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[76] M. Chen, X. Li, and T. Zhao, "On generalization bounds of a family of recurrent neural networks," *arXiv preprint arXiv:1910.12947*, 2019.

[77] V. K. Garg, S. Jegelka, and T. Jaakkola, "Generalization and representational limits of graph neural networks," *arXiv preprint arXiv:2002.06157*, 2020.

[78] P. L. Bartlett, O. Bousquet, S. Mendelson, *et al.*, "Local rademacher complexities," *The Annals of Statistics*, vol. 33, no. 4, pp. 1497–1537, 2005.

[79] V. Koltchinskii *et al.*, "Local rademacher complexities and oracle inequalities in risk minimization," *The Annals of Statistics*, vol. 34, no. 6, pp. 2593–2656, 2006.

[80] Y. Chen, C. Jin, and B. Yu, "Stability and convergence trade-off of iterative optimization algorithms," *arXiv preprint arXiv:1804.01619*, 2018.

[81] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[82] A. Maurer, "A vector-contraction inequality for rademacher complexities," in *International Conference on Algorithmic Learning Theory*, Springer, 2016, pp. 3–17.

[83] C. Cortes, V. Kuznetsov, M. Mohri, and S. Yang, "Structured prediction theory based on factor graph complexity," in *Advances in Neural Information Processing Systems*, 2016, pp. 2514–2522.

[84] O. Bousquet and A. Elisseeff, "Stability and generalization," *Journal of machine learning research*, vol. 2, no. Mar, pp. 499–526, 2002.

[85] S. Agarwal and P. Niyogi, "Generalization bounds for ranking algorithms via algorithmic stability," *Journal of Machine Learning Research*, vol. 10, no. Feb, pp. 441–474, 2009.

[86] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," *arXiv preprint arXiv:1509.01240*, 2015.

[87] O. Rivasplata, E. Parrado-Hernández, J. S. Shawe-Taylor, S. Sun, and C. Szepesvári, "Pac-bayes bounds for stable algorithms with instance-dependent priors," in *Advances in Neural Information Processing Systems*, 2018, pp. 9214–9224.

[88] S. Verma and Z.-L. Zhang, "Stability and generalization of graph convolutional neural networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1539–1548.

[89] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 1126–1135.

[90] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," in *Advances in Neural Information Processing Systems*, 2019, pp. 113–124.

[91] P. Donti, B. Amos, and J. Z. Kolter, "Task-based end-to-end model learning in stochastic optimization," in *Advances in Neural Information Processing Systems*, 2017, pp. 5484–5494.

[92] P. Knobelreiter, C. Reinbacher, A. Shekhovtsov, and T. Pock, "End-to-end training of hybrid cnn-crf models for stereo," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2339–2348.

[93] M. MacKay, P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse, "Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions," *arXiv preprint arXiv:1903.03088*, 2019.

[94] G. Denevi, C. Ciliberto, R. Grazzi, and M. Pontil, "Learning-to-learn stochastic gradient descent with biased regularization," in *International Conference on Machine Learning*, 2019, pp. 1566–1575.

[95] V. Koltchinskii and D. Panchenko, "Rademacher processes and bounding the risk of function learning," in *High dimensional probability II*, Springer, 2000, pp. 443–457.

[96] V. Koltchinskii, "Rademacher penalties and structural risk minimization," *IEEE Transactions on Information Theory*, vol. 47, no. 5, pp. 1902–1914, 2001.

[97] V. Koltchinskii, D. Panchenko, *et al.*, "Empirical margin distributions and bounding the generalization error of combined classifiers," *The Annals of Statistics*, vol. 30, no. 1, pp. 1–50, 2002.

[98] P. L. Bartlett and S. Mendelson, "Rademacher and gaussian complexities: Risk bounds and structural results," *Journal of Machine Learning Research*, vol. 3, no. Nov, pp. 463–482, 2002.

[99] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," in *Advances in Neural Information Processing Systems*, 2019, pp. 688–699.

[100] L. El Ghaoui, F. Gu, B. Travacca, and A. Askari, "Implicit deep learning," *arXiv preprint arXiv:1908.06315*, 2019.

[101] X. Chen, H. Dai, Y. Li, X. Gao, and L. Song, "Learning to stop while learning to predict," in *International Conference on Machine Learning*, PMLR, 2020, pp. 1520–1530.

[102] M. Jones, S. Kinoshita, and M. C. Mozer, "Optimal response initiation: Why recent experience matters," in *Advances in neural information processing systems*, 2009, pp. 785–792.

[103] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[104] J. Sun, H. Li, Z. Xu, *et al.*, "Deep admm-net for compressive sensing mri," in *Advances in neural information processing systems*, 2016, pp. 10–18.

[105] C. Metzler, A. Mousavi, and R. Baraniuk, "Learned d-amp: Principled neural network based compressive image recovery," in *Advances in Neural Information Processing Systems*, 2017, pp. 1772–1783.

[106] J. Domke, "Parameter learning with truncated message-passing," in *CVPR 2011*, IEEE, 2011, pp. 2937–2943.

[107] T. B. Yakar, R. Litman, P. Sprechmann, A. M. Bronstein, and G. Sapiro, "Bilevel sparse models for polyphonic music transcription.," in *ISMIR*, 2013, pp. 65–70.

[108] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," 2017.

[109] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few-shot learning," *arXiv preprint arXiv:1707.09835*, 2017.

[110] S. Qiao, C. Liu, W. Shen, and A. L. Yuille, "Few-shot image recognition by predicting parameters from activations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7229–7238.

[111] Y. Lee and S. Choi, "Gradient-based meta-learning with learned layerwise metric and subspace," *arXiv preprint arXiv:1801.05558*, 2018.

[112] D. Na *et al.*, "Learning to balance: Bayesian meta-learning for imbalanced and out-of-distribution tasks," in *International Conference on Learning Representations*, 2020.

[113] B. Oreshkin, P. R. López, and A. Lacoste, "Tadam: Task dependent adaptive metric for improved few-shot learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 721–731.

[114] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, IEEE, 2016, pp. 2464–2469.

[115] A. R. Zamir *et al.*, "Feedback networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1308–1317.

[116] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *International Conference on Learning Representations*, 2018.

[117] Y. Kaya, S. Hong, and T. Dumitras, "Shallow-deep networks: Understanding and mitigating network overthinking," in *International Conference on Machine Learning*, 2019, pp. 3301–3310.

[118] A. N. Shiryaev, *Optimal stopping rules*. Springer Science & Business Media, 2007, vol. 8.

[119] H. Pham, "Optimal stopping of controlled jump diffusion processes: A viscosity solution approach," in *Journal of Mathematical Systems, Estimation and Control*, Citeseer, 1998.

[120] C. Ceci and B. Bassan, "Mixed optimal stopping and stochastic control problems with semicontinuous final reward for diffusion processes," *Stochastics and Stochastic Reports*, vol. 76, no. 4, pp. 323–337, 2004.

[121] R. Dumitrescu, C. Reisinger, and Y. Zhang, "Approximation schemes for mixed optimal stopping and control problems with nonlinear expectations and jumps," *arXiv preprint arXiv:1803.03794*, 2018.

[122] S. Becker, P. Cheridito, and A. Jentzen, "Deep optimal stopping.," *Journal of Machine Learning Research*, vol. 20, no. 74, pp. 1–25, 2019.

[123] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[124] I. Higgins *et al.*, "Beta-VAE: Learning basic visual concepts with a constrained variational framework.," *ICLR*, vol. 2, no. 5, p. 6, 2017.

[125] S. Nowozin, B. Cseke, and R. Tomioka, "F-gan: Training generative neural samplers using variational divergence minimization," in *Advances in neural information processing systems*, 2016, pp. 271–279.

[126] T. Blumensath and M. E. Davies, "Iterative thresholding for sparse approximations," *Journal of Fourier analysis and Applications*, vol. 14, no. 5-6, pp. 629–654, 2008.

[127] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[128] T. A. Le, M. Igl, T. Rainforth, T. Jin, and F. Wood, "Auto-encoding sequential monte carlo," in *International Conference on Learning Representations*, 2018.

[129] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum, "One shot learning of simple visual concepts," in *Proceedings of the annual meeting of the cognitive science society*, vol. 33, 2011.

[130] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 898–916, 2010.

[131] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, IEEE, vol. 2, 2001, pp. 416–423.

[132] S. Lefkimmiatis, "Universal denoising networks: A novel cnn architecture for image denoising," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3204–3213.

[133] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[134] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Transactions on image processing*, vol. 16, no. 8, pp. 2080–2095, 2007.

[135] S. Gu, L. Zhang, W. Zuo, and X. Feng, "Weighted nuclear norm minimization with application to image denoising," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 2862–2869.

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

[137] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, vol. 9, no. 3, pp. 432–441, 2008.

[138] O. Banerjee, L. E. Ghaoui, and A. d'Aspremont, "Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data," *Journal of Machine learning research*, vol. 9, no. Mar, pp. 485–516, 2008.

[139] B. Rolfs, B. Rajaratnam, D. Guillot, I. Wong, and A. Maleki, "Iterative thresholding algorithm for sparse inverse covariance estimation," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1574–1582, 2012.

[140] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[141] A. J. Rothman, P. J. Bickel, E. Levina, J. Zhu, *et al.*, "Sparse permutation invariant covariance estimation," *Electronic Journal of Statistics*, vol. 2, pp. 494–515, 2008.

[142] P. Ravikumar, M. J. Wainwright, G. Raskutti, and B. Yu, "High-dimensional covariance estimation by minimizing l_1-penalized log-determinant divergence," *Electronic Journal of Statistics*, vol. 5, pp. 935–980, 2011.

[143] T. Van den Bulcke *et al.*, "Syntren: A generator of synthetic gene expression data for design and analysis of structure learning algorithms," *BMC bioinformatics*, vol. 7, no. 1, p. 43, 2006.

[144] X. Chen, H. Sun, and L. Song, "Provable learning-based algorithm for sparse recovery," in *International Conference on Learning Representations*, 2021.

[145] Z. Wang, H. Liu, and T. Zhang, "Optimal computational and statistical rates of convergence for sparse nonconvex learning problems," *Annals of statistics*, vol. 42, no. 6, p. 2164, 2014.

[146] A. Bora, A. Jalal, E. Price, and A. G. Dimakis, "Compressed sensing using generative models," in *International Conference on Machine Learning*, PMLR, 2017, pp. 537–546.

[147] L. Franceschi, P. Frasconi, M. Donini, and M. Pontil, "A bridge between hyperparameter optimization and larning-to-learn," *stat*, vol. 1050, p. 18, 2017.

[148] G. Denevi, C. Ciliberto, D. Stamos, and M. Pontil, "Learning to learn around a common mean," *Advances in Neural Information Processing Systems*, vol. 31, pp. 10 169–10 179, 2018.

[149] R. Liu, S. Cheng, Y. He, X. Fan, Z. Lin, and Z. Luo, "On the convergence of learning-based iterative methods for nonconvex inverse problems," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 12, pp. 3027–3039, 2019.

[150] P. Indyk, A. Vakilian, and Y. Yuan, "Learning-based low-rank approximations," *arXiv preprint arXiv:1910.13984*, 2019.

[151] A. Grover, E. Wang, A. Zweig, and S. Ermon, "Stochastic optimization of sorting networks via continuous relaxations," *arXiv preprint arXiv:1903.08850*, 2019.

[152] S. Wu *et al.*, "Learning a compressed sensing measurement matrix via gradient unrolling," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6828–6839.

[153] X. Wang, S. Yuan, C. Wu, and R. Ge, "Guarantees for tuning the step size using a learning-to-learn approach," in *International Conference on Machine Learning*, PMLR, 2021, pp. 10 981–10 990.

[154] A. Behboodi, H. Rauhut, and E. Schnoor, "Compressive sensing and neural networks from a statistical learning perspective," *arXiv preprint arXiv:2010.15658*, 2020.

[155] B. J. Joukovsky, T. Mukherjee, N. Deligiannis, *et al.*, "Generalization error bounds for deep unfolding rnns," in *Proceedings of Machine Learning Research*, Journal of Machine Learning Research, 2021.

[156] R. Gupta and T. Roughgarden, "A pac approach to application-specific algorithm selection," *SIAM Journal on Computing*, vol. 46, no. 3, pp. 992–1017, 2017.

[157] M.-F. Balcan, D. DeBlasio, T. Dick, C. Kingsford, T. Sandholm, and E. Vitercik, "How much data is sufficient to learn high-performing algorithms? generalization guarantees for data-driven algorithm design," in *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021, pp. 919–932.

[158] J. Fan and R. Li, "Variable selection via nonconcave penalized likelihood and its oracle properties," *Journal of the American statistical Association*, vol. 96, no. 456, pp. 1348–1360, 2001.

[159] J. Fan, Y. Feng, and Y. Wu, "Network exploration via the adaptive lasso and scad penalties," *The annals of applied statistics*, vol. 3, no. 2, p. 521, 2009.

[160] P.-L. Loh and M. J. Wainwright, "Regularized m-estimators with nonconvexity: Statistical and algorithmic theory for local optima," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 559–616, 2015.

[161] C.-H. Zhang, "Nearly unbiased variable selection under minimax concave penalty," *The Annals of statistics*, vol. 38, no. 2, pp. 894–942, 2010.

[162] T. Zhang, "Analysis of multi-stage convex relaxation for sparse regularization.," *Journal of Machine Learning Research*, vol. 11, no. 3, 2010.

[163] E. J. Candes and T. Tao, "Decoding by linear programming," *IEEE transactions on information theory*, vol. 51, no. 12, pp. 4203–4215, 2005.

[164] D. Guillot, B. Rajaratnam, B. T. Rolfs, A. Maleki, and I. Wong, "Iterative thresholding algorithm for sparse inverse covariance estimation," *arXiv preprint arXiv:1211.2532*, 2012.

[165] E. Belilovsky, K. Kastner, G. Varoquaux, and M. B. Blaschko, "Learning to discover sparse graphical models," in *International Conference on Machine Learning*, PMLR, 2017, pp. 440–448.

[166] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[167] T. Kouno *et al.*, "Temporal dynamics and transcriptional control using single-cell gene expression analysis," *Genome biology*, vol. 14, no. 10, pp. 1–12, 2013.

[168] A. Tsanas, M. Little, P. McSharry, and L. Ramig, "Accurate telemonitoring of parkinson's disease progression by non-invasive speech tests," *Nature Precedings*, pp. 1–1, 2009.

[169] J. Zhou, J. Chen, and J. Ye, "Malsar: Multi-task learning via structural regularization," *Arizona State University*, vol. 21, 2011.