Implicit Models: Theories and Applications

by

Fangda Gu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Laurent El Ghaoui, Chair
Professor Somayeh Sojoudi
Professor Murat Arcak
Professor Wenwu Zhu

Fall 2021

Implicit Models: Theories and Applications

Abstract

Implicit Models: Theories and Applications

by

Fangda Gu

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Laurent El Ghaoui, Chair

Deep implicit models are very recent developments on deep learning. Traditionally, deep learning methods rely on explicit forward feeding structures. Super deep structures are proposed to give better performance in various domains. Such approaches have posed difficulty in theoretical analysis and under perform shallower models in some domains. By introducing a recursive structure involving solution to an equilibrium equation in the forward feeding, implicit deep models capture the idea of infinitely deep neural networks while preserving simplicity in model representation, allowing theoretical analysis and better connection to previous efforts in math and control communities. Recent works on implicit models have demonstrated state-of-the-art empirical performances.

Despite great ambition, implicit models are very new and see theoretical and empirical challenges. From the theoretical aspect, efficient and effective training and evaluation of implicit models are still open problems. The naive training methods for implicit models are highly inefficient. Trivial initialization easily violates the validity conditions for implicit models. Robustness of implicit models are not well studied. From the empirical aspect, there are still a limited number of works on applying implicit models to solve real world problems. Such works also have not demonstrated significant performance boost over deep learning in general. Applications of implicit models in most areas are mostly unexplored even though implicit models fit in the deep learning framework easily.

In the dissertation, we introduce our theoretical and empirical contributions on deep implicit models. The presentation of the dissertation is split into two parts. The first part focuses on theoretical foundations for deep implicit models where we research the evaluation, training, and other topics for implicit deep learning and deep learning in general. The second part then explores the applications of deep implicit models and corresponding theories on real world machine learning applications. We show that implicit models can out-perform existing deep learning techniques in a set of tasks, thanks to the implicit structure which resembles infinitely deep neural networks.

To My Family

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express the greatest gratitude to my advisor, Professor Laurent El Ghaoui. I met Professor El Ghaoui when I was interning at Berkeley in my undergraduate. Since then, I have been deeply attracted by his genius and passion. It is such a privilege to join Professor El Ghaoui's lab. He pays a lot of attention to my growth and is always patient. Throughout my PhD research, he gives me a lot of freedom in choosing my desired research and has been very supportive all the time. His guidance has always been to the point and highly effective even when the subjects are out of his expertise. I have enjoyed every second I spend with Professor El Ghaoui in exploring interesting ideas. Outside of research, Professor El Ghaoui is very approachable, the leisure talk and hike with him has fueled my research and my life.

I would like to thank Professor Somayeh Sojoudi for the insightful discussions on implicit graph models and power grids. I would also like to thank her for her gracious support in serving as my master thesis co-signer, prelim exam member, qual committee chair and dissertation committee member. I would like to thank Professor Murat Arcak for his guidance on robust control and support in serving as my prelim exam chair, qual committee and dissertation committee member. I would like to thank Professor Wenwu Zhu for support both in graph neural network research and in serving as my qual committee and dissertation committee member. Thanks to Professor Tomlin Claire for laying a solid control background for me through her instructions and her support in serving as my prelim exam member.

I would like to thank other researchers and friends who have collaborated with me or helped me during my PhD study, without whom the work would not have been possible. Thanks to Armin Askari for the guidance in academic writing and for the discussions on implicit models and Fenchel lifted models. Thanks to Bertrand Travacca and Alicia Tsai for the discussions on implicit models. Thanks to Galaxy Yin, Professor Peter Seiler and Professor Ming Jin for their decisive support and guidance on building stabilizing controllers. Thanks to Heng Chang for his major contributions on implicit graph neural networks research. Thanks to Elisabeth Glista and Guillaume Darmet for discussions on power grids. Thanks to Forest Yang, Gaven Ma, and James Li for working together on academic studies of various optimization topics and happy time in leisure meals. Thanks to friends from Professor El Ghaoui's Lab, Armin Askari, Geoffrey Negiar, Forest Yang, and Alicia Tsai. I have enjoyed the discussions on optimization and our BBQs and dinners. Thanks to friends from 1771 Highland Place, Kaichen Dong, Jiachen Li, and Tiancheng Zhang. I have enjoyed the games and traveling with you. I would like to give special thanks to Shirley Salanio for her consistent support to push forward the logistics for my graduate study.

I would like to acknowledge organizations that provided financial support directly for me in my graduate study: Department of Electrical Engineering and Computer Sciences, UC Berkeley; Berkeley AI Research Lab; Haas School of Business, UC Berkeley; EDF Inc.; Alphabet Inc.

Finally I want to thank my family, Jingchun Cha, Nanzhou Gu for their love and support.

Chapter 1

Introduction

The advances in computation in both hardware and software have enabled the development of deep learning. Starting from AlexNet [95], a deep learning algorithm on computer vision classification, a completely different level of applications has emerged. Over the past few years, we have seen advanced computer vision (CV) applications such as object detection, segmentation, and image generation. In natural language processing (NLP), we also see dramatic improvements in tasks like documentation comprehension, machine translation, and knowledge graph learning. Apart from CV and NLP, most other computation areas including robotics perception and planning, reinforcement learning, graph representation learning, and biochemistry have seen unprecedented breakthroughs. All of these are made possible because of the development of deep learning software on deep neural network models and the computation hardware supporting efficient matrix algebra calculations.

Compared with the mind-blowing empirical glory of deep learning, we have seen limited theoretical results in support for it. This is due to the complexity introduced by deep neural networks where the model usually sees a huge number of layers, each of which contains nonlinear operations. This flexibility has enabled the empirical development of deep learning but also limited the theoretical analysis of it. Another observation on the field reveals the conflicts between researcher's intention on proposing deeper and deeper neural networks in search for better empirical performance and the ever-existing limitation by the finite nature of explicit deep models. In order to restore a simple mathematical representation for deep learning and capture the unlimited depth researchers are heading to, we propose implicit deep learning where the model prediction rule is as simple as a linear operation over the states that are generated from solution to an equilibrium equation. With appropriate training and inference, the model represents an infinitely deep neural network and covers most existing deep learning structures.

1.1 Design of Deep Learning Models

Although implicit models exceed the limitation of finite layers and bring mathematical simplicity, the design is actually largely originated from the current deep learning models. The designs for deep learning are highly diverse because the paradigm of feed forward evaluation and gradient back propagation applies almost no limitation other than that the forward and backward calculations are defined (not even need to be differentiable, *e.g.* ReLU). This enables high flexibility. It will be impossible to cover all structures here. So, we will go through two deep learning models which we find informative on the intuition of implicit models, residual networks and graph neural networks.

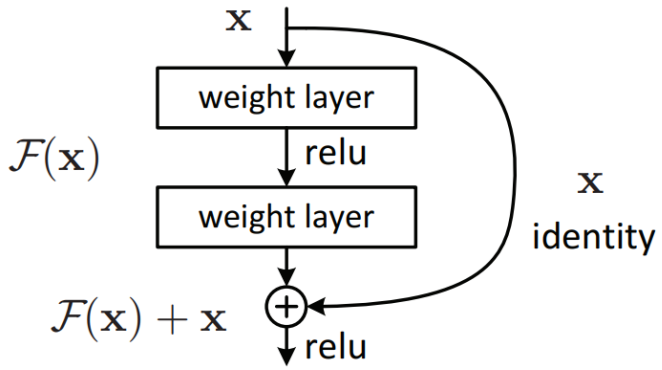


Figure 1.1: A residual block in ResNet [79]. Usually, the input and output share the same shape.

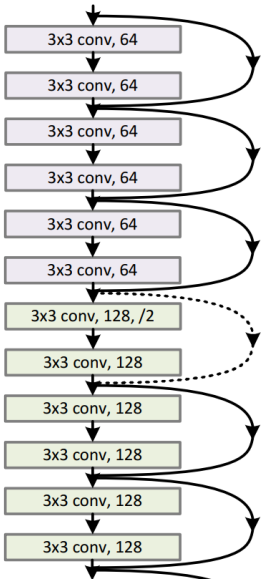


Figure 1.2: Part of ResNet-34 [79]. We see there are a lot of repetition of residual blocks in purple and green.

Residual network, or ResNet [79] is a major milestone on the performance of deep learning. Before the invention of ResNet, representative deep learning models including AlexNet [95], VGG [155], InceptionNet [162], have tens of layers and deeper models at the time that are created using combinations of the original convolution structures suffer from performance degradation and under performed these representative models. ResNet has introduced the so-called residual connection between layers to provide a shortcut data passage to make the layers learn the residual mapping instead of some desired underlying mapping. By applying the residual connection, the models are deepened to include hundreds and even thousands of layers. This has empirically improved the performance of deep CV models by a significant amount such that ResNet becomes the new representative deep learning model in CV. The ResNet is built from repetitive blocks of layers, where each block contains two convolution layers and a shortcut connection from the input to the output, see Figure 1.1. The depth of

ResNet comes from numerous repeating residual blocks, see Figure 1.2. To some extent, the larger the number of repetition, the higher the performance (see performance of ResNet-34, ResNet-50, ResNet-101, ResNet-152 in [79]). From such observation, it is natural to think of models that repeats such structure indefinitely.

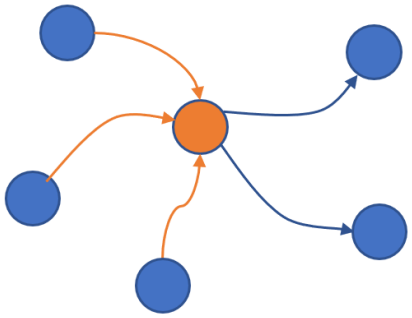


Figure 1.3: A graph convolution step aggregates neighbour information through a neural network to obtain new representation for the orange node.

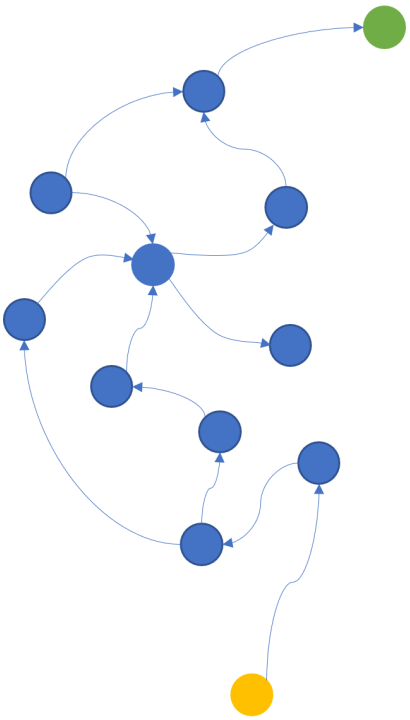


Figure 1.4: An example graph with nodes represented by circles and edges represented by arrows. Successful graph methods will capture contribution from the yellow node in the green node representation.

Graph neural networks, or GNNs are models developed to operate on graph-structured data, a more general and less structural type of data involving nodes and edges between nodes, see Figure 1.4. This type of model relies on the key message passing step between nodes, see Figure 1.3. The message passing step can be seen as a layer in deep learning and popular models contain a finite number of such steps. This is different from traditional graph algorithms like Pagerank [131] where some equilibrium state is sought and information passage steps are iterated to convergence. Numerous efforts are proposed to increase the number of layers for GNNs but, unlike ResNet, they have seen limited performance benefits and sometimes even under perform shallower models. One of the notable differences between GNNs and CV models is the fact that the forward iterations in GNNs can go in loops when there are loops in the graph. This departure from the low performance of deep GNNs and

the success of traditional graph algorithms signals the potential for infinitely deep GNNs and the need for theories for it to ensure higher performance.

1.2 Implicit Architectures in Machine Learning

Implicit architectures have a long history. In the 80s, Almeida [2] and Pineda [139] introduced perceptron networks with feedback or recurrent structures, which are the early forms of implicit models. In the early works, the idea of having a feedback structure in the design of neural networks is introduced and the gradient calculation via implicit function theorem [94] is demonstrated. This is later referred to as recurrent back propagation in [111] and implicit gradients in [55] and [15]. Despite the long history, these early implicit models have not received too much attention mainly due to the lack of theoretical support for stability in training and evaluation, and low empirical performance compared with explicit feed forward models.

Recently, implicit models and frameworks have received new attention from the machine learning community. Pioneered by [55] and [15] in 2019, the implicit models are reintroduced to the context of deep learning. By introducing a notationally simple but highly expressive form of implicit models, [55] answers the key question of when implicit models are well-posed (*i.e.* admit unique prediction outputs), explores the generalization of implicit models to cover modern deep learning models, and extends the robustness analysis as a demonstration of mathematical analysis made possible by the simple notations. The notation by [55] has great similarity to LTI systems and uncertain systems which are widely studied in the control community. It also fits into the analysis framework for recent control advances like [187] and [73].

Following the introduction of implicit models, [15, 72, 172, 17] have explored variants of implicit models and tested their empirical performance in language modeling, graph neural networks, object detection, and image classification settings. The empirical test performance on-par or better than state-of-the-art deep learning models are demonstrated. In these works, implicit models offer higher representation capability than explicit feed forward models and/or use fewer parameters when the test performance is on-par with the baselines. At the same time, [143, 177] study different well-posedness conditions under different assumptions. Recently, we also see study on the training of implicit models with a focus on efficient and stable implicit gradient calculation [25, 18, 135]. On the other side of the training, we also see Fenchel types of methods which solves convex sub-problems via introduced Fenchel divergence instead of calculating gradients [71, 166, 167].

Prior to the implicit models, implicit methods are used in model design. These methods focus on designing layers where the outputs cannot be given as a closed form solution of the inputs and model parameters. Compared with the implicit models, the designs are limited to specific domains and applications (*e.g.* logical structures [173], model predictive control [4], physical engines [13]). The methods are mostly trained using stochastic gradient methods with gradients obtained from implicit function theorem [94]. Other efforts on providing novel

new structures or plug-ins for deep learning framework using implicit designs [3, 50] have also inspired the development of implicit models.

1.3 Thesis Contributions and Organization

This dissertation is broken apart into two parts on theoretical and empirical sides of deep implicit models. The first part focuses on the theoretical foundations of implicit models and the second part discusses the empirical applications of deep implicit models.

Part I: Theoretical Foundations for Deep Implicit Models

Since the proposals of the early implicit models [139, 2], there are a lot of fundamental theoretical glitches with implicit models. Even when the implicit models start receiving new perspectives from the deep learning community in 2019, the key questions on the validity or well-posedness of implicit models (*i.e.* Is the implicit model giving a unique prediction?) is not well discussed. On the training side, only gradient methods based on the implicit function theorem are explored. But we have seen little discussions on the existence of such gradients.

The first half of the dissertation aims to lay a theoretical ground for deep implicit models through the introduction of a novel implicit deep learning framework and discovery of Fenchel divergence in reformulating training problems in deep learning. We show that the novel implicit deep learning framework is notationally simple and at the same time covers a wide range of deep learning architectures. Through the lenses of the simple notation, we perform a multitude of theoretical analysis including the discussion of well-posedness, robustness, sparsity, and training. The discovery of Fenchel divergence enables new alternative minimizing training methods for deep learning, which gives on-par performance to gradient methods without computing gradients. Both venues together inspire new empirical and theoretical advances on implicit models.

Chapter 2: Implicit Deep Learning

This chapter presents our pioneering effort on building a general and powerful implicit deep learning framework. Implicit deep learning prediction rules generalize the recursive rules of feedforward neural networks. Such rules are based on the solution of a fixed-point equation involving a single vector of hidden features, which is thus only implicitly defined. Through introduction of a greatly simplified notation highly similar to LTI systems and uncertainty systems in control, the new framework uses Perron-Frobenius theory to characterize the well-posedness of implicit models and extends a range of analysis in training through implicit gradients, Fenchel reformulation and robustness design through sensitivity matrix. Numerical experiments in synthetic datasets and real world datasets including hand-written character classification and traffic sign identification are conducted. On-par performance against

feedforward neural networks is demonstrated. The framework also opens up possibilities in terms of novel architectures and algorithms, robustness analysis and design, interpretability, sparsity, and network architecture optimization.

The content of the chapter is based on a paper, Implicit Deep Learning, released online in 2019 and later published in 2021 on SIAM Journal on Mathematics of Data Science [55].

Chapter 3: Fenchel Lifted Networks

To give a different training methods for implicit models and deep learning in general, we study the key invention of Fenchel divergence on lifted problems which converts the training problems into multiple convex sub-problems and enables training without gradients. Despite the recent successes of deep neural networks, the corresponding training problem remains highly non-convex and difficult to optimize. Classes of models have been proposed that introduce greater structure to the objective function at the cost of lifting the dimension of the problem. However, these lifted methods sometimes perform poorly compared to traditional gradient based training methods. We introduce a new class of lifted models, Fenchel lifted networks, that enjoy the same benefits as previous lifted models, without suffering a degradation in performance over classical gradient methods. Our model uses Fenchel divergence to represent activation functions as equivalent biconvex constraints and uses Lagrange multipliers to arrive at a rigorous lower bound of the traditional neural network training problem. This model is efficiently trained using block-coordinate descent and is parallelizable across data points and/or layers. We compare our model against standard fully connected and convolutional networks and show that we are able to match or beat their performance.

The content of the chapter is based on a paper, Fenchel Lifted Networks: A Lagrange Relaxation of Neural Network Training, published in 2020 on International Conference on Artificial Intelligence and Statistics [71].

Part II: Applications of Deep Implicit Models

The empirical applications of implicit models are parallelly important as the theoretical foundations. Unless the implicit models are competent with explicit feed forward deep learning models, the development of implicit models will remain niche. In the second half of the dissertation, we present our efforts on pushing implicit models to give high empirical performances in two different important settings, graph neural networks and stable controller synthesis. In the graph neural networks settings, we use implicit models to perform an infinite message passing process in graphs and obtain highly informative representation that captures long-range dependencies missed out by previous state-of-the-art graph neural network methods. On the stable controller synthesis side, we leverage the great simplicity of implicit deep learning framework to capture a recurrent neural network structure and formulate a controller synthesis method with stability guarantee. These two applications of implicit models signify the empirical prospects of implicit models in deep learning.

Chapter 4: Implicit Graph Neural Networks

In an exploration of infinite depth in deep learning, representation learning in graph-structured data comes naturally. Graph Neural Networks (GNNs) are widely used deep learning models that learn meaningful representations from graph-structured data. Due to the finite nature of the underlying recurrent structure, current GNN methods may struggle to capture long-range dependencies in underlying graphs. To overcome this difficulty, we propose a graph learning framework, called Implicit Graph Neural Networks (IGNNs), where we employ an implicit prediction rule based on the solution of a fixed-point equilibrium equation involving implicitly defined "state" vectors. We use the Perron-Frobenius theory to derive sufficient conditions that ensure well-posedness of the framework. Leveraging implicit differentiation, we derive a tractable projected gradient descent method to train the framework. Experiments on a comprehensive range of tasks show that IGNNs consistently capture long-range dependencies and outperform the state-of-the-art GNN models.

The content of the chapter is based on a paper, Implicit Graph Neural Networks, published in 2020 on Advances in Neural Information Processing Systems [72].

Chapter 5: Stable Controller Synthesis for Partially Observed Systems

The simple notations from implicit deep learning framework fits in the control analysis in a straightforward manner. We use the implicit deep learning framework to capture a recurrent neural network controller and follow up with stability analysis to synthesize stable neural network controllers in partially observed systems. Neural network controllers have become popular in control tasks thanks to their flexibility and expressivity. Stability is a crucial property for safety-critical dynamical systems, while stabilization of partially observed systems, in many cases, requires controllers to retain and process long-term memories of the past. We consider the important class of recurrent neural networks (RNNs) as dynamic controllers for nonlinear uncertain partially-observed systems, and derive convex stability conditions based on integral quadratic constraints, S-lemma and sequential convexification. To ensure stability during the learning and control process, we propose a projected policy gradient method that iteratively enforces the stability conditions in the reparameterized space taking advantage of mild additional information on system dynamics. Numerical experiments show that our method learns stabilizing controllers while using fewer samples and achieving higher final performance compared with policy gradient.

The content of the chapter is based on a work released on Arxiv in 2021 [73].

Chapter 6: Conclusion and Future Research

The chapter concludes the dissertation on the theoretical and empirical contributions to deep implicit models. Implicit models are still in its early stage in the context of deep learning where we see numerous innovative applications. Most of them, we believe, will benefit from the implicit models either in reduced parameter memory usage or in higher empirical performance. On the theoretical front, we see potential new developments following better

well-posedness conditions, more efficient training and inference, and new results on deep learning in general.

Part I

Theoretical Foundations for Deep Implicit Models

Chapter 2

Implicit Deep Learning

Implicit deep learning is an important implicit model framework which also pivots this dissertation. At the time when we initially put forward this work in 2019, implicit models were not popular and many details are unclear. We would like to bring more popularity to the field by coming up with a general and complete framework which not only is flexible to connect to deep learning methods but also links to the vast literature of the control community. As a result, we have proposed this framework that is notationally simple but also powerful enough to capture a range of deep learning models. We have done a few numerical experiments to compare against deep learning in a direct manner. And later in Chapter 4, we extend this framework to graph neural networks and show it enjoys unparalleled performance. In Chapter 5, we use the notation to capture recurrent neural network controllers which are iteratively synthesized to stabilize non-linear plants.

2.1 Introduction

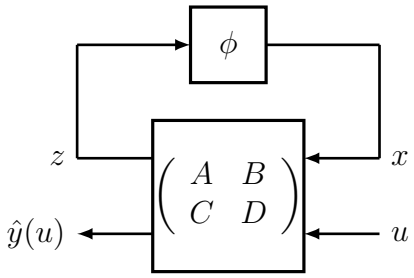


Figure 2.1: A block-diagram view of an implicit model.

Implicit prediction rules

In this chapter, we consider a new class of deep learning models that are based on implicit prediction rules. Such rules are not obtained via a recursive procedure through several layers, as in current neural networks. Instead, they are based on solving a fixed-point equation in some single "state" vector $x \in \mathbb{R}^n$. Precisely, for a given input vector u, the predicted vector is

$$\hat{y}(u) = Cx + Du \qquad \text{(prediction equation)} \tag{2.1a}$$

$$x = \phi(Ax + Bu) \qquad \text{(equilibrium equation)} \tag{2.1b}$$

where $\phi : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear vector map (the "activation" map), and matrices A, B, C, D contain model parameters. Figure 2.1 provides a block-diagram view of an implicit model, to be read from right to left, so as to be consistent with matrix-vector multiplication rules.

We can think of the vector $x \in \mathbb{R}^n$ as a "state" corresponding to n "hidden" features that are extracted from the inputs, based on the so-called equilibrium equation (2.1b). In general, that equation cannot be solved in closed-form, and the model above provides x only *implicitly*. This equation is not necessarily well-posed, in the sense that it may not admit a solution, let alone a unique one; we discuss this important issue of well-posedness in Section 2.3.

For notational simplicity only, our rule does not contain any bias terms; we can easily account for those by considering the vector $(u, 1)$ instead of u, thereby increasing the column dimension of B by one.

Perhaps surprisingly, as seen in Section 2.4, the implicit framework includes most current neural network architectures as special cases. Implicit models are a much wider class: they present much more capacity, as measured by the number of parameters for a given dimension of the hidden features; also, they allow for cycles in the network, which is not permitted under the current paradigm of deep networks.

Implicit rules open up the possibility of using novel architectures and prediction rules for deep learning, which are not based on any notion of "network" or "layers", as is classically understood. In addition, they allow one to consider rigorous approaches to challenging problems in deep learning, ranging from robustness analysis, sparsity and interpretability, and feature selection.

Contributions and chapter outline

Our contributions in this chapter, and its outline, are as follows.

- *Well-posedness and composition* (2.3): In contrast with standard deep networks, implicit models may not be well-posed, in the sense that the equilibrium equation may have no or multiple solutions. We establish rigorous and numerically tractable conditions for implicit rules to be well-posed. These conditions are then used in the training problem, guaranteeing the well-posedness of the learned prediction rule. We also discuss the composition of implicit models, via cascade connections for example.

- *Implicit models of neural networks* (2.4): We provide details on how to represent a wide variety of neural networks as implicit models, building on the composition rules of Section 2.3.
- *Robustness analysis* (2.5): We describe how to analyze the robustness properties of a given implicit model, deriving bounds on the state under input perturbations, and generating adversarial attacks. We also discuss which penalties to include into the training problem so as to encourage robustness of the learned rule.
- *Interpretability, sparsity, compression and deep feature selection* (2.6): Here we focus on finding appropriate penalties to use in order to improve properties such as model sparsity, or obtain feature selection. We also discuss the impact of model errors.
- *Training problem: formulations and algorithms* (2.7): Informed by our previous findings, we finally discuss the corresponding training problem. Following the work of [71] and [108], we represent activation functions using so-called Fenchel divergences, in order to relax the training problem into a more tractable form. We discuss several algorithms, including stochastic projected gradients, Frank-Wolfe, and block-coordinate descent.

Finally, Section 2.8 provides a few experiments supporting the theory put forth in this chapter. Our Section 2.2 is devoted to prior work and references.

Notation

For a matrix U, $|U|$ (resp. U_+) denotes the matrix with the absolute values (resp. positive part) of the entries of U. For a vector v, we denote by $\mathbf{diag}(v)$ the diagonal matrix formed with the entries of v; for a square matrix V, $\mathbf{diag}(V)$ is the vector formed with the diagonal elements of V. The notation $\mathbf{1}$ refers to the vector of ones, with size inferred from context. The Hadamard (componentwise) product between two n-vectors x, y is denoted $x \odot y$. We use $s_k(z)$ to denote the sum of the largest k entries of a vector z. For a matrix A, and integers $p, q \geq 1$, we define the induced norm

$$\|A\|_{p\to q} = \max_{\xi} \ \|A\xi\|_q \ : \ \|\xi\|_p \leq 1.$$

The case when $p = q = \infty$ corresponds to the l_∞-induced norm of A, also known as its *max-row-sum norm*:

$$\|A\|_\infty := \max_i \sum_j |A_{ij}|.$$

We denote the set $\{1, \cdots, L\}$ compactly as $[L]$. For a n-vector partitioned into L blocks, $z = (z_1, \ldots, z_L)$, with $z_l \in \mathbb{R}^{n_l}$, $l \in [L]$, with $n_1 + \ldots + n_L = n$, we denote by $\eta(z)$ the L-vector of norms:

$$\eta(z) := (\|z_1\|_{p_1}, \ldots, \|z_L\|_{p_L})^\top. \tag{2.2}$$

Finally, any square, non-negative matrix M admits a real eigenvalue that is larger than the modulus of any other eigenvalue; this non-negative eigenvalue is the so-called *Perron-Frobenius eigenvalue* [121], and is denoted $\lambda_{\text{pf}}(M)$.

2.2 Related Work

In implicit deep learning

Recent works have considered versions of implicit models, and demonstrated their potential in deep learning. In the pioneering work by Bai and collaborators [16, 93, 17, 72] the authors demonstrated empirical success of an entirely implicit framework, which they call Deep Equilibrium Models. They present a general form of implicit model based on an implicit equation of the form

$$z_{1:T}^{[i+1]} = f_\theta(z_{1:T}^{[i]}; u_{1:T}),\ z_{1:T}^{[0]} = 0$$

where i is the layer index; $z_{1:T}^{[i]}$ is the hidden sequence of length T at layer i; $u_{1:T} = [u_1, \cdots, u_T] \in \mathbb{R}^{T\times p}$ is the input sequence, where $u_i \in \mathbb{R}^p$ and $T \in \mathbb{N}$; and f_θ is some non-linear transformation. This formulation represents the class of *weight-tied* sequence models, where the same transformation f_θ is used for all layers, reminiscent of recurrent neural networks. The authors show that any deep network can be represented by this weight-tied representation, akin to the reformulation in Section 2.4.

The models are then trained using quasi-Newton methods and gradients are computed using the implicit function theorem. The main difference with our approach lies in the fact that the above-mentioned work focuses on obtaining empirical results in the context of natural language processing and computer vision, where our work focuses on theoretical foundations such as well-posedness and robustness.

In the more recent work [177], the authors use the same structure as ours (2.1b), and do provide results pertaining to well-posedness. The difference with our approach there lies in the assumptions made on the activation function ϕ. Instead of the BLIP (blockwise Lipschitz) assumption, the authors propose that the activation should be a proximal operator for some convex function g, of the form

$$\phi(x) = \arg\min_z \frac{1}{2}\|x - z\|^2 + g(z).$$

Note that the ReLU activation can be represented as a proximal operator with $g(z) = I(z \geq 0)$, where I is the indicator function. The authors then observe that under this assumption on ϕ, a condition for well-posedness is

$$(1 - m)I \succeq \frac{A + A^\top}{2} \tag{2.3}$$

with $m > 0$. This condition is different from ours, but the two are not equivalent. As an example, we can choose ϕ to be the ReLU and $A = -2I$. A does not satisfy our condition of

well-posedness, since $\lambda_{\text{pf}}(A) = 2$, but does satisfy (2.3) for $m = 1$. Conversely, for the choice

$$A = \begin{bmatrix} 0.5 & 0 \\ 2 & -0.5 \end{bmatrix}$$

we have $\lambda_{\text{pf}}(A) = 0.5 < 1$, but the eigenvalues of $\frac{1}{2}(A + A^\top)$ are $\{\pm\frac{\sqrt{5}}{2}\}$, therefore (2.3) is not satisfied. The authors then show how to compute a solution to the equilibrium equation using splitting techniques for monotone operators, mainly the forward-backward and Peaceman-Rachford algorithms, which serves a similar purpose to the Picard iterations we propose. Finally, the authors also use a form of implicit differentiation using these algorithms to learn the parameters of the model.

Prior to the implicit frameworks, some authors have used implicit types of methods in model design. [39] uses implicit methods to solve and construct a general class of models known as neural ordinary differential equations, while [13] uses implicit models to construct a differentiable physics engine that enables gradient-based learning and high sample efficiency. Furthermore, many papers explore the concept of integrating implicit models with modern deep learning methods in a variety of ways. For example, [173] show promise in integrating logical structures into deep learning by incorporating a semidefinite programming (SDP) layer into a network in order to solve a (relaxed) MAXSAT problem; see also [173]. In [4] the authors propose to include a model predictive control as a differentiable policy class for deep reinforcement learning, which can be seen as a kind of implicit architecture. In [3] the authors introduced implicit layers where the activation is the solution of some quadratic programming problem; in [50], the authors incorporate stochastic optimization formulation for end-to-end learning task, in which the model is trained by differentiating the solution of a stochastic programming problem.

In robust control

Our approach is rooted in the field of robust control analysis and design. The idea of analyzing (linear) dynamical systems subject to uncertainty via optimization-based approaches has a long history; most relevant to our approach are the landmark references [45, 86], which delineate an approach, based on linear programming, that focuses on the so-called l_∞-to-l_∞ gain of a dynamical system; it employs a technique that embeds non-linearities in so-called sector bounds, and uses corresponding relaxations. Our results pertaining to sensitivity matrices are in direct line of that kind of analysis. Also relevant is the more recent work [186, 187], which analyzes stability of a system controlled by a neural network, and obtain the region of attraction for such system using a "state-space" representation for the neural network similar to ours.

In lifted models

In implicit learning, there is usually no way to express the state variable in closed-form, which makes the task of computing gradients with respect to model parameters challenging. Thus,

a natural idea in implicit learning is to keep the state vector as a variable in the training problem, resulting in a higher-dimensional (or, "lifted") expression of the training problem. The idea of lifting the dimension of the training problem in (non-implicit) deep learning by introducing "state" variables has been studied in a variety of works [164, 9, 71, 64, 190, 194, 31, 108]. Lifted models are trained using block coordinate descent methods, Alternating Direction Method of Multipliers (ADMM) or iterative, non-gradient based methods. In this work, we introduce a novel aspect of lifted models, namely the possibility of defining a prediction rule implicitly.

In robustness analysis of neural networks

The issue of robustness in deep learning is generating quite a bit of attention, due to the fact that many deep learning models suffer from the lack of robustness. Prior relevant work have demonstrated that deep learning models are vulnerable to adversarial attacks [68, 97, 133, 98]. The work [140] explores SDP relaxations to the attack problem. The vulnerability issue of deep learning models have motivated the study of corresponding defense strategies [117, 132, 140, 70, 150, 178, 44]. However, many of the defense strategies are later shown to be ineffective [10, 30], suggesting the needs for the theoretical understanding of robustness evaluations for deep learning model. In this work, we formalize the robustness analysis of deep learning via the lens of the implicit model. A large number of deep learning architectures can be modeled using implicit prediction rules, making our robustness evaluation a versatile analysis tool.

In sparsity, compression and deep feature selection

Sparsity and compression, which are well understood in classical settings, have found their place in deep learning and are an active branch of research. Early work in pruning dates back to as early as the 90s [101, 78] and has since gained interest. In [156], the authors showed that by randomly dropping units (*i.e.* increasing the sparsity level of the network or compressing the network) reduces overfitting and improved the generalization performance of networks. Recently, more sophisticated ways of pruning networks have been proposed, in an effort to reduce the overall size of the model, while retaining or accepting a modest decrease in accuracy: a non-extensive list of works include [202, 126, 76, 149, 6, 100, 36, 114, 56].

2.3 Well-Posedness and Composition

Assumptions on the activation map

We restrict our attention to activation maps ϕ that obey a "Blockwise LIPschitz" (BLIP) continuity condition. This condition is satisfied for most popular activation maps, and arises

naturally when "composing" implicit models (see Section 2.3). Precisely, we assume that:

1. *Blockwise:* the map ϕ acts in a block-wise fashion, that is, there exist a partition of n: $n = n_1 + \ldots + n_L$ such that for every vector partitioned into the corresponding blocks: $z = (z_1, \ldots, z_L)$ with $z_l \in \mathbb{R}^{n_l}$, $l \in [L]$, we have $\phi(z) = (\phi_1(z_1), \ldots, \phi_L(z_L))$ for appropriate maps $\phi_l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$, $l \in [L]$.

2. *Lipschitz:* For every $l \in [L]$, the maps ϕ_l are Lipschitz-continuous with constant $\gamma_l > 0$ with respect to the l_{p_l}-norm for some integer $p_l \geq 1$:

$$\forall\, u, v \in \mathbb{R}^{n_l} \ : \ \|\phi_l(u) - \phi_l(v)\|_{p_l} \leq \gamma_l \|u - v\|_{p_l}.$$

In the remainder of the chapter, we refer to such maps with the acronym BLIP, omitting the dependence on the underlying structure information (integers n_l, p_l, γ_l, $l \in [L]$). We shall consider a special case, referred to a COmponentwise Non-Expansive (CONE) maps, when $n_l = 1$, $\gamma_l = 1$, $l \in [L]$. Such CONE maps satisfy

$$\forall\, u, v \in \mathbb{R}^n \ : \ |\phi(u) - \phi(v)| \leq |u - v|, \tag{2.4}$$

with inequality and absolute value taken componentwise. Examples of CONE maps include the ReLU (defined as $\phi(\cdot) = \max(0, \cdot)$) and its "leaky" variants, tanh, sigmoid, each applied componentwise to a vector input. Our model also allows for maps that do not operate componentwise, such as the softmax function, which operates on a n-vector z as:

$$z \to \text{SoftMax}(z) := \left(\frac{e^{z_i}}{\sum_i e^{z_j}} \right)_{i \in [n]}, \tag{2.5}$$

The softmax map is 1-Lipschitz-continuous with respect to the l_1-norm [65].

Well-posed matrices

We consider the prediction rule (2.1a) with input point $u \in \mathbb{R}^p$ and predicted output vector $\hat{y}(u) \in \mathbb{R}^q$. The equilibrium equation (2.1b) does not necessarily have a well-defined, unique solution x. In order to ensure this, we assume that the $n \times n$ matrix A satisfies the following well-posedness property.

Definition 2.3.1 (Well-posedness property)**.** *The $n \times n$ matrix A is said to be well-posed for ϕ (in short, $A \in WP(\phi)$) if, for any n-vector b, the equation in $x \in \mathbb{R}^n$:*

$$x = \phi(Ax + b) \tag{2.6}$$

has a unique solution.

There are many classes of matrices that satisfy the well-posedness property. As seen next, strictly upper-triangular matrices are well-posed with respect to any activation map that acts componentwise; such a class arises when modeling feedforward neural networks as implicit models, as seen in Section 2.4.

Tractable sufficient conditions for well-posedness

Our goal now is to understand how we can constrain A to have the well-posedness property, in a numerically tractable way.

We assume that ϕ is a BLIP map, as defined in Section 2.3. We partition the A matrix according to the tuple $(n_1, \ldots, n_L)$, into blocks $A_{ij} \in \mathbb{R}^{n_i \times n_j}$, $1 \leq i, j \leq L$, and define a $L \times L$ matrix of induced norms $N(A, \gamma) \in \mathbb{R}_+^{L \times L}$, with elements for $l, h \in [L]$ given by

$$(N(A,\gamma))_{ij} := \gamma_i \|A_{ij}\|_{p_j \to p_i} = \gamma_i \max_{\xi} \ \|A_{ij}\xi\|_{p_i} \ : \ \|\xi\|_{p_j} \leq 1. \tag{2.7}$$

In the case of CONE maps, the vector γ is all ones, and we have simply $N(A, \gamma) = |A|$.

The sufficient condition stated next is based on the contraction mapping theorem [145, p.83].

Theorem 2.3.1 (PF sufficient condition for well-posedness for BLIP activation)**.** *Assume that ϕ satisfies the BLIP condition, as defined in Section 2.3. Then, A is well-posed with respect to ϕ if*

$$\lambda_{pf}(N(A,\gamma)) < 1, \tag{2.8}$$

where $N(A, \gamma)$ is the matrix of induced norms defined in (2.7). Then, for any n-vector b, the solution to the equation (2.6) can be computed via the fixed-point iteration

$$x(0) = 0, \ \ x(t+1) = \phi(Ax(t) + b), \ \ t = 0, 1, 2, \ldots. \tag{2.9}$$

When ϕ is a CONE map, the PF condition (2.8) reduces to $\lambda_{pf}(|A|) < 1$.

Proof. Let $b \in \mathbb{R}^n$. Our first step is to establish that for the Picard iteration (2.9), we have, for every $t \geq 1$:

$$\eta(x(t+1) - x(t)) \leq N(A,\gamma)\eta(x(t) - x(t-1))$$

Here, η is a vector of norms, as defined in (2.2). For every $l \in [L]$, $t \geq 0$:

$$\begin{aligned}
[\eta(x(t+1) - x(t))]_l &= \|\phi_l([Ax(t) + b]_l) - \phi_l([Ax(t-1) + b]_l)\|_{p_l} \quad \text{[using (2.9)]} \\
&\leq \gamma_l \|[A(x(t) - x(t-1))]_l\|_{p_l} = \gamma_l \| \sum_{h \in [L]} A_{lh}(x(t) - x(t-1))_h \|_{p_l} \\
&\leq \gamma_l \sum_{h \in [L]} \|A\|_{p_h \to p_l} \|x_h(t) - x_h(t-1)\|_{p_h} = [N(A,\gamma)\eta(x(t) - x(t-1))]_l,
\end{aligned}$$

which establishes the desired bound, where $M := N(A, \gamma)$.

Assume now that $\lambda_{\text{pf}}(M) < 1$, as posited in the Theorem. Then, from the Perron-Frobenius theorem, $I - M$ is non-singular and all the other (possibly complex) eigenvalues λ of $N(A, \gamma)$ satisfy $|\lambda| \leq \lambda_{\text{pf}}(N(A, \gamma)) < 1$. We prove existence of a solution to the equilibrium equation by showing that the sequence of Picard iterates is Cauchy: for every $t, \tau \geq 0$,

$$\eta(x(t+\tau) - x(t)) \leq \sum_{k=t}^{t+\tau} M^k \eta(x(1) - x(0)) \leq M^t \sum_{k=0}^{\tau} M^k \eta(x(1) - x(0)) \leq M^t w,$$

where $w \in \mathbb{R}^L_+$ defined by,

$$w := \sum_{k=0}^{+\infty} M^k \eta(x(1) - x(0)) = (I - M)^{-1} \eta(x(1) - x(0)).$$

As M^t converges to 0 irrespective of τ, Picard iterates is Cauchy and therefore the sequence $\{x(t)\}$ converges to $x \in \mathbb{R}^n$ and $x = \phi(Ax + b)$. The above proves the existence of a solution.

To prove unicity, consider $x^1, x^2 \in \mathbb{R}^n_+$ two solutions to the equation. Using the hypotheses in the theorem, we have, for any $k \geq 1$:

$$\eta(x^1 - x^2) \leq M\eta(x^1 - x^2) \leq M^k \eta(x^1 - x^2).$$

The fact that $M^k \to 0$ as $k \to +\infty$ then establishes unicity. ■

Remark 2.3.1. *The fixed-point iteration (2.9) has linear convergence; each iteration is a matrix-vector product, hence the complexity of solving the equilibrium equation is comparable to that of a forward pass through a network of similar size.*

Remark 2.3.2. *The PF condition $\lambda_{pf}(N(A, \gamma)) < 1$ is not convex in A, but the convex condition $\|N(A, \gamma)\|_\infty < 1$, is sufficient, in light of the bound $\|M\|_\infty \geq \lambda_{pf}(|M|)$, valid for any square matrix M.*

Remark 2.3.3. *The PF condition of Theorem 2.3.1 is conservative. For example, a triangular matrix A is well-posed with respect to the ReLU and if only if $\boldsymbol{diag}(A) < \mathbf{1}$, a consequence of the upcoming Theorem 2.3.2. The corresponding equilibrium equation can then be solved via the backward recursion*

$$x_n = \frac{(b_n)_+}{1 - A_{nn}}, \quad x_i = \frac{1}{1 - A_{ii}}(b_i + \sum_{j>i} A_{ij} x_j)_+, \quad i = n-1, \ldots, 1.$$

Such a matrix does not necessarily satisfy the PF condition; we can have in particular $A_{11} < -1$, which implies $\lambda_{pf}(|A|) > 1$.

Remark 2.3.4. *The well-posedness property is invariant under row and column permutation, provided ϕ acts componentwise. Precisely, if A is well-posed with respect to a componentwise CONE map ϕ, then for any $n \times n$ permutation matrix P, $PAP^\top$ is well-posed with respect to ϕ. The PF sufficient condition is also invariant under row and column permutations. A similar statement can be made for the more general BLIP case.*

Composition of implicit models

Implicit models can be easily composed via matrix algebra. Sometimes, the connection preserves well-posedness, thanks to the following result.

Theorem 2.3.2 (Well-posedness of block-triangular matrices, componentwise activation)**.** *Assume that the activation map ϕ acts componentwise. The upper block-triangular matrix*

$$A := \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

with $A_{ii} \in \mathbb{R}^{n_i \times n_i}$, $i = 1, 2$, is well-posed with respect to ϕ if and only if its the diagonal blocks A_{11}, A_{22} are.

Proof. Express the equation $x = \phi(Ax + b)$ as

$$x_1 = \phi(A_{11}x_1 + A_{12}x_2 + b_1), \;\; x_2 = \phi(A_{22}x_2 + b_2),$$

where $b = (b_1, b_2)$, $x = (x_1, x_2)$, with $b_i \in \mathbb{R}^{n_i}$, $x_i \in \mathbb{R}^{n_i}$, $i = 1, 2$. Here, since ϕ acts componentwise, we use the same notation ϕ in the two equations.

Now assume that A_{11} and A_{22} are well-posed with respect to ϕ. Since A_{22} is well-posed for ϕ, the second equation has a unique solution x_2^*; plugging $x_2 = x_2^*$ into the second equation, and using the well-posedness of A_{11}, we see that the first equation has a unique solution in x_1, hence A is well-posed.

To prove the converse direction, assume that A is well-posed. The second equation above must have a unique solution x_2^*, irrespective to the choice of b_2, hence A_{22} must be well-posed. To prove that A_{11} must be well-posed too, set $b_2 = 0$, b_1 arbitrary, leading to the system

$$x_1 = \phi(A_{11}x_1 + A_{12}x_2 + b_1), \;\; x_2 = \phi(A_{22}x_2).$$

Since A_{22} is well-posed for ϕ, there is a unique solution x_2^* to the second equation; the first equation then reads $x_1 = \phi(A_{11}x_1 + b_1 + A_{12}x_2^*)$. It must have a unique solution for any b_1, hence A_{11} is well-posed. ∎

This result establishes the fact stated previously, that when ϕ is the ReLU, an upper-triangular matrix $A \in \mathrm{WP}(\phi)$ if and only if $\mathbf{diag}(A) < \mathbf{1}$. A similar result holds with the lower block-triangular matrix

$$A := \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix},$$

where $A_{12} \in \mathbb{R}^{n_1 \times n_2}$ is arbitrary. It is possible to extend this result to activation maps ϕ that satisfy the block Lipschitz continuity (BLIP) condition, in which case we need to assume that the partition of A into blocks is consistent with that of ϕ. As seen later, this feature arises naturally when composing implicit models from well-posed blocks.

Theorem 2.3.3 (Well-posedness of block-triangular matrices, blockwise activation)**.** *Assume that the matrix A can be written as*

$$A := \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$$

with $A_{ii} \in \mathbb{R}^{n_i \times n_i}$, $i = 1, 2$, and ϕ acts blockwise accordingly, in the sense that there exist two maps ϕ_1, ϕ_2 such that $\phi((z_1, z_2)) = (\phi_1(z_1), \phi_2(z_2))$ for every $z_i \in \mathbb{R}^{n_i}$, $i = 1, 2$. Then A is well-posed with respect to ϕ if and only if for $i = 1, 2$, A_{ii} is well-posed with respect to ϕ_i.

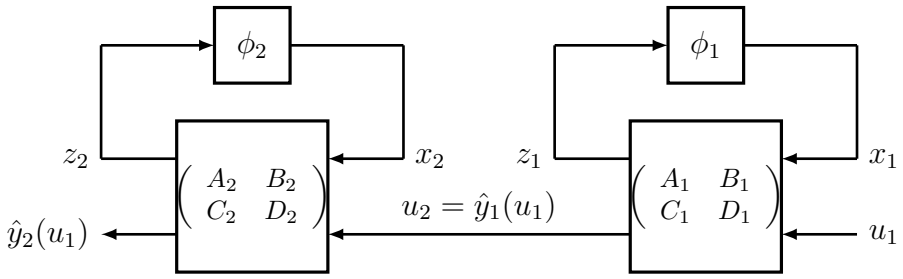


Figure 2.2: Cascade connection of two implicit models.

Using the above results, we can preserve well-posedness of implicit models via composition. For example, given two models with matrix parameters (A_i, B_i, C_i, D_i) and activation functions ϕ_i, $i = 1, 2$, we can consider a "cascaded" prediction rule:

$$\hat{y}_2 = C_2 x_2 + D_2 u_2 \text{ where } u_2 = \hat{y}_1 = C_1 x_1 + D_1 u_1, \text{ where } x_i = \phi_i(A_i x_i + B_i u_i), \quad i = 1, 2.$$

The above rule can be represented as (2.1a), with $x = (x_2, x_1)$, $\phi((z_2, z_1)) = (\phi_2(z_2), \phi_1(z_1))$ and

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{cc|c} A_2 & B_2 C_1 & B_2 D_1 \\ 0 & A_1 & B_1 \\ \hline C_2 & D_2 C_1 & D_2 D_1 \end{array}\right).$$

Due to Theorem 2.3.3, the cascaded rule is well-posed for the componentwise map with values $\phi(z_1, z_2) = (\phi_1(z_1), \phi_2(z_2))$ if and only if each rule is.

A similar result holds if we put two or more well-posed models in parallel, and do a (weighted) sum the outputs. With the above notation, setting $\hat{y}(u_1, u_2) = \hat{y}_1(u_1) + \hat{y}_2(u_2)$ leads to a new implicit model that is also well-posed. Other possible connections include concatenation: $\hat{y}(u) = (\hat{y}_1(u), \hat{y}_2(u))$, and affine transformations (a special case of cascade connection where one of the systems has no activation). We leave the details to the reader.

In both cascade and parallel connections, the triangular structure of the matrix A of the composed system ensures that the PF sufficient condition for well-posedness is satisfied for the composed system if and only if it holds for each sub-system.

Multiplicative connections are in general are not Lipschitz-continuous, unless the inputs are bounded. Precisely, consider two activation maps ϕ_i that are Lipschitz-continuous with constant γ_i and are bounded, with $|\phi_i(v)| \leq c_i$ for every v, $i = 1, 2$; then, the multiplicative map

$$(u_1, u_2) \in \mathbb{R}^2 \to \phi(u) = \phi_1(u_1)\phi_2(u_2)$$

is Lipschitz-continuous with respect to the l_1-norm, with constant $\gamma := \max\{c_2\gamma_1, c_1\gamma_2\}$. Such connections arise in the context of attention units in neural networks, which use (bounded) activation maps such as tanh.

Finally, feedback connections are also possible. Consider two well-posed implicit systems:

$$y_i = C_i x_i + D_i u_i, \quad x_i = \phi_i(A_i x_i + B u_i), \quad i = 1, 2.$$

Now let us connect them in a feedback connection: the combined system is described by the implicit rule (2.1), where $u_1 = u + y_2$, $u_2 = y_1 = y$. The feedback system is also an implicit model of the form (2.1), with appropriate matrices (A, B, C, D), and activation map acting blockwise: $\phi(z_1, z_2) = (\phi_1(z_1), \phi_2(z_2))$ and state (x_1, x_2). In the simplified case when $D_1 = D_2 = 0$, the feedback connection has the model matrix

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{cc|c} A_1 & B_1C_2 & B_1 \\ B_2C_1 & A_2 & 0 \\ \hline C_1 & 0 & 0 \end{array}\right).$$

Note that the connection is not necessarily well-posed, even if both sub-systems are.

Scaling implicit models

Assume that the activation map ϕ is componentwise non-expansive (CONE) and positively homogeneous, as is the ReLU, or its leaky version. Consider an implicit model of the form (2.1), and assume it satisfies the PF sufficient condition for well-posedness of Theorem 2.3.1: $\lambda_{\rm pf}(|A|) \leq \kappa$, where $0 \leq \kappa < 1$ is given. Then, there is another implicit model with the same activation map ϕ, matrices (A', B', C', D'), which has the same prediction rule (i.e. $\hat{y}'(u) = \hat{y}(u), \forall u$), and satisfies $\|A'\|_\infty < 1$.

This result is a direct consequence of the following expression of the PF eigenvalue as an optimally scaled l_∞-norm, known as the Collatz-Wielandt formula, see [121, p. 666]:

$$\lambda_{\rm pf}(|A|) = \inf_S \ \|S^{-1}|A|S\|_\infty \ : \ S = \mathbf{diag}(s), \ \ s > 0. \tag{2.10}$$

The above expression implies that the condition $\lambda_{\rm pf}(|A|) < 1$ guarantees the existence of a diagonal state scaling operator S such that $\|S^{-1}|A|S\|_\infty < 1$; in fact such a scaling can be obtained via fixed-point iterations, based on the formula $s = (I - |A|)^{-1}\mathbf{1}$. The new model matrices are then $A' = S^{-1}AS$, $B' = S^{-1}B$, $C' = CS$, $D' = D$.

In a training problem, this result allows us to consider the convex constraint $\|A\|_\infty < 1$ in lieu of its Perron-Frobenius eigenvalue counterpart. This result also allows us to rescale any given implicit model, such as one derived from deep neural networks, so that the norm condition is satisfied; we will exploit this in our robustness analyses in Section 2.5.

2.4 Implicit Models of Deep Neural Networks

A large number of deep neural networks can be modeled as implicit models, including convolutional and recurrent networks; attention units; residual connections; etc. Our goal here is to show how to build a well-posed implicit model for a given neural network, assuming the activation map satisfies the componentwise non-expansiveness (CONE), or the more general block Lipschitz-continuity (BLIP) condition, as detailed in Section 2.3. Thanks to the

composition rules of Section 2.3 it suffices to model individual layers, since a neural network is just a cascade connection of such layers. The block Lipschitz structure then emerges naturally as the result of composing the layers.

We will find that the resulting models always have a strictly (block) upper triangular matrix A, which automatically implies that these models are well-posed; in fact the equilibrium equation can be simply solved via backwards substitution. In turn, models with strictly upper triangular structure also naturally satisfy the PF sufficient condition for well-posedness: for example, in the case of a componentwise non-expansive map ϕ, the matrix $|A|$ arising in Theorem 2.3.1 is also strictly upper triangular, and therefore all of its eigenvalues are zero. A similar result also holds for the case when ϕ is block Lipschitz continuous, as defined in Section 2.3.

Well-posedness

Thanks to the composition rules of Section 2.3, it suffices to model individual layers, since a neural network is just a cascade connection of such layers. The block-Lipschitz structure of the activation map, and the strictly triangular structure of the matrix A, then emerge naturally as the result of composing the layers in a "cascade" fashion. This implies that the implicit models we obtain are well-posed, as the corresponding Perron-Frobenius eigenvalue of the matrix $N(A, \gamma)$ defined in (2.7) is zero, since $N(A, \gamma)$ is then strictly triangular.

We may always assume that the resulting implicit model satisfies the stronger norm condition for well-posedness mentioned in Remark 2.3.2. For example, in the case of a CONE map ϕ, the stronger condition $\|A\|_\infty < 1$ can always be obtained by appropriately scaling the weight matrices of the network's layers, and using a scaled version for the state vector x.

Basic operations

Activation at the output It is common to have an activation at the output level, as in the rule

$$\hat{y}(u) = \phi_o(Cx + Du), \quad x = \phi(Ax + Bu), \tag{2.11}$$

with ϕ_o the output activation function. We can represent this rule as in (2.1), by introducing a new state variable: with $\tilde{\phi} := (\phi_o, \phi)$,

$$\begin{pmatrix} z \\ x \end{pmatrix} = \tilde{\phi}\left(\begin{pmatrix} 0 & C \\ 0 & A \end{pmatrix}\begin{pmatrix} z \\ x \end{pmatrix} + \begin{pmatrix} D \\ B \end{pmatrix} u\right), \quad y = z.$$

The rule is well-posed with respect to $\tilde{\phi}$ if and only if the matrix A is, with respect to ϕ.

Bias terms and affine rules The affine rule

$$y = Cx + Du + d, \quad x = \phi(Ax + Bu + b),$$

where $d \in \mathbb{R}^q$, $b \in \mathbb{R}^n$, is handled by simply appending a 1 at the end of the input vector u.

A related transformation is useful with activation functions ϕ, such as the sigmoid, that do not satisfy $\phi(0) = 0$. Consider the rule

$$y = Cx + Du, \;\; x = \phi(Ax + Bu).$$

Defining $\tilde{\phi}(\cdot) := \phi(\cdot) - \phi(0)$ and using the state vector $\tilde{x} := x + \phi(0)$, we may represent the above rule as

$$y = C\tilde{x} + \begin{pmatrix} D & -C\phi(0) \end{pmatrix} \begin{pmatrix} u \\ 1 \end{pmatrix}, \;\; \tilde{x} = \tilde{\phi}(A\tilde{x} + \begin{pmatrix} B & -A\phi(0) \end{pmatrix} \begin{pmatrix} u \\ 1 \end{pmatrix}).$$

Again, the rule is well-posed if and only if the matrix A is.

Batch normalization Batch normalization consists in including in the prediction rule a normalization step using some estimates of input mean $\hat{u}$ and variance $\sigma > 0$ that are based on a batch of training data. The parameters $\hat{u}, \sigma$ are given at test time. This step is a simple affine rule:

$$y = Du + d, \text{ where } [D, d] := \mathbf{diag}(\sigma)^{-1}[I, -\hat{u}].$$

Fully connected feedforward neural networks

Consider the following prediction rule, with $L > 1$ fully connected layers:

$$\hat{y}(u) = W_L x_L, \;\; x_{l+1} = \phi_l(W_l x_l), \;\; x_0 = u. \tag{2.12}$$

Here $W_l \in \mathbb{R}^{n_{l+1} \times n_l}$ and $\phi_l : \mathbb{R}^{n_{l+1}} \to \mathbb{R}^{n_{l+1}}$, $l = 1, \ldots, L$, are given weight matrices and activation maps, respectively. We can express the above rule as (2.1a), with $x = (x_L, \ldots, x_1)$, and

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{cccc|c} 0 & W_{L-1} & \ldots & 0 & 0 \\ & 0 & \ddots & \vdots & \vdots \\ & & \ddots & W_1 & 0 \\ & & & 0 & W_0 \\ \hline W_L & 0 & \ldots & 0 & 0 \end{array}\right), \tag{2.13}$$

and with an appropriately defined blockwise activation function ϕ, defined as operating on an n-vector $z = (z_L, \ldots, z_1)$ as $\phi(z) = (\phi_L(z_L), \ldots, \phi_1(z_1))$. Due to the strictly upper triangular structure of A, the system is well-posed, irrespective of A; in fact the equilibrium equation $x = \phi(Ax + Bu)$ is easily solved via backward block substitution, which corresponds to a simple forward pass through the network.

Convolutional layers and max-pooling

A single convolutional layer can be represented as a linear map: $y = Du$, where u is the input, D is a matrix that represents the (linear) convolution operator, with a "constant-along-diagonals", Toeplitz-like structure. For example a 2D convolution with a 2D kernel K takes a 3×3 matrix U and produces a 2×2 matrix Y. With

$$U = \begin{pmatrix} u_1 & u_2 & u_3 \\ u_4 & u_5 & u_6 \\ u_7 & u_8 & u_9 \end{pmatrix}, \quad K = \begin{pmatrix} k_1 & k_2 \\ k_3 & k_4 \end{pmatrix},$$

the convolution can be represented as $y = Du$, with y, u vectors formed by stacking the rows of U, Y together, and

$$D = \begin{pmatrix} k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 \\ 0 & 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 \end{pmatrix}.$$

Often, a convolutional layer is combined with a max-pooling operation. The latter forms a

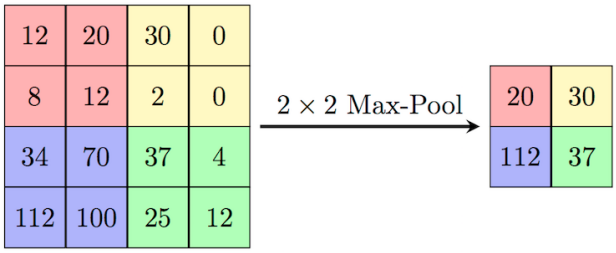


Figure 2.3: A max-pooling operation: the smaller image contains the maximal pixel values of each colored area.

down-sample of an image, which is a smaller image that contains the largest pixel values of specific sub-areas of the original image. Such an operation can be represented as

$$y_j = \max_{1 \le i \le h} (B_j u)_i, \quad j \in [q].$$

In the above, $p = qh$, and the matrices $B_j \in \mathbb{R}^{h \times p}$, $j \in [q]$, are used to select specific sub-areas of the original image. In the example of Figure 2.3, the number of pixels selected in each area is $h = 4$, the output dimension is $q = 4$, that of the input is $p = qh = 16$; vectorizing images row by row:

$$\left(\begin{array}{c} B_1 \\ \hline B_2 \\ \hline B_3 \\ \hline B_4 \end{array}\right) = \mathbf{diag}(M, M) \in \mathbb{R}^{16 \times 16}, \quad M := \begin{pmatrix} I_2 & 0 & 0 & 0 \\ 0 & 0 & I_2 & 0 \\ 0 & I_2 & 0 & 0 \\ 0 & 0 & 0 & I_2 \end{pmatrix} \in \mathbb{R}^{8 \times 8},$$

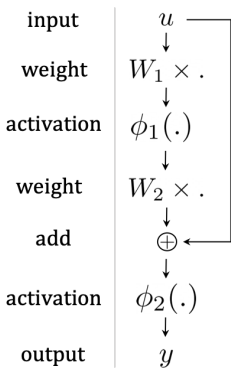


Figure 2.4: Building block of residual networks.

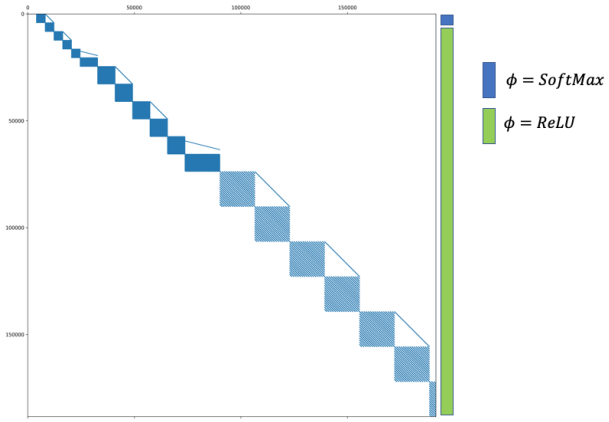


Figure 2.5: The matrix A for a 20-layer residual network. Nonzero elements are colored in blue.

where I_2 is the 2×2 identity matrix.

Define the mapping $\phi : \mathbb{R}^n \to \mathbb{R}^n$, where $n = p$, as follows. For a p-vector z decomposed in q blocks $(z_1, \ldots, z_q)$, we set $\phi(z_1, \ldots, z_q) = (\max(z_1), \ldots, \max(z_q), 0, \ldots, 0)$. (The padded zeroes are necessary in order to make sure the input and output dimensions of ϕ are the same). We obtain the implicit model

$$y = C\phi(Bu) = Cx, \text{ where } x := \phi(Bu).$$

Here C is used to select the top q elements,

$$C = \begin{pmatrix} I_q & 0 & \ldots & 0 \end{pmatrix}, \quad B := \begin{pmatrix} B_1^\top & \ldots & B_q^\top \end{pmatrix}^\top.$$

The Lipschitz constant of the max-pooling activation map ϕ, with respect to the l_∞-norm, is 1, hence it verifies the BLIP condition of Section 2.3.

Residual nets

A building block in residual networks involves the relationship illustrated in Figure 2.4. Mathematically, a residual block combines two linear operations, with non-linearities in the middle and the end, and adds the input to the resulting output:

$$y = \phi_2(u + W_2\phi_1(W_1 u)),$$

where W_1, W_2 are weight matrices of appropriate size. The above is a special case of the implicit model (2.1): defining the blockwise map $\phi(z_2, z_1) = (\phi_2(z_2), \phi_1(z_1))$,

$$\begin{pmatrix} x_2 \\ x_1 \end{pmatrix} = \phi\left(\begin{pmatrix} 0 & W_2 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} x_2 \\ x_1 \end{pmatrix} + \begin{pmatrix} I \\ W_1 \end{pmatrix} u\right), \quad y = x_2.$$

Figure 2.5 displays the model matrix A for a 20-layer residual network. Convolutional layers appear as matrix blocks with Toeplitz (constant along diagonal) structure; residual unit correspond to the straight lines on top of the blocks. The network only uses the ReLU map, except for the last layer, which uses a softmax map.

Recurrent units

Recurrent neural nets (RNNs) can be represented in an "unrolled" form as shown in [85], which is the perspective we will consider here. The input to an RNN block is a sequence of vectors $\{u_1, \cdots, u_T\}$, where for every $t \in [T]$, $u_t \in \mathbb{R}^p$. At each time step, the network takes in a input u_t and the previous hidden state h_{t-1} to produce the next hidden state h_t; the hidden state h_t defines the state space or "memory" of the network. The rule can be written as,

$$h_t = \phi_H(W_H h_{t-1} + W_I u_t), \;\; y_t = \phi_O(W_O h_t), \;\; t = 1, \ldots, T, \tag{2.14}$$

Then, equations (2.14) can be expressed as an implicit model (2.11), with $x = (h_T, \ldots, h_0)$, $u = (u_T, \ldots, u_1)$, and weight matrices

$$\left(\begin{array}{c|c} A & B \\ \hline C & 0 \end{array}\right) = \left(\begin{array}{ccccc|cccc} 0 & W_H & \cdots & 0 & 0 & W_I & \cdots & 0 & 0 \\ 0 & 0 & W_H & \vdots & \vdots & 0 & W_I & \vdots & \vdots \\ & & \ddots & \ddots & 0 & & \ddots & \ddots & 0 \\ & & & 0 & W_H & & & 0 & W_I \\ \hline W_O & 0 & \cdots & 0 & 0 & 0 & \cdots & \cdots & 0 \end{array}\right) \tag{2.15}$$

where A, B are strictly upper block triangular and share the same block diagonal submatrices, W_H and W_I respectively, shared across all hidden states and inputs.

Note that the above approach leads to matrices A, B, C, D with a special structure; during training and test time, it is possible to exploit that structure in order to avoid "unrolling" the recurrent layers.

Multiplicative units: LSTM, attention mechanisms and variants

Some deep learning network architectures use multiplicative units. As mentioned in Section 2.3, multiplication between variables in not Lipschitz-continuous in general, unless the inputs are bounded. Luckily, most of the multiplicative units used in practice involve bounded inputs.

In the case of Long Short-Term Memory (LSTM) [189] or gated recurrent units (GRUs) [41], a basic building block involves the *product* of two input variables after each one passes through a bounded non-linearity. Precisely, the output takes the form

$$y = \phi(u) := \phi_1(u_1)\phi_2(u_2),$$

where $u_1, u_2 \in \mathbb{R}$, and ϕ_1, ϕ_2 are both *bounded* (scalar) non-linearities.

Attention models [14] use componentwise vector multiplication, usually involving a softmax operation (2.5):

$$y = \phi(u) = \phi_1(u_1) \odot \mathrm{SoftMax}(u_2),$$

where ϕ_1 is a bounded, componentwise Lipschitz-continuous activation map, such as the sigmoid; and W is a matrix of weights. The map ϕ is then Lipschitz-continuous with respect to the l_1-norm. As shown in Section 2.3, we can formulate these multiplicative units as well-posed implicit models.

2.5 Robustness

In this section, our goal is to analyze the robustness properties of a given implicit model (2.1a). We seek to bound the state, output and loss function, under unknown-but-bounded inputs. This robustness analysis task is of interest in itself for diagnosis or for generating adversarial attacks. It will also inform our choice of appropriate penalties or constraints in the training problem. We assume that the activation map is BLIP, and that the matrix A of the implicit model satisfies the sufficient conditions for well-posedness outlined in Theorem 2.3.1.

Input uncertainty models

We assume that the input vector is uncertain, and only known to belong to a given set $\mathcal{U} \subseteq \mathbb{R}^p$. Our results apply to a wide variety of sets $\mathcal{U}$; we will focus on the following two cases.

A first case corresponds to a box:

$$\mathcal{U}^{\mathrm{box}} := \left\{ u \in \mathbb{R}^p \ : \ |u - u^0| \leq \sigma_u \right\}. \tag{2.16}$$

Here, the p-vector $\sigma_u > 0$ is a measure of componentwise uncertainty affecting each data point, while u^0 corresponds to a vector of "nominal" inputs. The following variant limits the number of changes in the vector u:

$$\mathcal{U}^{\mathrm{card}} := \left\{ u \in \mathbb{R}^p \ : \ |u - u^0| \leq \sigma_u, \ \ \mathbf{Card}(u - u^0) \leq k \right\}, \tag{2.17}$$

where **Card** denotes the cardinality (number of non-zero components) in its vector argument, and $k < p$ is a given integer.

Box bounds on the state vector

Assume first that ϕ is a CONE map, and that the input is only known to belong to the box set $\mathcal{U}^{\mathrm{box}}$ (2.16). We seek to find componentwise bounds on the state vector x, of the form

$|x - x^0| \leq \sigma_x$, with x and x^0 the unique solution to $\xi = \phi(A\xi + Bu)$ and $\xi = \phi(A\xi + Bu^0)$ respectively, and $\sigma_x > 0$. We have

$$|x - x^0| = |\phi(Ax + Bu) - \phi(Ax^0 + Bu^0)| \leq |A||x - x^0| + |B(u - u^0)|,$$

which shows in particular that

$$\|x - x^0\|_\infty \leq \|A\|_\infty \|x - x^0\|_\infty + \|B(u - u^0)\|_\infty,$$

hence, provided $\|A\|_\infty < 1$, we have:

$$\|x - x^0\|_\infty \leq \frac{\||B|\sigma_u\|_\infty}{1 - \|A\|_\infty}. \tag{2.18}$$

With the cardinality constrained uncertainty set $\mathcal{U}^{\text{card}}$ (2.17), we obtain

$$\|x - x^0\|_\infty \leq \frac{\delta}{1 - \|A\|_\infty}, \quad \delta := \max_{1 \leq i \leq n} s_k(\sigma_u \odot |B|^\top e_i), \tag{2.19}$$

with e_i the i-th unit vector in $\mathbb{R}^n$, and s_k the sum of the top k entries in a vector.

We may refine the analysis above, with a "box" (componentwise) bound. When ϕ is block-Lipschitz (BLIP) map, our result involves the matrix of norms $N(A, \gamma)$ defined in (2.7), as well as a similar matrix defined for the input matrix B: decomposing B into blocks $B = (B_{li})_{l \in [L], i \in [p]}$, with $B_{li} \in \mathbb{R}^{n_l}$, $l \in [L]$, we define the $L \times p$ matrix of norms

$$N(B, \gamma) := \gamma_l(\|B_{li}\|_{p_l})_{l \in [L],\, i \in [p]}. \tag{2.20}$$

Theorem 2.5.1 (Box bound on the vector norms of the state, BLIP map)**.** *Assuming that ϕ is BLIP, and the corresponding sufficient well-posedness condition of Theorem 2.3.1 is satisfied. Then, $I - N(A, \gamma)$ is invertible, and*

$$\eta(x - x^0) \leq (I - N(A, \gamma))^{-1} N(B, \gamma)\sigma_u, \tag{2.21}$$

where the vector of norms function $\eta(\cdot)$ is defined in (2.2).

Proof. For every $l \in [L]$:

$$\begin{aligned}
[\eta(x - x^0)]_l &\leq \|[\phi(A(x - x^0) + B(u - u^0)b)]_l\|_{p_l} \\
&\leq \gamma_l \| \sum_{h \in [L]} A_{lh}(x - x^0)_h \|_{p_l} + \gamma_l \| \sum_{i \in [p]} B_{li}(u - u^0)_i \|_{p_l} \\
&\leq \gamma_l \sum_{h \in [L]} \|A_{lh}\|_{p_h \to p_l} \eta(x - x^0)_h + \gamma_l \sum_{i \in [p]} \|B_{li}\|_{p_l} |u - u^0|_i \\
&\leq [N(A, \gamma)\eta(x - x^0)]_l + [N(B, \gamma)|u - u^0|]_l,
\end{aligned}$$

which establishes the desired bound.

■

Note that the box bound can be computed via fixed-point iterations. For example when ϕ is a CONE map, we solve $(I-|A|)\sigma_x = |B|\sigma_u$, as the limit point of the fixed-point iteration

$$\sigma_x(0) = 0, \;\; \sigma_x(t+1) = |A|\sigma_x(t) + |B|\sigma_u, \;\; t = 0, 1, 2, \ldots,$$

which converges since $\lambda_{\rm pf}(|A|) < 1$.

Bounds on the output and the sensitivity matrix

The above allows us to analyze the effect of effect of input noise on the output vector y. Let us assume that the function ϕ satisfies the CONE (componentwise non-expansiveness) condition (2.4). In addition, we assume that the stronger condition for well-posedness, $\|A\|_\infty < 1$, is satisfied. (As noted in Section 2.3, we can always rescale the model so as to ensure that property, provided $\lambda_{\rm pf}(|A|) < 1$.) For the implicit prediction rule (2.1), we then have

$$\forall\, u, u^0 \;\; : \;\; \|\hat{y}(u) - \hat{y}(u^0)\|_\infty \leq \rho \|u - u^0\|_\infty, \;\; \text{where } \rho := \frac{\|B\|_\infty \|C\|_\infty}{1 - \|A\|_\infty} + \|D\|_\infty.$$

The above shows that the prediction rule is Lipschitz-continuous, with a constant bounded above by ρ. This result motivates the use of the $\|\cdot\|_\infty$ norm as a penalty on model matrices A, B, C, D, for example via a convex penalty that bounds the Lipschitz constant above:

$$\rho \leq \frac{1}{2} \frac{\|B\|_\infty^2 + \|C\|_\infty^2}{1 - \|A\|_\infty} + \|D\|_\infty. \tag{2.22}$$

We can refine this analysis with the following theorem, applicable to the case when ϕ is block Lipschitz (BLIP). Decomposing C into column blocks $C = (C_1, \ldots, C_L)$, with $C_l \in \mathbb{R}^{q \times n_l}$, $l \in [L]$, we define the matrix of (dual) norms

$$N(C) := (\|C_{il}\|_{p_l^*})_{l \in [L],\, i \in [q]},$$

where $p_l^* := 1/(1 - 1/p_l)$, $l \in [L]$. Also recall the corresponding matrix of norms related to A in (2.7) and B in (2.20).

Theorem 2.5.2 (Box bound on the output, BLIP map)**.** *Assuming that ϕ is a BLIP map, and that the sufficient condition for well-posedness $\lambda_{pf}(N(A, \gamma)) < 1$ is satisfied. Then, $I - N(A, \gamma)$ is invertible, and*

$$\forall\, u, u^0 \;\; : \;\; |\hat{y}(u) - \hat{y}(u^0)| \leq S|u - u^0|, \tag{2.23}$$

where the (non-negative) $q \times p$ matrix

$$S := N(C)(I - N(A, \gamma))^{-1} N(B, \gamma) + |D|$$

is a "sensitivity matrix" of the implicit model with a BLIP map. In the case of a CONE map, the sensitivity matrix is given by

$$S = |C|(I - |A|)^{-1}|B| + |D|.$$

Proof. For given $i \in [q]$, we have

$$\begin{aligned}|\hat{y}(u)-\hat{y}(u^0)|_i \\ \leq |\sum_{l\in[L]} C_{il}(x-x^0)_l| + (|D||u-u^0|)_i \leq \sum_{l\in[L]} \|C_{il}\|_{p_l^*}\eta(x-x^0)_l + (|D||u-u^0|)_i.\end{aligned}$$

■

The sensitivity matrix can be computed via fixed-point iterations that are guaranteed to converge, thanks to well-posedness assumption in the theorem. In the case of CONE maps, the iterations involve finding the limit point X_∞ of

$$X(t+1) = |A|X(t) + |B|, \;\; t = 0, 1, 2, \ldots,$$

and setting $S = |C|X_\infty + |D|$. Our refined analysis suggests a "natural" penalty to use during the training phase in order to improve robustness, namely $\|S\|_\infty$.

Linear Programming (LP) relaxation for CONE maps

The previous bounds does not provide a direct way to generate an adversarial attack, that is, a feasible point $u \in \mathcal{U}$ that has maximum impact on the state. In some cases it may be possible to refine our previous box bounds via an LP relaxation, which has the advantage of suggesting a specific adversarial attack. Here we restrict our attention to the ReLU activation: $\phi(z) = \max(z, 0) = z_+$, which is a CONE map.

We consider the problem

$$p^* := \max_{x,u\in\mathcal{U}} \sum_{i\in[n]} f_i(x_i) \;:\; x = z_+, \;\; z = Ax + Bu, \;\; |x - x^0| \leq \sigma_x, \tag{2.24}$$

where f_i's are arbitrary functions. Setting $f_i(\xi) = (\xi - x_i^0)^2$, $i \in [n]$, leads to the problem of finding the largest discrepancy (measured in l_2-norm) between x and x^0; setting $f_i(\xi) = -\xi$, $i \in [n]$, finds the minimum l_1-norm state. By construction, our bound can only improve on the previous state bound, in the sense that

$$p^* \leq \sum_{i\in[n]} \max_{|\alpha|\leq 1} \; f_i(x_i^0 + \alpha\sigma_{x,i}).$$

Our result is expressed for general sets $\mathcal{U}$, based on the support function $\sigma_\mathcal{U}$ with values for $b \in \mathbb{R}^p$ given by

$$\sigma_\mathcal{U}(b) := \max_{u\in\mathcal{U}} \; b^\top u. \tag{2.25}$$

Note that this function depends only on the convex hull of the set $\mathcal{U}$. In the case of the box set $\mathcal{U}^{\text{box}}$ given in (2.16), there is a convenient closed-form expression:

$$\sigma_{\mathcal{U}^{\text{box}}}(b) = b^\top u^0 + \sigma_u^\top |b|.$$

Likewise for the cardinality-constrained set $\mathcal{U}^{\text{card}}$ defined in (2.17), we have

$$s_{\mathcal{U}_k}(b) = \max_{u \in \mathcal{U}^{\text{card}}} b^\top u = b^\top u^0 + s_k(\sigma_u \odot |b|),$$

where s_k is the sum of the top k entries of its vector argument — a convex function.

The only coupling constraint in (2.24) is the affine equation, which suggests the following relaxation.

Theorem 2.5.3 (LP bound on the state)**.** *An upper bound on the objective of problem (2.24) is given by*

$$p^* \leq \overline{p} := \min_{\lambda} \ \sigma_{\mathcal{U}}(B^\top \lambda) + \sum_{i \in [n]} g_i(\lambda_i, (A^\top \lambda)_i),$$

where $s_{\mathcal{U}}$ is the support function defined in (2.25), and g_i, $i \in [n]$, are the convex functions

$$g_i \ : \ (\alpha, \beta) \in \mathbb{R}^2 \to g_i(\alpha, \beta) := \max_{\zeta \, : \, |\zeta_+ - x_i^0| \leq \sigma_{x,i}} \ f_i(\zeta_+) - \alpha\zeta + \beta\zeta_+, \ \ i \in [n].$$

If the functions g_i, $i \in [n]$ are closed, we have the bidual expression

$$\overline{p} := \max_{x, \, u \in \mathcal{U}} \ -\sum_{i \in [n]} g_i^*(-(Ax + Bu)_i, x_i),$$

where g_i^* is the conjuguate of g_i, $i \in [n]$.

Proof. We have

$$\begin{aligned}
p^* \leq \overline{p} \ &:= \ \min_{\lambda} \ \max_{x, \, u \in \mathcal{U}} \ \sum_{i \in [n]} f_i(x_i) + \lambda^\top (Ax + Bu - z) \ : \ x = z_+, \ \ |x - x^0| \leq \sigma_x \\
&= \ \min_{\lambda} \ \max_{u \in \mathcal{U}} \ \lambda^\top Bu + \max_{z \, : \, |z^+ - x^0| \leq \sigma_x} \ \sum_{i \in [n]} (f_i(z_i^+) + (A^\top \lambda)_i z_i^+ - \lambda_i z_i) \\
&= \ \min_{\lambda, \, \mu = A^\top \lambda} \ \left(\max_{u \in \mathcal{U}} \ \lambda^\top Bu \right) + \sum_{i \in [n]} g_i(\lambda_i, \mu_i),
\end{aligned}$$

which establishes the first part of the theorem. If we further assume that the functions g_i, $i \in [n]$ are closed, strong duality holds, so that

$$\overline{p} \ = \ \min_{\lambda, \, \mu} \ \max_{x, \, u \in \mathcal{U}} \ \lambda^\top Bu + x^\top (A^\top \lambda - \mu) + \sum_{i \in [n]} g_i(\lambda_i, \mu_i) = \max_{x, \, u \in \mathcal{U}} \ -\sum_{i \in [n]} g_i^*(-(Ax + Bu)_i, x_i)$$

where g_i^* is the conjugate of g_i, $i \in [n]$. ∎

In the case when$f_i(\xi) = c_i \xi$, $i \in [n]$, where $c \in \mathbb{R}^n$ is given, it turns out that our relaxation, when expressed in bidual form, has a natural look:

$$p^* \ \leq \ \overline{p} = \max_{x, \, u \in \mathcal{U}} \ c^\top x \ : \ x \geq Ax + Bu, \ \ x \geq 0, \ \ |x - x^0| \leq \sigma_x.$$

When the cardinality of changes in the input is constrained in the set $\mathcal{U}^{\text{card}}$, the bound takes the form

$$p^* \;\leq\; \overline{p} = \max_{x,\,u} \; c^\top x \;:\quad x \geq Ax + Bu,\;\; x \geq 0,\;\; |x - x^0| \leq \sigma_x, \\ \|\,\mathbf{diag}(\sigma_u)^{-1}(u - u^0)\|_1 \leq k,\;\; |u - u^0| \leq \sigma_u.$$

2.6 Sparsity and Model Compression

In this section, we examine the role of sparsity and low-rank structure in implicit deep learning, specifically in the model parameter matrix

$$M := \begin{pmatrix} A & B \\ C & D \end{pmatrix}.$$

We assume that the activation map ϕ is a CONE map; most of our results can be generalized to BLIP maps.

Since a CONE map acts componentwise, the prediction rule (2.1a) is invariant under permutations of the state vector, in the sense that, for any $n \times n$ permutation matrix, the matrix $\mathbf{diag}(P, I)M\,\mathbf{diag}(P^\top, I)$ represents the same prediction rule as M given above.

Various kinds of sparsity of M can be encouraged in the training problem, with appropriate penalties. For example, we can use penalties that encourage many elements in M to be zero; the advantage of such "element-wise" sparsity is, of course, computational, since sparsity in matrices A, B, C, D will allow for computational speedups at test time. Another interesting kind of sparsity is rank sparsity, which refers to the case when model matrices are low-rank.

Next, we examine the benefits of row- (or, column-) sparsity, which refers to the fact that entire rows (or, columns) of a matrix are zero. Note that column sparsity in a matrix N can be encouraged with a penalty in the training problem, of the form

$$\mathcal{P}(N) = \sum_i \|Ne_i\|_\alpha$$

where $\alpha > 1$. Row sparsity can be handled via $\mathcal{P}(N^\top)$.

Feature selection

We may use the implicit model to select features. Any zero column in the matrix $(B^\top, D^\top)^\top$ means that the corresponding element in an input vector does not play any role in the prediction rule. We may thus use a column-norm penalty in the training problem, in order to encourage such a sparsity pattern:

$$\mathcal{P}(B, D) = \sum_{j=1}^{p} \left\| \begin{pmatrix} B \\ D \end{pmatrix} e_j \right\|_\alpha, \tag{2.26}$$

with $\alpha > 1$.

Dimension reduction via row- and column-sparsity

Assume that the matrix A is row-sparse. Without loss of generality, using permutation invariance, we can assume that M writes

$$M = \begin{pmatrix} A_{11} & A_{12} & B_1 \\ 0 & 0 & B_2 \\ C_1 & C_2 & D \end{pmatrix},$$

where A_{11} is square of order $n_1 < n$. We can then decompose x accordingly, as $x = (x_1, x_2)$ with $x_1 \in \mathbb{R}^{n_1}$, and the above implies $x_2 = \phi(B_2 u)$. The prediction rule for an input $u \in \mathbb{R}^p$ then writes

$$\hat{y}(u) = C_1 x_1 + Du, \;\; x_1 = \phi(A_{11} x_1 + A_{12}\phi(B_2 u) + B_1 u).$$

The rule only involves x_1 as a true hidden feature vector. In fact, the row sparsity of A allows for a computational speedup, as we simply need to solve a fixed-point equation for the vector with reduced dimensions, x_1.

Further assume that (A, B) is row-sparse. Again without loss of generality we may put M in the above form, with $B_2 = 0$. Then the prediction rule can be written

$$\hat{y}(u) = C_1 x_1 + Du, \;\; x_1 = \phi(A_{11} x_1 + B_1 u).$$

This means that the dimension of the state variable can be fully reduced, to $n_1 < n$. Thus, row sparsity of (A, B) allows for a reduction in the dimension of the prediction rule.

When (A, B) is column-sparse, we obtain similar speedups: we only need to solve for x_1, and x_2 can be directly expressed as closed-form function of x_1. We leave the details to the reader.

Now assume that $(A^\top, C^\top)^\top$ is column-sparse. Again, the prediction rule does not need x_2 at all, so that the computation of the latter vector can be entirely avoided. This means that the dimension of the state variable can be fully reduced, to $n_1 < n$. Thus, column sparsity of $(A^\top, C^\top)^\top$ allows for a reduction in the dimension of the prediction rule.

To summarize, row or column sparsity of A allows for a computational speedup; if the corresponding rows of B (resp. columns of C) are zero, then the prediction rule involves only a vector of reduced dimensions.

Rank sparsity

Assume that the matrix A is rank $k \ll n$, and that a corresponding factorization is known: $A = LR^\top$, with $L, R \in \mathbb{R}^{n \times k}$. In this case, for any p-vector u, the equilibrium equation $x = \phi(Ax + Bu)$ can be written as $x = \phi(Lz + Bu)$, where $z := R^\top x$. Hence, we can obtain a prediction for a given input u via the solution of a low-dimensional fixed-point equation in $z \in \mathbb{R}^k$:

$$z = R^\top \phi(Lz + Bu).$$

It can be shown that, when ϕ is a CONE map, the above rule is well-posed if $\lambda_{\rm pf}(|R|^T|L|) < 1$. Once a solution z is found, we simply set the prediction to be $\hat{y}(u) = C\phi(Lz + Bu) + Du$.

At test time, if we use fixed-point iterations to obtain our predictions, then the computational savings brought about by the low-rank representation of A can be substantial, with a per-iteration cost going from $O(n^2)$, to $O(kn)$ if we use the above.

Model error analysis

The above suggests to replace a given model matrix A with a low-rank or sparse approximation, denoted A^0. The resulting state error can be then bounded, as follows. Assume that $|A - A^0| \leq E$, where $E \geq 0$ is a known upper bound on the componentwise error in A. The following theorem provides *relative* error bounds on the state, provided the perturbed system satisfies the well-posedness condition $\lambda_{\rm pf}(|A|) < 1$. As before, we denote by x^0, x the (unique) solutions to the unperturbed and perturbed equilibrium equations $\xi = \phi(A^0\xi + Bu)$ and $\xi = \phi(A\xi + Bu)$, respectively.

Theorem 2.6.1 (Effect of errors in A)**.** *Assuming that ϕ is a CONE map, and that $\lambda_{pf}(|A^0 + E|) < 1$. Then:*

$$|x - x^0| \leq (I - (|A^0 + E|))^{-1} E x_0. \tag{2.27}$$

Proof. We have

$$|x - x^0| \leq |Ax - Ax^0| = |A^0(x - x^0) + E(x - x^0) + Ex^0| \ \leq |A^0 + E||x - x^0| + |E||x^0|.$$

Applying a technique similar to that employed in the proof of Theorem 2.5.1, we obtain the desired relative error bounds. ∎

2.7 Training Implicit Models

Setup

We are now given an input data matrix $U = [u_1, \ldots, u_m] \in \mathbb{R}^{p\times m}$ and response matrix $Y = [y_1, \ldots, y_m] \in \mathbb{R}^{q\times m}$, and seek to fit a model of the form (2.1a), with A well-posed with respect to ϕ, which we assume to be a BLIP map. We note that the rule (2.1a), when applied to a collection of inputs $(u_i)_{1\leq i\leq m}$, can be written in matrix form, as

$$\hat{Y}(U) = CX + DU, \text{ where } X = \phi(AX + BU).$$

where $U = [u_1, \ldots, u_m] \in \mathbb{R}^{p\times m}$, and $\hat{Y}(U) = [\hat{y}(u_1), \ldots, \hat{y}(u_m)] \in \mathbb{R}^{q\times m}$.

We consider a training problem of the form

$$\min_{A,B,C,D,X} \mathcal{L}(Y, CX + DU) + \mathcal{P}(A, B, C, D) \ : \ X = \phi(AX + BU), \ \ A \in \mathrm{WP}(\phi). \tag{2.28}$$

In the above, $\mathcal{L}$ is a loss function, assumed to be convex in its second argument, and $\mathcal{P}$ is a convex penalty function, which can be used to enforce a given (linear) structure (such as, A strictly upper block triangular) on the parameters, and/or encourage their sparsity.

In practice, we replace the well-posedness condition by the sufficient PF condition of Theorem 2.3.1. As argued in Section 2.3, the latter can be further replaced without loss of generality with an easier-to-handle (convex) norm constraint; in the case of CONE maps, this condition is $\|A\|_\infty \leq \kappa$, where $\kappa \in (0,1)$ is a hyper-parameter. In the more general case of BLIP maps, we can use a similar norm constraint $\|N(A,\gamma)\|_\infty \leq \kappa$, where $N(A,\gamma)$ is defined in (2.7).

Examples of loss functions For regression tasks, we may use the squared Euclidean loss: for $Y, \hat{Y} \in \mathbb{R}^{q\times m}$,

$$\mathcal{L}(Y,\hat{Y}) := \frac{1}{2}\|Y-\hat{Y}\|_F^2.$$

For multi-class classification, a popular loss is a combination of negative cross-entropy with the soft-max: for two q-vectors $y, \hat{y}$, with $y \geq 0$, $y^\top \mathbf{1} = 1$, we define

$$\mathcal{L}(y,\hat{y}) = -y^\top \log\left(\frac{e^{\hat{y}}}{\sum_{i=1}^q e^{\hat{y}_i}}\right) = \log(\sum_{i=1}^q e^{\hat{y}_i}) - y^\top \hat{y}.$$

We can extend the definition to matrices, by summing the contribution to all columns, each corresponding to a data point: for $Y, Z \in \mathbb{R}^{q\times m}$,

$$\mathcal{L}(Y,\hat{Y}) = \sum_{j=1}^m \log\left(\sum_{i=1}^q e^{\hat{Y}_{ij}}\right) - \sum_{j=1}^m\sum_{i=1}^q Y_{ij}\hat{Y}_{ij} = \log(\mathbf{1}^\top \exp(\hat{Y}))\mathbf{1} - \mathbf{Tr}\, Y^\top \hat{Y}, \tag{2.29}$$

where both the log and the exponential functions apply componentwise.

Examples of penalty functions Beyond well-posedness, the penalty can be used to encourage desired properties of the model. For robustness, the convex penalty (2.22) can be used. We may also use an l_∞-norm penalty on the sensitivity matrices S appearing in Theorem 2.5.2. For feature selection, an appropriate penalty may involve the block norms (2.26); sparsity of the model matrices can be similarly handled with ordinary l_1-norm penalties on the elements of the model matrices (A,B,C,D).

Gradient Methods

Assuming that ϕ is a CONE map for simplicity, we consider the problem

$$\min_{A,B,C,D,X} \mathcal{L}(Y, CX+DU) + \mathcal{P}(A,B,C,D) \;:\; X = \phi(AX+BU),\;\; \|A\|_\infty \leq \kappa, \tag{2.30}$$

where $\kappa < 1$ is given. We assume that the map ϕ is differentiable. We can solve (2.30) using (stochastic) projected gradient descent, by differentiating through the equilibrium equation.

It turns out that this differentiation requires solving an equilibrium equation involving a matrix variable, which, thanks to well-posedness, can be very efficiently solved via fixed-point iterations.

Computing gradients Considering a mini-batch of size 1 first, we define $\hat{y} = Cx + Du$, $z = Ax + Bu$. We have

$$\begin{pmatrix} \nabla_A \mathcal{L} & \nabla_B \mathcal{L} \\ \nabla_C \mathcal{L} & \nabla_D \mathcal{L} \end{pmatrix} = \begin{pmatrix} \nabla_z \mathcal{L} \\ \nabla_{\hat{y}} \mathcal{L} \end{pmatrix} \begin{pmatrix} x \\ u \end{pmatrix}^\top,$$

where $\nabla_{\hat{y}} \mathcal{L}$ is easy to compute, and $\nabla_z \mathcal{L}$ is obtained via implicit differentiation:

$$\nabla_z \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial x} \cdot \frac{\partial x}{\partial z} \right)^\top, \quad \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial (Cx + Du)}{\partial x},$$
$$\frac{\partial x}{\partial z} = \frac{\partial \phi(z)}{\partial z} + \frac{\partial \phi(Ax + Bu)}{\partial x} \cdot \frac{\partial x}{\partial z} = (I - \Phi A)^{-1} \Phi,$$

where $\Phi := \frac{\partial \phi(z)}{\partial z}$ is a diagonal matrix. Since ϕ is a CONE map, we have $\|\Phi\|_\infty \leq 1$; since the current matrix A satisfies the norm condition $\|A\|_\infty < 1$, the inverse of the matrix $(I - \Phi A)$ exists. The gradient of the loss function $\nabla_{\hat{y}} \mathcal{L}$ can be easily computed, and we have

$$\nabla_z \mathcal{L} = \left(C(I - \Phi A)^{-1} \Phi \right)^\top \nabla_{\hat{y}} \mathcal{L}. \tag{2.31}$$

Thanks to well-posedness, the gradient $\nabla_z \mathcal{L}$ is the unique solution to the following equilibrium equation in vector v:

$$v = \Phi \left(A^\top v + C^\top \nabla_{\hat{y}} \mathcal{L} \right). \tag{2.32}$$

Turning to the case of a mini-batch of size say b, the main effort in computing the gradient consists in solving matrix equations in a $n \times b$ matrix V:

$$V = \Psi \odot \left(A^\top V + C^\top G \right),$$

where the columns of G contains the gradients of the loss with respect to $\hat{y}$, and Ψ is a matrix whose columns contain the derivatives of the activation map, both evaluated at a specific training point, and $\odot$ represents element-wise multiplication. Due to the fact that A satisfies the PF sufficient condition for well-posedness with respect to ϕ, the equation above has a unique solution; the matrix V can be computed as the limit point of the convergent fixed-point iterations

$$V(t+1) = \Psi \odot \left(A^\top V(t) + C^\top L \right), \quad t = 0, 1, 2, \ldots. \tag{2.33}$$

Projection step In order to handle the well-posedness constraint $\|A\|_\infty \leq \kappa$, the projected gradient method requires a projection at each step. This step corresponds to a sub-problem of the form:

$$\min_A \ \|A - A^0\|_F \ : \ \|A\|_\infty \leq \kappa, \tag{2.34}$$

with matrix A^0 given. The above problem is decomposable across rows, leading to n subproblems of the form

$$\min_a \ \frac{1}{2}\|a - a_i^0\|_2^2 \ : \ \|a\|_1 \leq \kappa,$$

which $a_i^0 \in \mathbb{R}^n$ the i-th row of A^0. The problem cannot be solved in closed form, but a bisection method can be applied to the dual:

$$p^* = \max_{\lambda \geq 0} \ -\kappa\lambda + \sum_{i \in [n]} s_i(\lambda),$$

where, for $\lambda \geq 0$ given:

$$s_i(\lambda) := \min_\xi \ \frac{1}{2}(\xi - a_i^0)^2 + \lambda|\xi|, \ \ i \in [n].$$

A subgradient of the objective is

$$g_i(\lambda) := -\kappa + \sum_{i \in [n]} \max(|a_i^0| - \lambda, 0), \ \ i \in [n].$$

Observe that $p^* \geq 0$, hence at optimum:

$$0 \leq \lambda \leq \frac{1}{\kappa} \sum_{i \in [n]} s(\lambda, a_i^0) \leq \lambda^{\max} := \frac{1}{2\kappa}\|a_i^0\|_2^2.$$

The bisection can be initialized with the interval $\lambda \in [0, \lambda^{\max}]$.

Returning to the original problem (2.34), we see that all the iterations can be expressed in a "vectorized" form, where updates for the different rows of A are done in parallel. The dual variables corresponding to each row are collected in a vector $\lambda \in \mathbb{R}^n$. We initialize the bisection with a vector interval $[\lambda_l, \lambda_u]$, with $\lambda^l = 0$, $\lambda_i^u = \frac{1}{2}\|a_i^0\|_2^2/\kappa$, $i \in [n]$. We update the current vector interval as follows:

1. Set $\lambda = (\lambda_l + \lambda_u)/2$.
2. Form a vector $g(\lambda)$ containing the sub-gradients corresponding to each row, evaluated at λ_i, $i \in [n]$:
$$g(\lambda) = -\kappa\mathbf{1} + (|A^0| - \lambda\mathbf{1}^T)_+^T\mathbf{1}.$$
3. For every $i \in [n]$, reset $\lambda_i^u = \lambda_i$ if $g_i(\lambda) > 0$, $\lambda_i^l = \lambda_i$ if $g_i(\lambda) \leq 0$.

Block-coordinate descent methods

Block-coordinate descent methods use cyclic updates, optimizing one matrix variable at a time, fixing all the other variables. Such methods are easier to apply to a so-called Fenchel formulation of the problem, which is equivalent to problem (2.28):

$$\min_{A,B,C,D,X} \ \mathcal{L}(Y, CX + DU) + \mathcal{P}(A, B, C, D) \ : \ F_\phi(X, AX + BU) \leq 0, \ \ \|N(A, \gamma)\|_\infty \leq \kappa,$$

where F_ϕ is the so-called Fenchel divergence adapted to ϕ [71] (also see Chapter 3), applied column-wise to matrix inputs. In the case of the ReLU activation, for two given vectors x, z of the same size, we have

$$F_\phi(x,z) := \frac{1}{2}x \odot x + \frac{1}{2}z_+ \odot z_+ - x \odot z \text{ if } x \geq 0, \tag{2.35}$$

with $\odot$ the componentwise multiplication. We can then define $F_\phi(X,Z)$ with X, Z two matrix inputs having the same number of columns, by summing over these columns. As seen in [71], a large number of popular activation maps can be expressed similarly.

The BCD methods are particularly interesting when the updates require solving convex problems. For instance, considering the training problem (2.30), given X, the problem is convex in the model matrices A, B, C, D. Then, given the model matrices, finding X consists in a feasibility problem that can be solved with fixed-point iterations.

We may also consider a relaxed form of the problem:

$$\min_{A,B,C,D,X} \mathcal{L}(Y, CX + DU) + \mathcal{P}(A,B,C,D) + \Lambda \odot F_\phi(X, AX + BU), \quad \|N(A,\gamma)\|_\infty \leq \kappa,$$

where $\Lambda > 0$ is a $n \times m$ matrix parameter. Here, all the updates involve solving convex problems, as shown in [71], since the Fenchel divergence, for most of activation maps, is bi-convex in its two arguments—meaning that given the first argument fixed, it is convex in the second and vice-versa. We refer to [71, 167] for more on Fenchel divergences in the context of implicit deep learning and neural networks.

2.8 Numerical Experiments

Learning real nonlinear functions via regression

We start by illustrating the ability of the gradient method, as presented in Section 2.7, to learn the parameters of the implicit model, with well-posedness enforced via a max-row-sum norm constraint, $\|A\|_\infty \leq 0.5$. We aim at learning a real function, as an example we focus on

$$f(u) = 5\cos(\pi u)\exp\left(-\frac{1}{2}|u|\right),$$

We select the input u_i, $i \in \{1, \cdots, m\}$ uniformly at random between -5 and 5 with $m = 200$; we add a random noise to the output, $y(u) = f(u) + w$ with w taken uniformly at random between -1 and 1, hence the standard deviation for $y(u)$ is $1/\sqrt{3} \approx 0.57$. We consider an implicit model of order $n = 75$. We learn the parameters of the model by doing only two successive block updates: first, we update (A, B) using stochastic projected gradient descent, the gradient being obtained with the implicit chain rule described in Section 2.7. The RMSE across iterations for this block-update is shown in Figure 2.7. After this first update we achieve a RMSE of 1.77. Second, we update (C, D) using linear regression. After

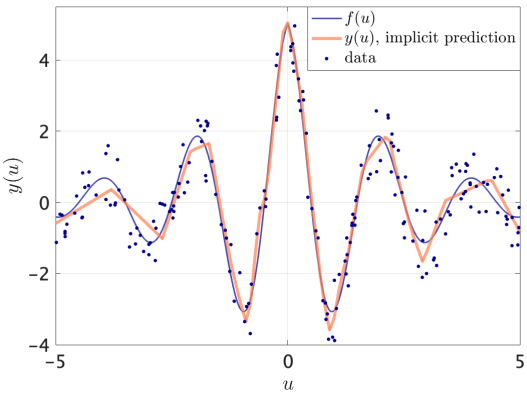


Figure 2.6: Implicit prediction $y(u)$ comparison with $f(u)$

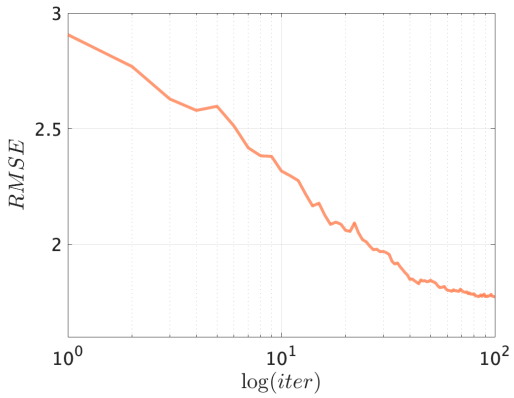


Figure 2.7: RMSE across projected gradient iterations for the (A, B) block update

this update we achieve a RMSE of 0.56. For comparison purposes, we also train a neural network with 3 hidden layers of width $n/3 = 25$ using ADAM, mini-batches, and a tuned learning rate. We run Adam until convergence. We get a RMSE $= 0.65$, which is slightly above that of our implicit model. This first simple experiment shows the ability to fit nonlinear functions with implicit models as illustrated in Figure 2.6.

Comparison with neural networks

In this section we compare the performance of implicit models with that of neural networks. Experiments on both synthetic datasets and real datasets are conducted. The experiment results show that implicit model has the potential of matching or exceeding the performance of neural networks in various settings. To simplify the experiments, we do not apply any specific network structure or regularization during training. When it comes to training neural networks we will always use ADAM with a grid-search tuned step-size. Similarly, we will always use stochastic projected gradient descent as detailed in Section 2.7. The choice of the number of hidden features n for implicit models is always aligned with the neural network architecture for fair comparison as explained in Section 2.4.

Synthetic datasets

We consider two synthetic classification datasets: one generated from a neural network and another from an implicit model. For each dataset, we then aim at fitting both an implicit model and a neural network. Each data point in the datasets contains an input $u \in \mathbb{R}^5$ and output $y \in \mathbb{R}^2$. The model architectures and data generation details are deferred to the supplementary materials.

We find that the implicit model outperforms neural networks in both synthetic experiments. This may be explained by the increased modeling capacity of the implicit model,

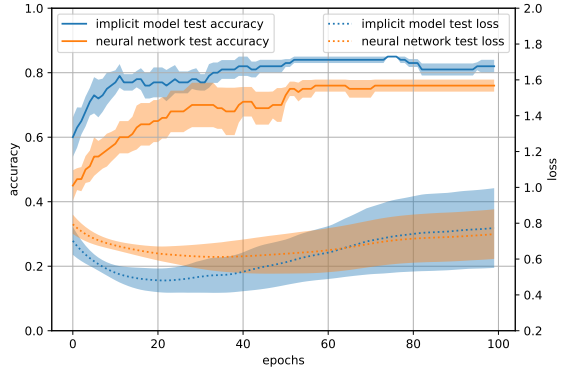


Figure 2.8: Performance comparison on a synthetic dataset generated from a neural network. Average best accuracy, implicit: 0.85, neural network: 0.76. The curves are generated from 5 the different runs with the lines marked as mean and region marked as the standard deviation

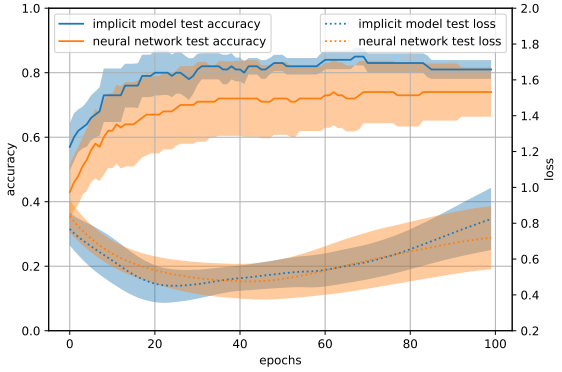


Figure 2.9: Performance comparison on a synthetic dataset generated from an implicit model. Average best accuracy, implicit: 0.85, neural networks: 0.74. The curves are generated from 5 different runs with the lines marked as mean and region marked as the standard deviation over the runs.

given similar parameter size, with respect to its neural network counterpart as mentioned in Section 2.1.

Experimental details: synthetic datasets. The dataset for experiment in Section 2.8 is generated as follows: we chose the input datapoints $u_i \in \mathbb{R}^5$ i.i.d. by sampling each entry independently uniformly between -1 and 1. The output data $y_i \in \mathbb{R}^2$ is the one-hot representation of the argmax of the output from the generating model, whose parameters are chosen uniformly at random between -1 and 1.

- For a neural network generating model, we us a fully connected 3-layer (5-5-5-2) feed-forward neural network with ReLU activation.
- For an implicit model, we consider an implicit model with $n = 10$ to match the number of hidden nodes in the neural network model. To ensure well-posedness, after sampling matrix the A, we proceed with a projection on the unit-sized infinity norm ball, as detailed in Section 2.7.

For each dataset, we consider 20 training and test datapoints. The size of datasets are kept small to maintain the over-parameterized regime, where the number of parameters is larger than the number of data points, as is the case in many deep learning architectures [20]. We run 5 separate experiments for both the neural network and implicit model generated dataset with the training model having the same hyperparameters and initialization as the generating models.

Real-world datasets

We continue to compare the performance of the implicit model with that of neural networks in real-world settings. For the purpose, we pick two real-world datasets, the hand-written digit classification dataset MNIST, and the German Traffic Sign Recognition Benchmark (GTSRB) dataset. The performance is given in Figure 2.10 and 2.11. Similar to what we observed with synthetic datasets, the implicit model is capable of matching and even outperform classical neural networks.

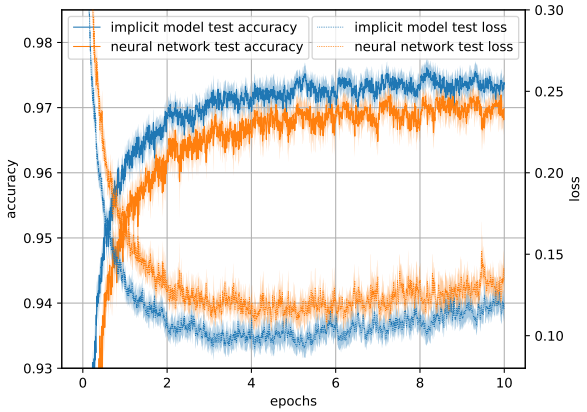


Figure 2.10: Performance comparison on MNIST. Average best accuracy, implicit: 0.976, neural networks: 0.972. The curves are generated from 5 different runs with the lines marked as mean and region marked as the standard deviation over the runs.

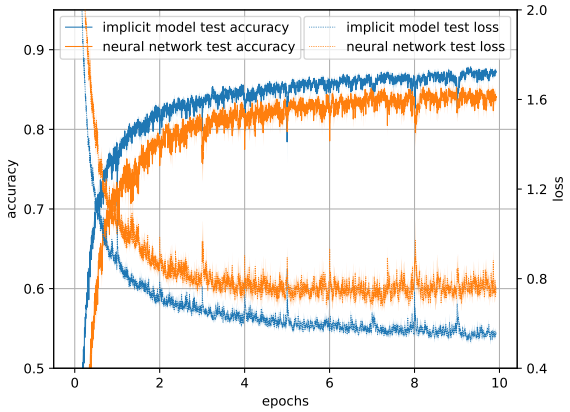


Figure 2.11: Performance comparison on GTSRB. Average best accuracy, implicit: 0.874, neural networks: 0.859. The curves are generated from 5 different runs with the lines marked as mean and region marked as the standard deviation over the runs.

Experimental details: MNIST dataset. In the digit classification dataset MNIST, the input data points are 28×28 pixels images of hand written digits, the output is the corresponding digit label. For training purposes, each image is reshaped into 784 dimensional vectors and normalized before training. There are 5×10^4 training data points and 10^4 testing data points.

The architecture we use for the neural network as a reference is a three-layer feedforward neural network (784-60-40-10) with ReLU activation. For the implicit model, we set $n = 100$. We choose a batchsize of 100 for both algorithms. The accuracy with respect to iterations is shown in Figure 2.10. We observe that the accuracy of the implicit model matches that of its neural network counterpart.

Experimental details: German Traffic Sign Recognition benchmark. In the German Traffic Sign Recognition Benchmark (GTSRB) [157], the input data points are 32×32 images with rgb channels of traffic signs consisting of 43 classes. Each image is turned into gray-scale before being reshaped into a 1024 dimensional vector and re-scaled to be between 0 and 1. There are 34799 training data points and 12630 testing data points.

For reference, we use a (1024-300-100-43) feedforward neural network with ReLU activations. We choose $n = 400$ for the implicit model and a batchsize of 100 for both models. The accuracy with respect to iterations is shown in Figure 2.11.

Adversarial attack

Visualization of sensitivity matrix

Figure 2.12 shows the sensitivity values for a particular class on MNIST and CIFAR-10 dataset. The sensitivity values are obtain from the implicit representation of the feed-forward network and ResNet-20. The 784 and 3072 input dimensions are arranged to correspond to the 28×28 (grey scale) and 32×32 (color) image pixel alignment. Brighter colors correspond to features with a higher impact on the output when perturbed. As a result, the sensitivity matrix can be used to generate adversarial attacks.

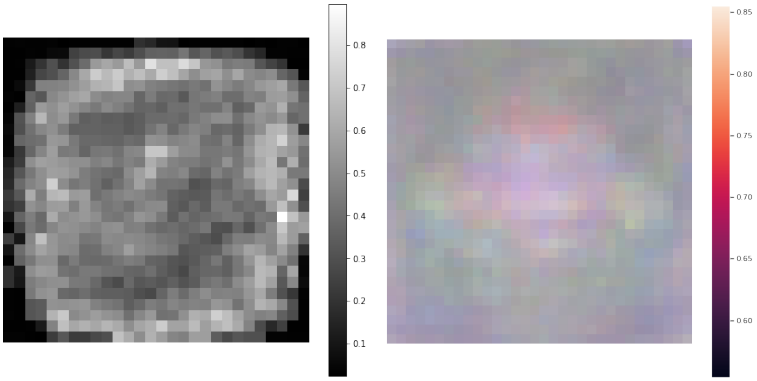


Figure 2.12: *Left*: sensitivity values of a feed-forward network for the class "digit 0" in MNIST. *Right*: sensitivity values of a ResNet-20 model for the class "airplane" in CIFAR-10. Brighter colors correspond to higher sensitivity when perturbed.

Attack via the sensitivity matrix

Our analysis above highlights the use of *sensitivity matrix* as a measure for robustness. In this section, we show how sensitivity matrix can be used to generate effective attacks on two public datasets, MNIST and CIFAR-10. Examples of sensitivity matrix are shown in the supplementary material. We compare our method against commonly used gradient-based attacks [68, 133]. In this experiment, we consider two models: 1) feed-forward network

trained on the MNIST dataset (98% clean accuracy) and 2) ResNet-20 [79] trained on the CIFAR-10 dataset (92% clean accuracy).

We compare our method with commonly used gradient-based attacks. Precisely, for a given function F (prediction rule) learned by a deep neural network, a benign sample $u \in \mathbb{R}^p$ and the target y associated with u, we compute the gradient of the function F with respect to the given sample u, $\nabla_u F(u, y)$. We then take the absolute value of the gradient as an indication of which input features an adversary should perturb, similar to the saliency map technique [154, 133]. The absolute value of the gradient can be seen as a "local" version of the sensitivity matrix; however, unlike the gradient, the sensitivity matrix does not depend on the input data, making it a more general measurement of robustness for any given model.

Table 2.1 presents the experimental results of an adversarial attack using the sensitivity matrix and the absolute value of gradient on MNIST and CIFAR-10. For sensitivity matrix attack, we start from perturbing the input features that have the highest values according to the sensitivity matrix. For gradient-based attack, we do the same according to the absolute value of the gradient. We perturb the input features into small random values. Our experiments show that the sensitivity matrix attack is as effective as the gradient-based attack, while being very simple to implement.

Interestingly, our attack does not rely on any input samples that the gradient-based attack needs. An adversary with the model parameters could easily craft adversarial samples using the sensitivity matrix. In the absence of access to the model parameters, an adversary can rely on the principle of *tranferrability* [113] and train a surrogate model to obtain the sensitivity matrix. Figure 2.13 displays adversarial images generated using the sensitivity matrix. An interesting case is to use the sensitivity matrix to generate a sparse attack as seen in Figure 2.13.

Table 2.1: Experimental results of attack success rate against percentage of perturbed inputs on MNIST and CIFAR-10 (10000 samples from test set).

	Sensitivity matrix attack		**Gradient-based attack**	
% of perturbed inputs	MNIST	CIFAR-10	MNIST	CIFAR-10
0.1%	1.01%	3.04%	2.42%	1.75%
1%	13.41%	10.16%	26.92%	6.66%
10%	70.67%	36.21%	74.90%	33.18%
20%	89.82%	57.01%	87.10%	52.57%
30%	90.22%	67.45%	89.82%	66.59%

Attack with LP relaxation for CONE maps

Although one can use sensitivity matrix to generate an effective adversarial example, one may wish to perform a more sophisticated attack by exploiting the weakness of an individual data point. This can be done by considering the LP relaxation (Theorem 2.5.3), which has the advantage of generating a specific adversarial example for a given input data. The

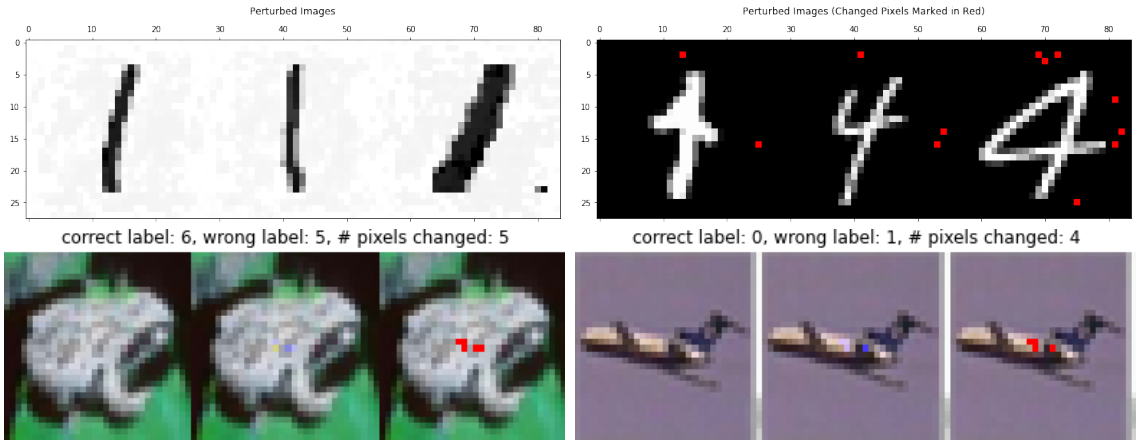


Figure 2.13: *Top*: adversarial samples from MNIST. On the left are dense attacks with small perturbations and on the right are sparse attacks with random perturbations (perturbed pixels are marked as red). *Bottom*: example sparse attack on CIFAR-10. The left ones are cleaned images, the middle ones are perturbed images, and the right ones mark the perturbed pixels in red for higher visibility.

experiment in this section is again on MNIST and CIFAR-10 images. Here, the problem outlined in (2.24) is solved by LP relaxation, with the function $f_i(\xi) = (\xi - x_i^0)^2$. The optimization problem then finds a perturbed image that leads to the largest discrepancy between the perturbed state x and the nominal state x^0. Figure 2.14 shows five example images. The perturbed images generated by the LP relaxation, appear quite similar to the original images; however, the model fails to predict these otherwise correctly predicted images.

Our framework also allows for sparse adversarial attack by adding a cardinality constraint. Figure 2.15 shows three examples of perturbed images under non-sparse and sparse attack. Images on the left are the results of non-sparse attack and those on the right are the results of sparse attack. The model fails to predict the label correctly under both conditions. These results illustrate how the implicit prediction rules can be used to generate powerful adversarial attacks. It is also useful for adversarial training as a large amount of adversarial examples can be generated using the technique and be added back to the training data.

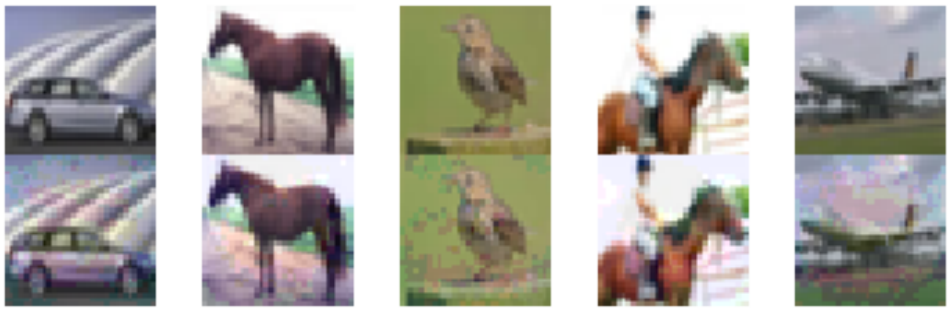


Figure 2.14: Example attack on CIFAR dataset. Top: clean data. Bottom: perturbed data.

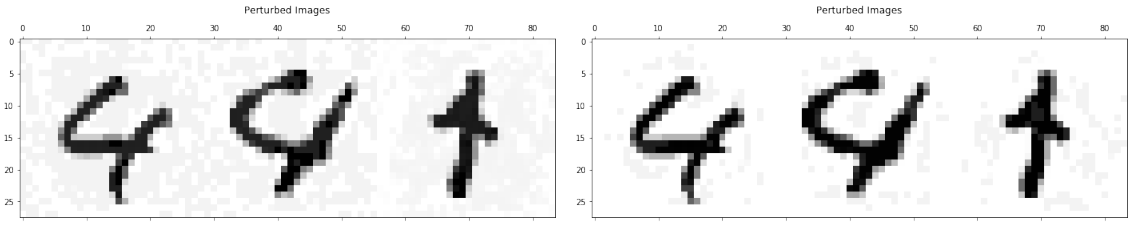


Figure 2.15: Example attack on MNIST dataset. Left: non-sparse attack. Right: sparse attack.

2.9 Summary

We summarize with a few perspectives for future research on implicit models.

Cousins of implicit models

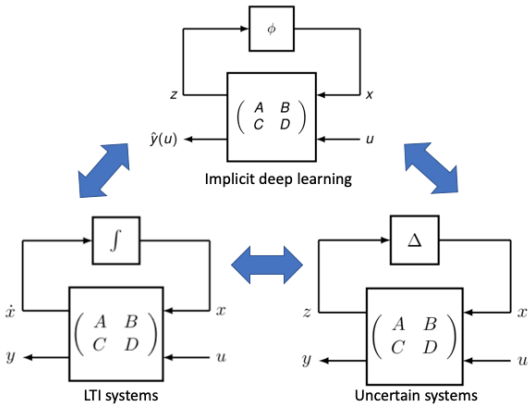


Figure 2.16: Some cousins of implicit models: LTI systems (bottom left) and uncertain systems (bottom right).

Implicit models rely on a representation of the prediction rule where the linear operations are clearly separated from the (parameter-free) nonlinear ones, leading to the block-diagram in Figure 2.1.

This idea is strongly reminiscent of earlier representations arising in systems and control theory. Perhaps the most famous example lies with linear time-invariant (LTI) systems, where the idea is essentially equivalent to the state-space representation of transfer functions [7]. In that context, the block-diagram representation involves an integrator (in continuous or discrete time) in place of the static nonlinear map ϕ. The idea is also connected to the linear-fractional representation (or, transformation, LFT) arising in uncertain systems [54], where now the linear operations actually represent a dynamic system (that is, the matrix M is an LTI system itself), and the map ϕ is replaced by an uncertain matrix of parameters or LTI system.

These connections raise the prospect of a unified theory tackling deep networks inside the loop of a dynamical system, where such networks can be composed (via feedback connections) with LTI or more generally uncertain systems. With the machinery of Linear Matrix Inequalities or the more general Integral Quadratic Constraints framework [120], one can develop rigorous analyses performed on systems with deep networks in the loop, so that bounds on, say, stability margins, can be computed. This connection is already explored in [186, 187].

Chapter 3

Fenchel Lifted Networks

Lifted methods are another branch of training methods for deep learning. By introducing auxiliary variables, it decomposes the training problem into multiple sub problems and solves them iteratively without using gradients. However, existing methods greatly under perform traditional gradient based methods. We find careful conversion of the layer feed forward constraints into bi-convex terms using Fenchel divergence (3.4) bumps the empirical performance to competitive level. This opens up practical discussion of lifted methods in deep learning training. In Chapter 2, a lifted training approach for implicit deep learning is discussed in Section 2.7 where Fenchel divergence (2.35) is a key step to the formulation. However this method is still not highly popular in the deep learning community because gradient methods are highly flexible and easy to use. But we believe lifted methods will outperform gradient methods in tasks where gradient updates are not very effective.

3.1 Introduction

Deep neural networks (DNNs) have become the preferred model for supervised learning tasks after their success in various fields of research. However, due to their highly non-convex nature, DNNs pose a difficult problem during training time; the optimization landscape consists of many saddle points and local minima which make the trained model generalize poorly [37, 48]. This has motivated regularization schemes such as weight decay [96], batch normalization [81], and dropout [156] so that the solutions generalize better to the test data.

In spite of this, backprop used along with stochastic gradient descent (SGD) or similar variants like Adam [90] suffer from a variety of problems. One of the most notable problems is the vanishing gradient problem which slows down gradient-based methods during training time. Several approaches have been proposed to deal with the problem; for example, the introduction of rectified linear units (ReLU). However, the problem persists. For a discussion on the limitations of backprop and SGD, we direct the reader to Section 2.1 of [164].

One approach to deal with this problem is to introduce auxiliary variables that increase the dimension of the problem. In doing so, the training problem decomposes into multiple,

local sub-problems which can be solved efficiently without using SGD or Adam; in particular, the methods of choice have been block coordinate descent (BCD) [8, 99, 195, 31] and the alternating direction method of multipliers (ADMM) [164, 196]. By lifting the dimension of the problem, these models avoid many of the problems DNNs face during training time. They also offer new avenues towards interpretability and robustness of networks by allowing for the penalization of the additional variables.

While these methods, which we refer to as "lifted" models for the remainder of the chapter, offer an alternative to the original problem with some added benefits, they have their limitations. Most notably, traditional DNNs are still able to outperform these methods in spite of the difficult optimization landscape. As well, most of the methods are unable to operate in an online manner or adapt to continually changing data sets which is prevalent in most reinforcement learning settings [161]. Finally, by introducing auxiliary variables, the dimensionality of the problem greatly increases, making these methods very difficult to train with limited computational resources.

Chapter contribution

To address the problems listed above, we propose Fenchel lifted networks, a biconvex formulation for deep learning based on Fenchel's duality theorem that can be optimized using BCD. We show that our method is a rigorous *lower bound* for the learning problem and admits a natural batching scheme to adapt to changing data sets and settings with limited computational power. We compare our method against other lifted models and against traditional fully connected and convolutional neural networks. We show that we are able to outperform the former and that we can compete with or even outperform the latter.

Chapter outline. In Section 3.2, we give a brief overview of related works on lifted models. In Section 3.3 we introduce the notation for the remainder of the chapter. Section 3.4 introduces Fenchel lifted networks, their variants and discusses how to train these models using BCD. Section 3.5 compares the proposed method against fully connected and convolutional networks on MNIST and CIFAR-10.

3.2 Related Work

Lifted methods Related works that lift the dimension of the training problem are primarily optimized using BCD or ADMM. These methods have experienced recent success due to their ability to exploit the structure of the problem by first converting the constrained optimization problem into an unconstrained one and then solving the resulting sub-problems in parallel. They do this by relaxing the network constraints and introducing penalties into the objective function. The two main ways of introducing penalties into the objective function are either using quadratic penalties [158, 164, 99] or using equivalent representations of the activation functions [8, 195].

As a result, these formulations have many advantages over the traditional training problem, giving superior performance in some specific network structures [31, 195]. These methods also enjoy great potential for parallelization as shown by [164]; the authors parallelize training over different cores and show a linear scaling between reduction in training time and the number of cores used. However, there has been little evidence showing that these methods can compete with traditional DNNs which shadows the nice structure these formulations bring about.

An early example of auxiliary variables being introduced into the training problem is the method of auxiliary coordinates (MAC) by [31] which uses quadratic penalties to enforce network constraints. They test their method on auto encoders and show that their method is able to outperform SGD. Followup work by [32, 164] demonstrate the huge potential for parallelizing these methods. [99] gives some convergence guarantee on a modified problem.

Another class of models that lift the dimension of the problem do so by representing activation functions in equivalent formulations. [127, 8, 195, 108] explore the structure of activation functions and use arg min maps to represent activation functions. In particular, [8] show how a strictly monotone activation function can be seen as the arg min of a specific optimization problem. Just as with quadratic penalties, this formulation of the problem still performs poorly compared to traditional neural networks. In an independent and concurrent work by [108], the authors arrive at a lifted formulation of the training problem via the use of proximal operators. While the approach in aforementioned work appears unrelated to the work presented here, they are in fact deeply related (for a more complete discussion, see a note at the end of the chapter).

3.3 Background and Notation

Feedforward neural networks. We are given an input data matrix of m data points $X = [x_1, x_2, ..., x_m] \in \mathbb{R}^{n\times m}$ and a response matrix $Y \in \mathbb{R}^{p\times m}$. We consider the supervised learning problem involving a neural network with $L \geq 1$ hidden layers. The neural network produces a prediction $\hat{Y} \in \mathbb{R}^{p\times m}$ with the feed forward recursion $\hat{Y} = W_L X_L + b_L \mathbf{1}^\top$ given below

$$X_{l+1} = \phi_l(W_l X_l + b_l \mathbf{1}^\top), \quad l = 0, \ldots, L-1. \tag{3.1}$$

where $\phi_l, l = 0, \ldots, L$ are the activation functions that act column-wise on a matrix input, $\mathbf{1} \in \mathbb{R}^m$ is a vector of ones, and $W_l \in \mathbb{R}^{p_{l+1}\times p_l}$ and $b_l \in \mathbb{R}^{p_{l+1}}$ are the weight matrices and bias vectors respectively. Here p_l is the number of output values for a single data point (i.e., hidden nodes) at layer l with $p_0 = n$ and $p_{L+1} = p$. Without loss of generality, we can remove $b_l \mathbf{1}^\top$ by adding an extra column to W_l and a row of ones to X_l. Then (3.1) simplifies to

$$X_{l+1} = \phi_l(W_l X_l), \quad l = 0, \ldots, L-1. \tag{3.2}$$

In the case of fully connected networks, ϕ_l is typically sigmoidal activation functions or ReLUs. In the case of Convolutional Neural Networks (CNNs), the recursion can accommodate convolutions and pooling operations in conjunction with an activation. For classification tasks, we typically apply a softmax function after applying an affine transformation to X_L.

The initial value for the recursion is $X_0 = X$ and $X_l \in \mathbb{R}^{p_l \times m},\ l = 0, \ldots, L$. We refer to the collections $(W_l)_{l=0}^L$ and $(X_l)_{l=1}^L$ as the W and X-variables respectively.

The weights are obtained by solving the following constrained optimization problem

$$\begin{aligned} \min_{(W_l)_{l=0}^L, (X_l)_{l=1}^L} \ & \mathcal{L}(Y, W_L X_L) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ \text{s.t.} \ & X_{l+1} = \phi_l(W_l X_l),\ l = 0, \ldots, L-1 \\ & X_0 = X \end{aligned} \tag{3.3}$$

Here, $\mathcal{L}$ is a loss function, $\rho \in \mathbb{R}_+^{L+1}$ is a hyper-parameter vector, and π_l's are penalty functions used for regularizing weights, controlling network structures, etc. In (3.3), optimizing over the X-variables is trivial; we simply apply the recursion (3.2) and solve the resulting unconstrained problem using SGD or Adam. After optimizing over the weights and biases, we obtain a prediction $\hat{Y}$ for the test data X by passing X through the recursion (3.2) one layer at a time.

Our model. We develop a family of models where we approximate the recursion constraints (3.2) via penalties. We use the arg min maps from [8] to create a biconvex formulation that can be trained efficiently using BCD and show that our model is a lower bound of (3.3). Furthermore, we show how our method can naturally be batched to ease computational requirements and improve the performance.

3.4 Fenchel Lifted Networks

In this section, we introduce Fenchel lifted networks. We begin by showing that for a certain class of activation functions, we can equivalently represent them as biconvex constraints. We then dualize these constraints and construct a lower bound for the original training problem. We show how our lower bound can naturally be batched and how it can be trained efficiently using BCD.

Activations as bi-convex constraints

In this section, we show how to convert the equality constraints of (3.3) into inequalities which we dualize to arrive at a *relaxation* (lower bound) of the problem. In particular, this lower bound is biconvex in the W-variables and X-variables. We make the following assumption on the activation functions ϕ_l.

BC Condition The activation function $\phi : \mathbb{R}^p \to \mathbb{R}^q$ satisfies the BC condition if there exists a *biconvex* function $B_\phi : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_+$, such that

$$v = \phi(u) \iff B_\phi(v, u) \leq 0.$$

We now state and prove a result that is at the crux of Fenchel lifted networks.

Theorem 3.4.1. *Assume $\phi : \mathbb{R} \to \mathbb{R}$ is continuous, strictly monotone and that $0 \in$ range(ϕ) or $0 \in$ domain(ϕ). Then ϕ satisfies the BC condition.*

Proof. Without loss of generality, ϕ is strictly increasing. Thus it is invertible and there exists ϕ^{-1} such that $u = \phi^{-1}(v)$ for $v \in \text{range}(\phi)$ which implies $v = \phi(u)$. Now, define $F : \mathbb{R}^p \to \mathbb{R}$ as

$$F(v) := \int_z^v \phi^{-1}(\xi)\, d\xi$$

where $z \in \text{range}(\phi)$ and is either 0 or satisfies $\phi^{-1}(z) = 0$. Then we have

$$\begin{aligned} F^*(u) &= \int_{\phi^{-1}(z)}^u \phi(\eta)\, d\eta \\ B(v, u) &= F(v) + F^*(u) - uv \end{aligned} \tag{3.4}$$

where F^* is the Fenchel conjugate of F, and the bi-convex function $B(v, u)$ is also called Fenchel divergence. By the Fenchel-Young inequality, $B(v, u) \geq 0$ with equality if and only if

$$v^* = \arg\max_v \; uv - F(v) \;:\; v \in \text{range}(\phi)$$

By construction, $v^* = \phi(u)$. Note furthermore since ϕ is continuous and strictly increasing, so is ϕ^{-1} on its domain, and thus F, F^* are convex. It follows that $B(v, u)$ is a biconvex function of (u, v). Discussion on learning with similar loss terms can be found in [23, 24].

We simply need to prove that $F^*(u)$ above is indeed the Fenchel conjugate of F. By definition of the Fenchel conjugate we have that

$$F^*(u) = \max_v \; uv - F(v) \;:\; v \in \text{range}(\phi)$$

It is easy to see that $v^* = \phi(u)$. Thus

$$\begin{aligned} F^*(u) &= u\phi(u) - F(\phi(u)) \\ &= u\phi(u) - \int_z^{\phi(u)} \phi^{-1}(\xi)\, d\xi \\ &= \int_z^{\phi(u)} \xi \frac{d}{d\xi} \phi^{-1}(\xi)\, d\xi \\ &= \int_{\phi^{-1}(z)}^u \phi(\eta)\, d\eta \end{aligned}$$

where the third equality is a consequence of integration by parts, and the fourth equality we make the subsitution $\eta = \phi^{-1}(\xi)$ ■

Note that Theorem 3.4.1 implies that activation functions such as sigmoid and tanh can be equivalently written as a biconvex constraint. Although the ReLU is not strictly monotone, we can simply restrict the inverse to the domain $\mathbb{R}_+$; specifically, for $\phi(x) = \max(0, x)$ define

$$\phi^{-1}(z) = \begin{cases} +\infty & \text{if } z < 0, \\ z & \text{if } z \geq 0, \end{cases}$$

Then, we can rewrite the ReLU function as the equivalent set of biconvex constraint

$$v = \max(0, u) \iff \begin{cases} \frac{1}{2}v^2 + \frac{1}{2}u_+^2 - uv \leq 0 \\ v \geq 0 \end{cases}$$

where $u_+ = \max(0, u)$. This implies

$$B_\phi(v, u) = \begin{cases} \frac{1}{2}v^2 + \frac{1}{2}u_+^2 - uv & \text{if } v \geq 0 \\ +\infty & \text{otherwise} \end{cases} \tag{3.5}$$

Despite the non-smoothness of u_+, for fixed u or fixed v, (3.5) belongs in C^1 – that is, it has continuous first derivative and can be optimized using first order methods. We can trivially extend the result of Theorem 3.4.1 for matrix inputs: for matrices $U, V \in \mathbb{R}^{p \times q}$, we have

$$B_\phi(V, U) = \sum_{i,j} B_\phi(V_{ij}, U_{ij}).$$

Lifted Fenchel model

Assuming the activation functions of (3.3) satisfy the hypothesis of Theorem 3.4.1, we can reformulate the learning problem equivalently as

$$\begin{aligned} \min_{(W_l)_{l=0}^L, (X_l)_{l=1}^L} \; & \mathcal{L}(Y, W_L X_L) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ \text{s.t. } & B_l(X_{l+1}, W_l X_l) \leq 0, \; l = 0, \ldots, L-1 \\ & X_0 = X, \end{aligned} \tag{3.6}$$

where B_l is the short-hand notation of B_{ϕ_l}. We now dualize the inequality constraints and obtain the lower bound of the standard problem (3.3) via Lagrange relaxation

$$\begin{aligned} G(\lambda) := \min_{(W_l)_{l=0}^L, (X_l)_{l=1}^L} \; & \mathcal{L}(Y, W_L X_L) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ & + \sum_{l=0}^{L-1} \lambda_l B_l(X_{l+1}, W_l X_l) \\ \text{s.t. } & X_0 = X, \end{aligned} \tag{3.7}$$

where $\lambda_l \geq 0$ are the Lagrange multipliers. The maximum lower bound can be achieved by solving the dual problem

$$p^* \geq d^* = \max_{\lambda \geq 0} G(\lambda) \tag{3.8}$$

where p^* is the optimal value of (3.3). Note if all our activation functions are ReLUs, we must also include the constraint $X_l \geq 0$ in the training problem as a consequence of (3.5). Although the new model introduces L new parameters (the Lagrange multipliers), we can show that using variable scaling we can reduce this to only *one* hyperparameter (for details, see the *Variable Scaling* paragraph below).The learning problem then becomes

$$\begin{aligned} G(\lambda) := \min_{(W_l)_{l=0}^L, (X_l)_{l=1}^L} \; & \mathcal{L}(Y, W_L X_L) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ & + \lambda \sum_{l=0}^{L-1} B_l(X_{l+1}, W_l X_l) \\ \text{s.t. } & X_0 = X. \end{aligned} \tag{3.9}$$

In a regression setting where the data is generated by a one layer network, we are able to provide global convergence guarantees of the above model (for details, see the *One-layer Regression Formulation* paragraph below).

Variable Scaling. Note that the new model (3.9) has introduced $L+1$ more hyperparameters. We can use variable scaling and the dual formulation to show how to effectively reduce this to only *one* hyperparameter. Consider the model with ReLU activations, that is, the biconvex function as in (3.5) and regularization functions $\pi_l(W_l) = \|W_l\|_F^2$ for $l = 0, \ldots, L$. Note that B_ϕ is homogeneous of degree 2, that is for any U, V and γ we have

$$\gamma B_\phi(V, U) = B_\phi(\sqrt{\gamma} V, \sqrt{\gamma} U)$$

Define $\lambda_{-1} = 1$ and the scalings

$$\bar{X}_l := \sqrt{\lambda_{l-1}} X_l, \;\; \bar{W}_l := \sqrt{\frac{\lambda_l}{\lambda_{l-1}}} W_l,$$

Then (3.9) becomes

$$\begin{aligned} G(\lambda) := \min_{(\bar{W}_l)_{l=0}^{L},(\bar{X}_l)_1^{L+1}} \; & \mathcal{L}(Y, \sqrt{\lambda_L}(\bar{W}_L\bar{X}_L)) \\ & + \sum_{l=0}^{L} \rho_l \pi_l(\sqrt{\frac{\lambda_{l-1}}{\lambda_l}} W_l) + \sum_{l=0}^{L-1} B_l(\bar{X}_{l+1}, \bar{W}_l\bar{X}_l) \\ \text{s.t. } & \bar{X}_0 = X, \; \bar{X}_l \geq 0, \; l = 0, \ldots, L \end{aligned} \tag{3.10}$$

Using the fact $\pi_l(W_l) = \|W_l\|_F^2$ and defining $\bar{\rho}_l = \rho_l \dfrac{\lambda_{l-1}}{\lambda_l}$ we have

$$\begin{aligned} G(\lambda) := \min_{(\bar{W}_l)_{l=0}^{L},(\bar{X}_l)_1^{L+1}} \; & \mathcal{L}(Y, \sqrt{\lambda_L}(\bar{W}_L\bar{X}_L)) \\ & + \sum_{l=0}^{L} \bar{\rho}_l \|\bar{W}_l\|_F^2 + \sum_{l=0}^{L-1} B_l(\bar{X}_{l+1}, \bar{W}_l\bar{X}_l) \\ \text{s.t. } & \bar{X}_0 = X, \; \bar{X}_l \geq 0, \; l = 0, \ldots, L \end{aligned} \tag{3.11}$$

where $G(\lambda)$ is now only a function of one variable λ_L as opposed to L variables. Note that this argument for variable scaling still works when we use average pooling or convolution operations in conjunction with a ReLU activation since they are linear operations. Note furthermore that the same scaling argument works in place of any norm due to the homogeneity of norms – the only thing that would change is how $\bar{\rho}$ is scaled by λ_{l-1} and λ_l.

Another way to show that we only require one hyperparameter λ is to note the equivalence

$$B_l(v, u) \leq 0 \;\; \forall l \iff \sum_l B_l(v, u) \leq 0$$

Then we may replace the L biconvex constraints in (3.6) by the equivalent constraint $\sum_l B_l(v, u) \leq 0$. Since this is only one constraint, when we dualize we only introduce *one* Lagrange multiplier λ.

One-layer Regression Formulation. In this section, we show that for a one layer network we are able to convert a non-convex optimization problem into a convex one by using the BC condition described in the main text.

Consider a regression setting where $Y = \phi(W^*X)$ for some fixed $W^* \in \mathbb{R}^{p\times n}$ and a given data matrix $X \in \mathbb{R}^{n\times m}$. Given a training set (X, Y) we can solve for W by solving the following non-convex problem

$$\min_W \; \|Y - \phi(WX)\|_F^2. \tag{3.12}$$

We could also solve the following relaxation of (3.12) based on the BC condition

$$\min_{W} \; B_{\phi}(Y, WX) \tag{3.13}$$

Note (3.13) is trivially convex in W by definition of $B_{\phi}(\cdot, \cdot)$. Furthermore, by construction $B_{\phi}(Y, WX) \geq 0$ and $B_{\phi}(Y, WX) = 0$ if and only if $Y = \phi(WX)$. Since $Y = \phi(W^*X)$, it follows W^* (which is the minimizer of (3.12)) is a global minimizer of the convex program (3.13). Therefore, we can solve the original non-convex problem (3.12) to global optimality by instead solving the convex problem presented in (3.13).

Comparison with other methods. For ReLU activations, $B(v, u)$ as in (3.5) differs from the penalty terms introduced in previous works. In [8, 195] they set $B(v, u) = \|v - u\|_2^2$ and in [164, 31] they set $B(v, u) = \|v - u_+\|_2^2$. Note that $B(v, u)$ in the latter is not biconvex. While the $B(v, u)$ in the former is biconvex, it does not perform well at test time. [108] set $B(v, u)$ based on a proximal operator that is similar to the BC condition.

Convolutional model. Our model can naturally accommodate average pooling and convolution operations found in CNNs, since they are linear operations. We can rewrite $W_l X_l$ as $W_l * X_l$ where $*$ denotes the convolution operator and write $\text{Pool}(X)$ to denote the average pooling operator on X. Then, for example, the sequence Conv $\rightarrow$ Activation can be represented via the constraint

$$B_l(X_{l+1}, W_l * X_l) \leq 0, \tag{3.14}$$

while the sequence Pool $\rightarrow$ Conv $\rightarrow$ Activation can be represented as

$$B_l(X_{l+1}, W_l * \text{Pool}(X_l)) \leq 0. \tag{3.15}$$

Note that the pooling operation changes the dimension of the matrix.

Prediction rule.

In previous works that reinterpret activation functions as arg min maps [8, 195], the prediction at test time is defined as the solution to the optimization problem below

$$\begin{aligned} \hat{y} = \arg\min_{y, (x_l)} \; & \mathcal{L}(y, W_L x_L) + \lambda \sum_{l=0}^{L-1} B_l(x_{l+1}, W_l x_l) \\ \text{s.t. } & x_0 = x, \end{aligned} \tag{3.16}$$

where x_0 is test data point, $\hat{y}$ is the predicted value, and x_l, $l = 1, \ldots, L$ are the intermediate representations we optimize over. Note if $\mathcal{L}$ is a mean squared error, applying the traditional feed-forward rule gives an optimal solution to (3.16). We find empirically that applying the standard feed-forward rule works well, even with a cross-entropy loss.

Batched model

The models discussed in the introduction usually require the entire data set to be loaded into memory which may be infeasible for very large data sets or for data sets that are continually changing. We can circumvent this issue by batching the model. By sequentially loading a part of the data set into memory and optimizing the network parameters, we are able to train the network with limited computational resources. Formally, the batched model is

$$\begin{aligned}\min_{(W_l)_{l=0}^L,(X_l)_{l=1}^L} \quad & \mathcal{L}(Y, W_L X_L) + \sum_{l=0}^{L} \rho_l \pi_l(W_l) \\ +\lambda \sum_{l=0}^{L-1} B_l(W_l X_l, X_{l+1}) + & \sum_{l=0}^{L} \gamma_l \|W_l - W_l^0\|_F^2 \\ & \text{s.t. } X_0 = X, \end{aligned} \tag{3.17}$$

where X_0 contains only a batch of data points instead of the complete data set. The additional term in the objective $\gamma_l \|W_l - W_l^0\|_F^2, \quad l = 0, \ldots, L$ is introduced to moderate the change of the W-variables between subsequent batches; here W_l^0 represents the optimal W variables from the previous batch and $\gamma \in \mathbb{R}_+^{L+1}$ is a hyperparameter vector. The X-variables are reinitialized each batch by feeding the new batch forward through the equivalent standard neural network.

Block-coordinate descent algorithm

The model (3.9) satisfies the following properties:

- For fixed W-variables, and fixed variables $(X_j)_{j \neq l}$, the problem is convex in X_l, and is decomposable across data points.
- For fixed X-variables, the problem is convex in the W-variables, and is decomposable across layers, and data points.

The non-batched and batched Fenchel lifted network are trained using block coordinate descent algorithms highlighted in Algorithms 1 and 2. By exploiting the biconvexity of the problem, we can alternate over updating the X-variables and W-variables to train the network.

Note Algorithm 2 is different from Algorithm 1 in three ways. First, re-initialization is required for the X-variables each time a new batch of data points are loaded. Second, the sub-problems for updating W-variables are different as shown in Section 3.4. Lastly, an additional parameter K is introduced to specify the number of training alternations for each batch. Typically, we set $K = 1$.

Algorithm 1 Non-batched BCD Algorithm

Initialize $(W_l)_{l=0}^L$.
Initialize X_0 with input matrix X.
Initialize $X_1, \ldots, X_L$ with neural network feed forward rule.
repeat
    $X_L \leftarrow \arg\min_Z \ \mathcal{L}(Y, W_L Z) + \lambda B_{L-1}(Z, X_{L-1}^0)$
    for $l = L-1, \ldots, 1$ **do**
        $X_l \leftarrow \arg\min_Z \ B_l(X_{l+1}, W_l Z) + B_{l-1}(Z, X_{l-1}^0)$
    end for
    $W_L \leftarrow \arg\min_W \ \mathcal{L}(Y, W X_L) + \rho_l \pi_l(W)$
    for $l = L-1, \ldots, 0$ **do**
        $W_l \leftarrow \arg\min_W \ \lambda B_l(X_{l+1}, W X_l) + \rho_l \pi_l(W)$
    end for
until convergence

Algorithm 2 Batched BCD Algorithm

Initialize $(W_l)_{l=0}^L$.
repeat
    $(W_l^0)_{l=0}^L \leftarrow (W_l)_{l=0}^L$
    Re-initialize X_0 with a batch sampled from input matrix X.
    Re-initialize $X_1, \ldots, X_L$ with neural network feed forward rule.
    for alternation$= 1, \ldots, K$ **do**
        $X_L \leftarrow \arg\min_Z \ \mathcal{L}(Y, W_L Z) + \lambda B_{L-1}(Z, X_{L-1}^0)$
        for $l = L-1, \ldots, 1$ **do**
            $X_l \leftarrow \arg\min_Z \ \lambda B_l(X_{l+1}, W_l Z) + \lambda B_{l-1}(Z, X_{l-1}^0)$
        end for
        $W_L \leftarrow \arg\min_W \ \mathcal{L}(Y, W X_L) + \rho_L \pi_L(W) + \gamma_l \|W - W_L^0\|_F^2$
        for $l = L-1, \ldots, 0$ **do**
            $W_l \leftarrow \arg\min_W \ \lambda B_l(X_{l+1}, W X_l) + \rho_l \pi_l(W) + \gamma_l \|W - W_l^0\|_F^2$
        end for
    end for
until convergence

Updating X-variables

For fixed W-variables, the problem of updating X-variables can be solved by cyclically optimizing $X_l, \ \ l = 1, \ldots, L$, with $(X_j)_{j \neq l}$ fixed. We initialize our X-variables by feeding forward through the equivalent neural network and update the X_l's backward from X_L to X_1 in the spirit of backpropagation.

We can derive the sub-problem for X_l, $l = 1, \ldots, L-1$ with $(X_j)_{j \neq l}$ fixed from (3.6). The sub-problem writes

$$X_l^+ = \arg\min_Z B_l(X_{l+1}, W_l Z) + B_{l-1}(Z, X_{l-1}^0) \tag{3.18}$$

where $X_{l-1}^0 := W_{l-1} X_{l-1}$. By construction, the sub-problem (3.18) is convex and parallelizable across data points. Note in particular when our activation is a ReLU, the objective function in (3.18) is in fact strongly convex and has a continuous first derivative.

For the last layer (i.e., $l = L$), the sub-problem derived from (3.6) writes differently

$$X_L^+ = \arg\min_Z \mathcal{L}(Y, W_L Z) + \lambda B_{L-1}(Z, X_{L-1}^0) \tag{3.19}$$

where $X_{L-1}^0 := W_{L-1} X_{L-1}$. For common losses such as mean square error (MSE) and cross-entropy, the subproblem is convex and parallelizable across data points. Specifically, when the loss is MSE and we use a ReLU activation at the layer before the output layer, (3.19) becomes

$$X_L^+ = \arg\min_{Z \geq 0} ||Y - W_L Z||_F^2 + \frac{\lambda}{2} ||Z - X_{L-1}^0||_F^2$$

where $X_{L-1}^0 := W_{L-1} X_{L-1}$ and we use the fact that X_{L-1}^0 is a constant to equivalently replace B_{L-1} as in (3.5) by a squared Frobenius term. The sub-problem is a non-negative least squares for which specialized methods exist [87].

For a cross-entropy loss and when the second-to-last layer is a ReLU activation, the sub-problem for the last layer takes the convex form

$$\begin{aligned} X_L^+ = &\arg\min_{Z \geq 0} - \mathbf{Tr}\, Y^\top \log s(W_L Z) + \\ &\frac{\lambda}{2} ||Z - X_{L-1}^0||_F^2, \end{aligned} \tag{3.20}$$

where $s(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$ is the softmax function and log is the element-wise logarithm. [8] show how to solve the above problem using bisection.

Updating W-variables

With fixed X-variables, the problem of updating the W-variables can be solved in parallel across layers.

Sub-problems for non-batched model. The problem of updating W_l at intermediate layers becomes

$$W_l = \arg\min_{W,} \lambda B_l(X_{l+1}, W X_l) + \rho_l \pi_l(W). \tag{3.21}$$

Again, by construction, the sub-problem (3.21) is convex and parallelizable across data points. Also, since there is no coupling in the W-variables between layers, the sub-problem (3.21) is parallelizable across layers.

For the last layer, the sub-problem becomes

$$W_L = \arg\min_W \mathcal{L}(Y, WX_L) + \rho_L \pi_L(W). \tag{3.22}$$

Sub-problems for batched model. As shown in Section 3.4, the introduction of regularization terms between W and values from a previous batch require the sub-problems (3.21, 3.22) be modified. (3.21) now becomes

$$\begin{aligned} W_l = \arg\min_W \ & \lambda B_l(X_{l+1}, WX_l) + \rho_l \pi_l(W) \\ & + \gamma_l \|W - W_l^0\|_F^2, \end{aligned} \tag{3.23}$$

while (3.22) becomes

$$\begin{aligned} W_L = \arg\min_W \ & \mathcal{L}(Y, WX_L) + \rho_L \pi_L(W) \\ & + \gamma_L \|W - W_L^0\|_F^2. \end{aligned} \tag{3.24}$$

Note that these sub-problems in the case of a ReLU activation are strongly convex and parallelizable across layers.

3.5 Numerical Experiments

In this section, we compare Fenchel lifted networks against other lifted models discussed in the introduction and against traditional neural networks. In particular, we compare our model against the models proposed by [164], [99] and [8] on MNIST. Then we compare Fenchel lifted networks against a fully connected neural network and LeNet-5 [102] on MNIST. Finally, we compare Fenchel lifted networks against LeNet-5 on CIFAR-10.

Fenchel lifted networks vs. lifted models

Here, we compare the non-batched Fenchel lifted network against the models proposed by [164][1], [99][2] and [8]. The former model is trained using ADMM and the latter ones using the BCD algorithms proposed in the respective papers. In Figure 3.1, we compare these models on MNIST with a 784-300-10 architecture (inspired by [102]) using a mean square error (MSE) loss.

After multiple iterations of hyperparameter search with little improvement over the base model, we chose to keep the hyperparameters for [164] and [99] as given in the code. The

[1]Code available in `https://github.com/PotatoThanh/ADMM-NeuralNetworks`

[2]Code available in `https://github.com/deeplearning-math/bcd_dnn`

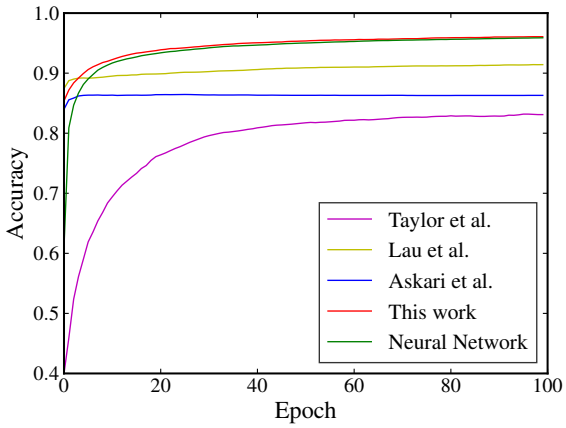


Figure 3.1: Test set performance of different lifted methods with a 784-300-10 network architecture on MNIST with a MSE loss. Final test set performances: **Taylor et al.** 0.834, **Lau et al.** 0.914, **Askari et al.** 0.863, **Neural Network** 0.957, **This work** 0.961.

hyperparameters for [8] were tuned using cross validation on a hold-out set during training. Our model used these same parameters and cross validated the remaining hyperparameters. The neural network model was trained using SGD. The resulting curve of the neural network is smoothed in Figure 3.1 for visual clarity. From Figure 3.1 it is clear that Fenchel lifted networks vastly outperform other lifted models and achieve a test set accuracy on par with traditional networks.

Fenchel lifted networks vs. neural networks on MNIST

For the same 784-300-10 architecture as the previous section, we compare the batched Fenchel lifted networks against traditional neural networks trained using first order methods. We use a cross entropy loss in the final layer for both models. The hyperparameters for our model are tuned using cross validation. Figure 3.2 shows the results.

As shown in Figure 3.2, Fenchel lifted networks learn faster than traditional networks as shown by the red curve being consistently above the blue and green curve. Although not shown, between batch 600 and 1000, the accuracy on a training batch would consistently hit 100% accuracy. The advantage of the Fenchel lifted networks is clear in the early stages of training, while towards the end the test set accuracy and the accuracy of an Adam-trained network converge to the same values.

We also compare Fenchel lifted networks against a LeNet-5 convolutional neural network on MNIST. The network architecture is 2 convolutional layers followed by 3 fully-connected layers and a cross entropy loss on the last layer. We use ReLU activations and average pooling in our implementation. Figure 3.3 plots the test set accuracy for the different models.

In Figure 3.3, our method is able to nearly converge to its final test set accuracy after

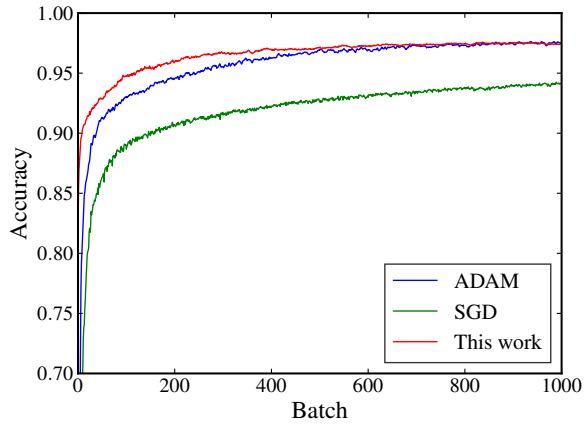


Figure 3.2: Test set performance of Fenchel lifted networks and fully connected networks trained using Adam and SGD on a 784-300-10 network architecture on MNIST with cross entropy loss. Total training time was 10 epochs. Final test set performances: **SGD** 0.943, **Adam** 0.976, **This work** 0.976.

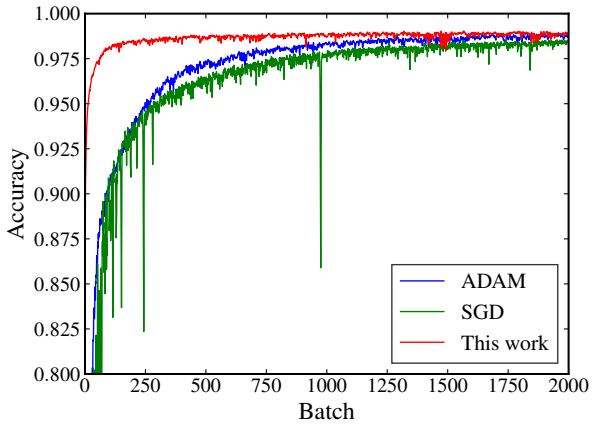


Figure 3.3: Test set performance of Fenchel lifted networks and LeNet-5 trained using Adam and SGD on MNIST with a cross entropy loss. Total training time was 20 epochs. Final test set performances: **SGD** 0.986, **Adam** 0.989, **This work** 0.990.

only 2 epochs while Adam and SGD need the full 20 epochs to converge. Furthermore, after the first few batches, our model is attaining over 90% accuracy on the test set while the other methods are only at 80%, indicating that our model is doing something different (in a positive way) compared to traditional networks, giving them a clear advantage in test set accuracy.

Fenchel lifted networks vs CNN on CIFAR-10

In this section, we compare the LeNet-5 architechture and with Fenchel lifted networks on CIFAR-10. Figure 3.4 compares the accuracies of the different models.

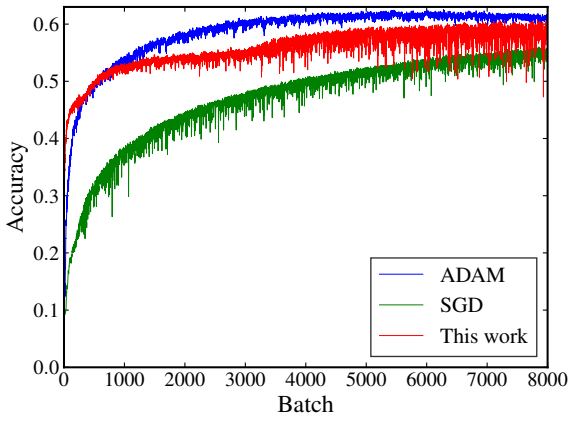


Figure 3.4: Test set performance of Fenchel lifted networks and LeNet-5 trained using Adam and SGD on CIFAR-10 with a cross entropy loss. Total training time was 80 epochs. Final test set performance: **SGD** 0.565, **Adam** 0.625, **This work** 0.606

In this case, the Fenchel lifted network still outperforms the SGD trained network and only slightly under performs compared to the Adam trained network. The larger variability in the accuracy per batch for our model can be attributed to the fact that in this experiment, when updating the W-variables, we would only take one gradient step instead of solving (3.23) and (3.24) to completion. We did this because we found empirically solving those respective sub-problems to completion would lead to poor performance at test time.

Hyperparamters for Experiments

For all experiments that used batching, the batch size was fixed at 500 and $K = 1$. We observed empirically that larger batch sizes improved the performance of the lifted models. To speed up computations, we set $K = 1$ and empirically find this does not affect final test set performance. For batched models, we do not use $\pi_l(\cdot)$ since we explicitly regularize through batching (see (3.17)) while for the non-batched models we set $\pi_l(W_l) = \|W_l\|_F^2$ for all l. For models trained using Adam, the learning rate was set to $\eta = 10^{-3}$ and for models trained using SGD, the learning rate was set to $\eta = 10^{-2}$. The learning rates were a hyperparamter that we picked from $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ to give the best final test performance for both Adam and SGD.

For the network architechtures described in the experimental results, we used the following hyperparamters:

- Fenchel Lifted Network for LeNet-5 architecture
  1. $\rho_1 = 1e-4, \lambda_1 = 5$
  2. $\rho_2 = 1e-2, \lambda_2 = 5$
  3. $\rho_3 = 1, \quad \lambda_3 = 1$
  4. $\rho_4 = 1, \quad \lambda_4 = 1$
  5. $\rho_5 = 1$
- Fenchel Lifted Network for 784-300-10 architecture (batched)
  1. $\rho_1 = 1, \quad \lambda_1 = 0.1$
  2. $\rho_2 = 100$
- Fenchel Lifted Network for 784-300-10 architecture (non-batched)
  1. $\rho_1 = 1e-2, \ \lambda_1 = 0.1$
  2. $\rho_2 = 10$

For all weights the initialization is done through Xavier initialization implemented in TensorFlow. The ρ variables are chosen to balance the change of variables across layers in iterations. Although we use a vector of λ for convenience in tuning, the theory in Section 3.4 on variable scaling states we can collapse all λ hyperparameters into a single hyperparameter. We also stress that the hyperparamter search over the ρ's were very coarse and a variety of ρ values worked well in practice; for simplicity we only present the ones we used to produce the plots in the experimental results.

3.6 Summary

In this chapter we propose Fenchel lifted networks, a family of models that provide a rigorous lower bound of the traditional neural network training problem. Fenchel lifted networks are similar to other methods that lift the dimension of the training problem, and thus exhibit many desirable properties in terms of scalability and the parallel structure of its sub-problems. As a result, we show that our family of models can be trained efficiently using block coordinate descent where the sub-problems can be parallelized across data points and/or layers. Unlike other similar lifted methods, Fenchel lifted networks are able to compete with traditional fully connected and convolutional neural networks on standard classification data sets, and in some cases are able to outperform them.

Future work will look at extending the ideas presented here to Recurrent Neural Networks, as well as exploring how to use the class of models described in the chapter to train deeper networks.

Fenchel Conjugates and Proximal Operators

Here we discuss the similarities between [108] and the approach of this work (for simplicity, we only concern ourselves with the ReLU activation since it is convex). In what follows, when we refer to equation numbers, they are the equation numbers in [108]. First we derive an elementary result relating conjugate functions and proximal operators.

Lemma 3.6.1. *Let $\lambda > 0$ and let $f(x)$ be a closed, convex and proper function. Define $\tilde{f}(x) = \lambda f(x) + \frac{1}{2}\|x\|_2^2$ and let $f^*(y)$ be the fenchel conjugate of $f(x)$. Furthermore, define the proximal operator as $\boldsymbol{prox}_{\lambda f}(x) = \arg\min_y f(y) + \frac{1}{2\lambda}\|x-y\|_2^2$ and for a given x, let $y^*(x) = \arg\max x^\top y - \tilde{f}(y)$. Then $\boldsymbol{prox}_{\lambda f}(x) = y^*(x)$.*

Proof.

$$\begin{aligned}\arg\min_y f(y) + \frac{1}{2\lambda}\|y-x\|_2^2 &= \arg\min_y f(y) + \frac{1}{2\lambda}\|x\|_2^2 - \frac{1}{\lambda}x^\top y + \frac{1}{2\lambda}\|y\|_2^2 \\ &= \arg\min_y \left(\lambda f(y) + \frac{1}{2}\|y\|_2^2\right) - x^\top y \\ &= \arg\max_y x^\top y - \tilde{f}(y)\end{aligned}$$

The left hand side is exactly $\texttt{prox}_{\lambda f}(x)$ and the right hand side is exactly $y^*(x)$. Note furthermore that the problem defining $\texttt{prox}_{\lambda f}(x)$ is strongly convex and hence there is only one unique global optima and similarly for the problem defining $y^*(x)$. ∎

The above lemma shows the natural connection between proximal operators and fenchel conjugates. We now highlight this in the case of the ReLU function $\phi(x) = \max(0, x)$ and make the connection explicit. Below we consider the scalar case, and the multivariate case is a simple generalization of the argument below.

As in [108], if we set $f(x) = \int_0^x \phi^{-1}(z) - z \; dz$ as defined below (11) and set $g(x) = \int_0^x \phi(z) - z \, dz$ as defined below (18) in the aforementioned reference and, we then have

$$\begin{aligned}f(x) &= \int_0^x \phi^{-1}(z) - z \; = 0 \\ g(x) &= \int_0^x \phi(z) - z \, dz = \frac{1}{2}\max(x, 0)^2 - \frac{1}{2}x^2\end{aligned}$$

where we use the fact that $\phi^{-1}(z) = z$ for $z \in [0, \infty)$ and set $\phi^{-1}(z) = +\infty$ otherwise. Modulo hyperparameters in their objective function, the term inside the summand in (18)

(in the scalar case), reduces to

$$\begin{aligned}
&f(x^i) + \frac{1}{2}(x^i - w^{i-1}x^{i-1})^2 + g(w^{i-1}x^{i-1}) \\
=&0 + \frac{1}{2}(x^i - w^{i-1}x^{i-1})^2 + \frac{1}{2}(w^{i-1}x^{i-1})_+^2 - \frac{1}{2}(w^{i-1}x^{i-1})^2 \\
=&\frac{1}{2}(x^i)^2 - \langle w^{i-1}x^{i-1}, x^i\rangle + \frac{1}{2}(w^{i-1}x^{i-1})_+^2 \\
=&B_\phi(x^i, w^{i-1}x^{i-1})
\end{aligned}$$

Hence

$$B_\phi(v, u) = f(v) + \frac{1}{2}\|v - u\|_2^2 + g(u)$$

As a result, the term in the summand the authors use in (18) is equivalent to the fenchel lifted formulation.

Part II

Applications of Deep Implicit Models

Chapter 4

Implicit Graph Neural Networks

After the introduction of the implicit deep learning framework, we are actively searching for applications of implicit models that enjoy unprecedented advantages compared with explicit deep learning methods and greatly outperforms them. Finding such an example will surely boost our confidence and popularize deep implicit models to the greater community. The idea of applying such a framework to graph neural networks is sparked when the author was reviewing a demonstration for information propagation process in graph neural network methods. The repeated information passage matches the design of implicit models. We then extend the implicit deep learning framework from Chapter 2 to work on graph-structured data. We find the resulting model captures long-range dependencies in graphs which are missed out by existing methods and therefore outperforms existing methods.

4.1 Introduction

Graph neural networks (GNNs) [200, 197] have been widely used on graph-structured data to obtain a meaningful representation of nodes in the graph. By iteratively aggregating information from neighboring nodes, GNN models encode graph-relational information into the representation, which then benefits a wide range of tasks, including biochemical structure discovery [67, 171], computer vision [84], and recommender systems [188]. Recently, newer convolutional GNN structures [181] have drastically improved the performance of GNNs by employing various techniques, including renormalization [91], attention [170], and simpler activation [180].

The aforemetioned modern convolutional GNN models capture relation information up to T-hops away by performing T iterations of graph convolutional aggregation. Such information gathering procedure is similar to forward-feeding schemes in popular deep learning models, such as multi-layer perceptron and convolutional neural networks. However, despite their simplicity, these computation strategies cannot discover the dependency with a range longer than T-hops away from any given node.

One approach tackling this problem is to develop recurrent GNNs that iterate graph

convolutional aggregation until convergence, without any *a priori* limitation on the number of hops. This idea arises in many traditional graph metrics, including eigenvector centrality [129] and PageRank [131], where the metrics are implicitly defined by some fixed-point equation. Intuitively, the long-range dependency can be better captured by iterating the information passing procedure for an infinite number of times until convergence. Pioneered by [69], new recurrent GNNs leverage partial training [63, 62] and approximation [47] to improve performance. With shared weights, these methods avoid exploding memory issues and achieve accuracies competitive with convolutional counterparts in certain cases.

While these methods offer an alternative to the popular convolutional GNN models with added benefits for certain problems, there are still significant limitations in evaluation and training for recurrent GNN models. Conservative convergence conditions and sophisticated training procedures have limited the use of these methods in practice, and outweighed the performance benefits of capturing the long-range dependency. In addition, most of these methods cannot leverage multi-graph information or adapt to heterogeneous network settings, as prevalent in social networks as well as bio-chemical graphs [171].

Chapter contributions. In this work, we present the **I**mplicit **G**raph **N**eural **N**etwork (IGNN) framework to address the problem of evaluation and training for recurrent GNNs. We first analyze graph neural networks through a rigorous mathematical framework based on the Perron-Frobenius theory [22], in order to establish general well-posedness conditions for convergence. We show that most existing analyses are special cases of our result. As for training, we propose a novel projected gradient method to efficiently train the IGNN, where we leverage implicit differentiation methods to obtain the exact gradient, and use projection on a tractable convex set to guarantee well-posedness. We show that previous gradient methods for recurrent graph neural networks can be interpreted as an approximation to IGNN. Further, we extend IGNN to heterogeneous network settings. Finally, we conduct comprehensive comparisons with existing methods, and demonstrate that our method effectively captures long-range dependencies and outperforms the state-of-the-art GNN models on a wide range of tasks.

Chapter outline. In Section 4.2, we give an overview of related work on GNN and implicit models. In Section 4.3, we introduce the background and notations for this chapter. Section 4.4 discusses the IGNN framework together with its well-posedness and training under both ordinary and heterogeneous settings. Section 4.5 empirically compares IGNN with modern GNN methods.

4.2 Related Work

GNN models. Pioneered by [69], GNN models have gained influence for graph-related tasks. Led by GCN [91], convolutional GNN models [170, 74, 180, 83, 35] involve a finite number of modified aggregation steps with different weight parameters. On the other hand,

recurrent GNN models [69] use the same parameters for each aggregation step and potentially enable infinite steps. [110] combines recurrent GNN with recurrent neural network structures. Methods such as Fast and Deep Graph Neural Network (FDGNN) [63, 62] use untrained recurrent GNN models with novel initialization to its aggregation step for graph classification. While the Stochastic Steady-State Embedding (SSE) method [47] uses an efficient approximated training and evaluation procedure for node classification. Recently, global method Geom-GCN [137] employs additional embedding approaches to capture global information. However, Geom-GCN [137] also belongs to convolutional-based GNNs, which struggle to capture very long range dependency due to the finite iterations they take.

Implicit Models. Implicit models are emerging structures in deep learning where the outputs are determined implicitly by a solution of some underlying sub-problem. Recent works [15] demonstrate the potential of implicit models in sequence modeling, physical engine [13] and many others [40, 4].[55] proposes a general implicit framework with the prediction rule based on the solution of a fixed-point equilibrium equation and discusses the well-posedness of the implicit prediction rule.

Oversmoothing. To catch the long-range dependency, another intuitive approach is to construct deeper convolutional GNNs by stacking more layers. However, [109] found that the learned node embeddings become indistinguishable as the convolutional GNNs get deeper. This phenomenon is called *over-smoothing*. Since then, a line of empirical [109, 38, 144] and theoretical [130, 198] works follows on the *over-smoothing* phenomenon. Unlike convolutional GNNs, IGNN adopts a different approach for long-range dependency based on recurrent GNNs and doesn't seem to suffer performance degradation as much even though it could be viewed as an infinite-layer GNN. See Section 4.5 for details.

4.3 Preliminaries

Graph neural networks take input data in the form of graphs. A graph is represented by $G = (V, E)$ where V is the set of $n := |V|$ nodes (or vertices) and $E \subseteq V \times V$ is the set of edges. In practice, we construct an adjacency matrix $A \in \mathbb{R}^{n \times n}$ to represent the graph G: for any two nodes $i, j \in V$, if $(i, j) \in E$, then $A_{ij} = 1$; otherwise, $A_{ij} = 0$. Some data sets provide additional information about the nodes in the form of a feature matrix $U \in \mathbb{R}^{p \times n}$, in which the feature vector for node i is given by $u_i \in \mathbb{R}^p$. When no additional feature information from nodes is provided, the data sets would require learning a feature matrix U separately in practice.

Given graph data, graph models produce a prediction $\hat{Y}$ to match the true label Y whose shape depends on the task. GNN models are effective in graph-structured data because they involve trainable aggregation steps that pass the information from each node to its neighboring nodes and then apply nonlinear activation. The aggregation step at iteration t

can be written as follows:

$$X^{(t+1)} = \phi(W^{(t)}X^{(t)}A + \Omega^{(t)}U), \tag{4.1}$$

where $X^{(t)} \in \mathbb{R}^{m\times n}$ stacks the state vectors of nodes in time step t into a matrix, in which the state vector for node i is denoted as $x^{(t)} \in \mathbb{R}^m$; $W^{(t)}$ and $\Omega^{(t)}$ are trainable weight matrices; ϕ is an activation function. The state vectors $X^{(T)}$ at final iteration can be used as the representation for nodes that combine input features and graph spatial information. The prediction from the GNN models is given by $\hat{Y} = f_\Theta(X^{(T)})$, where f_Θ is some trainable function parameterized by Θ. In practice, a linear f_Θ is often satisfactory.

Modern GNN approaches adopt different forms of graph convolution aggregation (4.1). Convolutional GNNs [181] iterate (4.1) with $\Omega = 0$ and set $X^{(0)} = U$. Some works temper with the adjacency matrix using renormalization [91] or attention [170]). While recurrent GNNs use explicit input from features at each step with tied weights W and Ω, some methods replace the term ΩU with $\Omega_1 UA + \Omega_2 U$, in order to account for feature information from neighboring nodes [47]. Our framework adopts a similar recurrent graph convolutional aggregation idea.

A heterogeneous network is an extended type of graph that contains different types of relations between nodes instead of only one type of edge. We continue to use $G = (V, \mathcal{E})$ to represent a heterogeneous network with the node set V and the edge set $\mathcal{E} \subseteq V \times V \times R$, where R is a set of $N := |R|$ relation types. Similarly, we define the adjacency matrices A_i, where A_i is the adjacency matrix for relation type $i \in R$. Some heterogeneous networks also have relation-specific feature matrices U_i.

Notation. For a matrix $V \in \mathbb{R}^{p\times q}$, $|V|$ denotes its absolute value (*i.e.* $|V|_{ij} = |V_{ij}|$). The infinity norm, or the max-row-sum norm, writes $\|V\|_\infty$. The 1-norm, or the max-column-sum norm, is denoted as $\|V\|_1 = \|V^\top\|_\infty$. The 2-norm is shown as $\|V\|$ or $\|V\|_2$. We use $\otimes$ to represent the Kronecker product, $\langle\cdot,\cdot\rangle$ to represent inner product and use $\odot$ to represent component-wise multiplication between two matrices of the same shape. For a $p \times q$ matrix V, $\mathbf{vec}(V) \in \mathbb{R}^{pq}$ represents the vectorized form of V, obtained by stacking its columns (See a note on Kronecker product at the end of the chapter for details). According to the Perron-Frobenius theory [22], every squared non-negative matrix M has a real non-negative eigenvalue that gives the largest modulus among all eigenvalues of M. This non-negative eigenvalue of M is called the *Perron-Frobenius (PF) eigenvalue* and denoted by $\lambda_{\text{pf}}(M)$ throughout the chapter.

4.4 Implicit Graph Neural Networks

We now introduce a framework for graph neural networks called **I**mplicit **G**raph **N**eural **N**etworks (IGNN), which obtains a node representation through the fixed-point solution of

a non-linear "equilibrium" equation. The IGNN model is formally described by

$$\hat{Y} = f_\Theta(X), \tag{4.2a}$$
$$X = \phi(WXA + b_\Omega(U)). \tag{4.2b}$$

In equation (4.2), the input feature matrix $U \in \mathbb{R}^{p\times n}$ is passed through some affine transformation $b_\Omega(\cdot)$ parametrized by Ω (*i.e.* a linear transformation possibly offset by some bias). The representation, given as the "internal state" $X \in \mathbb{R}^{m\times n}$ in the rest of the chapter, is obtained as the fixed-point solution of the equilibrium equation (4.2b), where ϕ preserves the same shape of input and output. The prediction rule (4.2a) computes the prediction $\hat{Y}$ by feeding the state X through the output function f_Θ. In practice, a linear map $f_\Theta(X) = \Theta X$ may be satisfactory.

Unlike most existing methods that iterate (4.1) for a finite number of steps, an IGNN seeks the fixed point of equation (4.2b) that is trained to give the desired representation for the task. Evaluation of fixed point can be regarded as iterating (4.1) for an infinite number of times to achieve a steady state. Thus, the final representation potentially contains information from all neighbors in the graph. In practice, this gives a better performance over the finite iterating variants by capturing the long-range dependency in the graph. Another notable benefit of the framework is that it is memory-efficient in the sense that it only maintains one current state X without other intermediate representations.

Despite its notational simplicity, the IGNN model covers a wide range of variants, including their multi-layer formulations by stacking multiple equilibrium equations similar to (4.2b). The SSE [47] and FDGNN [62] models also fit within the IGNN formulation.

Examples of IGNN

In this section we introduce some examples of the variation of IGNN.

Multi-layer Setup. It is straight forward to extend IGNN to a multi-layer setup with several sets of W and Ω parameters for each layer. For conciseness, we use the ordinary graph setting. By treating the fixed-point solution X_{l-1} of the $(l-1)$-th layer as the input U_l to the l-th layer of equilibrium equation, a multi-layer formulation of IGNN with a total of L layers is created.

$$\begin{aligned}
\hat{Y} &= f_\Theta(X_L),\\
X_L &= \phi_L(W_L X_L A + b_{\Omega_L}(X_{L-1})),\\
&\vdots\\
X_l &= \phi_l(W_l X_l A + b_{\Omega_l}(X_{l-1})),\\
&\vdots\\
X_1 &= \phi_1(W_1 X_1 A + b_{\Omega_1}(U)),
\end{aligned} \tag{4.3}$$

where $\phi_1, \ldots, \phi_L$ are activation functions. We usually assume that CONE property holds on them. And (W_l, Ω_l) is the set of weights for the l-th layer. Thus the multi-layer formulation (4.3) with parameters $(W_l, l = 1, \ldots, L,\ A)$ is well-posed (*i.e.* gives unique prediction $\hat{Y}$ for any input U) when (W_l, A) is well-posed for ϕ_l for any layer l. This is true since the well-posedness for a layer guarantees valid input for the next layer. Since all layers are well-posed, the formulation will give unique final output for any input of compatible shape. FDGNN [62] uses a similar multi-layer formulation for graph classification but is only partially trained in practive.

In terms of the affine input function, $b_\Omega(U) = \Omega UA$ is a good choice. We show that the multi-layer IGNN with such b_Ω is equivalent to a single layer IGNN (4.2) with higher dimensions, the same A matrix and f_Θ function. The new activation map is given by $\phi = (\phi_L, \ldots, \phi_l, \ldots, \phi_1)$. Although ϕ is written in a block-wise form, they still operate on entry level and remain non-expansive. Thus the well-posedness results still hold. The new $\tilde{W}$ and $\tilde{b_\Omega}$ write,

$$\tilde{W} = \begin{pmatrix} W_L & \Omega_L & & \\ & \ddots & \ddots & \\ & & \ddots & \Omega_2 \\ & & & W_1 \end{pmatrix}, \quad \tilde{b_\Omega}(U) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \Omega_1 \end{pmatrix} UA. \tag{4.4}$$

Special Cases. Many existing GNN formulations including convolutional and recurrent GNNs can be treated as special cases of IGNN. We start by showing that GCN [91], a typical example of convolutional GNNs, is indeed an IGNN. We give the matrix representation of a 2-layer GCN as follows,

$$\begin{aligned} \hat{Y} &= W_2 X_1 A, \\ X_1 &= \phi_1(W_1 UA), \end{aligned} \tag{4.5}$$

where A is the renormalized adjacency matrix; W_1 and W_2 are weight parameters; ϕ_1 is a CONE activation map for the first layer; and X_1 is the hidden representation of first layer. We show that GCN (4.5) is in fact a special case of IGNN by constructing an equivalent single layer IGNN (4.2) with the same A matrix.

$$\hat{Y} = \tilde{f_\Theta}(\tilde{X}), \tag{4.6a}$$

$$\tilde{X} = \phi(\tilde{W}\tilde{X}A + \tilde{b_\Omega}(U)). \tag{4.6b}$$

The new state $\tilde{X} = (X_2, X_1)$. The new activation map is given by $\phi = (\phi_1, \mathbb{I})$, where $\mathbb{I}$ represents an identity map. And the new $\tilde{W}$, $\tilde{b_\Omega}$, and $\tilde{f_\Theta}(\tilde{X})$ are,

$$\tilde{W} = \begin{pmatrix} 0 & W_2 \\ 0 & 0 \end{pmatrix}, \quad \tilde{b_\Omega}(U) = \begin{pmatrix} 0 \\ W_1 \end{pmatrix} UA, \quad \tilde{f_\Theta}(\tilde{X}) = \begin{pmatrix} I \\ 0 \end{pmatrix} \tilde{X}. \tag{4.7}$$

This reformulation of single layer IGNN also extends to multi-layer GCNs with more than 2 layers as well as other convolutional GNNs. Note that the new $\tilde{W}$ for the equivalent single

layer IGNN is always strictly upper triangular. Thus $|\tilde{W}|$ has only 0 eigenvalue. As a result, $\lambda_{\text{pf}}(|A^\top \otimes W|) = \lambda_{\text{pf}}(A)\lambda_{\text{pf}}(|W|) = 0$ and the sufficient condition for well-posedness is always satisfied.

Another interesting special case is SSE [47], an example of recurrent GNN, that is given by

$$\begin{aligned} \hat{Y} &= W_2 X, \\ X &= \phi(W_{1r} W_2 X A + W_{1u} U A + W'_{1u} U), \end{aligned} \tag{4.8}$$

which can be easily converted into a single layer IGNN with the same A matrix and CONE activation ϕ. The new $\tilde{W}$, $\tilde{b_\Omega}$, and $\tilde{f_\Theta}(X)$ are,

$$\tilde{W} = W_{1r} W_2, \quad \tilde{b_\Omega}(U) = W_{1u} U A + W'_{1u} U, \quad \tilde{f_\Theta}(X) = W_2 X. \tag{4.9}$$

IGNN models can generalize to heterogeneous networks with different adjacency matrices A_i and input features U_i for different relations. In that case, we have the parameters W_i and Ω_i for each relation type $i \in R$ to capture the heterogenerity of the graph. A new equilibrium equation (4.10) is used:

$$X = \phi\left(\sum_i (W_i X A_i + b_{\Omega_i}(U_i))\right). \tag{4.10}$$

In general, there may not exist a unique solution for the equilibrium equation (4.2b) and (4.10). Thus, the notion of well-posedness comes into play.

Well-posedness of IGNNs

For the IGNN model to produce a valid representation, we need to obtain some *unique* internal state $X(U)$ given any input U from equation (4.2b) for ordinary graph settings or equation (4.10) for heterogeneous network settings. However, the equilibrium equation (4.2b) and (4.10) can have no well-defined solution X given some input U. We give a simple example in the scalar setting below, where the solution to the equilibrium equation (4.2b) does not even exist.

A Scalar Example

Consider the following scalar equilibrium equation (4.11),

$$x = \text{ReLU}(wxa + u), \tag{4.11}$$

where $x, w, a, u \in \mathbb{R}$ and $\text{ReLU}(\cdot) = \max(\cdot, 0)$ is the rectified linear unit. If we set $w = a = 1$, the equation (4.11) will have no solutions for any $u > 0$. See Figure 4.1 for the example with $u = 1$.

In order to ensure the existence and uniqueness of the solution to equation (4.2b) and (4.10), we define the notion of *well-posedness* for equilibrium equations with activation ϕ for both ordinary graphs and hetergeneous networks. This notion has been introduced in [55] for ordinary implicit models.

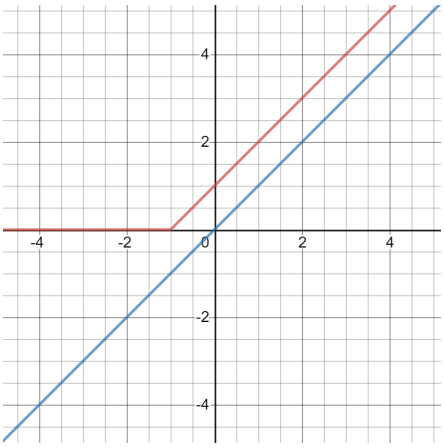


Figure 4.1: Plots of x (red plot) and $\mathrm{ReLU}(wxa+u) = \mathrm{ReLU}(x+1)$ with $w = a = u = 1$ (blue plot). The two plots will intersect at some point whenever a solution exists. However in this case the two plots have no intersections, meaning that there is no solution to equation (4.11).

Definition 4.4.1 (Well-posedness on ordinary graphs)**.** *The tuple (W, A) of the weight matrix $W \in \mathbb{R}^{m\times m}$ and the adjacency matrix $A \in \mathbb{R}^{n\times n}$ is said to be well-posed for ϕ if for any $B \in \mathbb{R}^{m\times n}$, the solution $X \in \mathbb{R}^{m\times n}$ of the following equation*

$$X = \phi(WXA + B) \tag{4.12}$$

exists and is unique.

Definition 4.4.2 (Well-posedness on heterogeneous networks)**.**
The tuple $(W_i, A_i, i = 1, \ldots, N)$ of the weight matrices $W_i \in \mathbb{R}^{m\times m}$ and the adjacency matrices $A_i \in \mathbb{R}^{n\times n}$ is said to be well-posed for ϕ if for any $B_i \in \mathbb{R}^{m\times n}$, the solution $X \in \mathbb{R}^{m\times n}$ of the following equation

$$X = \phi\left(\sum_{i=1}^{N}(W_i X A_i + B_i)\right) \tag{4.13}$$

exists and is unique.

We first develop sufficient conditions for the well-posedness property to hold on ordinary graph settings with a single edge type. The idea is to limit the structure of W and A together to ensure well-posedness for a set of activation ϕ.

In the following analysis, we assume that ϕ is component-wise non-expansive, which we refer to as the component-wise non-expansive (CONE) property. Most activation functions in deep learning satisfy the CONE property (*e.g.* Sigmoid, tanh, ReLU, Leaky ReLU, *etc.*). For simplicity, we assume that ϕ is differentiable.

We can now establish the following sufficient condition on (W, A) for our model with a CONE activation to be well-posed. Our result hinges on the notion of Perron-Frobenius

(PF) eigenvalue $\lambda_{\text{pf}}(M)$ for a non-negative matrix M, as well as the notion of Kronecker product $A \otimes B \in \mathbb{R}^{pm \times qn}$ between two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$. See a note on Kronecker product at the end of the chapter for details.

Theorem 4.4.1 (PF sufficient condition for well-posedness on ordinary graphs)**.** *Assume that ϕ is a component-wise non-expansive (CONE) activation map. Then, (W, A) is well-posed for any such ϕ if $\lambda_{pf}(|A^\top \otimes W|) < 1$. Moreover, the solution X of equation (4.12) can be obtained by iterating equation (4.12).*

Proof. Recall that for any three matrices A, W, X of compatible sizes, we have $(A^\top \otimes W)\,\mathbf{vec}(X) = \mathbf{vec}(WXA)$ [146]. Showing equation (4.12) has an unique solution is equivalent to showing that the following "vectorized" equation has a unique solution:

$$\mathbf{vec}(X) = \phi(A^\top \otimes W\,\mathbf{vec}(X) + \mathbf{vec}(B))$$

It follows directly from Lemma 4.4.1 that if $\lambda_{\text{pf}}(|A^\top \otimes W|) = \lambda_{\text{pf}}(A)\lambda_{\text{pf}}(|W|) < 1$, then the above equation has unique solution that can be obtained by iterating the equation. ∎

Lemma 4.4.1. *If ϕ is component-wise non-negative (CONE), M is some squared matrix and v is any real vector of compatible shape, the equation $x = \phi(Mx + v)$ has a unique solution if $\lambda_{pf}(|M|) < 1$. And the solution can be obtained by iterating the equation. Hence, $x = \lim_{t\to\infty} x_t$.*

$$x_{t+1} = \phi(Mx_t + v),\ x_0 = 0,\ t = 0, 1, \ldots \tag{4.14}$$

Proof. For existence, since ϕ is component-wise and non-expansive, we have that for $t \geq 1$ and the sequence $x_0, x_1, x_2, \ldots$ generated from iteration (4.14),

$$|x_{t+1} - x_t| = |\phi(Mx_t + v) - \phi(Mx_{t-1} + v)| \leq |M(x_t - x_{t-1})| \leq |M||x_t - x_{t-1}|.$$

For $n > m \geq 1$, the following inequality follows,

$$|x_n - x_m| \leq |M|^m \sum_{i=0}^{n-m-1} |M|^i |x_1 - x_0| \leq |M|^m \sum_{i=0}^{\infty} |M|^i |x_1 - x_0| \leq |M|^m w, \tag{4.15}$$

where

$$w := \sum_{i=0}^{\infty} |M|^i |x_1 - x_0| = (I - |M|)^{-1} |x_1 - x_0|.$$

Because $\lambda_{\text{pf}}(|M|) < 1$, the inverse of $I - |M|$ exists. It also follows that $\lim_{t\to\infty} |M|^t = 0$. From inequality (4.15), we show that the sequence $x_0, x_1, x_2, \ldots$ is a Cauchy sequence because $0 \leq \lim_{m\to\infty} |x_n - x_m| \leq \lim_{m\to\infty} |M|^m w = 0$. And thus the sequence converges to some solution of $x = \phi(Mx + v)$.

For uniqueness, suppose both x_a and x_b satisfy $x = \phi(Mx + v)$, then the following inequality holds,

$$0 \leq |x_a - x_b| \leq |M||x_a - x_b| \leq \lim_{t\to\infty} |M|^t |x_a - x_b| = 0.$$

It follows that $x_a = x_b$ and there exists unique solution to $x = \phi(Mx + v)$. ∎

We find Theorem 4.4.1 so general that many familiar and interesting results will follow from it, as discussed in the following remarks.

Remark 4.4.1. *For some non-negative adjacency matrix A, and arbitrary real parameter matrix W, $\lambda_{pf}(|A^\top \otimes W|) = \lambda_{pf}(A^\top \otimes |W|) = \lambda_{pf}(A)\lambda_{pf}(|W|)$.*

The final equality of the above remark follows from the fact that, the spectrum of the Kronecker product of matrix A and B satisfies that $\Delta(A \otimes B) = \{\mu\lambda : \mu \in \Delta(A),\ \lambda \in \Delta(B)\}$, where $\Delta(A)$ represents the spectrum of matrix A. And that, the left and right eigenvalues of a matrix are the same.

Remark 4.4.2 (Contraction sufficient condition for well-posedness [69])**.** *For any component-wise non-expansive (CONE) ϕ, if $\mathcal{A}(X) = \phi(WXA + B)$ is a contraction of X (w.r.t. vectorized norms), then (W, A) is well-posed for ϕ.*

The above remark follows from the fact that the contraction condition for any CONE activation map is equivalent to $\|A^\top \otimes W\| < 1$, which implies $\lambda_{\text{pf}}(|A^\top \otimes W|) < 1$.

Remark 4.4.3 (Well-posedness for directed acyclic graph)**.** *For a directed acyclic graph (DAG), let A be its adjacency matrix. For any real squared W, it holds that (W, A) is well-posed for every CONE activation map. Note that $\mathcal{A}(X) = \phi(WXA + B)$ need not be a contraction of X.*

Note that for DAG, A is nilpotent ($\lambda_{\text{pf}}(A) = 0$) and thus $\lambda_{\text{pf}}(|A^\top \otimes W|) = \lambda_{\text{pf}}(A)\lambda_{\text{pf}}(|W|) = 0$.

Remark 4.4.4 (Sufficient well-posedness condition for k-regular graph [62])**.** *For a k-regular graph, let A be its adjacency matrix. (W, A) is well-posed for every CONE activation map if $k\|W\|_2 < 1$.*

It follows from that for a k-regular graph, the PF eigenvalue of the adjacency matrix $\lambda_{\text{pf}}(A) = k$. And $\lambda_{\text{pf}}(A)\lambda_{\text{pf}}(|W|) \leq k\|W\|_2 < 1$ guarantees well-posedness.

A similar sufficient condition for well-posedness holds for heterogeneous networks.

Theorem 4.4.2 (PF sufficient condition for well-posedness on heterogeneous networks)**.** *Assume that ϕ is some component-wise non-expansive (CONE) activation map. Then, $(W_i, A_i,\ i = 1, \ldots, N)$ is well-posed for any such ϕ if $\lambda_{pf}\left(\sum_{i=1}^N |A_i^\top \otimes W_i|\right) < 1$. Moreover, the solution X of equation (4.13) can be obtained by iterating equation (4.13).*

Proof. Similarly, we can rewrite equation (4.13) into the following "vectorized" form.

$$\mathbf{vec}(X) = \phi\left(\sum_{i=1}^N (A_i^\top \otimes W_i)\,\mathbf{vec}(X) + \sum_{i=1}^N \mathbf{vec}(B_i)\right)$$

It follows from a similar scheme as the proof of Lemma 4.4.1 that if $\lambda_{\text{pf}}\left(\sum_{i=1}^{N}|A_i^\top \otimes W_i|\right) < 1$, the above equation has unique solution which can be obtained by iterating the equation. ■

Sufficient conditions in Theorems 4.4.1 and 4.4.2 guarantee convergence when iterating aggregation step to evaluate state X. Furthermore, these procedures enjoy exponential convergence in practice.

Tractable Well-posedness Condition for Training

At training time, however, it is difficult in general to ensure satisfaction of the PF sufficient condition $\lambda_{\text{pf}}(|W|)\lambda_{\text{pf}}(A) < 1$, because $\lambda_{\text{pf}}(|W|)$ is non-convex in W. To alleviate the problem, we give a numerically tractable convex condition for well-posedness that can be enforced at training time efficiently through projection. Instead of using $\lambda_{\text{pf}}(|W|) < \lambda_{\text{pf}}(A)^{-1}$, we enforce the stricter condition $\|W\|_\infty < \lambda_{\text{pf}}(A)^{-1}$, which guarantees the former inequality by $\lambda_{\text{pf}}(|W|) \le \|W\|_\infty$. Although $\|W\|_\infty < \lambda_{\text{pf}}(A)^{-1}$ is a stricter condition, we show in the following theorem that it is equivalent to the PF condition for positively homogeneous activation functions, (*i.e.* $\phi(\alpha x) = \alpha\phi(x)$ for any $\alpha \ge 0$ and x), in the sense that one can use the former condition at training without loss of generality.

Theorem 4.4.3 (Rescaled IGNN)**.** *Assume that ϕ is CONE and positively homogeneous. For an IGNN $(f_\Theta, W, A, b_\Omega, \phi)$ where (W, A) satisfies the PF sufficient condition for well-posedness, namely $\lambda_{pf}(|W|) < \lambda_{pf}(A)^{-1}$, there exists a linearly-rescaled equivalent IGNN $(\tilde{f_\Theta}, W', A, \tilde{b_\Omega}, \phi)$ with $\|W'\|_\infty < \lambda_{pf}(A)^{-1}$ that gives the same output $\hat{Y}$ as the original IGNN for any input U.*

Proof. Similarly, we can rewrite equation (4.13) into the following "vectorized" form.

$$\mathbf{vec}(X) = \phi\left(\sum_{i=1}^{N}(A_i^\top \otimes W_i)\,\mathbf{vec}(X) + \sum_{i=1}^{N}\mathbf{vec}(B_i)\right)$$

It follows from a similar scheme as the proof of Lemma 4.4.1 that if $\lambda_{\text{pf}}\left(\sum_{i=1}^{N}|A_i^\top \otimes W_i|\right) < 1$, the above equation has unique solution which can be obtained by iterating the equation. ■

The above-mentioned condition can be enforced by selecting a $\kappa \in [0, 1)$ and projecting the updated W onto the convex constraint set $\mathcal{C} = \{W : \|W\|_\infty \le \kappa/\lambda_{\text{pf}}(A)\}$.

For heterogeneous network settings, we recall the following:

Remark 4.4.5. *For any non-negative adjacency matrix A and arbitrary real parameter matrix W, it holds that $\|A^\top \otimes W\|_\infty = \|A^\top\|_\infty\|W\|_\infty = \|A\|_1\|W\|_\infty$.*

The above remark follows from the facts that, $\|\cdot\|_\infty$ (resp. $\|\cdot\|_1$) gives maximum row (resp. column) sum of the absolute values of a given matrix. And that, for some real matrices A and B, $\|A\otimes B\|_\infty = \max_{i,j}\left(\sum_{k,l}|A_{ik}B_{jl}|\right) = \max_{i,j}\left(\sum_k|A_{ik}|\ \sum_l|B_{jl}|\right) = \max_i\left(\sum_k|A_{ik}|\right)\ \max_j\left(\sum_l|B_{jl}|\right) = \|A\|_\infty\|B\|_\infty$.

Similar to the difficulty faced in the ordinary graph settings, ensuring the PF sufficient condition on heterogeneous networks is hard in general. We propose to enforce the following tractable condition that is convex in W_i's: $\sum_{i=1}^N \|A_i\|_1\|W_i\|_\infty \le \kappa < 1$, $\kappa\in[0,1)$. Note that this condition implies $\left\|\sum_{i=1}^N A_i^\top\otimes W_i\right\|_\infty \le \kappa$, and thus $\lambda_{\rm pf}\left(\sum_{i=1}^N|A_i^\top\otimes W_i|\right)\le\kappa<1$. The PF sufficient condition for well-posedness on heterogeneous networks is then guaranteed.

Training of IGNN

We start by giving the training problem (4.16), where a loss $\mathcal{L}(Y,\hat{Y})$ is minimized to match $\hat{Y}$ to Y and yet the tractable condition $\|W\|_\infty \le \kappa/\lambda_{\rm pf}(A)$ for well-posedness is enforced with $\kappa\in[0,1)$:

$$\min_{\Theta,W,\Omega}\ \mathcal{L}(Y,f_\Theta(X))\ :\ \ X=\phi(WXA+b_\Omega(U)),\ \ \|W\|_\infty\le\kappa/\lambda_{\rm pf}(A). \tag{4.16}$$

The problem can be solved by projected gradient descent (involving a projection to the well-posedness condition following a gradient step), where the gradient is obtained through an implicit differentiation scheme. From the chain rule, one can easily obtain $\nabla_\Theta\mathcal{L}$ for the parameter of f_Θ and $\nabla_X\mathcal{L}$ for the internal state X. In addition, we can write the gradient with respect to scalar $q\in W\cup\Omega$ as follows:

$$\nabla_q\mathcal{L} = \left\langle \frac{\partial\left(WXA+b_\Omega(U)\right)}{\partial q}, \nabla_Z\mathcal{L}\right\rangle, \tag{4.17}$$

where $Z = WXA+b_\Omega(U)$ *assuming fixed* X (see Section 4.4). Here, $\nabla_Z\mathcal{L}$ is given as a solution to the equilibrium equation

$$\nabla_Z\mathcal{L} = D\odot\left(W^\top\nabla_Z\mathcal{L}\ A^\top+\nabla_X\mathcal{L}\right), \tag{4.18}$$

where $D=\phi'(WXA+b_\Omega(U))$ and $\phi'(z)=d\phi(z)/dz$ refers to the element-wise derivative of the CONE map ϕ. Since ϕ is non-expansive, it is 1-Lipschitz (*i.e.* the absolute value of $d\phi(z)/dz$ is not greater than 1), the equilibrium equation (4.18) for gradient $\nabla_Z\mathcal{L}$ admits a *unique* solution by iterating (4.18) to convergence, if (W,A) is well-posed for any CONE activation ϕ. (Note that $D\odot(\cdot)$ can be seen as a CONE map with each entry of D having absolute value less than or equal to 1.) Again, $\nabla_Z\mathcal{L}$ can be efficiently obtained due to exponential convergence when iterating (4.18) in practice.

Once $\nabla_Z\mathcal{L}$ is obtained, we can use the chain rule (via autograd software) to easily compute $\nabla_W\mathcal{L}$, $\nabla_\Omega\mathcal{L}$, and possibly $\nabla_U\mathcal{L}$ when input U requires gradients (*e.g.* in cases of features learning or multi-layer formulation). The deriviation has a deep connection to the Implicit Function Theorm. See Section 4.4 for details.

Due to the norm constraint introduced for well-posedness, each update to W requires a projection step (See Section 4.4). The new W is given by
$W^+ = \pi_{\mathcal{C}}(W) := \min_{\|M\|_\infty \le \kappa/\lambda_{\text{pf}}(A)} \|M - W\|_F^2$, where $\pi_{\mathcal{C}}$ is the projection back onto $\mathcal{C} = \{\|W\|_\infty \le \kappa/\lambda_{\text{pf}}(A)\}$. The projection is decomposible across the rows of W. Each subproblem will be a projection onto an $\mathcal{L}_1$-ball for which efficient methods exist [53]. A similar projected gradient descent training scheme for heterogeneous network settings is detailed in Section 4.4. Note that the gradient method in SSE [47] uses a first-order approximated solution to equation (4.18). FDGNN [62] only updates Θ at training using gradient descent.

Implicit differentiation for IGNN

To compute gradient of $\mathcal{L}$ from the training problem (4.16) w.r.t. a scalar $q \in W \cup \Omega$, we can use chain rule. It follows that,

$$\nabla_q \mathcal{L} = \left\langle \frac{\partial X}{\partial q}, \nabla_X \mathcal{L} \right\rangle, \tag{4.19}$$

where $\nabla_X \mathcal{L}$ can be easily calculated through modern autograd frameworks. But $\frac{\partial X}{\partial q}$ is nontrivial to obtain because X is only implicitly defined. Fortunately, we can still leverage chain rule in this case by carefully taking the "implicitness" into account.

To avoid taking derivatives of matrices by matrices, we again introduce the vectorized representation $\mathbf{vec}(\cdot)$ of matrices. The vectorization of a matrix $X \in \mathbb{R}^{m\times n}$, denoted $\mathbf{vec}(X)$, is obtained by stacking the columns of X into one single column vector of dimension mn. For simplicity, we use $\vec{X} := \mathbf{vec}(X)$ and $\nabla_{\vec{X}} \mathcal{L} = \mathbf{vec}(\nabla_X \mathcal{L})$ as a short hand notation of vectorization.

$$\frac{\partial \vec{X}}{\partial q} = \frac{\partial \vec{X}}{\partial \vec{Z}} \cdot \frac{\partial \vec{Z}}{\partial q}, \tag{4.20}$$

where $Z = WXA + b_\Omega(U)$ ($\vec{Z} = (A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}$) *assuming fixed* X. Unlike X in equation (4.2b), Z is not implicitly defined and should only be considered as a closed evaluation of $Z = WXA + b_\Omega(U)$ assuming X doesn't change depending on Z. In some sense, the Z in equation (4.20) doesn't equal to $WXA + b_\Omega(U)$. However, the closeness property will greatly simplify the evaluation of $\frac{\partial \vec{Z}}{\partial q}$. It turns out that we can still employ chain rule in this case to calculate $\frac{\partial \vec{X}}{\partial \vec{Z}}$ for such Z by taking the change of X before hand into account as follows,

$$\frac{\partial \vec{X}}{\partial \vec{Z}} = \frac{\partial \phi(\vec{Z})}{\partial \vec{Z}} + \frac{\partial \phi\left((A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}\right)}{\partial \vec{X}} \cdot \frac{\partial \vec{X}}{\partial \vec{Z}}, \tag{4.21}$$

where the second term accounts for the change in X that was ignored in Z. Another way to view this calculation is to right multiply $\frac{\partial \vec{Z}}{\partial q}$ on both sides of equation (4.21), which gives

the chain rule evaluation of $\frac{\partial \vec{X}}{\partial q}$ that takes the gradient flowing back to X into account:

$$\frac{\partial \vec{X}}{\partial q} = \frac{\partial \phi\left((A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}\right)}{\partial q} + \frac{\partial \phi\left((A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}\right)}{\partial \vec{X}} \cdot \frac{\partial \vec{X}}{\partial q}.$$

The equation (4.21) can be simplified as follows,

$$\frac{\partial \vec{X}}{\partial \vec{Z}} = (I - J)^{-1}\tilde{D}, \tag{4.22}$$
$$J = \frac{\partial \phi\left((A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}\right)}{\partial \vec{X}} = \tilde{D}(A^\top \otimes W),$$

where $\tilde{D} = \frac{\partial \phi(\vec{Z})}{\partial \vec{Z}} = \mathbf{diag}\left(\phi'\left((A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}\right)\right)$. Now we can rewrite equation (4.19) as

$$\nabla_q \mathcal{L} = \left\langle \frac{\partial \vec{Z}}{\partial q}, \nabla_{\vec{Z}} \mathcal{L} \right\rangle, \tag{4.23}$$

$$\nabla_{\vec{Z}} \mathcal{L} = \left(\frac{\partial \vec{X}}{\partial \vec{Z}}\right)^\top \nabla_{\vec{X}} \mathcal{L}, \tag{4.24}$$

which is equivalent to equation (4.17). $\nabla_{\vec{Z}} \mathcal{L}$ should be interpreted as the direction of steepest change of $\mathcal{L}$ for $Z = WXA + b_\Omega(U)$ *assuming fixed* X. Plugging equation (4.21) to (4.24), we arrive at the following equilibrium equation (equivalent to equation (4.18))

$$\begin{aligned} \nabla_{\vec{Z}} \mathcal{L} &= \tilde{D}(A \otimes W^\top)\nabla_{\vec{Z}} \mathcal{L} + \tilde{D}\, \nabla_{\vec{X}} \mathcal{L}, \\ \nabla_Z \mathcal{L} &= D \odot \left(W^\top \nabla_Z \mathcal{L} A^\top + \nabla_X \mathcal{L}\right), \end{aligned} \tag{4.25}$$

where $D = \phi'(WXA + b_\Omega(U))$. Interestingly, $\nabla_Z \mathcal{L}$ turns out to be given as a solution of an equilibrium equation particularly similar to equation (4.2b) in the IGNN "forward" pass. In fact, we can see element-wise multiplication with D as a CONE "activation map" $\tilde{\phi}(\cdot) = D \odot (\cdot)$. And it follows from Section 4.4 that if $\lambda_{\text{pf}}(W)\lambda_{\text{pf}}(A) < 1$, then $\lambda_{\text{pf}}(W^\top)\lambda_{\text{pf}}(A^\top) < 1$ and $\nabla_Z \mathcal{L}$ can be uniquely determined by iterating the above equation (4.25). Although the proof will be more involved, if (W, A) is well-posed for any CONE activation map, we can conclude that equilibrium equation (4.25) is also well-posed for $\tilde{\phi}$ where ϕ can be any CONE activation map.

Finally, by plugging the evaluated $\nabla_Z \mathcal{L}$ into equation (4.23), we get the desired gradients. Note that it is also possible to obtain gradient $\nabla_U \mathcal{L}$ by setting the q in the above calculation to be $q \in U$. This is valid because we have no restrictions on selection of q other than that it is not X, which is assumed fixed. Following the chain rule, we can give the closed form formula for $\nabla_W \mathcal{L}$, $\nabla_\omega \mathcal{L}, \omega \in \Omega$, and $\nabla_u \mathcal{L}, u \in U$.

$$\nabla_W \mathcal{L} = \nabla_Z \mathcal{L}\, A^\top X^\top, \quad \nabla_\omega \mathcal{L} = \left\langle \frac{\partial b_\Omega(U)}{\partial \omega}, \nabla_Z \mathcal{L} \right\rangle, \quad \nabla_u \mathcal{L} = \left\langle \frac{\partial b_\Omega(U)}{\partial u}, \nabla_Z \mathcal{L} \right\rangle.$$

Heterogeneous Network Setting We start by giving the training problem for heterogeneous networks similar to training problem (4.16) for ordinary graphs,

$$\begin{aligned} \min_{\Theta, W, \Omega} \;& \mathcal{L}(Y, f_\Theta(X)) \\ \text{s.t. } & X = \phi\left(\sum_{i=1}^{N} (W_i X A_i + b_{\Omega_i}(U_i))\right), \\ & \sum_{i=1}^{N} \|A_i\|_1 \|W_i\|_\infty \le \kappa. \end{aligned} \tag{4.26}$$

The training problem can be solved again using projected gradient descent method where the gradient of W_i and Ω_i for $i \in R$ can be obtained with implicit differentiation. Using chain rule, we write the gradient of a scalar $q \in \bigcup_i (W_i \cup \Omega_i)$,

$$\nabla_q \mathcal{L} = \left\langle \frac{\partial \left(\sum_{i=1}^{N} (W_i X A_i + b_{\Omega_i}(U_i))\right)}{\partial q}, \nabla_Z \mathcal{L} \right\rangle, \tag{4.27}$$

where $Z = \sum_{i=1}^{N} (W_i X A_i + b_{\Omega_i}(U_i))$ and $\nabla_Z \mathcal{L}$ in equation (4.27) should be interpreted as "direction of fastest change of $\mathcal{L}$ for Z *assuming fixed* X". Similar to the derivation in ordinary graphs setting, such notion of $\nabla_Z \mathcal{L}$ enables convenient calculation of $\nabla_q \mathcal{L}$. And the vectorized gradient w.r.t. Z can be expressed as a function of the vectorized gradient w.r.t. X:

$$\nabla_{\vec{Z}} \mathcal{L} = \left(\frac{\partial \vec{X}}{\partial \vec{Z}}\right)^\top \nabla_{\vec{X}} \mathcal{L} \tag{4.28}$$

$$\begin{aligned} \frac{\partial \vec{X}}{\partial \vec{Z}} &= \frac{\partial \phi(\vec{Z})}{\partial \vec{Z}} + \frac{\partial \phi\left(\sum_{i=1}^{N} \left((A_i^\top \otimes W_i)\vec{X} + \overrightarrow{b_{\Omega_i}(U_i)}\right)\right)}{\partial \vec{X}} \cdot \frac{\partial \vec{X}}{\partial \vec{Z}} \\ &= (I - J)^{-1} \tilde{D} \end{aligned} \tag{4.29}$$

$$J = \frac{\partial \phi\left(\sum_{i=1}^{N} \left((A_i^\top \otimes W_i)\vec{X} + \overrightarrow{b_{\Omega_i}(U_i)}\right)\right)}{\partial \vec{X}} = \tilde{D} \sum_{i=1}^{N} (A_i^\top \otimes W_i),$$

where $\tilde{D} = \frac{\partial \phi(\vec{Z})}{\partial \vec{Z}} = \mathbf{diag}\left(\phi'\left(\sum_{i=1}^{N} \left((A_i^\top \otimes W_i)\vec{X} + \overrightarrow{b_{\Omega_i}(U_i)}\right)\right)\right)$. Plugging the expression (4.29) into (4.28), we arrive at the following equilibrium equation for $\nabla_{\vec{Z}} \mathcal{L}$ and

$\nabla_Z \mathcal{L}$,

$$\nabla_{\vec{Z}} \mathcal{L} = \tilde{D} \sum_{i=1}^{N} (A_i \otimes W_i^\top) \nabla_{\vec{Z}} \mathcal{L} + \tilde{D}\, \nabla_{\vec{X}} \mathcal{L}$$
$$\nabla_Z \mathcal{L} = D \odot \left(\sum_{i=1}^{N} (W_i^\top \nabla_Z \mathcal{L} A_i^\top) + \nabla_X \mathcal{L} \right), \tag{4.30}$$

where $D = \phi' \left(\sum_{i=1}^{N} (W_i X A_i + b_{\Omega_i}(U_i)) \right)$. Not surprisingly, the equilibrium equation (4.30) again appears to be similar to the equation (4.10) in the IGNN "forward" pass. We can also view element-wise multiplication with D as a CONE "activation map" $\tilde{\phi}(\cdot) = D \odot (\cdot)$. And it follows from Section 4.4 that if $\lambda_{\text{pf}}(|A^\top \otimes W|) < 1$, then $\lambda_{\text{pf}}(|A \otimes W^\top|) < 1$ and $\nabla_Z \mathcal{L}$ can be uniquely determined by iterating the above equation (4.25). It also holds that if $(W_i, A_i, i \in \{1, \ldots, N\})$ is well-posed for any CONE activation ϕ, then we can conclude that equilibrium equation (4.30) is also well-posed for $\tilde{\phi}$ where ϕ can be any CONE activation map.

Finally, by plugging the evaluated $\nabla_Z \mathcal{L}$ into equation (4.27), we get the desired gradients. It is also possible to obtain gradient $\nabla_{U_i} \mathcal{L}$ by setting the q in the above calculation to be $q \in \bigcup_i U_i$. This is valid because we have no restrictions on selection of q other than that it is not X, which is assumed fixed.

After the gradient step, the projection to the tractable condition mentioned in Section 4.4 can be done approximately by assigning κ_i for each relation $i \in R$ and projecting W_i onto $\mathcal{C}_i = \{\|W_i\|_\infty \leq \kappa_i / \|A\|_1\}$. Ensuring $\sum_i \kappa_i = \kappa < 1$ will guarantee that the PF condition for heterogeneous network is satisfied. However, empirically, setting $\kappa_i < 1$ with $\sum_i \kappa_i > 1$ in some cases is enough for the convergence property to hold for the equilibrium equations.

4.5 Numerical Experiment

In this section, we demonstrate the capability of IGNN on effectively learning a representation that captures the long-range dependency and therefore offers the state-of-the-art performance on both synthetic and real-world data sets. More specifically, we test IGNN against a selected set of baselines on 6 node classification data sets (Chains, PPI, AMAZON, ACM, IMDB, DBLP) and 5 graph classification data sets (MUTAG, PTC, COX2, PROTEINS, NC11), where Chains is a synthetic data set; PPI and AMAZON are multi-label classification data sets; ACM, IMDB and DBLP are based on heterogeneous networks. We inherit the same experimental settings and reuse the results of baselines from literatures in some of the data sets. The test set performance is reported. Detailed description of the data sets, our preprocessing procedure, hyper-parameters, and other information of experiments can be found in Section 4.5.

Synthetic Chains Data Set. To evaluate GNN's capability for capturing the underlying long-range dependency in graphs, we create the Chains data set where the goal is to classify nodes in a chain of length l. The information of the class is only sparsely provided as the feature in an end node. We use a small training set, validation set, and test set with only 20, 100, and 200 nodes, respectively. For simplicity, we only consider the binary classification task. Four representative baselines are implemented and compared. We show in Figure 4.2 that IGNN and SSE [47] both capture the long-range dependency with IGNN offering a better performance for longer chains, while finite-iterating convolutional GNNs with $T = 2$, including GCN [91], SGC [180] and GAT [170], fail to give meaningful predictions when the chains become longer. However, selecting a larger T for convolutional GNNs does not seem to help in this case of limited training data. We further discuss this aspect in Section 4.5.

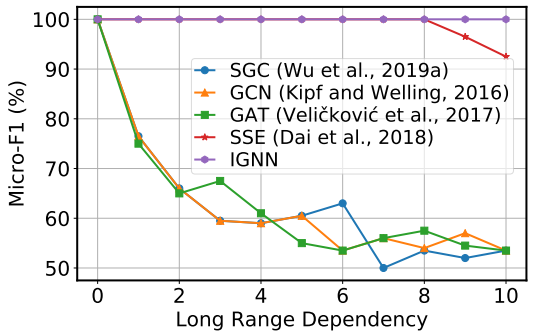


Figure 4.2: Micro-F_1 (%) performance with respect to the length of the chains.

Table 4.1: Multi-label node classification Micro-F_1 (%) performance on PPI data set.

Method	Micro-F_1 /%
Multi-Layer Perceptron	46.2
GCN [91]	59.2
GraphSAGE [74]	78.6
SSE [47]	83.6
GAT [170]	97.3
IGNN	**97.6**

Node Classification. The popular benchmark data set Protein-Protein Interaction (PPI) models the interactions between proteins using a graph, with nodes being proteins and edges being interactions. Each protein can have at most 121 labels and be associated with additional 50-dimensional features. The train/valid/test split is consistent with GraphSage [74]. We report the micro-averaged F_1 score of a multi-layer IGNN against other popular baseline models. The results can be found in Table 4.1. By capturing the underlying long-range dependency between proteins, the IGNN achieves the best performance compared to other baselines.

To further manifest the scalability of IGNN towards larger graphs, we conduct experiments on a large multi-label node classification data set, namely the Amazon product co-purchasing network data set [185] [1]. The data set renders products as nodes and co-purchases as edges but provides no input features. 58 product types with more than 5,000 products are selected from a total of 75,149 product types. While holding out 10% of the total nodes as test set, we vary the training set fraction from 5% to 9% to be consistent with [47]. The

[1] http://snap.stanford.edu/data/#amazon

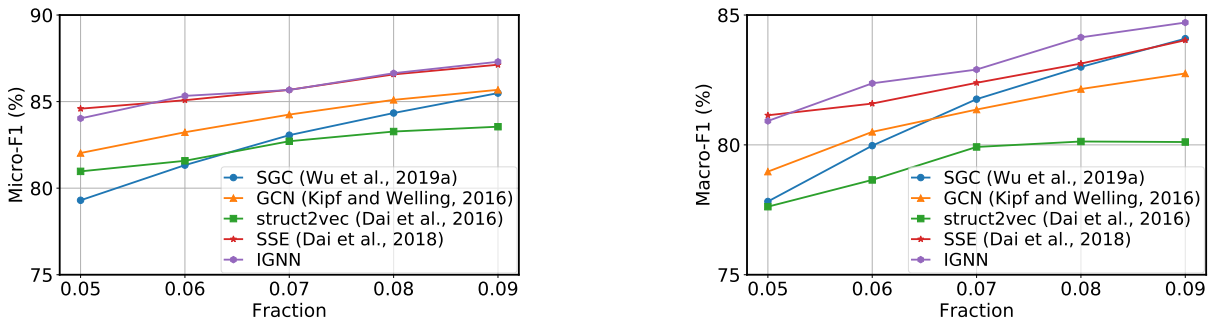


Figure 4.3: Micro/Macro-F_1 (%) performance on the multi-label node classification task with Amazon product co-purchasing network data set.

data set come with no input feature vectors and thus require feature learning at training. Both Micro-F_1 and Macro-F_1 are reported on the held-out test set, where we compare IGNN with a set of baselines consistent with those in the synthetic data set. However, we use struct2vec [46] as an alternative to GAT since GAT faces a severe out-of-memory issue in this task.

As shown in Figure 4.3, IGNN again outperforms the baselines in most cases, especially when the amount of supervision grows. When more labels are available, more high-quality feature vectors of the nodes are learned and this enables the discovery of more long-range dependency. This phenomenon is aligned with our observation that IGNN achieves a better performance when there is more long-range dependency in the underlying graph.

Graph Classification. Aside from node classification, we test IGNN on graph classification tasks. A total of 5 bioinformatics benchmarks are chosen: MUTAG, PTC, COX2, NCI1 and PROTEINS [184]. See details of data sets in Section 4.5. Under the graph classification setting, we compare a multi-layer IGNN with a comprehensive set of baselines, including a variety of GNNs and a number of graph kernels. Following identical settings as [184, 182], 10-fold cross-validation with LIB-SVM [34] is conducted. The average prediction accuracy and standard deviations are reported in Table 4.2. In this experiment, IGNN achieves the best performance in 4 out of 5 experiments given the competitive baselines. Such performance further validates IGNN's success in learning converging aggregation steps that capture long-range dependencies when generalized to unseen testing graphs.

Heterogeneous Networks. Following our theoretical analysis on heterogeneous networks, we investigate how IGNN takes advantage of heterogeneity on node classification tasks. Three benchmarks based on heterogeneous network are chosen, *i.e.*, ACM, IMDB and DBLP [174, 134]. More information regarding the heterogeneous network data sets can be found in Section 4.5. Table 4.3 compares IGNN against a set of state-of-the-art GNN baselines for heterogeneous networks. The heterogeneous variant of IGNN continues to offer a competitive performance on all 3 data sets where IGNN gives the best performance in ACM and IMDB

Table 4.2: Graph classification accuracy (%). Results are averaged (and std are computed) on the outer 10 folds.

Data sets	MUTAG	PTC	COX2	PROTEINS	NCI1
# graphs	188	344	467	1113	4110
Avg # nodes	17.9	25.5	41.2	39.1	29.8
DGCNN [193]	85.8	58.6	–	75.5	74.4
DCNN [12]	67.0	56.6	–	61.3	62.6
GK [151]	81.4 ± 1.7	55.7 ± 0.5	–	71.4 ± 0.3	62.5 ± 0.3
RW [66]	79.2 ± 2.1	55.9 ± 0.3	–	59.6 ± 0.1	–
PK [128]	76.0 ± 2.7	59.5 ± 2.4	81.0 ± 0.2	73.7 ± 0.7	82.5 ± 0.5
WL [152]	84.1 ± 1.9	58.0 ± 2.5	83.2 ± 0.2	74.7 ± 0.5	$\mathbf{84.5 \pm 0.5}$
FDGNN [62]	88.5 ± 3.8	63.4 ± 5.4	83.3 ± 2.9	76.8 ± 2.9	77.8 ± 1.6
GCN [91]	85.6 ± 5.8	64.2 ± 4.3	–	76.0 ± 3.2	80.2 ± 2.0
GIN [182]	89.0 ± 6.0	63.7 ± 8.2	–	75.9 ± 3.8	82.7 ± 1.6
IGNN	$\mathbf{89.3 \pm 6.7}$	$\mathbf{70.1 \pm 5.6}$	$\mathbf{86.9 \pm 4.0}$	$\mathbf{77.7 \pm 3.4}$	80.5 ± 1.9

data sets. While on DBLP, IGNN underperforms DMGI but still outperforms other baselines by large margin. Good performance on heterogeneous networks demonstrates the flexibility of IGNN on handling heterogeneous relationships.

Table 4.3: Node classification Micro/Macro-F_1 (%) performance on heterogeneous network data sets.

Data sets	ACM		IMDB		DBLP	
Metric	Micro-F_1	Macro-F_1	Micro-F_1	Macro-F_1	Micro-F_1	Macro-F_1
DGI [169]	88.1	88.1	60.6	59.8	72.0	72.3
GCN/GAT	87.0	86.9	61.1	60.3	71.7	73.4
DeepWalk [138]	74.8	73.9	55.0	53.2	53.7	53.3
mGCN [116]	86.0	85.8	63.0	62.3	71.3	72.5
HAN [174]	87.9	87.8	60.7	59.9	70.8	71.6
DMGI [134]	89.8	89.8	64.8	64.8	**76.6**	**77.1**
IGNN	**90.5**	**90.6**	**65.5**	**65.5**	73.8	75.1

More on Experiments

In this section, we give detailed information about the experiments we conduct.

For preprocessing, we apply the *renormalization trick* consistent with GCN [91] on the adjacent matrix of all data sets.

In terms of hyperparameters, unless otherwise specified, for IGNN, we use affine transformation $b_\Omega(U) = \Omega UA$; linear output function $f_\Theta(X) = \Theta X$; ReLU activation $\phi(\cdot) = \max(\cdot, 0)$; learning rate 0.01; dropout with parameter 0.5 before the output function; and $\kappa = 0.95$. We tune layers, hidden nodes, and κ through grid search. The hyperparameters for other baselines are consistent with that reported in their papers. Results with identical experimental settings are reused from previous works.

Synthetic Chains Data Set

We construct a synthetic node classification task to test the capability of models of learning to gather information from distant nodes. We consider the chains directed from one end to the other end with length l (*i.e.* $l + 1$ nodes in the chain). For simplicity, we consider binary classification task with 2 types of chains. Information about the type is only encoded as 1/0 in first dimension of the feature (100d) on the starting end of the chain. The labels are provided as one-hot vectors (2d). In the data set we choose chain length $l = 9$ and 20 chains for each class with a total of 400 nodes. The training set consists of 20 data points randomly picked from these nodes in the total 40 chains. Respectively, the validation set and test set have 100 and 200 nodes.

A single-layer IGNN is implemented with 16 hidden unites and weight decay of parameter 5×10^{-4} for all chains data sets with different l. Four representative baselines are chosen: Stochastic Steady-state Embedding (**SSE**) [47], Graph Convolutional Network (**GCN**) [91], Simple Graph Convolution (**SGC**) [180] and Graph Attention Network (**GAT**) [170]. They all use the same hidden units and weight decay as IGNN. For (**GAT**), 8 head attention is used. For (**SSE**), we use the embedding directly as output and fix-point iteration $n_h = 8$, as suggested [47].

As mentioned in Section 4.5, convolutional GNNs with $T = 2$ cannot capture the dependency with a range larger than 2-hops. To see how convolutional GNNs capture the long-range dependency as T grows, we give an illustration of Micro-F_1 verses T for the selected baselines in Figure 4.4. From the experiment, we find that convolutional GNNs cannot capture the long-range dependency given larger T. This might be a result of the limited number of training nodes in this chain task. As T grows, convolutional GNNs experience an explosion of number of parameters to train. Thus the training data becomes insufficient for these models as the number of parameters increases.

Node Classification

For node classification task, we consider the applications under both transductive (Amazon) [185] and inductive (PPI) [74] settings. Transducive setting is where the model has access to the feature vectors of all nodes during training, while inductive setting is where the graphs

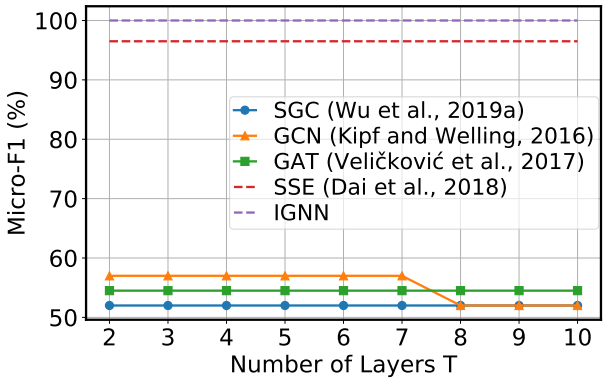


Figure 4.4: Chains with $l = 9$. Traditional methods fail even with more iterations.

Table 4.4: The overview of data set statistics in node classification tasks.

Data set	# Nodes	# Edges	# Labels	Label type	Graph type
Amazon (transductive)	334,863	925,872	58	Product type	Co-purchasing
PPI (inductive)	56,944	818,716	121	Bio-states	Protein

for testing remain completely unobserved during training. The statistics of the data sets can be found in Table 4.4.

For experiments on Amazon, we construct a one-layer IGNN with 128 hidden units. No weight decay is utilized. The hyper parameters of baselines are consistent with [185, 47].

For experiments on PPI, a five-layer IGNN model is applied for this multi-label classification tasks with hidden units as [1024, 512, 512, 256, 121] and $\kappa = 0.98$ for each layer. In addition, four MLPs are applied between the first four consecutive IGNN layers. We use the identity output function. Neither weight decay nor dropout is employed. We keep the experimental settings of baselines consistent with [170, 47, 91, 74].

Graph Classification

For graph classification, 5 bioinformatics data sets are employed with information given in Table 4.2. We compare IGNN with a comprehensive set of baselines, including a variety of GNNs: Deep Graph Convolutional Neural Network (**DGCNN**) [193], Diffusion-Convolutional Neural Networks (**DCNN**) [12], Fast and Deep Graph Neural Network (**FDGNN**) [62], **GCN** [91] and Graph Isomorphism Network (**GIN**) [182], and a number of state-of-the-art graph kernels: Graphlet Kernel (**GK**) [151], Random-walk Kernel (**RW**) [66], Propagation Kernel (**PK**) [128] and Weisfeiler-Lehman Kernel (**WL**) [152]. We reuse the results from literatures [182, 62] since the same experimental settings are maintained.

As of IGNN, a three-layer IGNN is constructed for comparison with the hidden units of

Table 4.5: Statistics of the data sets for heterogeneous graphs [134]. The node attributes are bag-of-words of text. Num. labeled data denotes the number of nodes involved during training.

	Relations (A-B)	Num. A	Num. B	Num. A-B	Relation type	Num. relations	Num. node attributes	Num. labeled data	Num. classes
ACM	Paper-Author	3,025	5,835	9,744	P-A-P	29,281	1,830 (Paper abstract)	600	3
	Paper-Subject	3,025	56	3,025	P-S-P	2,210,761			
IMDB	Movie-Actor	3,550	4,441	10,650	M-A-M	66,428	1,007 (Movie plot)	300	3
	Movie-Director	3,550	1,726	3,550	M-D-M	13,788			
DBLP	Paper-Author	7,907	1,960	14,238	P-A-P	144,783	2,000 (Paper abstract)	80	4
	Paper-Paper	7,907	7,907	10,522	P-P-P	90,145			
	Author-Term	1,960	1,975	57,269	P-A-T-A-P	57,137,515			

each layer as 32 and $\kappa = 0.98$ for all layers. We use an MLP as the output function. Besides, batch normalization is applied on each hidden layer. Neither weight decay nor dropout is utilized.

Heterogeneous Networks

For heterogeneous networks, three data sets are chosen (ACM, IMDB, and DBLP). Consistent with previous works [134], we use the the publicly available ACM data set [174], preprocessed DBLP and IMDB data sets [134]. For ACM and DBLP data sets, the nodes are papers and the aim is to classify the papers into three classes (Database, Wireless Communication, Data Mining), and four classes (DM, AI, CV, NLP)[2], respectively. For IMDB data set, the nodes are movies and we aim to classify these movies into three classes (Action, Comedy, Drama). The detailed information of data sets can be referred to Table 4.5. The preprocessing procedure and splitting method on three data sets keep consistent with [134].

State-of-the-art baselines are selected for comparison with IGNN, including no-attribute network embedding: **DeepWalk** [138], attributed network embedding: **GCN**, **GAT** and **DGI** [169], and attributed multiplex network embedding: **mGCN** [116], **HAN** [174] and **DMGI** [134]. Given the same experimental settings, we reuse the results of baselines from [134].

A one-layer IGNN with hidden units as 64 is implemented on all data sets. Similar to **DMGI**, a weight decay of parameter 0.001 is used. For ACM, $\kappa = (0.55, 0.55)$ is used for Paper-Author and Paper-Subject relations. For IMDB, we select $\kappa = (0.5, 0.5)$ for Movie-Actor and Movie-Director relations. For DBLP, $\kappa = (0.7, 0.4)$ is employed for Paper-Author and Paper-Paper relations. As mentioned in Section 4.4, in practice, the convergence property can still hold when $\sum_i \kappa_i > 1$.

[2]**DM**: KDD,WSDM,ICDM, **AI**: ICML,AAAI,IJCAI, **CV**: CVPR, **NLP**: ACL,NAACL,EMNLP

Over-smoothness

Convolutional GNNs has suffered from over-smoothness when the model gets deep. An interesting question to ask is whether IGNN suffers from the same issue and experience performance degradation in capturing long-range dependency with its "infinitively deep" GNN design.

In an effort to answer this question, we compared IGNN against two latest convolutional GNN models that solve the over-smoothness issue, GCNII [38] and DropEdge [144]. We use the same experimental setting as the Chains experiment in Section 4.5. Both GCNII and DropEdge are implemented with 10-layer and is compared with IGNN in capturing long-range dependency. The result is reported in Figure 4.5. We observe that IGNN consistently outperforms both GCNII and DropEdge as the chains gets longer. The empirical result suggest little suffering from over-smoothness for recurrent GNNs.

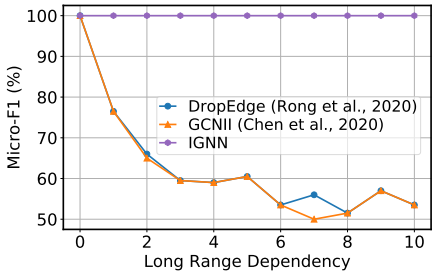


Figure 4.5: Micro-F_1 (%) performance with respect to the length of the chains.

4.6 Summary

In this chapter, we present the implicit graph neural network model, a framework of recurrent graph neural networks. We describe a sufficient condition for well-posedness based on the Perron-Frobenius theory and a projected gradient decent method for training. Similar to some other recurrent graph neural network models, implicit graph neural network captures the long-range dependency, but it carries the advantage further with a superior performance in a variety of tasks, through rigorous conditions for convergence and exact efficient gradient steps. More notably, the flexible framework extends to heterogeneous networks where it maintains its competitive performance.

Broader Impact

GNN models are widely used on applications involving graph-structured data, including computer vision, recommender systems, and biochemical strucature discovery. Backed by more

rigorous mathematical arguments, our research improves the capability GNNs of capturing the long-range dependency and therefore boosts the performance on these applications.

The improvements of performance in the applications will give rise to a better user experience of products and new discoveries in other research fields. But like any other deep learning models, GNNs runs into the problem of interpretability. The trade-off between performance and interpretability has been a topic of discussion. On one hand, the performance from GNNs benefits the tasks. On the other hand, the lack of interpretability might make it hard to recognize underlying bias when applying such algorithm to a new data set. Recent works [77] propose to address the fairness issue by enforcing the fairness constraints.

While our research focuses on performance by capturing the long-range dependency, like many other GNNs, it does not directly tackle the fairness and interpretability aspect. We would encourage further work on fairness and interpretability on GNNs. Another contribution of our research is on the analysis of heterogeneous networks, where the fairness on treatment of different relationships remains unexplored. The risk of discrimination in particular real-world context might require cautious handling when researchers develop models.

Kronecker Product

For two matrices A and B, the Kronecker product of $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ is denoted as $A \otimes B \in \mathbb{R}^{pm\times qn}$:

$$A \otimes B = \begin{pmatrix} A_{11}B & \cdots & A_{1n}B \\ \vdots & \ddots & \vdots \\ A_{m1}B & \cdots & A_{mn}B \end{pmatrix}.$$

By definition of the Kronecker product, $(A \otimes B)^\top = A^\top \otimes B^\top$. Additionally, the following equality holds assuming compatible shapes, $(A^\top \otimes W)\,\mathbf{vec}(X) = \mathbf{vec}(WXA)$ [146], where $\mathbf{vec}(X) \in \mathbb{R}^{mn}$ denotes the vectorization of matrix $X \in \mathbb{R}^{m\times n}$ by stacking the columns of X into a single column vector of dimension mn. Suppose $x_i \in \mathbb{R}^m$ is the i-th column of X, $\mathbf{vec}(X) = [x_1^\top, \ldots, x_n^\top]^\top$.

Leveraging the definition of Kronecker product and vectorization, the following equality holds, $(A^\top \otimes W)\,\mathbf{vec}(X) = \mathbf{vec}(WXA)$ [146]. Intuitively, this equality reshapes WXA which is linear in X into a more explicit form $(A^\top \otimes W)\,\mathbf{vec}(X)$ which is linear in $\mathbf{vec}(X)$, a flattened form of X. Through the transformation, we place WXA into the form of Mx. Thus, we can employ Lemma 4.4.1 to obtain the well-posedness conditions.

Chapter 5

Stable Controllers Synthesis for Partially Observed Systems

The connection from implicit deep learning as presented in Chapter 2 to control is natural. When we are exploring stable conditions for neural network controlled systems, we see that the notations from implicit deep learning applies flawlessly to capture recurrent neural networks as presented in (5.2) and (5.5). Through manipulations with the notation, we are able to reach at some convex stability condition for the closed loop partially observed system controlled with recurrent neural networks. We develop an iterative optimization algorithm to generate a sequence of guaranteed-stable controllers for this highly difficult task. This work serves as a demonstration of having implicit deep learning notations in control efforts and hopefully enables future pushes on combining "implicit neural networks" in control and innovations in implicit models from the control community.

5.1 Introduction

Neural network decision making and control has seen a huge advancement recently accompanied by the success of reinforcement learning (RL) [159]. In particular, deep reinforcement learning (DRL) has achieved super-human performance with neural network policies (also referred to as controllers in control tasks) in various domains [122, 112, 153].

Policy gradient [160] is one of the most important approaches to DRL that synthesizes policies for continuous decision making problems. For control tasks, policy gradient method and its variants have successfully synthesized neural network controllers to accomplish complex control goals [107] without solving potentially non-linear planing problems at test time [106]. However, most of these methods focus on maximizing the reward function which only indirectly enforce desirable properties. Specifically, global stability of the closed-loop system [145] guarantees convergence to the desired state of origin from any initial state and therefore is a very important property for safety critical systems (*e.g.* aircraft control [33]) where not a single diverging trajectory is acceptable. However, the set of parameters corresponding

to stabilizing controllers is in general nonconvex even in the simple setting of linear systems with linear controllers [57], which poses significant computational challenges for neural network controllers under the general setting of nonlinear systems.

Thanks to recent robustness studies of deep learning, we have seen attempts on giving stability certificates and/or ensuring stability at test time for fully-observed systems controlled by neural networks. Yet stability problems for neural network controlled partially observed systems remain open. Unlike fully-observed control systems where the plant states are fully revealed to the controller, most real-world control systems are only partially observed due to modeling inaccuracy, sensing limitations, and physical constraints [28]. Here, sensible estimates of the full system state usually depend on historical observations [29]. Some partially observed systems are modeled using partially observed Markov decision process (POMDP) [123] where an optimal solution is NP hard in general [125].

Chapter contributions. In the chapter, we propose a method to synthesize recurrent neural network (RNN) controllers with exponential stability guarantees for partially observed systems. We derive a convex inner approximation to the non-convex set of stable RNN parameters based on integral quadratic constraints [120], loop transformation [145]and a sequential semidefinite convexification technique, which guarantees exponential stability for both linear time invariant (LTI) systems and general nonlinear uncertain systems. A novel framework of projected policy gradient is proposed to maximize some unknown/complex reward function and ensure stability in the online setting where a guaranteed-stable RNN controller is synthesized and iteratively updated while interacting with and controlling the underlying system, which differentiates our works from most post-hoc validation methods. Finally, we carry out comprehensive comparisons with policy gradient, and demonstrate that our method effectively ensures the closed-loop stability and achieves higher reward on a variety of control tasks, including vehicle lateral control and power system frequency regulation.

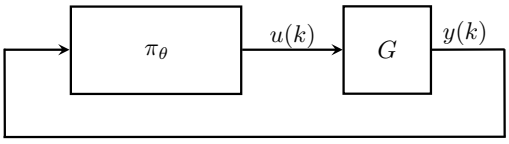


Figure 5.1: Feedback system of plant G and RNN controller π_θ

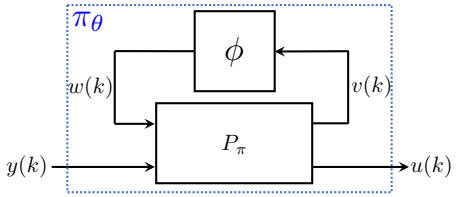


Figure 5.2: RNN as an interconnection of P_π and ϕ

Chapter outline. In Section 5.2, we outline related works on addressing partial observability, and enforcing stability in reinforcement learning. Section 5.3 discusses our proposed method for synthesizing RNN controllers for LTI plants with stability guarantees, and Section 5.4 extends it to systems with uncertainties and nonlinearities. Section 5.5 compares the proposed projected policy gradient method with policy gradient through numerical experiments.

Notation. $\mathbb{S}^n, \mathbb{S}^n_+, \mathbb{S}^n_{++}$ denote the sets of n-by-n symmetric, positive semidefinite and positive definite matrices, respectively. $\mathbb{D}^n_+$, $\mathbb{D}^n_{++}$ denote the set of n-by-n diagonal positive semidefinite, and diagonal positive definite matrices. The notation $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ denotes the standard 2-norm. We define ℓ^n_{2e} to be the set of all one-sided sequences $x : \mathbb{N} \to \mathbb{R}^n$. The subset $\ell^n_2 \subset \ell^n_{2e}$ consists of all square-summable sequences. When applied to vectors, the orders $>, \leq$ are applied elementwise.

5.2 Related Work

Partially Observed Decision Making and Output Feedback Control. In many problems [163, 19], only specific outputs but not the full system states are available for the decision maker. Therefore, memory in the controller is required to recover the full system states [147]. Control of these partially observed systems is often referred to as output feedback control [29], and has been studied extensively from both control and optimization perspectives [52, 199]. Under the setting with convexifiable objectives (e.g., H_∞ or H_2 performances), the optimal linear dynamic (*i.e.* with memory) controller can be obtained by using a change of variables or solving algebraic Riccati equations [61, 201]. However, for more sophisticated settings with unknown and/or flexibly defined cost functions, the problems become intractable for the aforementioned traditional methods, and RL techniques are proposed to reduced the computation cost and improve overall performance at test time, including the ones [105, 106] with static neural network controllers, and the ones [192, 80, 175] with dynamic controllers, represented by RNNs/long short-term memory neural networks.

Stability Guarantees For Neural Network Controlled Systems. As neural networks become popular in control tasks, safety and robustness of neural networks and neural network controlled systems has been actively discussed [124, 115, 60, 21, 43, 119, 75, 141, 42, 191, 59]. Closely related to this work are recent papers on robustness analysis of memoryless neural networks controlled systems based on robust control ideas. [186, 187, 136, 82] conduct stability analysis of neural network controlled linear and nonlinear systems and propose verification methods by characterizing activation functions using quadratic constraints. [51] adds additional projection layer on the controller to ensure stability for fully observed systems. [142] studies the stability of RNN itself when fitted to data but does not consider any plant to control by such RNN. The most related works are those that study dynamic neural network controllers. [5, 92] adapt RNN controllers through RL techniques to obtain stability guarantees. However, in these works, the reward function is assumed to be known, and conservative updates of controller parameters projected to a box neighborhood of the previous iterate are applied due to the non-convexity in their conditions. In contrast, our work enables much larger and more efficient updates thanks to jointly convex conditions derived through a novel sequential convexification and loop transformation approach unseen in these works.

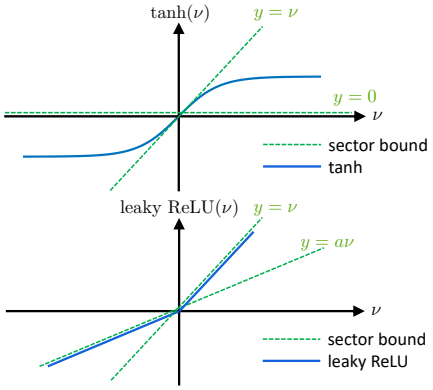


Figure 5.3: tanh $\in$ sector $[0, 1]$, Leaky ReLU $\in$ sector $[a, 1]$

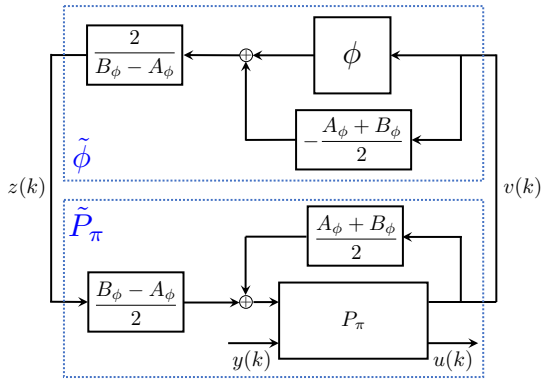


Figure 5.4: Loop transformation. If $\phi \in$ sector $[\alpha_\phi, \beta_\phi]$, then $\tilde{\phi} \in$ sector $[-1_{n_\phi \times 1}, 1_{n_\phi \times 1}]$.

5.3 Partially Observed Linear Systems

Problem Formulation

Consider the feedback system (shown in Figure 5.1) consisting of a plant G and an RNN controller π_θ which is expected to stabilize the system (*i.e.* steer the states of G to the origin). To streamline the presentation, we consider a partially observed, linear, time-invariant (LTI) system G defined by the following discrete-time model:

$$x(k+1) = A_G\, x(k) + B_G\, u(k) \tag{5.1a}$$
$$y(k) = C_G\, x(k) \tag{5.1b}$$

where $x(k) \in \mathbb{R}^{n_G}$ is the state, $u(k) \in \mathbb{R}^{n_u}$ is the control input, and $y(k) \in \mathbb{R}^{n_y}$ is the output. $A_G \in \mathbb{R}^{n_G \times n_G}$, $B_G \in \mathbb{R}^{n_G \times n_u}$, and $C_G \in \mathbb{R}^{n_y \times n_G}$. Since the plant G is partially observed, the observation matrix C_G may have a sparsity pattern or be column-rank deficient.

Assumption 5.3.1. *We assume that (A_G, B_G) is stabilizable, and (A_G, C_G) is detectable*[1].

Assumption 5.3.2. *We assume A_G, B_G, and C_G are known.*

Assumption 5.3.2 is partially lifted in Section 5.4 where we only assume partial information on the system dynamics.

Problem 5.3.1. *Our goal is to find a controller π that maps the observation y to an action u to both maximize some unknown reward $R = \sum_{k=0}^{T} r_k(x(k), u(k))$ over finite horizon T and stabilize the plant G.*

The single step reward $r_k(x(k), u(k))$ is assumed to be unknown and potentially highly complex to capture the vast possibility of desired controller. e.g. In many cases, to ensure

[1]The definitions of stabilizability and detectability can be found in [29].

extra safety, the reward is set to $r_k(x(k), u(k)) = 0, \forall k \geq l$ if there is a state violation at step l. This cannot be captured by any simple negative quadratic functions.

Controllers Parameterization

Output feedback control with known and convexifiable reward has been studied extensively [147], and linear dynamic controllers suffice for this case. However, in our problem setting, since the reward is unknown and nonconvex, and systems dynamics will become uncertain and nonlinear in Section 5.4, we consider a dynamic controller in the form of an RNN, which makes a class of high-capacity flexible controllers.

We model the RNN controller π_θ as an interconnection of an LTI system P_π, and combined activation functions $\phi : \mathbb{R}^{n_\phi} \to \mathbb{R}^{n_\phi}$ as shown in Figure 5.2. This parameterization is expressive, and contains many widely used model structures [142]. The RNN π_θ is defined as follows

$$P_\pi \begin{cases} \xi(k+1) &= A_K\ \xi(k)\ + B_{K1}\ w(k) + B_{K2}\ y(k) \\ u(k) &= C_{K1}\ \xi(k) + D_{K1}\ w(k) + D_{K2}\ y(k) \\ v(k) &= C_{K2}\ \xi(k) + D_{K3}\ y(k) \end{cases}$$
$$w(k) = \phi(v(k)) \tag{5.2}$$

where $\xi \in \mathbb{R}^{n_\xi}$ is the hidden state, $v, w \in \mathbb{R}^{n_\phi}$ are the input and output of ϕ, and matrices $A_K, \ldots, D_{K3}$ are parameters to be learned. Define $\theta = \begin{bmatrix} A_K & B_{K1} & B_{K2} \\ C_{K1} & D_{K1} & D_{K2} \\ C_{K2} & 0 & D_{K3} \end{bmatrix}$ as the collection of the learnable parameters of π_θ. We assume the initial condition of ξ to be zero $\xi(0) = 0_{n_\xi \times 1}$. The combined nonlinearity ϕ is applied element-wise, i.e., $\phi := [\varphi_1(v_1), ..., \varphi_{n_\phi}(v_{n_\phi})]^\top$, where φ_i is the i-th scalar activation function. We assume that the activation has a fixed point at origin, *i.e.* $\phi(0) = 0$.

Quadratic Constraints for Activation Functions

The stability condition relies on quadratic constraints (QCs) to bound the activation function. A typical QC is the sector bound as defined next.

Definition 5.3.1. *Let $\alpha \leq \beta$ be given. The function $\varphi : \mathbb{R} \to \mathbb{R}$ lies in the sector $[\alpha, \beta]$ if:*

$$(\varphi(\nu) - \alpha\nu) \cdot (\beta\nu - \varphi(\nu)) \geq 0\ \ \forall \nu \in \mathbb{R}. \tag{5.3}$$

The interpretation of the sector $[\alpha, \beta]$ is that φ lies between lines passing through the origin with slope α and β. Many activations are sector bounded, e.g., leaky ReLU is sector bounded in $[a, 1]$ with its parameter $a \in (0, 1)$; ReLU and tanh are sector bounded in $[0, 1]$ (denoted as tanh $\in$ sector $[0, 1]$). Figure 5.3 illustrates different activations (blue solid) and their sector bounds (green dashed).

Sector constraints can also be defined for combined activations ϕ. Assume the i-th scalar activation φ_i in ϕ is sector bounded by $[\alpha_i, \beta_i]$, $i = 1, ..., n_\phi$, then these sectors can be stacked

into vectors $\alpha_\phi, \beta_\phi \in \mathbb{R}^{n_\phi}$, where $\alpha_\phi = [\alpha_1, ..., \alpha_{n_\phi}]$ and $\beta_\phi = [\beta_1, ..., \beta_{n_\phi}]$, to provide QCs satisfied by ϕ.

Lemma 5.3.1. *Let $\alpha_\phi, \beta_\phi \in \mathbb{R}^{n_\phi}$ be given with $\alpha_\phi \leq \beta_\phi$. Suppose that ϕ satisfies the sector bound $[\alpha_\phi, \beta_\phi]$ element-wise. For any $\Lambda \in \mathbb{D}_+^{n_\phi}$, and for all $v \in \mathbb{R}^{n_\phi}$ and $w = \phi(v)$, it holds that*

$$\begin{bmatrix} v \\ w \end{bmatrix}^\top \begin{bmatrix} -2A_\phi B_\phi \Lambda & (A_\phi + B_\phi)\Lambda \\ (A_\phi + B_\phi)\Lambda & -2\Lambda \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \geq 0, \tag{5.4}$$

where $A_\phi = diag(\alpha_\phi)$, and $B_\phi = diag(\beta_\phi)$.

A proof is available in [59].

Loop Transformation

To derive convex stability conditions for their efficient enforcement in the learning process, we first perform a loop transformation on the RNN as shown in Figure 5.4. Through loop transformation, we obtain a new representation of the controller $\pi_{\tilde{\theta}}$, which is equivalent to the one shown in Figure 5.2:

$$\begin{bmatrix} v \\ u \end{bmatrix} = \tilde{P}_\pi \begin{bmatrix} z \\ y \end{bmatrix} \tag{5.5a}$$

$$z(k) = \tilde{\phi}(v(k)). \tag{5.5b}$$

The newly obtained nonlinearity $\tilde{\phi}$, defined in Figure 5.4, is sector bounded by $[-1_{n_\phi \times 1}, 1_{n_\phi \times 1}]$, and thus it satisfies a simplified QC: for any $\Lambda \in \mathbb{D}_+^{n_\phi}$, it holds that

$$\begin{bmatrix} v \\ z \end{bmatrix}^\top \begin{bmatrix} \Lambda & 0 \\ 0 & -\Lambda \end{bmatrix} \begin{bmatrix} v \\ z \end{bmatrix} \geq 0, \ \forall v \in \mathbb{R}^{n_\phi} \text{ and } z = \tilde{\phi}(v). \tag{5.6}$$

The transformed system $\tilde{P}_\pi$, defined in Figure 5.4, is of the form:

$$\begin{aligned} \xi(k+1) &= \tilde{A}_K \ \xi(k) + B_{K1} \tfrac{B_\phi - A_\phi}{2} \ z(k) + \tilde{B}_{K2} \ y(k) \\ u(k) &= \tilde{C}_{K1} \ \xi(k) + D_{K1} \tfrac{B_\phi - A_\phi}{2} \ z(k) + \tilde{D}_{K2} \ y(k) \\ v(k) &= C_{K2} \ \xi(k) + D_{K3} \ y(k) \end{aligned}$$

where

$$\begin{gathered} \tilde{A}_K = A_K + B_{K1} S_\phi C_{K2}, \quad \tilde{B}_{K2} = B_{K2} + B_{K1} S_\phi D_{K3}, \\ \tilde{C}_{K1} = C_{K1} + D_{K1} S_\phi C_{K2}, \ \tilde{D}_{K2} = D_{K2} + D_{K1} S_\phi D_{K3}, \\ S_\phi := \tfrac{A_\phi + B_\phi}{2}. \end{gathered} \tag{5.7}$$

The deriviation of $\tilde{P}_\pi$ is obtain by transforming the input to P_π by the following equation:

$$w(k) = \frac{B_\phi - A_\phi}{2} z(k) + \frac{A_\phi + B_\phi}{2} v(k). \tag{5.8}$$

Substituting the expression of $v(k)$ from (5.2) into (5.8) yields

$$w(k) = \frac{B_\phi - A_\phi}{2} z(k) + \frac{A_\phi + B_\phi}{2} C_{K2}\ \xi(k) + \frac{A_\phi + B_\phi}{2} D_{K3}\ y(k). \tag{5.9}$$

Finally, the transformed plant $\tilde{P}_\pi$ can be obtained by substituting (5.9) into (5.2).

We define the learnable parameters of $\pi_{\tilde{\theta}}$ as $\tilde{\theta} = \begin{bmatrix} \tilde{A}_K & B_{K1} & \tilde{B}_{K2} \\ \tilde{C}_{K1} & D_{K1} & \tilde{D}_{K2} \\ C_{K2} & 0 & D_{K3} \end{bmatrix}$. Since there is an one-to-one correspondence (5.7) between the transformed parameters $\tilde{\theta}$ and the original parameters θ, we will learn in the reparameterized space and uniquely recover the original parameters accordingly.

Convex Lyapunov Condition

The feedback system of plant G and RNN controller in $\pi_{\tilde{\theta}}$ (5.5) is defined by the following equations

$$\zeta(k+1) = \mathcal{A}\ \zeta(k) + \mathcal{B}\ z(k) \tag{5.10a}$$
$$v(k) = \ \mathcal{C}\ \zeta(k) + \mathcal{D}\ z(k) \tag{5.10b}$$
$$z(k) = \tilde{\phi}(v(k)) \tag{5.10c}$$

where $\zeta = [x^\top,\ \xi^\top]^\top$ gathers the states of G and $\tilde{P}_\pi$, and

$$\mathcal{A} = \begin{bmatrix} A_G + B_G \tilde{D}_{K2} C_G & B_G \tilde{C}_{K1} \\ \tilde{B}_{K2} C_G & \tilde{A}_K \end{bmatrix}, \mathcal{B} = \begin{bmatrix} B_G D_{K1} \frac{B_\phi - A_\phi}{2} \\ B_{K1} \frac{B_\phi - A_\phi}{2} \end{bmatrix},$$
$$\mathcal{C} = \begin{bmatrix} D_{K3} C_G & C_{K2} \end{bmatrix}, \qquad \mathcal{D} = 0_{n_\phi \times n_\phi}.$$

Note that matrices $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ are affine in $\tilde{\theta}$. The following theorem incorporates the QC for $\tilde{\phi}$ in the Lyapunov condition to derive the exponential stability condition of the feedback system using the S-Lemma [183, 27]

Theorem 5.3.1 (Sequential Convexification)**.** *Consider the feedback system of plant G in (5.1), and RNN controller $\pi_{\tilde{\theta}}$ in (5.5). Given a rate ρ with $0 \le \rho \le 1$, and matrices $\bar{P} \in \mathbb{R}^{n_\zeta \times n_\zeta}$ and $\bar{\Lambda} \in \mathbb{R}^{n_\phi \times n_\phi}$, if there exist matrices $Q_1 \in \mathbb{S}^{n_\zeta}_{++}$ and $Q_2 \in \mathbb{D}^{n_\phi}_{++}$, and parameters $\tilde{\theta}$ such that the following condition holds*

$$\begin{bmatrix} \rho^2(2\bar{P} - \bar{P}^\top Q_1 \bar{P}) & 0 & \mathcal{A}^\top & \mathcal{C}^\top \\ 0 & 2\bar{\Lambda} - \bar{\Lambda}^\top Q_2 \bar{\Lambda} & \mathcal{B}^\top & \mathcal{D}^\top \\ \mathcal{A} & \mathcal{B} & Q_1 & 0 \\ \mathcal{C} & \mathcal{D} & 0 & Q_2 \end{bmatrix} \succeq 0, \tag{5.11}$$

then for any $x(0)$, we have $\|x(k)\| \le \sqrt{cond(P)}\rho^k \|x(0)\|$ for all k, where $cond(P)$ is the condition number of P, and $P := Q_1^{-1}$ i.e., the feedback system is exponentially stable with rate ρ.

Proof. Assume there exist $Q_1 \in \mathbb{S}^{n_\zeta}_{++}$, $Q_2 \in \mathbb{D}^{n_\phi}_{++}$, and $\tilde{\theta}$, such that (5.11) holds. It follows from Schur complements that (5.11) is equivalent to

$$\begin{bmatrix} \mathcal{A} & \mathcal{B} \\ \mathcal{C} & \mathcal{D} \end{bmatrix}^\top \begin{bmatrix} Q_1^{-1} & 0 \\ 0 & Q_2^{-1} \end{bmatrix} \begin{bmatrix} \mathcal{A} & \mathcal{B} \\ \mathcal{C} & \mathcal{D} \end{bmatrix} - \begin{bmatrix} \rho^2(2\bar{P} - \bar{P}^\top Q_1 \bar{P}) & 0 \\ 0 & 2\bar{\Lambda} - \bar{\Lambda}^\top Q_2 \bar{\Lambda} \end{bmatrix} \preceq 0. \tag{5.12}$$

It follows from the inequalities $\bar{P}^\top Q_1 \bar{P} - 2\bar{P} \succeq -Q_1^{-1}$ and $\bar{\Lambda}^\top Q_2 \bar{\Lambda} - 2\bar{\Lambda} \succeq -Q_2^{-1}$ for any $\bar{P} \in \mathbb{R}^{n_\zeta \times n_\zeta}$ and $\bar{\Lambda} \in \mathbb{R}^{n_\phi \times n_\phi}$ [165, 142] that (5.12) implies

$$\begin{bmatrix} \mathcal{A} & \mathcal{B} \\ \mathcal{C} & \mathcal{D} \end{bmatrix}^\top \begin{bmatrix} Q_1^{-1} & 0 \\ 0 & Q_2^{-1} \end{bmatrix} \begin{bmatrix} \mathcal{A} & \mathcal{B} \\ \mathcal{C} & \mathcal{D} \end{bmatrix} - \begin{bmatrix} \rho^2 Q_1^{-1} & 0 \\ 0 & Q_2^{-1} \end{bmatrix} \preceq 0. \tag{5.13}$$

Defining $P = Q_1^{-1}$, and $\Lambda = Q_2^{-1}$, and rearranging (5.13), we have that P, Λ, and $\tilde{\theta}$ satisfy the following condition

$$\begin{bmatrix} \mathcal{A}^\top P \mathcal{A} - \rho^2 P & \mathcal{A}^\top P \mathcal{B} \\ \mathcal{B}^\top P \mathcal{A} & \mathcal{B}^\top P \mathcal{B} \end{bmatrix} + \begin{bmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{bmatrix}^\top \begin{bmatrix} \Lambda & 0 \\ 0 & -\Lambda \end{bmatrix} \begin{bmatrix} \mathcal{C} & \mathcal{D} \\ 0 & I \end{bmatrix} \preceq 0, \tag{5.14}$$

Define the Lyapunov function $V(\zeta) := \zeta^\top P \zeta$. Multiplying (5.14) on the left and right by $[\zeta(k)^\top,\ z(k)^\top]$ and its transpose yields

$$V(\zeta(k+1)) - \rho^2 V(\zeta(k)) + \begin{bmatrix} v(k) \\ z(k) \end{bmatrix}^\top \begin{bmatrix} \Lambda & 0 \\ 0 & -\Lambda \end{bmatrix} \begin{bmatrix} v(k) \\ z(k) \end{bmatrix} \leq 0. \tag{5.15}$$

It follows from $\tilde{\phi} \in$ sector $[-1_{n_\phi \times 1}, 1_{n_\phi \times 1}]$ that the last term in (5.15) is nonnegative, and thus $V(\zeta(k+1)) \leq \rho^2 V(\zeta(k))$. Iterate it down to $k = 0$, we have $V(\zeta(k)) \leq \rho^{2k} V(\zeta(0))$, which implies $\|\zeta(k)\| \leq \sqrt{\text{cond}(P)}\rho^k \|\zeta(0)\|$. Recall $\xi(0) = 0$. Therefore

$$\|x(k)\| \leq \|\zeta(k)\| \leq \sqrt{\text{cond}(P)}\rho^k \|\zeta(0)\| = \sqrt{\text{cond}(P)}\rho^k \|x(0)\|,$$

and this completes the proof. ∎

The above convex relaxation of the non-convex condition (5.14) leverages a "linearizing" semi-definite inequality based on a previous guess of Q_1^{-1} and Q_2^{-1} (as $\bar{P}$ and $\bar{\Lambda}$). The linear matrix inequality (LMI) condition (5.11) is jointly convex in the decision variables $\tilde{\theta}$, Q_1, Q_2, where Q_1 and Q_2 are the inverse matrices of the Lyapunov certificate and the multiplier in (5.14), and this allows for its efficient enforcement in the reinforcement learning process. Denote the LMI (5.11), $Q_1 \in \mathbb{S}^{n_\zeta}_{++}$, and $Q_2 \in \mathbb{D}^{n_\phi}_{++}$ altogether as $\text{LMI}(Q_1, Q_2, \tilde{\theta}, \bar{P}, \bar{\Lambda})$, which will later be incorporated in the policy gradient process to provide exponential stability guarantees.

Based on the stability condition (5.11), define the convex stability set $\mathcal{C}(\bar{P}, \bar{\Lambda})$ as

$$\mathcal{C}(\bar{P}, \bar{\Lambda}) := \left\{ \tilde{\theta} :\ \exists\ Q_1, Q_2,\ \text{s.t. LMI}(Q_1, Q_2, \tilde{\theta}, \bar{P}, \bar{\Lambda}) \right\}. \tag{5.16}$$

Given matrices $\bar{P}$ and $\bar{\Lambda}$, any parameter $\tilde{\theta}$ drawn from $\mathcal{C}(\bar{P}, \bar{\Lambda})$ ensures the exponential stability of the feedback system (5.10). The set $\mathcal{C}(\bar{P}, \bar{\Lambda})$ is a convex inner-approximation to the set of parameters that renders the feedback system stable, and the choice of $\bar{P}$ and $\bar{\Lambda}$ affects the conservatism in the approximation. One way of choosing $(\bar{P}, \bar{\Lambda})$ is provided in Algorithm 3.

Remark 5.3.1. *Although only sector bounds* (5.6) *are used to describe the activation functions, we can further reduce the conservatism by using off-by-one integral quadratic constraints [103] to also capture the slope information of activation functions as done in [186].*

Remark 5.3.2. *Note that although we only consider LTI plant dynamics, the framework can be immediately extended to plant dynamics described by RNNs, or neural state space models provided in [88].*

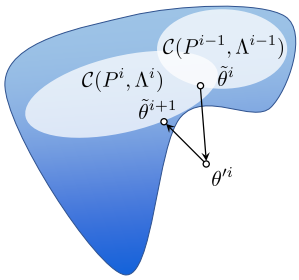
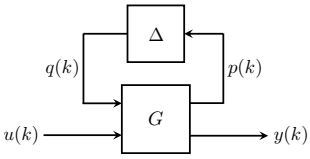
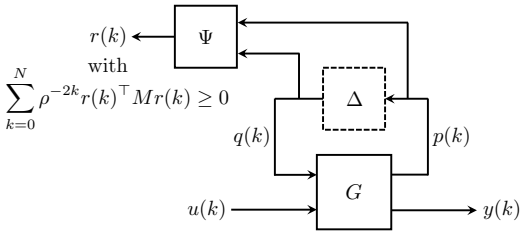


Figure 5.5: Illustration of Algorithm 3. The set of all the stabilizing $\tilde{\theta}$ is given in blue.



(a) uncertain plant $F_u(G, \Delta)$



(b) extended system of G and Ψ with IQCs

Figure 5.6: Uncertain plant and its corresponding constrained extended system

Projected Policy Gradient

Policy gradient methods [160, 176] enjoy convergence to optimality under the tabular setting and achieve good empirical performance for more challenging problems. However, with little assumption about the problem setting, they do not offer any stability guarantee for the closed loop system. We propose the projected policy gradient method that enforces the stability of the interconnected system while the policy is dynamically explored and updated.

Policy gradient approximates the gradient with respect to the parameters of a stochastic controller using samples of trajectories via (5.17) without any prior knowledge of plant parameters and the reward structures. Gradient ascent is then applied to refine the controller with the estimated gradients.

$$\begin{aligned} \nabla_{\tilde{\theta}} R(\pi_{\tilde{\theta}}) &= \int_{\mathcal{X}} d^{\pi_{\tilde{\theta}}}(x) \int_{\mathcal{U}} \nabla_{\tilde{\theta}} \pi_{\tilde{\theta}}(u|x) Q^{\pi_{\tilde{\theta}}}(x,u) du dx \\ &= \mathbb{E}_{\tilde{\theta}, x\sim d^{\pi}, u\sim\pi_{\tilde{\theta}}}[Q^{\pi}(x,u) \nabla_{\tilde{\theta}} \log \pi_{\tilde{\theta}}(u|x)]. \end{aligned} \tag{5.17}$$

In the above, $\tilde{\theta}$ represents the parameters of $\pi_{\tilde{\theta}}$. $R(\pi_{\tilde{\theta}})$ is the expected reward (negative cost) of the controller $\pi_{\tilde{\theta}}$. $d^{\pi}(x)$ is the distribution of states $x \in \mathcal{X}$ under $\pi_{\tilde{\theta}}$, where $\mathcal{X}$ is a set of states. $Q^{\pi}(x,u)$ is the reward-to-go after executing control $u \in \mathcal{U}$ at state x under $\pi_{\tilde{\theta}}$, where $\mathcal{U}$ is a set of actions.

Like any gradient method, policy gradient does not ensure the controller is in some specific set of preference (the set of stabilizing controller in our setting). To that end, a projection to the stability set $\mathcal{C}(\bar{P},\bar{\Lambda})$, $(Q_1,Q_2,\tilde{\theta}) \leftarrow \Pi_{\mathcal{C}(\bar{P},\bar{\Lambda})}(\theta')$, is applied between gradient updates, where θ' is the updated parameter, and the projection operator $\Pi_{\mathcal{C}(\bar{P},\bar{\Lambda})}(\theta')$ is defined as the following convex program,

$$\begin{aligned}\Pi_{\mathcal{C}(\bar{P},\bar{\Lambda})}(\theta') \in \arg\min_{Q_1,Q_2,\tilde{\theta}} \ & \left\| \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} - \begin{bmatrix} \bar{P}^{-1} \\ \bar{\Lambda}^{-1} \end{bmatrix} \right\|_F^2 + \|\tilde{\theta} - \theta'\|_F^2 \\ \text{s.t.} \quad & \mathrm{LMI}(Q_1,Q_2,\tilde{\theta},\bar{P},\bar{\Lambda}). \end{aligned} \tag{5.18}$$

Through the recursively feasible projection step (*i.e.* the feasibility is inherited in subsequent steps, summarized in Theorem 5.3.2), we conclude with a projected policy gradient method to synthesize stabilizing RNN controllers as summarized in Algorithm 3 and illustrated in Figure 5.5.

Theorem 5.3.2 (Recursive Feasibility)**.** *If LMI$(Q_1,Q_2,\tilde{\theta},P^{i-1},\Lambda^{i-1})$ is feasible (i.e. LMI$(Q_1,Q_2,\tilde{\theta},P^{i-1},\Lambda^{i-1})$ holds for some Q_1, Q_2, and $\tilde{\theta}$), then LMI$(Q_1,Q_2,\tilde{\theta},P^{i},\Lambda^{i})$ is also feasible, where $P^i = (Q_1^i)^{-1}$ and $\Lambda^i = (Q_2^i)^{-1}$ are from the i-th step of projected policy gradient, for $i = 1,2,\ldots$*

Proof. The main idea is to show that $(Q_1^i, Q_2^i, \tilde{\theta}^i)$ is already a feasible point for $\mathrm{LMI}(Q_1,Q_2,\tilde{\theta},P^i,\Lambda^i)$. Since $\mathrm{LMI}(Q_1,Q_2,\tilde{\theta},P^{i-1},\Lambda^{i-1})$ is feasible, at optimum of the projection step, we obtain the minimizer $(Q_1^i,Q_2^i,\tilde{\theta}^i)$ and $\mathrm{LMI}(Q_1^i,Q_2^i,\tilde{\theta}^i,P^{i-1},\Lambda^{i-1})$ holds. It follows from inequalities $2P^{i-1} - P^{i-1\top}Q_1^iP^{i-1} \preceq (Q_1^i)^{-1}$ and $2\Lambda^{i-1} - \Lambda^{i-1\top}Q_2^i\Lambda^{i-1} \preceq (Q_2^i)^{-1}$ that

$$\begin{bmatrix} \rho^2(Q_1^i)^{-1} & 0 & \mathcal{A}^\top & \mathcal{C}^\top \\ 0 & (Q_2^i)^{-1} & \mathcal{B}^\top & \mathcal{D}^\top \\ \mathcal{A} & \mathcal{B} & Q_1^i & 0 \\ \mathcal{C} & \mathcal{D} & 0 & Q_2^i \end{bmatrix} \succeq 0, \tag{5.19}$$

which renders $\mathrm{LMI}(Q_1^i,Q_2^i,\tilde{\theta}^i,P^i,\Lambda^i)$ true at a feasible point of $(Q_1^i,Q_2^i,\tilde{\theta}^i)$. ∎

In the algorithm, the gradient step performs gradient ascent using the estimated gradient $\nabla_{\tilde{\theta}} R(\pi(\tilde{\theta}))$. The projection step projects the updated parameters θ'^i from the gradient step to the convex stability set $\mathcal{C}(P^i,\Lambda^i)$, where P^i and Λ^i are computed using Q_1^i and Q_2^i from the previous projection step. We choose $\Lambda^0 = I_{n_\phi}$, and construct P^0 based on the method in [147].

Remark 5.3.3. *The projection step (5.18) is a semi-definite program (SDP) involving $O((n_\xi + n_\phi) \times (n_\xi + n_\phi))$ variables. The complexity of interior point SDP solvers usually scales cubically with the number of variables, potentially bringing computational burden when $(n_\xi + n_\phi)$ is large. Luckily, most high dimensional problems admit low dimension structures [179] and such overhead is only paid at training without further operations at deployment.*

Algorithm 3 Projected Policy Gradient

Input: Matrices P^0 and Λ^0 s.t. $\mathcal{C}(P^0, \Lambda^0)$ is not empty, learning rate $\sigma > 0$.

1: $i \leftarrow 0$
2: **while** not converged **do**
3: $\quad \theta'^i \leftarrow \tilde{\theta}^i + \sigma \nabla_{\tilde{\theta}} R(\pi_{\tilde{\theta}^i})$ ▷ gradient step
4: $\quad (Q_1^{i+1}, Q_2^{i+1}, \tilde{\theta}^{i+1}) \leftarrow \Pi_{\mathcal{C}(P^i, \Lambda^i)}(\theta'^i)$ ▷ proj. step
5: $\quad P^{i+1} \leftarrow (Q_1^{i+1})^{-1}, \;\; \Lambda^{i+1} \leftarrow (Q_2^{i+1})^{-1}$
6: $\quad i \leftarrow i + 1$
7: **end while**

Output: $\pi_{\tilde{\theta}}$

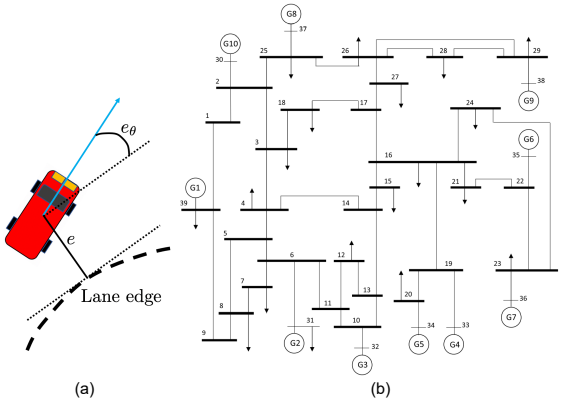


Figure 5.7: (a) Vehicle [1]; (b) Frequency Regulation on IEEE 39-bus New England Power System [11]

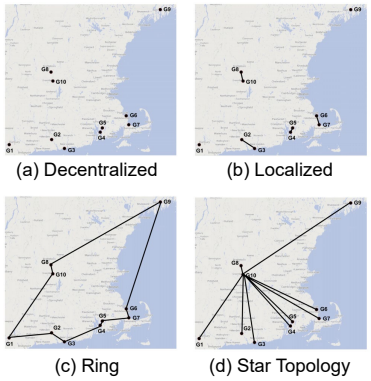


Figure 5.8: Four communication topologies for IEEE 39-bus power system [58].

5.4 Partially Observed Nonlinear Systems with Uncertainty

In the context of RL, we often need to handle systems with nonlinear dynamics and/or unmodeled dynamics. Here we model such a nonlinear and uncertain plant $F_u(G, \Delta)$ (shown in Figure 5.6a) as an interconnection of the nominal plant G, and the perturbation Δ representing the nonlinear, and uncertain part of the system. Therefore, in this new problem setting, we only require system dynamics to be partially known, and we use Δ to cover the difference between the original real system dynamics, and partially known dynamics G. The plant $F_u(G, \Delta)$ is defined by the following equations:

$$G \begin{cases} x(k+1) &= A_G \;\; x(k) + B_{G1} \; q(k) + B_{G2} \; u(k) \\ p(k) &= C_{G1} \; x(k) + D_{G1} \; q(k) \\ y(k) &= C_{G2} \; x(k) \end{cases}$$
$$q(\cdot) \quad = \Delta(p(\cdot)) \tag{5.20}$$

where $x(k) \in \mathbb{R}^{n_G}$, $u(k) \in \mathbb{R}^{n_u}$, and $y(k) \in \mathbb{R}^{n_y}$ are the state, control input, and output of the nominal plant G, and $p(k)$ and $q(k)$ are the input and output of Δ. The perturbation $\Delta : \ell_{2e}^{n_p} \to \ell_{2e}^{n_q}$ is a causal and bounded operator.

The perturbation Δ can represent various types of uncertainties and nonlinearities, including sector bounded nonlinearities, slope restricted nonlinearities, and unmodeled dynamics. Thus considering Δ extends our framework to the class of plants beyond LTI plants. The input-output relationship of Δ is characterized with an integral quadratic constraint (IQC) [120], which consists of a filter Ψ applied to the input p and output q of Δ, and a constraint on the output r of Ψ. The filter Ψ is an LTI system with the zero initial condition $\psi(0) = 0_{n_\psi \times 1}$:

$$\begin{aligned} \psi(k+1) &= A_\psi\ \psi(k) + B_{\psi 1}\ p(k) + B_{\psi 2}\ q(k), \\ r(k) &= C_\psi\ \psi(k) + D_{\psi 1}\ p(k) + D_{\psi 2}\ q(k). \end{aligned} \tag{5.21a}$$

To enforce exponential stability of the feedback system, we characterize Δ using the time-domain ρ-hard IQC, which is introduced in [103], and its definition is also provided below.

Definition 5.4.1. *Let Ψ be an LTI system defined in (5.21), and $M \in \mathbb{S}^{n_r}$. Suppose $0 \le \rho \le 1$. A bounded, causal operator $\Delta : \ell_{2e}^{n_p} \to \ell_{2e}^{n_q}$ satisfies the time-domain ρ-hard IQC defined by Ψ, M, and ρ, if the following condition holds for all $p \in \ell_{2e}^{n_p}$, $q = \Delta(p)$, and $N \ge 0$*

$$\sum_{k=0}^{N} \rho^{-2k} r(k)^\top M r(k) \ge 0. \tag{5.22}$$

where r is the output of Ψ driven by inputs (p, q).

Remark 5.4.1. *For a particular perturbation Δ, there is typically a class of valid ρ-hard IQCs defined by a fixed filter Ψ and a matrix M drawn from a convex set $\mathcal{M}$. Thus, in the stability condition derived later, $M \in \mathcal{M}$ will also be treated as a decision variable. A library of frequency-domain ρ-IQCs is provided in [26] for various types of perturbations. As shown in [148], a general class of frequency-domain ρ-IQCs can be translated into time-domain ρ-hard IQC by a multiplier factorization.*

When deriving the stability condition, the perturbation Δ will be replaced by the time-domain ρ-hard IQC (5.22) that describes it, and the associated filter Ψ, as shown in Figure 5.6b. Therefore, the stabilizing controller will be designed for the extended system (an interconnection of G and Ψ) subject to IQCs, instead of the original $F_u(G, \Delta)$. This controller will also be able to stabilize the original $F_u(G, \Delta)$. Define the extended state as $x_e = [x^\top,\ \psi^\top]^\top$.Also define $\zeta = [x_e^\top,\ \ \xi^\top]^\top$ to gather the states of the extended system and the controller. The feedback system of the extended system and the controller has the dynamics

$$\begin{aligned} \zeta(k+1) &= \mathcal{A}\ \zeta(k) + \mathcal{B}_1\ q(k) + \ \mathcal{B}_2\ z(k), \\ v(k) &= \mathcal{C}_1\ \zeta(k) + \mathcal{D}_1\ q(k) + \mathcal{D}_2\ z(k), \\ r(k) &= \mathcal{C}_2\ \zeta(k) + \mathcal{D}_3\ q(k) + \mathcal{D}_4\ z(k), \end{aligned} \tag{5.23}$$

where

$$\begin{aligned}
&\mathcal{A} = \begin{bmatrix} A_e + B_{e2}\tilde{D}_{k2}C_{e2} & B_{e2}\tilde{C}_{k1} \\ \tilde{B}_{k2}C_{e2} & \tilde{A}_k \end{bmatrix}, && \mathcal{B}_1 = \begin{bmatrix} B_{e1} \\ 0_{n_\xi \times n_q} \end{bmatrix}, \\
&\mathcal{B}_2 = \begin{bmatrix} B_{e2}D_{k1}\frac{B_\phi - A_\phi}{2} \\ B_{k1}\frac{B_\phi - A_\phi}{2} \end{bmatrix}, && \mathcal{C}_1 = \begin{bmatrix} D_{k3}C_{e2} & C_{k2} \end{bmatrix}, \\
&\mathcal{D}_1 = 0_{n_\phi \times n_q}, \quad \mathcal{D}_2 = 0_{n_\phi \times n_\phi}, && \mathcal{C}_2 = \begin{bmatrix} C_{e1} & 0_{n_r \times n_\xi} \end{bmatrix}, \\
&\mathcal{D}_3 = D_{e1}, \quad \mathcal{D}_4 = 0_{n_r \times n_\phi}, &&
\end{aligned}$$

and the extended system (shwon in Figure 5.6b) and its state space matrices $(A_e, B_{e1}, ..., D_{e1})$ are defined as follows.

$$x_e(k+1) = A_e\ x_e(k) + B_{e1}\ q(k) + B_{e2}\ u(k) \tag{5.24a}$$
$$r(k) = C_{e1}\ x_e(k) + D_{e1}\ q(k) \tag{5.24b}$$
$$y(k) = C_{e2}\ x_e(k) \tag{5.24c}$$

where

$$\begin{aligned}
A_e &= \begin{bmatrix} A_G & 0 \\ B_{\psi 1}C_{G1} & A_\psi \end{bmatrix},\ B_{e1} = \begin{bmatrix} B_{G1} \\ B_{\psi 1}D_{G1} + B_{\psi 2} \end{bmatrix},\ B_{e2} = \begin{bmatrix} B_{G2} \\ 0 \end{bmatrix} \\
C_{e1} &= \begin{bmatrix} D_{\psi 1}C_{G1} & C_\psi \end{bmatrix}, \quad D_{e1} = \begin{bmatrix} D_{\psi 1}D_{G1} + D_{\psi 2} \end{bmatrix}, \quad C_{e2} = \begin{bmatrix} C_{G2} & 0 \end{bmatrix}.
\end{aligned} \tag{5.25a}$$

The next theorem merges the QC for $\tilde{\phi}$ and the time-domain ρ-hard IQC for Δ with the Lyapunov theorem to derive the exponential stability condition for the uncertain feedback system.

Theorem 5.4.1. *Consider the feedback system of uncertain plant $F_u(G, \Delta)$, and RNN controller $\pi_{\tilde{\theta}}$. Assume Δ satisfies the time-domain ρ-hard IQC defined by Ψ, $\mathcal{M}$, and ρ, with $0 \le \rho \le 1$. Given $\bar{P} \in \mathbb{R}^{n_\zeta \times n_\zeta}$ and $\bar{\Lambda} \in \mathbb{R}^{n_\phi \times n_\phi}$. If there exist matrices $Q_1 \in \mathbb{S}^{n_\zeta}_{++}$, $Q_2 \in \mathbb{D}^{n_\phi}_{++}$, $M \in \mathcal{M}$, and parameters $\tilde{\theta}$ such that the following condition holds*

$$\begin{bmatrix} R^\top \Gamma R & \begin{bmatrix} \mathcal{A} & \mathcal{B}_1 & \mathcal{B}_2 \\ \mathcal{C}_1 & \mathcal{D}_1 & \mathcal{D}_2 \end{bmatrix}^\top \\ \begin{bmatrix} \mathcal{A} & \mathcal{B}_1 & \mathcal{B}_2 \\ \mathcal{C}_1 & \mathcal{D}_1 & \mathcal{D}_2 \end{bmatrix} & \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \end{bmatrix} \succeq 0, \tag{5.26}$$

where $\Gamma = \boldsymbol{diag}(\rho^2(2\bar{P} - \bar{P}^\top Q_1 \bar{P}), 2\bar{\Lambda} - \bar{\Lambda}^\top Q_2 \bar{\Lambda}, -M)$ is a block diagonal matrix and $R = \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & I \\ \mathcal{C}_2 & \mathcal{D}_3 & \mathcal{D}_4 \end{bmatrix}$. Then for any $x(0)$, we have $\|x(k)\| \le \sqrt{cond(P)}\rho^k \|x(0)\|$ for all k, where $P := Q_1^{-1}$, i.e., the feedback system is exponentially stable with rate ρ.

Proof. Assume there exist $Q_1 \in \mathbb{S}^{n_\zeta}_{++}$, $Q_2 \in \mathbb{D}^{n_\phi}_{++}$, $M \in \mathcal{M}$, and $\tilde{\theta}$ such that (5.26) holds. It follows from Schur complements that (5.26) is equivalent to

$$\begin{bmatrix} \mathcal{A} & \mathcal{B}_1 & \mathcal{B}_2 \\ \mathcal{C}_1 & \mathcal{D}_1 & \mathcal{D}_2 \end{bmatrix}^\top \begin{bmatrix} Q_1^{-1} & 0 \\ 0 & Q_2^{-1} \end{bmatrix} \begin{bmatrix} \mathcal{A} & \mathcal{B}_1 & \mathcal{B}_2 \\ \mathcal{C}_1 & \mathcal{D}_1 & \mathcal{D}_2 \end{bmatrix} - R^\top \begin{bmatrix} \rho^2(2\bar{P} - \bar{P}^\top Q_1 \bar{P}) & 0 & 0 \\ 0 & 2\bar{\Lambda} - \bar{\Lambda}^\top Q_2 \bar{\Lambda} & 0 \\ 0 & 0 & -M \end{bmatrix} R \preceq 0. \tag{5.27}$$

By inequalities $\bar{P}^\top Q_1 \bar{P} - 2\bar{P} \succeq -Q_1^{-1}$ and $\bar{\Lambda}^\top Q_2 \bar{\Lambda} - 2\bar{\Lambda} \succeq -Q_2^{-1}$ for any $\bar{P} \in \mathbb{R}^{n_\zeta \times n_\zeta}$ and $\bar{\Lambda} \in \mathbb{R}^{n_\phi \times n_\phi}$, we have that (5.27) implies

$$\begin{bmatrix} \mathcal{A} & \mathcal{B}_1 & \mathcal{B}_2 \\ \mathcal{C}_1 & \mathcal{D}_1 & \mathcal{D}_2 \end{bmatrix}^\top \begin{bmatrix} Q_1^{-1} & 0 \\ 0 & Q_2^{-1} \end{bmatrix} \begin{bmatrix} \mathcal{A} & \mathcal{B}_1 & \mathcal{B}_2 \\ \mathcal{C}_1 & \mathcal{D}_1 & \mathcal{D}_2 \end{bmatrix} - R^\top \begin{bmatrix} \rho^2 Q_1^{-1} & 0 & 0 \\ 0 & Q_2^{-1} & 0 \\ 0 & 0 & -M \end{bmatrix} R \preceq 0. \tag{5.28}$$

Defining $P = Q_1^{-1}$ and $\Lambda = Q_2^{-1}$, and rearranging (5.28), we have P, Λ, M, and $\tilde{\theta}$ satisfy the following condition

$$\begin{aligned} &\begin{bmatrix} \mathcal{A} & \mathcal{B}_1 & \mathcal{B}_2 \\ I & 0 & 0 \end{bmatrix}^\top \begin{bmatrix} P & 0 \\ 0 & -\rho^2 P \end{bmatrix} \begin{bmatrix} \mathcal{A} & \mathcal{B}_1 & \mathcal{B}_2 \\ I & 0 & 0 \end{bmatrix} + \begin{bmatrix} \mathcal{C}_1 & \mathcal{D}_1 & \mathcal{D}_2 \\ 0 & 0 & I \end{bmatrix}^\top \begin{bmatrix} \Lambda & 0 \\ 0 & -\Lambda \end{bmatrix} \begin{bmatrix} \mathcal{C}_1 & \mathcal{D}_1 & \mathcal{D}_2 \\ 0 & 0 & I \end{bmatrix} \\ &\quad + \begin{bmatrix} \mathcal{C}_2 & \mathcal{D}_3 & \mathcal{D}_4 \end{bmatrix}^\top M \begin{bmatrix} \mathcal{C}_2 & \mathcal{D}_3 & \mathcal{D}_4 \end{bmatrix} \preceq 0. \end{aligned} \tag{5.29}$$

Define the Lyapunov function $V(\zeta) := \zeta^\top P \zeta$. Multiplying (5.29) on the left and right by $\begin{bmatrix} \zeta(k)^\top, & q(k)^\top, & z(k)^\top \end{bmatrix}$ and its transpose yields

$$V(\zeta(k+1)) - \rho^2 V(\zeta(k)) + \begin{bmatrix} v(k) \\ z(k) \end{bmatrix}^\top \begin{bmatrix} \Lambda & 0 \\ 0 & -\Lambda \end{bmatrix} \begin{bmatrix} v(k) \\ z(k) \end{bmatrix} + r(k)^\top M r(k) \leq 0. \tag{5.30}$$

It follows from $\tilde{\phi} \in$ sector $[-1_{n_\phi \times 1}, 1_{n_\phi \times 1}]$ that the third term is nonnegative. This yields

$$V(\zeta(k+1)) - \rho^2 V(\zeta(k)) + r(k)^\top M r(k) \leq 0.. \tag{5.31}$$

Multiply (5.31) by ρ^{-2k} for each k and sum over k to obtain

$$\rho^{-2(k-1)} V(\zeta(k)) - \rho^2 V(\zeta(0)) + \sum_{t=0}^{k-1} \rho^{-2t} r(t) M r(t) \leq 0. \tag{5.32}$$

By the assumption that Δ satisfies the ρ-hard IQC, the last term is nonnegative, and thus $V(\zeta(k)) \leq \rho^{2k} V(\zeta(0))$ for all k, which implies $\|\zeta(k)\| \leq \sqrt{\text{cond}(P)} \rho^k \|\zeta(0)\|$. Recall $\xi(0) = 0_{n_\xi \times 1}$ and $\psi(0) = 0_{n_\psi \times 1}$. Therefore

$$\|x(k)\| \leq \|\zeta(k)\| \leq \sqrt{\text{cond}(P)} \rho^k \|\zeta(0)\| = \sqrt{\text{cond}(P)} \rho^k \|x(0)\|,$$

and this completes the proof. ■

This LMI (5.26) is jointly convex in $\tilde{\theta}, Q_1, Q_2$ and M for any given $\bar{P}$ and $\bar{\Lambda}$. Based on this LMI, we define the convex robust stability set $\mathcal{C}_R(\bar{P}, \bar{\Lambda})$:

$$\mathcal{C}_R(\bar{P}, \bar{\Lambda}) := \left\{ \tilde{\theta} : \ \exists\, Q_1 \in \mathbb{S}_{++}^{n_\zeta},\ Q_2 \in \mathbb{D}_{++}^{n_\phi},\ M \in \mathcal{M},\ \text{s.t. (5.26)} \right\}.$$

Any parameter $\tilde{\theta}$ drawn from $\mathcal{C}_R(\bar{P}, \bar{\Lambda})$ ensures the exponential stability of the feedback system of $F_u(G, \Delta)$ and $\pi_{\tilde{\theta}}$, and this convex robust stability set can be used in the projection step.

Remark 5.4.2. *If we only require the feedback system to be stable ($\rho = 1$ in (5.26)), a more general class of IQCs, the time-domain hard IQCs [120], can be used to describe Δ.*

5.5 Numerical Experiment

To compare our method against regular RNN controller trained without projection, we consider 6 different tasks involving control of partially observed dynamical systems, including a linearized inverted pendulum and its nonlinear variant, a cartpole, vehicle lateral dynamics, a pendubot, and a high dimensional power system. Figure 5.7 gives a demonstrative visualization of tasks including vehicle lateral control and IEEE 39-bus power system frequency regulation, whose communication topologies are shown in Figure 5.8.

Detailed Experimental Setup

In this section, we give detailed information about the experiments. All experiments are conducted on a custom built machine with 36-core Intel Broadwell Xeon CPUs with 64 GB of RAM and are terminated in tens of minutes. In all tasks, both our method and policy gradient are trained to convergence and are capped at 1000 epochs. For each epoch, the gradient is estimated from a batch of 6000 step data sampled from the controller interacting with the plant and is applied to update the parameters. The trajectory length is capped at 200. The average reward from the sample of trajectories from the 6000 steps are reported. We show 500 epoch plots in Figure 5.9 for clearer capture of the convergence process. The experiments and models are coded in Python [168] with Tensorflow [118], CVXPY [49] and MOSEK[2]. We choose the learning rate of 1e-3, picked from grid search from 1e-1, 1e-2, 1e-3, 1e-4 to give best reward at convergence for policy gradient on the inverted pendulum task. We use ADAM optimizer [89] and gradient clipping to a maximum magnitude of 10 for each parameter. We use tanh activation for all our neural network controllers. The implementation of policy gradient is consistent with [104]. And our method adds the projection updates on the same code base. The initial guess for the Lyapunov matrix P^0 of Algorithm 3 is constructed using the method introduced in [147], which computes the Lyapunov matrix for output feedback control problem through LMIs. The initial guess for Λ^0 is an identity matrix.

Inverted Pendulum

We consider both a linearized inverted pendulum system and a full nonlinear version whose dynamics are given below. Both examples have two states x_1, x_2, representing the angular position (rad) and velocity (rad/s). Only the plant output, $y = x_1$, is observed. Our methods as described in Sections 5.3 and 5.4 are applied for the linear, and nonlinear variants, with the goal of balancing the inverted pendulum around the upright position.

Consider an inverted pendulum with mass $m = 0.15$ kg, length $l = 0.5$ m, and friction coefficient $\mu = 0.5$ Nms/rad. The discretized and linearized dynamics are:

$$\begin{aligned}\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} &= \begin{bmatrix} 1 & \delta \\ \frac{g\delta}{l} & 1 - \frac{\mu\delta}{ml^2} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\delta}{ml^2} \end{bmatrix} u(k), \\ y(k) &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}, \end{aligned} \tag{5.33}$$

[2] www.mosek.com

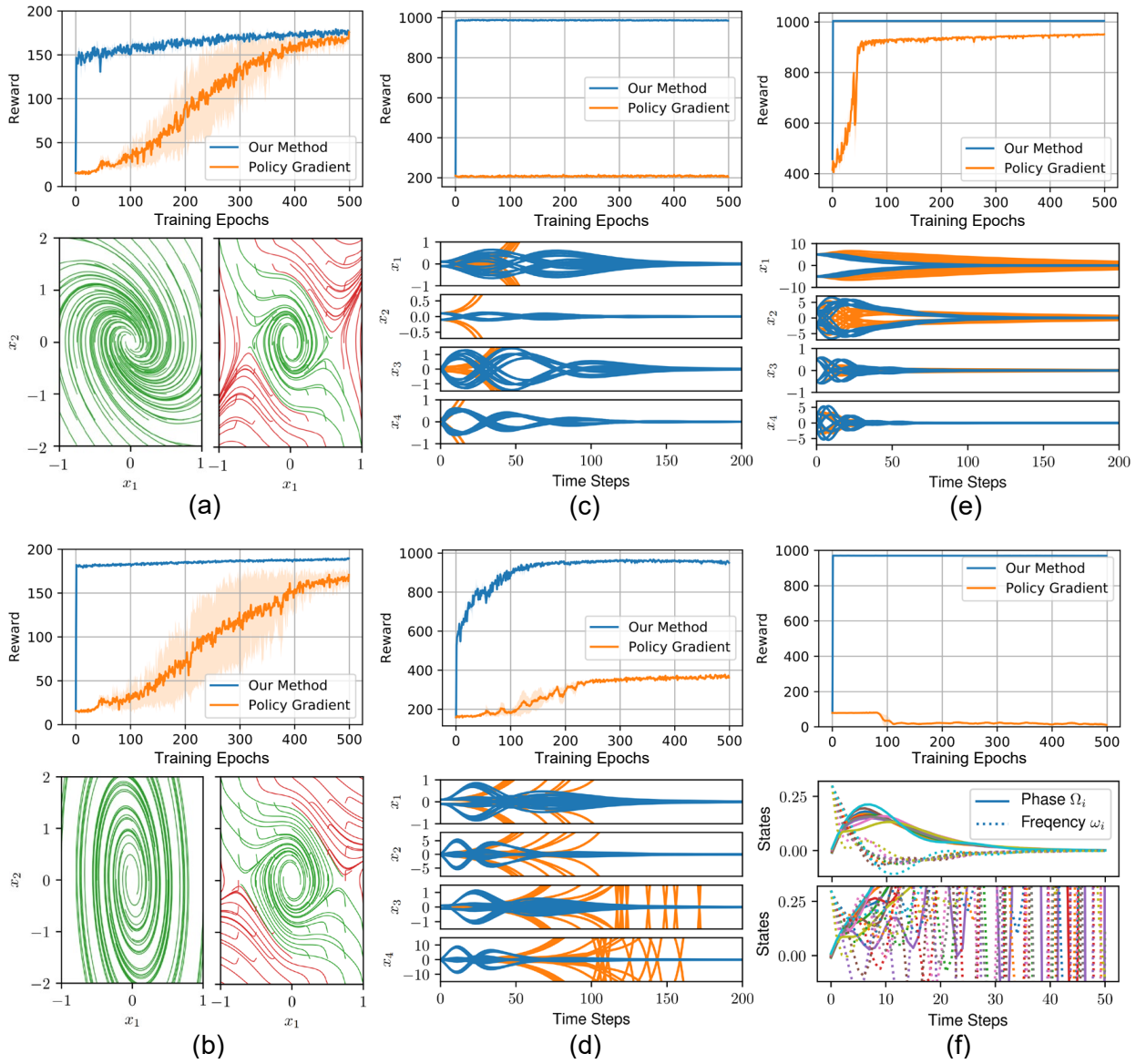


Figure 5.9: (a) Inverted Pendulum (linear); (b) Inverted Pendulum (nonlinear); (c) Cartpole; (d) Pendubot; (e) Vehicle lateral control; (f) IEEE 39-bus New England Power System frequency regulation. The error bars of reward plots characterize standard deviation across 3 runs with different seeds. For (a) and (b), the left figures are from our method and right figures from policy gradient. Converging trajectories are rendered in green while diverging ones in red. For (c), (d), (e), trajectories from our method are given in blue while those from policy gradient are in orange. For (f), top figure is given by our method and bottom one by policy gradient.

where u is the control input (1/10 Nm), and $\delta = 0.02$ s is the sampling time.

The discretized nonlinear dynamics of the inverted pendulum are

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 1 & \delta \\ \frac{g\delta}{l} & 1 - \frac{\mu\delta}{ml^2} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{-g\delta}{l} \end{bmatrix} q(k) + \begin{bmatrix} 0 \\ \frac{\delta}{ml^2} \end{bmatrix} u(k),$$
$$y(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix},$$
$$q(k) = \Delta(x_1(k)) = x_1(k) - \sin(x_1(k)).$$

The nonlinearity $\Delta(x_1) = x_1 - \sin(x_1)$ lies in the sector $[0, 0.41]$ for $x_1 \in [-1.4\,\text{rad}, 1.4\,\text{rad}]$. The time domian ρ-hard IQC for describing the nonlinearity Δ is defined by the static filter $\Psi = \left[\begin{smallmatrix} 0.41 & -1 \\ 0 & 1 \end{smallmatrix}\right]$, the matrix $M = \left[\begin{smallmatrix} 0 & \lambda \\ \lambda & 0 \end{smallmatrix}\right]$ for all $\lambda \geq 0$, and any $\rho \geq 0$.

At training, we pick $\rho = 1$ for both tasks. The observation of output is limited to $[-0.15, 0.15]$ and is normalized (*i.e.* devided by 0.15) before feeding into the controller. The trajectory terminates when the limit is violated or the length arrives 200. The hyperparameters are set to $n_\phi = 16, n_\xi = 16$. The following reward is used,

$$R = \sum_{k=0}^{T} \left[1.0 - 100x_1(k)^2 - 10x_2(k)^2 + 100u(k)^2\right],$$

where $(x_1(k), x_2(k))$ is state and $u(k)$ is control at step k.

Cartpole

A linearized cartpole system is considered, the goal of which is to balance the pendulum around the upright position while keeping the position of the cart close to the origin. The discretized and linearized dynamics of the cartpole are

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ x_4(k+1) \end{bmatrix} = \begin{bmatrix} 1 & -0.001 & 0.02 & 0 \\ 0 & 1.005 & 0 & 0.02 \\ 0 & -0.079 & 1 & -0.001 \\ 0 & 0.55 & 0 & 1.005 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0.04 \\ -0.04 \end{bmatrix} u(k),$$
$$y(k) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix},$$

where x_1 and x_2 represent the cart position (m) and the angular position (rad) of the pendulum, x_3 and x_4 are the corresponding cart velocities (m/s) and angular velocity (rad/s) of the pendulum, u is the horizontal force (N) exerting on the cart, and y is the plant output.

At training, we pick $\rho = 0.98$. The observation of output is limited to $x_1 : [-1, 1], x_2 : [-\pi/2, \pi/2]$ and is normalized before feeding into the controller. The trajectory terminates when the limit is violated or the length arrives 200. The hyperparameters are set to $n_\phi = 16, n_\xi = 16$. The following reward is used,

$$R = \sum_{k=0}^{T} \left[5.0 - x_1(k)^2 - x_2(k)^2 - 0.04x_3(k)^2 - 0.1x_4(k)^2 - 0.2u(k)^2\right],$$

where $(x_1(k), x_2(k), x_3(k), x_4(k))$ is state and $u(k)$ is control at step k.

Pendubot

A linearized pendubot is considered. The main goal is to balance the pendubot around the upright position. The linearized and discretiezed dynamics (sampling time $\delta = 0.01$ s) of the pendubot are

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ x_4(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0.01 & 0 & 0 \\ 0.6738 & 1 & -0.2483 & 0 \\ 0 & 0 & 1 & 0.01 \\ -0.6953 & 0 & 1.0532 & 1 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 0.4487 \\ 0 \\ -0.8509 \end{bmatrix} u(k),$$

$$y(k) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix},$$

where x_1 and x_3 represent the angular positions (rad) of the first link and the second link (relative to the first link), x_2 and x_4 are the corresponding angular velocities (rad/s), u represents the torque (Nm) applied on the first link, and y is the plant output.

At training, we pick $\rho = 0.98$. The observation of output is limited to $x_1 : [-1, 1], x_3 : [-1, 1]$. The hyperparameters are set to $n_\phi = 16, n_\xi = 16$. The following reward is used,

$$R = \sum_{k=0}^{T} \left[5.0 - x_1(k)^2 - 0.05 x_2(k)^2 - x_3(k)^2 - 0.05 x_4(k)^2 - 0.2 u(k)^2 \right],$$

where $(x_1(k), x_2(k), x_3(k), x_4(k))$ is state and $u(k)$ is control at step k.

Vehicle Lateral Control

In this setting, we consider the vehicle lateral control problem from [1, 186]. The goal is for the vehicle to track the lane edge while avoiding strong control inputs. The continuous-time linear vehicle lateral dynamics are

$$\begin{bmatrix} \dot{e} \\ \ddot{e} \\ \dot{e}_\theta \\ \ddot{e}_\theta \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{C_{\alpha f}+C_{\alpha r}}{mU} & -\frac{C_{\alpha f}+C_{\alpha r}}{m} & \frac{aC_{\alpha f}-bC_{\alpha r}}{mU} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{aC_{\alpha f}-bC_{\alpha r}}{I_z U} & -\frac{aC_{\alpha f}-bC_{\alpha r}}{I_z} & \frac{a^2 C_{\alpha f}+b^2 C_{\alpha r}}{I_z U} \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \\ e_\theta \\ \dot{e}_\theta \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{C_{\alpha f}}{m} \\ 0 \\ -\frac{aC_{\alpha f}}{I_z} \end{bmatrix} u + \begin{bmatrix} 0 \\ \frac{aC_{\alpha f}-bC_{\alpha r}-mU^2}{m} \\ 0 \\ \frac{a^2 C_{\alpha f}+b^2 C_{\alpha r}}{I_z} \end{bmatrix} c$$

where e is the perpendicular distance to the lane edge (m), and e_θ is the angle between the tangent to the straight section of the road and the projection of the vehicle's longitudinal axis (rad). Let $x = [e, \dot{e}, e_\theta, \dot{e}_\theta]^\top$ denote the plant state. The control u is the steering angle of the front wheel (rad). The plant output is $y = [e,\ e_\theta]^\top$. The disturbance c is the road curvature (1/m). In this task, we consider a constant curvature $c \equiv 0$. The values for the rest of the parameters are given in [186]. The controller is synthesized for the discretized dynamics with the sampling time $\delta = 0.02$ s.

At training, we pick $\rho = 0.98$. The observation of output is limited to $e : [-10, 10], e_\theta : [-1, 1]$ and is normalized before feeding into the controller. The trajectory terminates when the limit is violated or the length arrives 200. The hyperparameters are set to $n_\phi = 16, n_\xi = 16$. The following

reward is used,

$$R = \sum_{k=0}^{T} \left[5.0 - 0.01e(k)^2 - 0.04\dot{e}(k)^2 - e_\theta(k)^2 - 0.04\dot{e}_\theta(k)^2 - \frac{72}{\pi^2}u(k)^2 \right],$$

where $(e(k), \dot{e}(k), e_\theta(k), \dot{e}_\theta(k))$ is state and $u(k)$ is control at step k.

Power System Frequency Regulation

In this task, we address the distributed control problem for IEEE 39-Bus New England Power System frequency regulation [58, 82] with the decentralized communication topology shown in Figure 5.8 (a). The main goal is to optimally adjust the mechanical power input to each generator such that the phase and frequency at each bus can be restored to their nominal values after a possible perturbation. The linearized and discretized (sampling time $\delta = 0.2$ s) dynamics of the power system are

$$\begin{bmatrix} \Omega(k+1) \\ \omega(k+1) \end{bmatrix} = \begin{bmatrix} I_n & \delta I_n \\ -\delta M_p^{-1} L & -\delta M_p^{-1} D + I_n \end{bmatrix} \begin{bmatrix} \Omega(k) \\ \omega(k) \end{bmatrix} + \begin{bmatrix} 0_{n\times n} \\ \delta M_p^{-1} \end{bmatrix} u(k),$$
$$y(k) = \begin{bmatrix} I_n, 0_{n\times n} \end{bmatrix} \begin{bmatrix} \Omega(k) \\ \omega(k) \end{bmatrix},$$

where n is the number of rotors ($n = 10$ in this example), the states $\Omega, \omega \in \mathbb{R}^n$ represent the phases and frequencies of the rotors, $u \in \mathbb{R}^n$ represents the generator mechanical power injections, values for inertia coefficient matrix M_p, damping coefficient matrix D, and Laplacian matrix L are specified in Section IV of fazelnia2016convex.

Designing an optimal controller for these systems is challenging, because they consist of interconnected subsystems that have limited information sharing. For the case of distributed control, which requires the resulting controller to follow a sparsity pattern, it has been long known that finding the optimal solution amounts to an NP-hard optimization problem in general (even if the underlying system is linear). End-to-end reinforcement learning comes in handy, because it does not require model information by simply interacting with the environment while collecting rewards.

At training, we pick $\rho = 0.98$. The observation of output is limited to $\Omega_i : [-0.5, 0.5]$ and is normalized before feeding into the controller. The trajectory terminates when the limit is violated or the length arrives 200. The hyperparameters are set to $n_\phi = 20, n_\xi = 20$. The following reward is used,

$$R = \sum_{k=0}^{T} \left[5.0 - \|\Omega(k)\|^2 - \|\omega(k)\|^2 - 0.2\|u(k)\|^2 \right],$$

where $(\Omega(k), \omega(k))$ is state and $u(k)$ is control at step k.

The experimental results including rewards and sample trajectories at convergence are reported in Figure 5.9. In all experiments, our method achieves high reward after the first few projection steps that ensures stability, greatly outperforming the regular method which suffers from instability even after converging. For pendubot and inverted pendulum tasks, our method keeps perfecting the performance after the first projection steps which already give high performance. While for cartpole, vehicle lateral control, and power system frequency regulation tasks, our method converges to

optimal performance in one step. Our method gives converging trajectories for all tasks and achieves faster converging trajectories on the vehicle lateral control task. In comparison, policy gradient has been greatly impacted by the partial observability and converges to sub-optimal performance in cartpole, pendubot, and power system frequency regulation tasks and requires more steps to achieve optimal performance in inverted pendulum and vehicle lateral control tasks. Without stability guarantee, policy gradient fails to ensure converging trajectories from some initial conditions for all tasks excluding vehicle lateral control which is open-loop stable.

5.6 Summary

In this work, we present a method to synthesize stabilizing RNN controllers, which ensures the stability of the feedback systems both during learning and control process. We develop a convex set of stabilizing RNN parameters for nonlinear and partially observed systems. A novel projected policy gradient method is developed to synthesize a controller while enforcing stability by recursively projecting the parameters of the RNN controller to the convex set. By evaluating on a variety of control tasks, we demonstrate that our method learns stabilizing controllers with fewer samples, faster converging trajectories, and higher final performance than policy gradient. Future directions include extensions to implicit models [15, 55] or other memory units.

Chapter 6

Conclusion and Future Research

In this dissertation, we look into deep implicit models, a new type of deep learning model involving an infinite number of layers through the introduction of an equilibrium equation. Unlike deep learning models with explicit feed forward neural networks, the output of implicit models bases on a solution to some fixed-point equation for forward and backward pass in order to capture the infinite number of layers. The new model has great prospects in the machine learning community in bringing higher model capacity and enabling better performance.

The dissertation lays the theoretical and empirical foundations for deep implicit models. In the first part, we present implicit deep learning, a pioneering framework of deep implicit models. We discuss different theoretical aspects of implicit deep learning including well-posedness, training, robustness, and more. In the second part, we extend the framework to solve empirical tasks including representation learning in graph-structured data and stable controller synthesis. We show that implicit models achieve better performances than deep learning counterparts in these settings. In graph representation settings, as discussed in Chapter 4, we introduce the implicit graph neural networks framework (IGNN). IGNN uses an infinite number of message passing processes which resembles the class of traditional graph algorithms where some equilibrium state is attained through graph operations. As a result, IGNN achieves unprecedented performance in capturing long-range dependency in graphs. In stable controller synthesis settings, as discussed in Chapter 5, implicit deep learning enables flexible modeling of neural networks via simple notations. Thanks to the simplicity, we manage to derive convex conditions to specify stability of the closed loop system and come up with an efficient algorithm to iteratively synthesize stabilizing controllers.

Deep implicit models have seen more and more attention from the machine learning community in recent years thanks to research works laying the theoretical and empirical foundations. Even though we have seen some successes of implicit models in a range of areas, there are still a lot of future directions to look at. We list a few below.

Well-posdness in practice. In Chapter 2, we have discussed some sufficient well-posedness conditions for implicit deep learning. Through the enforcement of such conditions, we are able to obtain well-posed implicit models (*e.g.* implicit graph neural networks in Chapter 4). However, the conditions proposed are still sufficient conditions and are potentially conservative. There are other variants of implicit models that have not explicitly enforced well-posedness conditions when doing training and inference but are still able to push the models to work with tricks and tweaks.

We believe exploration into the nature of well-posedness and practical yet efficient enforcement of well-posedness are interesting directions to look into and will be of high impact in pushing implicit models to the public.

Training and Inference. The training and inference of implicit models have mostly relied on implicit gradient calculation through implicit function theorem. It turns out that both the forward and backward calculations require a solution to an equilibrium equation. Deep implicit models discussed in Chapter 2 and 4 are good examples. In the dissertation, we mostly use Picard iteration to obtain the solution. At convergence, the solution quality is best but the calculation can incur too much effort. For the model to empirically deploy to mobile devices or large scale environments, the cost of computation needs to be lowered as much as possible. Future research on lowering the training and inference cost will significantly benefit implicit models. The alternative gradient-less training methods for implicit models through the introduction of Fenchel divergence (as introduced in Chapter 3) is also a relatively unexplored but interesting direction to look at.

Capacity of implicit models. Recent empirical efforts (*e.g.* Chapter 4) on implicit models have shown that implicit models are able to achieve higher capacity and achieve unprecedented performances compared with deep learning with explicit feed forward networks. Intuitive explanations for the performance bump have focused on the fact that implicit models can be seen as infinitely deep neural networks. Theoretical views for such observation are still mostly unexplored. Future research on theoretical depiction of higher capacity of implicit models will direct the new designs and applications of implicit models.

Implicit model compression. The higher capacity of implicit models have been demonstrated empirically in several applications without intentionally introducing model compression techniques (*e.g.* encourage sparsity). We believe implicit models have great potential in further compressing the current numbers of parameters to reach unprecedented model compression results. Future research pushing on this would be highly impactful for the machine learning community overall.

Connection to control. The implicit deep learning framework in Chapter 2 has a natural connection to control systems. Specifically, it models all implicit models through the equilibrium equation $x = \phi(z)$ which can be captured using IQC for many activation functions, similar to what is presented in Chapter 5. This opens up new analysis opportunities for neural network or implicit model controlled systems. With the notational simplicity, we also expect future research to bring some control inspired results to deep implicit models through the implicit deep learning framework.

Bibliography

[1] Andrew Alleyne. "A comparison of alternative intervention strategies for unintended roadway departure (URD) control". In: *Vehicle System Dynamics* 27.3 (1997), pp. 157–186.

[2] Luís B. Almeida. "Backpropagation in Perceptrons with Feedback". In: *Neural Computers*. Ed. by Rolf Eckmiller and Christoph v.d. Malsburg. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 199–208. ISBN: 978-3-642-83740-1.

[3] Brandon Amos and J Zico 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.

[4] Brandon Amos et al. "Differentiable MPC for End-to-end Planning and Control". In: *Advances in Neural Information Processing Systems*. 2018, pp. 8289–8300.

[5] Charles W Anderson et al. "Robust reinforcement learning control using integral quadratic constraints for recurrent neural networks". In: *IEEE Transactions on Neural Networks* 18.4 (2007), pp. 993–1002.

[6] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. "Structured pruning of deep convolutional neural networks". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13.3 (2017), pp. 1–18.

[7] J Dwight Aplevich. *The essentials of linear state-space systems*. Wiley New York, 2000.

[8] Armin Askari et al. "Lifted Neural Networks". In: *arXiv preprint arXiv:1805.01532* (2018).

[9] Armin Askari et al. "Lifted neural networks". In: *arXiv preprint arXiv:1805.01532* (2018).

[10] Anish Athalye, Nicholas Carlini, and David A. Wagner. "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 274–283.

[11] T Athay, R Podmore, and S Virmani. "A practical method for the direct analysis of transient stability". In: *IEEE Transactions on Power Apparatus and Systems* 2 (1979), pp. 573–584.

[12] James Atwood and Don Towsley. "Diffusion-convolutional neural networks". In: *Advances in neural information processing systems*. 2016, pp. 1993–2001.

[13] Filipe de Avila Belbute-Peres et al. "End-to-End Differentiable Physics for Learning and Control". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 7178–7189. URL: `http://papers.nips.cc/paper/7948-end-to-end-differentiable-physics-for-learning-and-control.pdf`.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[15] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "Deep equilibrium models". In: *Advances in Neural Information Processing Systems*. 2019, pp. 688–699.

[16] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. "Deep Equilibrium Models". Preprint submitted. 2019.

[17] Shaojie Bai, Vladlen Koltun, and J Zico Kolter. "Multiscale Deep Equilibrium Models". In: *Advances in Neural Information Processing Systems* 33 (2020).

[18] Shaojie Bai, Vladlen Koltun, and J Zico Kolter. "Stabilizing Equilibrium Models by Jacobian Regularization". In: *arXiv preprint arXiv:2106.14342* (2021).

[19] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. "Learning to act using real-time dynamic programming". In: *Artificial intelligence* 72.1-2 (1995), pp. 81–138.

[20] Mikhail Belkin et al. "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.

[21] Felix Berkenkamp et al. "Safe model-based reinforcement learning with stability guarantees". In: *arXiv preprint arXiv:1705.08551* (2017).

[22] Abraham Berman and Robert J Plemmons. *Nonnegative matrices in the mathematical sciences*. SIAM, 1994.

[23] Mathieu Blondel, Andre Martins, and Vlad Niculae. "Learning classifiers with fenchel-young losses: Generalized entropies, margins, and algorithms". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 606–615.

[24] Mathieu Blondel, André FT Martins, and Vlad Niculae. "Learning with Fenchel-Young losses." In: *J. Mach. Learn. Res.* 21.35 (2020), pp. 1–69.

[25] Mathieu Blondel et al. "Efficient and Modular Implicit Differentiation". In: *arXiv preprint arXiv:2105.15183* (2021).

[26] Ross Boczar et al. "Exponential stability analysis via integral quadratic constraints". In: *arXiv preprint arXiv:1706.01337* (2017).

[27] Stephen Boyd et al. *Linear matrix inequalities in system and control theory*. SIAM, 1994.

[28] Darius Braziunas. "POMDP solution methods". In: *University of Toronto* (2003).

[29] Frank M Callier and Charles A Desoer. *Linear system theory*. Springer Science & Business Media, 2012.

[30] Nicholas Carlini and David A. Wagner. "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*. ACM, 2017, pp. 3–14.

[31] Miguel Carreira-Perpinan and Weiran Wang. "Distributed optimization of deeply nested systems". In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*. Ed. by Samuel Kaski and Jukka Corander. Vol. 33. Proceedings of Machine Learning Research. Reykjavik, Iceland: PMLR, 22–25 Apr 2014, pp. 10–19. URL: `http://proceedings.mlr.press/v33/carreira-perpinan14.html`.

[32] Miguel A Carreira-Perpinán and Mehdi Alizadeh. "ParMAC: distributed optimisation of nested functions, with application to learning binary autoencoders". In: *arXiv preprint arXiv:1605.09114* (2016).

[33] Abhijit Chakraborty, Peter Seiler, and Gary J Balas. "Susceptibility of F/A-18 flight controllers to the falling-leaf mode: Nonlinear analysis". In: *Journal of guidance, control, and dynamics* 34.1 (2011), pp. 73–85.

[34] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2011), pp. 1–27.

[35] Heng Chang et al. "Spectral Graph Attention Network". In: *arXiv preprint arXiv:2003.07450* (2020).

[36] Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. "The power of sparsity in convolutional neural networks". In: *arXiv preprint arXiv:1702.06257* (2017).

[37] Pratik Chaudhari et al. "Entropy-sgd: Biasing gradient descent into wide valleys". In: *arXiv preprint arXiv:1611.01838* (2016).

[38] Ming Chen et al. "Simple and Deep Graph Convolutional Networks". In: *arXiv preprint arXiv:2007.02133* (2020).

[39] Tian Qi Chen et al. "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 6571–6583. URL: `http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf`.

[40] Tian Qi Chen et al. "Neural ordinary differential equations". In: *Advances in neural information processing systems*. 2018, pp. 6571–6583.

[41] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[42] Jason Choi et al. "Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions". In: *arXiv preprint arXiv:2004.07584* (2020).

[43] Yinlam Chow et al. "A Lyapunov-based approach to safe reinforcement learning". In: *arXiv preprint arXiv:1805.07708* (2018).

[44] Jeremy M Cohen, Elan Rosenfeld, and J. Zico Kolter. *Certified Adversarial Robustness via Randomized Smoothing*. 2019. eprint: `arXiv:1902.02918`.

[45] Munther A Dahleh and Ignacio J Diaz-Bobillo. *Control of uncertain systems: a linear programming approach*. Prentice-Hall, Inc., 1994.

[46] Hanjun Dai, Bo Dai, and Le Song. "Discriminative embeddings of latent variable models for structured data". In: *International conference on machine learning*. 2016, pp. 2702–2711.

[47] Hanjun Dai et al. "Learning steady-states of iterative algorithms over graphs". In: *International conference on machine learning*. 2018, pp. 1106–1114.

[48] Yann N Dauphin et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in neural information processing systems*. 2014, pp. 2933–2941.

[49] Steven Diamond and Stephen Boyd. "CVXPY: A Python-embedded modeling language for convex optimization". In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[50] Priya Donti, Brandon Amos, and J Zico Kolter. "Task-based end-to-end model learning in stochastic optimization". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5484–5494.

[51] Priya L Donti et al. "Enforcing robust control guarantees within neural network policies". In: *arXiv preprint arXiv:2011.08105* (2020).

[52] John C Doyle. "Guaranteed margins for LQG regulators". In: *IEEE Transactions on automatic Control* 23.4 (1978), pp. 756–757.

[53] John Duchi et al. "Efficient projections onto the l 1-ball for learning in high dimensions". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 272–279.

[54] Geir E Dullerud and Fernando Paganini. *A course in robust control theory: a convex approach*. Vol. 36. Springer Science & Business Media, 2013.

[55] Laurent El Ghaoui et al. "Implicit deep learning". In: *SIAM Journal on Mathematics of Data Science* 3.3 (2021), pp. 930–958.

[56] Utku Evci et al. "The difficulty of training sparse neural networks". In: *arXiv preprint arXiv:1906.10732* (2019).

[57] Maryam Fazel et al. "Global convergence of policy gradient methods for the linear quadratic regulator". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1467–1476.

[58] Ghazal Fazelnia et al. "Convex relaxation for optimal distributed control problems". In: *IEEE Transactions on Automatic Control* 62.1 (2016), pp. 206–221.

[59] Mahyar Fazlyab, Manfred Morari, and George J Pappas. "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming". In: *IEEE Transactions on Automatic Control* (2020).

[60] Stefan R Friedrich and Martin Buss. "A robust stability approach to robot reinforcement learning based on a parameterization of stabilizing controllers". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3365–3372.

[61] Pascal Gahinet and Pierre Apkarian. "A linear matrix inequality approach to $\mathcal{H}_\infty$ control". In: *International journal of robust and nonlinear control* 4.4 (1994), pp. 421–448.

[62] Claudio Gallicchio and Alessio Micheli. "Fast and deep graph neural networks". In: *arXiv preprint arXiv:1911.08941* (2019).

[63] Claudio Gallicchio and Alessio Micheli. "Graph echo state networks". In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2010, pp. 1–8.

[64] Vignesh Ganapathiraman et al. "Inductive Two-layer Modeling with Parametric Bregman Transfer". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1636–1645.

[65] Bolin Gao and Lacra Pavel. "On the properties of the softmax function with application in game theory and reinforcement learning". In: *arXiv preprint arXiv:1704.00805* (2017).

[66] Thomas Gärtner, Peter Flach, and Stefan Wrobel. "On graph kernels: Hardness results and efficient alternatives". In: *Learning theory and kernel machines*. Springer, 2003, pp. 129–143.

[67] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1263–1272.

[68] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. URL: http://arxiv.org/abs/1412.6572.

[69] Marco Gori, Gabriele Monfardini, and Franco Scarselli. "A new model for learning in graph domains". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. IEEE. 2005, pp. 729–734.

[70] Sven Gowal et al. "On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models". In: *CoRR* abs/1810.12715 (2018). arXiv: 1810.12715. URL: http://arxiv.org/abs/1810.12715.

[71] Fangda Gu, Armin Askari, and Laurent El Ghaoui. "Fenchel lifted networks: A lagrange relaxation of neural network training". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 3362–3371.

[72] Fangda Gu et al. "Implicit graph neural networks". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020.

[73] Fangda Gu et al. "Recurrent Neural Network Controllers Synthesis with Stability Guarantees for Partially Observed Systems". In: *arXiv preprint arXiv:2109.03861* (2021).

[74] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems*. 2017, pp. 1024–1034.

[75] Minghao Han et al. "$\mathcal{H}_\infty$ model-free reinforcement learning with robust stability guarantee". In: *arXiv preprint arXiv:1911.02875* (2019).

[76] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149* (2015).

[77] Moritz Hardt, Eric Price, and Nati Srebro. "Equality of opportunity in supervised learning". In: *Advances in neural information processing systems*. 2016, pp. 3315–3323.

[78] Babak Hassibi, David G Stork, and Gregory J Wolff. "Optimal brain surgeon and general network pruning". In: *IEEE international conference on neural networks*. IEEE. 1993, pp. 293–299.

[79] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[80] Nicolas Heess et al. "Memory-based control with recurrent neural networks". In: *arXiv preprint arXiv:1512.04455* (2015).

[81] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[82] Ming Jin and Javad Lavaei. "Stability-certified reinforcement learning: A control-theoretic perspective". In: *IEEE Access* (2020).

[83] Ming Jin et al. "Power up! robust graph convolutional network against evasion attacks based on graph powering". In: *arXiv preprint arXiv:1905.10029* (2019).

[84] Michael Kampffmeyer et al. "Rethinking Knowledge Graph Propagation for Zero-Shot Learning". In: *arXiv preprint arXiv:1805.11724* (2018).

[85] Kazuya Kawakami. "Supervised sequence labelling with recurrent neural networks". In: *Ph. D. thesis* (2008).

[86] Mustafa Hani Khammash. "Stability and performance robustness of discrete-time systems with structured uncertainty". PhD thesis. 1990.

[87] Jingu Kim, Yunlong He, and Haesun Park. "Algorithms for nonnegative matrix and tensor factorizations: a unified view based on block coordinate descent framework". In: *Journal of Global Optimization* 58.2 (2014), pp. 285–319.

[88] K. K. Kim, E. R. Patrón, and R. D. Braatz. "Standard representation and unified stability analysis for dynamic artificial neural network models". In: *Neural Networks* 98 (2018), pp. 251–262.

[89] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[90] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *International Conference on Learning Representations* (2015).

[91] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[92] James Nate Knight and Charles Anderson. "Stable reinforcement learning with recurrent neural networks". In: *Journal of Control Theory and Applications* 9.3 (2011), pp. 410–420.

[93] J. Kolter. "Personal communication with A. Askari". Aug. 2019.

[94] Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.

[95] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[96] Anders Krogh and John A Hertz. "A simple weight decay can improve generalization". In: *Advances in neural information processing systems*. 1992, pp. 950–957.

[97] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=HJGU3Rodl.

[98] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial Machine Learning at Scale". In: 2017. URL: https://arxiv.org/abs/1611.01236.

[99] Tim Tsz-Kit Lau et al. "A Proximal Block Coordinate Descent Algorithm for Deep Neural Network Training". In: *Workshop track - International Conference on Learning Representations* (2018).

[100] Vadim Lebedev and Victor Lempitsky. "Fast convnets using group-wise brain damage". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2554–2564.

[101] Yann LeCun, John S Denker, and Sara A Solla. "Optimal brain damage". In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[102] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[103] Laurent Lessard, Benjamin Recht, and Andrew Packard. "Analysis and design of optimization algorithms via integral quadratic constraints". In: *SIAM Journal on Optimization* 26.1 (2016), pp. 57–95.

[104] Sergey Levine. *CS285: Deep Reinforcement Learning*. http://rail.eecs.berkeley.edu/deeprlcourse/. Course previously known as CS294: Deep Reinforcement Learning. Accessed: 2021-05-24. 2021.

[105] Sergey Levine and Vladlen Koltun. "Guided policy search". In: *International conference on machine learning*. PMLR. 2013, pp. 1–9.

[106] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[107] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436.

[108] Jia Li, Cong Fang, and Zhouchen Lin. "Lifted proximal operator machines". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 4181–4188.

[109] Qimai Li, Zhichao Han, and Xiaoming Wu. "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning". In: *AAAI-18 AAAI Conference on Artificial Intelligence*. 2018, pp. 3538–3545.

[110] Yujia Li et al. "Gated graph sequence neural networks". In: *arXiv preprint arXiv:1511.05493* (2015).

[111] Renjie Liao et al. "Reviving and improving recurrent back-propagation". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3082–3091.

[112] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning." In: *ICLR (Poster)*. 2016.

[113] Yanpei Liu et al. "Delving into Transferable Adversarial Examples and Black-box Attacks". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=Sys6GJqxl.

[114] Christos Louizos, Max Welling, and Diederik P Kingma. "Learning Sparse Neural Networks through L_0 Regularization". In: *arXiv preprint arXiv:1712.01312* (2017).

[115] Biao Luo, Huai-Ning Wu, and Tingwen Huang. "Off-policy reinforcement learning for $\mathcal{H}_\infty$ control design". In: *IEEE transactions on cybernetics* 45.1 (2014), pp. 65–76.

[116] Yao Ma et al. "Multi-dimensional graph convolutional networks". In: *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM. 2019, pp. 657–665.

[117] Aleksander Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rJzIBfZAb.

[118] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[119] Nikolai Matni et al. "From self-tuning regulators to reinforcement learning and back again". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE. 2019, pp. 3724–3740.

[120] Alexandre Megretski and Anders Rantzer. "System analysis via integral quadratic constraints". In: *IEEE Transactions on Automatic Control* 42.6 (1997), pp. 819–830.

[121] Carl D Meyer. *Matrix analysis and applied linear algebra*. Vol. 71. Siam, 2000.

[122] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[123] George E Monahan. "State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms". In: *Management science* 28.1 (1982), pp. 1–16.

[124] Jun Morimoto and Kenji Doya. "Robust reinforcement learning". In: *Neural computation* 17.2 (2005), pp. 335–359.

[125] Martin Mundhenk et al. "Complexity of finite-horizon Markov decision process problems". In: *Journal of the ACM (JACM)* 47.4 (2000), pp. 681–720.

[126] Sharan Narang et al. "Exploring sparsity in recurrent neural networks". In: *arXiv preprint arXiv:1704.05119* (2017).

[127] Geoffrey Negiar et al. "OPTML 2017: Lifted Neural Networks for Weight Initialization". In: (2017).

[128] Marion Neumann et al. "Propagation kernels: efficient graph kernels from propagated information". In: *Machine Learning* 102.2 (2016), pp. 209–245.

[129] Mark Newman. *Networks*. Oxford university press, 2018.

[130] Kenta Oono and Taiji Suzuki. "Graph Neural Networks Exponentially Lose Expressive Power for Node Classification". In: *ICLR 2020 : Eighth International Conference on Learning Representations*. 2020.

[131] Lawrence Page et al. *The pagerank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab, 1999.

[132] Nicolas Papernot et al. "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks". In: *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 582–597. DOI: 10.1109/SP.2016.41. URL: https://doi.org/10.1109/SP.2016.41.

[133] Nicolas Papernot et al. "The Limitations of Deep Learning in Adversarial Settings". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*. IEEE, 2016, pp. 372–387. DOI: 10.1109/EuroSP.2016.36. URL: https://doi.org/10.1109/EuroSP.2016.36.

[134] Chanyoung Park et al. "Unsupervised Attributed Multiplex Network Embedding". In: *arXiv preprint arXiv:1911.06750* (2019).

[135] Junyoung Park, Jinhyun Choo, and Jinkyoo Park. "Convergent Graph Solvers". In: *arXiv preprint arXiv:2106.01680* (2021).

[136] Patricia Pauli et al. "Linear systems with neural network nonlinearities: Improved stability analysis via acausal Zames-Falb multipliers". In: *arXiv preprint arXiv:2103.17106* (2021).

[137] Hongbin Pei et al. "Geom-GCN: Geometric Graph Convolutional Networks". In: *ICLR 2020 : Eighth International Conference on Learning Representations*. 2020.

[138] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[139] Fernando Pineda. "Generalization of back propagation to recurrent and higher order neural networks". In: *Neural information processing systems*. 1987, pp. 602–611.

[140] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. "Semidefinite relaxations for certifying robustness to adversarial examples". In: *Advances in Neural Information Processing Systems*. 2018, pp. 10877–10887.

[141] Benjamin Recht. "A tour of reinforcement learning: The view from continuous control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 2 (2019), pp. 253–279.

[142] Max Revay, Ruigang Wang, and Ian R Manchester. "A Convex Parameterization of Robust Recurrent Neural Networks". In: *IEEE Control Systems Letters* 5.4 (2020), pp. 1363–1368.

[143] Max Revay, Ruigang Wang, and Ian R Manchester. "Lipschitz Bounded Equilibrium Networks". In: *arXiv preprint arXiv:2010.01732* (2020).

[144] Yu Rong et al. "DropEdge: Towards Deep Graph Convolutional Networks on Node Classification". In: *ICLR 2020 : Eighth International Conference on Learning Representations*. 2020.

[145] Shankar Sastry. *Nonlinear systems: analysis, stability, and control*. Vol. 10. Springer Science & Business Media, 2013.

[146] Kathrin Schacke. "On the kronecker product". In: (2018).

[147] Carsten Scherer, Pascal Gahinet, and Mahmoud Chilali. "Multiobjective output-feedback control via LMI optimization". In: *IEEE Transactions on automatic control* 42.7 (1997), pp. 896–911.

[148] Lukas Schwenkel et al. "Model predictive control for linear uncertain systems using integral quadratic constraints". In: *arXiv preprint arXiv:2104.05444* (2021).

[149] Abigail See, Minh-Thang Luong, and Christopher D Manning. "Compression of neural machine translation models via pruning". In: *arXiv preprint arXiv:1606.09274* (2016).

[150] Uri Shaham, Yutaro Yamada, and Sahand Negahban. "Understanding Adversarial Training: Increasing Local Stability of Neural Nets through Robust Optimization". In: (2015). DOI: 10.1016/j.neucom.2018.04.027. eprint: arXiv:1511.05432.

[151] Nino Shervashidze et al. "Efficient graphlet kernels for large graph comparison". In: *Artificial Intelligence and Statistics*. 2009, pp. 488–495.

[152] Nino Shervashidze et al. "Weisfeiler-lehman graph kernels". In: *Journal of Machine Learning Research* 12.77 (2011), pp. 2539–2561.

[153] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[154] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014. URL: http://arxiv.org/abs/1312.6034.

[155] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[156] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[157] Johannes Stallkamp et al. "The German traffic sign recognition benchmark: a multi-class classification competition". In: *The 2011 international joint conference on neural networks*. IEEE. 2011, pp. 1453–1460.

[158] Ilya Sutskever et al. "On the Importance of Initialization and Momentum in Deep Learning". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, pp. III-1139–III-1147. URL: http://dl.acm.org/citation.cfm?id=3042817.3043064.

[159] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[160] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation." In: *NIPs*. Vol. 99. Citeseer. 1999, pp. 1057–1063.

[161] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.

[162] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[163] Victor Talpaert et al. "Exploring applications of deep reinforcement learning for real-world autonomous driving systems". In: *arXiv preprint arXiv:1901.01536* (2019).

[164] Gavin Taylor et al. "Training neural networks without gradients: A scalable admm approach". In: *International Conference on Machine Learning*. 2016, pp. 2722–2731.

[165] Mark M Tobenkin, Ian R Manchester, and Alexandre Megretski. "Convex parameterizations and fidelity bounds for nonlinear identification and reduced-order modelling". In: *IEEE Transactions on Automatic Control* 62.7 (2017), pp. 3679–3686.

[166] Bertrand Travacca. *Fenchel Young Neural Network on Synthetic Data, bertravacca GitHub repositories: implicit lifted Nets synthetic simulation for feedforward nn*. Version 1.0. Aug. 2019. DOI: 10.5281/zenodo.3364839. URL: https://zenodo.org/account/settings/github/repository/bertravacca/implicit-lifted_Nets-synthetic-simulation-for-feedforward-nn.

[167] Bertrand Travacca, Laurent El Ghaoui, and Scott Moura. "Implicit Optimization: Models and Methods". In: *2020 59th IEEE Conference on Decision and Control (CDC)*. 2020, pp. 408–415. DOI: 10.1109/CDC42340.2020.9304169.

[168] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[169] Petar Veličković et al. "Deep graph infomax". In: *arXiv preprint arXiv:1809.10341* (2018).

[170] Petar Veličković et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).

[171] Fangping Wan et al. "NeoDTI: neural integration of neighbor information from a heterogeneous network for discovering new drug–target interactions". In: *Bioinformatics* 35.1 (2019), pp. 104–111.

[172] Tiancai Wang, Xiangyu Zhang, and Jian Sun. "Implicit feature pyramid network for object detection". In: *arXiv preprint arXiv:2012.13563* (2020).

[173] Po-Wei Wang et al. "SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, Sept. 2019, pp. 6545–6554. URL: `http://proceedings.mlr.press/v97/wang19e.html`.

[174] Xiao Wang et al. "Heterogeneous graph attention network". In: *The World Wide Web Conference*. 2019, pp. 2022–2032.

[175] Daan Wierstra et al. "Solving deep memory POMDPs with recurrent policy gradients". In: *International conference on artificial neural networks*. Springer. 2007, pp. 697–706.

[176] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[177] Ezra Winston and J Zico Kolter. "Monotone operator equilibrium networks". In: *Advances in Neural Information Processing Systems* 33 (2020).

[178] Eric Wong and J. Zico Kolter. *Provable defenses against adversarial examples via the convex outer adversarial polytope*. 2017. eprint: `arXiv:1711.00851`.

[179] John Wright and Yi Ma. *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 2021.

[180] Felix Wu et al. "Simplifying Graph Convolutional Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019, pp. 6861–6871.

[181] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *arXiv preprint arXiv:1901.00596* (2019).

[182] Keyulu Xu et al. "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826* (2018).

[183] V.A. Yakubovich. "S-procedure in nonlinear control theory (in Russian)". In: *Vestnik Leningrad. Univ.* 1971.

[184] Pinar Yanardag and SVN Vishwanathan. "Deep graph kernels". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2015, pp. 1365–1374.

[185] Jaewon Yang and Jure Leskovec. "Defining and evaluating network communities based on ground-truth". In: *Knowledge and Information Systems* 42.1 (2015), pp. 181–213.

[186] He Yin, Peter Seiler, and Murat Arcak. "Stability analysis using quadratic constraints for systems with neural network controllers". In: *IEEE Transactions on Automatic Control* (2021).

[187] He Yin et al. "Imitation learning with stability and safety guarantees". In: *IEEE Control Systems Letters* (2021).

[188] Rex Ying et al. "Graph convolutional neural networks for web-scale recommender systems". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 974–983.

[189] Yong Yu et al. "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7 (2019), pp. 1235–1270.

[190] Jinshan Zeng et al. "Global convergence of block coordinate descent in deep learning". In: *arXiv preprint arXiv:1803.00225* (2018).

[191] Kaiqing Zhang, Bin Hu, and Tamer Basar. "Policy Optimization for $\mathcal{H}_2$ Linear Control with $\mathcal{H}_\infty$ Robustness Guarantee: Implicit Regularization and Global Convergence". In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 179–190.

[192] Marvin Zhang et al. "Learning deep neural network policies with continuous memory states". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 520–527.

[193] Muhan Zhang et al. "An end-to-end deep learning architecture for graph classification". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[194] Ziming Zhang and Matthew Brand. "Convergent Block Coordinate Descent for Training Tikhonov Regularized Deep Neural Networks". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1719–1728. ISBN: 978-1-5108-6096-4. URL: `http://dl.acm.org/citation.cfm?id=3294771.3294935`.

[195] Ziming Zhang and Matthew Brand. "Convergent block coordinate descent for training tikhonov regularized deep neural networks". In: *Advances in Neural Information Processing Systems*. 2017, pp. 1721–1730.

[196] Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. "Efficient training of very deep neural networks for supervised hashing". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1487–1495.

[197] Ziwei Zhang, Peng Cui, and Wenwu Zhu. "Deep learning on graphs: A survey". In: *IEEE Transactions on Knowledge and Data Engineering* (2020).

[198] Lingxiao Zhao and Leman Akoglu. "PairNorm: Tackling Oversmoothing in GNNs". In: *ICLR 2020 : Eighth International Conference on Learning Representations*. 2020.

[199] Yang Zheng, Yujie Tang, and Na Li. "Analysis of the Optimization Landscape of Linear Quadratic Gaussian (LQG) Control". In: *arXiv preprint arXiv:2102.04393* (2021).

[200] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *arXiv preprint arXiv:1812.08434* (2018).

[201] Kemin Zhou, John Comstock Doyle, Keith Glover, et al. *Robust and optimal control*. Vol. 40. Prentice hall New Jersey, 1996.

[202] Michael Zhu and Suyog Gupta. "To prune, or not to prune: exploring the efficacy of pruning for model compression". In: *arXiv preprint arXiv:1710.01878* (2017).