Implicit Learning in Deep Models: Enhancing Extrapolation Power and Sparsity

Alicia Tsai

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-214
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-214.html

December 14, 2024

Implicit Learning in Deep Models: Enhancing Extrapolation Power and Sparsity

By

Alicia Tsai

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Laurent El Ghaoui, Chair
Professor Alper Atamtürk
Professor Murat Arcak

Fall 2024

Implicit Learning in Deep Models: Enhancing Extrapolation Power and Sparsity

Abstract

Implicit Learning in Deep Models: Enhancing Extrapolation Power and Sparsity

by

Alicia Tsai

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Laurent El Ghaoui, Chair

This thesis investigates the transformative potential of implicit models in deep learning, with a focus on their capabilities to tackle challenges in extrapolation, sparsity, and robustness. Unlike traditional neural networks that rely on predefined, layer-by-layer architectures, implicit models define outputs through equilibrium equations, enabling dynamic adaptability and compact representations. We present a comprehensive framework for understanding implicit models, encompassing theoretical foundations of well-posedness, algorithms for constrained sparsification, and robustness analyses. The versatility and effectiveness of implicit models are demonstrated across diverse tasks, including mathematical operations, temporal forecasting, and geographical extrapolation, where they consistently outperform non-implicit baselines, particularly under distribution shifts. Key contributions include the introduction of the sensitivity matrix and error bounds, which provide interpretable robustness measurements and facilitate the generation of adversarial attacks. We also highlight depth adaptability and closed-loop feedback as fundamental mechanisms driving the superior extrapolation performance of implicit models. This work establishes implicit models as a robust, scalable, and interpretable alternative paradigm for neural network design, with significant implications for addressing real-world challenges involving noisy, sparse, and out-of-distribution data.

To my family and my lovely fiancé.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Laurent El Ghaoui. His unwavering encouragement and support throughout my Ph.D. have been a constant source of motivation. I deeply admire his intellectual curiosity and entrepreneurial spirit, which have profoundly shaped my academic journey. Beyond guiding my research, he has inspired me to thrive as a researcher who not only explores innovative ideas but also collaborates effectively, leads teams, and communicates research with audiences beyond our field. His new chapter as Dean at VinUniversity opened extraordinary opportunities for me—spending nearly half of my Ph.D. in Vietnam and traveling to many countries. These experiences have enriched my life in ways I will always cherish. His push to bring research closer to solving real-world problems has been instrumental in shaping my perspective and aspirations.

I am incredibly thankful to my thesis committee members, Professor Alper Atamtürk and Professor Murat Arcak. Alper's advice on the implicit modeling project directly contributed to the sparsification work in Chapter 3, and I am grateful for his warm welcome to the Berkeley chapter of the AI Research Institute for Advances in Optimization (AI4OPT). Murat provided invaluable guidance on early drafts of the robustness project in Chapter 4 and recommended me to serve as the organizer for the EECS Visit Day for the Control, Intelligent Systems, and Robotics Program, which was an enriching experience. I also thank Professor Khalid M. Mosalam for his role on my qualification committee. Khalid introduced me to the field of Earthquake Engineering, and our early collaborations taught me the value of interdisciplinary work, which has greatly influenced my research approach.

I especially want to thank my mentors who guided and inspired me even before my Ph.D. journey began—Professor Jason Hsu, Dr. Wenwey Hseush, Professor Shou-De Lin, Professor Ling-Chieh Kung, and Professor Snow Tseng from National Taiwan University. Their encouragement and belief have been a cornerstone of my academic and professional growth, providing me with the foundation to pursue my dreams and thrive throughout my career.

I am grateful to the El Ghaoui Research Group—Forest Yang, Armin Askari, Zihao Chen, Fangda Gu, and Geoff Negiar—for their camaraderie and the fun, often random, conversations about Ph.D. life at Berkeley. My heartfelt thanks also go to my collaborators, Selim Gunay and Lindsay Chuang, who taught me so much about structural engineering and seismology.

During my Ph.D., I was fortunate to intern at leading industry research labs—Amazon Alexa AI, Apple Research, and Google DeepMind. I am grateful for the brilliant colleagues I had the privilege to collaborate with and for the knowledge I gained from them.

A huge thanks to Zong-Yen Wu and Ricky Liu for being my pillars of support during my Ph.D. and for keeping our 1903 apartment filled with life and delicious food. To my friends at Berkeley—Mu-Ti, Tobey, Alice, Ming-Yu, Shangpo, Lili, Yi-Hsuan, Ping-Huang, Jeffery, Szu-Chi, Alex, Angela, Jennifer, and the BATS community—thank you for making

my Berkeley journey so memorable. To my beloved friends across the globe—Chava and Sherry—thank you for the joy, warmth, and friendship you have shared with me for nearly 15 years. Your presence in my life has been a constant source of happiness.

I am also deeply inspired by the friends and colleagues who believe in community outreach. Chi-Yi, Wendy, Sara, Melissa, Sarah, and Yinghan, your dedication to making a positive impact has been immensely inspiring. Organizations such as Women in Data Science Taipei, Women in Machine Learning, Taiwan Data Science Association, and Delta Analytics have been incredible sources of learning and growth.

Lastly, I thank my family. My parents, Tzu-Hsien and Ming-Chao, have been my strongest supporters, always believing in me and showering me with unconditional love. To my fiancé, Chien-Wei, thank you for always being by my side, especially during the lowest points of my Ph.D. when my anxiety was at its peak. Your strength and support have been my anchor.

To everyone who has been part of this journey, thank you for riding it with me and making it a truly unforgettable chapter of my life.

Chapter 1

Introduction

Artificial Neural Networks (ANNs) have revolutionized machine learning by providing a framework for approximating complex, non-linear functions. Inspired by the structure of biological neurons, ANNs consist of layers of interconnected nodes that process input data and extract meaningful representations. While shallow networks are effective for some problems, their ability to model complex data is limited. The introduction of *Deep Learning*, characterized by networks with many layers, has dramatically expanded the capability of neural networks across domains like computer vision, natural language processing, and time-series analysis.

Deep learning leverages hierarchical feature extraction, where lower layers capture simple patterns while deeper layers extract abstract representations. This capability has enabled breakthroughs in areas like image classification, speech recognition, and autonomous systems. However, despite their widespread success, traditional feed-forward deep neural networks (DNNs) face challenges, including their reliance on fixed architectures, difficulty in extrapolating to out-of-distribution (OOD) data, and sensitivity to adversarial perturbations.

Implicit models represent a fundamental shift in neural network design. Unlike traditional deep neural networks (DNNs), which process data through a predefined sequence of layers, implicit models determine outputs by solving equilibrium equations that define the relationships between inputs and internal states. This approach allows for dynamic adaptation to the complexity of tasks, as the model's depth is not fixed but emerges from the equilibrium computation.

Examples of implicit models span a diverse range of applications and designs. *Physics-Informed Neural Networks (PINNs)* integrate physical laws directly into the learning process, ensuring that predictions adhere to known scientific principles. By embedding partial differential equations into their architecture, PINNs solve complex physical problems while maintaining fidelity to established theories [76]. While PINNs incorporate physical constraints via embedded equations, the implicit models in this thesis generalize this approach by defining their behavior through equilibrium equations, enabling them to represent a wide variety

of systems beyond physical laws. *Boltzmann Machines*, introduced by Geoffrey Hinton [47], consist of symmetrically connected, neuron-like units that make stochastic decisions about their states. This symmetric connectivity allows them to model complex probability distributions and perform associative memory tasks. In contrast, the implicit models explored in this thesis leverage asymmetric weight matrices, providing a more dynamic framework for learning and adaptation. This asymmetry facilitates improved generalization and robustness under distribution shifts.

Deep Equilibrium Models (DEQs) closely align with the equilibrium-based approach of the implicit models studied here. DEQs define the output of the network as the fixed point of a single-layer transformation, effectively simulating infinite-depth networks without the computational burden of stacking layers. This characteristic makes DEQs particularly efficient for capturing long-range dependencies [8]. Similarly, *Neural Ordinary Differential Equations (ODEs)* extend the equilibrium framework to continuous-time dynamics by parameterizing the derivative of the hidden state using neural networks. This approach offers a principled method for modeling time-series data and dynamic systems with continuous-time behavior, bridging the gap between traditional deep learning and differential equations [16].

The implicit models discussed above share foundational principles with those explored in this thesis, particularly in their reliance on equilibrium-based formulations and their adaptability across diverse tasks. Together, these examples highlight the versatility of implicit models as a unifying paradigm. They demonstrate how equilibrium-driven frameworks provide a robust foundation for solving complex, real-world problems and offer insights into the development of scalable, interpretable, and adaptable neural networks.

This thesis discusses the theoretical properties and practical applications of implicit models, contrasting them with traditional deep learning architectures and investigating their potential to overcome the limitations of standard neural networks. The remainder of this thesis is organized as follows:

Chapter 2: Implicit Learning in Deep Models This chapter introduces implicit prediction rules and explores the theoretical underpinnings of implicit models, including system well-posedness and composition. We highlight how implicit models extend and generalize traditional neural network architectures, laying the groundwork for their application in deep learning.

Chapter 3: Constrained Implicit Learning Here, we develop a constrained implicit learning framework designed to sparsify neural networks by imposing constraints on their structure. An efficient algorithm for network sparsification is presented, along with numerical experiments that demonstrate the effectiveness of the method in maintaining model accuracy while reducing complexity.

Chapter 4: Robustness Analysis via Implicit Representation This chapter discusses the robustness of implicit models against adversarial attacks. We introduce the sensitivity matrix, a novel tool for evaluating input-output robustness. The chapter also explores how this matrix can be leveraged for generating adversarial samples and improving model defenses.

Chapter 5: The Extrapolation Power of Implicit Models The focus shifts to the extrapolation capabilities of implicit models, essential for generalizing beyond training data. Through a series of mathematical and real-world tasks, including time-series forecasting and earthquake location prediction, we demonstrate the adaptability and superior performance of implicit models in out-of-distribution scenarios.

By investigating the theoretical and practical advantages of implicit models, this thesis aims to contribute to the growing body of work that seeks to make neural networks more robust, efficient, and generalizable.

Chapter 2

Implicit Learning in Deep Models

2.1 Introduction

Deep learning has revolutionized artificial intelligence by achieving remarkable success across diverse domains, including computer vision, natural language processing, and scientific computing [59, 32]. At the heart of traditional deep learning models lies the explicit layer-by-layer architecture, where computations flow sequentially from input to output through a fixed-depth network [43]. While effective, this rigid structure can struggle to generalize to complex, dynamic, or out-of-distribution data, limiting its adaptability and robustness [78].

Implicit learning introduces a paradigm shift in how deep models are designed and optimized. Instead of relying on predefined, feedforward architectures, implicit models define their behavior through equilibrium equations, where the outputs of intermediate layers are solutions to implicit functions [7, 25]. These models, often referred to as equilibrium or implicit models, represent a departure from traditional architectures by focusing on the steady-state behavior of the network rather than its depth-dependent structure [2].

This equilibrium formulation offers several key advantages. First, implicit models adapt their effective depth to the complexity of the task, solving the equilibrium equation iteratively until convergence [42]. This flexibility allows them to model more intricate dependencies compared to their explicit counterparts. Second, implicit models inherently incorporate feedback mechanisms within their computations, enabling recurrent information flow during forward passes. Such feedback loops mirror biological neural systems and improve robustness and stability under challenging conditions, such as adversarial attacks or distribution shifts [61, 79]. By reframing neural networks as implicit systems, this approach unlocks new opportunities for designing models that are both robust and efficient, paving the way for further exploration into their theoretical foundations and practical applications in real-world scenarios [22, 50].

The relationship between implicit models and physics-based formulations has been explored in the context of differential equations. Physics-inspired models like Neural ODEs [17] and

Hamiltonian Neural Networks [39] leverage continuous-time dynamics to encode conservation laws and stability directly into their learning processes. These models share common ground with implicit methods in their ability to model long-term dependencies and provide robust solutions, particularly for applications in dynamical systems and scientific computing [50].

Boltzmann machines, a class of stochastic recurrent neural networks, have historically been designed with symmetric weight matrices to ensure convergence. Recent works have explored the use of asymmetric weight matrices to allow for richer representational power while maintaining stability [1, 46]. The equilibrium nature of these models aligns closely with the fixed-point solutions of implicit models, offering a promising avenue for further exploration in combining stochasticity and implicit dynamics.

Recent development of Deep equilibrium models, introduced by Bai et al. [7], are a pioneering class of implicit models that solve for a fixed-point equilibrium instead of propagating through explicit layers. These models allow for infinite depth at a fixed computational cost and demonstrate state-of-the-art performance in tasks such as sequence modeling and image recognition. DEQs have also been extended to incorporate memory efficiency and more robust solvers, enhancing their practical usability [25].

By integrating insights from these related fields, this work advances the understanding of implicit learning frameworks. Specifically, we investigate the unique properties of implicit models, including their adaptability to complex data distributions, their ability to incorporate feedback loops, and their robustness under adversarial or noisy conditions. These attributes position implicit models as a versatile and efficient framework for addressing the limitations of traditional deep neural networks, particularly in safety-critical and dynamically evolving environments [61, 42].

2.2 Implicit prediction rules

This thesis explores a novel class of deep learning models that utilize implicit prediction rules. Unlike traditional neural networks, which are based on a recursive, layer-by-layer computation, implicit models predict outcomes by solving a fixed-point equation involving a single "state" vector $x \in \mathbb{R}^n$. The prediction process is defined as follows: for a given input vector u, the predicted output $\hat{y}(u)$ is computed as:

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

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

Here, $\phi : \mathbb{R}^n \to \mathbb{R}^n$ represents a nonlinear activation function, and the matrices A, B, C, D contain the model parameters.

In this framework, the vector $x \in \mathbb{R}^n$ serves as a "state" that encodes n latent features extracted from the input u through the equilibrium equation (2.1b). Unlike explicit models,

which compute this state via forward propagation through a fixed number of layers, implicit models only provide x *implicitly* by solving the equilibrium equation. This flexibility enables implicit models to adaptively adjust their effective depth based on the input data, an important distinction highlighted in prior works such as [7] and [2].

However, solving the equilibrium equation is not always straightforward. The equation may lack a solution or fail to be unique, raising critical concerns about well-posedness. For simplicity, the formulation above excludes bias terms. These can be easily incorporated by increasing the input vector to $(u, 1)$, thereby increasing the column dimension of B by one. This minor adjustment maintains the generality of the framework.

Interestingly, implicit models encompass most current neural network architectures as special cases. For instance, recurrent neural networks (RNNs) and certain residual networks can be reformulated as implicit models under specific constraints [17]. Implicit models, however, represent a significantly broader class. They offer higher representational capacity, as measured by the number of trainable parameters for a given hidden feature dimension. Additionally, implicit models naturally accommodate feedback loops and cyclic dependencies within the network, which are typically disallowed in conventional deep learning paradigms [9, 17]. This implicit approach to learning introduces a fundamentally different perspective on model design, enabling more efficient, robust, and scalable solutions for tasks involving dynamic or complex data distributions.

Notation. Throughout this thesis, we adopt the following notation for matrices, vectors, and associated operations. For a matrix U, $|U|$ denotes the matrix obtained by taking the absolute value of each entry of U, while U_+ represents the matrix formed by retaining only the positive part of each entry. For a vector v, $\mathbf{diag}(v)$ refers to the diagonal matrix with the entries of v along its diagonal. Conversely, for a square matrix V, $\mathbf{diag}(V)$ denotes the vector formed by the diagonal entries of V.

The vector $\mathbf{1}$ represents a vector of ones, with its size inferred from context. The Hadamard (elementwise) product of two n-vectors x and y is denoted by $x \odot y$. To represent the sum of the largest k entries of a vector z, we use the notation $s_k(z)$.

For a matrix A and integers $p, q \geq 1$, the induced matrix norm is defined as:

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

In particular, when $p = q = \infty$, this reduces to the l_∞-induced norm of A, also referred to as the *max-row-sum norm*, expressed as:

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

We compactly represent the set $\{1, \ldots, L\}$ as $[L]$. For an n-vector z partitioned into L blocks, where $z = (z_1, \ldots, z_L)$ with $z_l \in \mathbb{R}^{n_l}$ for $l \in [L]$ and $n_1 + \ldots + n_L = n$, the L-vector

of norms is defined as:

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

Finally, for any square, non-negative matrix M, there exists a real eigenvalue that is larger than or equal to the modulus of all other eigenvalues. This eigenvalue, known as the *Perron-Frobenius eigenvalue* [66], is denoted by $\lambda_{PF}(M)$.

2.3 System Well-posedness

Blockwise Lipschitz (BLIP) Continuity Condition

In this section, we focus on activation maps ϕ that satisfy the *Blockwise Lipschitz (BLIP)* continuity condition, a property commonly held by popular activation functions.

Definition 1 (Blockwise Lipschitz (BLIP) Condition)**.** *An activation map ϕ satisfies the* BLIP condition *if it adheres to the following two properties:*

1. ***Blockwise Property:*** *The map ϕ operates in a blockwise fashion. Specifically, there exists a partition of n as $n = n_1 + \ldots + n_L$, such that for any vector z partitioned into corresponding blocks $z = (z_1, \ldots, z_L)$, where $z_l \in \mathbb{R}^{n_l}$ for $l \in [L]$, the map ϕ can be written as:*
$$\phi(z) = (\phi_1(z_1), \ldots, \phi_L(z_L)),$$
where $\phi_l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$ are block-specific maps for $l \in [L]$.

2. ***Lipschitz Continuity:*** *Each block-specific map ϕ_l is Lipschitz-continuous with a constant $\gamma_l > 0$, with respect to the l_{p_l}-norm for some integer $p_l \geq 1$:*
$$\forall u, v \in \mathbb{R}^{n_l}, \quad \|\phi_l(u) - \phi_l(v)\|_{p_l} \leq \gamma_l \|u - v\|_{p_l}.$$

A special case of interest is the *Componentwise Non-Expansive (CONE)* maps, characterized by $n_l = 1$ and $\gamma_l = 1$ for all $l \in [L]$.

Definition 2 (Componentwise Non-Expansive (CONE) Map)**.** *An activation map ϕ is said to be a* CONE map *if it satisfies the following condition:*

$$\forall u, v \in \mathbb{R}^n, \quad |\phi(u) - \phi(v)| \leq |u - v|, \tag{2.3}$$

where the inequality and absolute values are taken componentwise.

Examples of CONE maps include: ReLU ($\phi(x) = \max(0, x)$), Leaky ReLU and its variants, Hyperbolic tangent (tanh), Sigmoid function, all applied componentwise to vector inputs.

Our framework also accommodates maps that are not strictly componentwise, such as the *softmax* function. The softmax operation acts on an n-vector z as:

$$\text{SoftMax}(z) := \left(\frac{e^{z_i}}{\sum_j e^{z_j}} \right)_{i \in [n]} . \tag{2.4}$$

The softmax function is 1-Lipschitz-continuous with respect to the l_1-norm [26], making it compatible with the proposed framework despite its non-componentwise nature.

Well-posed Systems

We analyze the prediction rule (2.1a), which maps an input $u \in \mathbb{R}^p$ to a predicted output $\hat{y}(u) \in \mathbb{R}^q$. The equilibrium equation (2.1b), which governs the hidden state x, does not always guarantee a well-defined or unique solution. For example, Figure 2.1 illustrates a scalar case where multiple solutions can arise.

To ensure that the state x has a unique solution, we impose a well-posedness condition on the $n \times n$ matrix A, formalized as follows:

Definition 3 (Well-posedness Property)**.** *A matrix $A \in \mathbb{R}^{n \times n}$ is said to be* well-posed *for the activation map ϕ (denoted $A \in WP(\phi)$) if, for any n-vector b, the equation:*

$$x = \phi(Ax + b) \tag{2.5}$$

admits a unique solution $x \in \mathbb{R}^n$.

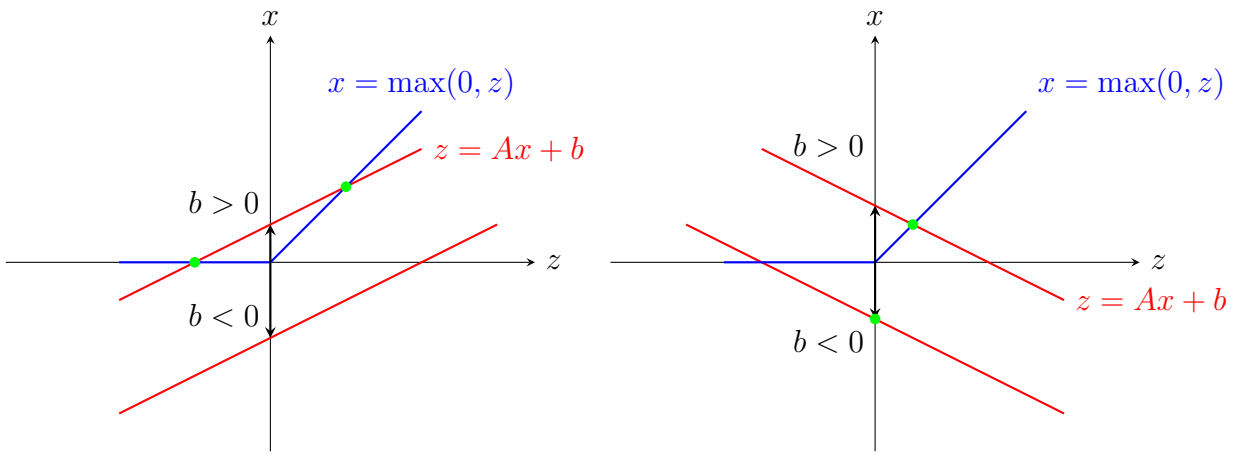

Figure 2.1: **Left:** equation $x = \phi(Ax + b)$ has two or no solutions, depending on the sign of b. **Right:** solution is unique for every b.

There are several classes of matrices that satisfy the well-posedness property. One notable example is the class of strictly upper-triangular matrices, which are well-posed for

any activation map that acts component-wise. This class naturally arises when modeling feedforward neural networks within the framework of implicit models, as the strictly upper-triangular structure ensures the absence of feedback loops, simplifying the solution of the equilibrium equation.

Tractable Sufficient Conditions for Well-posedness

Our goal is to identify numerically tractable constraints on A that ensure it satisfies the well-posedness property.

Assume that ϕ is a BLIP map as defined in (1). Partition the matrix A according to the tuple $(n_1, \ldots, n_L)$ into blocks $A_{ij} \in \mathbb{R}^{n_i \times n_j}$ for $1 \leq i, j \leq L$. Using this partition, define an $L \times L$ matrix of induced norms $N(A, \gamma) \in \mathbb{R}_+^{L \times L}$, where the elements are given for $i, j \in [L]$ by:

$$(N(A,\gamma))_{ij} := \gamma_i \|A_{ij}\|_{p_j \to p_i} = \gamma_i \max_{\xi} \|A_{ij}\xi\|_{p_i} \quad \text{subject to } \|\xi\|_{p_j} \leq 1. \tag{2.6}$$

In the case of CONE maps as defined in (2), the vector γ is the all-ones vector, and the matrix $N(A, \gamma)$ simplifies to $|A|$, the matrix of element-wise absolute values.

The sufficient condition for well-posedness, stated next, relies on the contraction mapping theorem [82, p.83].

Theorem 1 (Perron-Frobenius (PF) Sufficient Condition for Well-posedness for BLIP Activation)**.** *Assume that ϕ satisfies the BLIP condition* (1)*. Then, the matrix A is well-posed with respect to ϕ if:*

$$\lambda_{PF}(N(A,\gamma)) < 1, \tag{2.7}$$

where $N(A, \gamma)$ is the matrix of induced norms defined in (2.6)*. Under this condition, for any n-vector b, the solution to the equation* (2.5) *can be computed using the fixed-point iteration:*

$$x(0) = 0, \quad x(t+1) = \phi(Ax(t) + b), \quad t = 0, 1, 2, \ldots. \tag{2.8}$$

Moreover, if ϕ is a CONE map as defined in (2)*, the PF condition* (2.7) *simplifies to:*

$$\lambda_{PF}(|A|) < 1.$$

Proof of Theorem (1). Let $b \in \mathbb{R}^n$. We aim to show that for the Picard iteration (2.8), the following holds for every $t \geq 1$:

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

where η is the vector of norms defined in (2.2). For every $l \in [L]$ and $t \geq 0$, we have:

$$\begin{aligned}
[\eta(x(t+1) - x(t))]_l &= \|\phi_l([Ax(t) + b]_l) - \phi_l([Ax(t-1) + b]_l)\|_{p_l} \quad \text{(from (2.8))} \\
&\leq \gamma_l \|[A(x(t) - x(t-1))]_l\|_{p_l} \\
&= \gamma_l \left\| \sum_{h \in [L]} A_{lh}(x(t) - x(t-1))_h \right\|_{p_l} \\
&\leq \gamma_l \sum_{h \in [L]} \|A_{lh}\|_{p_h \to p_l} \|x_h(t) - x_h(t-1)\|_{p_h} \\
&= [N(A, \gamma)\eta(x(t) - x(t-1))]_l,
\end{aligned}$$

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

Now, assume that $\lambda_{PF}(M) < 1$, as posited in Theorem 1. By the Perron-Frobenius theorem, $I - M$ is non-singular, and all other eigenvalues λ of $N(A, \gamma)$ satisfy $|\lambda| \leq \lambda_{PF}(N(A, \gamma)) < 1$.

To prove the existence of a solution to the equilibrium equation, we show that the sequence of Picard iterates is Cauchy. For every $t, \tau \geq 0$, we have:

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

Since M^t converges to 0 as $t \to +\infty$, the sequence of Picard iterates is Cauchy. Define $w \in \mathbb{R}^L_+$ as:

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

It follows that $\eta(x(t+\tau) - x(t)) \to 0$ as $t \to +\infty$, which implies that $\{x(t)\}$ converges to $x \in \mathbb{R}^n$, satisfying $x = \phi(Ax + b)$. This proves the existence of a solution.

To prove uniqueness, suppose $x^1, x^2 \in \mathbb{R}^n$ are two solutions to the equilibrium equation. By the theorem's hypothesis:

$$\eta(x^1 - x^2) \leq M\eta(x^1 - x^2) \leq M^k \eta(x^1 - x^2), \quad \forall k \geq 1.$$

As $M^k \to 0$ as $k \to +\infty$, it follows that $x^1 = x^2$, establishing uniqueness. □

We now discuss several important observations and implications of the well-posedness conditions and the PF sufficient condition.

Remark 1. *The fixed-point iteration* (2.8) *exhibits linear convergence. Each iteration involves a matrix-vector product, making the computational complexity of solving the equilibrium equation comparable to a forward pass through a network of similar size. This computational efficiency highlights the practical feasibility of using implicit models in real-world scenarios.*

Remark 2. *The Perron-Frobenius (PF) condition $\lambda_{PF}(N(A,\gamma)) < 1$ provides a strong guarantee of well-posedness but is not convex in A. To address this limitation, the convex condition $\|N(A,\gamma)\|_\infty < 1$ can serve as a sufficient alternative. This sufficiency follows from the inequality:*

$$\|M\|_\infty \geq \lambda_{PF}(|M|),$$

which holds for any square matrix M. The convex condition offers a more tractable approach for practical verification of well-posedness.

Remark 3. *The PF condition in Theorem 1 can be conservative in certain cases. For instance, consider triangular matrices, which often arise in feedforward architectures. A triangular matrix A is well-posed with respect to the ReLU activation function if and only if $\boldsymbol{diag}(A) < \mathbf{1}$, as established in (2). For such matrices, the equilibrium equation can be solved explicitly via the backward recursion:*

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

Notably, such matrices may violate the PF condition $\lambda_{PF}(|A|) < 1$. For example, if $A_{11} < -1$, we have $\lambda_{PF}(|A|) > 1$, demonstrating that the PF condition can be overly restrictive.

Remark 4. *The well-posedness property and the PF sufficient condition exhibit invariance under row and column permutations, provided ϕ acts componentwise. Specifically, if A is well-posed with respect to a componentwise CONE map ϕ, then for any permutation matrix P, the matrix $PAP^\top$ is also well-posed with respect to ϕ. Similarly, the condition $\lambda_{PF}(N(A,\gamma)) < 1$ remains invariant under such permutations. This invariance extends naturally to the more general BLIP case, emphasizing the flexibility of the framework in modeling complex systems.*

In summary, the well-posedness property provides a crucial foundation for the implicit modeling framework, ensuring the existence and uniqueness of solutions to equilibrium equations. The PF sufficient condition $\lambda_{PF}(N(A,\gamma)) < 1$, while conservative in some cases, offers a practical and numerically efficient approach to verifying well-posedness, particularly when paired with convex relaxations such as $\|N(A,\gamma)\|_\infty < 1$. Furthermore, the invariance of well-posedness under row and column permutations highlights the flexibility of the framework in accommodating different structural representations. These results not only establish a robust theoretical foundation but also highlight the practical feasibility of implicit models.

2.4 Composition of Implicit Models

Implicit models can be naturally composed using matrix algebra, allowing for flexible and modular designs, as illustrated in Figure 2.2. In certain cases, the structure of the composition ensures that well-posedness is preserved, as shown in the following result.

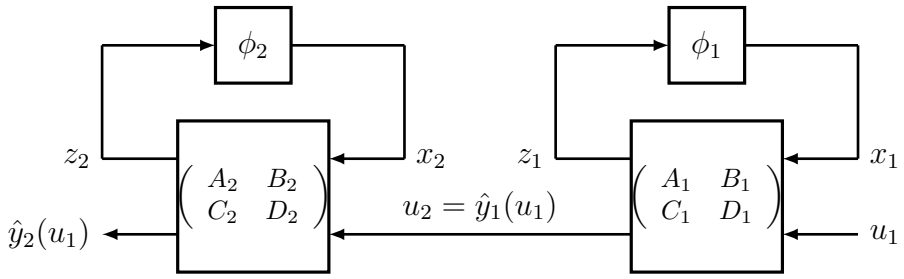

Figure 2.2: Cascade connection of two implicit models to be read from right to left, consistent with matrix-vector multiplication rules.

Theorem 2 (Well-posedness of Block-Triangular Matrices with Componentwise Activation)**.** *Assume that the activation map ϕ acts componentwise. Consider the upper block-triangular matrix:*

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

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

This result highlights an important structural property: the well-posedness of a block-triangular matrix depends solely on the well-posedness of its diagonal blocks. This property simplifies the analysis and design of complex implicit models by enabling modular verification of well-posedness. For example, when building a large implicit model, each subsystem (corresponding to a diagonal block) can be analyzed independently, ensuring the entire system remains well-posed.

Proof of Theorem 2. We express the equilibrium equation $x = \phi(Ax + b)$ in block form:

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

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

Sufficiency. Assume that A_{11} and A_{22} are well-posed with respect to ϕ. - From the second equation $x_2 = \phi(A_{22}x_2 + b_2)$, the well-posedness of A_{22} ensures a unique solution x_2^* for any b_2. - Substituting $x_2 = x_2^*$ into the first equation, it becomes:

$$x_1 = \phi(A_{11}x_1 + A_{12}x_2^* + b_1).$$

The well-posedness of A_{11} ensures that this equation has a unique solution x_1 for any b_1. Thus, the entire system has a unique solution $x = (x_1, x_2)$, proving that A is well-posed.

Necessity Assume that A is well-posed with respect to ϕ. - Consider the second equation $x_2 = \phi(A_{22}x_2 + b_2)$. The well-posedness of A implies that this equation must have a unique solution x_2^* for any b_2. Hence, A_{22} is well-posed with respect to ϕ. - To prove that A_{11} is well-posed, set $b_2 = 0$ and let b_1 be arbitrary. The system becomes:

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

Since A_{22} is well-posed, the second equation has a unique solution x_2^*. Substituting $x_2 = x_2^*$ into the first equation, it reduces to:

$$x_1 = \phi(A_{11}x_1 + b_1 + A_{12}x_2^*).$$

For the well-posedness of A, this equation must have a unique solution x_1 for any b_1, which implies that A_{11} is well-posed with respect to ϕ. Thus, the necessity of the condition is established.

Combining sufficiency and necessity, we conclude that A is well-posed with respect to ϕ if and only if A_{11} and A_{22} are well-posed with respect to ϕ. □

This result confirms the earlier claim that when ϕ is the ReLU activation function, an upper-triangular matrix $A \in \mathrm{WP}(\phi)$ if and only if $\mathbf{diag}(A) < \mathbf{1}$. A similar result holds for lower block-triangular matrices of the form:

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

where $A_{21} \in \mathbb{R}^{n_2 \times n_1}$ is arbitrary.

The framework can be extended to activation maps ϕ that satisfy the *Blockwise Lipschitz (BLIP)* condition. In this case, we assume that the partition of A into blocks is consistent with the block structure of ϕ. This scenario naturally arises when implicit models are composed from well-posed blocks, as we will see later. The following theorem formalizes this result.

Theorem 3 (Well-posedness of Block-Triangular Matrices with Blockwise Activation)**.** *Assume that the matrix A can be written as:*

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

where $A_{ii} \in \mathbb{R}^{n_i \times n_i}$ for $i = 1, 2$, and that the activation map ϕ acts blockwise. Specifically, there exist two maps ϕ_1 and ϕ_2 such that:

$$\phi((z_1, z_2)) = (\phi_1(z_1), \phi_2(z_2)), \quad \forall z_i \in \mathbb{R}^{n_i},\ i = 1, 2.$$

Then A is well-posed with respect to ϕ if and only if for $i = 1, 2$, the matrices A_{11} and A_{22} are well-posed with respect to ϕ_1 and ϕ_2, respectively.

This theorem extends the well-posedness property from componentwise activation functions (as in Theorem 2) to blockwise activation functions that satisfy the BLIP condition. It highlights the modularity of implicit models, where well-posedness of the system can be analyzed in terms of its constituent blocks. This result is especially useful when designing and composing complex models from simpler, well-posed components.

Using the results established above, we can preserve the well-posedness of implicit models through composition. For instance, consider two models with matrix parameters (A_i, B_i, C_i, D_i) and activation functions ϕ_i, $i = 1, 2$. We define a *cascaded prediction rule* as follows:

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

This cascaded rule can be expressed in the standard form of (P), where:

$$x = (x_2, x_1), \quad \phi((z_2, z_1)) = (\phi_2(z_2), \phi_1(z_1)),$$

and the overall system matrices are given by:

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

By Theorem 3, the cascaded prediction rule is well-posed for the blockwise activation map $\phi((z_2, z_1)) = (\phi_2(z_2), \phi_1(z_1))$ if and only if each individual prediction rule is well-posed. In other words, the modular well-posedness of each component ensures the well-posedness of the cascaded system. This result highlights the compositional nature of implicit models, enabling the design of complex, multi-layer systems while maintaining the foundational property of well-posedness. Such modularity is particularly advantageous in applications requiring scalable and interpretable models, as well-posedness can be verified independently for each subsystem.

In both cascade and parallel connections, the triangular structure of the matrix A in the composed system ensures that the Perron-Frobenius (PF) sufficient condition for well-posedness is satisfied for the composed system if and only if it holds for each sub-system. This property highlights the modularity of implicit models, allowing for scalable composition while preserving well-posedness.

However, multiplicative connections present additional challenges, as they are not generally Lipschitz-continuous unless the inputs are bounded. Specifically, consider two activation maps ϕ_i, $i = 1, 2$, that are Lipschitz-continuous with constants γ_i and bounded such that $|\phi_i(v)| \leq c_i$ for all v. In this case, 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 multiplicative connections frequently arise in the context of attention mechanisms in neural networks, which employ bounded activation functions like tanh. The boundedness of these activations ensures the Lipschitz property for multiplicative connections, enabling their integration into well-posed implicit models.

2.5 Implicit Models of Deep Neural Networks

A wide variety of deep neural networks can be represented as implicit models, including convolutional networks, recurrent networks, attention mechanisms, residual connections, and more. This unified perspective offers a powerful framework for analyzing and designing neural network architectures.

By leveraging the composition rules outlined in §2.4, we can focus on modeling individual layers, as a deep neural network is essentially a cascade of such layers. Each layer can be expressed as an implicit model, and when layers are composed in a cascade fashion, the overall model inherits a block-Lipschitz structure in its activation map. Moreover, the strictly triangular structure of the matrix A naturally emerges from this composition.

This structure has significant implications for well-posedness. Specifically, the triangular nature of A ensures that the implicit models derived from such neural networks are well-posed. In particular, the corresponding Perron-Frobenius eigenvalue of the matrix $N(A, \gamma)$, defined in (2.6), is zero because $N(A, \gamma)$ is strictly triangular. This guarantees that the equilibrium equation of the implicit model has a unique solution.

We can always ensure that the resulting implicit model satisfies the stronger norm condition for well-posedness mentioned in (2). Specifically, for a CONE map ϕ, the stronger condition $\|A\|_\infty < 1$ can always be achieved by appropriately scaling the weight matrices of the network's layers and using a scaled version of the state vector x. This scaling preserves the model's functionality while ensuring compliance with the sufficient condition for well-posedness, thereby improving the stability and robustness of the implicit model.

Dense Feedforward Networks

To illustrate the construction of an implicit model, we consider a fully dense feedforward network. The prediction rule for a feedforward network with $L > 1$ fully connected layers is given as follows:

$$\hat{y}(u) = W_L x_L, \quad x_{l+1} = \phi_l(W_l x_l), \quad x_0 = u, \tag{2.9}$$

where:

- $W_l \in \mathbb{R}^{n_{l+1} \times n_l}$ are weight matrices,
- $\phi_l : \mathbb{R}^{n_{l+1}} \to \mathbb{R}^{n_{l+1}}$ are activation maps,
- $l = 1, \ldots, L$.

We can represent this feedforward network as an implicit model in the form of (2.1a), where the state vector is $x = (x_L, \ldots, x_1)$ and the system matrices are:

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{cccc|c} 0 & W_{L-1} & \ldots & 0 & 0 \\ & 0 & \ddots & \vdots & \vdots \\ & & \ddots & W_1 & 0 \\ & & & 0 & W_0 \\ \hline W_L & 0 & \ldots & 0 & 0 \end{array}\right), \tag{2.10}$$

The activation function ϕ is defined blockwise and operates on an n-vector $z = (z_L, \ldots, z_1)$ as:

$$\phi(z) = (\phi_L(z_L), \ldots, \phi_1(z_1)).$$

The system is well-posed due to the strictly upper triangular structure of A, regardless of the specific values of A. The equilibrium equation:

$$x = \phi(Ax + Bu),$$

is efficiently solved via backward block substitution, which corresponds to a standard forward pass through the network.

If the state vector x lists the hidden variables in their natural order (rather than in reverse), the matrix A becomes strictly *lower* triangular. This alternative representation also preserves well-posedness but modifies the structure of the system.

Convolutional Layers and Max-Pooling

A single convolutional layer can be represented as a linear map:

$$y = Du,$$

where u is the input, and D is a matrix that represents the (linear) convolution operator. This operator exhibits a "constant-along-diagonals," Toeplitz-like structure, which encodes the convolution process efficiently.

Example of 2D Convolution. Consider a 3×3 input matrix U and a 2×2 kernel K. The convolution operation produces a 2×2 output matrix Y. Specifically:

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

The convolution can be represented as $y = Du$, where y and u are vectors obtained by stacking the rows of Y and U, respectively. The matrix D, which encodes the convolution operation, is given by:

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

Here, each row of D corresponds to a specific position of the sliding window in the convolution operation, and the Toeplitz-like structure ensures that the weights from K are consistently applied across different regions of U.

Max-Pooling. In practice, a convolutional layer is often combined with a max-pooling operation. Max-pooling downsamples the input by extracting the largest value from specific sub-areas of the original image, reducing the spatial dimensions while retaining the most prominent features.

The max-pooling operation can be represented as:

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

where:

- p is the number of elements in the input vector u,
- h is the size of the pooling window,
- q is the number of pooling regions ($p = qh$),
- $B_j \in \mathbb{R}^{h \times p}$ selects the sub-area corresponding to the j-th pooling window.

Here, each B_j extracts a specific sub-area of the input, and the max operation selects the largest value from that sub-area to form the output y_j.

In the example of Figure 2.3, the number of pixels selected in each area is $h = 4$, the output dimension is $q = 4$, and the input dimension is $p = qh = 16$. Vectorizing images row

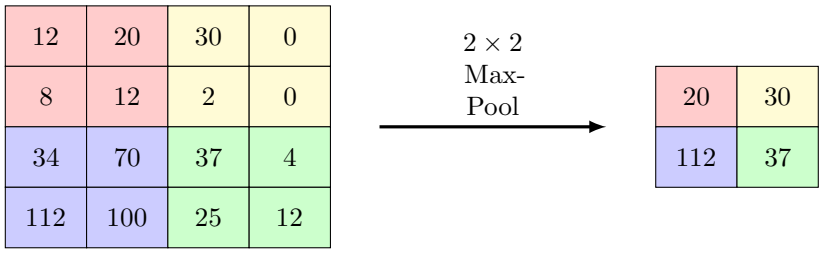

Figure 2.3: A max-pooling operation: the smaller image contains the maximal pixel values of each colored area. cite source

by row, we have:

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

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 into 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 to ensure that the input and output dimensions of ϕ are the same.) Using this mapping, we obtain the implicit model:

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

Here, C is used to select the top q elements at the output of ϕ:

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

The Lipschitz constant of the max-pooling activation map ϕ, with respect to the l_∞-norm, is 1.

Convolutional layers and max-pooling operations can be seamlessly incorporated into the implicit model framework. The Toeplitz-like structure of the convolution operator D and the pooling matrices B_j ensure efficient computation while enabling a mathematically rigorous representation of these common operations in convolutional neural networks (CNNs).

Residual Networks

Residual networks (ResNets) are constructed using a building block as illustrated in Figure 2.4.

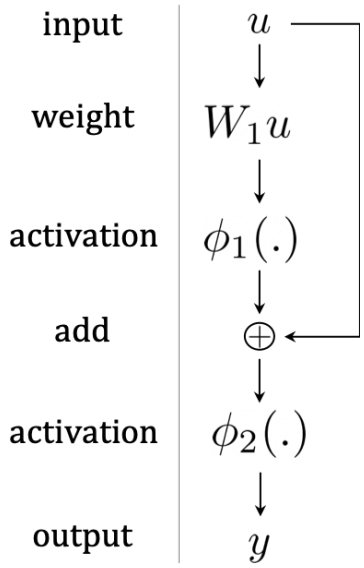

Figure 2.4: Building block of residual networks.

Mathematically, a residual block combines two linear transformations with non-linearities applied in the middle and at the end, while adding the input to the output of these transformations:

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

This formulation is a special case of the implicit model in (2.1). Defining the blockwise activation map:

$$\phi(z_1, z_2) = (\phi_1(z_1), \phi_2(z_2)),$$

the residual block can be expressed in matrix form as:

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

Figure 2.5 shows the model matrix A for a 20-layer residual network. Convolutional layers in the network correspond to matrix blocks with Toeplitz-like (constant along diagonals) structure. The residual connections are represented by the straight lines on top of these blocks. Throughout the network, the ReLU activation function is used, except in the final layer.

2.6 Conclusion

In this chapter, we explored the foundational aspects of implicit models, which define outputs through equilibrium equations rather than explicit sequential computations. This approach

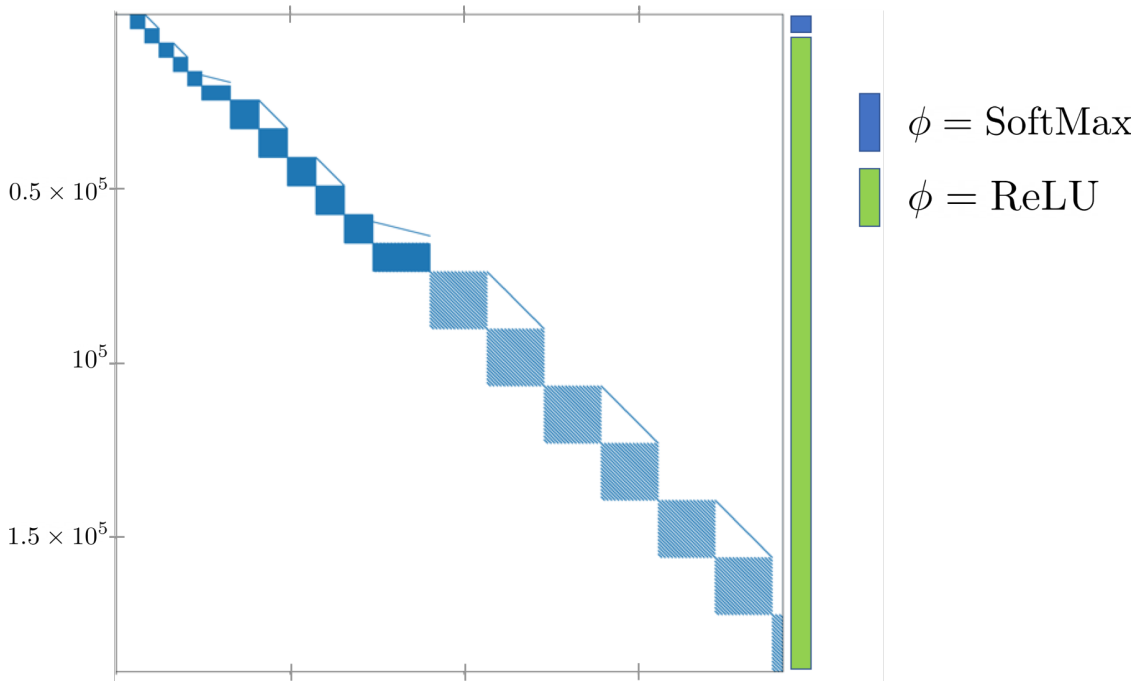

Figure 2.5: The model matrix A for a 20-layer residual network.

provides greater flexibility and efficiency in deep learning architectures. We examined the conditions ensuring well-posedness, guaranteeing the existence and uniqueness of solutions to these equilibrium equations, and discussed how implicit models can be composed to construct complex systems from simpler components.

Building upon this theoretical framework, Chapter 3 focuses on the practical application of implicit models, specifically constrained implicit learning for neural network sparsification. Sparsification seeks to reduce the number of parameters in a neural network without significantly compromising performance, thereby improving computational efficiency and interpretability. By leveraging the principles of implicit modeling, we reformulate sparsification as a constrained optimization problem, introducing a novel approach to parameter reduction in neural networks. This methodology streamlines the network while preserving, and in some cases enhancing, its robustness and generalization capabilities.

Chapter 3

Constrained Implicit Learning

3.1 Introduction

Implicit neural networks [8, 16, 41] offer a significant advantage over traditional neural networks: their model parameters are represented as simple data matrices. This structural simplicity allows for efficient model sparsification post-training by employing least-squares-based feature selection methods. In contrast, traditional neural networks typically necessitate retraining from scratch with varying regularization parameters to achieve sparsification.

The architecture of implicit neural networks, as proposed by [23] and [7], has garnered increasing attention due to its simplicity and versatility in integrating various traditional neural network architectures. Consequently, theories and applications initially developed for conventional neural networks are progressively being extended to implicit networks [72, 27, 30, 41].

Deep neural networks are often characterized by substantial redundancy, with a small subset of network coefficients retaining the majority of inference power [5, 20]. This observation has spurred research dedicated to sparsifying neural network coefficients without compromising inference efficiency. Common methods for network sparsification include:

1. Incorporating sparsity-inducing regularization terms into the training objective [63, 83].
2. Modifying the neural network architecture [96].
3. Applying post-training sparsification procedures that balance accuracy and sparsity [88, 63, 28, 5].

While existing sparsification methods yield satisfactory results on established networks, they present several limitations. Firstly, these techniques are often architecture-dependent, necessitating distinct sparsification strategies for different network architectures. Additionally, they must navigate the complex interplay between the training objective and sparsity,

often resulting in challenging optimization problems. Consequently, devising a universal sparsification scheme for traditional neural networks remains a formidable challenge. In light of this, our paper explores the emerging branch of implicit deep learning networks. As elaborated in the following section, sparsifying an implicit neural network can be conceptualized as a straightforward least-squares problem with sparsity penalties or constraints, irrespective of the underlying network architecture.

3.2 Constrained Implicit Learning Framework

Given a dataset with input $U \in \mathbb{R}^{p\times m}$ and output $Y \in \mathbb{R}^{q\times m}$, where each column represents an input or output vector, an implicit model is defined by the equilibrium equation and the prediction equation:

$$X = \phi(AX + BU) \tag{E}$$

$$\hat{Y}(U) = CX + DU \tag{P}$$

Here:

- $\phi : \mathbb{R}^{n\times m} \to \mathbb{R}^{n\times m}$ is a strictly increasing nonlinear activation function, such as ReLU, tanh, or sigmoid.
- $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{q\times n}$, and $D \in \mathbb{R}^{q\times p}$ are model parameters.

In equation (E), the input feature matrix U undergoes a linear transformation via B, and the internal state matrix X is obtained as the fixed-point solution. The output prediction $\hat{Y}$ is then derived using the prediction equation (P). This structure is illustrated in Figure 3.1, where the "pre-activation" state matrix Z and "post-activation" state matrix X are depicted; each column corresponds to a single data point.

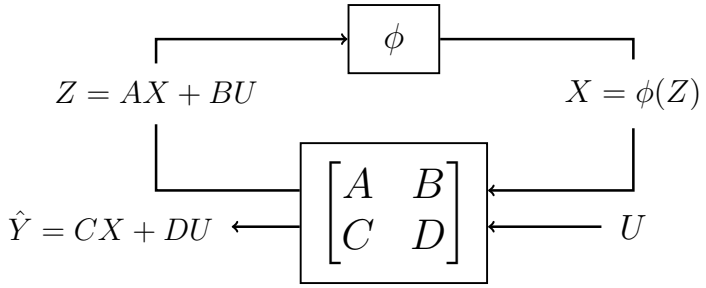

Figure 3.1: A diagram view of an implicit model, where Z is the pre-activation state "before" passing through the activation function ϕ and X is the post-activation state "after" passing through ϕ.

We provide a simple example of constructing X and Z from a 3-layer fully connected network of the form:

$$\hat{y}(u) = W_2 x_2, \quad x_2 = \phi(W_1 x_1), \quad x_1 = \phi(W_0 x_0), \quad x_0 = u,$$

where u is a single vector input. For notational simplicity, we exclude bias terms, which can be easily accounted for by considering the vector $(u, 1)$ instead of u. Each column of Z and X corresponds to the state from a single input. The column z is formed by stacking all the intermediate layers before applying ϕ, and the column x is formed by stacking all the intermediate layers after applying ϕ:

$$z = \begin{pmatrix} W_1 x_1 \\ W_0 x_0 \end{pmatrix}, \quad x = \phi(z) = \begin{pmatrix} x_2 \\ x_1 \end{pmatrix}.$$

In this example, we can verify that its equivalent implicit form is:

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{cc|c} 0 & W_1 & 0 \\ 0 & 0 & W_0 \\ \hline W_2 & 0 & 0 \end{array}\right).$$

For more complex networks, determining an equivalent implicit form can be a non-trivial task.

Constrained Implicit Model

The constrained implicit learning framework trains an implicit model with a key constraint: it must match both the state X and output $\hat{Y}$ of a given "baseline" (implicit or layered) model when the same inputs U are applied. This framework provides flexibility to incorporate any baseline deep neural network without directly addressing its architecture. Instead, we extract the pre-activation and post-activation state matrices.

For a given baseline model:

- If the baseline is implicit, the state matrix X can be obtained through fixed-point iterations.
- If the baseline is a standard layered network, X can be obtained via a simple forward pass.

In both cases, we extract the pre-activation state matrix Z, ensuring $X = \phi(Z)$, where ϕ is the activation function. Each column of Z and X corresponds to a single data point. For layered networks, these matrices are constructed by stacking all intermediate layers into a single column vector, with the first intermediate layer at the bottom and the last layer at the top.

The implicit models are *characterized* by the conditions:

$$Z = AX + BU, \quad \hat{Y} = CX + DU,$$

which ensure that the implicit model matches both the state and outputs of the baseline model. To find another well-posed model that satisfies these conditions, we solve the following *convex problem*:

$$\begin{aligned} \min_{A,B,C,D} \quad & \ell(A, B, C, D) && \text{(3.2a)} \\ \text{subject to} \quad & Z = AX + BU, && \text{(3.2b)} \\ & \hat{Y} = CX + DU, && \text{(3.2c)} \\ & \|A\|_\infty \leq \kappa, && \text{(3.2d)} \end{aligned}$$

where:

- ℓ is a user-defined loss function,
- $\kappa \leq 1$ is a hyper-parameter ensuring well-posedness (defined in Definition 3).

Equality Constraints Relaxation

The equality constraints (3.2b) and (3.2c), which enforce an exact match for the internal state and output, can be relaxed to allow for approximate matching. This relaxation introduces flexibility into the model by incorporating penalty terms into the objective function. The relaxed optimization problem becomes:

$$\min_{(A,B,C,D)\in\mathcal{C}, \|A\|_\infty \leq \kappa} \ell(A, B, C, D) + \lambda_1 \|Z - (AX + BU)\|_F^2 + \lambda_2 \|\hat{Y} - (CX + DU)\|_F^2, \quad \text{(3.3)}$$

where:

- ℓ: A user-defined loss function.
- $\mathcal{C}$: A user-defined constraint set for the model parameters.
- λ_1 and λ_2: Hyper-parameters that control the degree of matching for the state and output, respectively.
- $\|\cdot\|_F$: Frobenius norm, ensuring quadratic penalties for deviations from the constraints.

This relaxation allows for greater flexibility in optimization while still ensuring that the learned model parameters align closely with the internal states and outputs of the baseline model. By adjusting λ_1 and λ_2, the user can control the trade-off between exact matching and other objectives in the training process.

The training problem (3.3) can be decomposed into a series of parallel, smaller problems, each involving a single row, or a block of rows, if ℓ is decomposable. This decomposition is feasible for most objective functions commonly used in neural network training.

For a single row $(a^\top, b^\top)$ of (A, B), and with $z^\top$ being the corresponding row in Z, the problem reduces to:

$$\begin{aligned} \min_{a,b} \quad & \ell(a,b) + \lambda_1 \left\| z - \left(X^\top, U^\top\right) \begin{pmatrix} a \\ b \end{pmatrix} \right\|^2 \\ \text{subject to} \quad & \|a\|_1 \leq \kappa, \end{aligned} \tag{3.4}$$

where $\|a\|_1 \leq \kappa$ represents the well-posedness condition, as $\|A\|_\infty$ is separable in terms of rows.

The optimization problem for C, D is independent of that for A, B and has a similar form to problem (3.4), but without the well-posedness constraint:

$$\min_{c,d} \quad \ell(c,d) + \lambda_2 \left\| \hat{y} - \left(X^\top, U^\top\right) \begin{pmatrix} c \\ d \end{pmatrix} \right\|^2. \tag{3.5}$$

When $\ell = 0$, problem (3.4) reduces to the basis pursuit problem introduced in [18], for which efficient optimization algorithms have been developed over the years [91, 73]. These algorithms enable effective solutions for sparsity-driven tasks in various settings.

In the next section, we introduce tailored algorithms designed specifically for implicit model sparsification to solve problem (3.4) effectively.

3.3 Algorithm for Constrained Implicit Model Sparsification

In this section, we discuss algorithms for model sparsification in implicit models. The optimization problem (3.4) is a least-squares problem, which can be efficiently solved using stochastic gradient descent (SGD) or its numerous variants [29]. These methods are computationally effective and well-suited for modern machine learning workflows.

However, the optimization problem (3.4) presents a greater challenge due to the ℓ_1 constraint. This problem is equivalent to the LASSO in its constrained form [90]. While projected gradient descent and its variants can be applied directly to (3.4), each iteration requires a projection onto the ℓ_1-norm ball [58]. This operation becomes computationally expensive and less practical as the dimension of (a, b) grows.

Furthermore, state-of-the-art preprocessing techniques, such as safe screening rules and active set methods [98, 31], cannot be directly applied to the constrained formulation of (3.4).

This limitation highlights the need for tailored algorithms designed specifically to address the challenges posed by the sparsification of implicit models under these constraints.

In light of the challenges associated with directly solving problem (3.4), we propose an alternative algorithm that transforms it into a sequence of least-squares problems with ℓ_1 penalties. This approach leverages state-of-the-art LASSO techniques, facilitating more efficient optimization.

Our algorithm employs a bisection method along the LASSO regularization path to identify an approximate regularization parameter that satisfies the original constraint. By iteratively adjusting the regularization parameter and solving the corresponding LASSO problem, we converge to a solution that approximates the desired sparsity level while adhering to the constraints of the original problem. This method capitalizes on the efficiency of existing LASSO solvers and the structured exploration of the regularization path, offering a practical solution to the computational difficulties inherent in high-dimensional settings.

Algorithm Design

For brevity of exposition, we overload notation and define:

$$M := (X^\top, U^\top), \quad a := (a, b).$$

Using this notation, we formulate the following constrained LASSO problem:

$$P(\kappa) := \min_a \ \frac{1}{2}\|Ma - z\|^2 \quad \text{subject to} \quad \|a\|_1 \leq \kappa,$$

where $\|a\|_1 \leq \kappa$ simultaneously enforces sparsity and the well-posedness constraint. We assume that the constraint is not trivially satisfied:

A1: Non-trivial solution: $0 \leq \kappa < \|(M^\top M)^{-1} M^\top z\|_1$.

To simplify the formulation, we define:

$$h(a) := \frac{1}{2}\|Ma - z\|^2,$$

and use $h(a)$ and $\frac{1}{2}\|Ma - z\|^2$ interchangeably.

Before proceeding further, we introduce the unconstrained LASSO problem, which will serve as a critical subroutine in our analysis:

$$Q(\lambda) := \min_a \ h(a) + \lambda\|a\|_1 \quad \text{(Unconstrained LASSO)}.$$

$P(\kappa)$ and Root Finding

Our method is based on the following function:

$$g(\lambda) := \|a^*_{Q(\lambda)}\|_1, \tag{3.6}$$

$$a^*_{Q(\lambda)} = \arg\min_a \{h(a) + \lambda\|a\|_1\}, \tag{3.7}$$

where we apply a bisection method to find λ such that $g(\lambda) = \kappa$. Before establishing our algorithm, we provide the intuition behind our approach by analyzing the fundamental properties of $g(\lambda)$.

Lemma 1. *The function $g(\lambda)$ satisfies the following properties:*

- *$g(\lambda)$ is continuous and piecewise linear [68].*
- *$g(\lambda_1) - g(\lambda_2) \leq 0$ if $\lambda_1 > \lambda_2$. That is, $g(\lambda)$ is monotonically decreasing.*
- *$g(0) = \|(M^\top M)^{-1}M^\top z\|_1 \leq \frac{1}{m}\|M^\top z\|_1$.*
- *$g(\lambda) = 0$ if $\lambda \geq \|M^\top z\|_\infty$.*

Proof. Proof of Lemma 1

We start by proving the monotonicity of $g(\lambda)$. Assume $\lambda_1 > \lambda_2$. By the optimality condition, we have:

$$\frac{1}{2}\|Ma^*_{Q(\lambda_1)} - z\|^2 + \lambda_1\|a^*_{Q(\lambda_1)}\|_1 \leq \frac{1}{2}\|Ma^*_{Q(\lambda_2)} - z\|^2 + \lambda_1\|a^*_{Q(\lambda_2)}\|_1,$$
$$\frac{1}{2}\|Ma^*_{Q(\lambda_2)} - z\|^2 + \lambda_2\|a^*_{Q(\lambda_2)}\|_1 \leq \frac{1}{2}\|Ma^*_{Q(\lambda_1)} - z\|^2 + \lambda_2\|a^*_{Q(\lambda_1)}\|_1.$$

Summing these two inequalities and rearranging terms yields:

$$(\lambda_1 - \lambda_2)(g(\lambda_1) - g(\lambda_2)) = (\lambda_1 - \lambda_2)(\|a^*_{Q(\lambda_1)}\|_1 - \|a^*_{Q(\lambda_2)}\|_1) \leq 0.$$

Dividing both sides by $\lambda_1 - \lambda_2 > 0$ confirms that $g(\lambda)$ is monotonically decreasing.

For the second condition, note that $a^*_{Q(0)} = (M^\top M)^{-1}M^\top z$, and thus:

$$\|a^*_{Q(0)}\|_1 = \|(M^\top M)^{-1}M^\top z\|_1 \leq \|(M^\top M)^{-1}\| \cdot \|M^\top z\| \leq \frac{1}{m}\|M^\top z\|.$$

Finally, consider the case where $\lambda \geq \|M^\top z\|_\infty$. In this scenario, we have:

$$\begin{aligned}
\frac{1}{2}\|z\|^2 &= \frac{1}{2}\|Ma - z - Ma\|^2 \\
&= \frac{1}{2}\|Ma - z\|^2 - \langle Ma, Ma - z\rangle + \frac{1}{2}\|Ma\|^2 \\
&= \frac{1}{2}\|Ma - z\|^2 + \langle Ma, z\rangle \\
&\leq \frac{1}{2}\|Ma - z\|^2 + \|M^\top z\|_\infty\|a\|_1.
\end{aligned}$$

When $\lambda \geq \|M^\top z\|_\infty$, the penalty dominates the optimization, forcing $g(\lambda) = 0$. This completes the proof. □

Lemma 2. *Given $0 < \kappa < \|(M^\top M)^{-1} M^\top z\|_1$, there exists a $\lambda_\kappa > 0$ such that $a^*_{P(\kappa)} = a^*_{Q(\lambda_\kappa)}$.*

Proof. Proof of Lemma 2

By Lemma 1, we know there exists λ_κ such that $g(\lambda_\kappa) = \kappa \in (0, \|(M^\top M)^{-1} M^\top z\|_1)$. Next, we verify that for any $a^*_{Q(\lambda_\kappa)} = \arg\min \left\{ \frac{1}{2}\|Ma - z\|^2 + \lambda_\kappa \|a\|_1 \right\}$,

$$a^*_{Q(\lambda_\kappa)} = \arg \min_{\|a\|_1 \leq \kappa} \left\{ \frac{1}{2}\|Ma - z\|^2 \right\}.$$

To do so, recall the optimality condition of $Q(\lambda_\kappa)$, which states:

$$0 \in \partial_{a=a^*_{Q(\lambda_\kappa)}} \left\{ \tfrac{1}{2}\|Ma - z\|^2 + \lambda_\kappa \|a\|_1 \right\} = M^\top (M a^*_{Q(\lambda_\kappa)} - z) + \lambda_\kappa \partial(\|a^*_{Q(\lambda_\kappa)}\|_1).$$

Now, we plug $a^*_{Q(\lambda_\kappa)}$ and λ_κ into the optimality conditions of $P(\kappa)$ and verify:

$$\begin{aligned} \|a^*_{Q(\lambda_\kappa)}\|_1 &\leq \kappa, \\ \lambda_\kappa &\geq 0, \\ \lambda_\kappa(\kappa - \|a^*_{Q(\lambda_\kappa)}\|_1) &= 0, \\ \partial_{a=a^*_{Q(\lambda_\kappa)}} \left\{ \frac{1}{2}\|Ma - z\|^2 + \lambda_\kappa(\|a\|_1 - \kappa) \right\} &\ni 0. \end{aligned}$$

These conditions are satisfied, which completes the proof. □

Lemma 1 establishes that by identifying the correct λ, the solutions to $Q(\lambda)$ and $P(\kappa)$ are equivalent. Furthermore, Lemma 2 demonstrates that $g(\lambda)$ is monotonic and that finding λ_κ such that $g(\lambda_\kappa) = \kappa$ suffices to recover $a^*_{P(\kappa)}$.

This equivalence naturally suggests employing a bisection method to identify λ_κ. By iteratively narrowing the search interval for λ_κ, we can efficiently locate the value along the LASSO regularization path that satisfies the sparsity constraint.

Algorithm (1) summarizes the bisection algorithm to search for the matching λ_κ over the LASSO regularization path.

Analysis of the Bisection Method

Having established the intuition behind the bisection method, we now analyze its computational efficiency and convergence properties. Recall that $\lambda_\kappa \in (0, \|M^\top z\|_\infty)$, as established earlier. The bisection method terminates after at most

$$T = \mathcal{O}(\log(1/\varepsilon))$$

Algorithm 1 Bisection algorithm on LASSO regularization path

input $\kappa, M, z, \varepsilon$
initialize $u = \|M^\top z\|, l = 0$
while $u - l \geq \varepsilon$
  set $\lambda = \frac{1}{2}(u + l)$
  solve $a^*_{Q(\lambda)} = \arg\min_a \{h(a) + \lambda\|a\|_1\}$
  if $\|a^*_{Q(\lambda)}\|_1 \geq \kappa$
    $l \leftarrow \lambda$
  else
    $u \leftarrow \lambda$
  end
output $\lambda, a^*_{Q(\lambda)}$

iterations, where ε is the desired precision. This logarithmic dependence on ε ensures a fast convergence rate for identifying λ_κ.

In addition to the outer bisection iterations, the proximal gradient method is employed to solve the subproblem $Q(\lambda)$ at each step. The complexity of these internal iterations can be analyzed using the following standard result:

Lemma 3 ([53])**.** *If the proximal gradient method is applied to $Q(\lambda)$, then:*

$$h(a^k) - h(a^*) \leq \exp\left(-\frac{\mu}{L}K\right)\{h(a^0) - h(a^*)\},$$

where:

- *L is the largest eigenvalue of $M^\top M$,*
- *μ is the Polyak-Lojasiewicz constant of the problem,*
- *a^k is the output of the k-th proximal gradient iteration,*
- *a^* is the unique optimal solution to $Q(\lambda)$.*

This result guarantees an exponential convergence rate for the proximal gradient method under the Polyak-Lojasiewicz condition. Combining this with the logarithmic convergence of the bisection method yields an efficient overall procedure for solving $P(\kappa)$.

Intuitively, the bisection method achieves linear convergence in both the outer loop (bisection iterations) and the inner loop (proximal gradient iterations). This results in a worst-case iteration complexity of:

$$K = \mathcal{O}(T \cdot \log(1/\varepsilon)) = \mathcal{O}(\log^2(1/\varepsilon)),$$

where T is the number of bisection iterations and ε is the desired precision for each step.

In the following subsection, we demonstrate how the convergence rate can be further improved by employing warm-starting techniques for solving consecutive bisection subproblems. By initializing each new subproblem using the solution from the previous iteration, we effectively reduce the number of required proximal gradient iterations, leading to a faster overall convergence.

Effect of Warm-Starting

Lemma 4 (Effect of Warm-Starting)**.** *Given λ_1, λ_2, the following holds:*

$$h(a^*_{Q(\lambda_1)}) + \lambda_1 \|a^*_{Q(\lambda_1)}\|_1 - \left[h(a^*_{Q(\lambda_2)}) + \lambda_2 \|a^*_{Q(\lambda_2)}\|_1\right] \leq \sqrt{n} \cdot |\lambda_1 - \lambda_2| \cdot A,$$

where $A = \sup_\lambda \|a^*_{Q(\lambda)}\|_\infty$.

Proof. The proof leverages the Lipschitz continuity of the objective function with respect to λ and the boundedness of the solution $a^*_{Q(\lambda)}$. For any pair λ_1, λ_2, the difference in the objectives can be bounded as follows:

$$\left|\left(h(a^*_{Q(\lambda_1)}) + \lambda_1 \|a^*_{Q(\lambda_1)}\|_1\right) - \left(h(a^*_{Q(\lambda_2)}) + \lambda_2 \|a^*_{Q(\lambda_2)}\|_1\right)\right|$$

is proportional to $|\lambda_1 - \lambda_2|$ scaled by the maximum ℓ_∞-norm of the solutions across λ, which is $A = \sup_\lambda \|a^*_{Q(\lambda)}\|_\infty$. Without loss of generality, assume that $\lambda_1 > \lambda_2$. Then:

$$\begin{aligned} h(a^*_{Q(\lambda_1)}) + \lambda_1 \|a^*_{Q(\lambda_1)}\|_1 &\leq h(a^*_{Q(\lambda_2)}) + \lambda_1 \|a^*_{Q(\lambda_2)}\|_1 \\ &= h(a^*_{Q(\lambda_2)}) + \lambda_2 \|a^*_{Q(\lambda_2)}\|_1 \\ &\quad + (\lambda_1 - \lambda_2)\|a^*_{Q(\lambda_2)}\|_1 \\ &\leq h(a^*_{Q(\lambda_2)}) + \lambda_2 \|a^*_{Q(\lambda_2)}\|_1 \\ &\quad + \sqrt{n}|\lambda_1 - \lambda_2|A, \end{aligned} \tag{3.8}$$

where $A = \sup_\lambda \|a^*_{Q(\lambda)}\|_\infty$. The factor $\sqrt{n}$ accounts for the dimensionality of the solution space. Exchanging the position of λ_1, λ_2, this completes the proof. □

Implications of Lemma 4. Given Lemma 4, we are guaranteed that if λ_1, λ_2 are sufficiently close, warm-starting $Q(\lambda_1)$ with the optimal solution to $Q(\lambda_2)$ results in a small initial optimality gap, bounded by $\sqrt{n}|\lambda_1 - \lambda_2|A$. This bound ensures that as λ_1 approaches λ_κ, the computational cost of solving each subproblem decreases significantly.

This result highlights the benefit of warm-starting: when the change in λ between consecutive iterations is small, the change in the corresponding optimal solution is also bounded. This bounded change enables efficient reuse of the solution from the previous iteration as the initialization for the next, thereby accelerating convergence across the bisection path.

Theorem 4. *If each $Q(\lambda)$ is warm-started using the optimal solution from the previous iteration, the overall worst-case iteration complexity becomes:*

$$K = \mathcal{O}(\log^2(1/\varepsilon)).$$

Proof. Proof of Theorem 4

Let $I(\gamma, \varepsilon)$ denote the number of iterations required for the proximal gradient method to converge, given an initial optimality gap γ and tolerance ε. From Lemma 2, we know:

$$I(\gamma, \varepsilon) = \mu^{-1} L \log(\gamma/\varepsilon),$$

where μ is the Polyak-Lojasiewicz constant, and L is the largest eigenvalue of $M^\top M$.

Define $\Delta = \|(M^\top M)^{-1} M^\top z\|_1 \cdot A$, where A is the maximum ℓ_∞-norm of the solutions along the regularization path. For the t-th iteration, the initial optimality gap γ_t satisfies:

$$\gamma_t \leq 2^{-t}\sqrt{n}\Delta + \varepsilon_{t-1},$$

where ε_{t-1} is the tolerance used in the $(t-1)$-th iteration.

Thus, the number of iterations for the proximal gradient method at step t is:

$$I(\gamma_t, \varepsilon_t) \leq \mu^{-1} L \log\left(\frac{2^{-t}\sqrt{n}\Delta + \varepsilon_{t-1}}{\varepsilon^\top}\right),$$

where $\varepsilon^\top \equiv \varepsilon$, the desired global tolerance, and $T = \log_2(1/\varepsilon)$ is the total number of bisection steps.

Summing over all iterations, the total number of proximal gradient iterations is:

$$\begin{aligned} K &= \sum_{t=1}^{T} I(\gamma_t, \varepsilon_t) \\ &\leq \mu^{-1} L \sum_{t=1}^{T} \log\left(\frac{\sqrt{n}\Delta}{\varepsilon} 2^{-t} + 1\right) \\ &\lesssim \mu^{-1} L \sum_{t=1}^{T} \log\left(\frac{\sqrt{n}\Delta}{2^{t-T}} + 1\right) \\ &= \mathcal{O}(\log^2(1/\varepsilon)). \end{aligned}$$

Thus, the worst-case iteration complexity of the warm-started bisection algorithm is $\mathcal{O}(\log^2(1/\varepsilon))$.

□

Theorem 4 establishes that employing a warm-start strategy for initializing each $Q(\lambda)$ ensures that the computational complexity K grows at a rate no faster than the square of the logarithm of the inverse of the desired precision ε. This result highlights the efficiency of the warm-start approach, as it significantly reduces the computational burden by effectively narrowing the initial optimality gap in successive iterations. Consequently, warm-starting leads to notable performance enhancements across all solver iterations.

The complete algorithm, integrating the bisection method with warm-starting, is presented in Algorithm 2.

Algorithm 2 Constrained Implicit Model Sparsification

input Data matrix U; A trained standard (layered) neural network $\mathcal{N} : \mathbb{R} \times \mathbb{R} \to \mathbb{R} \times \mathbb{R}$; Hyper-parameter $\kappa \leq 1$ and ε.
initialize Run a single forward pass on $\mathcal{N}$ with U to obtain outputs $\hat{Y}$, *i.e.,* $\hat{Y} = \mathcal{N}(U)$. Collect all intermediate layers before and after passing through the activation ϕ. Construct Z and X by stacking all intermediate layers.
set $M := (X^T\ U^T), W := (A, B) = \mathbf{0}$
for each row z_i of Z
  warm start $Q(\lambda_i)$ with the optimal solution to $Q(\lambda_{i-1})$
  solve $Q(\lambda_i)$ with Algorithm 1
  update row i of $W \leftarrow a^*_{Q(\lambda_i)}$
  end
output W

3.4 Numerical Experiments

To evaluate the effectiveness of the proposed implicit model sparsification framework, we conduct experiments on two benchmark datasets: CIFAR-100 [56] and 20NewsGroup[1]. The experiments leverage ResNet-20 [21] and DistilBERT[2] [81] as baseline architectures. The baseline test accuracy for these models is 92.1% on CIFAR-100 and 92.8% on 20NewsGroup.

In our experiments, we fix the tolerance parameter at $\varepsilon = 0.01$. Optimization problems are solved using the MOSEK [4] optimization solver, ensuring robust and efficient convergence. To measure the efficiency of our sparsification algorithm, we report the total number of iterations required to traverse all rows of the weight matrix.

The following sections present detailed results on the accuracy, sparsity, and computational efficiency achieved using the proposed sparsification method, demonstrating its capability to maintain performance while significantly reducing model complexity.

[1] http://qwone.com/~jason/20Newsgroups/
[2] https://huggingface.co/docs/transformers/model_doc/distilbert

Table 3.1: Sparsity levels and accuracy on the CIFAR-100 and 20NewsGroup data.

	CIFAR-100		20NewsGroup	
κ	**Accuracy (%)**	**Sparsity (%)**	**Accuracy (%)**	**Sparsity (%)**
0.005	76.7	98.1	60.9	97.8
0.01	79.1	96.4	74.2	95.9
0.05	84.2	95.2	80.2	95.3
0.1	87.1	93.8	87.2	93.3
0.5	89.0	93.3	88.8	90.7
0.99	91.0	90.1	90.6	87.4
Dense	92.1	0	92.8	0

Table 3.1 illustrates the trade-off between sparsity and test performance for the CIFAR-100 and 20NewsGroup datasets, employing warm-starting as outlined in Algorithm 2. The hyper-parameter κ serves as a direct control for the sparsity level of the model, where higher κ values correspond to lower sparsity levels.

Our results demonstrate the algorithm's capability to compute highly sparse networks, achieving significant reductions in the number of parameters with only a 2% reduction in test accuracy on both datasets compared to the original dense networks. As the number of non-zero elements increases (i.e., sparsity decreases), a gradual improvement in test accuracy is observed, illustrating the inherent trade-off between model sparsity and performance.

Even with approximately 10% of the model weights retained, the models maintain competitive performance levels, exhibiting only a marginal decrease in accuracy. This resilience highlights the robustness of the proposed method, particularly in scenarios requiring aggressive pruning while preserving predictive performance. Such results validate the efficacy of our sparsification approach in balancing model complexity and task performance.

In our evaluation, we compare the proposed method with several state-of-the-art optimization-based parameter reduction techniques: SSS [51], SPR [11], and MLA [49]. SSS and SPR formulate parameter reduction as a sparse regularized optimization problem. SSS employs an ℓ_1-relaxation approach, while SPR utilizes perspective relaxation to enhance sparsity and stability. MLA, on the other hand, focuses on aligning semantic information between intermediate outputs and overall model performance by incorporating feature and semantic correlation losses alongside a classification loss. This approach is conceptually similar to our state- and outputs-matching conditions.

For experiments on CIFAR-100, we follow the original papers' settings by using ResNet-20 for SSS and SPR, and ResNet-18 for MLA, ensuring a fair comparison. The hyper-parameters are kept consistent with those reported in the original papers. As shown in Table 3.2, our

Table 3.2: Comparison between sparsity levels and accuracy on the CIFAR-100 and 20NewsGroup data with exisiting benchmark.

	CIFAR-100		20NewsGroup	
Method	**Accuracy** (%)	**Sparsity** (%)	**Accuracy** (%)	**Sparsity** (%)
SSS	88.4	44.4	89.5	48.7
SPR	89.8	45.9	90.6	50.5
MLA	89.1	50.0	90.3	51.8
Ours	91.0	90.1	90.6	87.4
Dense	92.1	0	92.8	0

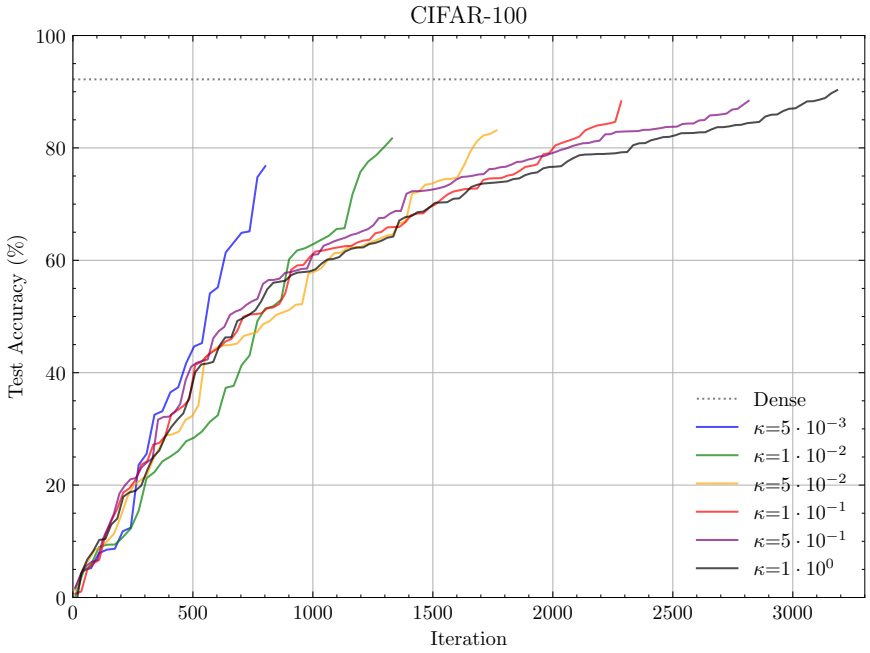

Figure 3.2: Performance of different κ at each iteration on CIFAR-100.

method consistently outperforms all baselines, achieving superior test performance while obtaining a significantly larger reduction in the number of parameters. This demonstrates the efficacy of the proposed sparsification framework, which not only achieves better trade-offs between sparsity and performance but also offers a scalable and robust solution for parameter reduction.

Figures 3.2 and 3.3 illustrate the performance of the algorithm for various values of κ, showcasing the corresponding test accuracy at each iteration. To enhance computational

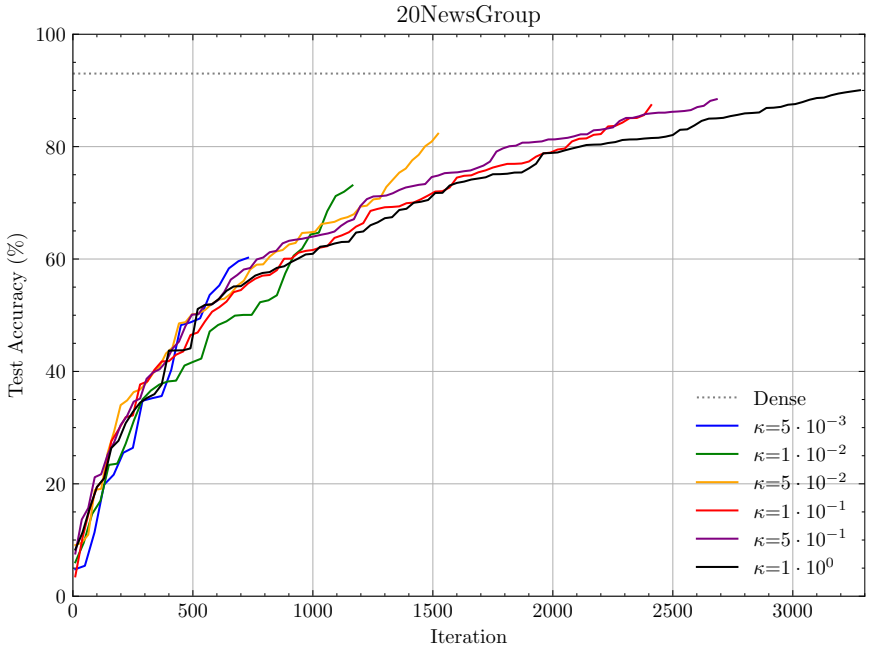

Figure 3.3: Performance of different κ at each iteration on 20NewsGroup.

efficiency, we employ warm-starting within the bisection inner loop. This involves initializing the solution with the optimal parameters obtained from the preceding iteration, thereby accelerating the exploration of the solution space.

A comparative analysis of the algorithm's performance with and without warm-starting is presented in Figures 3.4 and 3.5, for $\kappa = 1$, $\kappa = 0.1$, and $\kappa = 0.5$. The results demonstrate that warm-starting yields an average speedup of 1.8x in convergence time and significantly reduces computational overhead. These findings highlight the efficiency gains achieved by leveraging warm-starting, particularly in scenarios involving extensive parameter sparsification.

In solving the problem (3.4), the input matrix $U \in \mathbb{R}^{p\times m}$ does not need to encompass the entire training dataset. To explore the minimum sample size required for effectively training a sparse implicit model, we conducted experiments varying the number of samples. Figures 3.6 and 3.7 illustrate the performance of different κ values trained with subsets of the training data. As the percentage of total training samples increases, corresponding to higher m in the input matrix U, the dimension of M also increases. The results indicate that test performance improves initially with increasing training data, but eventually plateaus. Specifically, for CIFAR-100 (Figure 3.6), the performance stabilizes at approximately 20% of the total training data, while for 20NewsGroup (Figure 3.7), stabilization occurs around 30%.

For very small values of κ, the model may become excessively sparse, which hinders effective learning. Conversely, for sufficiently large κ, training can be efficiently conducted

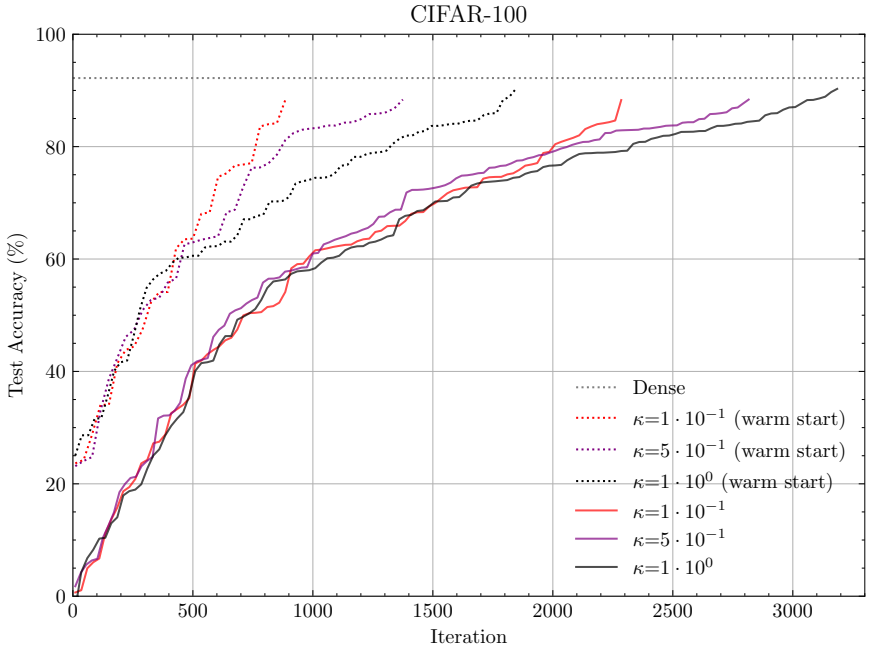

Figure 3.4: Performance of different κ at each iteration with warm starting on CIFAR-100.

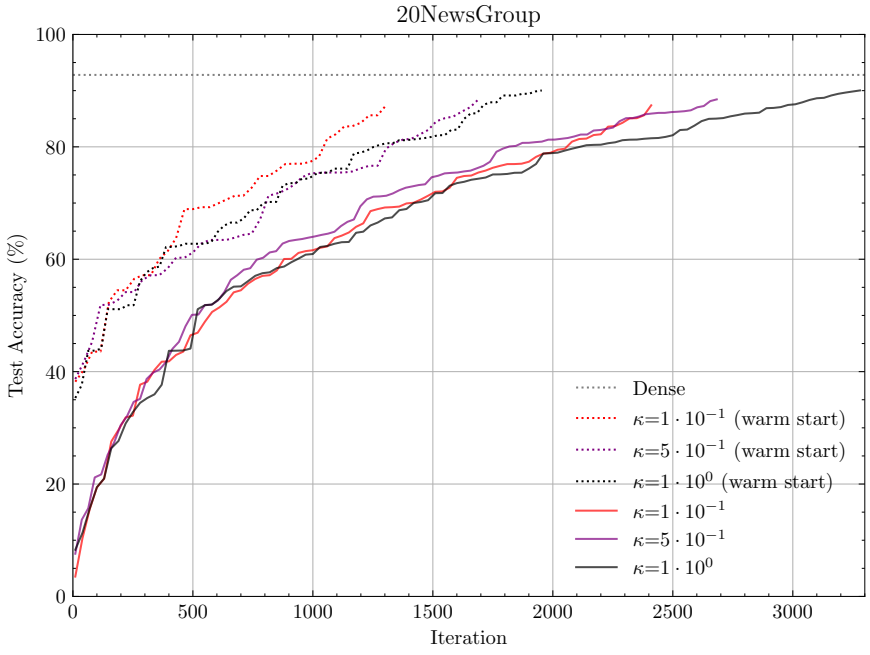

Figure 3.5: Performance of different κ at each iteration with warm starting on 20NewsGroup.

using only a fraction of the dataset. These findings highlight the state matrix X as a high-quality representation, capable of capturing significant underlying information, thereby enabling model training with substantially fewer samples.

Additionally, Table 3.3 summarizes the computational speed-ups achieved by leveraging partial datasets. The results underscore the practical implications of utilizing reduced datasets, demonstrating significant efficiency gains in terms of computational scalability and resource utilization.

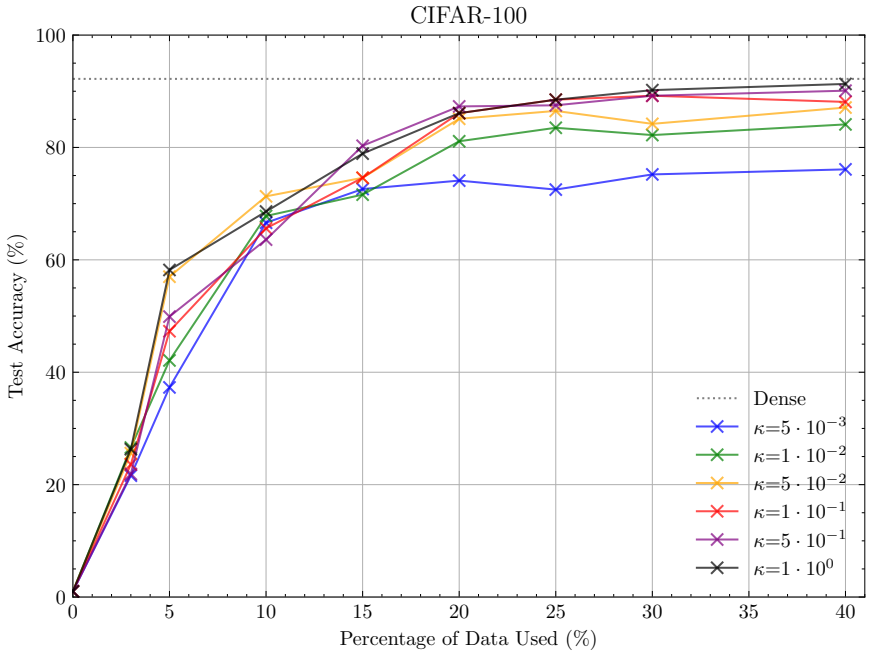

Figure 3.6: Performance of different κ trained with partial data on CIFAR-100.

Table 3.3: Training computational speed up by using warm-start and partial data matrix U.

	CIFAR-100		20NewsGroup	
κ	**Warm Start**	**20% Data**	**Warm Start**	**30% Data**
0.1	2.2x	9.1x	1.5x	8.7x
0.5	2.0x	7.8x	1.6x	8.4x
0.99	1.7x	7.5x	1.7x	7.4x

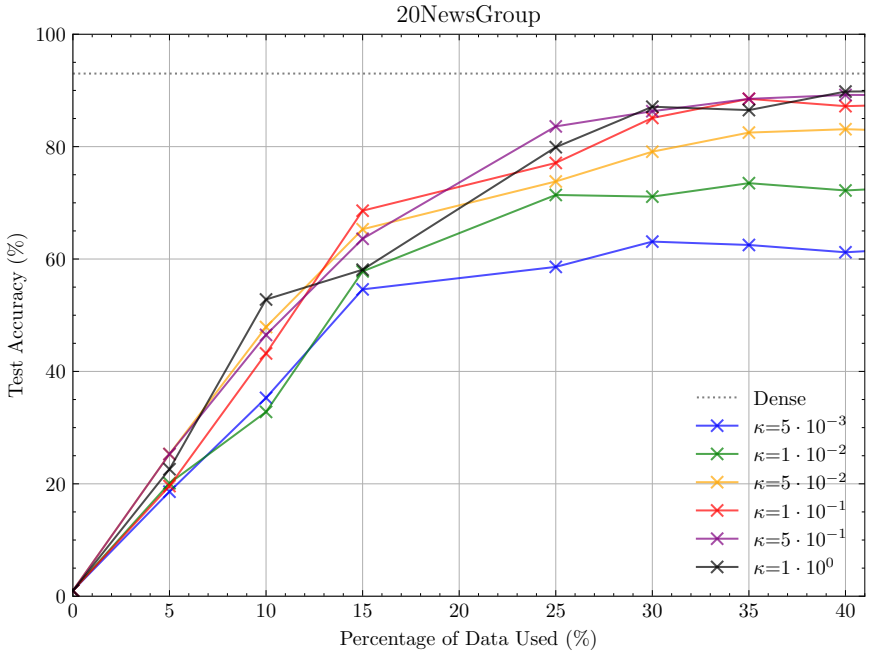

Figure 3.7: Performance of different κ trained with partial data on 20NewsGroup.

3.5 Conclusion

In conclusion, this work introduces a novel paradigm in neural network sparsification—*Implicit Model Sparsification*. Unlike conventional techniques that often depend on intricate network structures or specialized loss functions, our approach adopts a streamlined methodology. Implicit Model Sparsification is formulated as a straightforward least-squares problem, augmented with sparsity-inducing constraints or penalties. This simplicity significantly broadens the applicability of the method to a diverse range of neural network architectures.

To address computational challenges and enhance the scalability of our approach, we have developed a parallel algorithm. This algorithm effectively handles the complexities of transforming traditional neural networks into implicit models while maintaining computational efficiency and model performance. Our experimental results on the CIFAR-100 and 20NewsGroup datasets demonstrate the efficacy of the proposed method, particularly its robustness under high pruning rates. The models exhibit minimal loss in test accuracy even when trained with significantly reduced parameters, showcasing the resilience of our approach.

Additionally, our investigation into the optimal sample size required for training sparse implicit models provides valuable insights. The experiments reveal that a moderate subset of the training data suffices to achieve competitive performance, emphasizing the information-rich representation captured by the state matrix. This finding highlights the practicality of training with limited data while retaining robust performance.

In summary, *Implicit Model Sparsification* offers a versatile and innovative framework for neural network compression. With its simplicity, scalability, and demonstrated effectiveness, this paradigm holds significant promise for real-world applications where efficiency and resource optimization are paramount.

While the sparsification framework introduced in this chapter offers significant advances in model efficiency, neural networks deployed in real-world scenarios must also contend with challenges posed by input perturbations, adversarial attacks, and out-of-distribution data. To address these critical concerns, the next chapter discusses *Robustness Analysis via Implicit Representation*. By leveraging the unique structure of implicit models, we develop methods to quantify and enhance the resilience of neural networks against such adversities.

Chapter 4

Robustness Analysis via Implicit Representation

4.1 Introduction

Despite the remarkable success of deep neural networks (DNNs) across various domains, their vulnerability to adversarial perturbations remains a critical challenge. Since the seminal work by Szegedy et al., a plethora of research has revealed the susceptibility of state-of-the-art DNNs to adversarial samples—inputs that are imperceptibly modified to mislead the model [33, 57, 70]. Although adversarial samples only slightly deviate from the original data distribution, their impact on classification accuracy highlights the fragility of current DNN architectures.

This vulnerability has spurred significant interest in designing robust models capable of withstanding adversarial attacks [65, 69, 75, 35]. However, many defense mechanisms have proven ineffective against adaptive attacks [6, 12], underscoring fundamental gaps in our understanding of DNN robustness. These challenges have fueled efforts to develop robustness evaluations [13, 105], interpretability techniques, and visualization methods that aim to bridge these gaps [34, 86, 104]. Yet, critical questions surrounding the root causes of DNN vulnerabilities remain unanswered.

In this work, we introduce implicit models as a powerful framework for robustness analysis. Implicit models, defined by fixed-point equations rather than layer-wise mappings, provide a unifying perspective that encompasses many existing DNN architectures. Their scalability and well-posedness make them particularly suited for robustness evaluations. By leveraging the mathematical structure of implicit models, we establish a comprehensive framework that extends theoretical robustness guarantees to a wide range of DNN models. This approach not only strengthens our understanding of DNN behavior under adversarial conditions but also lays the foundation for designing inherently robust architectures.

4.2 Robustness bound

In this section, we analyze the robustness properties of implicit models defined by the prediction rule (2.1b). Specifically, we aim to derive bounds on the state, output, and loss function under the presence of input uncertainties. Such robustness analysis is valuable for multiple purposes, including diagnosing model vulnerabilities, generating adversarial attacks, and guiding the design of penalties or constraints during training. To ensure a well-posed framework, we assume that the activation map satisfies the Blockwise Lipschitz (BLIP) condition, and the matrix A of the implicit model adheres to the sufficient well-posedness conditions established in Theorem (3).

Input Uncertainty Models

We consider scenarios where the input vector is uncertain and is only known to belong to a given set $\mathcal{U} \subseteq \mathbb{R}^p$. Our results are applicable to a broad class of input uncertainty sets; however, we focus on the following two representative cases:

The first case corresponds to a *box-bounded uncertainty set*, where the input vector is constrained to lie within a specified range around a nominal value:

$$\mathcal{U}^{\text{box}} := \left\{ u \in \mathbb{R}^p \ : \ |u - u^0| \leq \sigma_u \right\}. \tag{4.1}$$

Here, the p-vector $\sigma_u > 0$ defines the componentwise uncertainty bounds for each dimension of the input, and $u^0 \in \mathbb{R}^p$ represents the vector of "nominal" inputs.

The second case introduces a *cardinality-limited uncertainty set*, which not only bounds the magnitude of deviations in the input but also restricts the number of components allowed to change:

$$\mathcal{U}^{\text{card}} := \left\{ u \in \mathbb{R}^p \ : \ |u - u^0| \leq \sigma_u, \ \ \mathbf{Card}(u - u^0) \leq k \right\}. \tag{4.2}$$

In this formulation, $\mathbf{Card}(\cdot)$ denotes the cardinality (i.e., the number of non-zero components) of its argument, and $k < p$ specifies the maximum allowable number of perturbed components. This constraint captures scenarios where a limited number of features can deviate from their nominal values, offering a more structured representation of input uncertainty.

These uncertainty models provide flexibility in characterizing diverse input perturbation scenarios. The following subsections explore how such input uncertainties propagate through the implicit model, affecting the state and output predictions.

Box Bounds on the State Vector

Assume that ϕ is a *CONE map*, and the input vector u is known to belong to the box-bounded uncertainty set $\mathcal{U}^{\text{box}}$ defined in (4.1). Our goal is to derive componentwise bounds on the state vector x, expressed as $|x - x^0| \leq \sigma_x$, where x and x^0 are the unique solutions to $\xi = \phi(A\xi + Bu)$ and $\xi = \phi(A\xi + Bu^0)$, respectively, and $\sigma_x > 0$ represents the bound.

Using the definition of x and x^0, we can write:

$$\begin{aligned} |x - x^0| &= |\phi(Ax + Bu) - \phi(Ax^0 + Bu^0)| \\ &\leq |A||x - x^0| + |B(u - u^0)|, \end{aligned}$$

where the inequality follows from the Lipschitz property of the CONE map ϕ.

This leads to the following upper bound:

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

Under the assumption that $\|A\|_\infty < 1$, we have:

$$\|x - x^0\|_\infty \leq \frac{\||B|\sigma_u\|_\infty}{1 - \|A\|_\infty}, \tag{4.3}$$

where σ_u is the componentwise uncertainty bound for the input vector.

For the cardinality-constrained uncertainty set $\mathcal{U}^{\text{card}}$ defined in (4.2), the bound becomes:

$$\|x - x^0\|_\infty \leq \frac{\delta}{1 - \|A\|_\infty}, \tag{4.4}$$

where

$$\delta := \max_{1 \leq i \leq n} s_k(\sigma_u \odot |B|^\top e_i),$$

with e_i representing the i-th unit vector in $\mathbb{R}^n$, and s_k denoting the sum of the top k entries in a vector. This formulation accounts for the sparsity constraint in the input perturbations, effectively limiting the number of perturbed components to k.

We can refine the analysis above by deriving a "box" (componentwise) bound for cases where ϕ is a *block-Lipschitz (BLIP)* map. The result involves the matrix of norms $N(A, \gamma)$ defined in (2.6), as well as a corresponding matrix of norms for the input matrix B.

To formalize this, we decompose B into blocks $B = (B_{li})_{l \in [L], i \in [p]}$, where each $B_{li} \in \mathbb{R}^{n_l}$ represents the interaction between the l-th block of the state vector and the i-th component of the input vector. Using this blockwise structure, we define an $L \times p$ matrix of norms, denoted as $N(B, \gamma)$, given by:

$$N(B, \gamma) := \left(\gamma_l \|B_{li}\|_{p_l}\right)_{l \in [L], i \in [p]}. \tag{4.5}$$

Here, γ_l is the Lipschitz constant of the l-th block of the activation map ϕ, and $\|B_{li}\|_{p_l}$ represents the norm of the block B_{li} with respect to the l_{p_l}-norm. This matrix captures the influence of input uncertainty on each block of the state vector, allowing for a more nuanced characterization of the state bounds in the presence of blockwise structure.

Theorem 5 (Box bound on the vector norms of the state, BLIP map)**.** *Assume that ϕ is BLIP, and the corresponding sufficient well-posedness condition of* (1) *is satisfied. Then, $I - N(A, \gamma)$ is invertible, and*

$$\eta(x - x^0) \leq (I - N(A, \gamma))^{-1} N(B, \gamma)\sigma_u, \tag{4.6}$$

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

Proof. Proof of Theorem (5)

For every $l \in [L]$, we proceed as follows:

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

which establishes the desired bound. Since $I - N(A, \gamma)$ is invertible under the well-posedness condition, we conclude:

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

□

Note that the box bound can be computed via fixed-point iterations. For instance, when ϕ is a CONE map, we solve the equation:

$$(I - |A|)\sigma_x = |B|\sigma_u,$$

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 $\lambda_{PF}(|A|) < 1$. This iterative procedure ensures efficient computation of the box bounds for the state vector.

Bounds on the Output

The previous analysis enables us to quantify the effect of input noise on the output vector y. Let us assume that the activation function ϕ satisfies the *Componentwise Non-Expansiveness (CONE)* condition of (2.3). Additionally, we assume the stricter well-posedness condition

$\|A\|_\infty < 1$ is satisfied. This condition can always be enforced by appropriately rescaling the model, provided $\lambda_{PF}(|A|) < 1$.

For the implicit prediction rule (2.1), the following bound holds:

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

This inequality shows that the prediction rule is Lipschitz-continuous, with a Lipschitz constant bounded above by ρ. The parameter ρ characterizes the sensitivity of the output predictions to perturbations in the input, providing a meaningful measure of the model's robustness to noise.

The above result motivates incorporating the $\|\cdot\|_\infty$ norm into the training objective, as a penalty on model parameters A, B, C, D. Specifically, a convex penalty can be designed to bound the Lipschitz constant ρ as follows:

$$\rho \leq \frac{1}{2} \frac{\|B\|_\infty^2 + \|C\|_\infty^2}{1 - \|A\|_\infty} + \|D\|_\infty. \tag{4.7}$$

By constraining ρ, we can ensure a trade-off between robustness and model complexity, improving the stability of the model under perturbations.

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

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

where the dual norm p_l^* is defined as $p_l^* := \frac{1}{1 - 1/p_l}$ for $l \in [L]$.

Additionally, recall the corresponding matrix of norms for A defined in (2.6) and for B in (4.5). These matrices provide a structured representation of how different blocks of the parameter matrices A, B, and C interact with the BLIP properties of ϕ.

The refined analysis extends the robustness bounds to models with BLIP activation functions, allowing for a more granular characterization of sensitivity to input noise. By leveraging the norm matrices $N(A, \gamma)$, $N(B, \gamma)$, and $N(C)$, we can compute tighter bounds on the Lipschitz constant and output sensitivity. This generalization ensures broader applicability across architectures with blockwise operations, such as convolutional or attention-based networks.

Theorem 6 (Box bound on the output, BLIP map)**.** *Assume 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*

$$\forall\, u, u^0 \quad |\hat{y}(u) - \hat{y}(u^0)| \leq S|u - u^0|, \tag{4.8}$$

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 special case where ϕ is a CONE map, the sensitivity matrix simplifies to:

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

Proof. Proof of Theorem (4.8)

For a given $i \in [q]$, consider the componentwise bound for the output difference:

$$|\hat{y}(u) - \hat{y}(u^0)|_i = \left| \sum_{l \in [L]} C_{il}(x - x^0)_l + \sum_{j \in [p]} D_{ij}(u - u^0)_j \right|.$$

Using the triangle inequality, we have:

$$|\hat{y}(u) - \hat{y}(u^0)|_i \leq \sum_{l \in [L]} |C_{il}(x - x^0)_l| + \sum_{j \in [p]} |D_{ij}(u - u^0)_j|.$$

For the first term, applying the dual norm relationship:

$$|C_{il}(x - x^0)_l| \leq \|C_{il}\|_{p_l^*} \|(x - x^0)_l\|_{p_l}.$$

Thus:

$$|\hat{y}(u) - \hat{y}(u^0)|_i \leq \sum_{l \in [L]} \|C_{il}\|_{p_l^*} \eta(x - x^0)_l + \sum_{j \in [p]} |D_{ij}(u - u^0)_j|.$$

Substituting the box bound on $\eta(x - x^0)$ from (4.6), we get:

$$\eta(x - x^0) \leq (I - N(A, \gamma))^{-1} N(B, \gamma)|u - u^0|.$$

Finally, substituting this into the output bound:

$$|\hat{y}(u) - \hat{y}(u^0)| \leq \left(N(C)(I - N(A, \gamma))^{-1} N(B, \gamma) + |D|\right) |u - u^0|,$$

proving the result. □

4.3 Sensitivity Matrix

The sensitivity matrix can be computed via fixed-point iterations, which are guaranteed to converge due to the well-posedness assumption stated in the theorem. For the case of CONE maps, these iterations are described as follows:

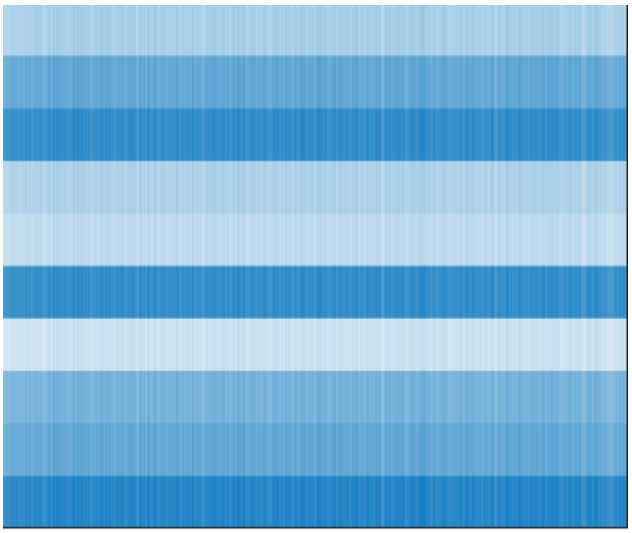

Figure 4.1: Sensitivity matrix for a 3-layer neural network with $q = 10$ outputs and $n = 1094$ states. The matrix has dimension $q \times n$ (10×1094 in this example)

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

where the sequence $X(t)$ converges to a limit point X_∞, provided $\lambda_{PF}(|A|) < 1$. Once X_∞ is determined, the sensitivity matrix S can be computed as:

$$S = |C|X_\infty + |D|.$$

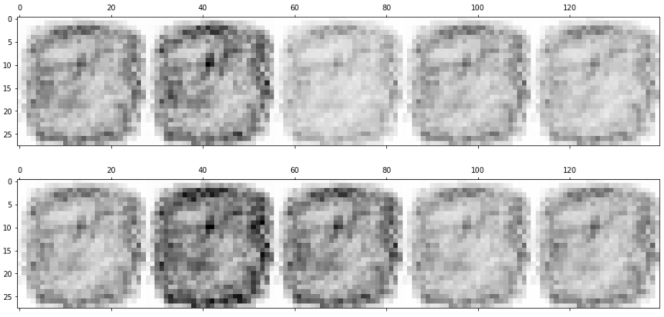

Figure 4.2: Sensitivity matrix for a 3-layer neural network with $q = 10$ outputs and $n = 1094$ states.

The implicit model formulation enables the analysis and bounding of distortions in the state vector x and the output $\hat{y}$ caused by unknown-but-bounded input noise. This provides valuable insights into the behavior of deep neural networks (DNNs) under uncertainty, a

condition inherent in most real-world applications. The sensitivity matrix, a key analytical tool, offers a means to visualize the input-output relationships of a given DNN model.

Figure 4.2 illustrates the sensitivity matrix for a neural network used in a classification task with $q = 10$ classes. Notably, certain classes exhibit higher sensitivity values, indicating greater susceptibility to input noise. This insight can guide resource allocation, as classes with higher sensitivity may require significantly more training data to achieve comparable classification accuracy.

Moreover, the sensitivity matrix serves as a powerful tool for adversarial analysis. By identifying the input features with the highest sensitivity, it informs adversaries on which features to perturb to significantly distort the model's output. This enables the generation of small, imperceptible, or sparse perturbations targeted at the most sensitive input features. Section 4.4 presents examples of adversarial images generated using this approach.

Our analysis also informs the design of adversarial defenses. We show that the prediction rule is Lipschitz-continuous, bounded above by κ, as defined in (4.9) in Section 4.2. This observation motivates the use of $\|\cdot\|_\infty$ as a regularization technique to promote robustness during training. Specifically, a convex penalty can be applied to bound the Lipschitz constant from above:

$$\kappa \leq P(A, B, C, D) := \frac{1}{2}\frac{\|B\|_\infty^2 + \|C\|_\infty^2}{1 - \|A\|_\infty} + \|D\|_\infty, \tag{4.9}$$

where P is the penalty function.

This regularization penalty can be further refined using the component-wise bound, making $\|S\|_\infty$, the norm of the sensitivity matrix, a natural choice for enhancing robustness during training. By integrating these insights, the approach not only strengthens model resilience against adversarial perturbations but also improves the interpretability and manageability of DNN behaviors under uncertain inputs.

Worst-case Loss Function

In this section, we analyze the worst-case behavior of the loss function under bounded input noise, assuming the activation map ϕ satisfies the CONE property in Section 2.3. Using the box bounds derived for the state and output vectors in Section 4.2, we compute bounds on the loss function evaluated between a given "target" $y \in \mathbb{R}^p$ and the prediction $\hat{y}$.

For the squared Euclidean loss function, defined as:

$$\mathcal{L}(y, \hat{y}) = \|y - \hat{y}\|_2^2,$$

the worst-case loss can be computed using the box bound (4.8) as:

$$\mathcal{L}_{\text{wc}}(y, \hat{y}^0) := \max_{|\hat{y} - \hat{y}^0| \leq \sigma_y} \mathcal{L}(y, \hat{y}) = \||y - \hat{y}^0| + \sigma_y\|_2^2,$$

where $\hat{y}^0$ represents the nominal prediction.

For the cross-entropy loss function, defined as:

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

where $y \geq 0$ and $\mathbf{1}^\top y = 1$ (representing the target class probabilities), we consider a simplified scenario with $D = 0$, leading to the nominal output $\hat{y}^0 := Cx^0$. Using the norm bound (4.3), we have $\|x - x^0\|_\infty \leq \rho$ for a suitable $\rho > 0$. The worst-case loss is then:

$$\mathcal{L}_{\text{wc}}(y, \hat{y}^0) := \max_{\delta x \,:\, \|\delta x\|_\infty \leq \rho} \mathcal{L}(y, \hat{y}^0 + C\delta x),$$

which can be expressed as:

$$\max_{\delta x \,:\, \|\delta x\|_\infty \leq \rho} \log\left(\sum_{i=1}^{q} e^{\hat{y}_i^0 + c_i^\top \delta x}\right) - y^\top(\hat{y}^0 + C\delta x),$$

where $c_i^\top$ is the i-th row of C.

Direct computation of this expression may be challenging, but we can derive an upper bound by separately evaluating the two terms:

$$\mathcal{L}_{\text{wc}}(y, \hat{y}^0) \leq \log\left(\sum_{i=1}^{q} e^{\hat{y}_i^0 + \rho\|c_i\|_1}\right) - y^\top \hat{y}^0 + \rho\|C^\top y\|_1.$$

Given $y \geq 0$ and $\mathbf{1}^\top y = 1$, we have $\|C^\top y\|_1 \leq \|C\|_\infty$, yielding:

$$\mathcal{L}(y, \hat{y}^0) \leq \mathcal{L}_{\text{wc}}(y, \hat{y}^0) \leq \mathcal{L}(y, \hat{y}^0) + 2\rho\|C\|_\infty.$$

Using the refined box bounds (4.8), we can further tighten this result:

$$\mathcal{L}_{\text{wc}}(y, \hat{y}^0) \leq \log\left(\sum_{i=1}^{q} e^{\hat{y}_i^0 + \sigma_{y,i}}\right) - y^\top \hat{y}^0 + y^\top \sigma_y.$$

These bounds highlight the impact of the sensitivity matrix on worst-case loss behavior, providing insights into robustness optimization during training.

Linear Programming (LP) Relaxation for CONE Maps

The previously discussed bounds do not provide a direct method to generate adversarial attacks, i.e., feasible points $u \in \mathcal{U}$ that maximize the impact on the state vector. In certain cases, however, it is possible to refine these box bounds using an LP relaxation, which has the added advantage of suggesting a specific adversarial attack. Here, we focus on the ReLU activation function $\phi(z) = \max(z, 0) = z_+$, which satisfies the CONE map property.

We consider the optimization problem:

$$p^* := \max_{x,u\in\mathcal{U}} \sum_{i\in[n]} f_i(x_i) \ : \ x = z_+, \ \ z = Ax + Bu, \ \ |x - x^0| \le \sigma_x, \tag{4.10}$$

where f_i are arbitrary functions. For example:

- Setting $f_i(\xi) = (\xi - x_i^0)^2$, $i \in [n]$, corresponds to finding the largest discrepancy (measured in ℓ_2-norm) between x and x^0.
- Setting $f_i(\xi) = -\xi$, $i \in [n]$, corresponds to minimizing the ℓ_1-norm of the state vector x.

By construction, this formulation improves on the previous state bound, ensuring:

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

To generalize our result for arbitrary sets $\mathcal{U}$, we use the *support function* $\sigma_{\mathcal{U}}$, which is defined for any $b \in \mathbb{R}^p$ as:

$$\sigma_{\mathcal{U}}(b) := \max_{u\in\mathcal{U}} b^\top u. \tag{4.11}$$

Note that the support function depends only on the convex hull of the set $\mathcal{U}$.

For specific cases, we have closed-form expressions for the support function:

- For the box set $\mathcal{U}^{\text{box}}$ defined in (4.1), the support function is:

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

  where u^0 is the nominal input, and σ_u represents the component-wise uncertainty.
- For the cardinality-constrained set $\mathcal{U}^{\text{card}}$ defined in (4.2), the support function is:

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

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

This LP relaxation provides a means to refine state bounds and identify specific input perturbations that maximize the impact on the state vector.

The only coupling constraint in (4.10) is the affine equation, which motivates the following relaxation.

Theorem 7 (LP Bound on the State)**.** *An upper bound on the objective of problem* (4.10) *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 $\sigma_{\mathcal{U}}$ is the support function defined in 4.11, and g_i, $i \in [n]$, are the convex functions defined as:

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

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

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

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

Proof. Proof of Theorem 7

We begin by rewriting the original optimization problem:

$$p^* \leq \overline{p} := \min_{\lambda} \ \max_{x,\, u\in\mathcal{U}} \ \sum_{i\in[n]} f_i(x_i) + \lambda^\top(Ax+Bu-z) \quad \text{subject to } x = z_+, \ |x - x^0| \leq \sigma_x.$$

Expanding the terms, we get:

$$\overline{p} = \min_{\lambda} \ \max_{u\in\mathcal{U}} \ \lambda^\top Bu + \max_{z\,:\,|z^+ - x^0|\leq\sigma_x} \ \sum_{i\in[n]} \left(f_i(z_i^+) + (A^\top\lambda)_i z_i^+ - \lambda_i z_i \right).$$

Defining $\mu = A^\top\lambda$, we rewrite the above expression:

$$\overline{p} = \min_{\lambda,\, \mu = A^\top\lambda} \ \left(\max_{u\in\mathcal{U}} \ \lambda^\top Bu \right) + \sum_{i\in[n]} g_i(\lambda_i, \mu_i),$$

which establishes the first part of the theorem.

Next, assume that the functions g_i, $i \in [n]$, are closed. By strong duality, we have:

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

Rearranging the terms, we arrive at the dual formulation:

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

where g_i^* is the conjugate of g_i, for $i \in [n]$. This completes the proof. □

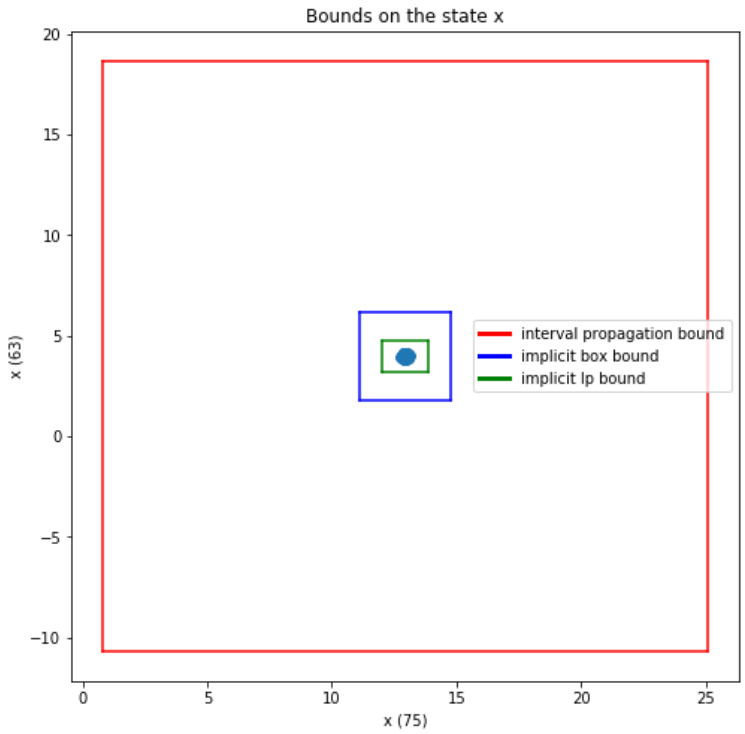

Figure 4.3: Visualization of bounds on the state vector x for different methods. The plots depict the implicit box bounds (blue), interval propagation bounds (red), and implicit LP bounds (green) for selected dimensions of the state vector.

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:

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

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

$$p^* \leq \overline{p} = \max_{x,\, u} c^\top x \quad \text{subject to} \quad \begin{array}{l} x \geq Ax + Bu, \;\; x \geq 0, \;\; |x - x^0| \leq \sigma_x, \\ \| \operatorname{\mathbf{diag}}(\sigma_u)^{-1}(u - u^0)\|_1 \leq k, \;\; |u - u^0| \leq \sigma_u. \end{array}$$

Figure 4.3 compares three types of bounds on the state vector x across two random dimensions: interval bound propagation (IBP) [36], implicit box bounds, and implicit LP bounds. Interval bound propagation refers to a method for propagating input uncertainty through a neural network layer by layer, using interval arithmetic to bound the range of possible values at each layer. While IBP is computationally efficient and widely used for robustness certification, it tends to produce conservative estimates of the uncertainty region.

This conservatism, illustrated by the red bounds in the figure, often leads to significant overestimation of the true uncertainty region, resulting in inefficiencies in robustness analysis or adversarial attack generation.

In contrast, the implicit box bounds (blue) provide a significantly tighter estimation of the feasible region by leveraging the structural properties of the implicit model and the sufficient well-posedness conditions. These bounds are more informative, capturing the constraints imposed by the model's architecture and activation functions.

The implicit LP bounds (green), derived using a Lagrange relaxation, are the tightest among the three methods. These bounds exploit both the linear structure of the system and the precise input uncertainty constraints to characterize the state space with high accuracy. The tightness of these bounds not only improves interpretability but also provides actionable insights for generating adversarial attacks. By identifying specific input perturbations that maximize the impact on the state or output, implicit LP bounds enable more targeted and effective adversarial analysis. Together, these results highlight the advantages of implicit bounds over interval propagation methods for applications requiring robust and precise estimates of state distortions.

4.4 Adversarial Attacks via Implicit Representation

Attack via the Sensitivity Matrix

The preceding analysis highlights the utility of the *sensitivity matrix* as a critical tool for evaluating robustness. In this section, we demonstrate how the sensitivity matrix can be leveraged to craft effective adversarial attacks on two widely used public datasets: MNIST and CIFAR-10.

We evaluate the performance of sensitivity-matrix-based attacks in comparison to commonly used gradient-based adversarial attack methods, including the *Fast Gradient Sign Method (FGSM)* [33] and the *Jacobian-based Saliency Map Attack (JSMA)* [70]. The experiments are conducted on the following two neural network architectures:

- *Feed-forward Neural Network (FFNN):* A three-layer fully connected feed-forward network trained on the MNIST dataset, achieving 98% clean accuracy. The model utilizes ReLU activations and is optimized using cross-entropy loss.
- *ResNet-20:* A 20-layer deep residual network [44] trained on the CIFAR-10 dataset, achieving 92% clean accuracy. The model incorporates batch normalization and ReLU activations for stable and efficient training.

Through these experiments, we highlight the effectiveness of sensitivity-based attacks in reducing model accuracy while adhering to a fixed perturbation budget. These results

underscore the utility of the sensitivity matrix for targeted adversarial attack generation, as well as its potential to inform robust model design.

We compare our method against commonly used gradient-based adversarial attacks. Specifically, for a given prediction function F learned by a deep neural network, a benign input sample $u \in \mathbb{R}^p$, and its corresponding target y, we compute the gradient of F with respect to u, denoted as $\nabla_u F(u, y)$. The absolute value of the gradient, $|\nabla_u F(u, y)|$, serves as an indication of which input features an adversary should perturb. This approach aligns with saliency map techniques such as those proposed in [86, 70].

The sensitivity matrix, however, offers a key distinction. Unlike the gradient, which depends on the specific input sample u, the sensitivity matrix provides a global measurement of robustness that is independent of the input data. As a result, it serves as a more general-purpose tool for evaluating a model's susceptibility to adversarial perturbations. By leveraging the sensitivity matrix, our method demonstrates its capability to generate attacks that are both effective and computationally efficient, while providing insights into the model's inherent vulnerabilities.

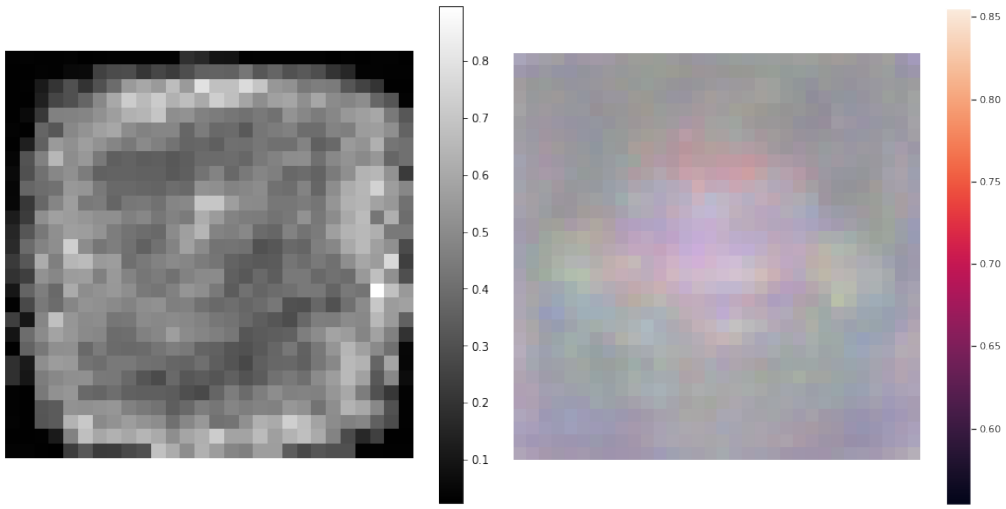

Figure 4.4: **Left:** Sensitivity values for a feed-forward network trained on MNIST, visualized for the class "digit 0." Darker regions indicate higher sensitivity to input perturbations, showing which pixels significantly influence the network's predictions. **Right:** Sensitivity values for a ResNet-20 model trained on CIFAR-10, visualized for the class "airplane." Color intensity highlights the areas most sensitive to perturbations, with high sensitivity values concentrated on key image regions associated with the class.

Figure 4.4 presents visualizations of the sensitivity matrix for two distinct models and datasets. On the left, the sensitivity values for a feed-forward network trained on MNIST are shown for the class "digit 0." The visualization highlights regions in the input space where small perturbations can have the most significant impact on the model's predictions.

Darker areas represent higher sensitivity, indicating that these pixels play a critical role in classification. On the right, the sensitivity matrix for a ResNet-20 model trained on CIFAR-10 is visualized for the class "airplane." The color intensity emphasizes regions of high sensitivity, which are often concentrated in areas containing features strongly associated with the target class, such as edges or structural details of the object. These sensitivity visualizations not only reveal the model's reliance on specific features for prediction but also provide insights into potential vulnerabilities. Adversaries can exploit these high-sensitivity regions to craft effective and targeted attacks, while developers can leverage this information to enhance model robustness by focusing on these critical areas during training.

Table 4.1: 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%

Table 4.1 presents the experimental results of adversarial attacks using the sensitivity matrix and the absolute value of the gradient on MNIST and CIFAR-10 datasets. For the sensitivity matrix attack, we start by perturbing the input features with the highest values according to the sensitivity matrix. Similarly, for the gradient-based attack, we perturb the features with the highest absolute gradient values. In both cases, the input features are perturbed into small random values.

Our experiments demonstrate that the sensitivity matrix attack is as effective as the gradient-based attack while being significantly simpler to implement. This highlights the practical advantage of the sensitivity matrix for adversarial attack generation. Interestingly, the sensitivity matrix attack does not require specific input samples, unlike gradient-based methods. An adversary with access to the model parameters can easily craft adversarial samples using the sensitivity matrix. In cases where the model parameters are unavailable, an adversary can leverage the principle of *transferability* [62], training a surrogate model to approximate the sensitivity matrix.

Figure 4.5 demonstrates the effectiveness of adversarial attacks using both dense and sparse perturbation strategies on MNIST (top) and CIFAR-10 (bottom) datasets. For MNIST, the left panel showcases dense attacks where small perturbations are applied uniformly across the image, causing the classifier to misidentify the digits while maintaining the visual appearance. The right panel highlights sparse attacks where only a few strategically chosen pixels, marked in red, are perturbed to achieve the same mis-classification. These sparse

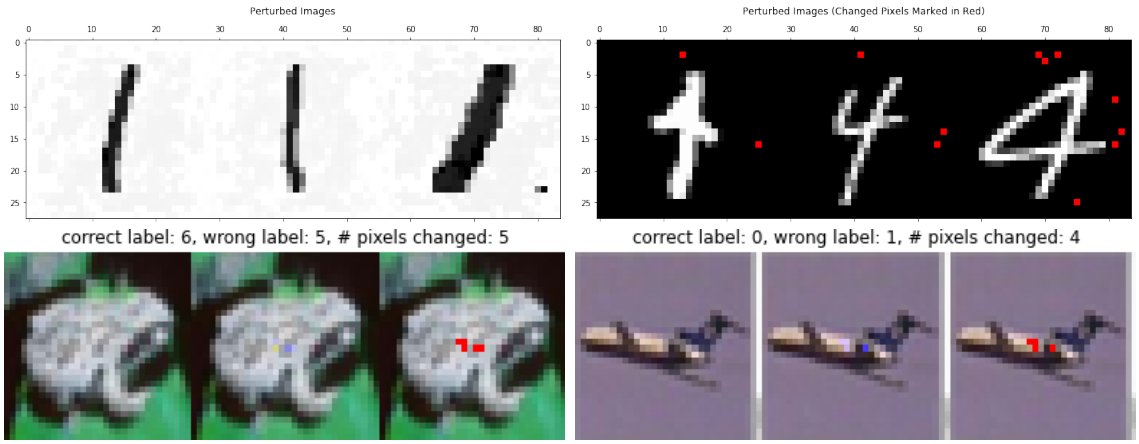

Figure 4.5: **Top**: adversarial samples from MNIST. On the left are dense attacks with small perturbations and on the right are sparse attacks with random perturbations (perturbed pixels are marked as red). **Bottom**: example sparse attack on CIFAR-10. The left ones are cleaned images, the middle ones are perturbed images, and the right ones mark the perturbed pixels in red for higher visibility.

attacks are particularly significant as they alter the classification (e.g., "6" to "5" or "0" to "1") with minimal modifications, emphasizing the potency of targeted perturbations.

For CIFAR-10, the first column displays clean images, the middle column shows sparsely perturbed adversarial examples, and the rightmost column marks the perturbed pixels in red for visibility. These sparse perturbations subtly manipulate the classifier's predictions (e.g., changing the label from "airplane" to another class) without noticeably affecting the visual quality of the images. This highlights the vulnerability of deep learning models to small, imperceptible changes, and the effectiveness of sparse attacks in generating adversarial samples with minimal input modification.

Attack with LP Relaxation for CONE Maps

Although the sensitivity matrix can be used to generate effective adversarial examples, a more sophisticated attack may be desirable by exploiting the specific weaknesses of an individual data point. This can be achieved using the LP relaxation outlined in Theorem (7), which has the advantage of producing a targeted adversarial example for a given input. The experiment in this section is conducted on MNIST and CIFAR-10 datasets. Specifically, the optimization problem outlined in (4.10) is solved using the LP relaxation, with the function $f_i(\xi) = (\xi - x_i^0)^2$. This formulation identifies perturbed images that maximize the discrepancy between the perturbed state x and the nominal state x^0.

Figure 4.6 presents five examples of perturbed images generated through the LP relaxation. Despite appearing visually similar to the original images, the perturbed images effectively

cause the model to misclassify instances that it otherwise predicts correctly. These results demonstrate the potency of adversarial examples generated by LP relaxation, offering a refined method for targeted attacks that leverage the inherent vulnerabilities of specific data points.

Our framework also supports sparse adversarial attacks by incorporating a cardinality constraint. Figure 4.7 illustrates three examples of perturbed images under both non-sparse and sparse attack scenarios. The images on the left represent the outcomes of non-sparse attacks, while those on the right depict sparse attacks. In both cases, the model fails to correctly predict the labels of the perturbed images. These results demonstrate the effectiveness of implicit prediction rules in generating powerful adversarial attacks. Furthermore, this capability is highly valuable for adversarial training, as it enables the generation of a large number of adversarial examples that can be added back to the training dataset, thereby enhancing the robustness of the model against such attacks.

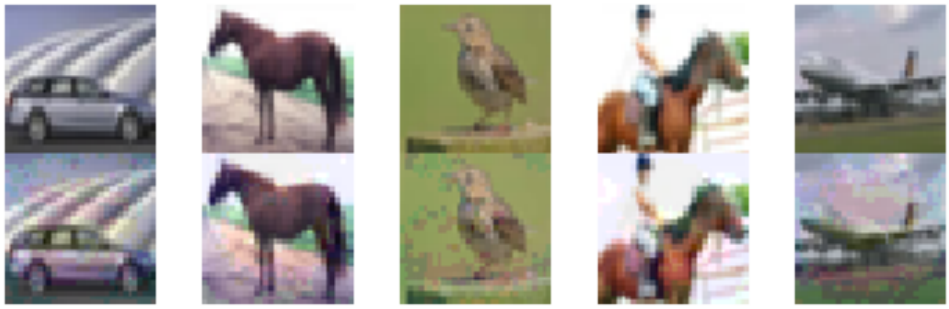

Figure 4.6: Example attack on CIFAR dataset. Top: clean data. Bottom: perturbed data.

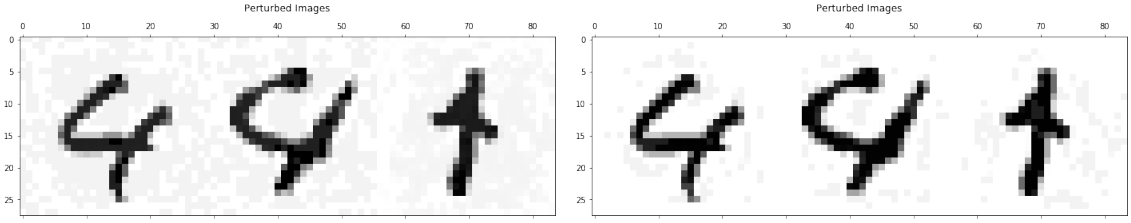

Figure 4.7: Example attack on MNIST dataset. Left: non-sparse attack. Right: sparse attack.

4.5 Conclusion

In this chapter, we presented a comprehensive robustness analysis for implicit models, emphasizing their theoretical properties and practical applications. By building on the

concept of well-posedness, we established a rigorous framework for studying the behavior of implicit models under input perturbations. We derived key results, including box bounds on the state and output vectors, as well as the *sensitivity matrix*, which serves as a valuable tool for quantifying and visualizing model robustness.

The sensitivity matrix emerged as a central theme in our analysis, offering an intuitive and effective measure for identifying input features most susceptible to adversarial perturbations. We demonstrated its practical utility through experiments on MNIST and CIFAR-10 datasets, where it facilitated the generation of targeted adversarial attacks comparable in effectiveness to gradient-based methods, but with greater interpretability and flexibility. The matrix also provided insights into the inherent robustness of different classes and model components, highlighting its potential for guiding robust model design and adversarial training.

Additionally, we extended our analysis to include more sophisticated adversarial attacks using LP relaxations, showcasing their ability to exploit individual data point weaknesses. These attacks, which can be tailored to generate both sparse and non-sparse perturbations, further underscore the versatility and power of implicit model representations in understanding and addressing adversarial vulnerabilities.

Our findings also offer actionable insights for improving model robustness during training. Specifically, the bounds and penalties derived in this chapter, such as those involving the sensitivity matrix or output bounds, point to natural regularization strategies that could be integrated into future training pipelines. These techniques provide a pathway for not only diagnosing but also mitigating vulnerabilities in neural networks.

In conclusion, this chapter demonstrates the strength of the implicit model framework in formalizing and addressing robustness challenges in modern neural networks. Building on this foundation of robustness, the next chapter explores a complementary strength of implicit models: their extrapolation power. While robustness focuses on how models handle perturbations within their training domain, extrapolation examines their ability to generalize effectively to out-of-distribution data and unseen scenarios. This property is crucial for understanding how implicit models adapt to novel environments and diverse tasks, making them a versatile tool for real-world applications. In the following chapter, we discuss the extrapolation capabilities of implicit models, investigating their unique potential to outperform traditional architectures in challenging settings.

Chapter 5

The Extrapolation Power of Implicit Models

5.1 Introduction

Learning to extrapolate–estimating unknown values beyond the scope of observed data–is a cornerstone of human intelligence and a critical step towards advancing machine learning systems capable of generalization. Despite their widespread success across diverse domains, modern neural networks often fail to extrapolate effectively when faced with data outside their training distribution. This limitation presents a fundamental challenge in deploying these models in dynamic and unpredictable environments.

In this chapter, we explore the extrapolation power of a general class of *implicit* deep learning models [8, 16, 17, 24], which generalize traditional layered neural networks. Implicit deep learning models determine their representations via fixed-point equations, allowing information to propagate both forwardly and backwardly through *closed-form feedback loops*. This distinctive architecture enables implicit models to overcome many limitations of traditional feed-forward networks and provides a promising framework for learning data representations in complex and unstructured domains.

Several formulations of implicit models include deep equilibrium models (DEQs) [8], Neural ODEs [17], and general implicit models discussed in this thesis. Unlike classical neural networks that compute outputs in a strictly layered, feed-forward manner, implicit models define their state vectors through an equilibrium equation. Outputs are implicitly determined by solving this equilibrium equation. This unique computational paradigm allows for the design of more flexible and expressive models, which recent studies have shown to excel in both practical applications and theoretical generalization [10, 40, 93].

From a neuroscience perspective, the equilibrium state in implicit models has been interpreted as a closed-loop feedback system, mimicking how the brain processes information through recurrent and feedback loops [64]. This feedback mechanism allows inputs to revisit

and reverse directions within the computational graph, distinguishing implicit models from traditional feed-forward architectures.

In the following sections, we explore how implicit models leverage their unique structure to extrapolate effectively in out-of-distribution scenarios. By investigating the theoretical and empirical properties of implicit models, we aim to demonstrate their potential to surpass traditional architectures in tasks requiring robust extrapolation capabilities. In this chapter, we investigate whether implicit deep learning models exhibit superior extrapolation capabilities—a fundamental aspect of human intelligence—compared to similarly sized non-implicit neural network models. Our contributions are summarized as follows:

- **Demonstrating Extrapolation Power:** We showcase the extrapolation capabilities of implicit deep learning models across three diverse domains: (1) well-defined mathematical arithmetic, (2) real-world earthquake location data, and (3) volatile time series forecasting. These experiments highlight the adaptability and robustness of implicit models in handling out-of-distribution data.
- **Analyses and Ablation Studies:** We perform detailed analyses and ablation studies focusing on two critical factors of implicit models: *depth adaptability* and *closed-loop feedback*. Our results reveal that features learned by implicit models are more generalizable than those of non-implicit models, underscoring their capacity for effective learning and extrapolation.

Mathematical tasks. Previous research has primarily focused on developing specialized neural network models capable of learning algorithms [37, 38, 52, 60, 85]. For instance, Neural Arithmetic Logic Units (NALUs) were specifically designed to represent mathematical relationships within their architecture, aiming to improve arithmetic extrapolation [92]. However, these models were later found to exhibit significant instability during training [84]. Neural Arithmetic Expression Calculators (NAECs) utilize reinforcement learning to solve mathematical expressions involving addition, subtraction, multiplication, and division [15]. Unlike NALU, NAECs require the mathematical operation as an additional input to the network. Studies by Nogueira, Jiang, and Lin demonstrated that Transformers performed effectively for addition and subtraction tasks, achieving high accuracy in interpolation experiments. However, challenges arose with other Transformer-based architectures like BART [97] and large language models (LLMs) [100], which struggled to accurately reproduce functions for wide distribution ranges. On the contrary, Charton showed that Transformers could provide "roughly correct" solutions for matrix inversion and eigenvalue decomposition tasks, even for out-of-distribution (OOD) inputs, indicating a notable level of mathematical understanding.

Out-of-distribution generalization. Only a handful of studies have explored out-of-distribution (OOD) generalization for implicit deep learning models [60, 74, 54]. These

works highlight the capabilities of implicit deep learning models on tasks like Blurry MNIST, sequential tasks [60], matrix inversion, and graph regression [3]. Researchers such as Liang et al. and Anil et al. emphasize a unique property of deep equilibrium models (DEQs) known as path independence, where the models converge to similar fixed points regardless of initialization. This property suggests that DEQs could gather more information on OOD inputs by iterating longer before converging, potentially outperforming other models. [77] theorized that this property is most beneficial when testing DEQs on data more complex than the training distribution. They also demonstrated that increasing the number of inner iterations could lead to overfitting on interpolation tasks. Our work provides further evidence supporting these hypotheses by examining well-defined functions and real-world datasets with OOD characteristics.

Function extrapolation. Xu et al. studied the extrapolation behavior of ReLU-based multi-layer perceptrons (MLPs) and graph neural networks on quadratic, cosine, and linear functions. They identified specific architectural choices that enhance extrapolation, such as encoding task-specific non-linearities into model features. Similarly, in the vision domain, Webb et al. introduced context normalization to obtain more generalized features. Additionally, Wu et al. demonstrated the benefits of neural networks with Hadamard products (NNs-Hp) and polynomial networks (PNNs) for arithmetic extrapolation. In this paper, we hypothesize that implicit models inherently "adapt" to distribution changes, eliminating the need for task-specific feature transformations to achieve effective extrapolation.

5.2 Problem Setup

We explore two types of implicit models: standard implicit models as defined in (2.1b), and a variant referred to as *implicitRNN*.

ImplicitRNN. The *implicitRNN* functions similarly to a vanilla RNN, processing sequential inputs one step at a time. For each time step i, where $s_i \in \mathbb{R}^p$ represents the i-th element in a sequence $(s_1, s_2, \cdots, s_t)$, the model input u for the implicitRNN is a concatenation of s_i and the previous hidden state h_{i-1}. This setup is analogous to a vanilla RNN. The implicit prediction rule for implicitRNN is defined as follows:

$$
\begin{aligned}
&h_0 = \mathbf{0}, \quad x_0 = \mathbf{0}, \quad u_i = \begin{pmatrix} s_i \\ h_{i-1} \end{pmatrix} \\
&x = \phi(Ax + Bu_i) \quad \text{(equilibrium equation)} \\
&\hat{y}_i(u_i) = Cx + Du_i \quad \text{(prediction equation)} \\
&h_i = \hat{y}_i(u_i).
\end{aligned}
$$

In this setup, the recurrent layer is replaced by an implicit structure consisting of the equilibrium and prediction equations, as illustrated in Figure 5.1. The implicitRNN model

is introduced to compare implicit models with explicit sequential models, while both retain representations of data step-by-step.

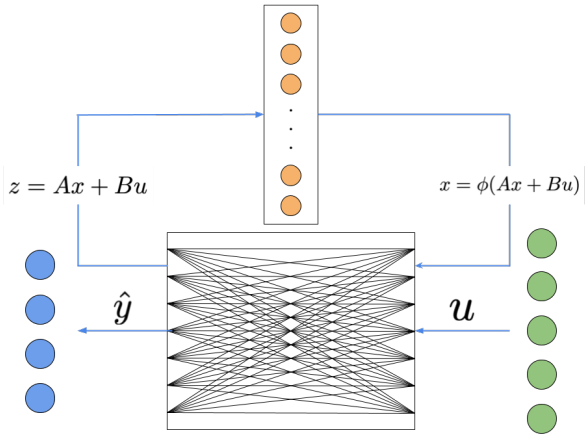

Figure 5.1: An implicit block. We replace the linear cell in a vanilla RNN with an implicit block not distinguishing between the output and the recurrent hidden state.

Extrapolate on Mathematical Tasks

We explore three categories of functions, ranging from simple to complex:

1. *Identity function:* A basic mapping task where the output is expected to match the input exactly.
2. *Arithmetic operations:* Tasks involving addition and subtraction with additional data transformation.
3. *Rolling functions over sequential data:* Tasks such as computing averages and identifying the index of the maximum value (argmax) in a rolling sequence.

To evaluate the extrapolation performance of implicit models, we generate two datasets:

- Training data: $u_{\text{train}} \sim P(\mu; \sigma)$
- Testing data: $u_{\text{test}} \sim P(\mu + \kappa; \sigma + \kappa)$

Here, P represents a known probability distribution, μ and σ are the mean and standard deviation, respectively, and κ is a hyper-parameter introducing a distributional shift to test the model's extrapolation capabilities.

Identity function. Recent research has highlighted the challenges neural networks face in learning the identity mapping $f(u) = u$, where the output should exactly replicate the input [45, 92]. Despite its simplicity, this task remains a benchmark for evaluating a model's ability to preserve input fidelity.

Arithmetic operations. We consider two arithmetic tasks: *addition* and *subtraction*, involving transformed data. Following the framework proposed by Trask et al., we randomly select indices i, j, k, l from the range [1, 50], ensuring $i < j$ and $k < l$. For each input vector $\vec{u} := \langle u_1, u_2, \cdots, u_{50} \rangle$, we compute:

$$a = \sum_{a=i}^{j} u_a, \quad b = \sum_{b=k}^{l} u_b$$

The outputs are defined as:

$$y = a + b \quad \text{(addition task)}, \quad y = a - b \quad \text{(subtraction task)}.$$

Rolling function over sequential data. We investigate two rolling tasks over sequences: *average* and *argmax*.

- *Rolling average*: For each timestep j, the model predicts the average of the sequence up to j, given by:
$$\frac{1}{j} \sum_{i=1}^{j} u_i.$$
- *Rolling argmax:* The model predicts the index of the maximum value observed so far in the sequence up to timestep j. This task is cast as a classification problem where the output is a one-hot encoded vector of length $L = 10$, representing the index of the maximum input observed. The evaluation focuses on predictions for the final element of the sequence.

These tasks provide a diverse set of challenges for assessing the extrapolation power of implicit models across structured and sequential data scenarios.

We compare implicit models against neural networks specifically designed to excel on each task:

- *MLPs* for simple functions such as the identity mapping.
- *LSTMs* for tasks involving sequential data [48].
- *NALUs* for out-of-distribution (OOD) arithmetic tasks [92].

- *Transformer-based models* for more complex mathematical tasks [95].

The details of each task including the architectures of the compared models, are provided in Table 5.5. All models are optimized using grid search and 5-fold cross-validation on in-distribution inputs to ensure consistency and reliability in performance evaluation.

To ensure a fair comparison, the number of parameters in the implicit models is matched to their non-implicit counterparts. This allows us to directly assess the extrapolation capabilities of the implicit models without confounding factors such as model size or capacity. These rigorous evaluation setups provide a strong foundation for analyzing the relative strengths and weaknesses of implicit models in various extrapolation scenarios.

Extrapolate on Noisy Real-world Data

Beyond mathematical extrapolation, where data is generated from a known underlying function, we explore extrapolation on real-world problems characterized by noisy data and the absence of explicit data generation functions. These tasks require robust generalization to predict unseen events, making them ideal benchmarks for evaluating extrapolation capabilities.

We focus on two challenging real-world applications:

- **Oscillating Time Series Forecasting:** Predicting future states of oscillatory systems, such as weather patterns or stock market indices, which exhibit non-linear and periodic behaviors.
- **Earthquake Location Prediction:** Estimating the location of earthquakes from seismic readings, where data often contains noise and limited coverage, making it inherently difficult to generalize beyond observed events.

These applications test the ability of models to extrapolate from incomplete and noisy training data to unobserved regions, providing insights into their robustness and adaptability under real-world conditions.

Oscillating Time Series Forecasting

Spiky Synthetic Data. We first examine a synthetic benchmark, *spiky time series forecasting*, where spikes are randomly inserted into a periodic time series derived from a combination of sine functions. This controlled setting tests the extrapolation capabilities of models on periodic and non-linear sequences.

The dataset consists of 7,000 training data points and 3,000 testing data points. The training set includes 20 spiky regions, each containing 100 data points, with a proportionate

number of spiky regions in the test set. Data points in the spiky regions are sampled from the function

$$y = 5 \times (\sin(2x) + \sin(23x) + \sin(78x) + \sin(100x)),$$

where the frequencies in the range $[0, 100]$ are chosen arbitrarily to generate a sufficiently spiky pattern. The magnitude of the spiky regions is capped at 20 to simulate high-intensity fluctuations.

Outside the spiky regions, data points are sampled from a simpler function,

$$y = \sin(x) + \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, 0.25)$ represents added Gaussian noise. This setup ensures a clear distinction between spiky and non-spiky regimes, allowing for a robust evaluation of extrapolation performance. Both LSTM and implicit models are evaluated under these conditions to compare their ability to generalize beyond the training distribution.

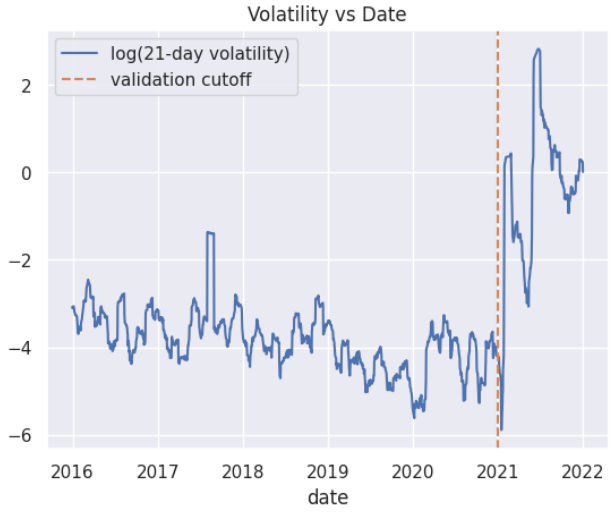

Figure 5.2: Time series of a 21-day rolling average of AMC stock volatility plotted on a log scale, highlights a drastic volatility increase at the beginning of our validation cutoff.

AMC Stock Volatility. We further investigate real-world financial data by forecasting AMC stock volatility, particularly focusing on the significant increase in average volatility observed in early 2021, as illustrated in Figure 5.2. Volatility, defined as the variance of volume-weighted average prices (VWAP), serves as a measure of the risk or uncertainty associated with the price fluctuations of a security. The task involves predicting AMC's volatility over the next 10 minutes, using the VWAP for each of the past 60 minutes as input.

To highlight distributional shifts, we deviate from the conventional approach of calculating volatility as the standard deviation of returns. Instead, we compute the variance of raw prices, amplifying the difference between the training and test datasets.

The training dataset covers the period from February 1, 2015, to December 31, 2020, while the validation set spans January 1, 2021, to December 31, 2021. To further challenge the models, we refrain from stationarizing the data via differencing or return calculations, emphasizing the importance of adaptability to evolving price distributions. We compare the performance of implicit models against several baseline approaches, including simple linear regression and non-implicit neural networks. Details of the baseline architectures are summarized in Table 5.5.

Earthquake Location Prediction

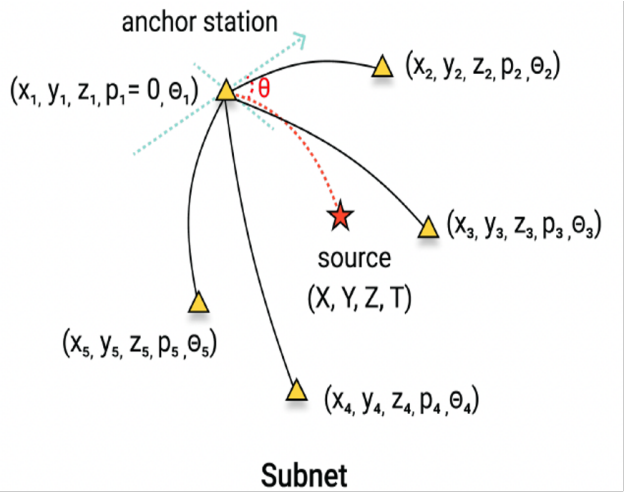

Figure 5.3: Geometric visualization of one set of training features (x_i, y_i, z_i, p_i, θ_i) and its corresponding labels (X, Y, Z, T). The triangles correspond to stations and the star corresponds to a source.

The earthquake location prediction problem is a well-established challenge in seismology [87, 80]. The task involves predicting the location (X, Y, and Z coordinates) and the seismic wave travel time (T) of an earthquake based on data recorded from nearby seismometers. Accurate solutions to this problem hold significant humanitarian importance, as they could enable early warnings before the arrival of potentially destructive secondary waves of an earthquake. Despite its importance, the problem remains challenging due to the sparsity of seismic event observations [19].

For this experiment, we compare the performance of general-purpose implicit models against EikoNet [87], a recently developed deep learning model specifically designed for earthquake location prediction. Notably, our results demonstrate that implicit models, despite their generality, can outperform EikoNet when tasked with out-of-distribution (OOD) location predictions. This highlights the potential of implicit models to generalize beyond

the constraints of domain-specific architectures, providing a robust alternative for seismic event analysis.

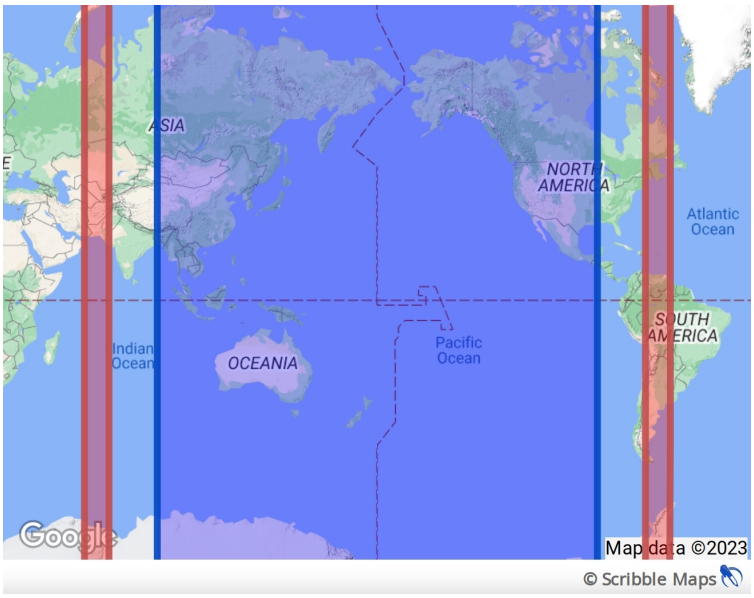

Figure 5.4: The map shows the training set region colored in blue, roughly corresponding to the Pacific Ring of Fire. The two red areas are the testing set regions for $k = 3$.

We follow the methodology outlined by Chuang et al., generating synthetic seismograph samples recorded by five stations, with one designated as the anchor station to serve as the reference for all seismic wave arrival times. As illustrated in Figure 5.3, the input features include a station's coordinates (x, y, z), event-station back-azimuths (θ), and relative wave travel times (p) with respect to the anchor station.

The training data is synthetically generated within a range of 90°E to −90°E, approximately corresponding to the Pacific Ring of Fire, as shown in Figure 5.4. Testing is conducted on regions shifted from 10°E to 90°E beyond this Ring of Fire.

For comparison, we use EikoNet, a deep learning model introduced by Smith, Azizzadenesheli, and Ross specifically designed for earthquake location prediction. While earthquakes primarily occur along active tectonic boundaries, an extrapolated earthquake location prediction system has broader applications, including detecting earthquakes in new areas, whether natural or human-induced (e.g., mining, oil, and gas activities). Such systems are also valuable for explosion monitoring, providing a universal mapping capability across diverse seismic events.

5.3 Numerical Experiments

Mathematical Extrapolation

Identity Function. The test mean squared error (MSE) for the identity function task is illustrated in Figure 5.5. The implicit model consistently achieves the lowest test MSE (below 5) for test data exhibiting a distribution shift from 0 to 25. Even under substantial distribution shifts of up to 200, where $u_{\text{test}} \in \mathbb{R}^{10} \sim U(-205, 205)$, the implicit model outperforms the Transformer encoder model by a factor of 10^5 and the MLP by a factor of 10^3.

For the identity function and arithmetic operation taks, we experimented with 15 different activation functions on our MLP: hardtanh, sigmoid, reLU6, tanh, tanhshrink, hardshrink, leakyrelu, softshrink, softsign, reLU, preLU, multipreLU, softplus, eLU and seLU. We tried to understand whether specific activations helped the MLP extrapolate as well as our implicit model. Table 5.1 summarizes the results of 5 of these activation functions on our identity function task as compared to the implicit deep learning model.

Table 5.1: Testing loss of our implicit model and five MLP models with specific activations on the identity function task. We observe the implicit model outperforms the MLP across activation functions. Description of the activation functions in the appendix.

	Train MSE		**Test MSE**	
Activation	**MLP**	**Implicit**	**MLP**	**Implicit**
ReLU	2.14×10^{-3}	12.4×10^{-1}	21.6	2.16
Leaky ReLU	3.28×10^{-3}	-	22.3	-
Softplus	1.57×10^{-2}	-	17.1	-
Softsign	3.01×10^{-1}	-	47.5	-
Log sigmoid	1.71×10^{-2}	-	17.1	-

This significant performance gap underscores the robustness of implicit models in handling distribution shifts, contrasting sharply with non-implicit models such as MLPs and Transformers, which tend to overfit to the training distribution and exhibit dramatically increased errors under larger distribution shifts. The identity function task highlights the spurious features often learned by these non-implicit models.

Moreover, our implicit model achieves equilibrium after only 4 training iterations, demonstrating its efficiency. By contrast, similarly sized MLPs and Transformer encoders face challenges with overfitting, which emphasizes the implicit model's ability to mitigate overfitting through rapid convergence, particularly for simpler tasks like identity mapping.

Arithmetic Operations. The results for the addition and subtraction arithmetic tasks are presented in Figure 5.6. The implicit model not only outperforms various Transformer encoders but also surpasses NALU, a model specifically designed for mathematical operations.

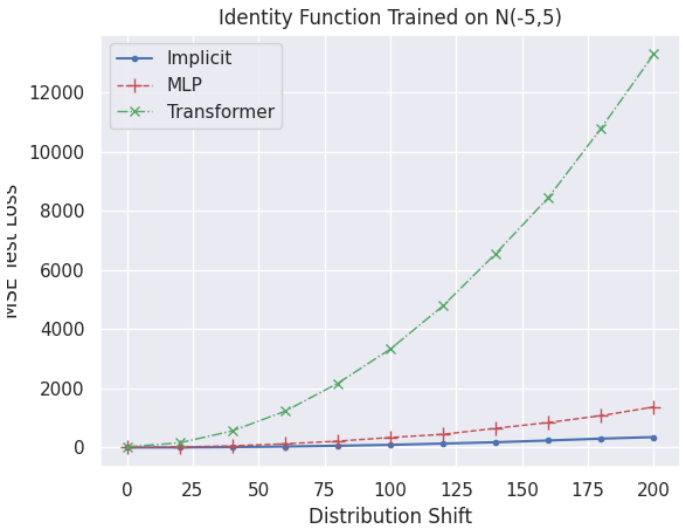

Figure 5.5: Test MSE for the identity function task. MSE for MLP and Transformers model increases as the distribution shift hyper-parameter κ increases.

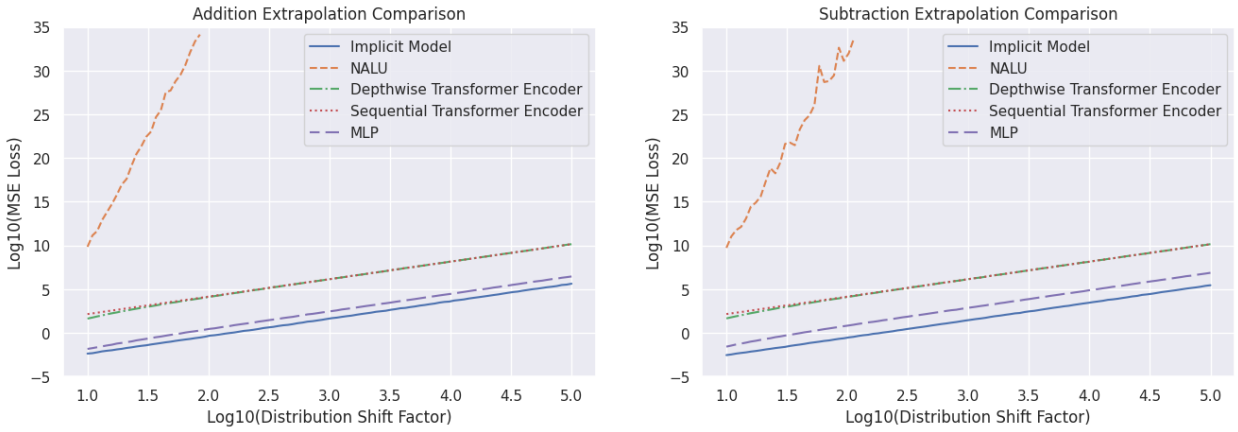

Figure 5.6: Test Log(MSE) for the arithmetic operations. The implicit model strongly outperforms all other models on OOD data.

As demonstrated by Wei et al., Transformer models typically require significantly larger model sizes (e.g., 10^{23} parameters) to perform arithmetic tasks effectively. In contrast, the implicit model, with only 7,650 parameters, successfully learns these operations, achieving a testing loss of less than 1 for distribution shifts smaller than 100.

Table 5.2 compares the test MSE for our MLP with ReLU activation, the best MLP across all 15 activations and our implicit model. We have $x_{\text{train}} \in \mathbb{R}^{100} \sim U(1,2)$ and $x_{\text{test}} \in \mathbb{R}^{100} \sim U(2,5)$. For both operations, the implicit model greatly outperforms the MLP regardless of the activation function.

Table 5.2: Test MSE table of two MLPs and our implicit model on arithmetic operations. The best MLP for both tasks was with ReLU6 activation.

Operation	**ReLU MLP**	**Best MLP**	**Implicit**
Addition	6.95×10^{31}	8.50×10^{3}	16.07
Subtraction	3.69×10^{19}	1.87×10^{4}	3.40×10^{-2}

Throughout the experiments, implicit models consistently outperformed non-implicit models in extrapolation tasks, particularly when training samples were limited. Surprisingly, the specialized NALU model exhibited the worst performance, with testing losses exceeding 10^{10} for an extrapolation shift of merely 10, as depicted in Figure 5.6. Across all experiments, the NALU model failed to produce robust out-of-distribution predictions, highlighting the limitations of its specialized architecture compared to the general-purpose implicit model.

Rolling Functions. Figures 5.7 and 5.8 illustrate the performance of implicit models compared to LSTMs and Transformers on two sequence modeling tasks. In the rolling average task (Figure 5.7), the LSTM and Transformers exhibit similar behaviors, with loss increasing as the test distribution shifts further from the training data. By contrast, the implicit model maintains a nearly constant loss, demonstrating superior robustness to distributional changes.

For the rolling argmax task (Figure 5.8), we employ a Transformer encoder-decoder architecture where the target sequence consists of right-shifted argmax labels. This setup inherently favors Transformers, as simply outputting the final element in the target sequence approximates the argmax of the input sequence up to (but not including) the current element. Given the uniform distribution of argmax labels, this approach implies an expected accuracy of 90% or $1 - 1/L$, where $L = 10$ is the sequence length. Surprisingly, both the standard implicit model and implicitRNN outperform LSTMs and Transformers in this task. Moreover, the implicitRNN achieves higher accuracy than the standard implicit model, highlighting the potential benefits of a rolling latent representation for tasks requiring positional awareness within sequences.

A comparison of Transformer variants reveals that small Transformers may overfit to their positional encodings (PE) on simpler tasks. Notably, the Transformer without positional

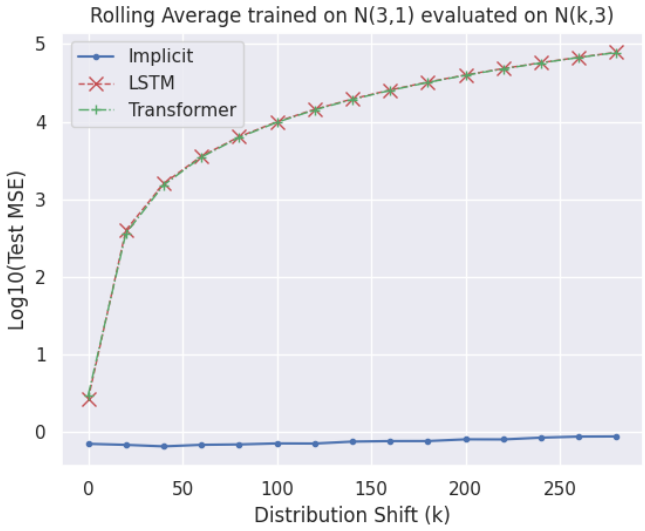

Figure 5.7: Training loss (MSE) for rolling average task. Implicit models maintain close to constant loss (↓) across shifts.

encodings, which lacks the ability to differentiate between sequence positions, performs better than its counterparts. This architecture consistently achieves accuracy close to the 90% benchmark, suggesting it effectively learns to always output the final value in the target sequence.

Overall, these results demonstrate the competitive performance of both the standard implicit model and implicitRNN, especially in scenarios requiring an understanding of relative positions within a sequence. The implicitRNN structure, which mirrors that of a vanilla RNN but replaces the recurrent cell with an implicit layer, demonstrates substantial performance improvements over LSTMs. This indicates the potential of the implicit layer to learn effective memory-gating mechanisms. Further experimentation with longer sequence lengths is necessary to confirm this conjecture. For additional model architectural details, refer to Table 5.5.

Oscillating Time Series Forecasting

In our prior experiments, we evaluated the performance of implicit models on well-defined mathematical tasks. Here, we transition to real-world noisy data, where the underlying functions are unknown. This shift allows us to investigate whether the extrapolation benefits of implicit models extend to more complex, real-world scenarios.

For the spiky time series forecasting task, the objective is to accurately capture sudden and short-lived distribution changes in the data. Such extreme events, commonly observed in

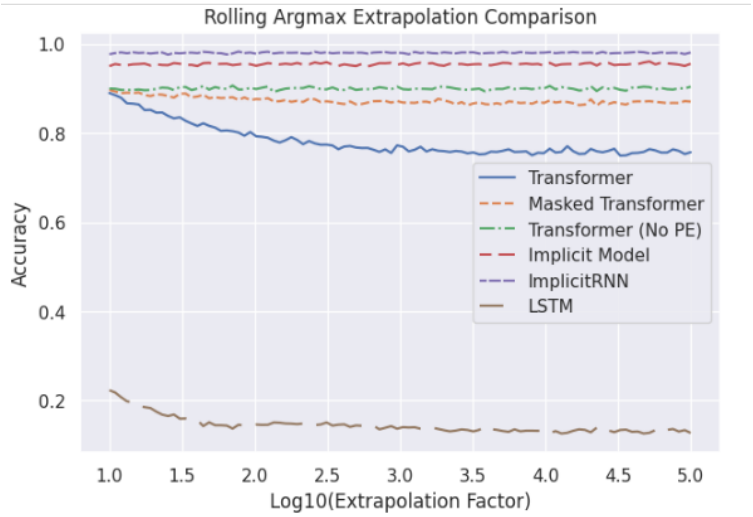

Figure 5.8: Test accuracy for rolling argmax task. ImplicitRNN and regular implicit model achieve the best results across shifts.

stock prices, sales volumes, or resource capacity predictions, challenge traditional forecasting models that struggle to extrapolate to unprecedented patterns. The ability to adapt effectively to these synthetic spikes has significant real-life implications.

Table 5.3 shows that the implicitRNN achieves a threefold reduction in test MSE compared to non-implicit models in the spiky time series task. Figure 5.9 further highlights the implicit model's ability to accurately predict the locations and magnitudes of these spikes. In contrast, both the transformer decoder and LSTM models frequently revert to predicting the average of the time series. This tendency to output mean values underscores the limitations of standard models in handling sharp fluctuations and complex time-series patterns.

Encouraged by these results, we applied implicit models to predict the sharp rise in AMC stock volatility observed in 2021. Volatility forecasting is particularly challenging due to abrupt changes in price behavior. We report the Mean Absolute Percentage Error (MAPE)[1] in Table 5.3. The implicit model demonstrates superior performance, outperforming other baselines by a factor of 1.67, further solidifying its ability to extrapolate effectively in real-world noisy data scenarios.

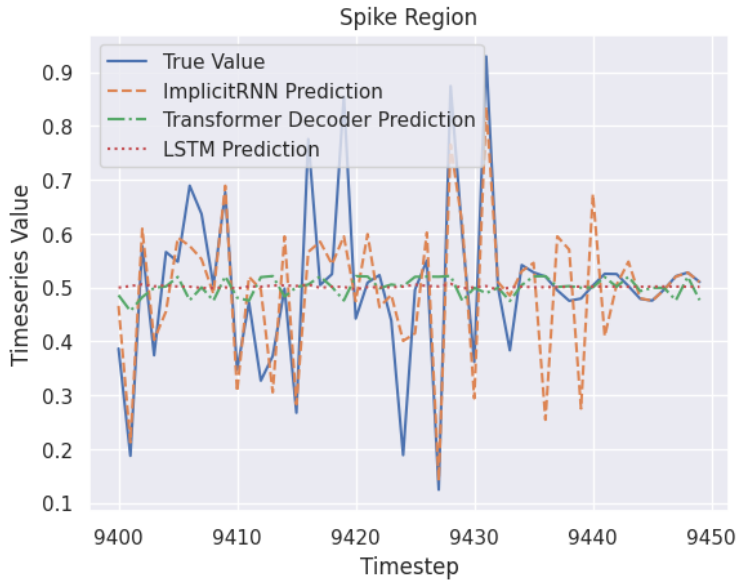

Figure 5.9: The implicitRNN most accurately models spike magnitudes.

Table 5.3: Train and test metrics (↓) in forecasting time series with sudden changes for synthetic (spiky time series) and real-world data (AMC stock volatility). Our architectures vary between tasks (see Table 5.5) but the implicit model outperforms all fine-tuned models.

	Spiky Data (MSE)		**AMC Stock (MAPE)**	
Model	Train	Test	Train	Test
Transformer	0.061	0.012	12.2	6.51
ImplicitRNN	0.015	**0.004**	**2.61**	**1.71**
LSTM	**0.011**	0.011	10.5	5.46
MLP	-	-	5.51	2.87
Linear regression	-	-	7.19	3.94

Earthquake Location Prediction

To generate samples of seismic waves between specific longitudes, based on the methods presented by Chuang et al., we used a 1D velocity model called Ak135 from the Python library obspy.taup. Obspy is a Python framework used to process seismological data. Kennett,

[1]The Mean Absolute Percentage Error (MAPE) metric evaluates a model's capability to predict changes in magnitudes relative to their sizes. While the Root Mean Square Error (RMSE) is higher for the implicit model on validation data (train RMSE = 0.001784, validation RMSE = 0.266366), the validation MAPE is lower because the average volatility in the validation set is orders of magnitude larger.

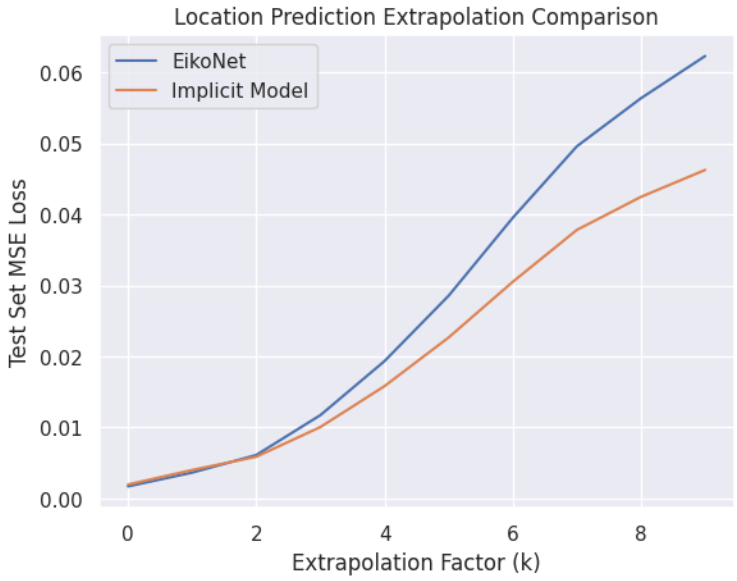

Figure 5.10: Extrapolation comparison between EikoNet and implicit model on the location prediction task as the extrapolation factor increases. The implicit model has the general edge in terms of MSE loss.

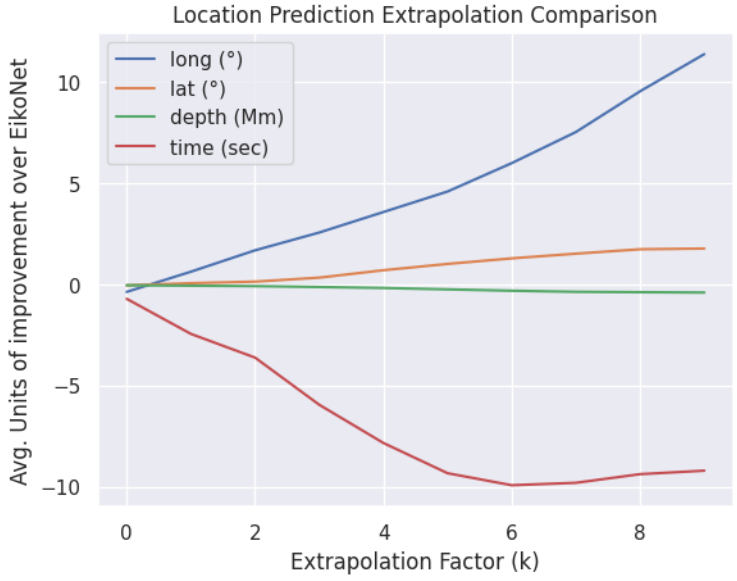

Figure 5.11: Breaking down the prediction values, we observe that the implicit model only outperforms EikoNet in longitude and latitude predictions.

Engdahl, and Buland demonstrate the accuracy of this model compared to real-world data (see specifically Figure 6). A 1D velocity model assumes the P-wave travel time (the duration the P-wave takes to travel from point A to point B) only depends on two attributes: the distance between the source and the receiver station and the depth of the source. We use this model to create a travel time lookup table based on these two attributes. We then generate source locations from a mesh that spans the entire globe while adding perturbation to each latitude and longitude pair. We generate station locations using the source-station distances we have from the lookup table and place the stations in random orientations (azimuths) from the source.

In the earthquake location prediction task, our implicit model demonstrates an improvement of 0.25e−3 in the in-distribution test loss compared to EikoNet, a model specifically designed for seismic data. As shown in Figure 5.10, the performance of the implicit model progressively improves in terms of extrapolated test MSE as k increases. By the time k reaches 2, the implicit model surpasses EikoNet, and at $k = 9$, the implicit model's test loss is 1.59e−2 lower than that of EikoNet. This improvement corresponds to an average enhancement of 11° in longitude and 2° in latitude.

Further refinement may be achieved by limiting the source latitude along with longitude during training, potentially leading to greater improvements in latitude extrapolation. However, as depicted in Figure 5.11, the implicit model faces greater difficulty in predicting time and depth, with performance deteriorating as k increases. Specifically, at $k = 9$, the implicit model's average error worsens by 9.2 seconds for time prediction and 409 km for depth prediction.

Future research should investigate whether training an implicit model exclusively on time and depth labels could enhance its performance in these aspects. This two-pass approach would parallel traditional location prediction methods, such as HYPOINVERSE (USGS), where constraints on depth and time present particularly challenging problems.

5.4 Depth Adaptability & Feedback Loop

In our analysis, we identify two key properties that contribute to the effective extrapolation capabilities of implicit models, even with limited datasets. The first property, *depth adaptability*, allows these models to dynamically adjust their effective depth rather than being constrained to a fixed number of layers. The second property, *closed-loop feedback*, enables inputs to revisit the same nodes during a single pass, enhancing learning and adaptability.

Depth adaptability. Figures 5.12 and 5.13 illustrate the concept of depth adaptability in implicit models, showcasing their dynamic ability to adjust the number of iterations required for convergence based on the complexity of the input data.

Figure 5.12 highlights the relationship between training accuracy and the number of

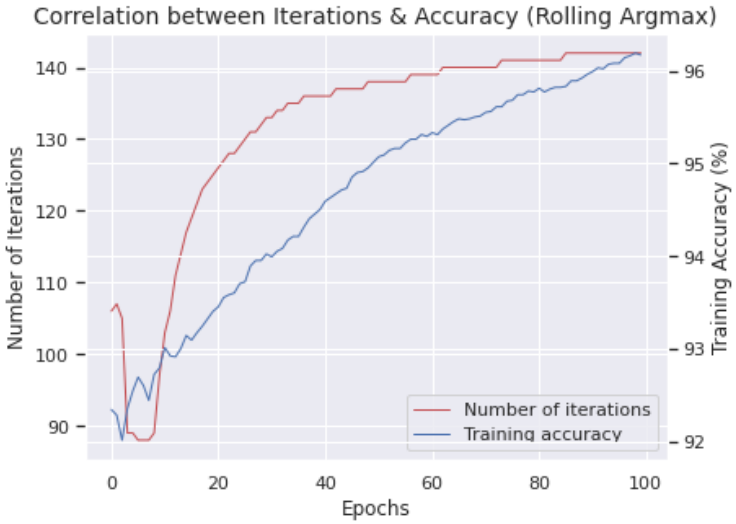

Figure 5.12: Relationship between model performance during training and the number of iterations.

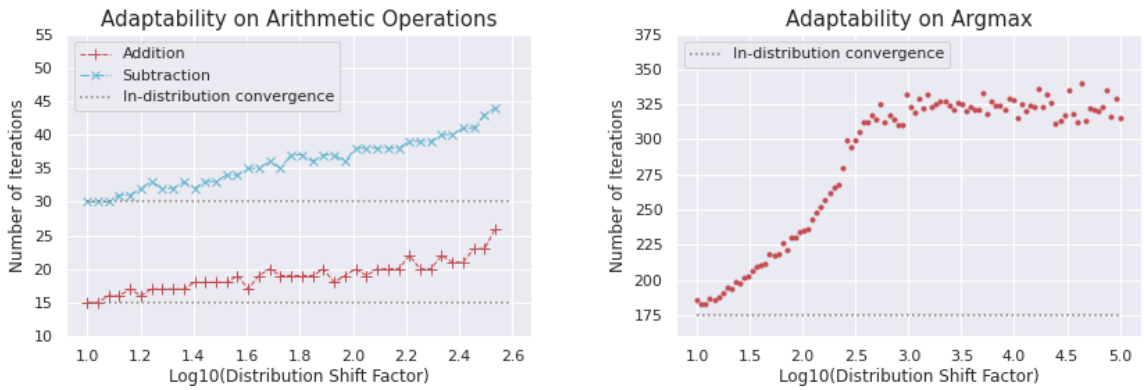

Figure 5.13: Growth of implicit models as the input complexity increases in the arithmetic tasks and rolling argmax.

iterations for the rolling argmax task. As training progresses, both metrics improve, demonstrating how implicit models refine their representations through iterative transformations. This behavior underscores the inherent adaptability of implicit models during the training process, where higher iterations correspond to better accuracy, particularly for complex tasks.

Figure 5.13 expands on this concept by showing how the number of iterations grows with the complexity of input data for arithmetic operations (left) and the rolling argmax task (right). For arithmetic tasks like addition and subtraction, the iteration count increases as the distribution shift factor grows, reflecting the model's need for more transformations to handle

out-of-distribution (OOD) inputs. Similarly, in the rolling argmax task, the distribution shift factor correlates with the required iterations, emphasizing the model's ability to adapt dynamically when encountering increasingly complex data.

During a forward pass, an implicit model concludes either by converging to a fixed point x^* or reaching a predefined iteration limit. The required number of iterations serves as an indicator of the model's perceived task complexity. On average, implicit models required approximately 15 iterations for addition, 30 for subtraction, and 175 for the rolling argmax task. For in-distribution inputs, iterations stabilized as the model familiarized itself with the parameter matrices. However, as inputs deviated further from the training distribution, the number of iterations increased, suggesting that the input U underwent more transformations via the model parameters.

This behavior aligns with the findings of Liang et al., highlighting how implicit models dynamically "grow" in depth to adapt their feature space to OOD inputs. This adaptability offers dual benefits: for in-distribution data, low depth minimizes overfitting, while for OOD data, higher depth reduces underfitting. The figures demonstrate the implicit model's capacity to balance these challenges effectively, leveraging its dynamic depth adaptability to achieve robust extrapolation.

Closed-loop feedback. Introduced by Wiener, the concept of closed-loop feedback refers to a system's ability to self-correct by using its outputs as part of the input for subsequent steps. As elaborated by Patten and Odum, this mechanism involves returning a portion of the output to influence the input at the next step. In neural networks, feedback is typically encoded through the backward pass during training. However, implicit models, as depicted in Figure 5.14, inherently incorporate feedback within a single iteration via the feedback connections in the parameter matrix A (specifically, its lower-triangular part).

Unlike standard feed-forward neural networks, where inputs move strictly forward layer by layer, implicit models allow inputs to revisit the same nodes or even move backward within a single iteration. This mechanism enables the model to correct itself iteratively from one layer to the next, enhancing its ability to refine intermediate states dynamically. In contrast, non-implicit models only correct themselves after completing a full pass through all layers. Ma, Tsao, and Shum emphasized the importance of closed-loop feedback in implicit models, drawing parallels to human neural networks where feedback loops are fundamental. To assess the role of closed-loop feedback in extrapolation, we conducted ablation studies on the lower-triangular part of the A matrix. Specifically, we considered a simplified implicit model with a strictly upper-triangular A matrix, effectively removing feedback. At time $t+1$, the iteration equation for such a model becomes $x^{t+1} = \phi(Ax^t + Bu)$. Ignoring the activation ϕ and Bu terms, this formulation highlights how x^t directly carries outputs from the previous iteration without any feedback correction. By contrasting this setup with the typical behavior of implicit models, we illustrate the critical role of closed-loop feedback in improving extrapolation capabilities.

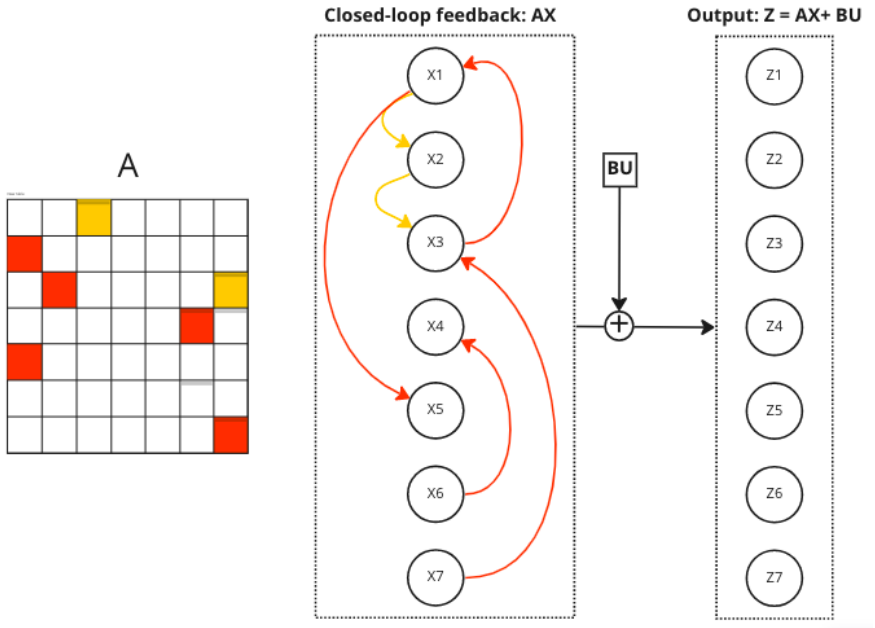

Figure 5.14: A matrix and node diagram representation of one iteration through the implicit model: $Ax + Bu$. In red, we have the weights that induce feedback in A, and in yellow, the non-feedback weights.

$$x^{t+1} \approx Ax^t = \begin{pmatrix} x_0^{t+1} \\ x_1^{t+1} \\ x_2^{t+1} \end{pmatrix} = \begin{pmatrix} \star & W_0 & W_1 \\ \star & \star & W_2 \\ \star & \star & \star \end{pmatrix} \begin{pmatrix} x_0^t \\ x_1^t \\ x_2^t \end{pmatrix} \tag{5.1}$$
$$= \begin{pmatrix} \star \cdot x_0^t + W_0 x_1^t + W_1 x_2^t \\ \star \cdot x_0^t + \star \cdot x_1^t + W_2 x_2^t \\ \star \cdot x_0^t + \star \cdot x_1^t + \star \cdot x_2^t \end{pmatrix}.$$

Closed-loop feedback corresponds to x_i^t being used to generate x_i^{t+1}. Equation 5.1 illustrates how the $\star$ weights encode this feedback mechanism. Specifically, the i-th state at time step $t+1$, denoted as x_i^{t+1}, depends not only on its past output x_i^t directly through weights on the diagonal but also indirectly through weights in the lower triangular part of the matrix. For example, x_2^{t+1} depends not only on x_2^t but also on x_1^t due to the lower triangular $\star$ weights. Furthermore, both x_1^t and x_2^t depend on x_0^{t-1}, highlighting the chain of dependencies facilitated by the feedback mechanism.

We compared a standard implicit model (with feedback loops) against an ablated implicit model (without feedback loops), where the upper-triangularity of A was enforced during training. The standard implicit models correspond to those used in our experiments, as described in Section 5.2. The results of the mathematical tasks for implicit models with and

without feedback are presented in Table 5.4. It is evident that feedback loops play a crucial role in helping the models achieve significantly lower testing loss, particularly on inputs with distribution shifts.

Figure 5.15 presents the ablation results for both models across distribution shifts in arithmetic and rolling sequential tasks. Ablating feedback harms model performance for subtraction but not for addition. The regular model, with its feedback loops, has twice as many weights, which could potentially lead to overfitting on simpler tasks. Feedback loops appear to enhance stability under distribution shifts for rolling average tasks, whereas they only provide better performance in calculating the rolling argmax of a sequence.

Moreover, besides superior overall performance, the presence of closed-loop feedback makes the model more resistant to distribution shifts. Notably, its loss increases more gradually in the subtraction and rolling average experiments. Our analysis demonstrates that implicit models possess the capability to adapt their architecture dynamically by learning the necessary depth, node connections, and self-corrections based on past predictions within a single training iteration, thanks to their closed-loop feedback mechanism.

Task	**Feedback**	**No feedback**
Add.	3.06	4.07
Sub.	0.953	1.80
R. Max.	93.7%	91.8%
R. Avg.	1e−4	1e−3

Table 5.4: Testing metrics (MSE ↓ for the arithmetic operations and rolling average, accuracy ↑ for rolling argmax) for a distribution shift of 100 comparing an implicit model trained with and without closed-loop feedback.

5.5 Conclusion

This chapter explored the remarkable extrapolation capabilities of implicit models, showcasing their potential as robust alternatives to standard neural networks for out-of-distribution tasks. Through extensive experiments on mathematical operations, noisy real-world datasets, and sequential forecasting, we demonstrated the superiority of implicit models in handling temporal, geographical, and distributional shifts.

Two key properties—*depth adaptability* and *closed-loop feedback*—emerged as the primary drivers of this performance. Depth adaptability allows implicit models to dynamically adjust their depth to suit the complexity of inputs, while closed-loop feedback enables self-correction within a single iteration, enhancing their ability to refine intermediate representations. These characteristics help implicit models outperform non-implicit baselines across various tasks, including arithmetic operations, rolling functions, and earthquake location prediction.

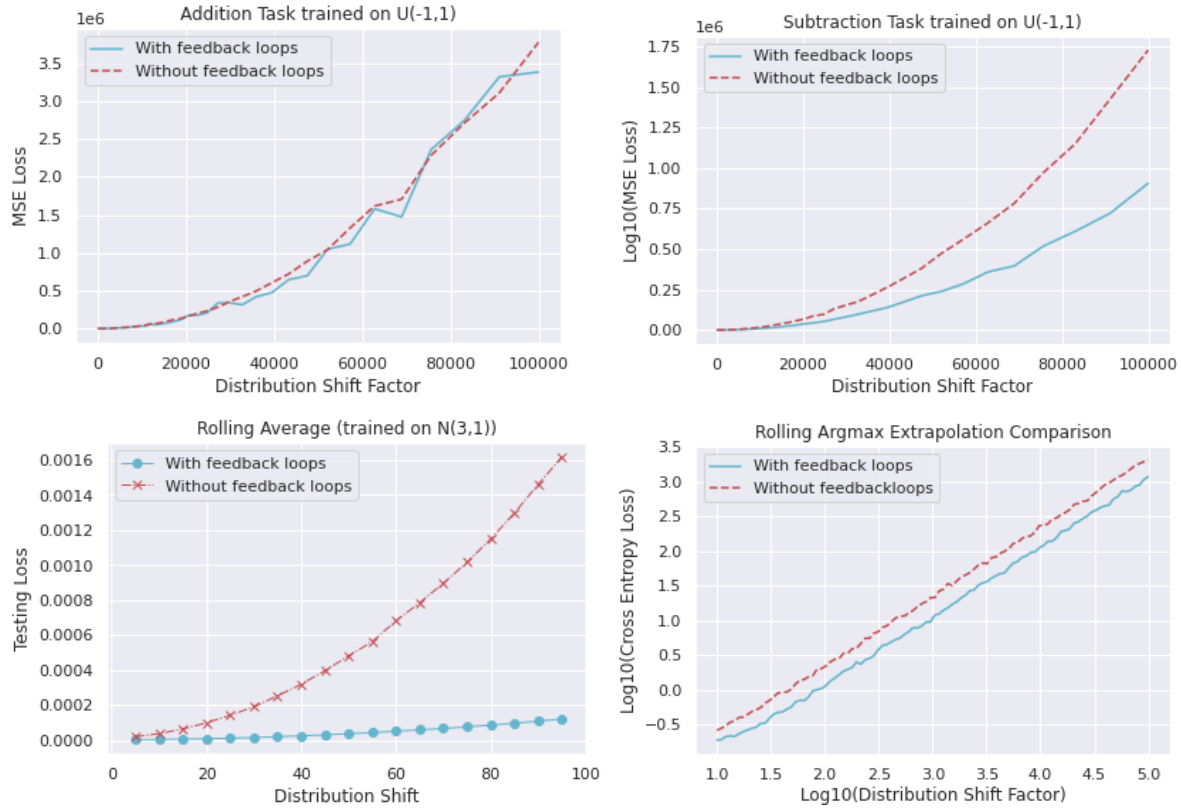

Figure 5.15: Implicit models benefit from closed-loop feedback specifically in harder extrapolation tasks.

Additionally, the flexibility of implicit models reduces reliance on task-specific architectural engineering, allowing them to generalize effectively with minimal data and training. Their capacity to integrate feedback loops further enhances their robustness, particularly in scenarios where data exhibits significant shifts or non-stationarity.

The findings in this chapter highlight the transformative potential of implicit models for tasks requiring generalization and robust extrapolation. This work lays the foundation for further research into their applications in other domains and motivates the development of training strategies that fully exploit their inherent strengths.

Task	Baseline model	Implicit models
Identity Function	**MLP**: $10 \times 9 \times 9 \times 10$	**Regular**: $A \in \mathbb{R}^{4\times4}, B \in \mathbb{R}^{4\times10}, C \in \mathbb{R}^{10\times4}, D \in \mathbb{R}^{10\times10}$
Arithmetic Operations	**MLP**: $50 \times 10 \times 10 \times 1$ **NALU**: $50 \times 10 \times 10 \times 1$	**Regular**: $A \in \mathbb{R}^{20\times20}, B \in \mathbb{R}^{20\times50}, C \in \mathbb{R}^{1\times20}, D \in \mathbb{R}^{1\times50}$
Rolling Average	**LSTM**: 1×18×18×1	**Regular**: $A \in \mathbb{R}^{32\times32}, B \in \mathbb{R}^{32\times10}, C \in \mathbb{R}^{10\times32}, D \in \mathbb{R}^{10\times10}$
Rolling Argmax	**LSTM**: 1×21×21×10	**Regular**: $A \in \mathbb{R}^{36\times36}, B \in \mathbb{R}^{36\times10}, C \in \mathbb{R}^{10\times36}, D \in \mathbb{R}^{10\times10}$ **ImplicitRNN**: $A \in \mathbb{R}^{21\times21}, B \in \mathbb{R}^{21\times23}, C \in \mathbb{R}^{22\times21}, D \in \mathbb{R}^{22\times23}$
Spiky Time Series	**LSTM**: 1×20×20×1	**ImplicitRNN**: $A \in \mathbb{R}^{20\times20}, B \in \mathbb{R}^{20\times21}, C \in \mathbb{R}^{20\times20}, D \in \mathbb{R}^{20\times21}$ with a 20×1 linear layer
Volatility Prediction	**LSTM**: $1 \times 38 \times 38 \times 1$ **SGD Linear Regression** **MLP**: $60 \times 50 \times 27 \times 27 \times 27 \times 10 \times 1$	**Regular**: $A \in \mathbb{R}^{53\times53}, B \in \mathbb{R}^{53\times60}, C \in \mathbb{R}^{1\times53}, D \in \mathbb{R}^{1\times60}$ **RNN**: $A \in \mathbb{R}^{37\times37}, B \in \mathbb{R}^{37\times41}, C \in \mathbb{R}^{40\times37}, D \in \mathbb{R}^{40\times41}$ with a 40×1 linear layer
Earthquake Prediction	**EikoNet**: 270×32×128×128×128×32×4	**Regular**: $A \in \mathbb{R}^{190\times190}, B \in \mathbb{R}^{190\times270}, C \in \mathbb{R}^{4\times190}, D \in \mathbb{R}^{4\times270}$

Task	Transformers
Identity Function	**Encoder-decoder**: $10 \times 10 \times 5$, 5 attention heads
Arithmetic Operations	**Sequential encoder**: 1 layer, 10 attention heads, feedforward dim 50 - processes each array as a single sequence. **Depth-wise encoder**: 1 layer, 1 attention head, feedforward dim 500, max PE length 50 - processes each element in a given array as a single sequence.
Rolling Average	**Encoder-decoder**: $10 \times 10 \times 5 \times 10$, 5 attention heads
Rolling Argmax	**Masked encoder-decoder**: 1 encoder layer, 1 decoder layer, 2 attention heads, feedforward dim 10, max PE length 10 **Unmasked encoder-decoder**: 1 encoder layer, 1 decoder layer, 2 attention heads, feedforward dim 10, max PE length 10 **Unmasked encoder-decoder without PE**: 1 encoder layer, 1 decoder layer, 2 attention heads, feedforward dim 10
Spiky Time Series	**Masked decoder**: 1x10 linear layer (expansion) → masked decoder (1 layer, 2 attention heads, feedforward dim 2048, max PE length 10) → 10x1 linear layer (contraction)
Volatility Prediction	**Sequential encoder**: 1 layer, 1 attention head, feedforward dim 2048, max PE length 60 → 60x1 linear layer
Earthquake Prediction	N/A

Table 5.5: Details of the explicit and implicit network architectures used in the experiments.

Bibliography

[1] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. "A Learning Algorithm for Boltzmann Machines". In: *Cognitive Science* 9.1 (1985), pp. 147–169.

[2] Brandon Amos and J. Zico Kolter. "OptNet: Differentiable Optimization as a Layer in Neural Networks". In: *International Conference on Machine Learning (ICML)*. 2017, pp. 136–145.

[3] Cem Anil et al. "Path Independent Equilibrium Models Can Better Exploit Test-Time Computation". In: *Advances in Neural Information Processing Systems (NeurIPS 2022)*. 2022. URL: https://arxiv.org/abs/2211.09961.

[4] MOSEK ApS. *The MOSEK Optimizer API for Python 9.3.21*. 2022. URL: https://docs.mosek.com/latest/pythonapi/index.html.

[5] Amir H Ashouri, Tarek S Abdelrahman, and Alwyn Dos Remedios. "Fast on-the-fly retraining-free sparsification of convolutional neural networks". In: *arXiv preprint arXiv:1811.04199* (2018).

[6] 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. URL: http://proceedings.mlr.press/v80/athalye18a.html.

[7] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "Deep equilibrium models". In: *Advances in Neural Information Processing Systems* 32 (2019).

[8] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. "Deep Equilibrium Models". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/01386bd6d8e091c2ab4c7c7de644d37b-Paper.pdf.

[9] Shaojie Bai, Vladlen Koltun, and J Zico Kolter. "Multiscale deep equilibrium models". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5238–5250.

[10] Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. "Multiscale Deep Equilibrium Models". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 5238–5250. URL: https://proceedings.neurips.cc/paper/2020/file/3812f9a59b634c2a9c574610eaba5bed-Paper.pdf.

[11] Matteo Cacciola et al. "Deep Neural Networks pruning via the Structured Perspective Regularization". In: *arXiv:2206.14056 [cs.LG]* (June 2022). URL: https://arxiv.org/pdf/2206.14056.pdf.

[12] 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. DOI: 10.1145/3128572.3140444. URL: https://doi.org/10.1145/3128572.3140444.

[13] Nicholas Carlini and David A. Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 39–57. DOI: 10.1109/SP.2017.49. URL: https://doi.org/10.1109/SP.2017.49.

[14] François Charton. *What is my math transformer doing? – Three results on interpretability and generalization*. 2022. DOI: 10.48550/ARXIV.2211.00170. URL: https://arxiv.org/abs/2211.00170.

[15] Kaiyu Chen et al. "Neural Arithmetic Expression Calculator". In: *CoRR* abs/1809.08590 (2018). arXiv: 1809.08590. URL: http://arxiv.org/abs/1809.08590.

[16] Ricky T. Q. Chen et al. "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

[17] Ricky TQ Chen et al. "Neural ordinary differential equations". In: *Advances in neural information processing systems* 31 (2018).

[18] Shaobing Chen and David Donoho. "Basis pursuit". In: *Proceedings of 1994 28th Asilomar Conference on Signals, Systems and Computers*. Vol. 1. IEEE. 1994, pp. 41–44.

[19] Lindsay Y. Chuang et al. "Deep Learning and Masksembles for Sparse Network Earthquake Location and Uncertainty Estimation". Manuscript in preparation. 2023.

[20] Misha Denil et al. "Predicting parameters in deep learning". In: *Advances in neural information processing systems* 26 (2013).

[21] Terrance Devries and Graham W. Taylor. "Improved Regularization of Convolutional Neural Networks with Cutout". In: *CoRR* abs/1708.04552 (2017). arXiv: 1708.04552. URL: http://arxiv.org/abs/1708.04552.

[22] Emilien Dupont, David Duvenaud, and Andrew Gordon Wilson. "Augmented Neural ODEs". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019, pp. 3133–3144.

[23] Laurent El Ghaoui et al. "Implicit Deep Learning". In: *SIAM Journal on Mathematics of Data Science* 3.3 (2021), pp. 930–958. DOI: 10.1137/20M1358517. eprint: https://doi.org/10.1137/20M1358517. URL: https://doi.org/10.1137/20M1358517.

[24] Laurent El Ghaoui et al. "Implicit Deep Learning". In: *SIAM Journal on Mathematics of Data Science* 3.3 (2021), pp. 930–958. DOI: 10.1137/20M1358517. eprint: https://doi.org/10.1137/20M1358517. URL: https://doi.org/10.1137/20M1358517.

[25] Samuel F. Fung and Jerome Darbon. "Implicit Networks and Convergence Guarantees for Learning". In: *International Conference on Machine Learning (ICML)*. 2021, pp. 456–468.

[26] 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).

[27] Tianxiang Gao and Hongyang Gao. "On the optimization and generalization of overparameterized implicit neural networks". In: *arXiv preprint arXiv:2209.15562* (2022).

[28] Yash Garg and K Selçuk Candan. "iSparse: Output informed sparsification of neural network". In: *Proceedings of the 2020 International Conference on Multimedia Retrieval*. 2020, pp. 180–188.

[29] Guillaume Garrigos and Robert M Gower. "Handbook of convergence theorems for (stochastic) gradient methods". In: *arXiv preprint arXiv:2301.11235* (2023).

[30] Zhengyang Geng et al. "On training implicit models". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 24247–24260.

[31] Laurent El Ghaoui, Vivian Viallon, and Tarek Rabbani. "Safe feature elimination for the lasso and sparse supervised learning problems". In: *arXiv preprint arXiv:1009.4219* (2010).

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[33] 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.

[34] Ian J. Goodfellow and Oriol Vinyals. "Qualitatively characterizing neural network optimization problems". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6544.

[35] 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.

[36] Sven Gowal et al. "On the effectiveness of interval bound propagation for training verifiably robust models". In: *arXiv preprint arXiv:1810.12715* (2018).

[37] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines". In: *CoRR* abs/1410.5401 (2014). arXiv: 1410.5401. URL: http://arxiv.org/abs/1410.5401.

[38] Alex Graves et al. "Hybrid computing using a neural network with dynamic external memory". In: *Nature.* 538.7626 (2016-10). ISSN: 0028-0836.

[39] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. "Hamiltonian Neural Networks". In: *arXiv preprint arXiv:1906.01563* (2019).

[40] Fangda Gu et al. "Implicit Graph Neural Networks". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.

[41] Fangda Gu et al. "Implicit graph neural networks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 11984–11995.

[42] Eldad Haber and Lars Ruthotto. "Stable Architectures for Deep Neural Networks". In: *Inverse Problems* 34.1 (2017), p. 014004.

[43] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2016, pp. 770–778.

[44] 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.

[45] Kaiming He et al. "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[46] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18.7 (2006), pp. 1527–1554.

[47] Geoffrey E. Hinton and Terrence J. Sejnowski. "Optimal perceptual inference". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 1983, pp. 448–453.

[48] Sepp Hochreiter and Jürgen Schmidhuber. "LSTM can Solve Hard Long Time Lag Problems". In: *Advances in Neural Information Processing Systems*. Ed. by M.C. Mozer, M. Jordan, and T. Petsche. Vol. 9. MIT Press, 1996. URL: https://proceedings.neurips.cc/paper_files/paper/1996/file/a4d2f0d23dcc84ce983ff9157f8b7f88-Paper.pdf.

[49] Yiming Hu et al. "Multi-Loss-Aware Channel Pruning of Deep Networks". In: *2019 IEEE International Conference on Image Processing (ICIP)* (2019), pp. 889–893.

[50] Lingyi Huang, Zhengang Xu, and Mihai Anitescu. "Understanding Implicit Models in Scientific Machine Learning". In: *Journal of Computational Physics* 464 (2022), p. 111363.

[51] Zehao Huang and Naiyan Wang. "Data-Driven Sparse Structure Selection for Deep Neural Networks". In: *ECCV* (2018).

[52] Łukasz Kaiser and Ilya Sutskever. *Neural GPUs Learn Algorithms*. 2015. DOI: 10.48550/ARXIV.1511.08228. URL: https://arxiv.org/abs/1511.08228.

[53] Hamed Karimi, Julie Nutini, and Mark Schmidt. "Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I 16*. Springer. 2016, pp. 795–811.

[54] Jacob Kelly et al. *Learning Differential Equations that are Easy to Solve*. 2020. arXiv: 2007.04504 [cs.LG].

[55] B. L. N. Kennett, E. R. Engdahl, and R. Buland. "Constraints on seismic velocities in the Earth from traveltimes". In: *Geophysical Journal International* 122.1 (July 1995), pp. 108–124. ISSN: 0956-540X. DOI: 10.1111/j.1365-246X.1995.tb03540.x. eprint: https://academic.oup.com/gji/article-pdf/122/1/108/1543667/122-1-108.pdf. URL: https://doi.org/10.1111/j.1365-246X.1995.tb03540.x.

[56] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: 2009.

[57] 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.

[58] Condat Laurent. "Fast projection onto the simplex and the l1 ball". In: *Math. Prog* 158 (2016), pp. 575–585.

[59] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[60] Kaiqu Liang et al. "Out-of-Distribution Generalization with Deep Equilibrium Models". In: *ICML 2021 Workshop on Uncertainty and Robustness in Deep Learning* (2021).

[61] Renjie Liao et al. "Reviving and Improving Recurrent Backpropagation". In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1188–1196.

[62] 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.

[63] Christos Louizos, Max Welling, and Diederik P Kingma. "Learning sparse neural networks through L_0 regularization". In: *arXiv preprint arXiv:1712.01312* (2017).

[64] Yi Ma, Doris Tsao, and Heung-Yeung Shum. "On the principles of Parsimony and Self-consistency for the emergence of intelligence". In: *Frontiers of Information Technology & Electronic Engineering* (2022), pp. 1–26.

[65] 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.

[66] Carl D Meyer. *Matrix analysis and applied linear algebra*. Vol. 71. Siam, 2000.

[67] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. "Investigating the Limitations of the Transformers with Simple Arithmetic Tasks". In: *CoRR* abs/2102.13019 (2021). arXiv: 2102.13019. URL: https://arxiv.org/abs/2102.13019.

[68] Michael R Osborne, Brett Presnell, and Berwin A Turlach. "A new approach to variable selection in least squares problems". In: *IMA journal of numerical analysis* 20.3 (2000), pp. 389–403.

[69] 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.

[70] 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.

[71] B. Patten and E. Odum. "The Cybernetic Nature of Ecosystems". In: 118.6 (Dec. 1981). DOI: https://doi.org/10.1086/283881.

[72] Hongwu Peng et al. "Towards sparsification of graph neural networks". In: *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE. 2022, pp. 272–279.

[73] Nicholas G Polson, James G Scott, and Brandon T Willard. "Proximal algorithms in statistics and machine learning". In: (2015).

[74] Christopher Rackauckas et al. *Universal Differential Equations for Scientific Machine Learning*. 2021. arXiv: 2001.04385 [cs.LG].

[75] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. "Semidefinite relaxations for certifying robustness to adversarial examples". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. 2018, pp. 10900–10910. URL: http://papers.nips.cc/paper/8285-semidefinite-relaxations-for-certifying-robustness-to-adversarial-examples.

[76] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[77] Zaccharie Ramzi et al. "Test like you Train in Implicit Deep Learning". In: (2023). arXiv: 2305.15042 [cs.LG].

[78] Benjamin Recht et al. "Do CIFAR-10 Classifiers Generalize to CIFAR-10?" In: *arXiv preprint arXiv:1805.12156* (2019).

[79] Sebastian Ruder. "An Overview of Gradient Descent Optimization Algorithms". In: *arXiv preprint arXiv:1609.04747* (2017).

[80] Omar M. Saad, Ali G. Hafez, and M. Sami Soliman. "Deep Learning Approach for Earthquake Parameters Classification in Earthquake Early Warning System". In: *IEEE Geoscience and Remote Sensing Letters* 18.7 (2021), pp. 1293–1297. DOI: 10.1109/LGRS.2020.2998580.

[81] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *ArXiv* abs/1910.01108 (2019).

[82] Shankar Sastry. *Nonlinear systems: analysis, stability, and control*. Vol. 10. Springer Science & Business Media, 2013.

[83] Simone Scardapane et al. "Group sparse regularization for deep neural networks". In: *Neurocomputing* 241 (2017), pp. 81–89.

[84] Daniel Schlör, Markus Ring, and Andreas Hotho. "iNALU: Improved Neural Arithmetic Logic Unit". In: *Frontiers in Artificial Intelligence* 3 (2020). ISSN: 2624-8212. DOI: 10.3389/frai.2020.00071. URL: https://www.frontiersin.org/articles/10.3389/frai.2020.00071.

[85] Avi Schwarzschild et al. "Can You Learn an Algorithm? Generalizing from Easy to Hard Problems with Recurrent Networks". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 6695–6706. URL: https://proceedings.neurips.cc/paper/2021/file/3501672ebc68a5524629080e3ef60aef-Paper.pdf.

[86] 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.

[87] Jonathan D. Smith, Kamyar Azizzadenesheli, and Zachary E. Ross. "EikoNet: Solving the Eikonal Equation With Deep Neural Networks". In: *IEEE Transactions on Geoscience and Remote Sensing* 59.12 (Dec. 2021), pp. 10685–10696. DOI: 10.1109/tgrs.2020.3039165. URL: https://doi.org/10.1109/TGRS.2020.3039165.

[88] Yi Sun, Xiaogang Wang, and Xiaoou Tang. "Sparsifying neural network connections for face recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4856–4864.

[89] Christian Szegedy et al. "Intriguing properties of neural networks". In: *International Conference on Learning Representations*. 2014. URL: http://arxiv.org/abs/1312.6199.

[90] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58.1 (1996), pp. 267–288.

[91] Kim-Chuan Toh and Sangwoon Yun. "An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems". In: *Pacific Journal of optimization* 6.615-640 (2010), p. 15.

[92] Andrew Trask et al. "Neural Arithmetic Logic Units". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/0e64a7b00c83e3d22ce6b3acf2c582b6-Paper.pdf.

[93] Alicia Y Tsai et al. "State-driven Implicit Modeling for Sparsity and Robustness in Neural Networks". In: *arXiv preprint arXiv:2209.09389* (2022).

[94] USGS. *HYPOINVERSE Earthquake Location*. Version 1.4. 2019. URL: https://www.usgs.gov/software/hypoinverse-earthquake-location.

[95] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[96] Li Wan et al. "Regularization of neural networks using dropconnect". In: *International conference on machine learning*. PMLR. 2013, pp. 1058–1066.

[97] Cunxiang Wang et al. "Exploring Generalization Ability of Pretrained Language Models on Arithmetic and Logical Reasoning". In: *Natural Language Processing and Chinese Computing*. Ed. by Lu Wang et al. Cham: Springer International Publishing, 2021, pp. 758–769. ISBN: 978-3-030-88480-2.

[98] Yun Wang. "Feature screening for the lasso". PhD thesis. Princeton University, 2015.

[99] Taylor W. Webb et al. "Learning Representations that Support Extrapolation". In: *CoRR* abs/2007.05059 (2020). arXiv: 2007.05059. URL: https://arxiv.org/abs/2007.05059.

[100] Jason Wei et al. "Emergent Abilities of Large Language Models". In: *Transactions on Machine Learning Research* (2022). Survey Certification. ISSN: 2835-8856. URL: https://openreview.net/forum?id=yzkSU5zdwD.

[101] N. Wiener. "Cybernetics." In: *MIT Press, Cambridge, Mass* (1948).

[102] Yongtao Wu et al. "Extrapolation and Spectral Bias of Neural Nets with Hadamard Product: a Polynomial Net Study". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=_cXUMAnWJJj.

[103] Keyulu Xu et al. "How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks". In: (2021). arXiv: 2009.11848 [cs.LG].

[104] Fuxun Yu et al. "Interpreting and Evaluating Neural Network Robustness". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. Ed. by Sarit Kraus. ijcai.org, 2019, pp. 4199–4205. DOI: 10.24963/ijcai.2019/583. URL: https://doi.org/10.24963/ijcai.2019/583.

[105] Hongyang Zhang et al. "Theoretically Principled Trade-off between Robustness and Accuracy". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 7472–7482. URL: http://proceedings.mlr.press/v97/zhang19p.html.