DISSERTATION

# CRACKING OPEN THE BLACK BOX: A GEOMETRIC AND TOPOLOGICAL ANALYSIS OF NEURAL NETWORKS

Submitted by

Christina Cole

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2024

Doctoral Committee:

Advisor: Michael Kirby
Co-Advisor: Chris Peterson

Margaret Cheney
Bruce Draper

# ABSTRACT

CRACKING OPEN THE BLACK BOX: A GEOMETRIC AND TOPOLOGICAL ANALYSIS OF NEURAL NETWORKS

Deep learning is a subfield of machine learning that has exploded in recent years in terms of publications and commercial consumption. Despite their increasing prevalence in performing high-risk tasks, deep learning algorithms have outpaced our understanding of them. In this work, we hone in on neural networks, the backbone of deep learning, and reduce them to their scaffolding defined by polyhedral decompositions. With these decompositions explicitly defined for low-dimensional examples, we utilize novel visualization techniques to build a geometric and topological understanding of them. From there, we develop methods of implicitly accessing neural networks' polyhedral skeletons, which provide substantial computational and memory savings compared to those requiring explicit access. While much of the related work using neural network polyhedral decompositions is limited to toy models and datasets, the savings provided by our method allow us to use state-of-the-art neural networks and datasets in our analyses. Our experiments alone demonstrate the viability of a polyhedral view of neural networks and our results show its usefulness. More specifically, we show that the geometry that a polyhedral decomposition imposes on its neural network's domain contains signals that distinguish between original and adversarial images. We conclude our work with suggested future directions. Therefore, we (1) contribute toward closing the gap between our use of neural networks and our understanding of them through geometric and topological analyses and (2) outline avenues for extensions upon this work.

# ACKNOWLEDGEMENTS

This work is supported by the United States Air Force under Contract No. FA865020C1121. In addition to thanking my entire committee for their role in making the research contained in these pages what it is, I would also like to thank Dr. Greg Arnold and Dr. Jared Culbertson for their frequent guidance and feedback. Their respective entities' involvement, that of Matrix Research Inc. and the Air Force Research Laboratory, have helped this research address questions that are relevant today. Additionally, the HPC resources of the Data Science Research Institute and thorough support provided by Ryan Job made many of the contained computations possible. I gratefully recognize my family, church community, friends, cat, and coffee for giving me pick-me-ups when they were needed. Lastly, I cannot fail to mention Dr. Raul Aparicio and Dr. Chris Peterson for their respective pivotal roles in my education journey. Thank you, Dr. Aparicio, for your energetic instruction of Calculus 3, which contributed to my choice to study mathematics over art, and Chris, for your personability and mathematical enthusiasm that contributed to my choice of Colorado State University for graduate school.

# DEDICATION

*With gratitude, I dedicate this dissertation to my husband, Garrett. To my late colleague and friend, Yajing, I also offer this dedication. Thank you both for always believing in me.*

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# Chapter 1

# Introduction

'Machine Learning' are two words that buzz in the technology world today. Many people outside of related fields may be familiar with the idea of machine learning either through (a) movies that depict the most advanced of its capabilities or ones that do not yet exist or (b) experience with existing products that leverage machine learning—language learning programs that adapt to student mistakes, athletic training software to prevent injury, the customization of media content based on viewing history, assisted sports betting technology, and self-driving cars—to name a few. One of the truths not conveyed in these movies and products about deep learning in particular, a subdiscipline of machine learning whose algorithms heavily rely on neural networks, is that experts in the field do not know why they break with relative ease. Despite this, the success of deep learning in a variety of tasks has led to a technological frenzy: the introduction of one neural network or training scheme is almost immediately outperformed by another that borrowed similar building blocks but made a couple of ad hoc tweaks that made it better in some capacity (often in memory/computational requirements or accuracy). These types of unexplained improvements have been praised in the field and leveraged in many products today. Historically, this has been acceptable. Historically, we have used deep learning in low-risk environments and for very specific tasks that do not require advanced features like robustness to adversarial attack or extrapolation to effectively handle new situations. After all, who would try to thwart your efforts to learn a new language? Or why would this technology need the ability to extrapolate? If you got question A wrong, maybe you need to revisit the material covered in question A. Deep learning's portfolio of applications has recently begun to shift to ones of high risk, thereby requiring the aforementioned advanced features. However, these applications have significantly outpaced the theory grounding them. Uncovering the mystery of their underlying mechanics—cracking open the black box—has become an increasingly daunting task. If we rely heavily on this technology, to the point of entrusting it with our money and even our lives, should we not understand how it works and how it can

be fooled? To this end, we strip neural networks down to their scaffolding and answer the questions, "What is a neural network and how does the learning process shape it?" First, we give some relevant background, both mathematically (in Chapter 2) and in the particular models and datasets we use (in Chapter 3). Then, we show in Chapter 4 that the foundational structure underlying every neural network is a polyhedral decomposition and that the training process is nothing more than a procedure to allocate its polyhedral and input-output Jacobian norm resources. We leverage this understanding in Chapter 5 to design algorithms that efficiently explore the underlying structure imposed by the polyhedral decomposition, algorithms which we use to identify differences between original and adversarial data. Next, we address some loose ends and interesting future directions of our work in Chapter 6 and conclude our work with Chapter 7.

**Portions of this work have been**

- presented at the Colorado State University Department of Mathematics' Data Science Seminar,
- presented at the 2022 Rocky Mountain Celebration of Women in Computing Conference,
- presented at the Colorado State University 2022 Graduate Student Showcase where it was recognized with the "College of Natural Sciences Outstanding Scholar Award",
- presented in the 2022 and 2023 DARPA Geometries of Learning (GoL) PI Meetings,
- published in 2022 IEEE International Conference on Big Data Workshop GTA3 [4],
- published in 2023 IEEE CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) [1], and
- published in 2023 International Conference on Machine Learning Topology, Algebra, and Geometry (TAG) Workshops [3].

# Chapter 2

# Mathematical Background

## 2.1 High-Dimensional Space is Counter-Intuitive

Because we live in a physically three-dimensional world, most of us do not have an intuition for high-dimensional spaces. In this section, we provide some facts about high-dimensional balls and spheres with the primary goal of encouraging the reader to abandon certain spatial intuitions that may be somewhat misleading in our exploration of shapes in $\mathbb{R}^N$, for some large $N$. A few mind-bending implications of the equations and theorems presented in this section are listed below.

- The volume of a hyperball is very small in large dimensions. For instance, the volume of the 1000-dimensional unit hyperball is roughly $1.5 \times 10^{-883}$.
- The volume of a 1000-dimensional hyperball with radius $R = 7.635$ has roughly the same volume as a 1000-dimensional unit hypercube. In Figure 2.1, we visualize these objects to scale in three dimensions to show how counter-intuitive this is.
- If you inscribe a 1000-dimensional hyperball of diameter 1 inside of a 1000-dimensional unit hypercube, a randomly chosen point from the hypercube also lies inside of the hyperball with a probability of roughly $1.4 \times 10^{-1184}$.
- If you center-inscribe a 1000-dimensional hyperball of radius 0.99 inside of a 1000-dimensional unit hyperball, a randomly chosen point from the unit hyperball also lies inside of the radius-.99 hyperball with a probability of roughly $4.3 \times 10^{-5}$.
- If you pick two random points on the $n$-dimensional unit hypersphere, then their expected dot product is $(n+1)^{-1/2}$. As a consequence, random high-dimensional vectors are expected to be very close to orthogonal.

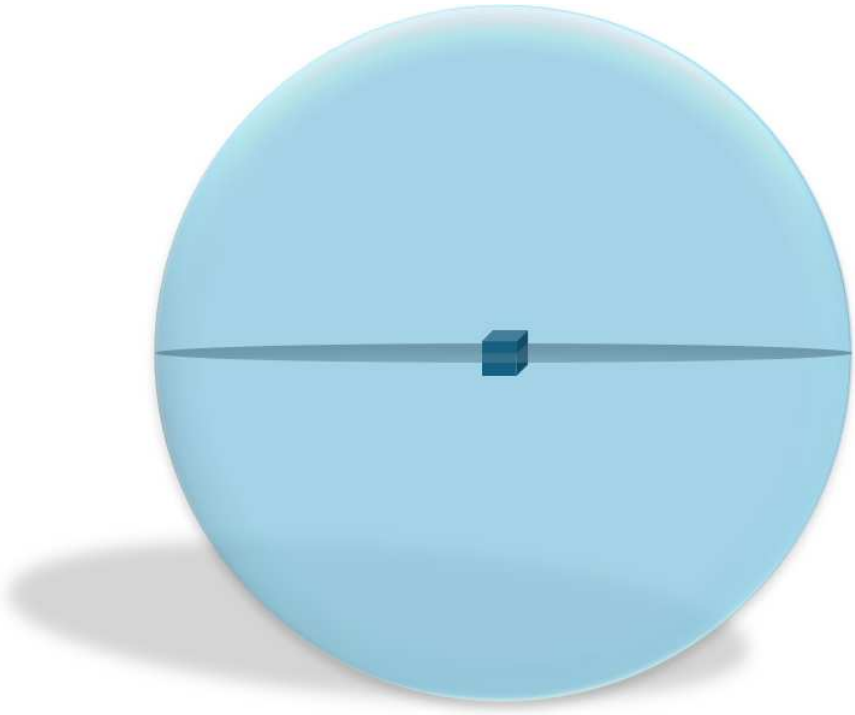


**Figure 2.1:** A unit cube nested inside of a sphere with radius R=7.635, to scale.

Beyond these being intriguing facts, we call back to some of them in Section 5.2. The last theorem we present about high-dimensional space can be leveraged toward a dimensionality reduction tool, tools covered further in Section 2.2.

### 2.1.1 Concentration of Volume

To establish some relevant terminology, we denote by $B(n,R)$ the $n$-dimensional hyperball (or $n$-ball) with radius $R$ centered at the origin (thus $B(n,R) = \{\vec{x} \in \mathbb{R}^n \ : \|\vec{x}\|_2 \leq R\}$). $B(n,R)$ has a volume of $V(n,R)$ where $V(n,R) = \frac{\pi^{n/2}}{\Gamma(n/2+1)}R^n$ with $\Gamma(\cdot)$ denoting the gamma function [5]. Note that if $n$ is an integer then $\Gamma(n+1) = n!$ and $\Gamma(n+1/2) = (n-1/2)(n-3/2)\cdots(1/2)\Gamma(1/2)$ and that $\Gamma(1/2) = \sqrt{\pi}$. This implies that after achieving a maximum at $n = 5$, $V(n,1)$ interestingly approaches zero as $n$ approaches infinity. After an arbitrary choice of north pole on the surface of the $n$-ball and letting $\vec{p}$ denote the vector from the $n$-ball center to this north pole, the $n$-ball's

*equator* $B_E(n,R)$ is the set

$$B_E(n,R) = \{\vec{x} \in B(n,R) \ : \ \vec{x} \cdot \vec{p} = 0\},$$

which is an $(n-1)$-dimensional ball. Let $B_{(a,b)}(n,R)$ denote the portion of $B(n,R)$ given by thickening the equator by $2b$ ($b$ in the direction of both $\vec{p}$ and -$\vec{p}$) and removing a core with a radius of $a$ from the center of this thickened equator. More formally, if we arbitrarily define $\vec{p} = [1,0,\ldots,0]^\top$, then

$$B_{(a,b)}(n,R) = \left\{\vec{x} \in B(n,R) \ : \ \left\|Proj_{\vec{p}}\vec{x}\right\|_2 \leq b \text{ and } R-a \leq \left\|[0,x_2,\ldots,x_n]^\top\right\|_2 \leq R\right\}$$

and call $B_{(a,b)}(n,R)$ the $(a,b)$-*annulus* of $B(n,R)$. Let $V_{(a,b)}(n,R)$ denote the volume of $B_{(a,b)}(n,R)$. Call $B_{(R,\delta)}(n,R)$ the ball's $\delta$-*thickened equator*. For illustrative purposes, consider a ball in three dimensions. The equation for the volume of its $\delta$-thickened equator is

$$V_{(R,\delta)}(3,R) = 2\pi \int_0^\delta R^2 - p^2 dp,$$

which is equal to the volume of the ball without either of its spherical caps with heights of $R-\delta$. As $n$ approaches infinity, $V_{(R,\delta)}(n,R)$ approaches $V(n,R)$. In other words, in high dimensions, a ball's $\delta$-thickened equator contains the majority of its volume. We state this more specifically in Theorem 2.1.1 whose statement and proof are adapted from those of Lemma 2.2 in [6]. Given an $n$-ball with radius $R$, its $\delta$-*thickened hypersphere* $B_{(\delta,R)}(n,R)$ is the portion of the ball lying within $\delta$ of its exterior. As $n$ approaches infinity, $V_{(\delta,R)}(n,R)$ approaches $V(n,R)$, or the volume of a hyperball concentrates about its surface with the increase of dimension, as stated in Theorem 2.1.2 whose statement and proof are formalized from discussion in Section 2.2.4 of [6].

**Theorem 2.1.1.** *For $n \geq 3$ and $\delta = \frac{c}{\sqrt{n-1}}$, $c \geq 1$, the $\delta$-thickened equator of the n-dimensional unit ball contains at least a $1 - \frac{2}{c}e^{-c^2/2}$ fraction of its total volume; i.e.,*

$$1 - \frac{2}{c}e^{-c^2/2} \leq \frac{V_{(1,\delta)}(n,1)}{V(n,1)}.$$

*Proof.* We have that

$$\begin{aligned} & V_{(1,\delta)}(n,1) + \big(V(n,1) \backslash V_{(1,\delta)}(n,1)\big) && = V(n,1) \\ \Rightarrow \quad & V_{(1,\delta)}(n,1) && = V(n,1) - \big(V(n,1) \backslash V_{(1,\delta)}(n,1)\big) \\ \Rightarrow \quad & \tfrac{V_{(1,\delta)}(n,1)}{V(n,1)} && = 1 - \tfrac{V(n,1) \backslash V_{(1,\delta)}(n,1)}{V(n,1)}. \end{aligned}$$

Therefore, it remains to show that $\frac{2}{c}e^{-c^2/2}$ is an upper bound for $\frac{V(n,1) \backslash V_{(1,\delta)}(n,1)}{V(n,1)}$, the fraction of the volume of $B(n,1)$ lying outside of its $\delta$-thickened equator with $\delta = \frac{c}{\sqrt{n-1}}$, $\forall c \geq 1$. By symmetry, it is sufficient to show that $\frac{2}{c}e^{-c^2/2}$ is an upper bound for the fraction of the volume of $H(n,1)$, the northern hemisphere of $B(n,1)$, satisfying $\{\vec{x} \in H(n,1) \ : \ Proj_{\vec{p}}\vec{x} > \delta\} \coloneqq T_{(\delta)}(n,1)$; i.e., that $\mathrm{Vol}\big(T_{(\delta)}(n,1)\big)/\mathrm{Vol}\big(H(n,1)\big) \leq \frac{2}{c}e^{-c^2/2}$ where $\mathrm{Vol}(x)$ denotes the volume of $x$. We do this by showing that $\mathrm{Vol}\big(T_{(\delta)}(n,1)\big) \leq \alpha$ and $\mathrm{Vol}\big(H(n,1)\big) \geq \beta$ where $\frac{\alpha}{\beta} = \frac{2}{c}e^{-c^2/2}$.

Find $\alpha$ such that $\mathrm{Vol}\big(T_{(\delta)}(n,1)\big) \leq \alpha$.

$$\begin{aligned} \mathrm{Vol}\big(T_{(\delta)}(n,1)\big) &= V(n-1,1)\int_{\delta}^{1}\big(1^2 - p^2\big)^{(n-1)/2}dp \\ &\leq V(n-1,1)\int_{\delta}^{\infty}\Big(e^{-p^2}\Big)^{(n-1)/2}dp && \text{because } 1 + x \leq e^x \\ &\leq V(n-1,1)\int_{\delta}^{\infty}\tfrac{p}{\delta}\Big(e^{-p^2}\Big)^{(n-1)/2}dp && \text{because } \tfrac{Proj_{\vec{p}}\vec{x}}{\delta} \geq 1 \ \forall \vec{x} \text{ in the region} \\ & && \text{of integration} \\ &= \tfrac{1}{\delta}V(n-1,1)\Big(-\tfrac{1}{n-1}e^{-\frac{n-1}{2}p^2}\Big)\Big|_{\delta}^{\infty} \\ &= \tfrac{1}{\delta}V(n-1,1)\Big(\tfrac{1}{n-1}e^{-\frac{n-1}{2}\delta^2}\Big). \end{aligned}$$

Therefore, define $\alpha = \frac{1}{\delta(n-1)} V(n-1,1) e^{-\frac{n-1}{2}\delta^2}$.

Find $\beta$ such that $\mathrm{Vol}\big(H(n,1)\big) \geq \beta$ and $\frac{\alpha}{\beta} = \frac{2}{c} e^{-c^2/2}$.

We require that $\frac{\alpha}{\beta} = \frac{2}{c} e^{-c^2/2}$ with $\alpha = \frac{1}{\delta(n-1)} V(n-1,1) e^{-\frac{n-1}{2}\delta^2}$. Consequently, we require $\beta = \frac{\delta}{2} V(n-1,1)$ after choosing $\delta = \frac{c}{\sqrt{n-1}}$. Therefore, it remains to show that $\mathrm{Vol}\big(H(n,1)\big) \geq \frac{\delta}{2} V(n-1,1)$. Since the volume of $H(n,1)$ is greater than any cylinder it contains and using the fact that $V(n,R) = R^n V(n,1)$, then

$$\begin{aligned} \mathrm{Vol}\big(H(n,1)\big) &> V(n-1,1)\big(1-h^2\big)^{\frac{n-1}{2}} h \quad \text{for } h \leq 1 \\ &\geq V(n-1,1)\big(1-\tfrac{n-1}{2}h^2\big)h \quad \text{since } (1-x)^a \geq 1-ax, a \geq 1, \end{aligned}$$

which thereby requires that $\frac{n-1}{2} \geq 1$ or that $n \geq 3$. If there exists a choice of $h \leq 1$ such that $\big(1-\frac{n-1}{2}h^2\big)h \geq \frac{\delta}{2}$, we are done. Letting $h = \frac{1}{\sqrt{n-1}}$ satisfies the given inequality, concluding the proof. □

**Theorem 2.1.2.** *Given $V(n,R)$, the volume of an n-dimensional hyperball with radius R, the fraction of the volume of $V(n,R)$ contained in its $\delta$-thickened hypersphere $V_{(\delta,R)}(n,R)$ has the upper bound:*

$$\frac{V_{(\delta,R)}(n,R)}{V(n,R)} \leq 1 - e^{-\delta n/R}.$$

*Proof.* We have that

$$\begin{aligned} V(n,R-\delta) + V_{(\delta,R)}(n,R) &= V(n,R) \\ \Rightarrow \quad V_{(\delta,R)}(n,R) &= V(n,R) - V(n,R-\delta) \\ \Rightarrow \quad \tfrac{V_{(\delta,R)}(n,R)}{V(n,R)} &= 1 - \tfrac{V(n,R-\delta)}{V(n,R)} \\ &= 1 - \tfrac{V(n,R-\delta)}{V(n,R)} \\ &= 1 - \left(1-\tfrac{\delta}{R}\right)^n \text{ because } \tfrac{V(n,R-\delta)}{V(n,R)} = \tfrac{(R-\delta)^n}{R^n} = \left(1-\tfrac{\delta}{R}\right)^n \\ &\leq 1 - \left(e^{-\delta/R}\right)^n \text{ because } 1+x \leq e^x \\ &\leq 1 - e^{-\delta n/R}. \end{aligned}$$

□

### 2.1.2 Concentration of Surface Area

The surface of the $(n)$-ball is the $(n-1)$-sphere $S(n-1,R)$ also with radius $R$ and centered about the origin $\big($thus $S(n-1,R) = \{\vec{x} \in \mathbb{R}^n \ : \ \|\vec{x}\|_2 = R\}\big)$. The surface area of $S(n-1,R)$, $A(n-1,R)$, is the derivative of $V(n,R)$ with respect to $R$. After choice of $\vec{p}$, the hypersphere's *equator* $S_E(n-1,R)$ is the set

$$S_E(n-1,R) = \{\vec{x} \in S(n-1,R) \ : \ \vec{x} \cdot \vec{p} = 0\}.$$

Let $S_{(\delta)}(n-1,R)$ be the intersection of the $\delta$-thickened equator of $B(n,R)$ and $S(n-1,R)$, so

$$S_{(\delta)}(n-1,R) = B_{(R,\delta)}(n,R) \ \cap \ S(n-1,R),$$

and let $A_{(\delta)}(n-1,R)$ denote its surface area. Call $S_{(\delta)}(n-1,R)$ the hypersphere's *$\delta$-thickened equator*. As $n$ increases, the surface area of a hypersphere concentrates about its equator, formally stated in Theorem 2.1.3 whose statement and proof are adapted from those of Lemma 2.3 in [6].

**Theorem 2.1.3.** *For $n \geq 4$, $\delta = \frac{c}{\sqrt{n-2}}$, and for any $c > 0$, the $\delta$-thickened equator of the $(n-1)$-sphere contains at least a $1 - \frac{2}{c}e^{-c^2/2}$ fraction of its total surface area; i.e.,*

$$1 - \frac{2}{c}e^{-c^2/2} \leq \frac{A_{(\delta)}(n-1,1)}{A(n-1,1)}.$$

*Proof.* The proof of Theorem 2.1.3 is almost identical to that of Theorem 2.1.1, so some details that can be found in the earlier proof are left out here. The main difference between the two proofs is that previously, we considered $n$-dimensional balls, which have slices (used in integration) that are $(n-1)$-balls, while here, we consider $(n-1)$-dimensional spheres, which have slices that are

$(n-2)$-dimensional spheres. We have that

$$\frac{A_{(\delta)}(n-1,1)}{A(n-1,1)} = 1 - \frac{A(n-1,1)\backslash A_{(\delta)}(n-1,1)}{A(n-1,1)}$$

and that it is sufficient to show that $\mathrm{SA}\big(T_{(\delta)}(n,1)\big) \leq \alpha$ and $\mathrm{SA}\big(H(n,1)\big) \geq \beta$ where $\frac{\alpha}{\beta} = \frac{2}{c}e^{-c^2/2}$ and $\mathrm{SA}(x)$ denotes the surface area of $x$.

Find $\alpha$ such that $\mathrm{SA}\big(T_{(\delta)}(n,1)\big) \leq \alpha$.

$$\begin{aligned}\mathrm{SA}\big(T_{(\delta)}(n,1)\big) &= A(n-1,1)\int_{\delta}^{1}\big(1^2-p^2\big)^{(n-2)/2}dp \\ &\leq \tfrac{1}{\delta}A(n-1,1)\Big(\tfrac{1}{n-2}e^{-\frac{n-2}{2}\delta^2}\Big).\end{aligned}$$

Therefore, define $\alpha = \frac{1}{\delta(n-2)}A(n-1,1)e^{-\frac{n-2}{2}\delta^2}$.

Find $\beta$ such that $\mathrm{SA}\big(H(n,1)\big) \geq \beta$ and $\frac{\alpha}{\beta} = \frac{2}{c}e^{-c^2/2}$.

We require that $\beta = \frac{\delta}{2}A(n-1,1)$ after choosing $\delta = \frac{c}{\sqrt{n-2}}$. Therefore, it remains to show that $\mathrm{SA}\big(H(n,1)\big) \geq \frac{\delta}{2}A(n-1,1)$. Since the surface area of $H(n,1)$ is greater than any cylinder it contains (given two nested convex shapes, the surface area of the inscribed shape is less than that of the circumscribed one) and using the fact that $A(n,R) = R^n A(n,1)$, then

$$\begin{aligned}\mathrm{SA}\big(H(n,1)\big) &> A(n-1,1)\big(1-h^2\big)^{\frac{n-2}{2}}h && \text{for } h \leq 1 \\ &\geq A(n-1,1)\big(1-\tfrac{n-2}{2}h^2\big)h && \text{since } (1-x)^a \geq 1-ax, a \geq 1, \text{requiring that } n \geq 4, \\ &\geq \tfrac{\delta}{2}A(n-1,1) && \text{after choosing } h = \tfrac{1}{\sqrt{n-2}},\end{aligned}$$

concluding the proof. □

### 2.1.3 Orthogonality of Large Random Vectors

The fact that high-dimensional vectors are nearly orthogonal is common knowledge in the fields of computational science and statistics (based on its frequent mention without citation or being called a 'well-known' fact [7–12]). To be more formal than 'nearly orthogonal', two vectors $x \neq y,\ x, y \in \mathscr{V}$, a finite-dimensional Euclidean vector space, are $\varepsilon$-*quasiorthogonal* if the cosine of their angular distance is between $-\varepsilon$ and $\varepsilon$; i.e. if $\cos\theta_{x,y} \in \big[-\varepsilon, \varepsilon\big]$. Therefore, two random vectors in high dimensions have a high probability of being $\varepsilon$-orthogonal, for some small $\varepsilon$. This is a result of Theorem 2.1.1 [13]. Given an $\varepsilon \in [0,1)$, the $\varepsilon$-*quasiorthogonal dimension* is the maximum cardinality of a set of $\varepsilon$-quasiorthogonal vectors in $\mathscr{V}$. The seemingly first work analyzing the relationship between orthogonality and ambient dimension shows that $\varepsilon$-quasiorthogonal dimension grows exponentially with the increase in the dimension of $\mathscr{V}$ [14]. Other work quantifies the probability that $\theta_{x,y}$ lies above some arbitrarily small threshold [15]. More recently, it has been shown that the dot product between two distinct uniformly random vectors converges to 0 with the increase in their dimension [7]. We formally state this in Theorem 2.1.4. Regardless of the precise statement of this theorem, it can be leveraged to define a basis for a low-dimensional representation of high-dimensional data. Namely, if an intrinsically $k$-dimensional representation is desired, one can generate $k$ random vectors in the large ambient dimension and use them as a basis for the data. Dimensionality reduction is a well-researched area of mathematics which we cover next.

**Theorem 2.1.4.** *Given two distinct vectors $x, y \in \mathbb{R}^m$,*

$$\frac{\langle x, y\rangle}{\|x\| \cdot \|y\|} \xrightarrow[m\to\infty]{} 0.$$

## 2.2 Dimensionality Reduction

There are many notions of dimension—intrinsic, ambient, fractal, Hausdorff, correlation, and embedding dimensions to name a few—and precisely defining their interplay can be quite nuanced [16]. To side-step this mathematical tongue twister, we illustrate some of these notions of dimension with an example. Consider a unit 2-sphere in $\mathbb{R}^5$ given as the slice of a unit 4-sphere

with two hyperplanes:

$$S = \left\{ x \in \mathbb{R}^5 \ : \ \left( \sum_{i=1}^{5} x_i^2 \right)^{1/2} = 1 \text{ and } x_4 = x_5 = 0 \right\}.$$

Its ambient dimension is five because any point on its surface is of the form $x = [x_1, x_2, x_3, x_4, x_5]^\top$, which has five components. Meanwhile, its intrinsic dimension, the minimum number of parameters required to represent $S$ without loss [17], is three because this sphere is the same as the object described by the equation $b_1^2 + b_2^2 + b_3^2 = 1$ where $b_i$ provide an orthonormal basis of $\mathbb{R}^3$. Lastly, this sphere has a topological dimension of two because its tangent space is a two-dimensional manifold; i.e., for any small local region, the sphere is approximately a plane. If we densely sample this sphere, this collection of data points inherits the dimensions of the sphere on which they lie. The introduction of noise or sparsity to these data introduces some ambiguity: at what point is a higher-dimensional manifold better suited to approximate the data than the sphere? Caculating the intrinsic dimension of data is still an open area of research with a substantial literature base [17–38]. While the geometry, topology, and dimension of data are important to our research (we will show that the structure of training data fundamentally impacts the neural network returned after training) the dimensionality reduction we use is not intended to be without loss. Our data is densely present in $\mathbb{R}^N$, for large $N$, so we simply define an intrinsically two-dimensional subspace $P$ of $\mathbb{R}^N$ and visualize the data intersecting $P$. We will define $P$ by choosing a basis consisting of two orthonormal vectors which each consist of $N$ components and the boundaries of $P$. Put differently, after choice of two orthonormal vectors, $u_1$ and $u_2$, and their respective upper and lower bounds, $\alpha_{lb}, \alpha_{ub}, \beta_{lb}$, and $\beta_{ub}$, we define $P = \left\{ x \in \mathbb{R}^N \ : \ x = \alpha u_1 + \beta u_2 \text{ and } \alpha \in [\alpha_{lb}, \alpha_{ub}], \ \beta \in [\beta_{lb}, \beta_{ub}] \right\}$. Therefore we are interested in methods for generating these orthonormal vectors $u_1$ and $u_2$. In this section, we define methods that we use in addition to the theorem in Section 2.1.3 to define the bases of input space landscapes in Section 5.3 and loss landscapes in Section 6.3.3.

### 2.2.1 Singular Value Decomposition (SVD)

**Theorem 2.2.1.** *If $A \in \mathbb{R}^{n\times m}$, $n \geq m$ $\left(\text{if } m > n \text{ then consider } A^\top\right)$, then there exists an $n \times n$ orthogonal matrix $U$, an $m \times m$ orthogonal matrix $V$, and an $n \times m$ "diagonal" matrix $\Sigma = diag(\sigma_1, \ldots, \sigma_m)$ where $\sigma_1 \geq \cdots \geq \sigma_m \geq 0$ such that $A = U\Sigma V^\top$. The columns $u_1, \ldots, u_m$ of $U$ are called left singular vectors, the columns $v_1, \ldots, v_m$ of $V$ are the right singular vectors, and the diagonal entries $\sigma_i$ of $\Sigma$ are singular values [39].*

Considering $A \in \mathbb{R}^{n\times m}$ as a mapping from $\mathbb{R}^m$ to $\mathbb{R}^n$, Theorem 2.2.1 provides a basis for $\mathbb{R}^m$, the columns of $V$, and a basis for $\mathbb{R}^n$, the columns of $U$:

$$\begin{array}{ccccc} A: & \mathbb{R}^m & \longrightarrow & \mathbb{R}^n & \\ & \cup & & \cup & . \\ & x = \sum_{i=1}^m \beta_i v_i & \mapsto & Ax = \sum_{i=1}^n \sigma_i \beta_i u_i & \end{array}$$

After constructing $A$ by horizontally stacking points from the input space (as column vectors), the first $k$ columns of $U$ can be used to define a $k$-dimensional subspace of $\mathbb{R}^m$. The multiplication of any orthogonal basis with any non-zero scalar still results in an orthogonal set of vectors. Choosing this scalar to be $-1$ does not change the length of the basis vectors, and therefore, if we start with an orthonormal set of vectors, multiplication of any of the vectors by $-1$ still results in an orthonormal set. SVD suffers from "sign indeterminacy," the resulting basis is unique up to a sign change. Implementations of the algorithm, including the one we use in Python, give results that differ by a sign while keeping the input matrix $A$ unchanged. Therefore, when using SVD, it is important to either save the basis used so that it can be loaded and used again if needed or to devise a method of checking the sign compatibility of a new basis with the one used to represent previous data in low dimensions.

### 2.2.2 Gram-Schmidt

**Theorem 2.2.2.** *Given $A \in \mathbb{R}^{n\times m}, n \geq m$, a matrix with full column rank, there exists a unique orthogonal matrix $Q \in \mathbb{R}^{n\times m}$ and a unique upper triangular matrix $R \in \mathbb{R}^{m\times m}$ with positive diagonal entries such that $A = QR$ [39].*

The particular goal of the Gram-Schmidt algorithm varies slightly in the literature: it has been used to compute the QR-decomposition of a matrix toward a least squares solution [39] and as an orthogonalization process [40]. These two are equivalent: the QR-decomposition of a matrix $A$ finds the matrices $Q$ and $R$ from Theorem 2.2.2 where the columns of $Q$ provide orthonormal vectors that span the same space as $A$. We provide the Gram-Schmidt procedure in Algorithm 1, noting that $r_{i,j}$ are saved only if the QR-decomposition is desired. Gram-Schmidt is known to be numerically unstable when the columns of $A$ are nearly linearly dependent [39]. By using the original columns of $A$ to update the columns of $Q$, round-off errors accumulate with each iteration of the inner for-loop and can result in a non-orthogonal matrix $Q$. Instead, the updated columns of $Q$ can be used to define the elements of $R$ $\big($replacing $r_{j,i} = q_j^\top a_i$ with $r_{j,i} = q_j^\top q_i\big)$, resulting in a mathematically equivalent algorithm that does not accumulate round-off errors, Modified Gram-Schmidt (MGS) (see Appendix A for proof of equivalence). To clearly distinguish between the two algorithms, Algorithm 1 is often called Classical Gram-Schmidt (CGS). Therefore, when $m$ (the ambient dimension of the input space) is small enough that Theorem 2.1.4 no longer guarantees near-orthogonality between two random vectors, MDS can be used instead to more stably determine an arbitrary $k$-dimensional representation of high-dimensional data $\big($after initializing a random $A \in \mathbb{R}^{m\times k}\big)$.

**Algorithm 1** Gram-Schmidt [39]

Let $A \in \mathbb{R}^{n \times m}$ be a full column rank matrix; i.e., a matrix whose columns are some basis $\mathscr{A} = \{a_1, \ldots, a_m\}$.

**for** $i = 1, \ldots, m$ **do**

$q_i = a_i$

**for** $j = 1, \ldots, i = 1$ **do**

$r_{j,i} = q_j^\top a_i$

$q_i = q_i - r_{j,i} q_j$

**end for**

$r_{i,i} = \|q_i\|_2$

**if** $r_{i,i} = 0$ **then** quit

**end if**

$q_i = \frac{q_i}{r_{i,i}}$

**end for**

# Chapter 3

# Models and Datasets

## 3.1 Introduction

Here, we continue to lay the groundwork for the experiments to ensue by defining relevant models and datasets. Beyond simple and small feed-forward neural networks (FFNNs) that are introduced as needed, our experiments use pretrained ResNet models: ResNet-18, ResNet-50, ResNet-56, and ResNet-56-noshort. The first two of these were pretrained using ImageNet, while the latter pair underwent training with CIFAR-10. We present these architectures and the datasets that molded their learning. We additionally introduce two adversarial datasets, one of which was created using the VGG-19 model, thereby warranting the introduction of the VGG architecture as well. The models and datasets we use are part of what set our work apart—most machine learning research of a polyhedral flavor like our own does not move beyond basic FFNNs, low dimensional data, or a combination of the two.

## 3.2 Models

### 3.2.1 VGG

The Visual Geometry Group (VGG) at the University of Oxford developed VGG-19 [41] for the ImageNet Challenge 2014, winning the localization and classification category of the competition with an impressive performance. Their classification error rate was half that of the second-place contender. Their model, VGG-19, consists of 19 non-pooling layers and has a simple forward pass flow: the output of one layer is exactly the input of the subsequent layer. The precise architecture is given in Figure 3.2a, which was created directly in LaTeX using the examples from the GitHub page associated with [42] as a starting point. Breaking the model into 8 blocks (as shown in Figure 3.2a), Blocks 1 and 2 consist of two convolutional layers followed by a max pooling layer; Blocks 3, 4, and 5 consist of four convolutional layers followed by a max pooling layer; and

Blocks 6, 7, and 8 are linear layers. All layers in Blocks 1–7 use the ReLU activation function while the final layer of the neural network uses the Softmax activation function. Table 3.1 shows that this model is significantly larger (in terms of the number of parameters) than the ResNet architectures that perform at least as well after PyTorch's pretraining as VGG-19. We observe that to linearly and exactly express this neural network, the only necessary adjustments involve substituting the softmax activation function in the final layer with a linear activation function. Following these minor modifications, linearization becomes straightforward: one can utilize the GitHub code associated with [43] to linearize both the convolutional and pooling layers.

As is common practice, transforms were done on data before training VGG-19. This preprocessing is usually specific to the dataset used to train the model of interest and is important to keep in mind. To effectively use a model, these same transforms should be applied to any data taken as input. VGG-19 input images are expected to have dimensions of at least $32 \times 32$. This flexibility of input size is common among neural networks and is why '$I$' is used to denote the convolutional layer sizes in Figure 3.2 where $I$ indicates the image size. Images are then expected to be converted from RGB to BGR, then channel-wise mean-centered with respect to the ImageNet dataset (which has an RGB channel-wise mean of $[123.68, 116.779, 103.939]$ if entries are scaled between 0 and 255, as images are), without scaling (without the use of a specified standard deviation). Resizing is not part of the pre-built keras.applications.vgg19.preprocess_input function, so freedoms can be taken in the resizing step of input preprocessing. In the creation of our dataset of interest introduced in Section 3.4.1 (the sole reason we introduce the VGG-19 model), inputs were made to be of size $224 \times 224$ using bilinear interpolation. The remainder of the preprocessing steps are completed by keras.applications.vgg19.preprocess_input.

### 3.2.2 ResNet

A year after the introduction of VGG, the residual network (ResNet) architecture was presented in [44] and entered in the ImageNet Challenge 2015 where it achieved almost a 30% improvement in classification from the performance of VGG-19. Despite this difference in accuracy,

the elements and their order in ResNet are actually very similar to those of VGG-19. Considering ResNet-18 in particular, its architecture differs from VGG-19 in that batch normalization is applied after convolutional layers and before the ReLU activation function, Blocks 1, 6, and 7 were removed along with all of the max pooling layers, and each of the convolutional layers of the remaining blocks have half as many filters as their VGG-19 counterparts. These changes significantly contribute toward the increased computational simplicity of ResNet-18 from VGG-19: for a single forward pass, ResNet-18 performs 1.81 billion floating-point operations per second (GFLOPS) while VGG-19 performs 19.63 GFLOPS. These differences and their benefits aside, the distinctive flow of ResNet's architecture is its most distinguishing feature. The output of every other layer is added to the output two layers later with the use of skip connections. In this way, information from previous layers trickles into later layers. The exact architecture is given in Figure 3.2b. These skip connections substantially cut the number of parameters required by the model and the number of computations it performs. To express ResNet-18 in a simple feed-forward fashion would require the strategic stacking of identity matrices on the layerwise weight matrices and of zero vectors on the layerwise bias vectors to preserve the output of each block so that it can be added to the output of the next block. By using skip connections, we do not need to perform these many multiplication and addition operations, but instead directly feed outputs where they are required. We also forego defining these larger weight matrices and bias vectors that simply apply the identity function. After ResNet-18, deeper versions such as ResNet-50 have been introduced. In some of our work assessing the effects of skip connections (Section 6.3.3), we also consider ResNet-56 and ResNet-56-noshort. ResNet-56-noshort is identical to ResNet-56 but removes the skip connections, so its architecture ends up being virtually a VGG model.

As mentioned in Section 3.2.1, transforms are often done before training models. The ResNet architectures we use are no exception. As indicated in Table 3.1, the ResNet-18 and ResNet-50 models we use were pretrained on the ImageNet dataset and are made available by PyTorch. The preprocessing done to the dataset before training is as follows: center-cropping or padding images with zero entries to the size of $224 \times 2224$, scaling values between 0 and 1, and normalizing them

**Table 3.1:** Summary of model sizes, accuracies, and the training data used to obtain the pretrained weights.

| Model | Training Data | Acc@1 | Acc@5 | Parameter Count |
|---|---|---|---|---|
| ResNet-18 | IMAGENET1K_V1 | 69.758% | 89.078% | 11.7M |
| ResNet-152 | IMAGENET1K_V1 | 78.312% | 94.046% | 60.2M |
| ResNet-56 | CIFAR-10 | 21.78% | 78.60% | 0.86M |
| ResNet-56-noshort | CIFAR-10 | 10.00% | 52.46% | 0.85M |
| ResNet-50 | IMAGENET1K_V2 | 80.858% | 95.434% | 25.6M |
| Wide ResNet-50 | IMAGENET1K_V2 | 81.602% | 95.758% | 68.9M |
| VGG19 (PyTorch) | IMAGENET1K_V1 | 72.376% | 90.876% | 143.7M |
| VGG19 (TensorFlow) | IMAGENET1K_V1 | 71.3% | 90.1% | 143.7M |

such that the RGB channel means are 0.485, 0.456, and 0.406[1] and the channel standard deviations are 0.229, 0.224, 0.225, respectively. Three examples of ImageNet images before and after these transforms are included in Figure 3.3. The ResNet-56 and ResNet-56-noshort models we use were pretrained on the CIFAR-10 dataset and are made available by the authors of [9] who preprocessed the images by normalizing them to have a channel-wise mean of $[125.3, 123.0, 113.9]$ and standard deviation of $[63.0, 62.1, 66.7]$, and then scaling their values between 0 and 1.

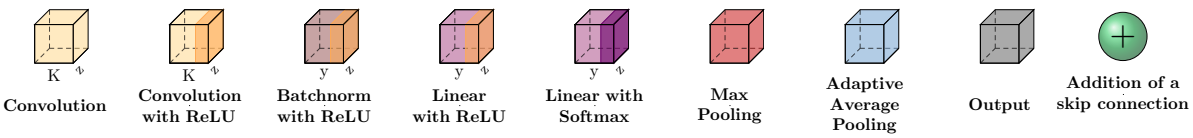
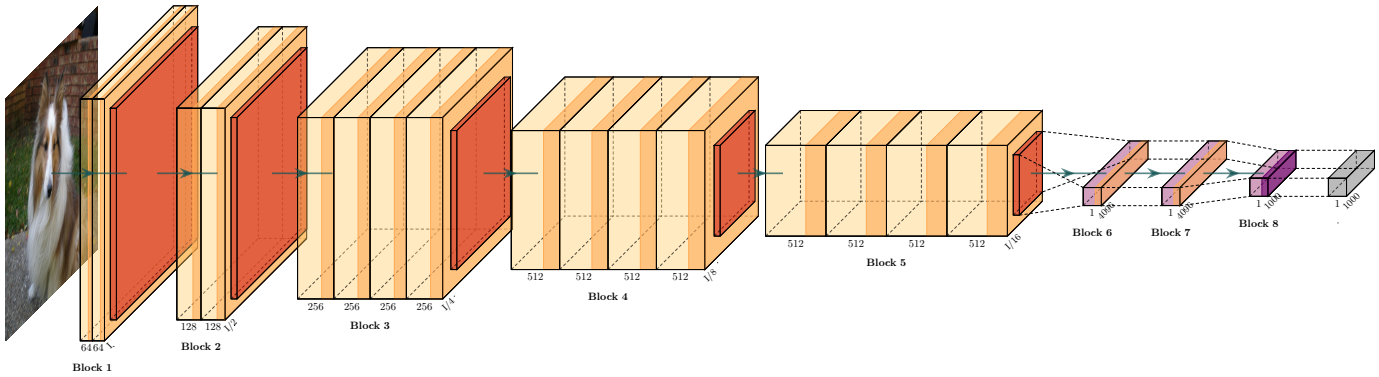
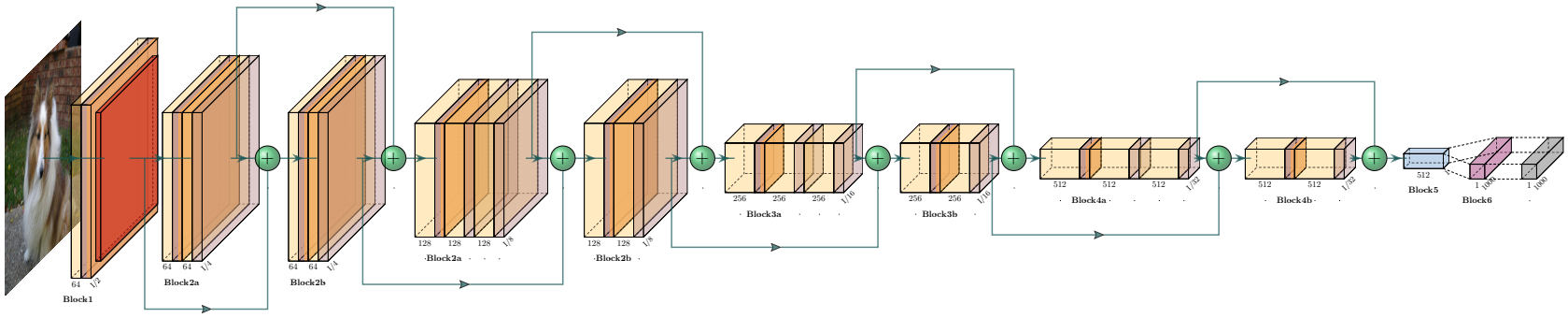



**Figure 3.1:** Legend for model architecture diagrams where K is the number of convolutional filters, z gives the layer input size (z $\times$ z), and y .

[1]Note that these channel means are identical to those provided in Section 3.2.1 once they are scaled between 0 and 1.




**(a)** VGG-19




**(b)** ResNet-18

**Figure 3.2:** The VGG-19 and ResNet-18 architectures as they are made available by TensorFlow and PyTorch, respectively (see Figure 3.1 for the legend).

## 3.3 Natural Image Datasets

### 3.3.1 MNIST

The Modified National Institute of Standards and Technology (MNIST) dataset is a collection of 70 thousand $28 \times 28$ black and white images of hand-written digits [45]. Due to the uniqueness of integers between 0 and 9, the limited variability among different samples of the same class, and the unrealistically pristine quality of the data (devoid of noise, occlusions, poor croppings, backscatter, and the like), one can reasonably expect that MNIST has a rather simple structure. While classes are not linearly separable [45], images that are close to each other in Euclidean space likely belong to the same class [46]. As a result of its simplicity, MNIST's primary utility lies in acquainting novices with fundamental machine learning concepts, rendering it less relevant for cutting-edge research in the field. Observations made, performance levels achieved, and conclusions drawn using MNIST often do not translate to other datasets. Results that are included in Section 5.3.1 contest claims made from experiments skewed, in part, by this dataset.

### 3.3.2 CIFAR-10

A decade after MNIST was introduced, the Tiny Images dataset [47] was created and addressed some of the shortcomings of MNIST. This dataset was created by scraping the web for a total of 80 million images, about 3000 results for each of the non-abstract English nouns in the WordNet hierarchy [48]. WordNet establishes 12 subtrees (mammal, bird, fish, reptile, amphibian, vehicle, furniture, musical instrument, geological formation, tool, flower, fruit), each containing 5247 "synonym sets"/"synsets" or subclasses with labels that are hyponyms of their respective subtree's label. Since the images in the Tiny Images dataset were pulled from the internet using automated methods, they are not characterized by the same artificial uniformity of MNIST. Also as a consequence of the data collection process, and more specifically, the imperfection of search engines, the labels of the data are unreliable—so much so that the dataset contains some offensive content and has been withdrawn from public availability. While this dataset was still accessible, two datasets were made from it, CIFAR-10 and CIFAR-100 [49], an effort that was funded by the Canadian

Institute for Advanced Research (CIFAR). CIFAR-10 is a set of 60 thousand $32 \times 32$ color images equally divided into 10 non-overlapping classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The images in each class were chosen by first manually iterating through the 3000 images in the analogous class of the Tiny Images dataset, and then iterating through classes belonging to hyponyms of the original class until 6000 correctly labeled images were collected. CIFAR-10 is split up into a training set and a test set where the training set consists of 5 batches of 10 thousand images and the test set contains the remaining 10 thousand images. All 5 batches of training data were used to pretrain ResNet-56 and ResNet-56-noshort [9]. Due to the size of the CIFAR-10 images ($3 \times 32 \times 32$) and the number of classes in the subset of data used in training (10), the mapping done by these networks is

$$\begin{array}{rccc} F: & \mathbb{R}^{3\times 32\times 32} & \longrightarrow & \mathbb{R}^{10} \\ & \cup & & \cup \\ & x & \mapsto & y \end{array} \tag{3.1}$$

where Softmax$(y)$ can be interpreted as a probability distribution on $\{1, \ldots, \mathrm{K}\}$ where the training set consists of $K$ classes (here $K = 10$), so softmax$(y)_i$ indicates the likelihood that $x$ belongs to class $i$ [50]. Given the number of classes in this dataset, we note that 'Acc@1' and 'Acc@5' in Table 3.1 associated with models that were trained on CIFAR-10 indicate the percentage of the test data whose true label is among the top 10% and 50% of predicted labels, respectively. Despite the relaxed criteria implied by 'Acc@1' and 'Acc@5' within the CIFAR-10 context compared to those within the ImageNet context, the percentage of CIFAR-10 test data achieving these criteria using ResNet-56 and ResNet-56-noshort is significantly lower than the percentage of ImageNet test data achieving them with pretrained PyTorch models.

### 3.3.3 ImageNet

The ImageNet dataset was introduced in 2009 as one of the largest, most diverse, and most accurate datasets at the time (in terms of correct labels being assigned to its images, in contrast

with the unreliable labeling of the Tiny Images dataset mentioned in the previous section) [51]. It consists of $224 \times 224$ mostly color images and also uses the WordNet hierarchy. At the time of publication, ImageNet consisted of 3.2 million images in total subdivided into the synsets of WordNet, with approximately 600 images in each synset. The dataset has grown to now contain between 732 and 1300 images in each synset. There remains plenty of opportunity for it to continue to grow into the WordNet hierarchy which consists of around 80 thousand synsets. The creators of the dataset aim to have the dataset grow to be tens of millions of quality-controlled and human-annotated images.

Unlike MNIST, the size and quality of ImageNet make it perfect for training neural networks for competitive performance. Many widely used neural network architectures have been trained on a subset of ImageNet (called 'IMAGENET1K_V1' in PyTorch and in Table 3.1 which spans 1000 object classes and consists of 1281167 training images, 50 thousand validation images, and 100 thousand test images) most of which are 3-channeled color images, that are unequally subdivided into 1000 classes. More specifically, 105 of the classes contain between 732 and 1299 images, while the remaining classes have 1300 images. Another commonly used version of ImageNet used to train widely available networks is a refined version of IMAGENET1K_V1 (called 'IMAGENET1K_V2' in PyTorch and in Table 3.1) [52]. Due to the size of the ImageNet images ($3 \times 224 \times 224$) and the number of classes into which they are subdivided (1000), the mapping done by these networks is

$$\begin{array}{rccc} F: & \mathbb{R}^{3\times 224\times 224} & \longrightarrow & \mathbb{R}^{1000} \\ & \cup & & \cup \\ & x & \mapsto & y \end{array} \tag{3.2}$$

where, again, $\text{Softmax}(y)_i$ indicates the likelihood that $x$ belongs to label $i$. Given the dimension of the neural network's output space, we note that 'Acc@1' and 'Acc@5' in Table 3.1 associated with models that were trained on ImageNet indicate the percentage of the test data whose true label is among the top 0.1% and 0.5% of predicted labels, respectively.

**Figure 3.3:** Three ImageNet images before (top) and after (bottom) preprocessing required by PyTorch pretrained models ResNet-18 and ResNet-50.

## 3.4 Adversarial Datasets

Adversarial attacks have been defined in many ways. Here, we define an *adversarial attack* to be any alteration to an image with the intent of compromising a neural network or its output, regardless of the success of the attack. Attacks can take place during the training process of the targeted neural network or to the inputs of the model once it has been trained. In our work, we consider the latter of these. Therefore, by our definition of adversarial attack, the difference between data augmentation during training and an attack that takes place during training is intent: while the goal of data augmentation is to increase model robustness and generalizability, that of an attack is to decrease the model's accuracy in some capacity. Our definition is also independent of the outcome of the attack. We make this choice with the philosophy that the success of an attack depends on the severity of the attack and the resilience of the network, rather than simply the occurrence of an attack on the model. In this section, we introduce two altered versions of the ImageNet dataset.

### 3.4.1 DAmageNet

The DAmageNet dataset presented in [53] and later rigorously defined in [54] was created by particularly altering, or 'DAmage-ing', the 50 thousand ImageNet validation images implying a

natural one-to-one mapping between ImageNet validation images and DAmageNet. We call an ImageNet validation image together with its DAmage-ed version an ImageNet-DAmageNet pair. These image pairs have a natural ordering; their naming convention is 'ILSVRC2012_val_000#### #' (ImageNet-DAmageNet pairs have the same name and only differ in file extension, ImageNet images are .PNGs and DAmageNet images are .JPEGs) where the digits at the end of the name range between 00001 and 50000. Although adversarial images need not be mislabeled by the network of interest, DAmageNet was created in a way that forces a change in the predicted label—given an image $x$ from the ImageNet validation set, $\Delta x$ is a perturbation such that the classification of $x + \Delta x$ is different from that of $x$ by VGG-19 pretrained on ImageNet made available by TensorFlow (the VGG-19 architecture made available by PyTorch slightly deviates from that by TensorFlow). Ideally, $\Delta x$ is as small as possible to avoid detection. The attack used to create DAmageNet minimizes this $\Delta x$ while still misclassifying $x$. The precise mathematical details of the attack are given in [54]. Figure 3.4 provides three ImageNet-DAmageNet pairs. Given the visual similarity in these image pairs, it is not surprising that the root mean squared deviation between all image pairs is about 3.8 [53]. However, we use DAmageNet in the context of PyTorch's ResNet models, so we first preprocess these images as training data was preprocessed before training these particular models (see Section 3.2.2 for preprocessing details). The distribution of Euclidean distances between ImageNet-DAmageNet pairs after preprocessing them ranges between 47.08 and 1550.59 and is given in Figure 3.5.

### 3.4.2 ImageNet-FGSM

We perform the Fast Gradient Sign Method (FGSM) [55] to alter ImageNet, creating ImageNet-FGSM. Unlike DAmageNet, we do not ensure that the change made to each of the original images caused a change in the model's predicted label. FGSM produces a perturbed image, $\hat{x}$, from an

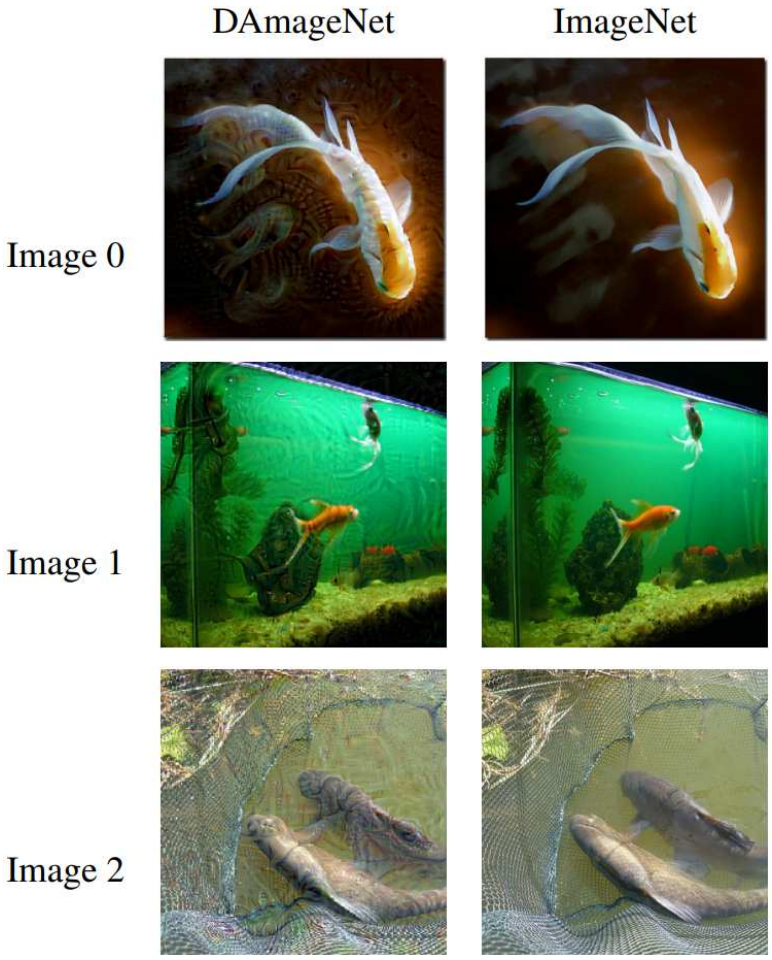



**Figure 3.4:** Examples of particular ImageNet images and their DAmageNet versions.

original image, $x$, given a neural network $F$ by

$$\hat{x} = x + \varepsilon * \text{sign}(\nabla_x C(\theta, x, y)) \text{ where } \begin{array}{l} \varepsilon \text{ is the perturbation parameter } (\varepsilon \in \mathbb{R}), \\ C \text{ is the cost function used to train } F, \\ \theta \text{ are the model variables from } F, \text{ and} \\ y = F(x). \end{array} \tag{3.3}$$

The authors of [55] found that the method caused misclassifications among a variety of models. Setting (1) $\varepsilon = 0.25$ resulted in (a) the misclassification rate of 99.9% with an average confidence of 79.3% of a shallow softmax classifier and (b) the misclassification rate of 89.4% with an average confidence of 97.6% of a maxout classifier, both on an FGSM-perturbed version of MNIST [45];

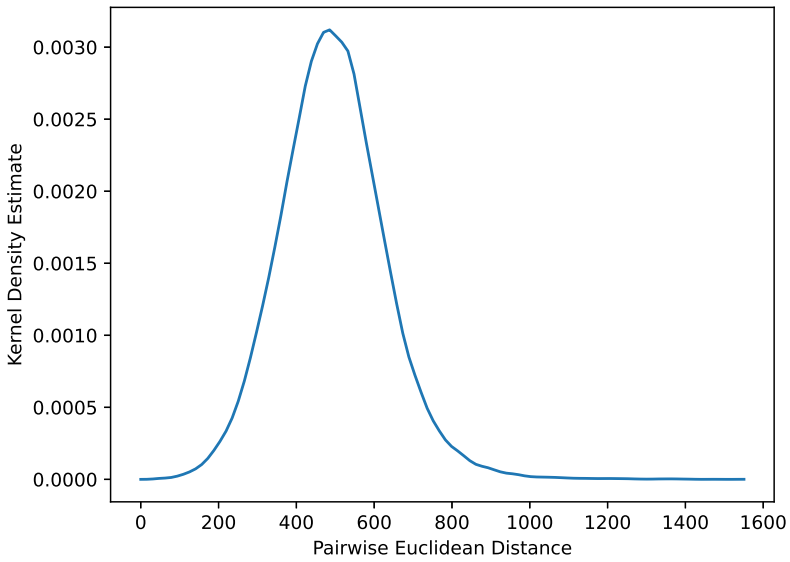



**Figure 3.5:** The Kernel Density Estimate of the Euclidean distances between all preprocessed ImageNet-DAmageNet pairs.

and (2) $\varepsilon = 0.1$ resulted in the misclassification rate of 87.2% with an average probability of 96.6% of a maxout classifier on an FGSM-perturbed version of CIFAR-10 [49]. We do not explore the error rates of our perturbation datasets but use FGSM as a method of traversing the input space of a neural network.

## 3.5 Conclusion

We have introduced VGG-19, four ResNet architectures (ResNet-18, ResNet-50, ResNet-56, and ResNet-56-noshort), and the MNIST, CIFAR-10, ImageNet, DAmageNet, and ImageNet-FGSM datasets. Some of these neural networks will be examined to identify geometric differences between original images and their adversarial counterparts. Meanwhile, others will be probed for topological information about their respective datasets, shedding light on how architectural decisions influence neural network topology. Before performing these analyses in later chapters, it is important to first understand a particular representation of a neural network: its polyhedral decomposition.

# Chapter 4

# Polyhedral Decompositions

## 4.1 Introduction

All neural networks have associated polyhedral decompositions that express their actions on input data. Polyhedral decompositions are partitions that span the neural network's entire input space and are composed of as many distinct regions as the number of unique neural firing patterns the model produces for all points in the input space. Each polyhedron defines a distinct composite function in terms of the neural network parameters from the input space to the output space. A linear neural network is exactly the piecewise union of all these functions [56, 57], so we formally define them and their respective domains from the parameters of the neural network in Section 4.2. It is worth mentioning that our formulation of these domains implies their convexity, a property that has been claimed [57, 58] and proven [59]. The only difference among these composite functions characterizing all polyhedra is the effect of their unique respective firing patterns or *binning vectors*, so we say that binning vectors label their respective polyhedra. These binning vectors impose a geometry on the input space which we quantify with the *Hamming distance*, which is equivalent to counting the number of entries in which two binning vectors differ. Thus, any polyhedral decomposition has an equivalent graph representation where the nodes are the polyhedra labeled by their bit vectors and the edges between any two nodes are weighted by their Hamming distance. We explore these graphical representations in Section 4.5. This graph exactly summarizes its associated neural network if (a) the neural network can be expressed linearly and (b) the chosen activation functions are piecewise linear (e.g. ReLU), and is approximate otherwise [3].

Combinatorially, the upper bound for the number of unique binning vectors (and, consequentially, the upper bound for the number of distinct regions of the polyhedral decomposition and the upper bound for the Hamming distance between any two bit vectors) is $2^h$ where $h$ is the number of neural nodes in the network. Often, the number of polyhedra that emerge after training a

neural network is well below this upper bound. Due to the relationship between the number of these distinct regions and the expressivity of the associated network, some work has been done to predict how many will emerge after training a network with a particular architecture [56, 59–63]. Regardless of the exact number of polyhedra, they are a limited resource of the neural network to accurately handle its training data during the learning process. We show in Section 4.3 that training, more than anything, is the strategic allocation of these resources. Therefore, the size and shape of polyhedra are also very relevant in the conversation about the fundamentals of neural networks. To approximate their sizes, the authors of [58] demonstrate that these partitions correspond to a power diagram and compute each tile's centroid and radius. We also explore polyhedral size and shape in Section 4.4 with the explicit use of a polyhedral decomposition. The geometric analysis of polytopes has also been done by implicitly using polyhedral decompositions, a topic we save for Chapter 5.

**Novel contributions in this chapter:**

- formally define the set of linear inequalities satisfied by all points within the same polyhedra (published in 2023 International Conference on Machine Learning Topology, Algebra, and Geometry (TAG) Workshops [3]),
- extend the above work to be applicable to other activation functions,
- show that the input-output Jacobian norm tends to increase in polyhedra that contain or are near the neural network's learned decision boundary (and can therefore be used to quantify neural network uncertainty) and that refined polyhedra tend to outline the decision boundary, and
- represent polyhedral decompositions as graphs to show the above behavior regarding polyhedral refinement and to quantify polyhedral proximity to polyhedra with high input-output Jacobian norms.

## 4.2 Formulation

### 4.2.1 Feed-Forward Neural Networks

Consider the feed-forward neural network (FFNN) $F : \mathbb{R}^m \to \mathbb{R}^n$ described in (4.2) with $L+2$ layers ($L$ hidden layers, an input layer, and an output layer) each of which is $h_i$-dimensional, $h_0 = m$, $h_{L+1} = n$, and $\sum_{i=1}^{L} h_i = h$. We can equivalently express $F$ as the composite function

$$F = F_{L+1} \circ f_L \circ F_L \circ \ldots \circ f_1 \circ F_1 \tag{4.1}$$

where passing information from the $(i-1)^{\text{th}}$ layer to the $i^{\text{th}}$ layer is described by the composition of the activation function $f_i$ with the affine map $F_i$. Each of the last $L+1$ layers of $F$ is equipped with weight matrix, $W^{(i)} \in \mathbb{R}^{h_i \times h_{i-1}}$, and a bias vector, $b^{(i)} \in \mathbb{R}^{h_i}$, both used to express $F_i$ as the weighted sum defined in (4.3). We denote the output of $F_i$ and $f_i$ by $x^{(i)} = \left[x_1^{(i)}, \ldots, x_{h_i}^{(i)}\right]^\top$ and $\tilde{x}^{(i)} = \left[\tilde{x}_1^{(i)}, \ldots, \tilde{x}_{h_i}^{(i)}\right]^\top$, respectively. For ease of notation, $x^{(i)} = \tilde{x}^{(i)}$ for $i = 0$, only true for other $i$ if $f_i = I$, the identity map.

$$\begin{array}{cccccccccc} F: & \mathbb{R}^m & \xrightarrow[f_1]{F_1} & \mathbb{R}^{h_1} & \to \ldots \to & \mathbb{R}^{h_{L-1}} & \xrightarrow[f_L]{F_L} & \mathbb{R}^{h_L} & \xrightarrow{F_{L+1}} & \mathbb{R}^n \\ & \cup & & \cup & & \cup & & \cup & & \cup \\ & x^{(0)} & \to & \tilde{x}^{(1)} & \to \ldots \to & \tilde{x}^{(L-1)} & \to & \tilde{x}^{(L)} & \to & x^{(L+1)} \end{array} \tag{4.2}$$

where

$$\tilde{x}^{(i)} = f_i\big(x^{(i)}\big) = f_i\Big(F_i\big(\tilde{x}^{(i-1)}\big)\Big) = f_i\big(W^{(i)}\tilde{x}^{(i-1)} + b^{(i)}\big) \quad \text{for } i = 1, \ldots, L. \tag{4.3}$$

We can express the activation functions in terms of linear operations as we have done for the affine mappings. All that is needed is to discretize the $f_i$'s appropriately for their particular inputs, $x^{(i)}$.

We introduce a diagonal matrix $A^{(i)} \in \mathbb{R}^{h_{i+1} \times h_i}$ and vector $a^{(i)} \in \mathbb{R}^{h_i}$ toward the discretization of $f_i$:

$$f_i\big(x^{(i)}\big) = A^{(i)}x^{(i)} + a^{(i)} \tag{4.4}$$

where

$$\begin{cases} A^{(i)}_{j,j} = \frac{f_i(x^{(i)}_j)}{x^{(i)}_j},\ a^{(i)}_j = 0 & \text{if } x^{(i)}_j \neq 0 \\ A^{(i)}_{j,j} = 0, \qquad a^{(i)}_j = f_i(x^{(i)}_j) & \text{otherwise} \end{cases} \quad \text{for } j = 1,\dots,h_i,\ i = 1,\dots,L. \tag{4.5}$$

with $x^{(i)}_j = \sum_{k=1}^{h_{i-1}} W^{(i)}_{j,k}\ \tilde{x}^{(i-1)}_j + b^{(i)}_j$ from (4.3). Now $f_i$ can indeed be expressed in terms of linear operations

$$f_i\big(x^{(i)}\big) = A^{(i)}x^{(i)} + a^{(i)} = A^{(i)}\big(W^{(i)}\tilde{x}^{(i-1)} + b^{(i)}\big) + a^{(i)} = \tilde{W}^{(i)}\tilde{x}^{(i-1)} + \tilde{b}^{(i)} \tag{4.6}$$

where

$$\tilde{W}^{(i)} = A^{(i)}W^{(i)} \text{ and } \tilde{b}^{(i)} = A^{(i)}b^{(i)} + a^{(i)}. \tag{4.7}$$

With this, the final neural network output, $x^{(L+1)}$, can be explicitly expressed by

$$x^{(L+1)} = F(x^{(0)}) = W^{(L+1)}\Big(\tilde{W}^{(L)}\dots\big(\tilde{W}^{(2)}(\tilde{W}^{(1)}x^{(0)} + \tilde{b}^{(1)}) + \tilde{b}^{(2)}\big)\dots + \tilde{b}^{(L)}\Big) + b^{(L+1)} \tag{4.8}$$

and layerwise outputs, $x^{(i)}$, by

$$x^{(i)} = F_i \circ f_{i-1} \dots \circ f_1 \circ F_1\big(x^{(0)}\big). \tag{4.9}$$

### 4.2.2 Binning Vectors and Activation Functions

#### Rectified Linear Unit (ReLU)

Once an activation function is chosen, its discretization defined by (4.4) and (4.5) can be simplified. Due to the piecewise linear nature of ReLU, we consider it first and build intuition as we

go. Let $x^{(q,0)}$ denote the $q^{\text{th}}$ point in the input space where $x^{(q,0)}$ gets mapped to $x^{(q,i)}$ and $\tilde{x}^{(q,i)}$ in the $i^{\text{th}}$ layer of the neural network before and after the activation function is applied, respectively. Let $x^{(\mathscr{Q},0)}$ denote the particular set of points in the input space $\{x^{(q,0)} : q \in \mathscr{Q}\}$. As an illustrative example, consider the rectified linear unit (ReLU) activation function

$$\text{ReLU}(x) = \max\{x, 0\} = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{4.10}$$

which is applied component-wise if $x$ is multi-dimensional. Say that the $i^{\text{th}}$ layer uses this activation function, so $f_i$ =ReLU. Consider the two cases presented in (4.5).

Case 1: If $x_j^{(q,i)} \neq 0$, split into two subcases: either (a) $x_j^{(q,i)} > 0$ or (b) $x_j^{(q,i)} < 0$. If (a) holds, then $f_i\big(x_j^{(q,i)}\big) = x_j^{(q,i)}$, which implies that $A_{j,j}^{(q,i)} = 1$. Otherwise, (b) holds and $f_i\big(x_j^{(q,i)}\big) = 0$ and $A_{j,j}^{(q,i)} = 0$. For both (a) and (b), $a_j^{(q,i)} = 0$.

Case 2: If $x_j^{(q,i)} = 0$, then $A_{j,j}^{(q,i)} = 0$ and $a_j^{(q,i)} = f_i\big(x_j^{(q,i)}\big) = 0$.

Thus, $a_j^{(q,i)} = 0 \; \forall x^{(q,0)}$, so we can restate (4.4) and (4.5) as

$$f_i\big(x^{(q,i)}\big) = A^{(q,i)} x^{(q,i)} \tag{4.11}$$

where

$$\begin{cases} A_{j,j}^{(q,i)} = 1 & \text{if } x_j^{(q,i)} > 0 \\ A_{j,j}^{(q,i)} = 0 & \text{otherwise} \end{cases} \qquad \text{for } j = 1, \ldots, h_i,\ i = 1, \ldots, L,\ q \in \mathbb{N}. \tag{4.12}$$

Then (4.11) and (4.12) implicitly show that the collection $x^{(\mathscr{K},0)}$ where

$$\begin{aligned} x^{(\mathscr{K},0)} = \big\{ x^{(k_j,0)} : \quad & x^{(k_1,i)} > 0 \implies x^{(k_2,i)} > 0, k_1 \neq k_2 \text{ and} \\ & x^{(k_1,i)} \leq 0 \implies x^{(k_2,i)} \leq 0, k_1 \neq k_2,\ \forall k_1, k_2 \in \mathscr{K} \big\} \end{aligned} \tag{4.13}$$

get treated the same by the neural network at ReLU layer $i$. In this way, the ReLU layer implicitly bins together nonnegative values and those that are negative. We make this binning explicit by introducing a label of '1' for the bin of nonnegative values, a label of '0' for the bin of negative

values, and a *binning vector*, $s^{(q,i)} \in \mathbb{R}^{h_i}$, indicating the component-wise binning of $\tilde{x}^{(q,i)}$ where

$$s_j^{(q,i)} = \begin{cases} 1 & \text{if } \tilde{x}_j^{(q,i)} > 0 \\ 0 & \text{if } \tilde{x}_j^{(q,i)} = 0 \end{cases} \tag{4.14}$$

or

$$s_j^{(q,i)} = \begin{cases} 1 & \text{if } x_j^{(q,i)} > 0 \\ 0 & \text{if } x_j^{(q,i)} \leq 0. \end{cases} \tag{4.15}$$

In the particular case of ReLU or any other activation function from which two bins emerge, we synonymously call these binning vectors, *binary vectors* or *bit vectors*. Without loss of generality, we demonstrate in Figure 4.1 what happens when we take the $j^{\text{th}}$ point, $x^{(j,0)}$ as input and have ReLU as the activation function of the first layer.

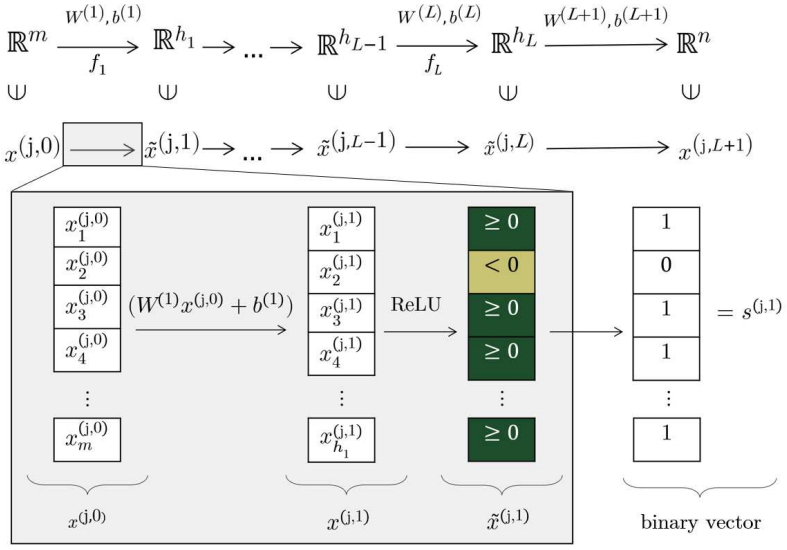



**Figure 4.1:** The process by which bit vectors are constructed for layers of a neural network that use the ReLU activation function.

Notice that $s_j^{(q,i)} = A_{j,j}^{(q,i)}$, so

$$\begin{aligned} f_i\big(x^{(q,i)}\big) = A^{(q,i)}x^{(q,i)} &= \operatorname{diag}\big(s^{(q,i)}\big)x^{(q,i)} \\ &= \operatorname{diag}\big(s^{(q,i)}\big)\big(W^{(i)}\tilde{x}^{(q,i-1)} + b^{(i)}\big) \\ &= \operatorname{diag}\big(s^{(q,i)}\big)W^{(i)}\tilde{x}^{(q,i-1)} + \operatorname{diag}\big(s^{(q,i)}\big)b^{(i)}. \end{aligned} \tag{4.16}$$

Also using the fact that $s_j^{(q,i)} = A_{j,j}^{(q,i)}$, we can restate (4.8), the mapping given by a FFNN, more explicitly for specifically a ReLU FFNN as

$$\begin{aligned} x^{(q,L+1)} = F\big(x^{(q,0)}\big) =& W^{(L+1)}\operatorname{diag}\big(s^{(q,L)}\big)W^{(L)}\ldots\operatorname{diag}\big(s^{(q,1)}\big)W^{(1)}x^{(0)} + \\ & W^{(L+1)}\operatorname{diag}\big(s^{(q,L)}\big)W^{(L)}\ldots\operatorname{diag}\big(s^{(q,1)}\big)b^{(1)} + \\ & W^{(L+1)}\operatorname{diag}\big(s^{(q,L)}\big)W^{(L)}\ldots\operatorname{diag}\big(s^{(q,2)}\big)b^{(2)} + \ldots + \\ & W^{(L+1)}\operatorname{diag}\big(s^{(q,L)}\big)b^{(L)} + b^{(L+1)} \qquad (4.17a) \\ =:& Ax^{(0)} + B \qquad (4.17b) \end{aligned}$$

where (4.17a) and (4.17b) are analogous to the definitions given by (3) in [64] and (3.3) in [57], respectively. We also give this as an iterative algorithm in Algorithm 2 for simplicity of implementation. This algorithm also highlights the significance of bit vectors in defining polyhedra's affine mapping from a neural network's input space to its output space: they are the *only* difference between any two distinct mappings. We can stack these $s^{(q,i)}$ to form a *composite binning vector* (also called the *activation pattern* in [60]), $s^{(q)} = \left[s^{(q,1)\top}, \ldots, s^{(q,L)\top}\right]^\top \in \mathbb{R}^h$, that describes the path that $x^{(q,0)}$ took through the neural network. If the neural network of choice only uses ReLU as its activation function, then this composite vector describes the firing pattern of the network's neurons when $x^{(q,0)}$ is taken as input, also noted in [65]. The neural network's input space, $\mathbb{R}^m$, can be partitioned into convex polytopes where the points belonging to each polytope have the same corresponding composite binning vector. Note that because these composite bit vectors are made up of an $h$-long combination of zeros or ones, there are $2^h$ possible

**Algorithm 2** Layer-wise outputs of a ReLU feed-forward neural network

Given a neural network $F : \mathbb{R}^m \to \mathbb{R}^n$ with layer-wise weights $W^{(i)}$ and biases $b^{(i)}$ for $i = 1, ..., L+1$, choose $x^{(0)} \in \mathbb{R}^m$ with the layer-wise binary vector $s^{(i)}$ for $i = 1, ..., L$ defined by the firing pattern of $x^{(0)}$ through $F$.

**Initialize** $\hat{W}^{(0)} = I_{m\times m},\ s^{(0)} = 1_{m\times 1},\ \hat{b}^{(0)} = 0_{m\times 1}$.

**for** $i = 1, \ldots, L$ **do**

$x^{(i)} = W^{(i)}\hat{W}^{(i-1)}x^{(0)} + W^{(i)}\hat{b}^{(i-1)} + b^{(i)}$

$\hat{W}^{(i)} = \text{diag}(s^{(i)})W^{(i)}\hat{W}^{(i-1)}$

$\hat{b}^{(i)} = \text{diag}(s^{(i)})W^{(i)}\hat{b}^{(i-1)} + \text{diag}(s^{(i)})b^{(i)}$

**end for**

$x^{(L+1)} = W^{(L+1)}x^{(L)} + b^{(L+1)}$

bit vectors a model can produce. Therefore, we say that the neural network has finite 'polyhedral resources' that it can use to partition the input space into convex regions. More explicitly, the points belonging to a polytope defined by a particular composite binning vector satisfy the system

$$\text{diag}\big(s'^{(q)}\big)\hat{W}^{(q,L)}x^{(q,0)} \leq \text{diag}\big(s'^{(q,i)}\big)\hat{b}^{(q,L)} \quad \text{where} \quad s_j'^{(q,i)} = \begin{cases} 1 & \text{if } s_j^{(q,i)} = 0 \\ -1 & \text{if } s_j^{(q,i)} = 1 \end{cases}$$

$$\begin{aligned} \hat{W}^{(q,i)} &= W^{(i)}A^{(q,i-1)}\hat{W}^{(q,i-1)} \\ &= W^{(i)}\text{diag}\big(s^{(q,i-1)}\big)\hat{W}^{(q,i-1)}, \\ \hat{b}^{(q,i)} &= W^{(i)}\big(A^{(q,i-1)}\hat{b}^{(q,i-1)} + a^{(q,i-1)}\big) + b^{(i)} \\ &= W^{(i)}\text{diag}\big(s^{(q,i-1)}\big)\hat{b}^{(q,i-1)} + b^{(i)} \end{aligned}$$

and we initialize $\hat{W}^{(q,0)} = I_{m\times m}, s^{(q,0)} = 1_{m\times 1}, \hat{b}^{(q,0)} = 0_{m\times 1}$ so that $\hat{W}^{(q,1)} = W^{(1)}$ and $\hat{b}^{(q,1)} = b^{(1)}$. Therefore, for any polyhedron of a polyhedral decomposition, there exists an $A$ and $c$ where $A = \big[A_1, A_2, \ldots, A_h\big]^\top$ and $c = \big[c_1, c_2, \ldots, c_h\big]^\top$ with $A_i \in \mathbb{R}^{m\times 1}$ and $c_i \in \mathbb{R}$ (explicitly formulated above)

such that the polyhedron can be defined as the set of points in the input space satisfying $Ax \leq c$ or $A_i^\top x \leq c_i$ for $i = 1, \ldots, h$.

This formulation of each polyhedron of a polyhedral decomposition as a system of linear inequalities highlights their consequential convexity since the feasible region of every linear program is convex. Also, we note that each bit of the bit vector indicates whether the input point of interest satisfies a particular linear inequality. This embeds in bit vectors rich geometric information that we explain and harness in Section 5.2.1. We can also view bit vectors as a code summarizing the neural network's response given a particular input, codes whose similarities can be quantified with the *Hamming distance*. The Hamming distance $H\left(s^{(a)}, s^{(b)}\right)$ is equivalent to counting the number of bits in which $s^{(a)}$ and $s^{(b)}$, the bit vectors of $x^{(a)}$ and $x^{(b)}$, differ. We say that $H\left(s^{(a)}, s^{(b)}\right) = H\left(x^{(a)}, x^{(b)}\right)$ for simplicity. We will use this interpretation of bit vectors in Section 5.2.2. Therefore, both interpretations are useful, which we will demonstrate with applications. In essence, input space decomposition is precisely what FFNNs do. Figure 4.2 depicts the polyhedral decomposition of $\mathbb{R}^2$ given the trained neural network $F : \mathbb{R}^2 \to \mathbb{R}^3 \to \mathbb{R}^3 \to \mathbb{R}$ trained in the unit square about the origin to approximate the function $x^2 + y^2 - \frac{2}{3}$. The figure is restricted to the square of side length of six centered about the origin for visualization purposes only; the peripheral polyhedra extend infinitely. Because of the linearity of the ReLU activation function, these composite binning vectors exactly describe the path their corresponding points took through the neural network, and therefore, these partitionings also precisely characterize their ReLU FFNNs. In contrast, the partitionings from neural networks using continuously differentiable activation functions are approximate.

**Parametric ReLU (PReLU)**

We next consider the Parametric ReLU (PReLU) activation function due to its similarity to ReLU.

$$\text{PReLU}(x) = \max\{x, 0\} + \alpha \min\{0, x\} = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{4.18}$$

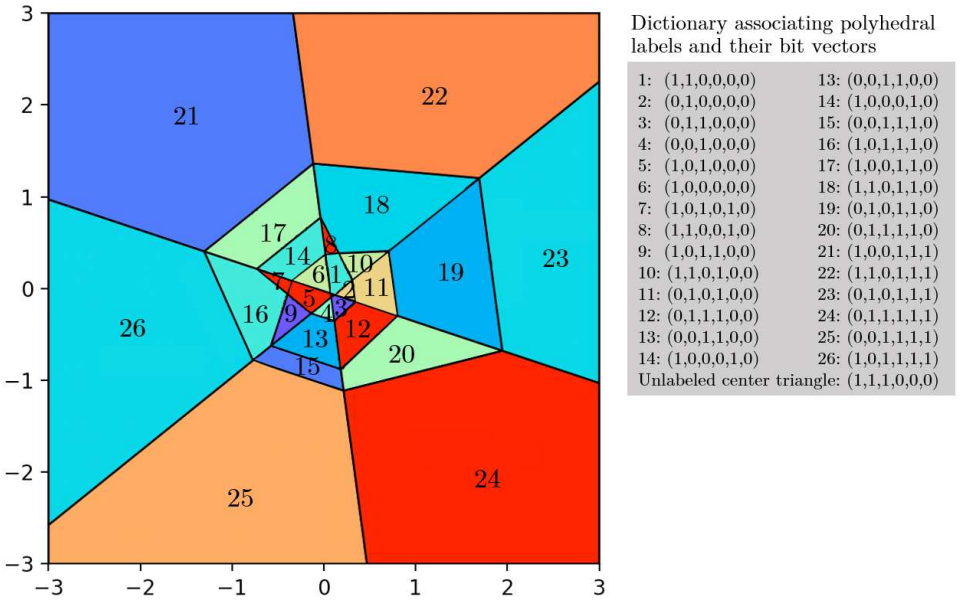



**Figure 4.2:** The polyhedral decomposition of the ReLU FFNN $F : \mathbb{R}^2 \to \mathbb{R}^3 \to \mathbb{R}^3 \to \mathbb{R}$ after training in the unit square with each polyhedron labeled by its binary vector.

where $\alpha$ is a tuneable parameter. Leaky ReLU (LReLU) is the specific PReLU function where $\alpha = 0.01$. Naturally, we define the binning vector identically as we did for ReLU. The points belonging to a polytope defined by a particular composite binning vector satisfy the system

$$\alpha \operatorname{diag}\left(s'^{(q)}\right)\hat{W}^{(q,L)}x^{(q,0)} \leq s'^{(q,i)} \odot \hat{b}^{(q,L)} \tag{4.19}$$

where $\odot$ denotes the Hadamard product.

### Sigmoid

The Sigmoid function is defined

$$\text{Sigmoid}(x) = S(x) = \frac{e^x}{e^x + 1}. \tag{4.20}$$

While the binning vectors for ReLU encode the firing pattern of the network's neurons, those for sigmoid encode the magnitude at which the neurons fired or the component-wise percentage of layer outputs passed on to the next layer because $0 < S(x) < 1 \ \forall x \in \mathbb{R}$. Choosing $f_i = S$ and

defining $g_1(x) = \frac{1}{x}S(x) = \frac{e^x}{xe^x+x}$, we reformulate (4.4) and (4.5) as

$$f_i\left(x^{(q,i)}\right) = A^{(q,i)}x^{(q,i)} + a^{(q,i)} \tag{4.21}$$

where $A^{(q,i)} \in \mathbb{R}^{h_{i+1}\times h_i}$ is a diagonal matrix, $a^{(q,i)} \in \mathbb{R}^{h_{i+1}\times 1}$, and

$$\begin{cases} A_{j,j}^{(q,i)} = g_1\left(x_j^{(q,i)}\right),\ a_j^{(q,i)} = 0 & \text{if } x_j^{(q,i)}e^{x_j^{(q,i)}} + x_j^{(q,i)} \neq 0 \\ A_{j,j}^{(q,i)} = 0, \qquad a_j^{(q,i)} = S\left(x_j^{(q,i)}\right) & \text{otherwise.} \end{cases} \tag{4.22}$$

We have that $x_j^{(q,i)}e^{x_j^{(q,i)}} + x_j^{(q,i)} = x_j^{(q,i)}\left(e^{x_j^{(q,i)}} + 1\right)$ has two roots: at $x_j^{(q,i)}$ that satisfy (a) $x_j^{(q,i)} = 0$ or (b) $e^{x_j^{(q,i)}} + 1 = 0$, but (b) only holds if $x_j^{(q,i)} = \pi\mathrm{i} \in \mathbb{C}$. We know that $x_j^{(q,i)} \in \mathbb{R}\ \forall i, j, q$, so only (a) can hold and we reformulate (4.22) as

$$\begin{cases} A_{j,j}^{(q,i)} = g_1\left(x_j^{(q,i)}\right),\ a_j^{(q,i)} = 0 & \text{if } x_j^{(q,i)} \neq 0 \\ A_{j,j}^{(q,i)} = 0, \qquad a_j^{(q,i)} = \frac{1}{2} & \text{otherwise} \end{cases}. \tag{4.23}$$

Now the layerwise outputs can be expressed

$$x^{(q,i)} = \hat{W}^{(q,i)}x^{(q,0)} + \hat{b}^{(q,i)} \tag{4.24}$$

with $\hat{W}^{(q,i)} = W^{(i)}A^{(q,i-1)}\hat{W}^{(q,i-1)}$, $\hat{b}^{(q,i)} = W^{(i)}A^{(q,i-1)}\hat{b}^{(q,i-1)} + W^{(i)}a^{(q,i-1)} + b^{(i)}$, and we initialize $A^{(q,0)} = I_{m\times m}$, $\hat{W}^{(q,0)} = I_{m\times m}, \hat{b}^{(q,0)} = 0_{m\times 1}$, $a^{(q,0)} = 0_{m\times 1}$ so that $\hat{W}^{(q,1)} = W^{(1)}$ and $\hat{b}^{(q,1)} = b^{(1)}$.

As shown in Figure 4.3, we define the binning vector

$$s_j^{(q,i)} = \begin{cases} 2 & \text{if } x_j^{(q,i)} > \xi \\ 1 & \text{if } \xi \geq x_j^{(q,i)} > -\xi \\ 0 & \text{if } -\xi \geq x_j^{(q,i)} \end{cases} \tag{4.25}$$

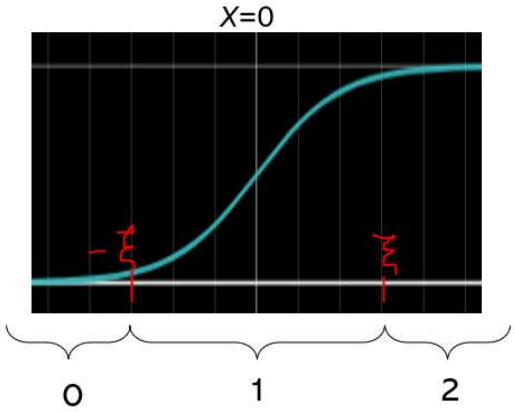



**Figure 4.3:** The Sigmoid activation function and a demonstration of how an associated binning vector can be defined.

or equivalently

$$s_j^{(q,i)} = \begin{cases} 2 & \text{if } -x_j^{(q,i)} < -\xi \\ 1 & \text{if } x_j^{(q,i)} \leq \xi \text{ and} \\ & \quad -x_j^{(q,i)} < \xi \\ 0 & \text{if } x_j^{(q,i)} \leq -\xi \end{cases} \tag{4.26}$$

and the composite binning vector

$$s_j^{(q)} = \begin{cases} 2 & \text{if } -\left(\hat{W}^{(q,L)}\right)_{j,1:m} \cdot x^{(q,0)} < -\xi + \hat{b}_j^{(q,L)} \\ 1 & \text{if } \left(\hat{W}^{(q,L)}\right)_{j,1:m} \cdot x^{(q,0)} < \xi - \hat{b}_j^{(q,L)} \\ & \quad \text{and } -\left(\hat{W}^{(q,L)}\right)_{j,1:m} \cdot x^{(q,0)} < \xi + \hat{b}_j^{(q,L)} \\ 0 & \text{if } \left(\hat{W}^{(q,L)}\right)_{j,1:m} \cdot x^{(q,0)} < -\xi - \hat{b}_j^{(q,L)} \end{cases} \tag{4.27}$$

where $\xi$ is a parameter tuned to produce sufficiently many distinct binning vectors. After experimentation with our particular network, we chose $\xi = 1$ because it provided sufficient distinct binning vectors. Define $s'^{(q)} = \left[s_1'^{(q)\top}, \ldots, s_h'^{(q)\top}\right]^\top \in \mathbb{R}^{2h\times 1}$ and $s''^{(q)} = \left[s_1''^{(q)\top}, \ldots, s_h''^{(q)\top}\right]^\top \in$

$\mathbb{R}^{2h\times(h+1)}$ where

$$s_j'^{(q)} = \begin{cases} [-1,0]^\top & \text{if } s_j^{(q)} = 2 \\ [1,-1]^\top & \text{if } s_j^{(q)} = 1 \\ [1,0]^\top & \text{if } s_j^{(q)} = 0 \end{cases} \tag{4.28}$$

and

$$s_j''^{(q)} = 0_{2\times(h+1)} \text{ with } \begin{cases} \left(s_j''^{(q)}\right)_{1:2,1} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \text{ and } \left(s_j''^{(q)}\right)_{1,j} = 1 & \text{if } s_j^{(q)} = 2 \\ \left(s_j''^{(q)}\right)_{1:2,1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } \left(s_j''^{(q)}\right)_{1:2,j} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} & \text{if } s_j^{(q)} = 1 \\ \left(s_j''^{(q)}\right)_{1:2,1} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \text{ and } \left(s_j''^{(q)}\right)_{1,j} = -1 & \text{if } s_j^{(q)} = 0. \end{cases} \tag{4.29}$$

Given a particular composite binning vector $s^{(q)}$, the points of the input space, $x^{(q,0)}$, whose path through the network it describes satisfy the $2h$ equations in the following system

$$\operatorname{diag}\left(s'^{(q)}\right) \begin{bmatrix} \left(\hat{W}^{(L)}\right)_{1,1:m} \\ \left(\hat{W}^{(L)}\right)_{1,1:m} \\ \vdots \\ \left(\hat{W}^{(L)}\right)_{h,1:m} \\ \left(\hat{W}^{(L)}\right)_{h,1:m} \end{bmatrix} x^{(q,0)} \leq s''^{(q)} \begin{bmatrix} \xi \\ \hat{b}^{(L)} \end{bmatrix}. \tag{4.30}$$

This however is not a linear set of inequalities if $L > 1$ because $\hat{W}^{(L)}$ and $\hat{b}^{(L)}$ depend on $x^{(q,0)}$ in this case, deriving their dependence from $A^{(q,L)}$ and $a^{(q,L)}$. To formulate (4.30) as a linear system,

we define $\bar{S}(x)$, a linear approximation of $S(x)$ :

$$\bar{S}(x) = \begin{cases} 1 & \text{if } x > \xi \\ \xi' x & \text{if } \xi \geq x > -\xi \\ 0 & \text{if } -\xi \geq x \end{cases} = \bar{A}x + \bar{a} \tag{4.31}$$

where

$$\xi' = \frac{S(\xi) - S(-\xi)}{2\xi}, \bar{A}_{j,j} = \begin{cases} 0 & \text{if } x_j > \xi \\ \xi' & \text{if } \xi \geq x_j > -\xi \\ 0 & \text{if } -\xi \geq x_j \end{cases}, \bar{a}_j = \begin{cases} 1 & \text{if } x_j > \xi \\ \frac{1}{2} & \text{if } \xi \geq x_j > -\xi \\ 0 & \text{if } -\xi \geq x_j \end{cases}.$$

Now the composite binning vector, $\bar{s}^{(q)}$, and utility vectors $\bar{s}'^{(q)}$ and $\bar{s}''^{(q)}$ are defined by slightly modifying (4.27), (4.28), and (4.29):

$$\bar{s}_j^{(q)} = \begin{cases} 2 & \text{if } -\left(\bar{W}^{(q,L)}\right)_{j,1:m} \cdot x^{(q,0)} < -\xi + \bar{b}_j^{(q,L)} \\ 1 & \text{if } \left(\bar{W}^{(q,L)}\right)_{j,1:m} \cdot x^{(q,0)} < \xi - \bar{b}_j^{(q,L)} \\ & \text{and } -\left(\bar{W}^{(q,L)}\right)_{j,1:m} \cdot x^{(q,0)} < \xi + \bar{b}_j^{(q,L)} \\ 0 & \text{if } \left(\bar{W}^{(q,L)}\right)_{j,1:m} \cdot x^{(q,0)} < -\xi - \bar{b}_j^{(q,L)} \end{cases} \tag{4.32}$$

where $\bar{W}^{(q,i)} = W^{(i)}\bar{A}^{(q,i-1)}\bar{W}^{(q,i-1)}, \quad \bar{b}^{(q,i)} = W^{(i)}\bar{A}^{(q,i-1)}\bar{b}^{(q,i-1)} + W^{(i)}\bar{a}^{(q,i-1)} + b^{(i)}, \quad$ and we initialize $\quad \bar{A}^{(q,0)}, \bar{W}^{(q,0)}, \bar{b}^{(q,0)}, \bar{a}^{(q,0)}$ just as we initialized $A^{(q,0)}, \hat{W}^{(q,0)}, \hat{b}^{(q,0)},$ and $a^{(q,0)},$

$$\bar{s}_j'^{(q)} = \begin{cases} [-1,0]^\top & \text{if } s_j^{(q)} = 2 \\ [1,-1]^\top & \text{if } s_j^{(q)} = 1 \\ [1,0]^\top & \text{if } s_j^{(q)} = 0, \end{cases} \tag{4.33}$$

and

$$\bar{s}_j'^{(q)} = 0_{2\times(h+1)} \text{ with } \begin{cases} \left(\bar{s}_j'^{(q)}\right)_{1:2,1} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \text{ and } \left(\bar{s}_j'^{(q)}\right)_{1,j} = 1 & \text{if } \bar{s}_j^{(q)} = 2 \\ \left(\bar{s}_j'^{(q)}\right)_{1:2,1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } \left(\bar{s}_j'^{(q)}\right)_{1:2,j} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} & \text{if } \bar{s}_j^{(q)} = 1 \\ \left(\bar{s}_j'^{(q)}\right)_{1:2,1} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \text{ and } \left(\bar{s}_j'^{(q)}\right)_{1,j} = -1 & \text{if } s_j^{(q)} = 0. \end{cases} \tag{4.34}$$

So (4.30) can be linearly approximated by

$$\operatorname{diag}\left(\bar{s}'^{(q)}\right) \begin{bmatrix} \left(\bar{W}^{(L)}\right)_{1,1:m} \\ \left(\bar{W}^{(L)}\right)_{1,1:m} \\ \vdots \\ \left(\bar{W}^{(L)}\right)_{h,1:m} \\ \left(\bar{W}^{(L)}\right)_{h,1:m} \end{bmatrix} x^{(q,0)} \leq \bar{s}'^{(q)} \begin{bmatrix} \xi \\ \bar{b}^{(L)} \end{bmatrix}. \tag{4.35}$$

**Choice of Activation Function**

We now compare a few of the most commonly used activation functions introduced earlier in this section. Historically, the Sigmoid activation function was the default choice in the 1990s. However, very early in training after random initialization, the sigmoid activation values in the last layer approach their lower saturation value of 0 while those in the remaining layers have a mean above 0.5, increasing in value as the layer number increases [66]. This saturation problem in the last layer makes the network's predictive task difficult. We can restate this problem through the new lens of polyhedral decompositions using binning vectors: the partitioning of the input space defined by a neural network using the Sigmoid activation function would be relatively coarse. A collection of data takes fewer distinct paths through the network than with symmetric activation

functions. This prevents gradients from flowing backward, preventing earlier layers from learning useful features [66]. This has also been observed with the ReLU activation function [1]. The authors constructed representational dissimilarity matrices (RDMs) between two classes of images (tench and thunder snake) from the ImageNet dataset using Hamming distance as their distance metric. As shown in Figure 4.4, the network can only begin to distinguish between the two classes (to some low degree) by the 9$^{th}$ ReLU layer of ResNet-50 pretrained on ImageNet-1K. [67] showed

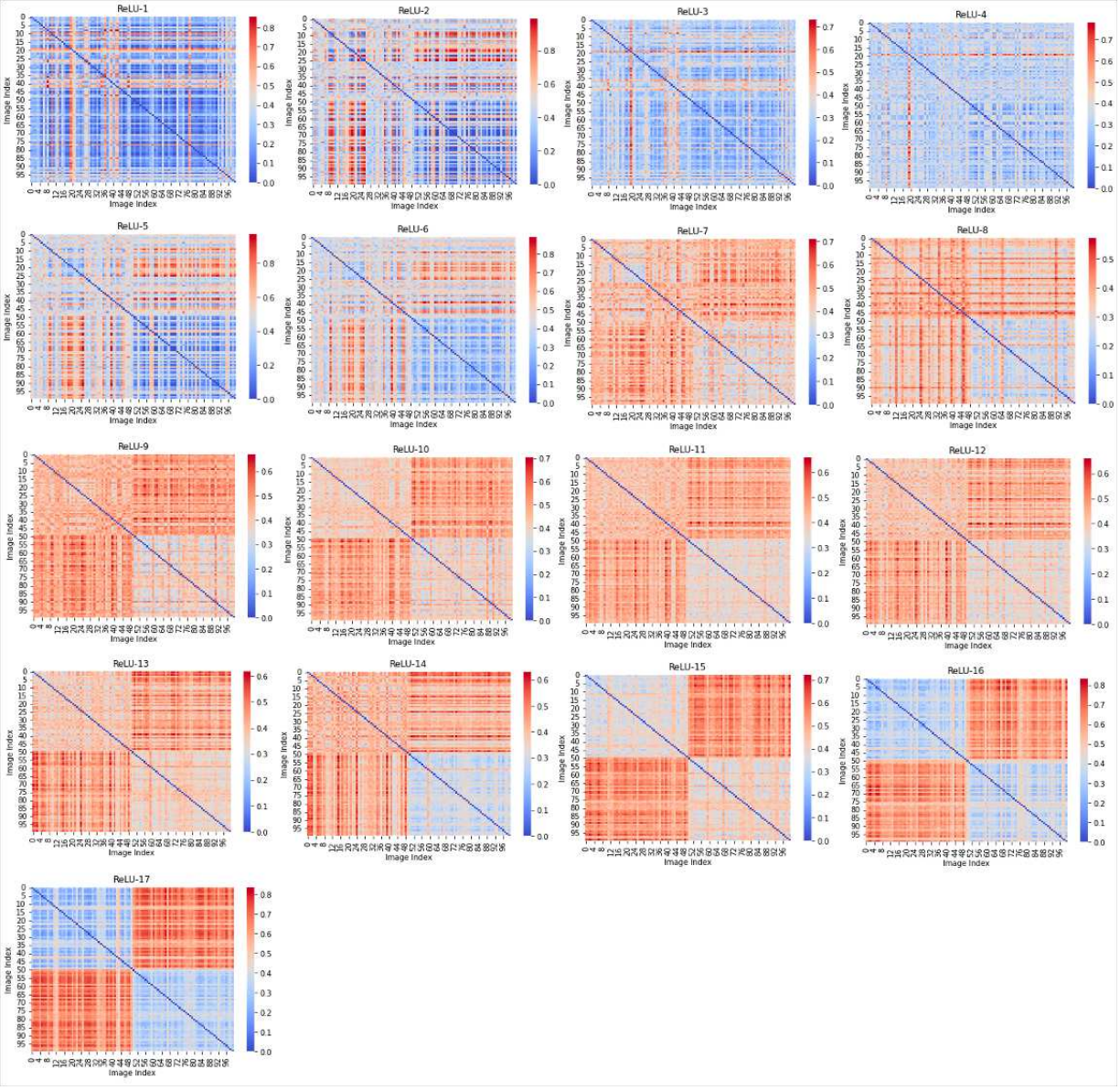



**Figure 4.4:** RDM for Tench and Thunder Snake on the test data at Layers 1-17 [1].

that symmetric variants of the Sigmoid function, like hyperbolic tangent (Tanh), tend to converge faster. By the early 2000s, Tanh replaced sigmoid as the default activation function. Both are susceptible to the vanishing gradient problem, which can thwart model training altogether. Despite this, it was actually the computational expense of evaluating Tanh for layerwise outputs of a neural network, an expense that grows with network size which has exploded in recent years, that led the authors of [67] to suggest using an approximation of Tanh defined by a ratio of polynomials. One such approximation, Softsign $\left(\text{Softsign}(x) = \frac{x}{1+|x|}\right)$, converges more slowly to its asymptotes at 1 and $-1$, which helps avoid the saturation behavior seen with Sigmoid and Tanh [66].

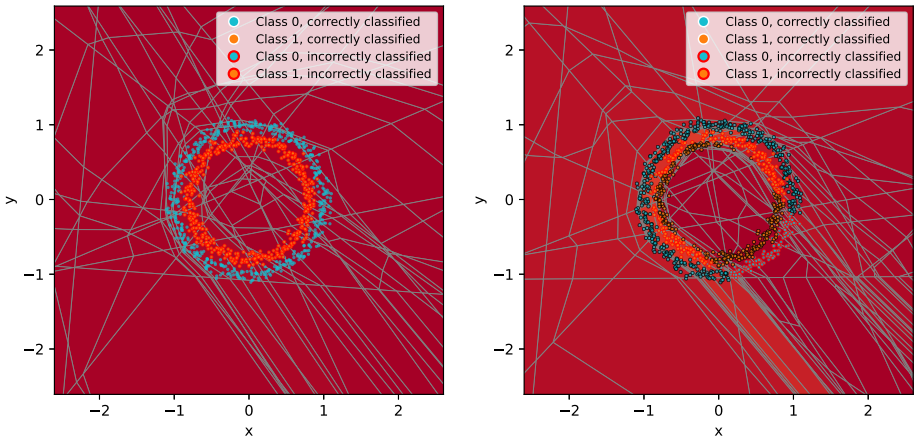
ReLU is currently the recommended default activation function and is, therefore, our focus for the remainder of this work. Their similarity to linear units make them easy to optimize. The image of half of their domain is 0, keeping the derivatives large for active nodes. Gradients are large and consistent, avoiding the vanishing gradient problem. Where ReLU is differentiable ($\forall x \in \mathbb{R}, x \neq 0$) the second derivative is 0, and its derivative for $x \geq 0$ is 1. Therefore, no second-order effects are introduced and the gradient direction is more useful for training [68]. As a big bonus toward making the work contained herein feasible, its bimodal nature allows its binning vectors to be represented with boolean elements, providing significant speed-ups and memory savings. In addition to the saturation problem previously mentioned, a drawback of the ReLU function is that they cannot learn via gradient-based methods on examples that do not activate the network's neurons. Generalizations of ReLU have been defined which overcome this by guaranteeing a gradient for all nodes [68–72]. Some work has been done to show that the saturation problem is not necessarily inherent to activation functions but could be attributed to poor model initialization.

## 4.3 Polyhedral Decompositions During Training

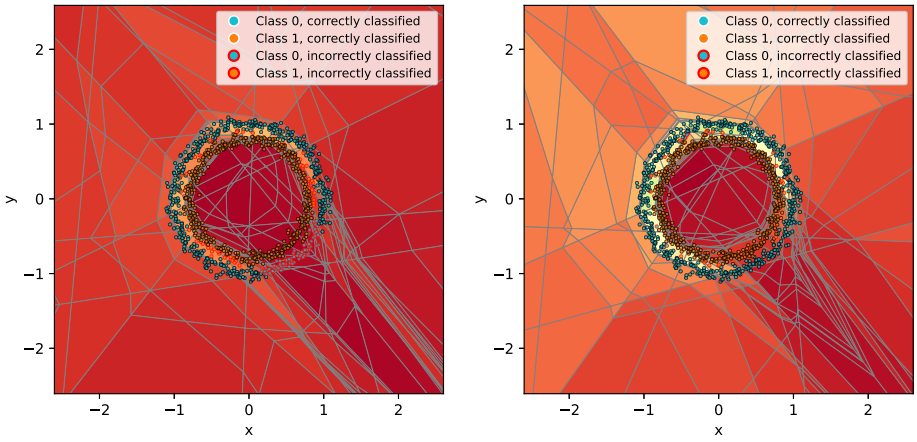
In this section, we conduct an observational study on how polyhedral decompositions morph during the learning process of their neural networks. Such visualization techniques appear to be very new in the field [43], so it is no surprise that leveraging them to understand the training process led to novel insights. Training can be thought of as an allocation procedure of 'polyhedral' and

'Jacobian resources' where they are suspected to help decrease the network's error rate with respect to its loss function. Continuing with this metaphorical language to best depict what is happening in very high dimensions, the training process can be more specifically thought of as the neural network developing a water pipe system. The pipes run along decision boundaries and carry input-output Jacobian norm resources along them instead of water and are encased by a dense layer of polytopes rather than metal or plastic. The state inside of tubes can be quite volatile, mirroring the neural network's prediction of multiple classes there, while the outside remains relatively stable. Analogous to observing a lengthwise cross-section of a water-filled tube would look like a strip of water outlined by the pipe material, our two-dimensional visualizations show that polyhedral refinement tends to outline regions of high input-output Jacobian norms. In short, we show that polytopes containing decision boundaries tend to have relatively high input-output Jacobian norm values, therefore making the input-output Jacobian a candidate metric for prediction uncertainty, and that the polytopes enveloping the decision boundaries tend to refine.

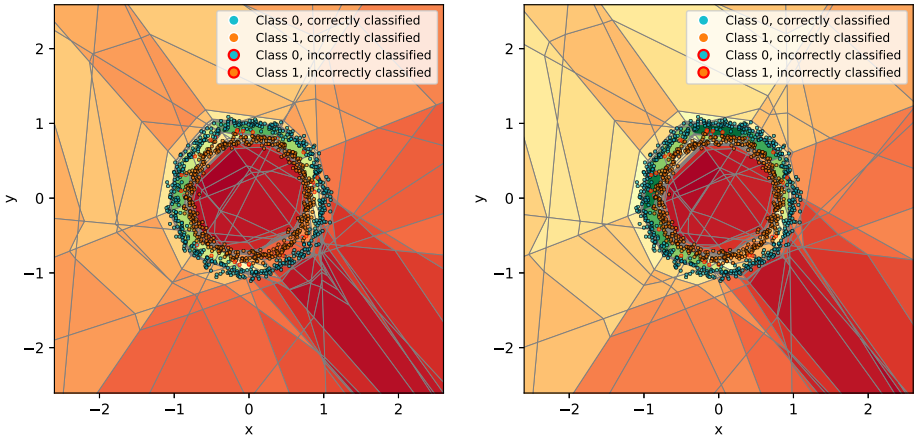
Let us now briefly define the input-output Jacobian norm of a polyhedron. We have shown in (4.17) that every ReLU FFNN input-output pair $(x,y)$ satisfies the relationship $y = Ax + B$ for a particular $A$ and $B$ explicitly given in (4.17). So the input-output Jacobian $\frac{\partial y}{\partial x}$ of a ReLU FFNN is simply $A$ (and the input-output Jacobian norm is then the norm of $A$), which has embedded in it the neuron firing patterns resulting from that particular input. We reiterate that this firing pattern is the only thing that makes these $A$ and $B$ different for each polyhedron. Therefore, the firing patterns of polyhedra with high input-output Jacobian norm, are at least approximate maximizers of $A$ from (4.17). In practice, we do not actually compute $A$, but cast the neural network as a function and take its partial derivative with respect to all the neural network variables. This makes the calculation of the input-output Jacobian free from the need or even the ability to express a particular neural network linearly.




**(a)** Epoch 22 **(b)** Epoch 34

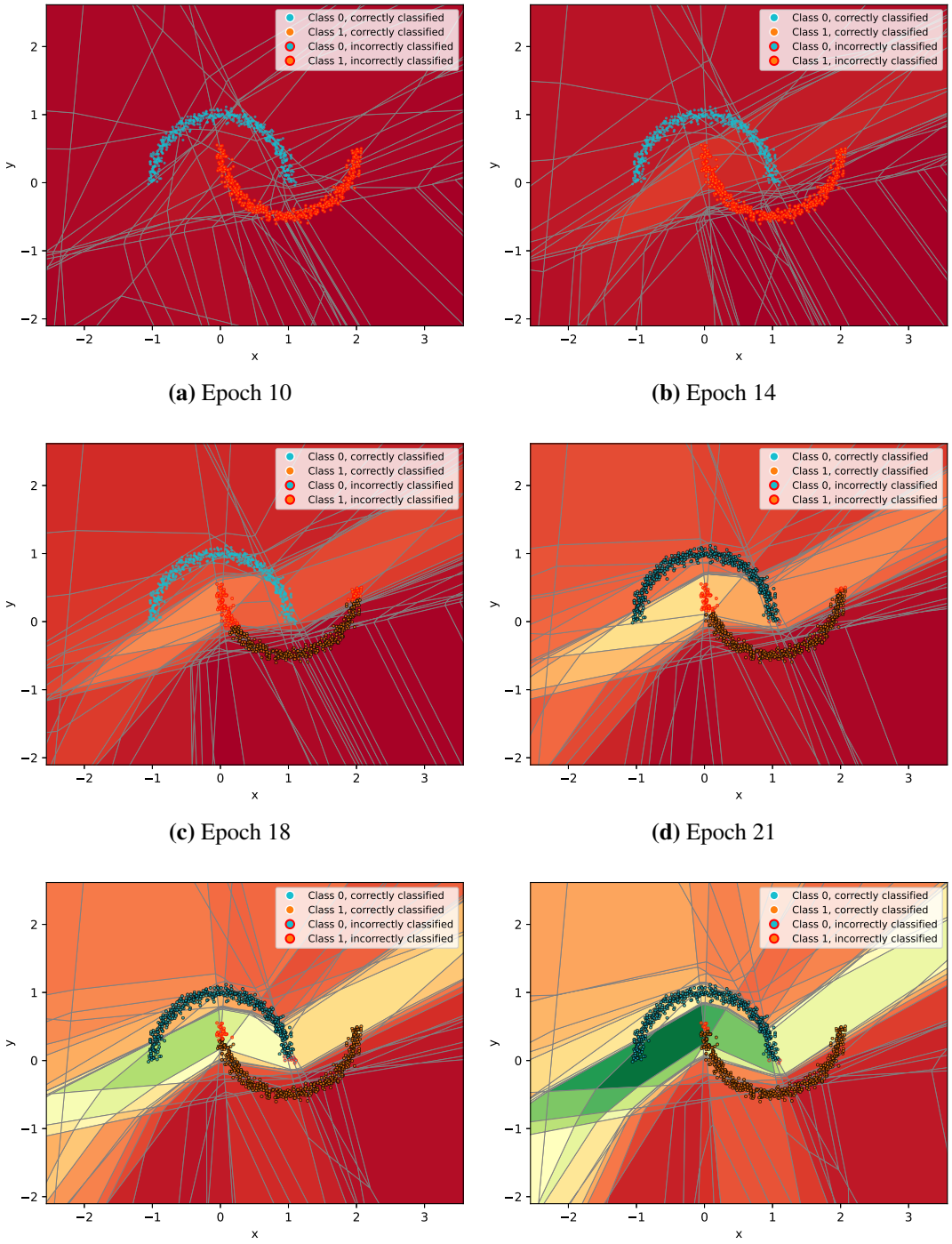



**(c)** Epoch 45 **(d)** Epoch 57




**(e)** Epoch 68 **(f)** Epoch 80

**Figure 4.5:** Polyhedral decomposition during training, colored by the input-output Jacobian norm values with a color scale between 0 and 64, until one batch of training data is perfectly predicted.




**(a)** Epoch 10

**(b)** Epoch 14

**(c)** Epoch 18

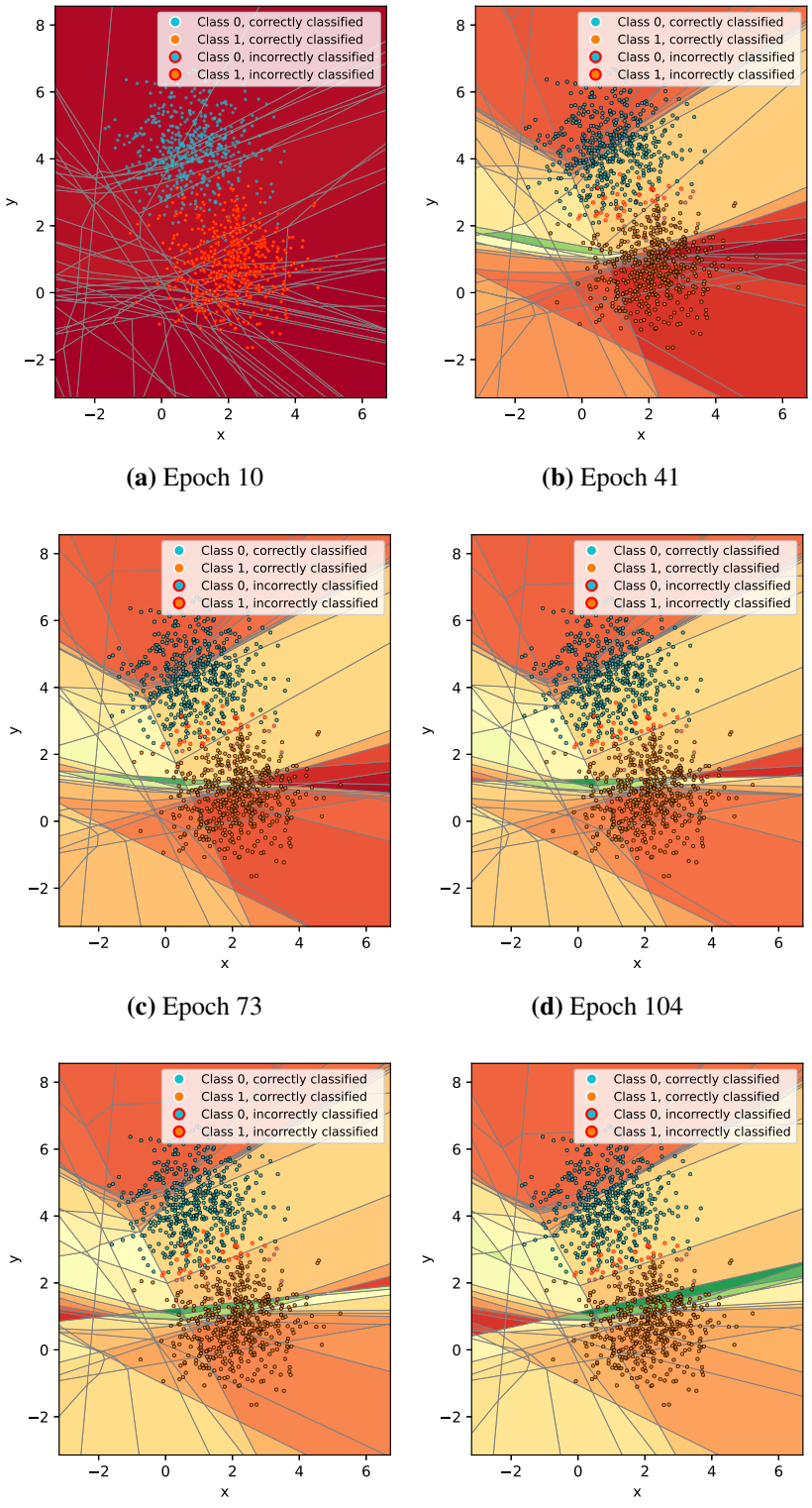
**(d)** Epoch 21

**(e)** Epoch 25

**(f)** Epoch 29

**Figure 4.6:** Polyhedral decomposition during training, colored by the input-output Jacobian norm values with a color scale between 0 and 11, trained until one batch of training data is perfectly predicted.

With this in mind, we proceed with our observational study. In Figures 4.5, 4.6, and 4.7, notice that the input-output Jacobian norm indeed increases in or close to polyhedra containing the decision boundary (which can be inferred from the locations of the correctly and incorrectly classified data) and polyhedral resources tend to be pulled toward these boundaries as training progresses. What seems to differ among the different chosen data shapes is the time at which this allocation process stops and the polyhedral decomposition stabilizes. We formally explained why the polyhedral resources are limited in Section 4.2.2. From this we can make sense of the stabilizing of the allocation of polyhedral resources: the neural network has run out of resources with which to dissect the input space in an attempt to improve prediction accuracy. Notice that the polyhedral decomposition trained with the interleaving half circles tends to stabilize in early epochs. In Figure 4.8, we see that the polyhedral decompositions at epochs 110 and 1000 look very similar to each other. On the other hand, polyhedral resources continue to condense around the decision boundary of the neural network trained on the cluster-shaped data as shown in Figure 4.9, subfigure (d) and (f). This is because more linear segments are required to approximate a function with high curvature like the line separating class 0 from class 1 for the interleaving half circles data than are needed to approximate a line that adequately separates class 0 from class 1 for the cluster data. Here we somewhat hand-wave our way through the hypothesis that the input-output Jacobian norm seems to be also a semi-limited resource of the neural network by again comparing the results from the interleaving half circles and the cluster data. Notice that the input-output Jacobian norm in Figure 4.8 quickly stabilizes, not achieving a value above 45 despite longer training on more noisy data. Meanwhile, extending the training of our network on the cluster-shaped data (despite the lack of performance improvement) grows the disparity in the input-output Jacobian norm of the polyhedra about the decision boundary and the rest of the decomposition as shown in Figure 4.9 subfigure (d) and (f). The long skinny polyhedra in subfigure (f) with high input-output Jacobian norm become so skinny that they are no longer visible in our plot but achieve values over 1200. We hypothesize that the input-output Jacobian norm ceases to increase for the polyhedra in Figure 4.8 because there are more polyhedra required to share the network's Jacobian resources (because

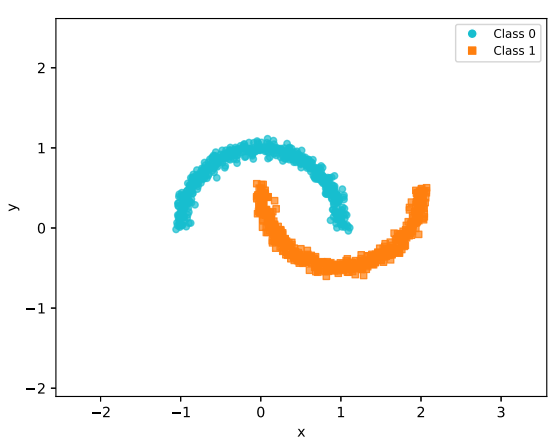



**(a)** Epoch 10 **(b)** Epoch 41

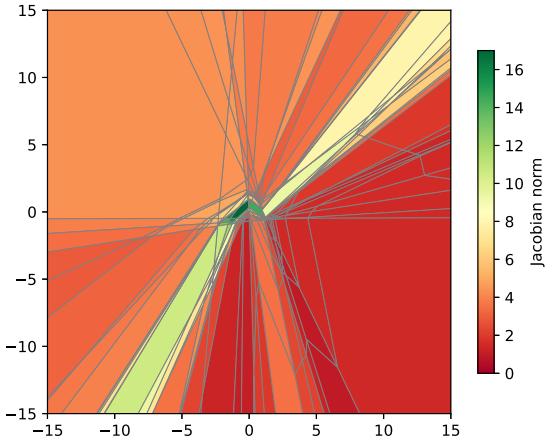
**(c)** Epoch 73 **(d)** Epoch 104

**(e)** Epoch 136 **(f)** 167

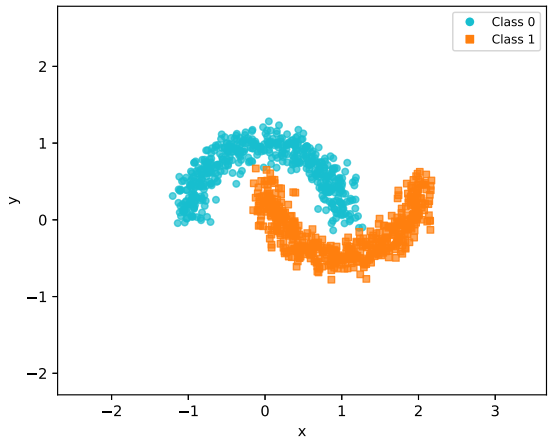
**Figure 4.7:** Polyhedral decomposition during training, colored by input-output Jacobian norm values with a color scale between 0 and 8.5, trained until one batch of training data perfectly predicted.



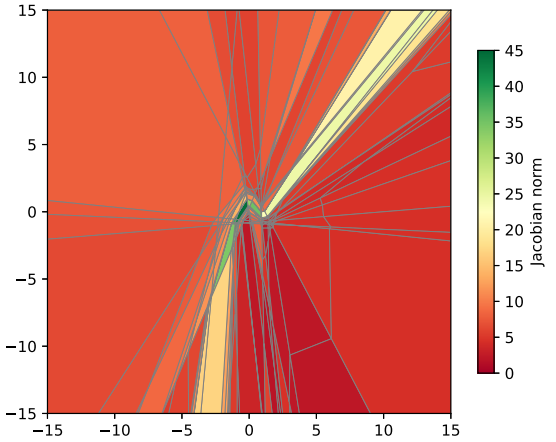

**(a)** Standard deviation = 0.05



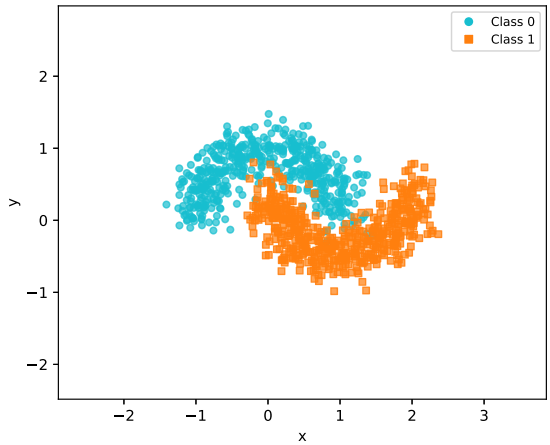

**(b)** Epoch 40 (until perfect prediction on training data attained)



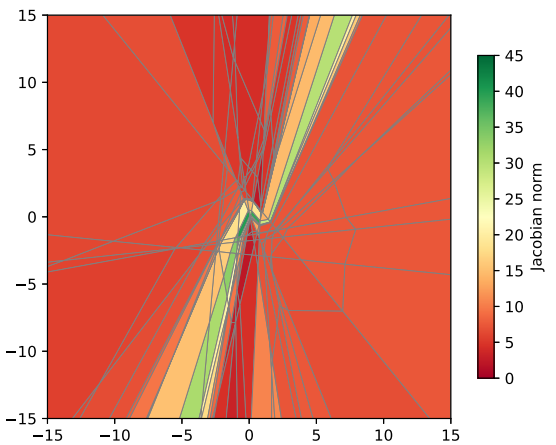

**(c)** Standard deviation = 0.12




**(d)** Epoch 110 (until perfect prediction on training data attained)

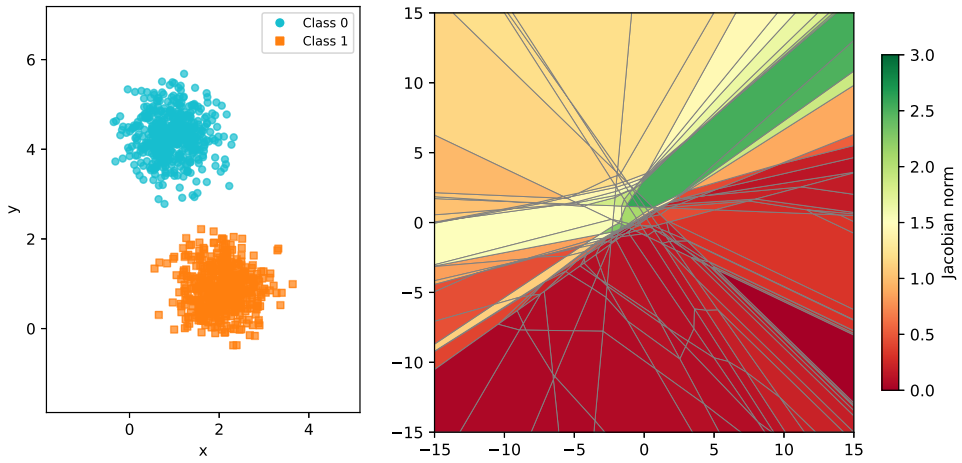



**(e)** Standard deviation = 0.20

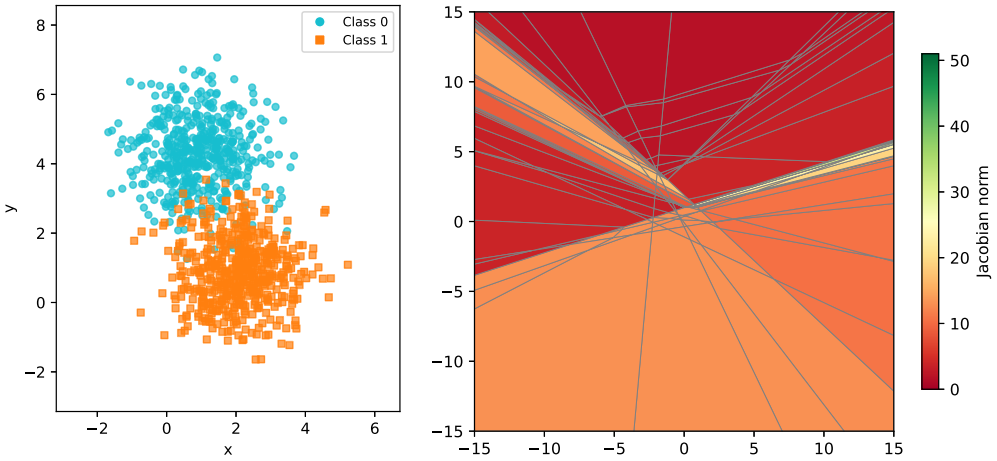



**(f)** Epoch 1,000 (26 training points misclassified)

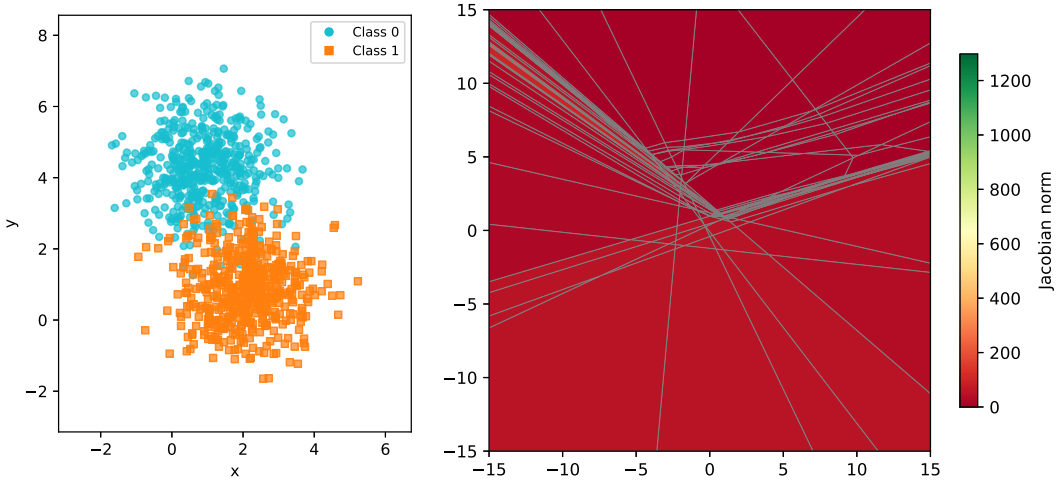
**Figure 4.8:** Effects of training data structure on the resulting polyhedral decomposition.




**(a)** Standard deviation = 0.5 **(b)** Epoch 25 (perfect accuracy on training data)




**(c)** Standard deviation = 1.0 **(d)** Epoch $1,000$




**(e)** Standard deviation = 1.0 **(f)** Epoch $5,000$

**Figure 4.9:** Effects of training data structure on the resulting polyhedral decomposition.

more polyhedra contain the decision boundary), while in Figure 4.9, the majority of the Jacobian resources can be given to one or two polyhedra. A network trained on a more complicated data structure (in terms of linear separability), like the interleaving half circles, is required to come up with a handful of bit vectors that produce relatively high input-output Jacobian norm values and then must contort itself such that these roughly align with a sensible decision boundary, while one trained on a more simple structure (highly linearly separable) is only required to produce one or two such bit vectors.

Since polyhedral size and shape as well as the input-output Jacobian norm of their mappings change during training based on the geometry of the data being learned, we explore these further in the following two sections.

### 4.3.1 Implementation Details

All of the datasets used in this section were created using the `sklearn.datasets` package's `make_circles`, `make_moons`, and `make_blobs` functions. Unless otherwise specified, the concentric circles training data was created using a standard deviation of 0.05 and a scale factor between the inner and outer circles of 0.8, the interleaving half circles training data was created using the same standard deviation value, and the cluster data has a standard deviation of 1.0. Each of the training datasets consists of 500 samples per class.

## 4.4 Polyhedral Size and Shape

We are interested in the size and shape of a single polytope and of a collection of polytopes in a local region of the input space, so we provide methods of exploring both. With access to a neural network's polyhedral decomposition, one could find the $A$ and $c$ that define the polytope, reduce them down to their minimal sets, use these (or their equivalent representation as a collection of hyperplanes or vertices depending on the algorithm of choice) and geometry to find the volume of the polytope. While this sounds simple enough, computing the volume of a convex polytope has been an active area of research [73–79]. The largest number of dimensions that seem to be

experimentally considered in the literature so far is 500 [79], but we are interested in the volume of polyhedra in over 150,000 dimensions. Computing this, developing a new approximation method for this, or shaving off fractions of decimal places of existing proxies' errors could very well prove to be an entire dissertation topic of its own. We instead turn our attention to metrics that convey information about polyhedral shape and size that can be used to compare those of different polytopes. Considering a single polyhedron, the term *Chebyshev center* has been used to refer to the center

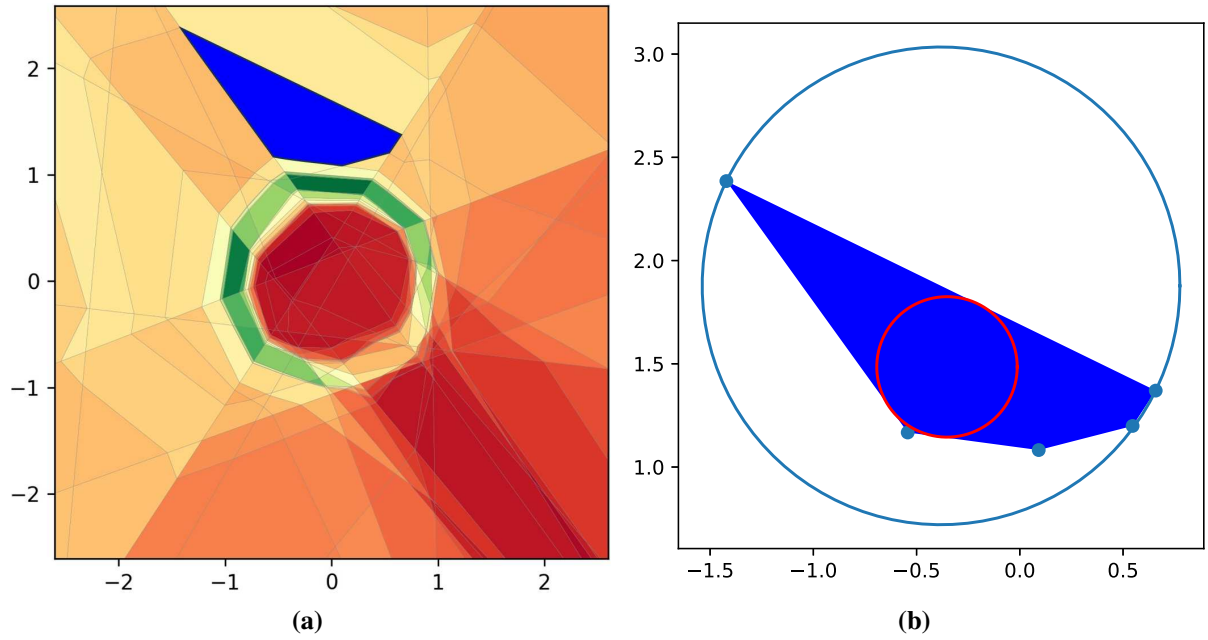



**Figure 4.10:** (a) A polyhedron of interest in the neural network's polyhedral decomposition. (b) The largest circle inscribed in the polyhedron of interest (in red), the smallest circle containing the polyhedron of interest (in blue), and the polyhedron's vertices.

of both the maximum inscribed hypersphere [80] and the minimum enclosing hypersphere [81, 82] of a convex region (see Figure 4.10). Since the current nomenclature does not differentiate these two, we call them the *inscribed Chebyshev center* and the *circumscribed Chebyshev center*, respectively. Often, algorithms computing these also compute their associated radii, which we call their *inscribed Chebyshev radius* and *circumscribed Chebyshev radius*. Computing the inscribed Chebyshev center and its radius is an easily implemented linear program (given in Algorithm 3) that remains tractable in high dimensions. On the other hand, the algorithm for

**Algorithm 3** Inscribed Chebyshev Center and Radius

Choose a point of interest in the input space, $x \in \mathbb{R}^n$, which belongs to the polytope satisfying $Ax \leq c$.

**Step 1.** Define $\xi = \begin{bmatrix} 1, 0, \ldots, 0 \end{bmatrix}^\top \in \mathbb{R}^{n+1}$ and $\hat{A}_i = \begin{bmatrix} \mathbf{0} \\ A_i \end{bmatrix}$.

**Step 2.** Solve the linear program

$$\begin{aligned} &\textbf{minimize} && -\xi^\top z \\ &\textbf{subject to} && \left(\hat{A}_i + \left\|\hat{A}_i\right\|_2 \xi^\top\right) z \leq c_i \quad \text{for } i = 1, \ldots, \beta \end{aligned}$$

where $z = \begin{bmatrix} r, y_1, \ldots, y_n \end{bmatrix}^\top$, and $r$ and $y \in \mathbb{R}^n$ are the inscribed Chebyshev radius and center, respectively [80].

computing the circumscribed Chebyshev center and its radius given in Algorithm 4 scales with the number of vertices in the polytope of interest. Formally enumerating vertices of polytopes is a #P-complete problem [83, 84]. Theoretical analysis shows that the number of vertices explodes as the dimension increases, shown in Figure 4.11. That is to say that Algorithm 4 becomes infeasible in high dimensions, but the use of Algorithm 5 to reduce the size of the problem to be solved can at least provide a speed-up. Since the shape of polytopes seems to inherit some structure from the data on which the neural network was trained, the ratio between these two radii may be of interest. In our small dimensional examples, relatively long and skinny polyhedra outline the neural network's learned decision boundaries. These rectangular polytopes would have a relatively large difference in radii while more square-shaped ones would have more similar inscribed/circumscribed Chebyshev radii. Now, consider the case in which polyhedral size is desired for a particular region of the input space, the region about some polytope, $P^*$. Define $G_\alpha(P^*) \backslash E$, the $\alpha$-*Hamming-hypersphere* about $P^*$, (or equivalently, about any point $x^{(0)}$ contained in $P$) to be the collection of polyhedra within a Hamming distance of $\alpha$ from $P^*$ for some positive integer $\alpha$; $G_\alpha(P^*) \backslash E = \{P : H(P, P^*) \leq \alpha\}$ (this choice of notation is made clear in Section 4.5). See

**Algorithm 4** Circumscribed Chebyshev Center and Radius

Choose a point of interest in the input space, $x \in \mathbb{R}^n$. Given a neural network $F$, identify the polytope $P$ that contains $x$.

**Step 1.** Find the polytope's vertices, the set of points: $\mathscr{V} = \{v_1, \ldots, v_\rho\}$.

**Step 2.** Perform Welzl's Algorithm on $\mathscr{V}$:

Initialize $i = 1, \mathscr{V}_i = \{v_i\}, B_i = v_i$, where $B_j$ denotes the minimum covering ball of points $v_1, \ldots, v_j$, described by two parameters: its center $c(B_j)$ and its radius $r(B_j)$.

**while** $\mathscr{V}_i \subset \mathscr{V}$ **do**

$i = i + 1$

$\mathscr{V}_i = \mathscr{V}_{i-1} \cup \{v_i\}$

**if** $v_i \in B_{i-1}$ **then** $B_i = B_{i-1}$

**else** define $B_i$ such that it contains $B_{i-1}$ and $v_i$ is on the boundary of $B_i$.

**end if**

**end while**

**return** $c(B_i)$, $r(B_i)$

Figure 4.12 for an illustrative example. Let $\partial G_\alpha(P^*)\backslash E$ denote the boundary of $G_\alpha(P^*)\backslash E$; $\partial G_\alpha(P^*)\backslash E = \{P : H(P,P^*) = \alpha\}$. When $\alpha = 0$, $G_\alpha(P^*)\backslash E = \partial G_\alpha(P^*)\backslash E = P^*$. To get an idea of the size of the polyhedra within a Hamming distance of $\alpha$ from $P^*$, we can produce a distribution of Euclidean distances in the following way. We first calculate $c^*$, the center of $P^*$ (either the inscribed or circumscribed center will do). Then, collect the polytopes belonging to $\partial G_\alpha(P^*)\backslash E$, find their centers, and calculate their Euclidean distances to $c^*$. If we perform this procedure iteratively with increasing $\alpha$ values, we consider larger local regions of the input space and can monitor how polyhedral sizes change. This is outlined in Algorithm 6. Notice that this provides a statistic that is *proportional* to the size of the polyhedra in the region of interest; for large polyhedra, the Euclidean distances between their centers will also be large. Algorithms 3, 4,

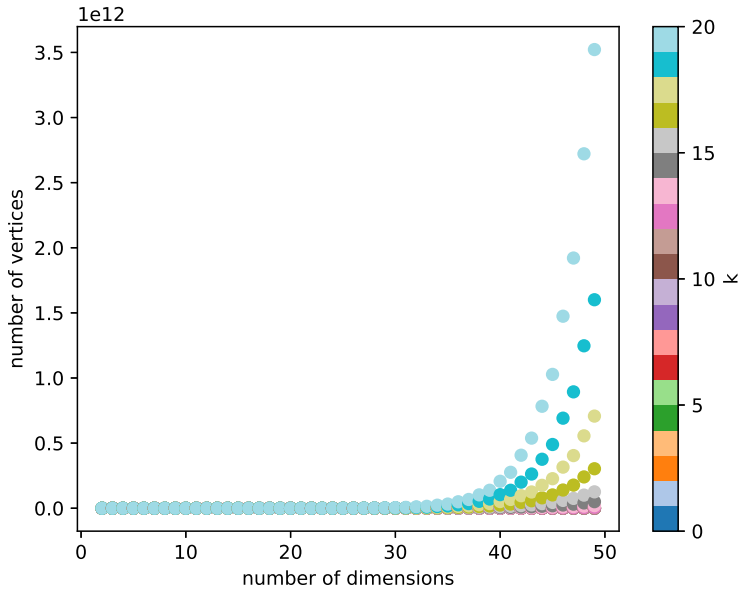



**Figure 4.11:** The relationship between the number of dimensions and the maximal number of vertices a convex polytope can have for a varying number of half-spaces given by $n+k$ where $n$ is the ambient dimension [2].

and 6 require that (a) the associated neural network be linearly represented and that (b) the system of inequalities describing the polyhedron of interest be computed beforehand, a process which has been outlined in [3]. Although work has been done to perform (a) with relative ease [43], the authors' available code base appears only to accommodate simple linear architectures (as opposed to ResNet [44] or DenseNet [85] architectures). Therefore, (a) and consequently also (b) are unrealistic requirements which we circumvent with algorithms presented in Chapter 5 that implicitly access a neural network's polyhedral decomposition.

**Algorithm 5** Reduce Linear System to Minimal Set [86]

Consider a linear system $Ax \leq c$, $A = \left[A_1, A_2, \ldots, A_q\right]^\top$ and $c = \left[c_1, c_2, \ldots, c_q\right]^\top$ where $A_i \in \mathbb{R}^{n\times 1}$ and $c_i \in \mathbb{R}$. Initialize $A_{\min} = A$, $c_{\min} = c$, and indices$= \{\}$.

**for** $i = 1, 2, \ldots, q$ **do**

**Step 1.** Define $\tilde{A}$ and $\tilde{c}$ by removing the $i^{\text{th}}$ row of $A_{\min}$ and $A_{\min}$, respectively.

**Step 2.** Solve the linear program

$$\begin{aligned} \textbf{minimize} \quad & -A_i^\top x \\ \textbf{subject to} \quad & \tilde{A}x \leq \tilde{c}. \end{aligned}$$

**if** $\min(-A_i^\top x) \leq c_i$ **then** remove the $i^{\text{th}}$ row of $A_{\min}$ and $A_{\min}$ **and** indices = indices $\cup\{i\}$.

**end if**

**end for**

**return** $A_{\min}$, $c_{\min}$, indices

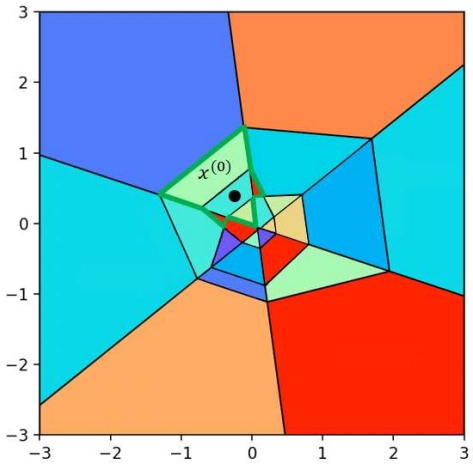



**Figure 4.12:** A polyhedral decomposition of a trained neural network with a Hamming-hypersphere with a radius of 1 outlined in green.

**Algorithm 6** Polyhedral Width Statistic Using Exact Width Measurements

Select (a) a point of interest $x^{(i)}$ in the neural network's domain, which belongs to polytope $P^*$ and (b) a Hamming distance $\alpha^*$ which together define a local region of interest, $G_{\alpha^*}(P^*)\backslash E$. Initialize $\alpha = 0$, $G_\alpha(P^*)\backslash E = P^*$, and an empty list $\mathscr{E}^{(i)}$. Compute $c^*$, the center of $P^*$.

**while** $\alpha < \alpha^*$ **do**

$\alpha = \alpha + 1$

**Step 1.** Collect the polyhedra belonging to $\partial G_\alpha(P^*)\backslash E$ by (a) getting $s^*$, the bit vector of $P^*$, (b) getting the linear system $Ax \leq c$ defining $P^*$, (c) finding the active bits of $s^*$ ('indices' from Algorithm 5), and (d) flipping the entries of $s^*$ in the locations of the active bits.

**Step 2.** Find their centers (either using Algorithm 3 or 4).

**Step 3.** Compute $\mathscr{E}_\alpha^{(i)}$ the set of Euclidean distances between the centers found in Step 2 and $c^*$.

**Step 4.** $\mathscr{E}^{(i)} = [\mathscr{E}^{(i)}, \mathscr{E}_\alpha^{(i)}]$

**end while**

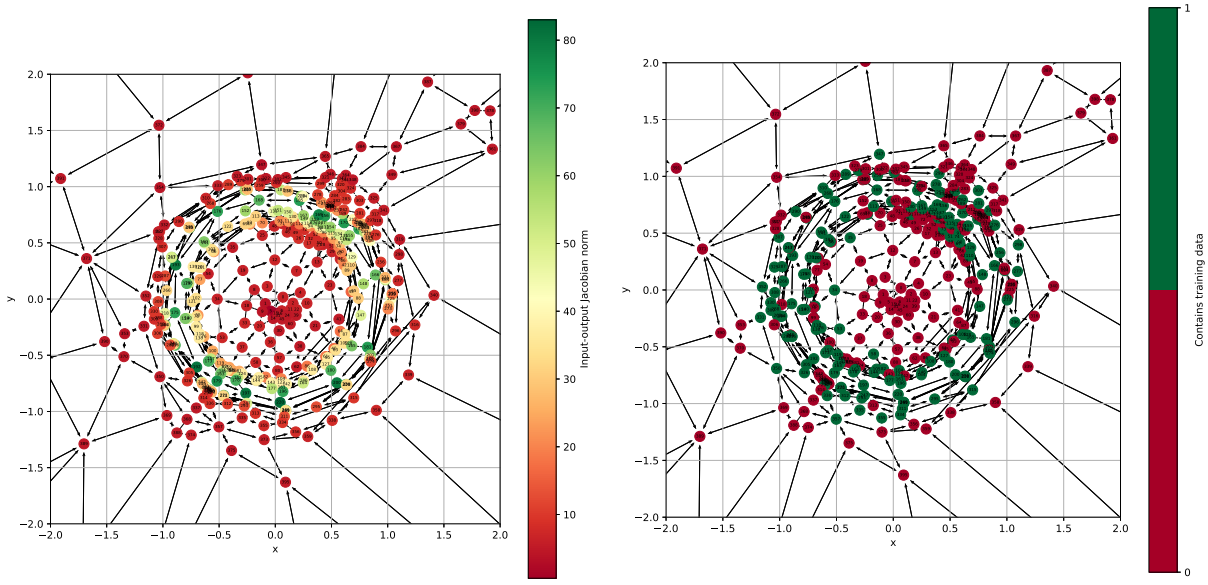
**return** $\mathscr{E}^{(i)}$

## 4.5 Polyhedral Topology

As alluded to in Section 4.1, polyhedral decompositions have equivalent graph representations where each node represents a distinct polyhedron and the edges can describe any number of relationships between these nodes. Before getting to the specifics, we first introduce some relevant notation. Let $G_\alpha(P)$ be the graph including nodes with a degree of separation from $P$ that is less than or equal to $\alpha$ (whose collection of nodes are simply the graph itself with the edges removed, $G_\alpha(P)\backslash E$). Let $\partial G_\alpha(P)$ denote the subgraph of $G_\alpha(P)$ only containing its peripheral nodes, those whose degree of separation from $P$ is exactly $\alpha$. This notation may seem familiar as it was introduced in Section 4.4 using different language. Here, we emphasize their underlying

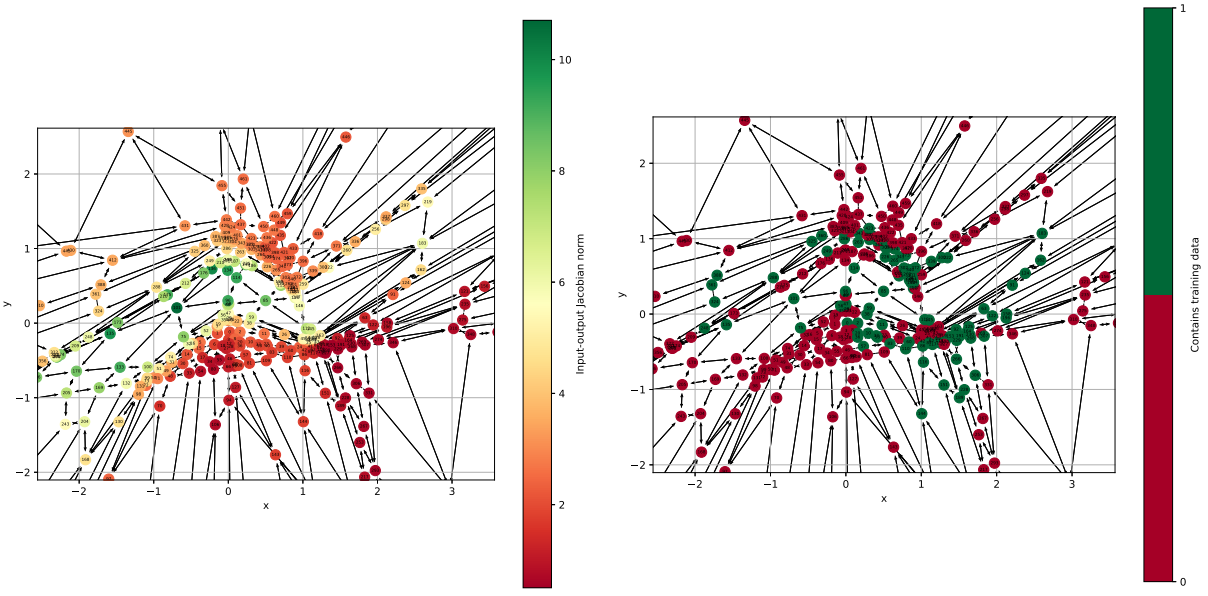
graph structure, which was indirectly used in Section 4.4. Further, let $|G|$ denote the cardinality of the graph or the number of nodes it contains.

Now, the edges among the nodes of $G_\alpha(P)$ can be drawn in one of three ways. Firstly, they can be unweighted and undirected (or equivalently, directed with bi-directional connections), only connecting nodes that have a Hamming distance of one. Secondly, they can be weighted by a symmetric metric (like the Hamming distance) and therefore undirected. Lastly, the edges can be weighted by a metric that does not have the symmetric property, resulting in a directed graph. In the last two cases, the edges can either connect all nodes or only those whose weight is below a certain threshold. In this section, we train a single neural network architecture given in (4.36) using data of three different shapes: concentric circles, interleaving half circles, and clusters. After training, we form the resulting neural networks as (a) undirected and unweighted graphs that only connect nodes whose Hamming distance is one and (b) directed weighted graphs where edges are weighted by the input-output Jacobian norm of the destination node and exist only between nodes whose associated bit vectors have a Hamming distance of one. The primary function of graphs of type (a) is to show the concentration of polyhedral resources. Indeed, observe that in Figures 4.13, 4.14, 4.15, the graph nodes concentrate in a way that outlines the neural network's learned decision boundary. Meanwhile, we use the graphs of type (b) to compute the proximity of a particular polyhedron of interest to those with high input-output Jacobian norms. The results of such computations can be found in Figures 4.16, 4.17, and 4.18. By construction, these analyses focus on a particular polyhedron's Hamming distance to polyhedra with high input-output Jacobian norm rather than on its Euclidean distance. In particular, observe that in Figure 4.18, subfigures b and c make nodes 327 and 35 seem roughly the same distance from the uncertain region of the input space. Node 327 (approximately located at $x = 0,\ y = 5$) is relatively close to the decision boundary in Euclidean space, but there are many tiny polyhedra between it and node 174, which contains the decision boundary. Meanwhile, node 35 (approximately located at $x = 9,\ y = -12$) lies much farther away from the decision boundary. Refer to Appendix C for more plots. This




**(a)** **(b)**

**Figure 4.13:** The neural network's polyhedral decomposition (after 80 epochs of training on the two-class concentric circles data, as shown in Figure 4.5) visualized graphically where the node colors in (a) reflect the input-output Jacobian norm of the affine mapping prescribed by the associated polyhedron and (b) reflect whether or not the associated polyhedron contains training data.




**(a)** **(b)**

**Figure 4.14:** The neural network's polyhedral decomposition (after 29 epochs of training on the two-class interleaving half circles data, as shown in Figure 4.6) visualized graphically where the node colors in (a) reflect the input-output Jacobian norm of the affine mapping prescribed by the associated polyhedron and (b) reflect whether or not the associated polyhedron contains training data.

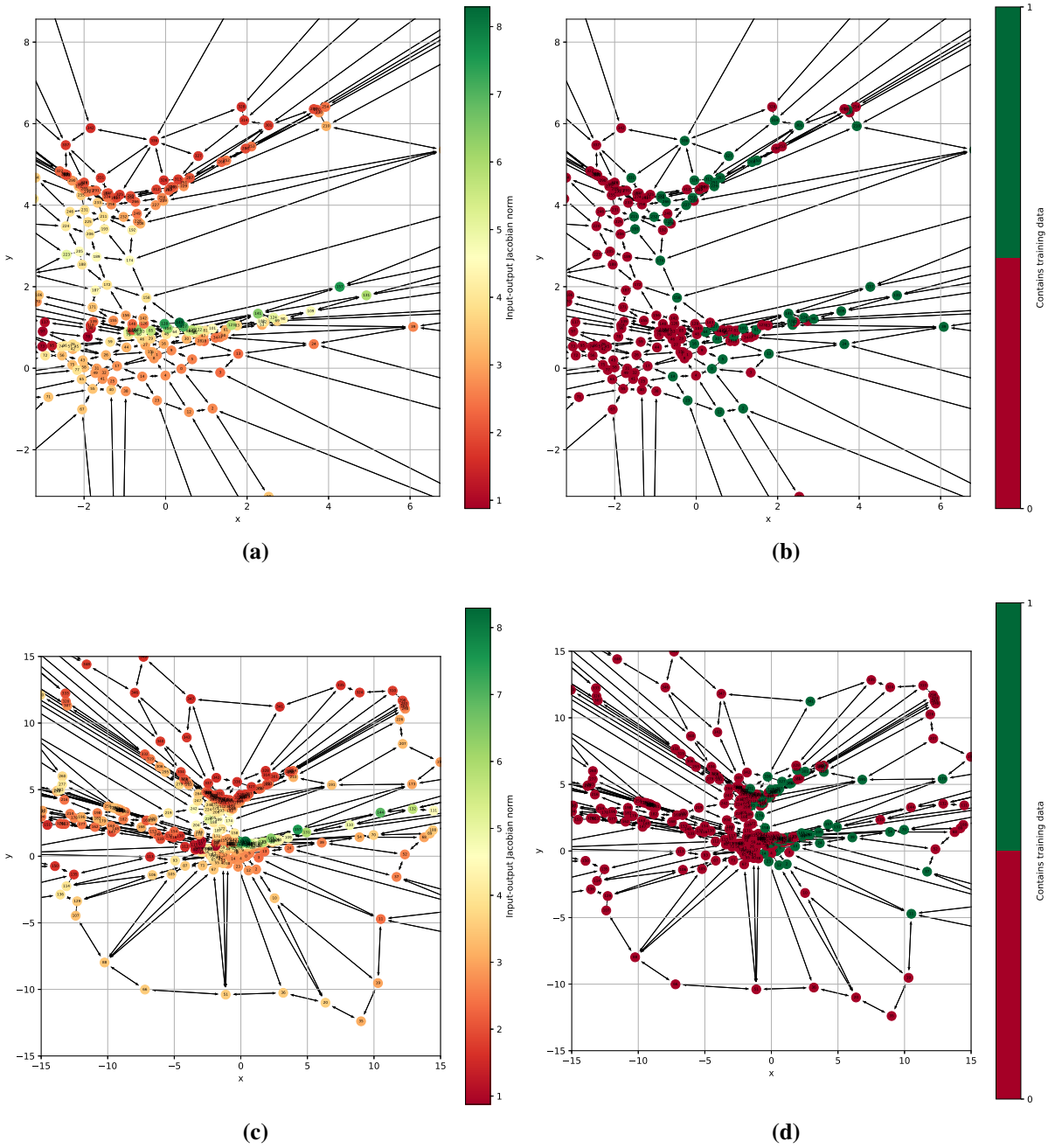



**Figure 4.15:** The neural network's polyhedral decomposition (after training on the two-class cluster data) visualized graphically where the node colors in (a) reflect the input-output Jacobian norm of the affine mapping prescribed by the associated polyhedron and (b) reflect whether or not the associated polyhedron contains training data. Plots (c) and (d) are identical to (a) and (b), respectively, but are zoomed out to include a larger area of the domain.

emphasis on Hamming distance might be useful if one suspects their neural network is targeted by a bit vector-altering attack.

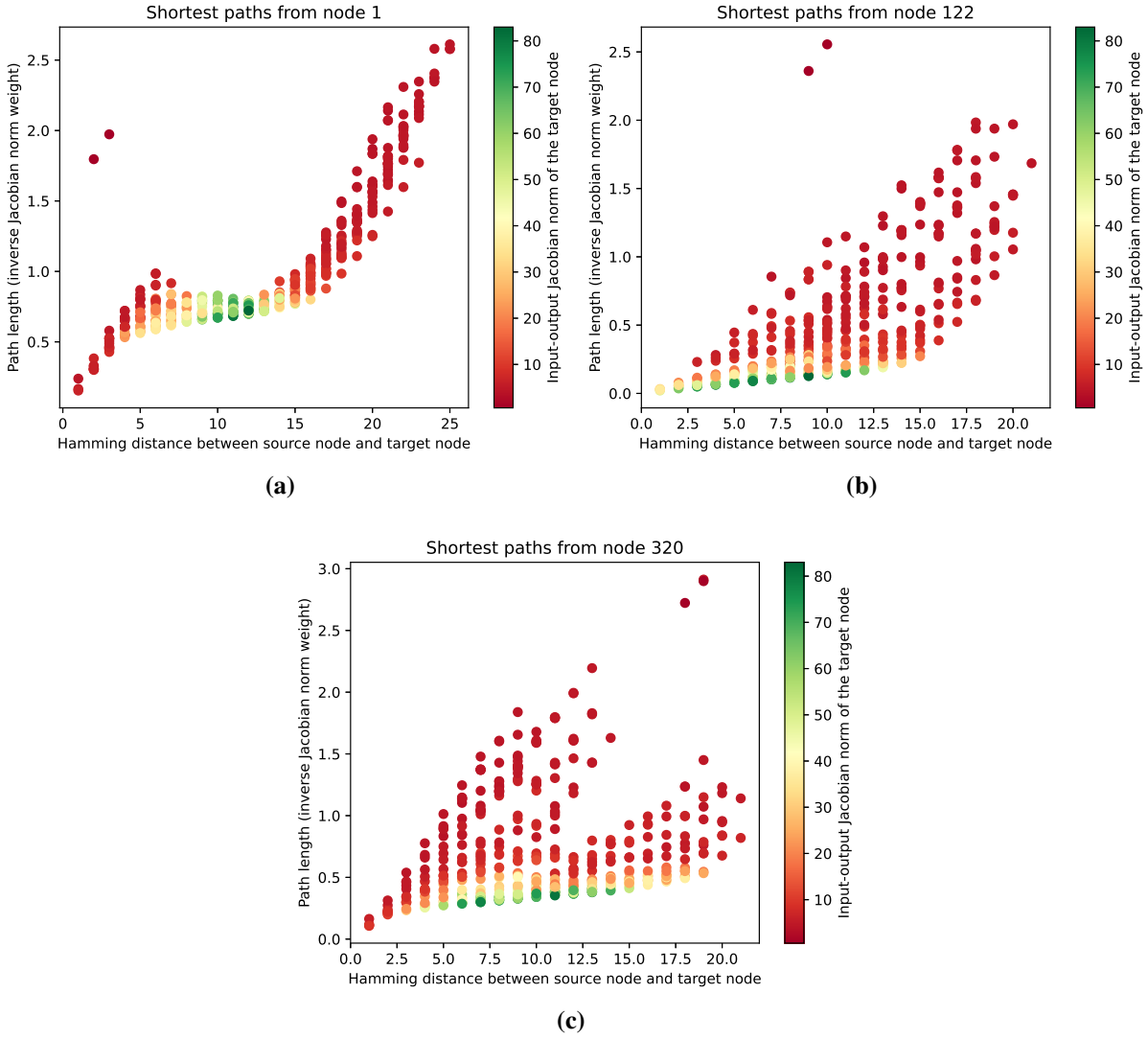



**Figure 4.16:** The shortest path (in terms of the inverse of the input-output Jacobian norm) along the edges of the graph given in Figure 4.13 from $P$ to the polyhedra in $\partial G_\alpha(P)$ where the x-axis gives $\alpha$ and the y-axis gives path lengths. Polyhedron $P$ (a) lies inside of the nested circles of training data, (b) contains training data, and (c) lies outside of the outer circle of training data.

While it is mathematically interesting to form polyhedral decompositions as graphs (what insights can graph theory offer?), we point out a couple of drawbacks of this approach. First, we come back to the difference between Euclidean and Hamming distances and their importance in

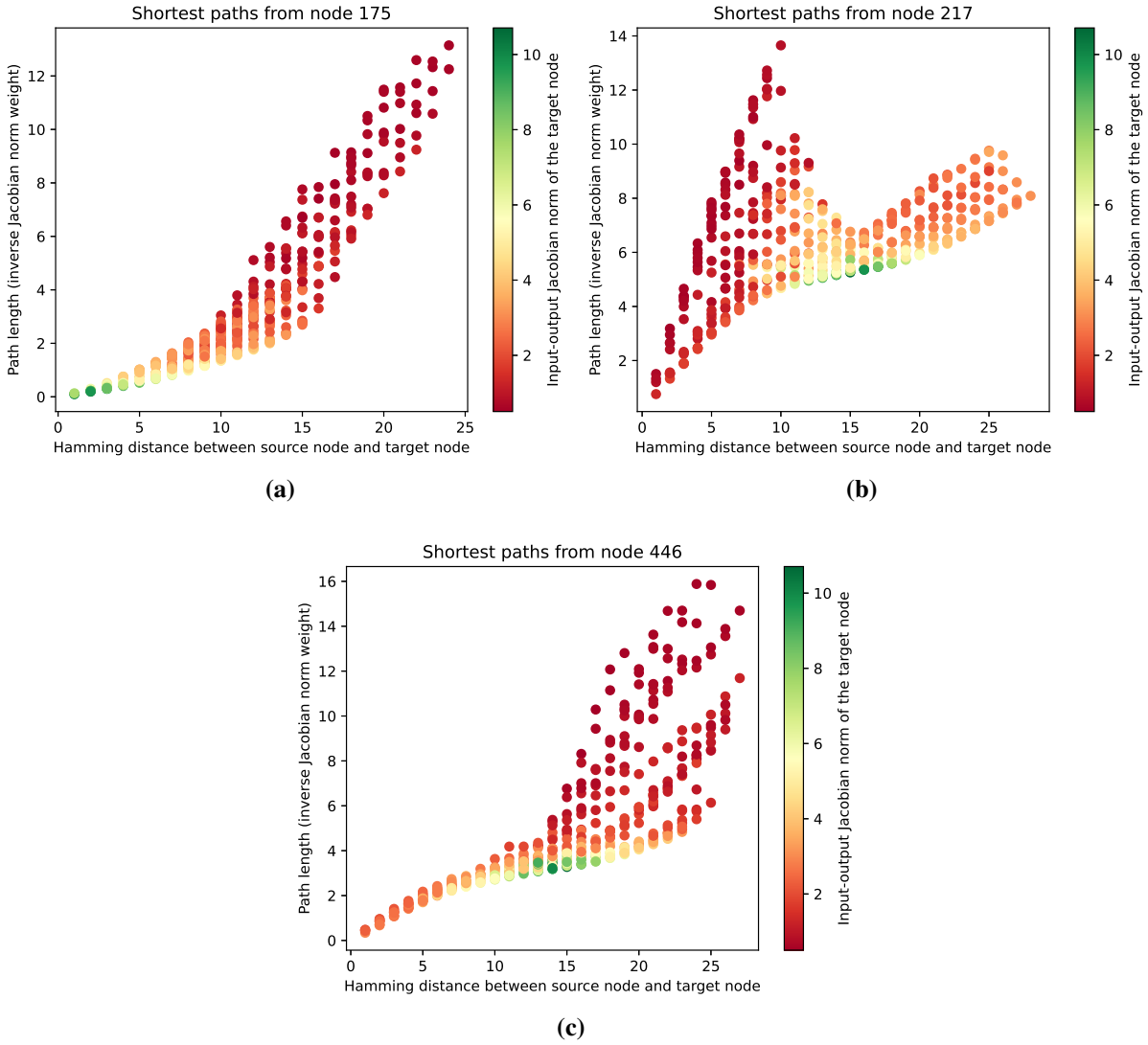



**Figure 4.17:** The shortest path (in terms of the inverse of the input-output Jacobian norm) along the edges of the graph given in Figure 4.14 from $P$ to the polyhedra in $\partial G_\alpha(P)$ where the x-axis gives $\alpha$ and the y-axis gives path lengths. Polyhedron $P$ (a) lies close to the learned decision boundary between classes, (b) and (c) lie away from the learned decision boundary.

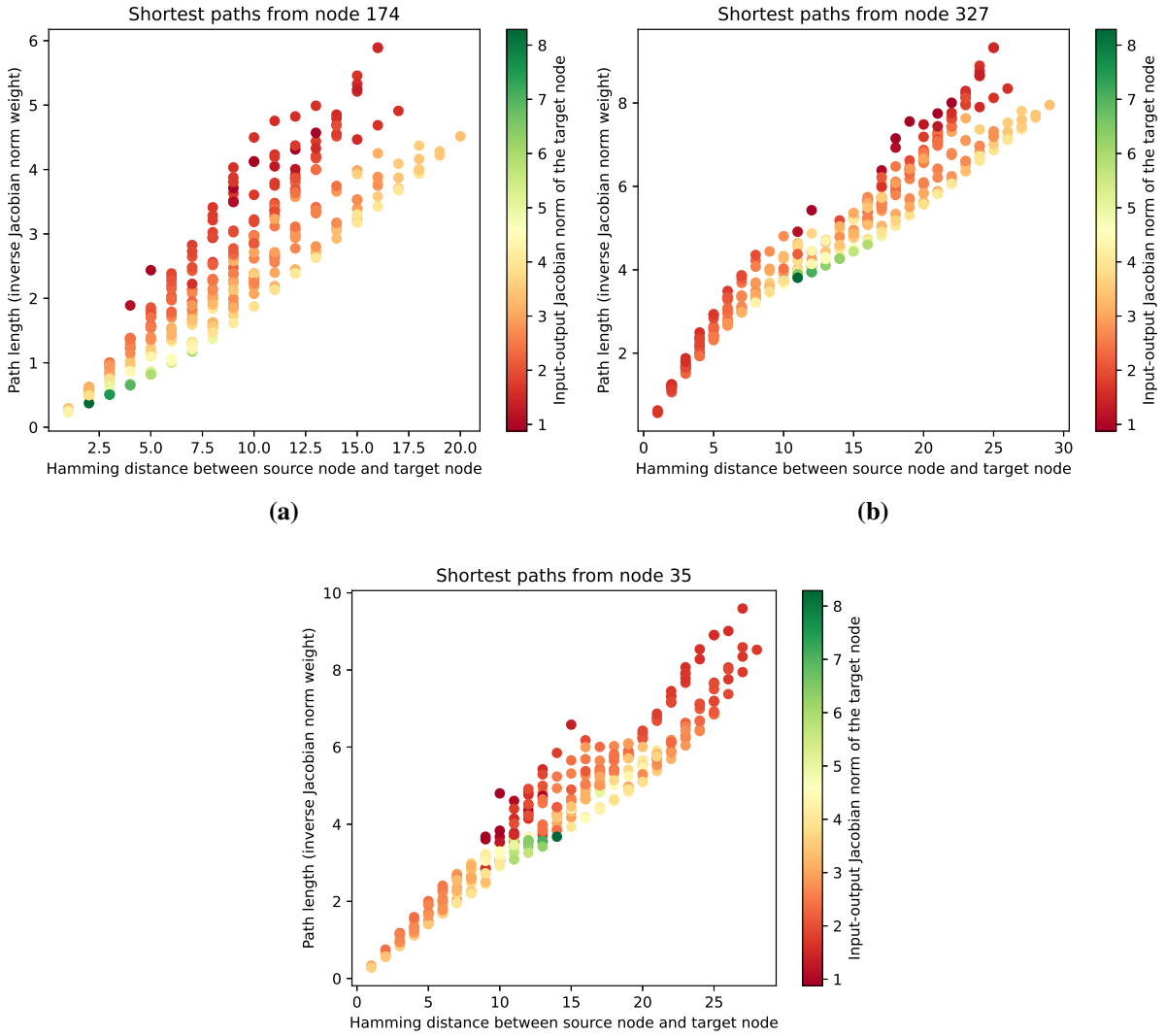



**Figure 4.18:** The shortest path (in terms of the inverse of the input-output Jacobian norm) along the edges of the graph given in Figure 4.15 from P to the polyhedra in $\partial G_\alpha(P)$ where the x-axis gives $\alpha$ and the y-axis gives path lengths. Polyhedron P (a) contains the learned decision boundary between classes, (b) contains training data and lies relatively close to the learned decision boundary, and (c) lies away from the learned decision boundary.

different adversarial attacks. Some attacks are bound by Euclidean distance from the original data to avoid detection, in which case, the data's Euclidean proximity to regions of uncertainty would be of higher interest. Second, we point out the huge memory demand that saving a neural network as a graph would be. One could expect to have millions of nodes and significantly more edges. If the edges are weighted, that is yet more information to save, and if they are weighted by a non-symmetric metric, that doubles the memory requirement of the edges. Lastly, even computing a piece of a model's graph would require immense computational resources. The collection of the nodes of interest would be the most tasking part and could be done by iteratively performing Step 1 of Algorithm 6.

$$\mathbb{R}^2 \to \mathbb{R}^{10} \to \mathbb{R}^{10} \to \mathbb{R}^{10} \to \mathbb{R}^{10} \to \mathbb{R}^1 \tag{4.36}$$

## 4.6 Conclusion

We have formally presented neural networks with different activation functions as partitioning schemes of their domains and as linear (or approximations of linear) mappings from their input to output spaces. We observed how these partitionings morph throughout training. This yielded two neural network characteristics, polyhedral size and input-output Jacobian norm, as being highly impacted by the geometry of the training data, and therefore, insightful of the neural network's learned representation of it. Consequently, we presented algorithms that explicitly use a polyhedral decomposition to probe its associated neural network regarding these two features. Meanwhile, our detailed formulation of neural networks' equivalent mappings emphasizes the crucial role neuron firing patterns play in distinguishing one polyhedron's mapping from another. Digging deeper into binning vectors, polyhedral size, and input-output Jacobian norms for the network's insights on the data that produced them is a large part of the research we discuss next.

In the next chapter, our research shifts from explicitly to implicitly accessing neural networks' polyhedral decompositions. The inequalities $Ax \leq c$ with $A \in \mathbb{R}^{h \times m}$, $x \in \mathbb{R}^{m \times 1}$, $c \in \mathbb{R}^{h \times 1}$ that define each region of neural networks are huge in practice. For reference, $h = 1.6\text{M}$ and $m = 3 \times 224 \times 224$

for ResNet-18, which is one of the smaller pretrained networks made available by PyTorch at https://pytorch.org/vision/stable/models.html. The sheer size of polyhedral decompositions makes them cumbersome, at best, to explicitly formulate and manipulate. At worst, they are impossible to define because their neural network cannot be expressed linearly. Therefore, tools to implicitly access them are essential.

# Chapter 5

# Implicit Use of Polyhedral Decompositions

## 5.1 Introduction

Due to the difficulty of explicitly accessing a model's associated polyhedral decomposition, we develop tools to do so implicitly which we will use in experiments guided by the fundamental observations made in the previous chapter. In Section 5.2.1, we harness the geometric information in bit vectors to accurately distinguish between original and adversarial images. Throwing away a lot of the spatial data encoded in bit vectors as we do in Section 5.2.2, by viewing them as codes conveying the similarity of neural firings given two domain points of interest, we show that faint remnant signals distinguishing original and adversarial images survive. Using this interpretation of bit vectors, we developed the first algorithm to measure polyhedral widths up to numerical precision. Lastly, we introduce the first use of input space landscapes in Section 5.3 to visualize the neural network's stability with respect to predictions and Jacobian norms under perturbations in the input.

**Novel contributions in this chapter:**

- present a classifier based on bit vectors that classify ImageNet images from ImageNet-FGSM ones with over 90% accuracy (published in 2022 IEEE International Conference on Big Data [4]),
- define the first algorithm to measure the exact polyhedral width without the explicit formulation of a polyhedral decomposition,
- develop input space landscapes as a tool to more thoroughly explore neural network stability in regions of interest in their domain, and using these we

- show that the behavior of the input-output Jacobian norm about training images is much more complex than previously thought, and
- show that the predictive stability about ImageNet and DAmageNet are surprisingly similar.

## 5.2 Geometry Imposed on Data

### 5.2.1 Bits and Splines

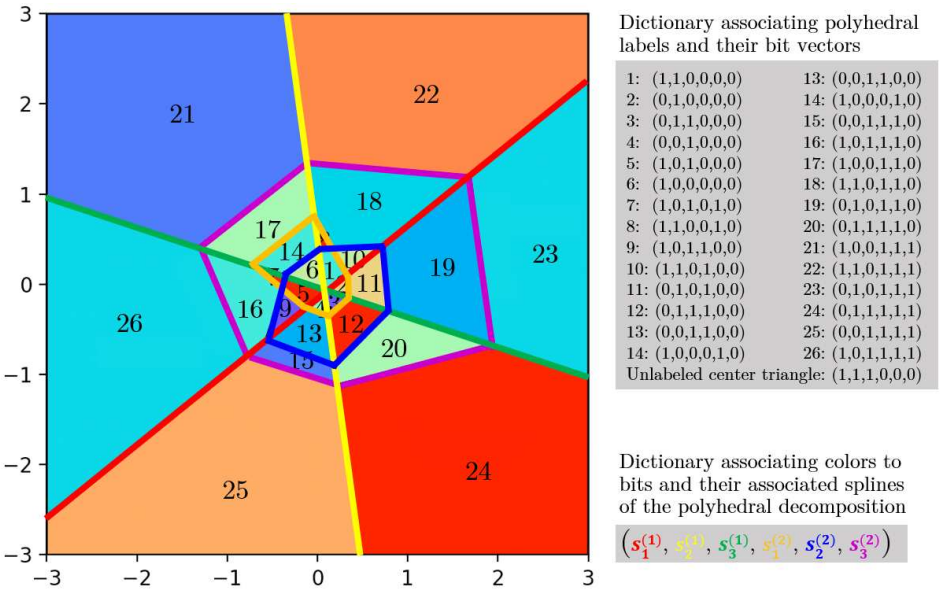



**Figure 5.1:** The polyhedral decomposition of the ReLU FFNN $F : \mathbb{R}^2 \to \mathbb{R}^3 \to \mathbb{R}^3 \to \mathbb{R}$ after training in the unit square with each polyhedron labeled by its binary vector. The dictionary at the top right associates bit vectors to polyhedra and the dictionary at the bottom right associates each bit of the bit vectors to splines in the input space [3].

In Section 4.2, we alluded to the geometric information encoded in the bit vectors of a neural network, which we explain further here. We established that the $i^{\text{th}}$ entry of a polyhedron's bit vector defines whether or not that polyhedron satisfies the linear inequality $A_i x \leq c_i$. Synonymously, each bit has a corresponding spline and the bit value indicates the input point's relative location to it. For this reason, points belonging to neighboring polyhedra (ones that share a polyhedral face) only differ by one bit, namely the bit associated with the spline defining the boundary between the two polyhedra. For illustration, consider Figure 5.1. Without loss of generality, notice that any

polyhedron lying to the left of the red spline has a bit vector whose first bit is one (e.g. polyhedra 26 and 22) while those lying to its right have a bit vector starting with a zero (e.g. polyhedra 25 and 23). Also, the bit vector of polyhedron 22 differs from that of 23 only in their first bit since its associated spline is the one separating them.

The set of inequalities defined, in part, by bit vectors often overdetermines a polyhedron. Consider polyhedron 19 in Figure 5.1, and say it satisfies the inequality $Ax \leq c$. Since polyhedron 19 is uniquely determined by the first, third, fifth, and sixth bits in its bit vector (or equivalently by its relative location to the red, purple, green, and blue splines), then only $A_i, c_i$ for $i = 1,3,5,6$ are required to define it. We call the system of inequalities defined using these particular rows of $A$ and entries of $c$ the minimal set of inequalities defining polyhedron 19 and call bits 1, 3, 5, and 6 the *active bits* of its bit vector. The active bits of a bit vector correspond to the inequalities that define the bounding edges of its polyhedron. The process of reducing a linear system to its minimal set is provided in Algorithm 5. With this geometric understanding of each bit of a neural network's bit vectors, we turn to an application.

**ImageNet versus ImageNet-FGSM**

Here, we use this geometric interpretation of bit vectors to distinguish original images from adversarial ones, ImageNet from ImageNet-FGSM, given a network of interest, ResNet-50. We set out to identify particular bits that correspond to splines that separate original data from adversarial data. We determined that no splines do this perfectly, so we try to find ones that do so with high accuracy. Equivalently, we search for bits where the bit vectors of most original data differ from those of most adversarial data. More specifically, we are interested in identifying a set $B_f$ of bits whose firings indicate a relatively high probability that the input image was an ImageNet image. This can be because many of the ImageNet images make this particular neuron fire OR because many DAmageNet images do not make this neuron fire. We are also interested in the converse: in the indices $B_n$ of bits whose failure to fire indicates a relatively high probability that the input image was an ImageNet image, either because many ImageNet images do not make these particular neurons fire OR because many DAmageNet images do. Collecting these special

bits, we can compare the bit vector of any image of interest in these locations, looking for ones in the bits of $B_f$ and for zeros in those of $B_n$. If 50% of the bit values of our image match the values of our special bits, we predict that the image is an original image; otherwise, that it is adversarial. We experiment with the threshold meant by 'relatively high probability.' We do not consider threshold values higher than 0.77 since no single neuron discriminated ImageNet images from DAmageNet images with 78% accuracy or better. Taking this threshold to mean at least 51% had our classification method attain 91% accuracy in predicting whether an image is adversarial or not, the highest accuracy we observed in our experiments. Further details can be found in [4]. Now we experiment with the second geometric notion of bit vectors induced by the Hamming distance.

### 5.2.2 Hamming Distance

Comparisons between the binning vectors of distinct polyhedra induce a metric with respect to a particular neural network. Mentioned in [60] is a length metric equivalent to counting the number of polyhedra encountered or the number of bit vector 'transitions' along a particular trajectory. The authors do not ensure that their traversal method hits all polyhedra encountered along a particular trajectory, rather, they define a small step size and count the number of unique binary vectors seen at all steps. Interestingly (but less immediately relevant to our work), they provide layer-wise estimates for the Euclidean stretch or compression of a trajectory along the input space as it passes through the network. They also show that length (in terms of the number of polyhedra encountered) grows exponentially with model depth, which speaks to our point that these deeper neural networks have more polyhedral resources at their disposal. The work in [65] uses 'transition density,' which is computed by counting the number of polyhedra encountered along a trajectory and normalizing based on the Euclidean length of the trajectory. Again, only equidistant samples are drawn from the trajectory, so this metric is approximate. The authors claim that polyhedra about training data tend to be small, but their experiments used MNIST, which lacks complexity as mentioned in Section 3.3. Additionally, they heavily used data augmentation and an unconventional training scheme, which make the function that the neural network is trained to fit highly nonlinear (because

of the pre-existing lack of linear separability of MNIST [45]) and force the training to continue for extended periods of time. In Section 4.3, we showed that these two things together will result in polyhedral refinement about training data, refinement that their results indicate. The only work that explicitly count polyhedra along a trajectory are [62] and [86]. [62] does so by explicitly formulating the polyhedral decomposition of their relatively small neural networks, layer by layer. The authors also approximate the size of a single polyhedron rather than those in a local region by calculating the distance of a point of interest to the nearest polyhedral bounding edge. Meanwhile, [86] foregoes the need for an explicit formulation of a polyhedral decomposition by sampling along a trajectory while ensuring that consecutive samples have bit vectors that only differ in one entry, thereby implying they belong to distinct polyhedra and that these polyhedra are neighbors.

At the heart of all of this work is the exploration of polyhedral size, either that of a collection of polyhedra in a local region or of a single polyhedron. Unexpectedly, there does not exist an algorithm that measures the width of a polyhedron along an axis of interest, so we present a method do to so in Algorithm 7. In this procedure, we use the Hamming distance, yet another proxy for the number of polyhedra between two points of the input space.We adapt the Bisection Method from [86] into an algorithm that measures the exact width of a polyhedron. While the Bisection Method's aim is to sample all polyhedra along a designated line segment of the input space, our goal is to shrink a line segment in the input space until both endpoints are within a specified tolerance of each other $\left(\text{we use } 1.0 \times 10^{-10}\right)$ and the Hamming distance between them is one, implying that they straddle an edge of the polyhedron of interest. We do this in two opposite directions, save either one of the endpoints from both directions (we choose to save the endpoint lying within the same polyhedron as the point of interest) and measure

**Algorithm 7** Exact Polyhedral Width

Given a neural network $F$, a point $x^{(a)}$ inside of a polyhedron of interest, and a direction of choice $v_1, \|v_1\| = 1$, define the list directions = $[v_1, v_2]$ where $v_2 = -v_1$, tol=$1.0 \times 10^{-10}$, and initialize $\mathscr{X} = [\mathscr{X}_0, \mathscr{X}_1] = [\mathbf{0}, \mathbf{0}]$ so that $\mathscr{X}_i$ are of the same dimension as $x^{(a)}$. Then:

**for** $d$ = 1, 2 **do**

$v =$ directions[$d$]

Initialize $\delta = 1$.

**Step 1.** Set $x^{(b)} = x^{(a)} + \delta * v$.

**Step 2.** Compute $s^{(a)}$ and $s^{(b)}$, the bit vectors of $x^{(a)}$ and $x^{(b)}$, respectively.

**Step 3.** Compute the Hamming distance between $s^{(a)}$ and $s^{(b)}$, $H(s^{(a)}, s^{(b)})$.

**if** $H(s^{(a)}, s^{(b)}) = 1$ and $\left\| x^{(a)} - x^{(b)} \right\| \leq$ tol **then** $\mathscr{X}_d = x^{(b)}$.

**else if** $H(s^{(a)}, s^{(b)}) \geq 1$ **then** $\delta = 0.5 * \delta$, return to Step 1.

**else** $x^{(a)} = x^{(b)}$.

**end if**

**end for**

**return** $\|\mathscr{X}_0 - \mathscr{X}_1\|$

their Euclidean distance from each other, providing the polyhedral width along a particular axis through the given input point. In practice, we do this for a random sample of directions rather than just a single direction which produces a distribution of measurements rather than a single measurement. Figure 5.2 shows a toy example of this computation for a neural network with a two-dimensional input space. We show results using two different points inside the polytope to show how the distribution of widths can change based on this. It is worth noting that, because of the concentration of volume in large dimensions about boundaries (covered in Section 2.1), the points we sample from our input space will almost always lie close to polyhedral boundaries. This will consistently give means that are an underestimate compared to those returned if the inscribed

Chebyshev center were used instead. Due to the consistency of this phenomenon, we consider the resulting width measurements comparable among polyhedra of the same ambient dimension.

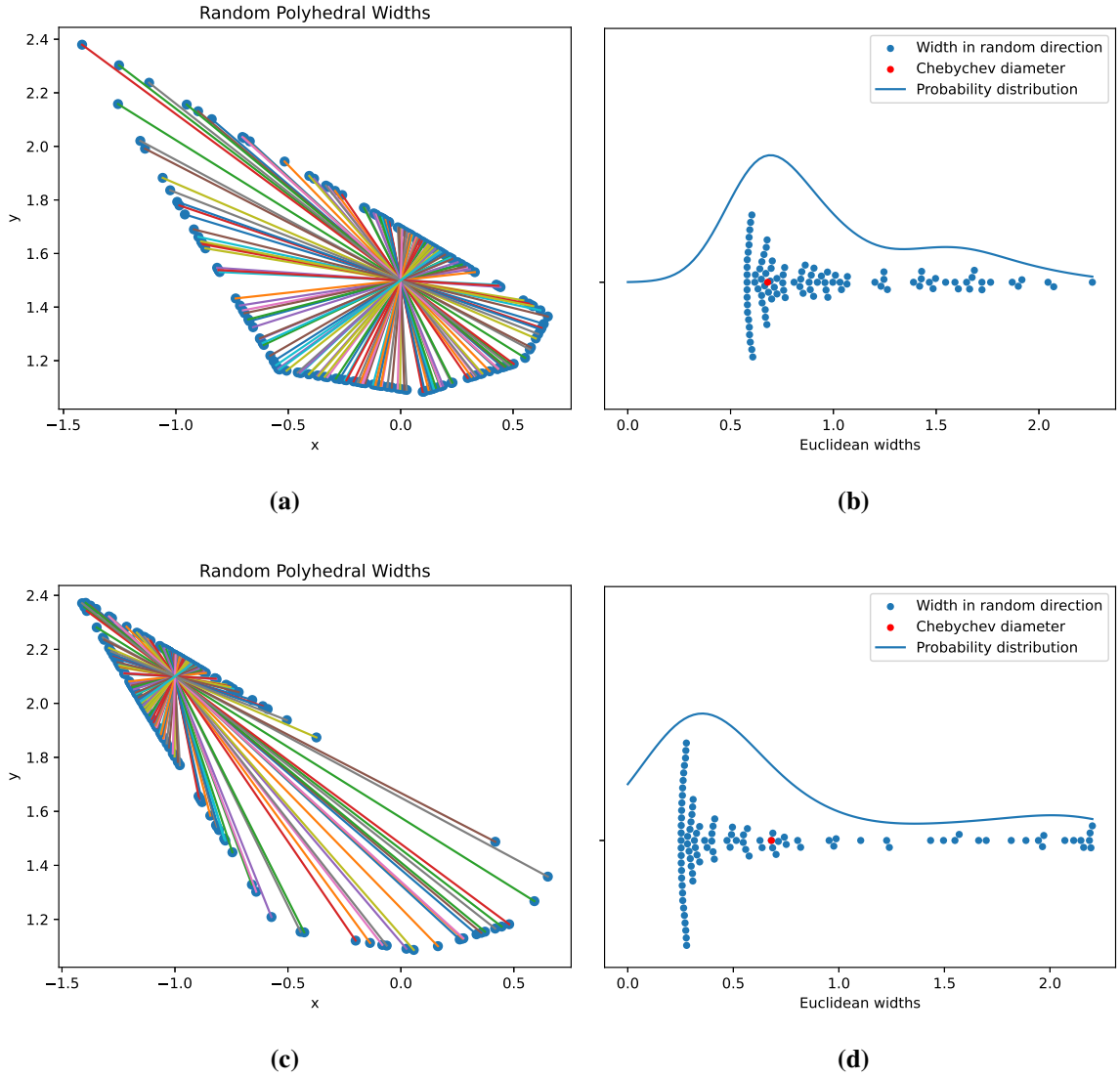



**Figure 5.2:** (a) 100 random axes through $p_1 = (0, 1.5)$ to the surface of a hypersphere about $p_1$ of the same dimension as the input space (i.e. a circle in this case). (b) The exact widths along the axes in (a) in blue, their corresponding probability distribution plot, and the inscribed Chebyshev diameter in red. Plots (c) and (d) are similar to (a) and (c), respectively with $p_2 = (-1, 2.1)$ to demonstrate variations in the distribution of widths depending on the point's location with respect to the polyhedron's boundaries. See Figure 4.10 for the polyhedron of reference.

Measuring the exact widths of a polyhedron, as in Algorithm 7, can be computationally expensive. Additionally, Algorithm 6 (from Section 4.4) requires that the linear inequality defining

the polyhedron of interest is reduced to its minimal set (which implies that the neural network is first expressed linearly and then that $h$ linear programs are completed for each polyhedron in $G_{\alpha^*}(P^*)\backslash E$). Therefore, we are interested in deriving a procedure that more efficiently measures some notion of the size and shape of either a single polyhedron or those present in a local region of the polyhedral decomposition. With this motivation, we harness the speed of the Hamming distance computation in Algorithm 8, wherein we perform persistent hamm-ology if you will. Notice that this provides a statistic that is *inversely proportional* to the size of the polyhedra in the region of interest; for small polyhedra, the Hamming distance between two points will be large. The choice of $\varepsilon$ and $D$ for Algorithm 8 depends on (a) the choice of neural network and (b) whether the size of the single polyhedron containing the point of interest or the sizes of the polyhedra in its vicinity are of interest. In particular, the choice of $D$ depends heavily on the dimension of the input space. Recall the concentration of measure that occurs in high-dimensions included in Section 2.1. We show in Figure 5.3 that the points in $\mathscr{P}_1$ do, in fact, concentrate about the equator [2]. To truly sample the surface of a hypersphere, not just the portion close to the equator, one must sample a number of points that grow exponentially with dimension[3]. However, we are not interested in ensuring a dense sampling of the unit hypersphere, but rather, that we have a distribution $\mathscr{H}_{\varepsilon}^{(i)}$ whose mean is stable under variations of $D$.

Similar to some of the aforementioned previous work, we also look at polyhedral size along an intrinsically one-dimensional trajectory, but we do so toward identifying the change in polyhedral size with increasing potent FGSM attacks on ImageNet validation images. We scale up to experiments within hyperspheres about an image of interest using Algorithm 8 and the DAmageNet dataset. We experiment with values of $D$ toward choosing one for which our resulting distribution is stable in high dimensions and with values of $\varepsilon$ toward, again, identifying differences in polyhedral size in the vicinity of ImageNet images and their adversarial variations.

[2]We point the interested reader to [8] for similar visualizations showing the growing concentration of samples about the equator with an increase in dimension.

[3]We point the interested reader to [87] for some theoretical background on high-dimensional convex geometry.

**Algorithm 8** Inverse Polyhedral Width Statistic Using Hamming Distances

Given a neural network $F$, a point $x^{(i)}$ inside of a polyhedron of interest, choose a value $\varepsilon$, a number of directions $D$, and initialize $\mathscr{H}_{\varepsilon}^{(i)} = \{\}$, a set where a distribution of $D$ Hamming distance will be saved. Once this set is populated by these $D$ measurements, $\mathscr{H}_{\varepsilon}^{(i)} = \{\mathscr{H}_{\varepsilon,j}^{(i)}$ for $j = 1,\ldots,D\} = \mathscr{H}_{\varepsilon,1:D}^{(i)}$.

**Step 1.** Produce $\mathscr{P}_1$, a set of $D$ random samples on the unit hypersphere about the origin.

**Step 2.** Shift $\mathscr{P}_1$ so that it is centered about $x^{(i)}$: $\mathscr{P}_1^{(i)} = \{p_j + x^{(i)} : p_j \in \mathscr{P}_1$ for $j = 1,\ldots,D\}$.

**Step 3.** Scale the points in $\mathscr{P}_1^{(i)}$ by $\varepsilon$ so that they lie on the surface of a hypersphere with desired radius, $\varepsilon$: $\mathscr{P}_{\varepsilon}^{(i)} = \{\varepsilon p_j^{(i)} : p_j^{(i)} \in \mathscr{P}_1^{(i)}$ for $j = 1,\ldots,D\}$.

**Step 4.** Compute the pairwise Hamming distances between antipodal points:

**for** $d = 1,\ldots,D$ **do** $\mathscr{H}_{\varepsilon}^{(i)} = \mathscr{H}_{\varepsilon}^{(i)} \cup H(\varepsilon p_d^{(i)}, -\varepsilon p_d^{(i)})$

**end for**

**return** $\mathscr{H}_{\varepsilon}^{(i)}$

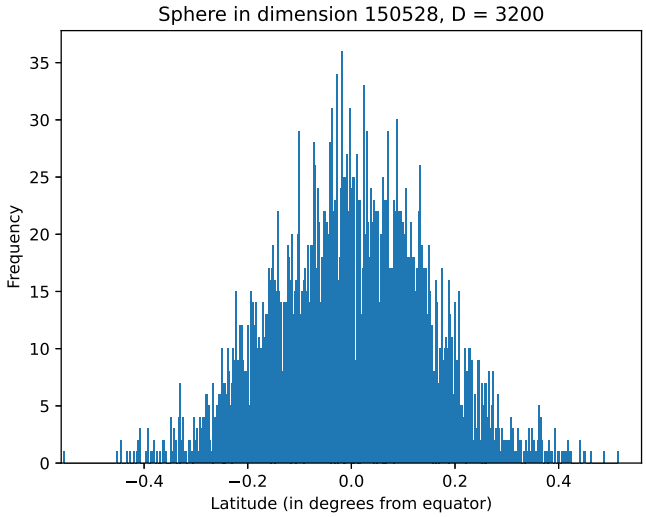



**Figure 5.3:** The angle between the equator (after the arbitrary choice of north pole to be $[1,0,\ldots,0]^\top$) and $D = 3200$ points on the unit sphere of 150528 dimensions that we use to define $\mathscr{P}_{1,j}$ for $j = 1,\ldots,D$ in Algorithm 8.

**ImageNet versus ImageNet-FGSM.**

Here, we use the Hamming distance to show how polyhedral size changes with the employment of the FGSM attack. Given an ImageNet image of interest $I$, we conduct experiments with its associated ImageNet-FGSM counterpart of varying corruption levels. To produce these images using the FGSM attack given in (3.3), take $F$ to be PyTorch's pretrained ResNet-18, $\theta$ to be our model's pretrained parameters, $C$ the cost function used during training, and $\varepsilon_0$ the perturbation parameter. So $x = I(\varepsilon_0)$ and $y = F(x)$. Starting at $I(\varepsilon_0)$, we traverse in the $\text{sign}(\nabla_x C(\theta, x, y))$ and $-\text{sign}(\nabla_x C(\theta, x, y))$ directions a distance of $\varepsilon_i$ and compute the Hamming distance between the resulting images, $I(\varepsilon_i)$ and $I(-\varepsilon_i)$, and the original image $I(\varepsilon_0)$. This experiment is summarized in Figure 5.4 and the results are given in Figure 5.5. The increasing rate of change in the Hamming distance between the original image and an image created with an increasingly small $\varepsilon$ value implies that the polyhedral mesh is more refined about the original image. Conversely, the decreasing rate of change away from $\varepsilon = 0$ implies a less refined decomposition as we move away from the original image. The symmetry of Figure 5.5 implies some level of symmetry in the polyhedral decomposition in these directions.

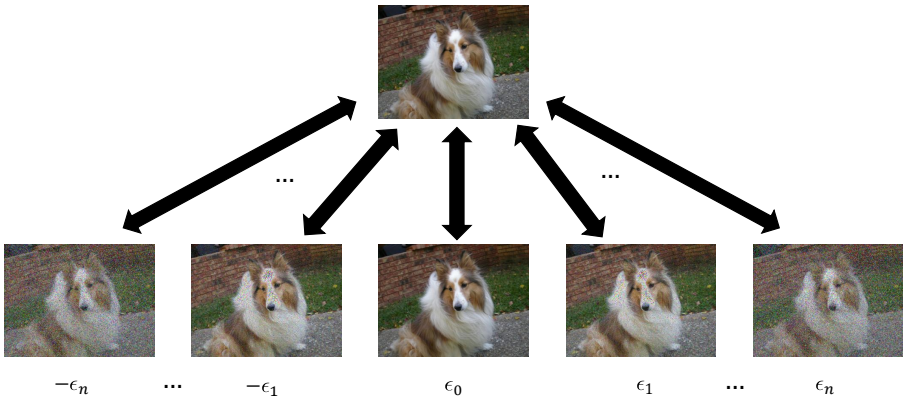



**Figure 5.4:** Summary of Experiment 1.

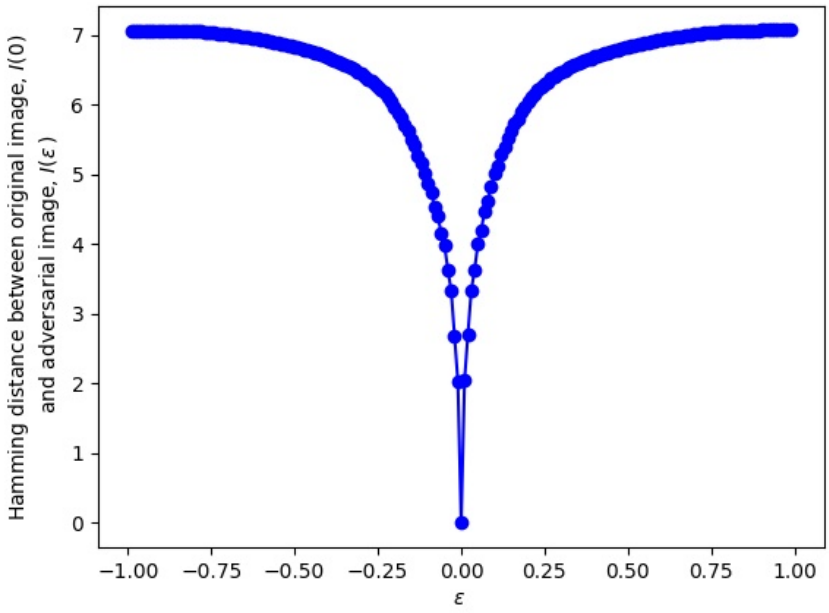



**Figure 5.5:** Hamming distances between the original ImageNet image and its ImageNet-FGSM versions created using a range of perturbation parameters from Experiment 1.

The second experiment we performed is constructing two roughly parallel lines in $\mathbb{R}^{3\times 224\times 224}$. Take $I(\varepsilon_0)$ from the previous experiment and call it $I(1,\varepsilon_0)$. Horizontally flip $I(1,\varepsilon_0)$ and call the resulting image $I(2,\varepsilon_0)$. Starting with $x_1 = I(1,\varepsilon_0)$ traverse in the $\text{sign}(\nabla_{x_1} C(\theta, x_1, y_1))$ and $-\text{sign}(\nabla_{x_1} C(\theta, x_1, y_1))$ directions a distance of $\varepsilon_i$. Simultaneously, starting with $x_2 = I(2,\varepsilon_0)$, traverse in the $\text{sign}(\nabla_{x_2} C(\theta, x_2, y_2))$ and $-\text{sign}(\nabla_{x_2} C(\theta, x_2, y_2))$ directions a distance of $\varepsilon_i$ and compute the Hamming distance between the resulting images, between $I(1,\varepsilon_i)$ and $I(2,\varepsilon_i)$, and between $I(1,-\varepsilon_i)$ and $I(2,-\varepsilon_i)$. This experiment is summarized in Figure 5.6 and its results are given in Figure 5.7. These results again support the claim that the polyhedral mesh is more refined toward original images than adversarial ones. There is an interesting dip at $\varepsilon = 0$; we hypothesize that this is because the original image may lie close to a decision boundary which would corroborate our story in Section 4.3.

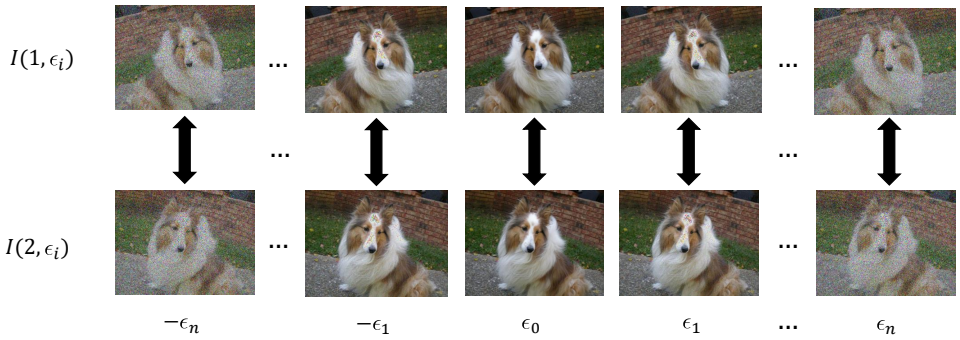



**Figure 5.6:** Summary of Experiment 2.

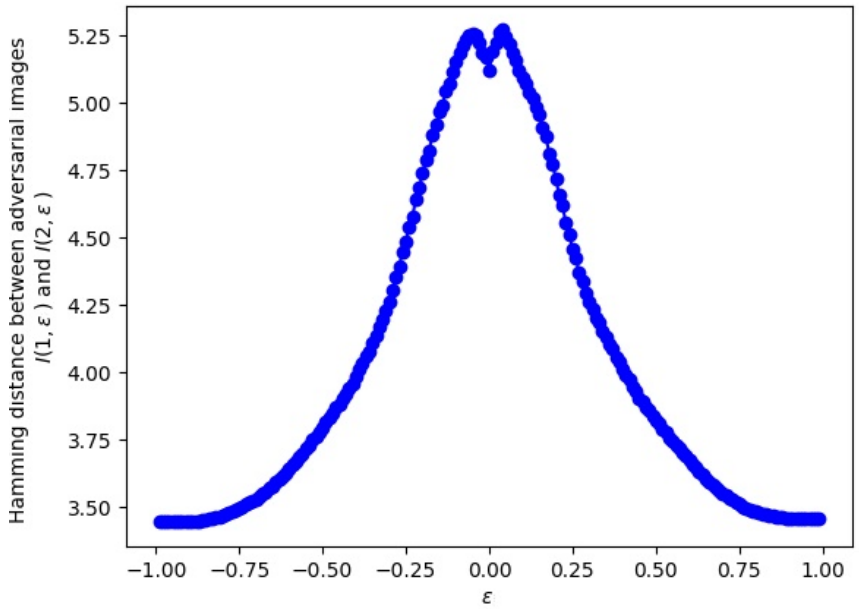
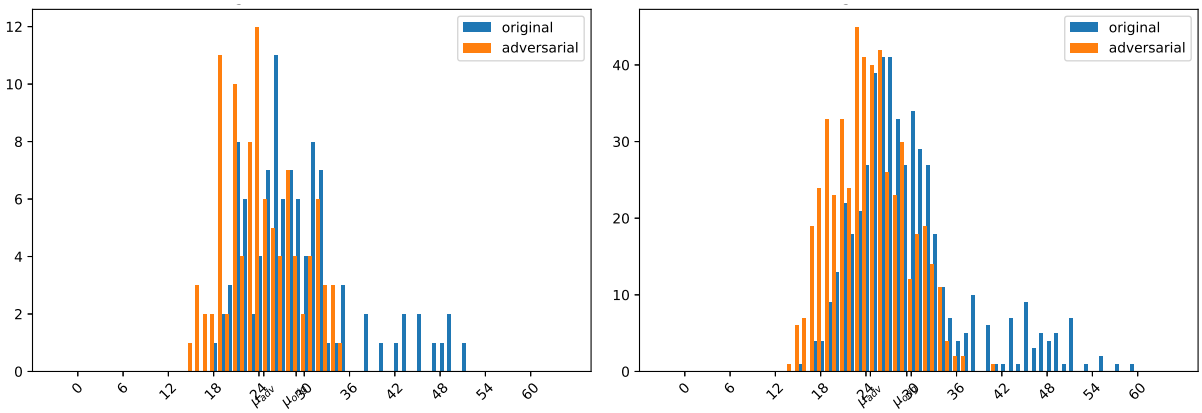
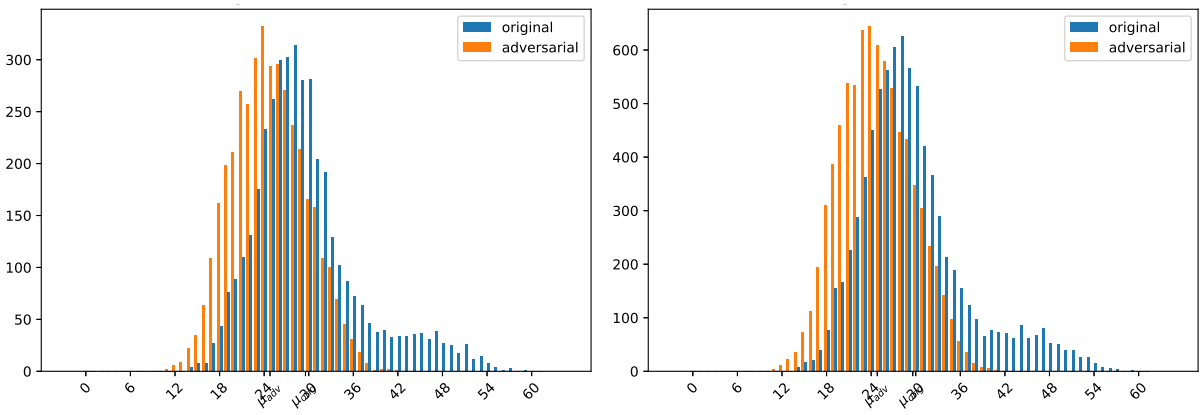


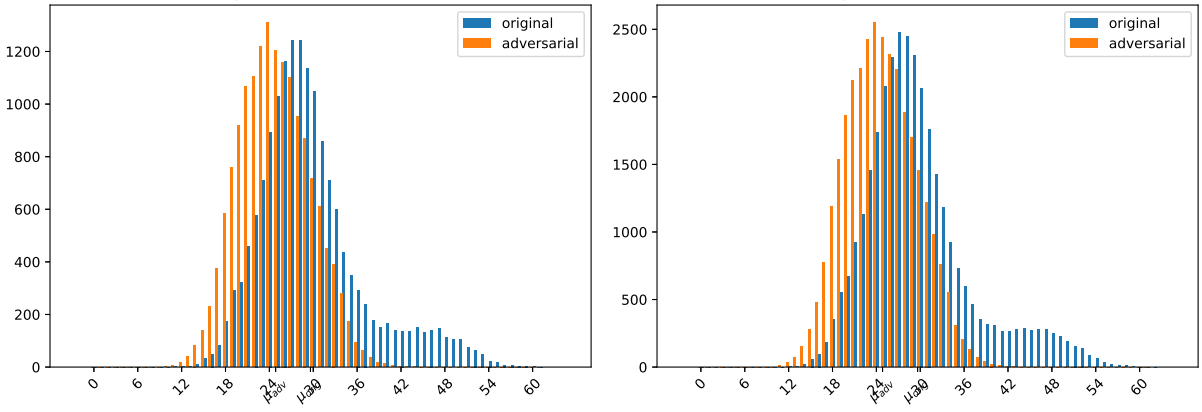

**Figure 5.7:** Hamming distances between $I(1, \varepsilon)$ and $I(2, \varepsilon)$ from Experiment 2.




**(a)** $D = 10$ **(b)** $D = 50$

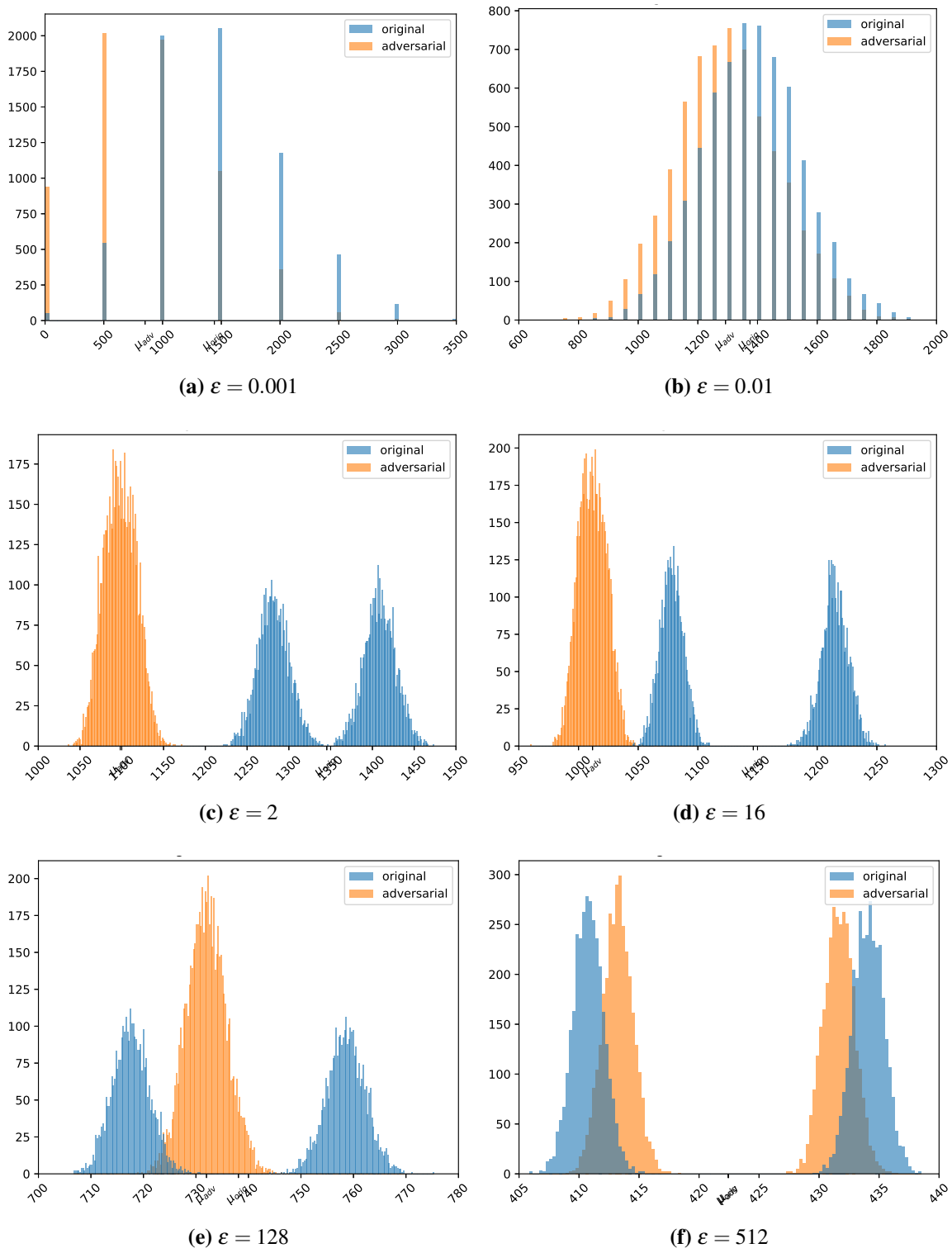



**(c)** $D = 400$ **(d)** $D = 800$




**(e)** $D = 1600$ **(f)** $D = 3200$

**Figure 5.8:** Results from performing Algorithm 8 using $\varepsilon = 0.01$ and varying $D$ between 10 and 3200 where the input points of interest correspond to an ImageNet-DAmageNet pair. The x-axis is the Hamming distances of antipodal points on the hypersphere with radius $\varepsilon$ and the y-axis is the frequency in which they occur.




**(a)** $\varepsilon = 0.001$ **(b)** $\varepsilon = 0.01$

**(c)** $\varepsilon = 2$ **(d)** $\varepsilon = 16$

**(e)** $\varepsilon = 128$ **(f)** $\varepsilon = 512$

**Figure 5.9:** Results from performing Algorithm 8 using $D = 3200$ varying $\varepsilon$ between 0.001 and 512 where the input points of interest correspond to an ImageNet-DAmageNet pair (which are separated by a Euclidean distance of about 496). The x-axis is the Hamming distances of antipodal points on the hypersphere with radius $\varepsilon$ (after scaling by their Euclidean distance of $2\varepsilon$) and the y-axis is the frequency in which they occur.

**ImageNet versus DAmageNet.**

We use Algorithm 8 in this section to probe a neural network for polyhedral size information in the vicinity of ImageNet and DAmageNet images. We first experimentally show that the choice $D = 10$ is sufficiently large for ResNet-18. We experiment with values of $\varepsilon$ ranging between 0.001 and 512 and values of $D$ ranging between 10 and 3200. Fixing an $\varepsilon$ value and an ImageNet-DAmageNet pair while varying the value of $D$, the respective means of the Hamming distance distributions about the original and adversarial images remain relatively constant. Results from this experiment on the third ImageNet-DAmageNet pair, whose Euclidean difference is about 496, are shown in Figure 5.8. We also fix $D = 3200$ and an ImageNet-DAmageNet pair while varying $\varepsilon$ to measure the Hamming distance of antipodal points of concentric hyperspheres, and then divide these Hamming distances by the diameter of their associated hyperspheres, scaling them so that results from hyperspheres of different sizes are comparable. Results from performing this experiment on the third image pair are given in Figure 5.9. The mean Hamming distance (after scaling by Euclidean distance) from measurements made about the DAmageNet image is smaller than that of its ImageNet counterpart, implying that the polyhedra about the DAmageNet image are larger than those about the ImageNet image. This is actually a highly distinguishing feature between the original and adversarial images of an ImageNet-DAmageNet pair. Fixing $\varepsilon = 1$, 87.05% of 6200 image pairs exhibit this behavior. As an interesting side note, the image pairs that have this property are closer to each other in the input space than those that do not (the average Euclidean distance between images satisfy $\mu_{adv} < \mu_{orig}$ is 485.79 and 564.46 between those that do not). Therefore, ImageNet and DAmageNet are accurately distinguishable by the local polyhedral decomposition structure of ResNet-18 when considering a single image pair at a time. This is an interesting fact since (a) ResNet-18 was not used to create DAmageNet and (b) no images from any of the ImageNet-DAmageNet pairs were seen during the training of ResNet-18. We summarize the results from 6200 image pairs for varying $\varepsilon$ values in Table 5.1. The distribution of means of Hamming distances between 100 antipodal points on the hypersphere of growing radius for all 6200 image pairs is given in Figure 5.10, showing that it is necessary to consider one ImageNet-

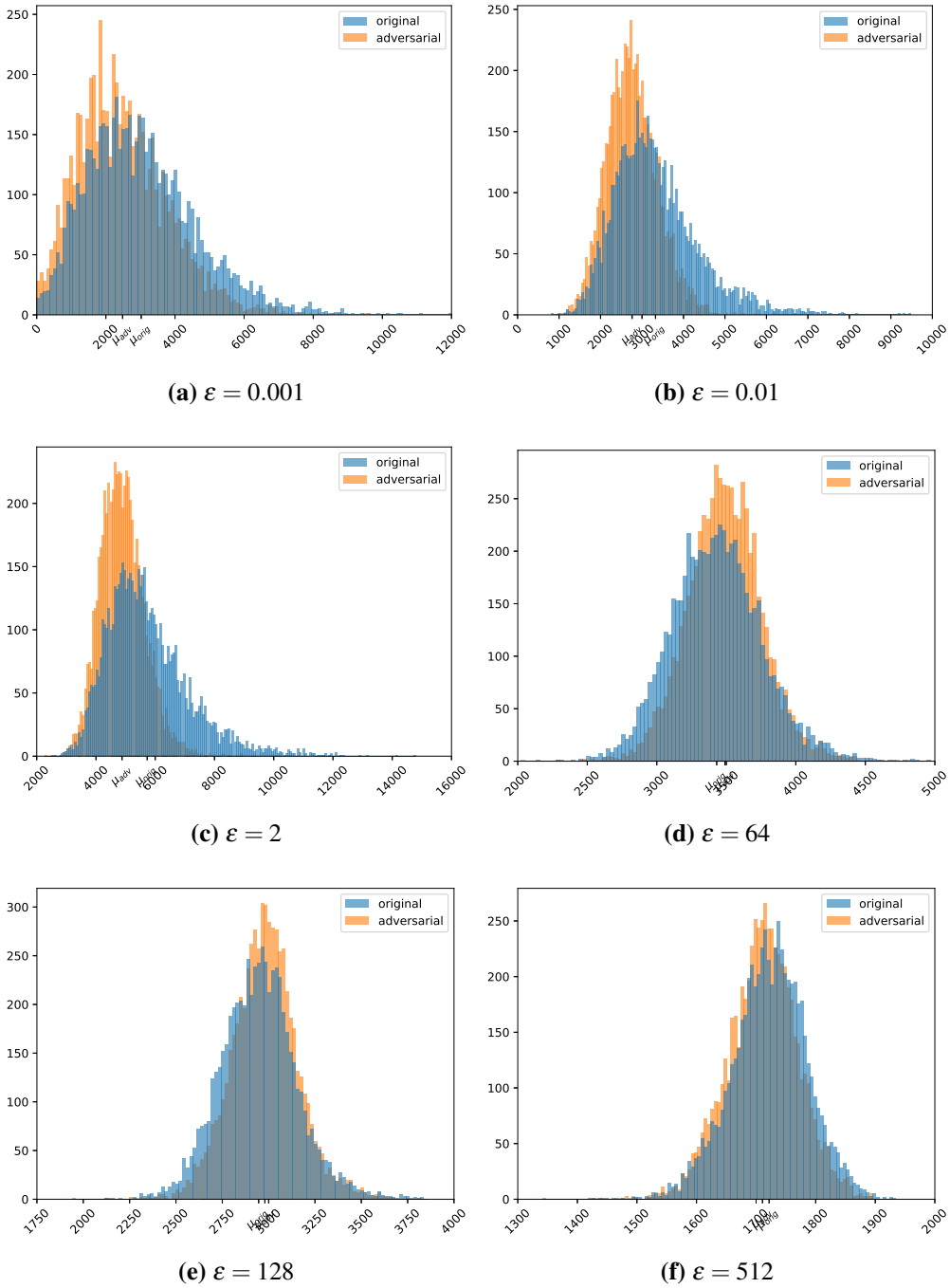
DAmageNet pair at a time to identify the adversarial sample by polyhedral size alone. Figure 5.10 also shows that polyhedra tend to get bigger away from the images in the 6200 image pairs (as seen in the decreasing transition density, given by the x-axis, with an increase in hypersphere radius, $\varepsilon$).

**Table 5.1:** (Column A) The percentage of ImageNet-DAmageNet pairs satisfying $\mu_{adv} < \mu_{orig}$ for Hamming distance measures between antipodal points of concentric spheres, (Column B) the Euclidean distance between the images of an ImageNet-DAmageNet pair satisfying this condition, (Column C) the Euclidean distance between the remaining image pairs.

| | Column A | Column B | Column C |
|---|---|---|---|
| $\varepsilon = 0.001$ | 60.95 | 490.29 | 504.86 |
| $\varepsilon = 0.01$ | 73.39 | 487.89 | 518.27 |
| $\varepsilon = 0.1$ | 84.71 | 486.72 | 547.28 |
| $\varepsilon = 1$ | 87.05 | 485.79 | 564.46 |
| $\varepsilon = 2$ | 86.87 | 485.74 | 563.73 |
| $\varepsilon = 4$ | 85.44 | 484.29 | 564.53 |
| $\varepsilon = 8$ | 82.74 | 484.02 | 553.32 |
| $\varepsilon = 16$ | 74.69 | 484.54 | 529.72 |
| $\varepsilon = 32$ | 54.29 | 490.88 | 502.03 |
| $\varepsilon = 64$ | 37.05 | 512.15 | 486.46 |
| $\varepsilon = 128$ | 36.84 | 526.69 | 478.06 |
| $\varepsilon = 256$ | 49.00 | 522.47 | 470.52 |
| $\varepsilon = 512$ | 62.29 | 508.90 | 474.64 |

The results from Algorithm 8 can be post-processed in a way that isolates the polyhedral behavior in the concentric rings from that in the concentric spheres (see Section 5.2.3 for details). Doing this post-processing for all 6200 image pairs previously considered, we observe a nuanced difference between the polyhedral decomposition about the original and adversarial images. Figure 5.11 shows that (a) polyhedra get significantly larger away from these images (even larger than Figure 5.10 seems to indicate), (b) between $\varepsilon = 2$ and $\varepsilon = 64$, the polyhedral decomposition about DAmageNet images tends to actually be more refined than that about their ImageNet versions, (c) between $\varepsilon = 64$ and $\varepsilon = 128$ their polyhedral decompositions tend to be fairly identical, and (d) between $\varepsilon = 128$ and $\varepsilon = 512$ the polyhedral decomposition reverts to being less refined about the DAmageNet images. Table 5.2 summarizes these differences. We speculate that this could be because the ImageNet validation images are closer than DAmageNet images to the neural network's

learned decision boundaries where we observed (in Section 4.3) that polyhedral decompositions exhibit high polyhedral size variance.

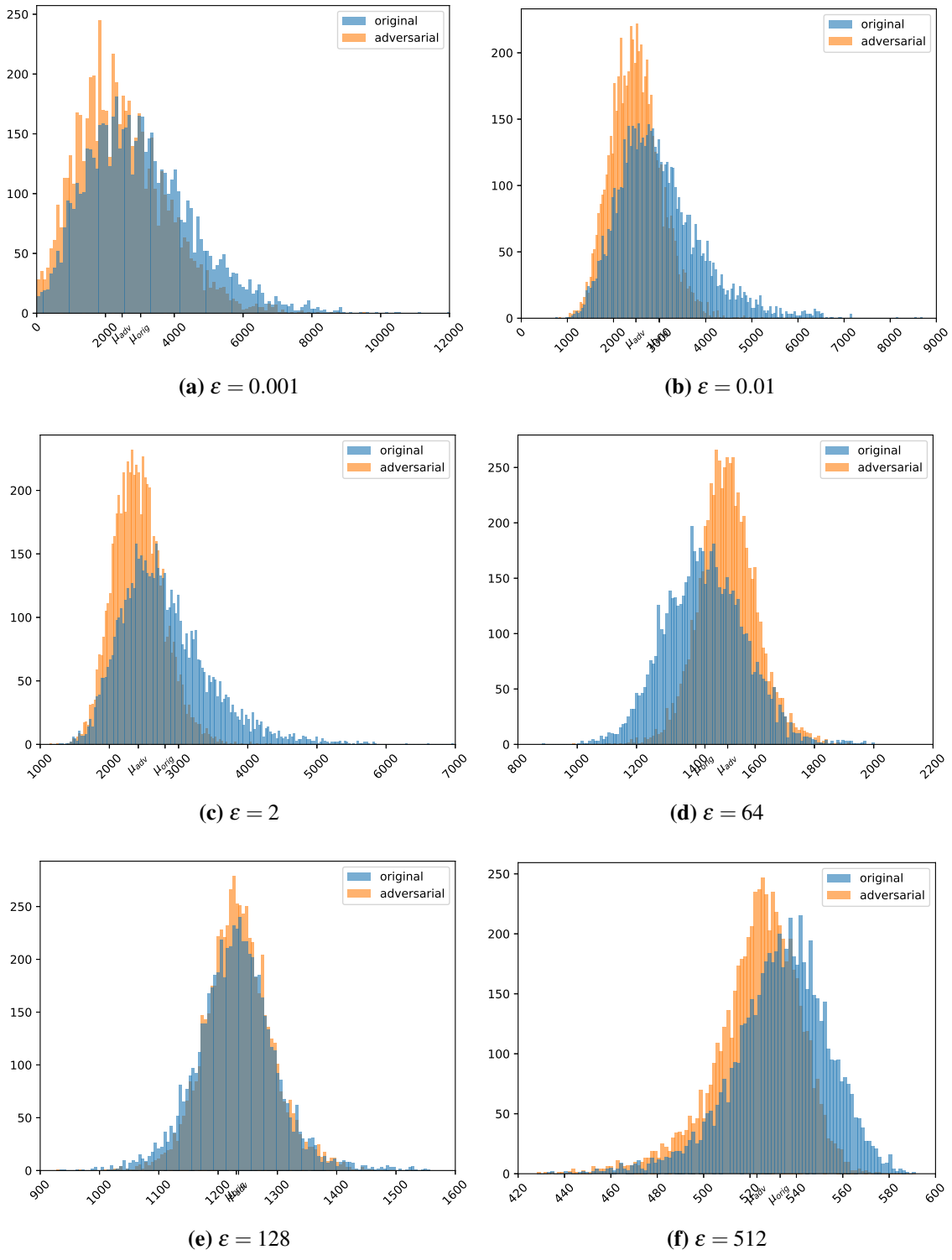



**(a)** $\varepsilon = 0.001$ **(b)** $\varepsilon = 0.01$

**(c)** $\varepsilon = 2$ **(d)** $\varepsilon = 64$

**(e)** $\varepsilon = 128$ **(f)** $\varepsilon = 512$

**Figure 5.10:** Results from performing Algorithm 8 for 6200 ImageNet-DAmageNet pairs using $D = 50$, varying $\varepsilon$ between 0.001 and 512, and computing the mean of the resulting distributions. The x-axis is the Hamming distances of antipodal points on the hypersphere with radius $\varepsilon$ (after scaling by their Euclidean distance of $2\varepsilon$) and the y-axis is the frequency in which they occur.




**(a)** $\varepsilon = 0.001$ **(b)** $\varepsilon = 0.01$

**(c)** $\varepsilon = 2$ **(d)** $\varepsilon = 64$

**(e)** $\varepsilon = 128$ **(f)** $\varepsilon = 512$

**Figure 5.11:** Results from performing Algorithm 8 for 6200 ImageNet-DAmageNet pairs using $D = 50$, varying $\varepsilon$ between 0.001 and 512, and computing the mean of the resulting distributions. The x-axis is the Hamming distances of antipodal points of the ring whose outer sphere has radius $\varepsilon$ (after scaling by the ring's thickness) and the y-axis is the frequency in which they occur.

**Table 5.2:** (Column A) The percentage of ImageNet-DAmageNet pairs satisfying $\mu_{adv} < \mu_{orig}$ for Hamming distance measures between antipodal points of concentric rings, (Column B) the Euclidean distance between the images of an ImageNet-DAmageNet pair satisfying this condition, (Column C) the Euclidean distance between the remaining image pairs.

| | Column A | Column B | Column C |
|---|---|---|---|
| $\varepsilon = 0.001$ | 60.95 | 490.29 | 504.86 |
| $\varepsilon = 0.01$ | 73.44 | 488.73 | 516.00 |
| $\varepsilon = 0.1$ | 84.56 | 486.03 | 550.45 |
| $\varepsilon = 1$ | 87.29 | 485.56 | 567.52 |
| $\varepsilon = 2$ | 86.15 | 484.88 | 564.99 |
| $\varepsilon = 4$ | 83.37 | 483.62 | 557.95 |
| $\varepsilon = 8$ | 76.90 | 484.55 | 534.01 |
| $\varepsilon = 16$ | 52.66 | 493.36 | 498.89 |
| $\varepsilon = 32$ | 25.34 | 516.70 | 488.94 |
| $\varepsilon = 64$ | 20.16 | 542.60 | 484.20 |
| $\varepsilon = 128$ | 46.27 | 532.80 | 464.26 |
| $\varepsilon = 256$ | 78.58 | 497.55 | 490.19 |
| $\varepsilon = 512$ | 80.73 | 489.93 | 521.31 |

### 5.2.3 Implementation Details

Performing Algorithm 8 for several ImageNet-DAmageNet pairs with $\varepsilon = 0.001$ frequently yielded Hamming distances of 0, so we consider 0.002 a reasonable estimate for polyhedral widths from the decomposition associated with PyTorch's pretrained ResNet-18. Therefore, starting with $\delta = 1$ in Algorithm 7, we can expect about 10 iterations to be necessary $\big($because a width of 0.002 is equivalent to a radius of 0.001, $0.001 \approx 2^{-9.97}\big)$ in each of the two directions, or 20 iterations to complete Algorithm 7. The initialization of $\delta$ can be modified based on preliminary width measurements for a particular neural network to save computation time. That being said, computing an approximate polyhedral size statistic as in Algorithm 8 is faster than computing an exact polyhedral width as in Algorithm 7. This should be expected as Algorithm 7 is essentially the same as iteratively performing Algorithm 8 until two conditions are met. Running Algorithm 7 1000 times parallelized on 16 V100s took about 1400 seconds, so we estimate that without parallelization, it would take 1400 seconds to do 62.5 of these computations or that each computation would take

about 0.045 seconds. Meanwhile, running Algorithm 8 for 1000 directions without parallelization took, on average, about 30 seconds, or 0.03 seconds each.

It is important to note that PyTorch does not automatically delete functions' local variables after executing them as Python does. Memory leaks impose huge and unnecessary memory demands and computational slow-downs that can be avoided by explicitly deleting these variables and clearing the cache. Consider performing Algorithm 7 for ResNet-18 with memory leaks: if 10 iterations are required and two bit vectors are calculated per iteration, then 32M bits of memory are required to save the computed bit vectors alone!

When experimenting with the parameter $D$ in Algorithm 8, we constructed $\mathscr{P}_1$ as a nested sequence: $\mathscr{P}_{1,1:j} \subset \mathscr{P}_{1,1:k}\ \forall j < k$. We did this for computational efficiency and repeatability of our results. When computing $\mathscr{H}^{(i)}_{\varepsilon,1:k}$ we reused $\mathscr{H}^{(i)}_{\varepsilon,1:j}$, and only performed $k-j$ new Hamming distance computations (namely, using the points $p^{(i)}_j \in \mathscr{P}_{1,j+1:k}$) and appended these measurements to $\mathscr{H}^{(i)}_{\varepsilon}$. For repeatability, we fix the random seed with `numpy.random.seed(0)`. After choosing a seed and producing $N_k$ and $N_j$ by drawing $k$ and $j$ random samples from the normal distribution $N(\mu,\sigma) = N(0,1)$, the first $j$ of the $k$ samples in $N_k$ will be identical to $N_j\ \forall j < k$, giving us the nested property of our sets $\mathscr{P}_{1,1:D}$ with the increase in parameter $D$. We note that this nesting quality implies that Figure 5.3 also conveys some information about how our particular samples on the unit sphere of 150528 dimensions concentrate about the equator for not just $D = 3200$, but also for $D \leq 3200$.

Lastly, we include details for the post-processing done on the results from Algorithm 8 that were stated to "isolate the polyhedral behavior in the concentric rings from that in the concentric spheres." Given two of these concentric spheres centered at $x^{(i)}$ with radii $\varepsilon_0$ and $\varepsilon_1$ with $\varepsilon_0 < \varepsilon_1$, since we fixed the directions we consider (with `numpy.random.seed(0)`), then the samples we took from the sphere with radius $\varepsilon_0$ have a corresponding sample on the sphere with radius $\varepsilon_1$. Without loss of generality, consider the sample $x_0 = \varepsilon_0 v$ on the sphere with radius $\varepsilon_0$ for some direction $v$. Then its corresponding sample on the sphere with radius $\varepsilon_1$ is $x_1 = \varepsilon_1 v$. To isolate the portion of the polyhedral decomposition that is contained by the sphere with radius $\varepsilon_1$ from that

which is contained also by the sphere with radius $\varepsilon_0$, we can approximate the Hamming distance between $x_0$ and $x_1$ by subtracting $H\left(x_0, x^{(i)}\right)$ from $H\left(x_1, x^{(i)}\right)$. We can do this for all corresponding pairs simultaneously by computing $\mathscr{H}_{\varepsilon_1}^{(i)} - \mathscr{H}_{\varepsilon_0}^{(i)}$ which we consider instead of $\mathscr{H}_{\varepsilon_1}^{(i)}$.

## 5.3 Input Space Landscapes

As alluded to in Section 2.2, *input space landscapes* are the result when we cut a two-dimensional slice out of our high-dimensional input space and visualize the data that it intersects. For our input space landscapes, these data are properties of the neural network when points lying on this plane are fed through the model. We visualize these landscapes by coloring their corresponding intrinsically two-dimensional planes according to the neural network property of interest, which adds a third dimension to these objects. Therefore, input space landscapes are intrinsically three-dimensional with an intrinsically two-dimensional parametric domain because we choose two orthonormal basis vectors to define the domain slice of interest. These landscapes are a method of implicitly accessing the neural network's polyhedral decomposition because the polyhedral decomposition is required to exactly partition the input space by input-output Jacobian norm and to draw exact decision boundaries, while here, we collect these data by coarsely sampling the input space. We are interested in both global and local behaviors of neural networks and explore them, in part, using input space landscapes. For globally/locally smooth input space landscapes, we say that the corresponding neural network appears to be globally/locally stable—small perturbations in the domain result in small perturbations in the metric of interest. Conversely, for landscapes that are characterized by dramatic terrain, we say their models are unstable, unreliable, or chaotic.

The highest-dimensional input space we consider is $\mathbb{R}^{3\times 224\times 224}$, so the choice of parameter space is no longer immediate, and in our particular work, they are also not unique. Since a hyperplane in the $m$-dimensional Euclidean space $\mathbb{R}^m$ is defined in the form

$$a_1 x_1 + a_2 x_2 + \cdots + a_m x_m + a_{m+1} = 0$$

where $a_i$ are all constants, $i = 1, \ldots, m, \exists i$ such that $a_i \neq 0$, and $x_i$ are variables for which

$$(x_1, \ldots, x_m) \in \mathbb{R}^m,$$

$n$ non-collinear points are required to define an $n$-dimensional hyperplane uniquely [88]. Substantial resources would be required to produce this number of high-dimensional samples in some cases, and in others, this number of samples does not even make sense (discussed in further detail shortly). Consequently, we can leverage two classical algorithms and a theorem of high-dimensional space to produce the basis vectors of high-dimensional parameter spaces of interest: Singular Value Decomposition (see Section 2.2.1), Gram Schmidt (see Section 2.2.2), and Orthogonality of Large Random Vectors (see Section 2.1.3). In our work, we only use SVD, but these other two methods could be used to make complementary visualizations to give a more complete picture of neural network behavior in the region of interest. Since we seek intrinsically two-dimensional representations of the input space to create input space landscapes, we choose $k = 2$ when using each of these three methods. In this section, we introduce two input space landscapes—input-output Jacobian norm and prediction landscapes—and use them to analyze neural network behaviors similar to the observational study in Section 4.3 but for a much larger model, ResNet-18.

### 5.3.1 Input-Output Jacobian Norm Landscapes

Because the input-output Jacobian norm seems to be a key feature of a neural network that changes during training, we believe there is much to be learned from probing neural networks for this information, research that does not yet seem to exist. The only other work that uses input-output Jacobian norms of a given network computes them along a trajectory of the input space [65]. They show that for their particular model—their model trained until perfect performance on a data-augmented version of MNIST—the input-output Jacobian norm crashes at training images and generally peaks in between those belonging to different classes. In Section 4.3, we showed how polyhedral decompositions adapt as their respective models are trained. We concluded that input-output Jacobian norms tend to increasingly spike at learned decision

boundaries as training is prolonged. Using training data like MNIST that is not linearly separable, increasing the dataset's size using data augmentation, and then demanding perfect performance on this expanded training dataset is sure to lead to the allocation of substantial Jacobian resources between classes, as shown in [65]. Modern training schemes, on the other hand, specifically avoid over-fitting neural networks (most commonly with the use of dropout, early-stopping, or some slight variant of these) to increase their generalizability [68, 89–93]. PyTorch's pretrained ResNet models were trained for a predefined number of epochs (see https://github.com/pytorch/vision/blob/main/references/classification/train.py for details), so we do not expect their 'Jacobian resources' to condense along classification boundaries to the same extent as they would for overfit models. Indeed, we show that this is true. We show six Jacobian norm landscapes in Figure 5.12, each going through three training images. One can see that there does not exist an elliptical path through them that has the properties present in the results of [65]. What we do consistently see in our plots (and in the polyhedral size plots in the Appendix D) is evidence that the neural network has allocated significant resources during training about the origin, suggesting that its results on dim images should be handled with caution. This same volatility about the origin can be observed in prediction landscapes, which we discuss next.

### 5.3.2 Predictive Landscapes

Neural networks have long been known to be susceptible to adversarial attacks (see [94] for a comprehensive survey of adversarial attacks and their success). This is because of their predictive instability under perturbations in the input space. We explore the relationship between changes in neural network domain and the neural network's predictions with the use of predictive landscapes. While the methods used in previous sections require access to a neural network's architecture (namely their bit values and the norm of the matrix $A$ where $y = Ax + B$ for some input-output pair $(x, y)$ and $A, B$ are given in (4.17)), this method does not.

Consider the first image pair given in Figure 3.4. From these two images, we define three planes in image space: planes through (a) both images, (b) five linearly independent variants of

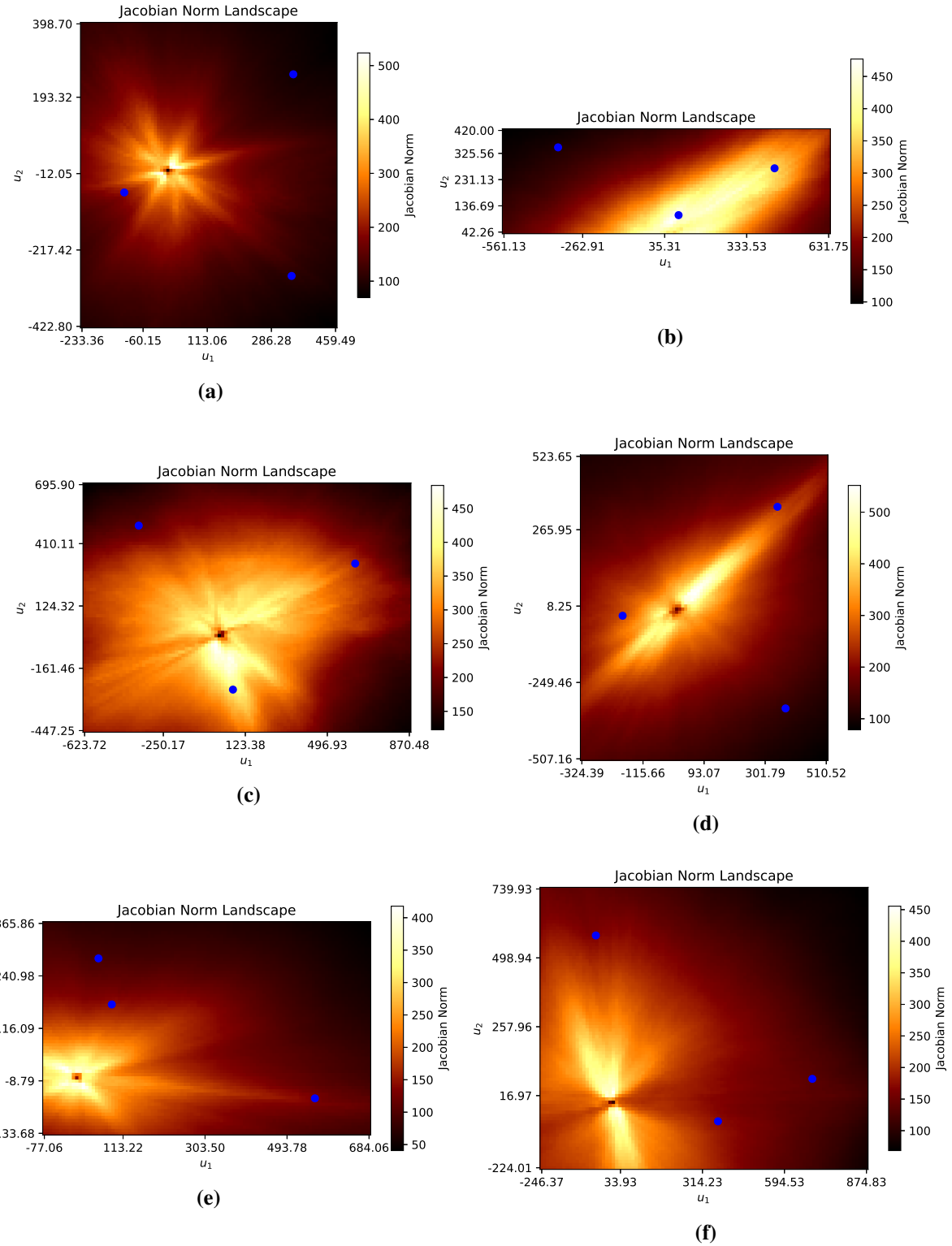



**Figure 5.12:** Input-output Jacobian norm landscapes through three ImageNet training images. The landscapes in plots (a)-(c) go through three images of the same class while the landscapes in plots (d)-(f) go through images of distinct classes.

the ImageNet image, and (c) five linearly independent variants of the DAmageNet image where the linearly independent collection of images is produced by performing nonlinear transforms on the image of interest (see 5.3.3 for details). These three planes are then discretized and sampled; the predictions by ResNet-18 of all samples are recorded and visualized in Figures 5.13, 5.14, and 5.15, respectively. Notice that even a slight perturbation in the ImageNet image in the direction of the DAmageNet image would lead to its misclassification. Additionally, a set of five images made by perturbing an ImageNet image is shown to lie in the classification region of multiple classes. Recalling that each polyhedron defines a distinct map through a neural network, the image (as in the destination of this mapping) of regions with smaller polytopes has more potential for belonging to multiple classes. In Section 5.2.2, we showed that polyhedra in the vicinity of DAmageNet images tend to be larger than those about their ImageNet counterparts. Those two facts along with the comparison of a few preliminary visualizations similar to Figures 5.14 and 5.15 lead us to the question: do DAmageNet images lie in regions of higher predictive convexity than ImageNet images?

To measure $PC^{(i)}$, the predictive convexity of the $i^{\text{th}}$ image in an ordered set of $K$-many images, we produce a collection of six images: the image of interest, five linearly independent variants, together with the origin (an all-black image). We draw lines connecting all possible pairs of these six images (resulting in $C(n,k) = C(6,2) = 15$ lines) and sample along each of them such that their step size is as close to 1 as possible while ensuring that the endpoints are sampled. We ensure this with the thought that the predictions of the images themselves should be taken into account in our calculation. Say that this results in a total of $N^{(i)}$ samples from all 15 lines, and we get the model's predictions for these $N^{(i)}$ images.

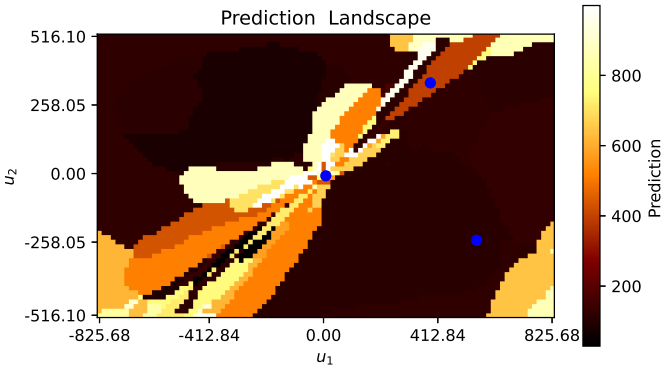



**Figure 5.13:** Prediction landscape centered about the origin through an ImageNet-DAmageNet pair with the ImageNet image at the top right and the DAmageNet image at the bottom right.

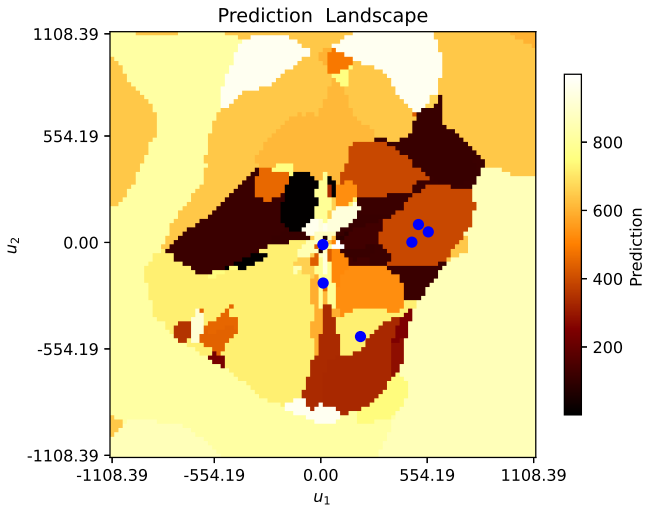



**Figure 5.14:** Predictive landscape centered about the origin through four linearly independent variants of the ImageNet image together with the ImageNet image itself.

From this collection of predictions, we identify which class is predicted the most, say it consists of $\nu^{(i)}$ of the $N^{(i)}$ predictions, then our predictive convexity statistic is given by $PC^{(i)} = \nu^{(i)}/N^{(i)}$. To our surprise, this statistic yielded no difference between the ImageNet and DAmageNet datasets. More specifically, we stacked the predictive convexity statistics of the ImageNet and DAmageNet images into their own vectors, respecting the order given to their images, and subtracted the DAmageNet vector from the ImageNet one. Due to the image ordering, this compared the predictive

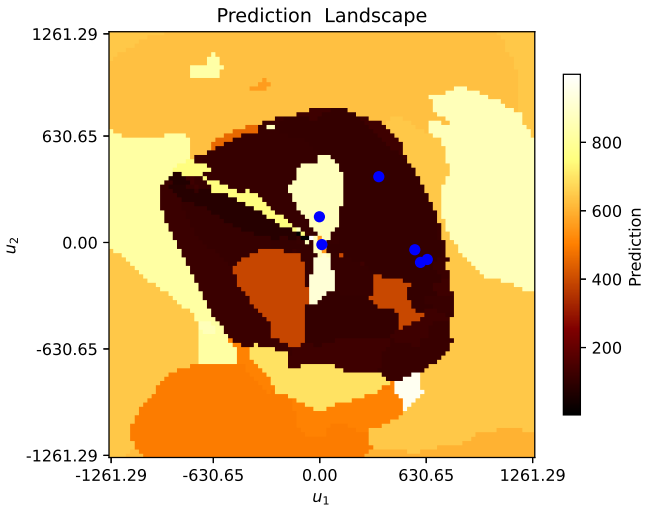



**Figure 5.15:** Predictive landscape centered about the origin through four linearly independent variants of the DAmageNet image together with the DAmageNet image itself.

convexity between images belonging to the same ImageNet-DAmageNet pair. The result had a mean of 0.016, a median of 0.013, and ranged between the values of $-0.520$ and 0.564. Therefore, the stability of the prediction map of ResNet-18 seems very similar about ImageNet and DAmageNet images.

### 5.3.3 Implementation Details

From a single image of interest, we produce a set of five linearly independent images by retaining the image as it is, adjusting its pixel-value histogram such that a uniform distribution of grayscale values is achieved, applying the Power Law Transform, increasing its contrast, and applying the sigmoid function to it. Each of these four nonlinear transforms is done using pre-built functionalities: `equalize`, `adjust_gamma` with `gamma=0.8`, `adjust_contrast` with `contrast_factor=1.2` (which increases the contrast of the image by a factor of 0.2) all from the `torchvision.transforms.functional` package, and `torch.nn.Sigmoid()`, respectively.

As mentioned in Section 2.2.1, SVD suffers from sign indeterminacy. To keep visualizations consistent, we would ideally compute the basis vectors that define a plane through images of interest and compute all landscapes at once. Of course, bugs occur and individual landscapes need to be recomputed. To do this efficiently, we compute the basis vectors for our images of interest,

produce a bounded plane using these basis vectors, and compare its boundaries to those in any one of the correct visualizations (which we strip from the visualization using `PyPDF2.PdfReader`). If the boundaries differ by a sign, we multiply both of our current basis vectors by $-1$, but if they are numerically equal to each other, we keep them as they are.

## 5.4 Conclusion

In this chapter, we have given a few examples of how polyhedral decompositions can be implicitly accessed through their bit vectors and, less directly, through input space landscapes. Viewing bit vectors in terms of the geometric implications of each individual bit, we identified splines that tend to separate original images from adversarial ones. From these splines, we developed a classifier that differentiates between these two image types with over 90% accuracy. Not handling bit vectors with this level of element-wise specificity as we do by computing the Hamming distance, we also showed that polyhedra containing adversarial images tend to be bigger than those that contain their original counterparts. More specifically, adversarially perturbing images move them to regions of lower refinement in the polyhedral decomposition about 87% of the time. The goal of this work was not necessarily to distinguish between original and adversarial images (if it were, we would have trained a neural network specifically for this task), but to detect signals toward this end that already exist in a neural network trained to perform an entirely different task. After these analyses of polyhedral size, we also probed ResNet-18 for information about its input-output Jacobian norm and decision boundaries, which the results in Chapter 4 indicate have a direct correlation. We did this using input space landscapes which visually demonstrate the variability in neural network predictions under perturbations in image space and the complexity of the input-output Jacobian norm of a neural network trained on ImageNet using a state-of-the-art training scheme. Since the observations we made in Chapter 4 with the explicit use of polyhedral decompositions are so new to the field, we suspect that we have only seen the tip of the iceberg as far as what insights the implicit access of polyhedral decompositions can offer about their respective neural networks.

# Chapter 6

# Loose Ends

## 6.1 Introduction

Exploring intriguing, open-ended questions can frequently present obstacles that prompt us to reframe the initial inquiry or shift focus. Even more often, we are led to follow-up questions which tend to multiply with every found answer. This chapter compiles all such loose ends of our research. Before taking a polyhedral approach to neural networks and reaping dividends toward uncovering their underlying mechanics, we started with the data seriation problem, which seeks to give an order to data of choice. This work is covered in Section 6.2. Availability (or lack thereof) of data particularly relevant to this problem was a hurdle we faced, leaving some interesting questions unanswered—substantial and likely proprietary data would be required to begin to answer them. With this setback, we pivoted subject matter altogether to the analysis of neural networks. The next and final section in this chapter, Section 6.3, investigates the generalizability and trainability of models from a few different angles. Pulling inspiration from loss landscapes, visualizations that convey the performance of a vast collection of model instances with the same architecture, we look at (a) the trajectory of model parameters during training and (b) the level sets of these landscapes which prescribe model parameter values for models of similar performance. Also toward understanding model generalizability and trainability, we focus on the effects of two architectural features: skip connections and model depth. In the early stages of these analyses, we were interested in models' input-output Jacobians. We handled and performed computations with these massive objects which led to some interesting implications about skip connections. To overcome some of the difficulties in manipulating these huge matrices, we proceeded to consider merely their norms, which surprisingly still had a lot to say for themselves as we showed in Section 4.3. Next, we considered depth. The unanswered questions we pose regarding model depth are ones that we find most exciting because they (a) show promise in the limited relevant literature and

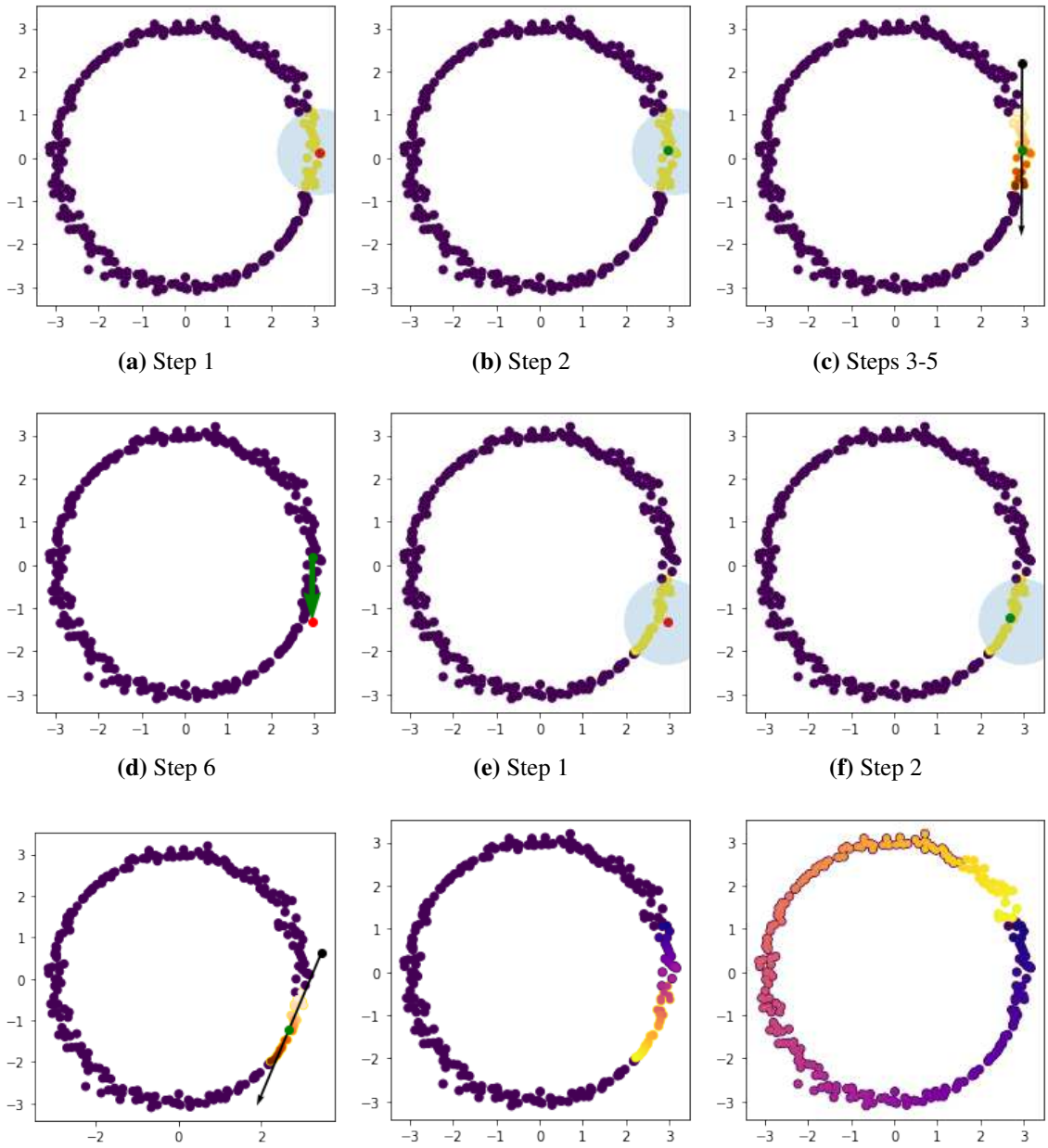
(b) pull on concepts from previous chapters, which will bring us to the end of our work.

**Novel contributions in this chapter:**

- present an implementation of a data seriation algorithm intended for data with a local intrinsic dimension of one and discuss two interesting applications of interest,
- treat each parameter value over the course of training as its own trajectory and compute its curvature,
- formally express the optimization problem satisfied by models of a certain performance level,
- use network input-output Jacobians to explore the effects of skip connections on network topology, and
- use loss landscapes to visualize the compatibility of different loss functions for fine-tuning.

## 6.2 Data Seriation

In this section, we consider data that is intrinsically one-dimensional with an inherent order (like frames in a video) or a natural order after the choice of initial point and direction [95]. Can a trajectory be constructed from a shuffled series of location data or can a video be reconstructed after its frames have been shuffled? We first consider points densely sampled from a circle with the addition of limited noise. For data such as this, our baseline Data Seriation algorithm, pictured in Figure 6.1 and more formally given in Algorithm 9, will suffice. In this algorithm, we iteratively identify local regions of data by Euclidean proximity, assume that their principal curve [96] is linear, and order the points based on their projections onto their principal curve, rectify this ordering with the global order given to points previously seen in the algorithm, move along a chosen direction of the trajectory, and repeat. Moving from smooth data in Figure 6.1 to real data, what modifications to our baseline algorithm should be expected? Consider a video taken from a relatively constant vantage point represented as a trajectory. For example, consider the video



**(a)** Step 1 **(b)** Step 2 **(c)** Steps 3-5

**(d)** Step 6 **(e)** Step 1 **(f)** Step 2

**(g)** Steps 3-4 **(h)** Step 5 **(i)** Global order at the end of the algorithm

**Figure 6.1:** Baseline data seriation procedure Algorithm 9 after arbitrarily choosing one of the data points (3.13, 0.13) as the starting point labeled by their corresponding step of the algorithm.

**Algorithm 9** Data Seriation (baseline)

Given a random initial point $c_i,\ i = 0$, a choice of $\varepsilon$ and of projection constant $k$, a dataset horizontally stacked into a matrix $\mathbf{X} \in \mathbb{R}^{m \times d}$, initialize a dictionary $Y$, where the global ordering of the data will be saved, with $d$ keys $Y_k$ that are equal to the $m$-dimensional points in $X$. Let $Y_v$ denote the values of $Y$.

**while** $|Y_v| < d$ **do**

**Step 1.** Let $S_i = \{x \in \mathbf{X} : \|x - c_i\|_2 < \varepsilon\}$ and let $M$ be the matrix with the column vectors of $S_i$ horizontally stacked together.

**Step 2.** Compute $\mu_i$, the mean of $M$.

**Step 3.** Compute $u_1$ the principal component of $M$, its first left singular vector, and arbitrarily choose its direction.

**Step 4.** Locally order the points of $M$ with respect to $u_1$ by (a) defining a vector $v_i$ in the direction of $u_1$ with magnitude of at least $2\varepsilon$ centered through $\mu_i$, (b) projecting the points of $M$ onto $v_i$, and (c) giving them a local order with respect to their distance from the source of $v_i$.

**Step 5.** Rectify the local ordering with the global order in $Y$:

**if** $i = 0$ **then** the global order is exactly the local order.

**else** call $k$ the last global index of the data in $S_{i-1} \backslash S_i$.

**if** the ordering of $S_{i-1} \cap S_i$ differ **then** rectify this difference by either (a) taking the local ordering of $S_i$ to be the new global ordering starting with index $k+1$ or (b) taking the average between $v_{i-1}$ and $v_i$ and ordering the points of $S_{i-1} \cap S_i$ along it (as is done in Step 4).

**end if**

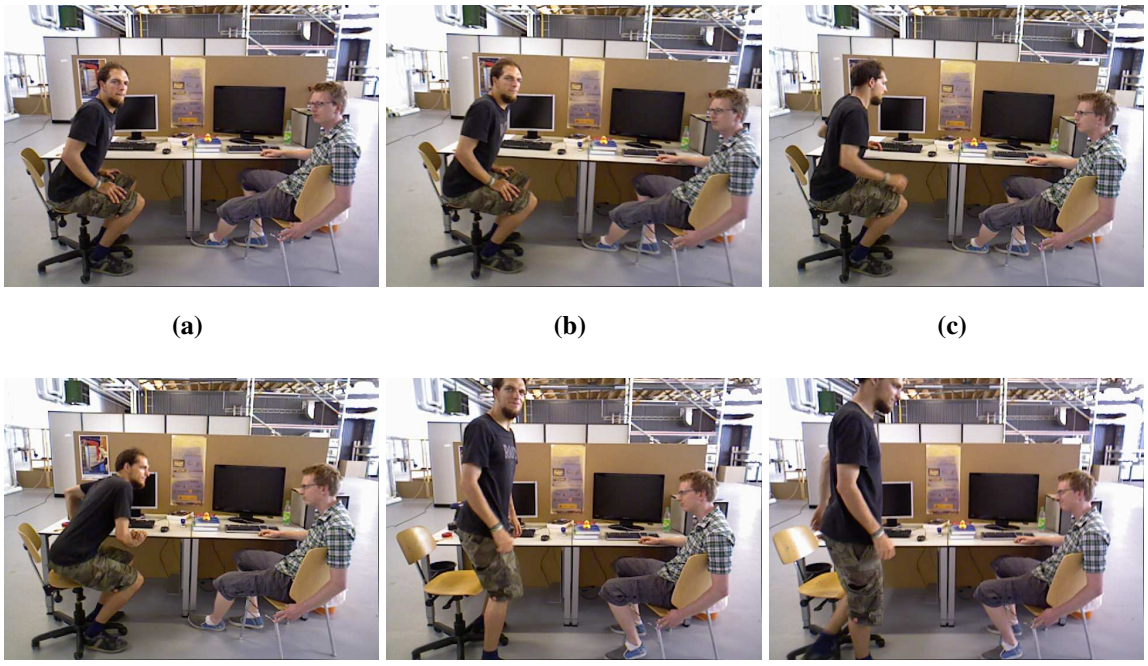
Call $j$ the last index of the ordered data belonging to $S_{i-1} \cap S_i$.

Order $S_i \backslash S_{i-1}$ based on their local order starting with global index $j+1$.

**end if**

**Step 6.** Update $c_{i+1} = \mu_i + k\varepsilon u_1$ and $i = i+1$.

**end while**



**(a)** **(b)** **(c)**

**(d)** **(e)** **(f)**

**Figure 6.2:** Frames from the walking_static dataset.

'walking_static' from a dataset made available by the Computer Vision Group of the Technical University of Munich (TUM) at https://cvg.cit.tum.de/data/datasets/rgbd-dataset/download#freiburg3_sitting_static (a few frames are given in Figure 6.2) visualized in three dimensions in Figure 6.3. This trajectory is characterized by instances of high acceleration or sharp kinks. Consequently, we cannot expect that the data within a Euclidean distance of $\varepsilon$ belong to time steps that are also close to each other. We have to come up with a way of defining local neighborhoods of data that consider more than just Euclidean distance. For this, we turn to the calculation of curvature from local SVD calculations from [97] which provides that

$$\kappa_{i-2}(t) = a_{i-2}(t)\frac{\sigma_{i-1}(t)}{\sigma_0(t)\sigma_{i-2}(t)}, \quad a_{i-2}(t) = \frac{i}{i+(-1)^i}\cdot\sqrt{\frac{4i^2-1}{3}} \text{for } i = 2,\ldots,m \tag{6.1}$$

where $m$ is the ambient dimension of the data. A projection of the trefoil knot to $\mathbb{R}^2$ leads to a a self-intersecting, intrinsically one-dimensional trajectory that we consider to experiment with

curvature's effectiveness in identifying natural collections of data points. Notice in Figures 6.4 and 6.5 that the curvature crashes when even a single data point is included in the local neighborhood that does not belong to the same region of the trajectory as the others. Therefore, the ball used to define a neighborhood of data can be grown until the curvature of its contained points crashes. Better yet, an ellipse whose principal axes are defined by the singular values of the neighborhood of data can be expanded until this behavior is exhibited by the curvature. This modification is given in Algorithm 10. Such an adaptive scheme was able to order video frames from portions of videos from the TUM dataset whose low-dimensional representation were non-overlapping. One can imagine that it is the artificial nature of the videos in this dataset that makes their low-dimensional representations overlap. The background in the video is kept constant while two people are in motion. As soon as they step out of the frame and then back into the frame, a self-intersection is created. Meanwhile, I would imagine that video from an outdoor surveillance camera would not have such self-intersections. I would expect the lighting and background objects to be changing with relative frequency. The proprietary nature of such security footage proved to be a roadblock to pursuing that avenue further. Another potential application of interest is one of kinesiology. Repetitive motion, such as running or the swing of a bat, should also be able to be represented as a trajectory whose ambient dimension is the same as the number of motion sensors on the subjects. The trajectory taken by professionals in the sport should somehow be optimized for performance while those of amateurs would be less so. One could compare the order of the data from an amateur, the curvatures computed during the execution of the ordering algorithm, and the number of data points belonging to each neighborhood to that of a professional or the average of several professionals for improved performance. Professional athlete movement data also proved to be unavailable for such analysis.

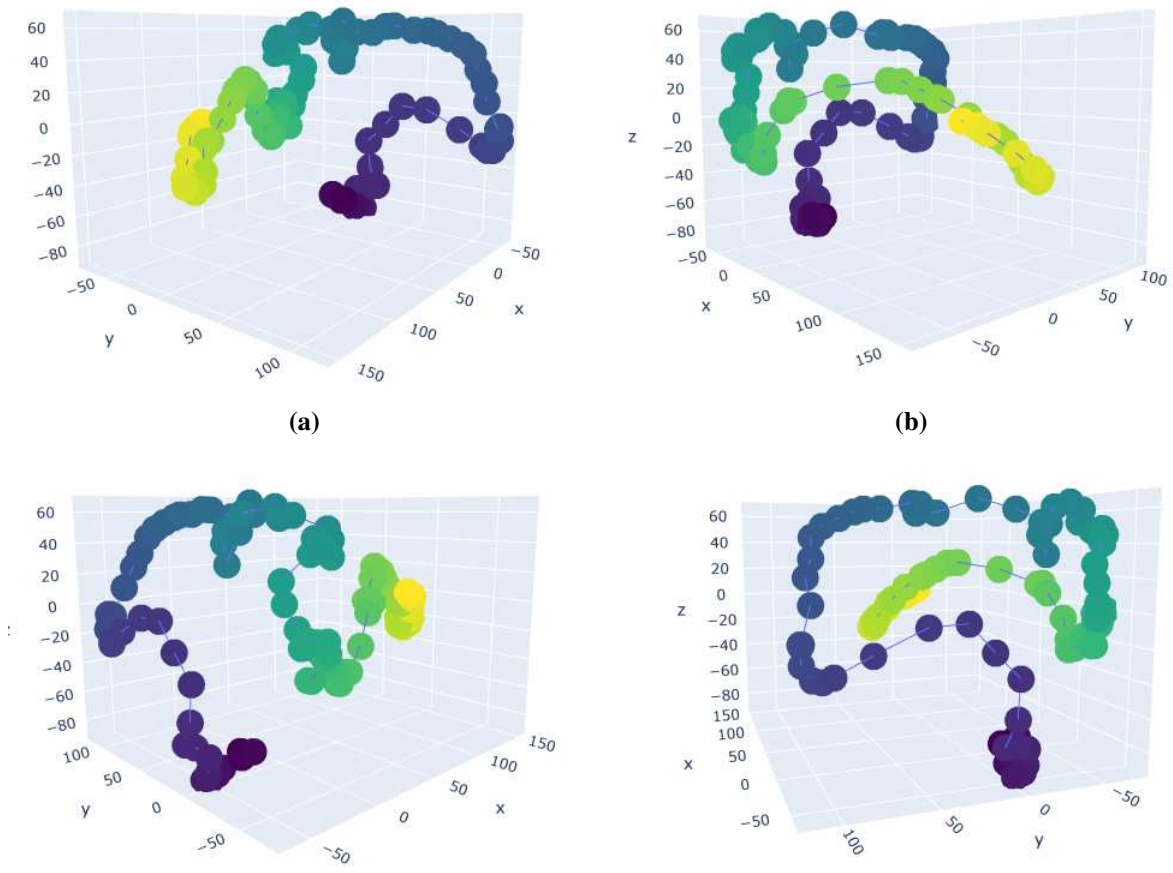
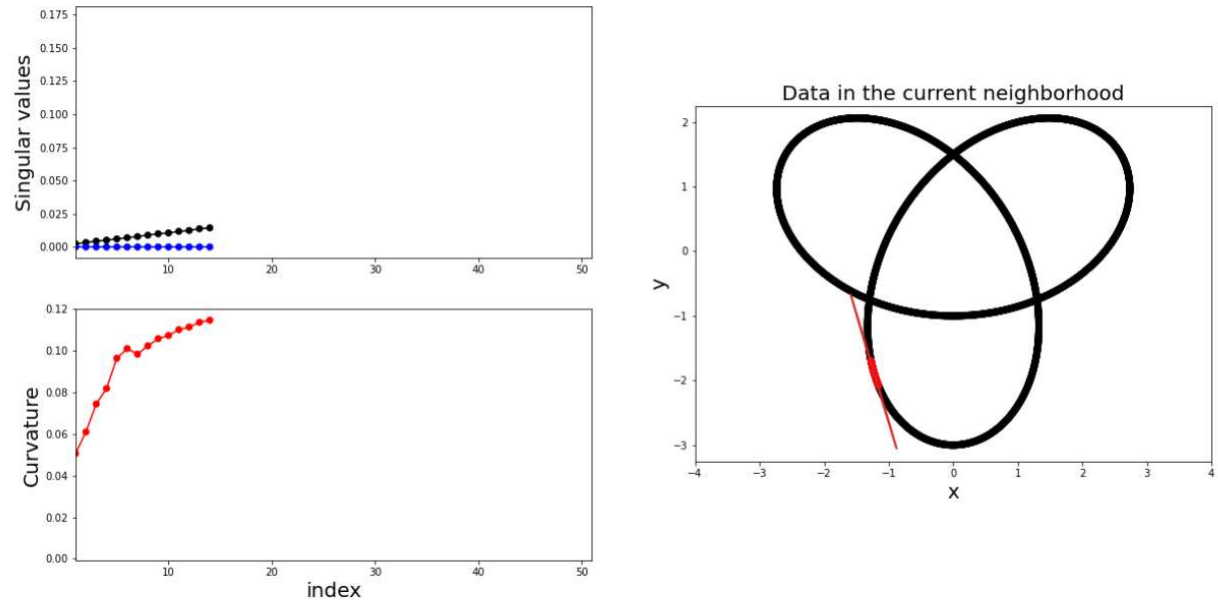
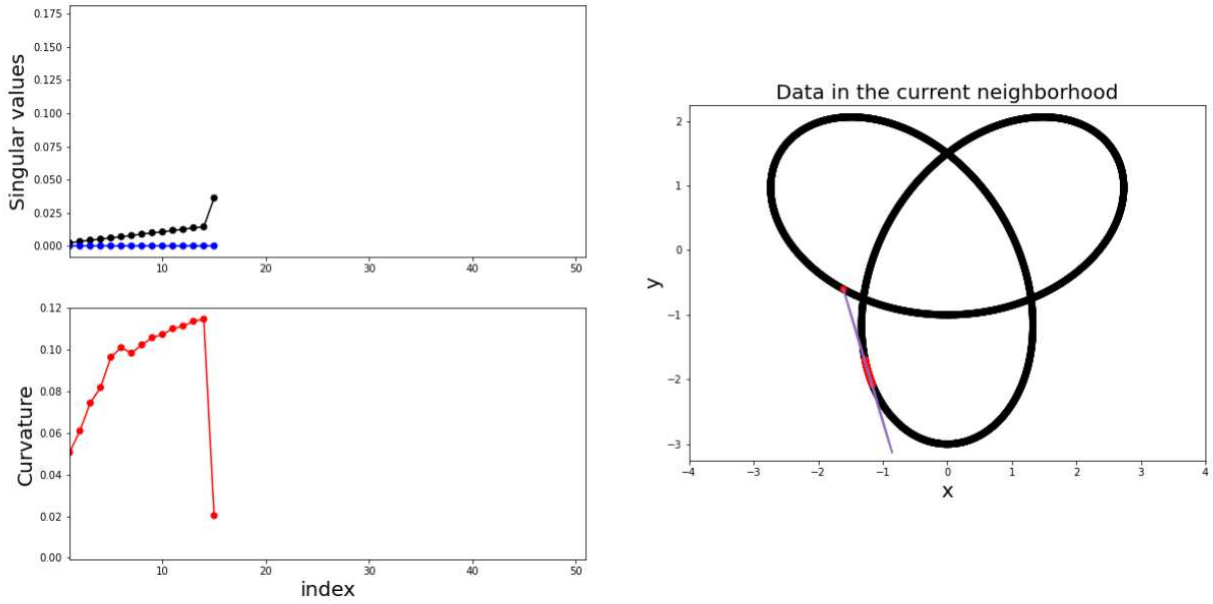



**Figure 6.3:** The first 100 frames from the walking_static dataset visualized in 3D using the first three left singular vectors, colored in chronological order.

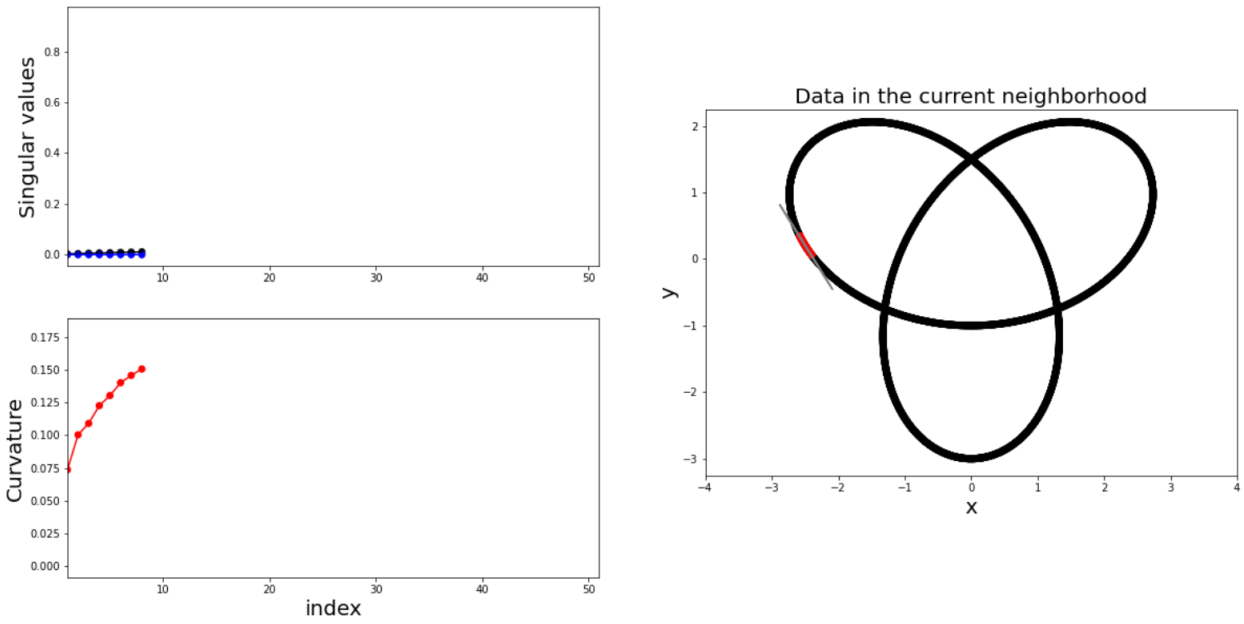



**(a)**

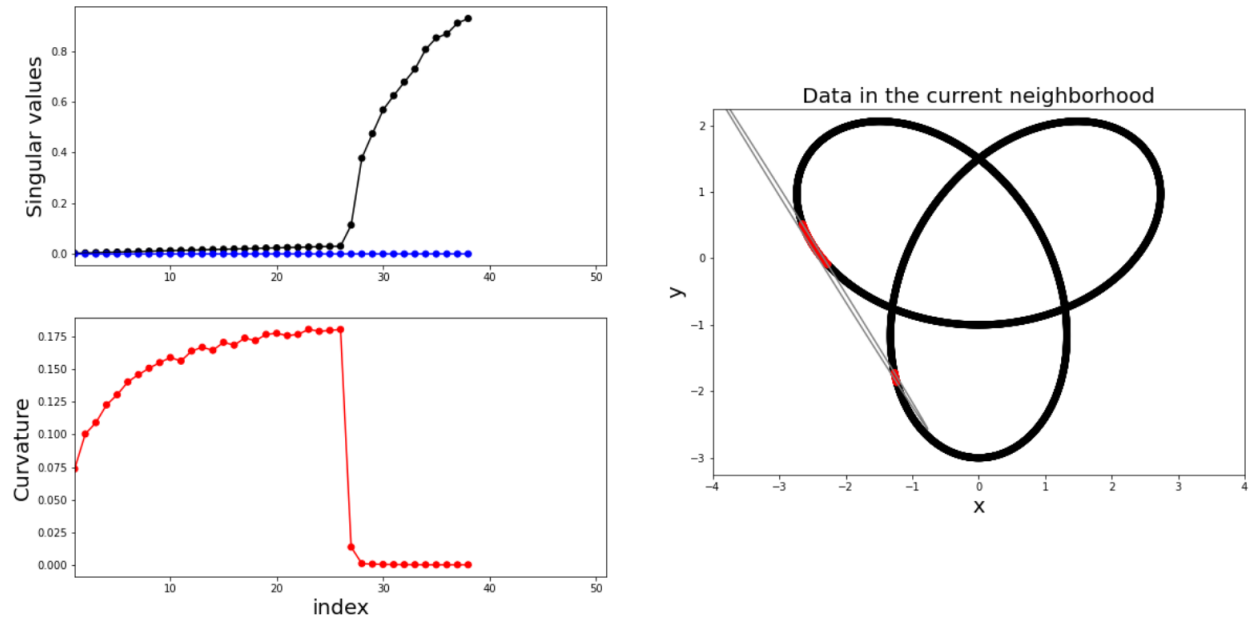



**(b)**

**Figure 6.4:** The singular values and curvature given a selection of data from the trefoil knot in two dimensions that (a) belong to the same local region of the intrinsically one-dimensional trajectory and that (b) do not.




**(a)**




**(b)**

**Figure 6.5:** The singular values and curvature given a selection of data from the trefoil knot in two dimensions that (a) belong to the same local region of the intrinsically one-dimensional trajectory and that (b) do not.

**Algorithm 10** Data Seriation with Curvature

Given a random initial point $c_i$, $i = 0$; a choice of $\varepsilon$, projection constant $k$, growth parameter $g > 1$, and tolerance value $tol$ for the curvature; and a dataset horizontally stacked into a matrix $\mathbf{X} \in \mathbb{R}^{m \times d}$, initialize a dictionary $Y$, where the global ordering of the data will be saved, with $d$ keys $Y_k$ that are equal to the $m$-dimensional points in $X$. Let $Y_v$ denote the values of $Y$.

**while** $|Y_v| < d$ **do**

**Step 1a.** Let $S_i = \{x \in \mathbf{X} : \|x - c_i\|_2 < \varepsilon\}$.

**Step 1b.** Compute the curvature of $S_i$.

**Step 1c.** Compute the first two principal components of $S_i$ and define an ellipse from them that contains all the points in $S_i$.

**Step 1d.** Iteratively grow the ellipse by a factor of $g$ until the curvature of the points (calculated using (6.1)) contained in the ellipse are no longer within $tol$ of the curvature from Step 1b.

**Step 1e.** Call $S_i^*$ the data points from the last iteration of Step 1d that were within $tol$ of the curvature from Step 1b. Let $M$ be the matrix with the column vectors of $S_i^*$ horizontally stacked together.

**Steps 2–6.** Perform Steps 2–6 from Algorithm 9.

**end while**

## 6.3 Model Generalizability and Trainability

Explaining a network's performance on data not seen during the training process, or its *generalizability*, by exploring its parameter space is still an open area of research. The literature in this area has taken on a topological flavor by reinforcing the concept that local regions of a network's parameter space at flat minima of the cost function tend to generalize better [9, 98, 99]. At around the time of these publications, [100] expressed reservations about the accuracy of this claimed correlation and urged the rethinking of the meaning of 'flatness'. The notion of flatness in previous work varies: [98] and [99] consider local curvature information encoded in the eigenval-

ues of the Hessian of the loss function, [100] use the spectral norm and trace of the Hessian, the authors of [98] also introduce their own metric to quantify the 'sharpness' of a minimizer (which is computationally less demanding to calculate than the eigenvalues of the Hessian). Loss landscapes introduced in [9], work that we have mentioned in previous chapters, are used in this section. We traverse the networks' loss landscapes in a way prescribed by the learning scheme, step size, and stop criteria under different initialization in Section 6.3.1. Toward understanding the topology of the loss landscape in the vicinity of the trained models at the end of these trajectories, we define level sets of parameter space in Section 6.3.2.

### 6.3.1 Training Trajectories

To start, we trained two small ReLU FFNNs from scratch:

$$F_1 : \mathbb{R}^2 \to \mathbb{R}^3 \to \mathbb{R}^3 \to \mathbb{R} \tag{6.2}$$

and

$$F_2 : \mathbb{R}^2 \to \mathbb{R}^3 \to \mathbb{R}^7 \to \mathbb{R}^{50} \to \mathbb{R}^{100} \to \mathbb{R}^{100} \to \mathbb{R}^{100} \to \mathbb{R}^{50} \to \mathbb{R}^7 \to \mathbb{R}^3 \to \mathbb{R}. \tag{6.3}$$

Specifically, both were trained to approximate $f(x,y) = x^2 + 2y^2 + x + 1$ on the unit square. The Adam optimizer and mean-squared error loss function were used. The model was trained under 10 different random initializations (using `seed` from the `numpy.random` package for reproducibility), 10 times for each initialization, until the loss value failed to decrease for 20 consecutive epochs. The number of epochs required for each initialization is summarized in Table 6.1. Because we kept the optimizer, learning rate, and stop criteria identical for all runs under each initialization, all of their model parameter values from each epoch of training were identical. Thus, Table 6.1 does not reflect the number of training epochs required for the convergence for one particular run, but for arbitrarily many runs, keeping the training parameters constant. In this way, model initialization immediately determines the model returned at the end of training for a particular set of learning

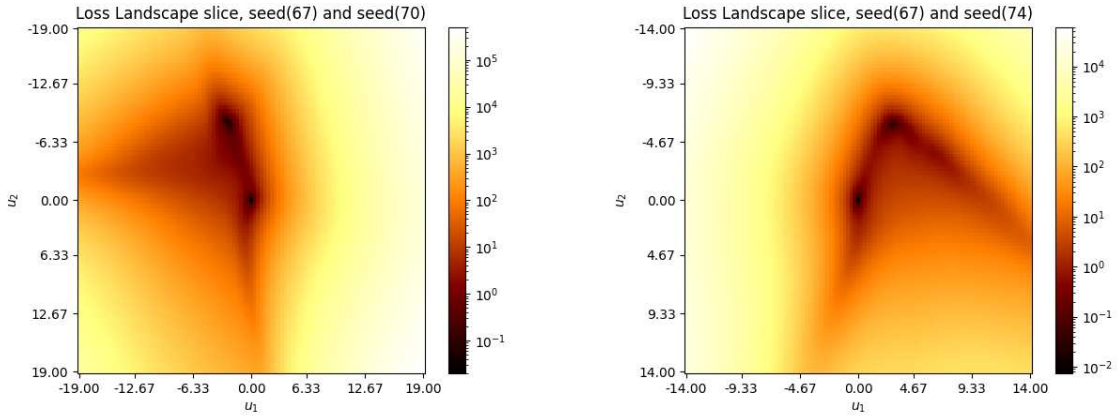
parameters. This is because of the fixed topology of the neural network's loss landscape. Two resulting landscapes are given in Figure6.6. Training can be thought of as a traversal method of the network's parameter space. Assuming an appropriate basis is chosen for the loss landscape, this walk in the model's weight space can be projected onto its surface. Take, for example, Figure 6.8 which shows the loss landscape of Model 6.2 with the parameter values at each epoch of training superimposed as a scatter plot from two different random initializations. The visualized training trajectories corresponding to the maximum and minimum number of epochs seen in training Model 6.2 (as reflected in Table 6.1) and the basis vectors $u_1$ and $u_2$ are given by the SVD of the model parameters from all epochs of training.

**Table 6.1:** Number of epochs of training required until convergence under different random seeds.

| `numpy.random seed` | **Model 6.2** | **Model 6.3** |
|---|---|---|
| **67** | 739 | 148 |
| **68** | 267 | 74 |
| **69** | 68 | 156 |
| **70** | 201 | 101 |
| **71** | 90 | 182 |
| **72** | 104 | 170 |
| **73** | 169 | 41 |
| **74** | 184 | 112 |
| **75** | 173 | 129 |
| **76** | 263 | 41 |

Instead of viewing training as the progression of the entire parameter set, we could split up the parameters and track each individually over the course of training. To illustrate, Model 6.2 has the parameters $W^{(i)}, b^{(i)}, i = 1,2,3$. Consider the first of these, $W^{(1)}$, which can be projected down to 3 dimensions and visualized as shown in Figure 6.7. We use local SVD to compute the curvatures of this trajectory, using equations from [97] given in (6.1). We performed these curvature computations for all of Model 6.2's parameters and the results are shown in Figure 6.9. We are interested in considering curvature as a metric to (1) quantify the non-convexity of a neural network's loss landscape (2) define adaptive step sizes in training to minimize its computation and

time requirements and (3) predict whether or not the network attained at the end of training will have certain desirable features.




**(a)** **(b)**

**Figure 6.6:** Slices of the Model 6.2's loss landscape through two trained models after using the initialization defined by (a) seed 67 and (b) seed 69.

### 6.3.2 Level Sets of the Parameter Space

We unexpectedly found inspiration in the brief mention of parameter space work in [56]. More specifically, they choose a neural network of interest, $F$, and consider a particular epsilon-ball of parameter space about $F$ with the interest of determining the robustness of the number of distinct regions in $F$'s polyhedral mesh. Taking that idea further, if all training points lie within a polyhedron of the mesh (not on a bounding edge of a polyhedron for example) then surely there is a small enough perturbation to the model parameters that preserves the binary vector labels of all of the polyhedra, preserving the bits of the training data inputs that are required to map within a certain threshold of their training labels. Then there must also be a small enough perturbation in the weights and biases that does, in fact, keep the output of the network close enough to the ground truth labels. A series of these small perturbations would produce a local collection of neural networks in the parameter space, but are there disconnected portions of the parameter space that produce networks that perform just as well? The non-uniqueness of the solution to the optimiza-

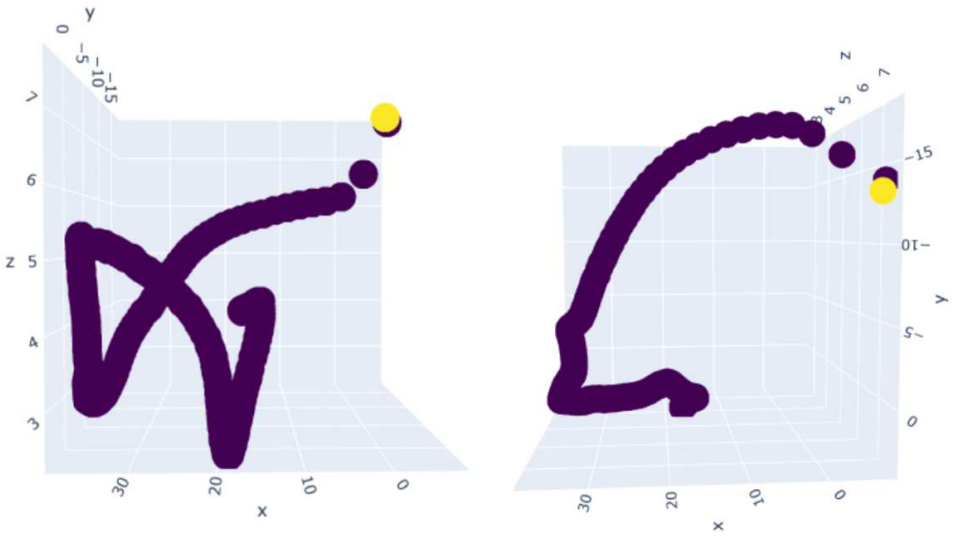


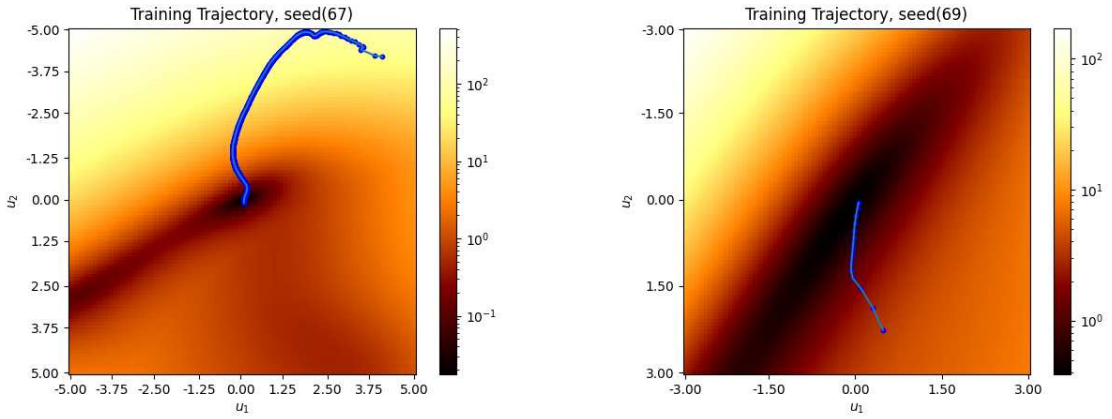

**Figure 6.7:** Model 6.2's first weight matrix, $W^{(1)} \in \mathbb{R}^{3\times 2}$, from 739 epochs of training reduced to 3 dimensions. The yellow point is from the first epoch, the ordering is natural thereafter.

tion problem iteratively solved during the training process of a neural network, the non-uniqueness made apparent in Figure 6.6, encouraged us to formalize this optimization problem and study the structure of its solution.

We start by considering models that train until perfect performance on $n$ training points in $\mathbb{R}^m$. It has been proven that the loss landscape of a two-layered ReLU FFNN has at least one global minima of $z = 0$ [101], results that extend to two-layered FFNNs that use an activation function can be characterized as a max-affine spline operator (MASO) [57]. However, as perfect performance on training data is not ideal for model generalizability, we will loosen this criterion to accuracy above a certain threshold and add a constraint based on performance on a validation data set. Based on the limited relevant literature [9], we hypothesized that the structure of the collection of models of interest varies based on network architecture. The formal definition of these level sets and the exploration of their structure might provide mathematical grounding to argue that certain architectures are naturally more generalizable and trainable than others.

Take the two-layered neural network

$$F : \mathbb{R} \xrightarrow[\text{ReLU}]{W^{(1)},\, b^{(1)}} \mathbb{R}^2 \xrightarrow{W^{(2)},\, b^{(2)}} \mathbb{R} \tag{6.4}$$




**(a)** Loss landscape of Model 6.2 centered about the trained model given the random variable initialization defined by seed 67.

**(b)** Loss landscape of Model 6.2 centered about the trained model given the random variable initialization defined by seed 69.

**Figure 6.8:** Loss landscapes of Model 6.2 with respect to different orthonormal vectors.

which maps $x^{(q,0)}$ to $x^{(q,2)} = W^{(2)}\text{diag}(v)W^{(1)}x^{(q,0)} + W^{(2)}\text{diag}(v)b^{(1)} + b^{(2)},\ \forall q \in \mathbb{N}$, where $v$ is the binary vector associated with $x^{(q,0)}$ and

$$W^{(1)} = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix},\ W^{(2)} = \begin{bmatrix} W_3 & W_4 \end{bmatrix},\ b^{(1)} = \begin{bmatrix} b_5 \\ b_6 \end{bmatrix},\ b^{(2)} = \begin{bmatrix} b_7 \end{bmatrix},\ \text{and } v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}.$$

Let $\theta$ denote all of the tunable parameters of $F$. Without loss of generality, take the training data set of size $t$ to be

$$\mathscr{X} = \{x^{(i,0)} \ :\ x^{(i,0)} \in \mathbb{R},\ i = 1,\ldots,t\}$$

and associated labels

$$\mathscr{L} = \{\ell^{(i)} \ :\ \ell^{(i)} \in \mathbb{R},\ i = 1,\ldots,t\}.$$

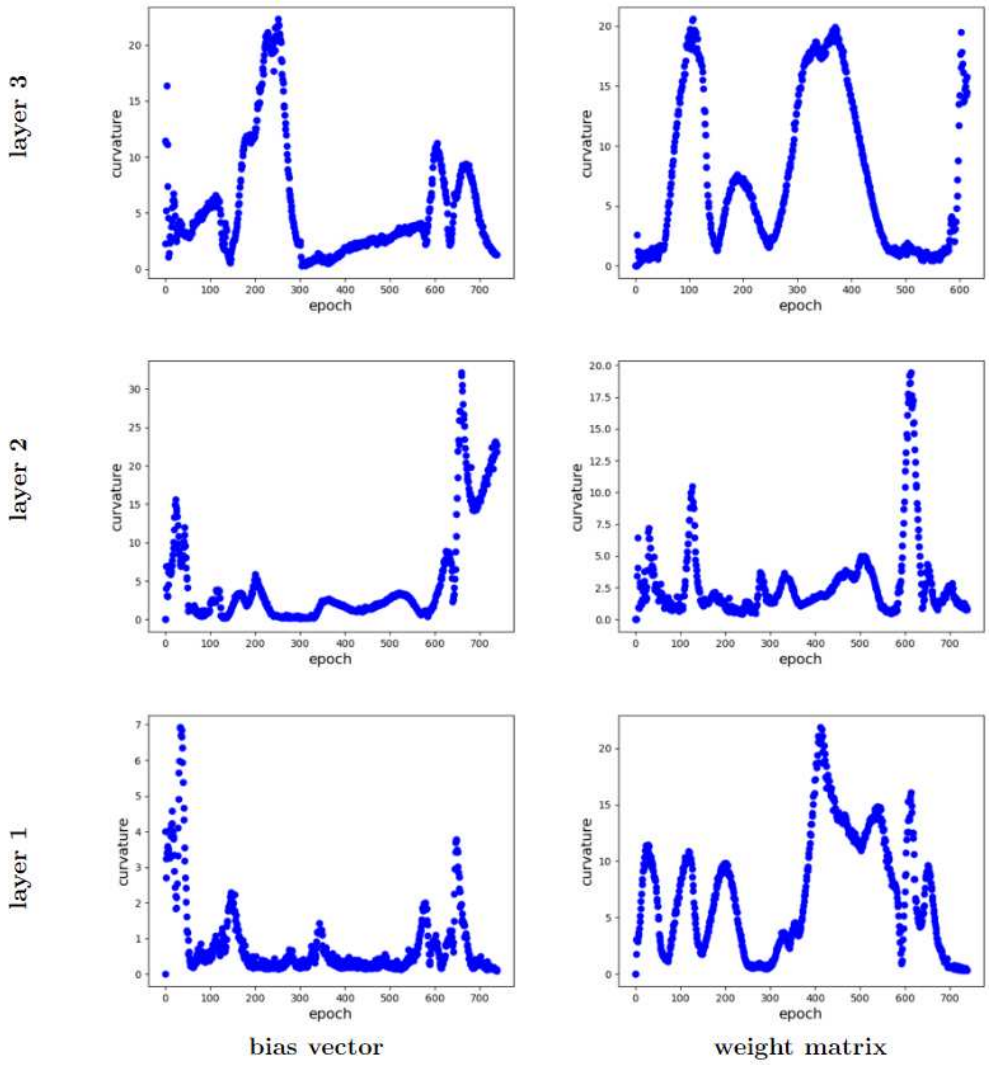



**Figure 6.9:** Curvature of all six of Model 6.2's parameters for all epochs of training after random initialization using seed 67.

Define the following cost function that quantifies the performance of $F$ on the training set

$$\begin{aligned} C(\theta) &= \sum_{i=0}^{t} \left\| x^{(i,2)} - \ell^{(i)} \right\|^2 \\ &= \sum_{i=0}^{t} \left\| F(x^{(i,0)}; \theta) - \ell^{(i)} \right\|^2 \\ &= [F(\mathscr{X}) - \mathscr{L}]^\top [F(\mathscr{X}) - \mathscr{L}]. \end{aligned} \tag{6.5}$$

**Theorem 6.3.1.** *(First-Order Necessary Condition) If $x^*$ is a local minimizer of $C$ and $C$ is continuously differentiable in an open neighborhood of $x^*$, then $\nabla C(x^*) = 0$ [102].*

So we require that $\nabla C = 0$ with

$$\nabla C = \left[ \frac{\partial C}{\partial F}\frac{\partial F}{\partial W_1} \quad \frac{\partial C}{\partial F}\frac{\partial F}{\partial W_2} \quad \frac{\partial C}{\partial F}\frac{\partial F}{\partial W_3} \quad \frac{\partial C}{\partial F}\frac{\partial F}{\partial W_4} \quad \frac{\partial C}{\partial F}\frac{\partial F}{\partial b_5} \quad \frac{\partial C}{\partial F}\frac{\partial F}{\partial b_6} \quad \frac{\partial C}{\partial F}\frac{\partial F}{\partial b_7} \right] \tag{6.6}$$

where $\frac{\partial C}{\partial F} = 2[F(\mathscr{X}) - \mathscr{L}]$, $\frac{\partial F}{\partial W_1} = W_3 v_1 \mathscr{X}$, $\frac{\partial F}{\partial W_2} = W_4 v_2 \mathscr{X}$, $\frac{\partial F}{\partial W_3} = v_1 W_1 \mathscr{X} + v_1 b_5 e_t$, $\frac{\partial F}{\partial W_4} = v_2 W_2 \mathscr{X} + v_2 b_6 e_t$, $\frac{\partial F}{\partial b_5} = W_3 v_1$, $\frac{\partial F}{\partial b_6} = W_4 v_2$, $\frac{\partial F}{\partial b_7} = 1$, and $e_i = \mathbf{1} \in \mathbb{R}^i$.

**Definition 6.3.1.** *A square matrix $A \in \mathbb{R}^{n \times n}$ is positive definite if there is a positive scalar $\alpha$ such that $x^\top Ax \geq \alpha x^\top x$, $\forall x \in \mathbb{R}^n$. It is positive semidefinite if $x^\top Ax \geq 0$, $\forall x \in \mathbb{R}^n$ [102].*

**Theorem 6.3.2.** *(Second-Order Necessary Condition) If $x^*$ is a local minimizer of $C$ and $H(C) \coloneqq \nabla^2 C$ ($H$ for 'Hessian') exists and is continuous in an open neighborhood of $x^*$, then $\nabla C(x^*) = 0$ and $\nabla^2 C(x^*)$ is positive semidefinite [102].*

So, by definition of positive semidefinite, we require that $pH(C)p^\top \geq 0 \; \forall p$ where

$$H(C) = \begin{bmatrix} \frac{\partial^2 C}{\partial W^{(1)2}} & \frac{\partial^2 C}{\partial W^{(1)} \partial W^{(2)\top}} & \frac{\partial^2 C}{\partial W^{(1)} \partial b^{(1)}} & \frac{\partial^2 C}{\partial W^{(1)} \partial b^{(2)}} \\ \frac{\partial^2 C}{\partial W^{(2)\top} \partial W^{(1)}} & \frac{\partial^2 C}{\partial (W^{(2)\top})^2} & \frac{\partial^2 C}{\partial W^{(2)\top} \partial b^{(1)}} & \frac{\partial^2 C}{\partial W^{(2)\top} \partial b^{(2)}} \\ \frac{\partial^2 C}{\partial b^{(1)} \partial W^{(1)}} & \frac{\partial^2 C}{\partial b^{(1)} \partial W^{(2)\top}} & \frac{\partial^2 C}{\partial b^{(1)2}} & \frac{\partial^2 C}{\partial b^{(1)} \partial b^{(2)}} \\ \frac{\partial^2 C}{\partial b^{(2)} \partial W^{(1)}} & \frac{\partial^2 C}{\partial b^{(2)} \partial W^{(2)}} & \frac{\partial^2 C}{\partial b^{(2)} \partial b^{(1)}} & \frac{\partial^2 C}{\partial b^{(2)2}} \end{bmatrix}$$

$$= \left[\begin{array}{cc|cc|ccc} \frac{\partial^2 C}{\partial W_1^2} & \frac{\partial^2 C}{\partial W_1 \partial W_2} & \frac{\partial^2 C}{\partial W_1 \partial W_3} & \frac{\partial^2 C}{\partial W_1 \partial W_4} & \frac{\partial^2 C}{\partial W_1 \partial b_5} & \frac{\partial^2 C}{\partial W_1 \partial b_6} & \frac{\partial^2 C}{\partial W_1 \partial b_7} \\ \frac{\partial^2 C}{\partial W_2 \partial W_1} & \frac{\partial^2 C}{\partial W_2^2} & \frac{\partial^2 C}{\partial W_2 \partial W_3} & \frac{\partial^2 C}{\partial W_2 \partial W_4} & \frac{\partial^2 C}{\partial W_2 \partial b_5} & \frac{\partial^2 C}{\partial W_2 \partial b_6} & \frac{\partial^2 C}{\partial W_2 \partial b_7} \\ \hline \frac{\partial^2 C}{\partial W_3 \partial W_1} & \frac{\partial^2 C}{\partial W_3 \partial W_2} & \frac{\partial^2 C}{\partial W_3^2} & \frac{\partial^2 C}{\partial W_3 \partial W_4} & \frac{\partial^2 C}{\partial W_3 \partial b_5} & \frac{\partial^2 C}{\partial W_3 \partial b_6} & \frac{\partial^2 C}{\partial W_3 \partial b_7} \\ \frac{\partial^2 C}{\partial W_4 \partial W_1} & \frac{\partial^2 C}{\partial W_4 \partial W_2} & \frac{\partial^2 C}{\partial W_4 \partial W_3} & \frac{\partial^2 C}{\partial W_4^2} & \frac{\partial^2 C}{\partial W_4 \partial b_5} & \frac{\partial^2 C}{\partial W_4 \partial b_6} & \frac{\partial^2 C}{\partial W_4 \partial b_7} \\ \hline \frac{\partial^2 C}{\partial b_5 \partial W_1} & \frac{\partial^2 C}{\partial b_5 \partial W_2} & \frac{\partial^2 C}{\partial b_5 \partial W_3} & \frac{\partial^2 C}{\partial b_5 \partial W_4} & \frac{\partial^2 C}{\partial b_5^2} & \frac{\partial^2 C}{\partial b_5 \partial b_6} & \frac{\partial^2 C}{\partial b_5 \partial b_7} \\ \frac{\partial^2 C}{\partial b_6 \partial W_1} & \frac{\partial^2 C}{\partial b_6 \partial W_2} & \frac{\partial^2 C}{\partial b_6 \partial W_3} & \frac{\partial^2 C}{\partial b_6 \partial W_4} & \frac{\partial^2 C}{\partial b_6 \partial b_5} & \frac{\partial^2 C}{\partial b_6^2} & \frac{\partial^2 C}{\partial b_6 \partial b_7} \\ \hline \frac{\partial^2 C}{\partial b_7 \partial W_1} & \frac{\partial^2 C}{\partial b_7 \partial W_2} & \frac{\partial^2 C}{\partial b_7 \partial W_3} & \frac{\partial^2 C}{\partial b_7 \partial W_4} & \frac{\partial^2 C}{\partial b_7 \partial b_5} & \frac{\partial^2 C}{\partial b_7 \partial b_6} & \frac{\partial^2 C}{\partial b_7^2} \end{array}\right].$$

Now that we have this optimization problem formally defined, exploring the geometry of the solution set(s) would be interesting. It might also be worthwhile to see how these solution sets change with added constraints on the magnitude of the eigenvalues of $H(C)$ as *flat minimizers* are desirable and are characterized by small eigenvalues [98].

### 6.3.3 Architecture Considerations

The literature highlights two architectural features of neural networks that affect their trainability, generalizability, and overall performance: skip connections and model depth. We give a brief overview of each of these features and show the effects of these architectural differences.

#### Skip connections

Skip connections were introduced in light of the difficulties in training deep neural architectures (difficulties covered in more detail later). Their effects are hardly noticeable with shallow networks (e.g. ResNet-20 versus ResNet-20-noshort) [9] but their importance grows with network depth [103]. For deep neural networks (e.g. ResNet-56 and ResNet-56-noshort), [9] shows that the use of shortcut connections increases the stability of the neural network with respect to perturbations in its parameter space and makes this association between performance and model configurations increasingly convex. That is to say, shortcut connections make the neural network's loss landscape more smooth and more convex. This increases the chance that the model returned at the end of training will be the global minima of the loss function or at least in its vicinity. Conversely, the less stable the relationship between performance and model parameters, the more likely training will get stuck in local minima, returning a model that does not achieve accurate performance.

In this section, we chime in on the ever-growing topological conversation about neural networks. Rather than considering many of the suggestions in the literature toward better-performing networks that require modifications to different aspects of the training process [104–106], we focus on the topological features already present in readily available pretrained neural networks. We look at the input-output Jacobian norms of two neural networks, ResNet-56 and ResNet-56-noshort, to observe the difference that skip connections make in some notion of 'smoothness' of the neural

network. We fix trajectories along the input space of these networks, feed samples along them through both networks, save each of their input-output Jacobians, and compute pairwise cosine similarities among them. Since both of the networks of interest were trained on CIFAR-10, their input spaces are $3 \times 32 \times 32$ images. The two trajectories that we consider through the neural network's input space are:

- Trajectory 1 (Figure 6.10) is a sampling of 998 synthetic images along the line between two distinct frog images from the CIFAR-10 dataset, including these two images as endpoints for a total of 1000 images. Both images were correctly classified by both models.
- Similarly, Trajectory 2 (Figure 6.11) is a sampling of 998 synthetic images along the line between two CIFAR-10 images, but these two images belong to two different classes: frog and truck. Again, our sampling is a total of 1000 images and both endpoint images were correctly classified by both models.

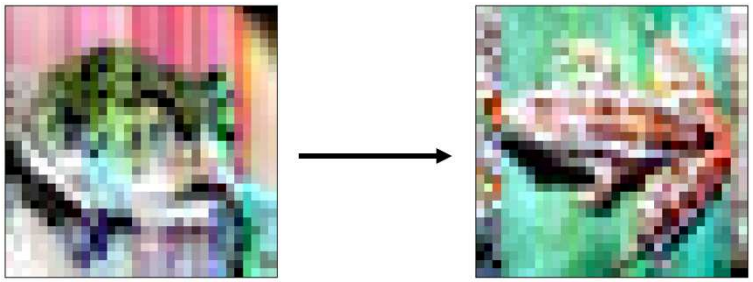


**Figure 6.10:** Trajectory 1

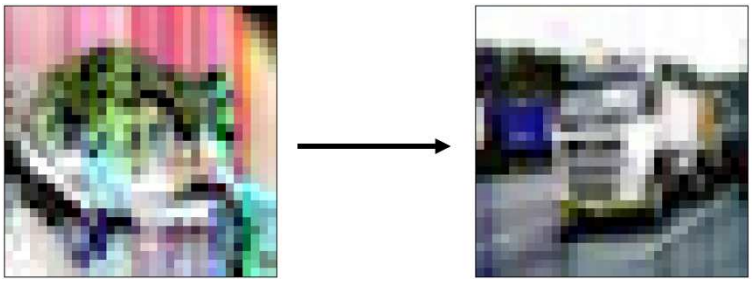


**Figure 6.11:** Trajectory 2

As stated in Section 5.3.1, it seems that the only work in the literature using the input-output Jacobian of a neural network computes its norm along an intrinsically one-dimensional path through its input space. Instead of condensing this high-dimensional information ($(32 \times 32 \times 3 \times 10)$-dimensional for ResNet-56 and ResNet-56-noshort) down to a single value, we preserve the topological richness of the data. The pairwise cosine similarities between all input-output Jacobians with respect to ResNet-56 along both geodesics are given in Figure 6.12. We note that the diagonal entries fade smoothly into the off-diagonals, implying that consecutive and nearby samples have similar Jacobians, meaning that the network is relatively smooth along these trajectories. On the other hand, Figure 6.14 is characterized by high contrast between diagonal and off-diagonal entries, implying a more chaotic topology along these two geodesics. Due to the implicit association between binary vectors and Jacobians, we also provide pairwise Hamming distances in Figures 6.13 and 6.15. While preserving the input-output Jacobians' original dimensionality is insightful and interesting, this becomes quite a grueling task whose memory requirements push the memory capabilities of most machines. To produce an $k \times k$ cosine similarity plot for input-output Jacobians like those given in Figures 6.12 and 6.14, $k$ input-output Jacobians need to be saved, which are $(m \times n)$-dimensional matrices for a neural network $F : \mathbb{R}^m \to \mathbb{R}^n$, with either float or double precision entries, and then $k \times (k-1)/2$ cosine similarity computations are completed. In Section 4.3, we were surprised to find that condensing the input-output Jacobian down to its norm still revealed interesting patterns. The huge memory saving this allows while still providing enough information for interesting observations led us to pivot to focus on the input-output Jacobian norm rather than the input-output Jacobian itself.

**Model Depth**

Convolutional neural networks (CNNs) are among the deepest types of neural networks [103]. The remarkable improvements achieved by increasing model depth are precisely what inspired the term "deep learning" [103]. Improved performance from convolutional deep neural networks is a well-known and thoroughly exploited phenomenon; contest leaderboards provide a concise and compelling argument in their favor. The first of these results—which introduced AlexNet [107]

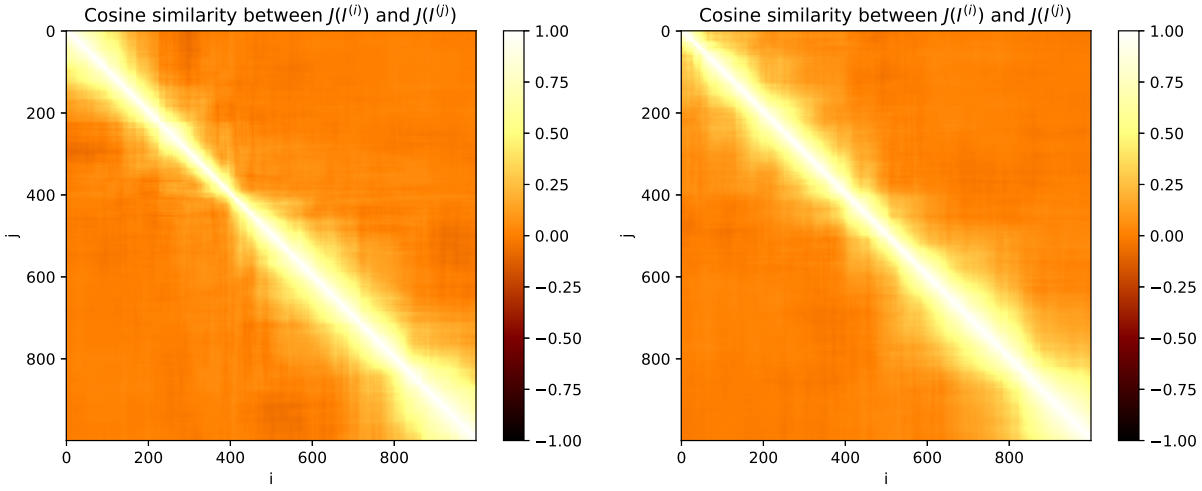



**Figure 6.12:** Pairwise cosine similarities of input-output Jacobians with respect to ResNet-56 of 1000 samples along (a) trajectory 1 and (b) trajectory 2.

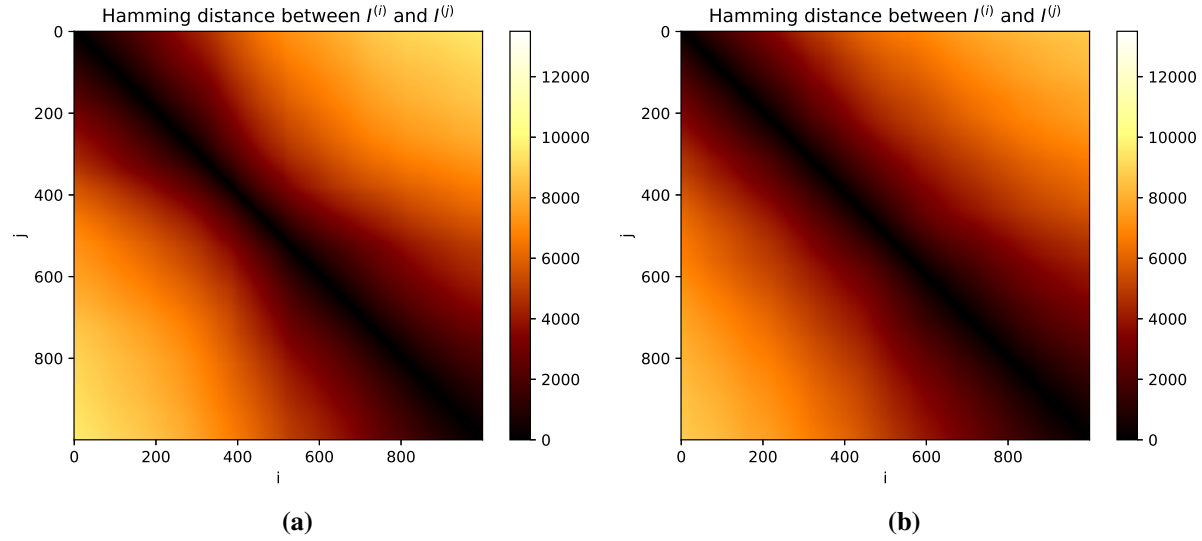
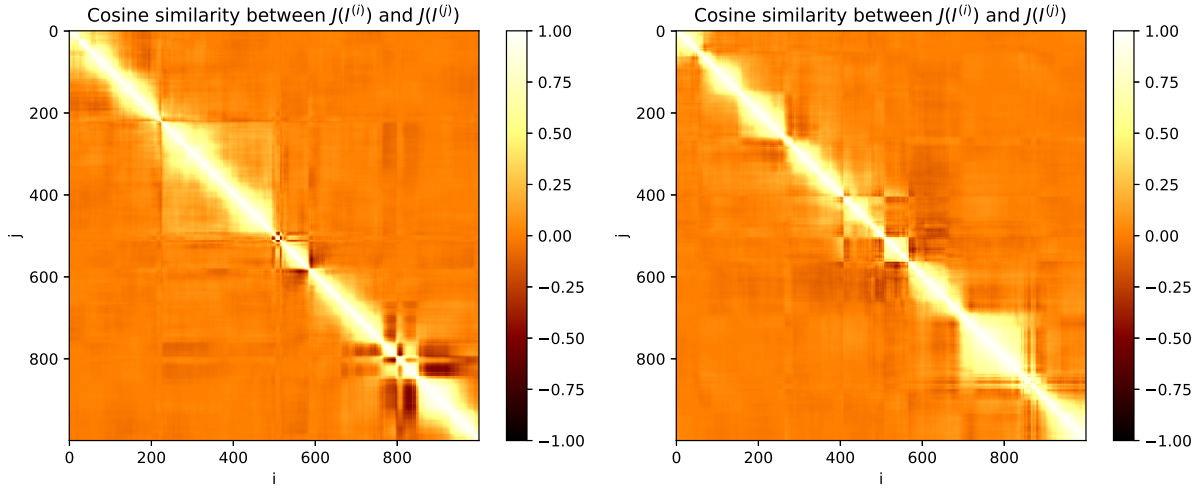
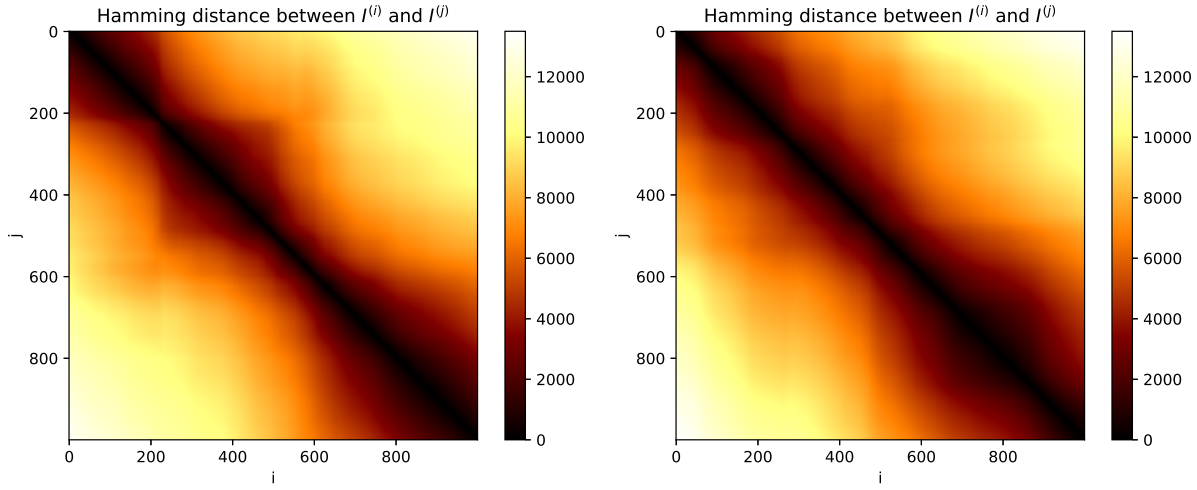



**Figure 6.13:** Pairwise Hamming distances with respect to ResNet-56 of 1000 samples along (a) trajectory 1 and (b) trajectory 2.




**(a)** **(b)**

**Figure 6.14:** Pairwise cosine similarities of input-output Jacobians with respect to ResNet-56-noshort of 1000 samples along (a) trajectory 1 and (b) trajectory 2.




**(a)** **(b)**

**Figure 6.15:** Pairwise Hamming distances with respect to ResNet-56-noshort for 1000 samples along (a) trajectory 1 and (b) trajectory 2.

and established its decisive victory in the ILSVRC-2012 challenge by being the first to combine model complexity, a huge dataset, and parallel processing power—was the catalyst for research on the topic, the bang beginning the race toward publishing subsequent findings. Some of the most notable work extended the material in [107] by proposing architectures including the VGG architecture [41], ResNet [44], Inception-v4 and Inception-ResNet-v2 [108], DenseNet [109], which are all still heavily used architectures. Others explore specific techniques: [110] proposed a region-based CNN for object detection and segmentation and [91] improves upon model training by avoiding overfitting. These are just a few among thousands of other publications that acknowledge the foundational contributions in [107] to the field of deep learning.

Although there is a vast literature base on the topic of deep learning, only a fraction of this work attempts to explain the reasons behind the improvement that comes with model depth [100]. One possible explanation is that increasing the number of nodes introduces implicit regularization [111]. Model regularization can be explicit or implicit; explicit methods include data augmentation, weight decay, and dropout, and implicit methods include training with stochastic gradient descent (SGD), early stopping, and batch normalization. A good introduction to these and an analysis of their effects on model generalization can be found in [101]. Regularization may improve model generalization but is neither necessary nor sufficient to control generalization error [101], leaving more to be desired from this attempt at an explanation. Another potential reason for deep models' note-worthy performance is that they may be able to generalize better by accurately approximating solutions of lower "complexity" [111]. The exact measure of "complexity" has yet to be determined [101] but does not appear to be among a few metrics from statistical learning theory including VC dimension [112], Rademacher complexity [113], or uniform stability [101, 114].

There are a few disadvantages to deep neural networks, all of which have a hand in the lack of research thoroughly exploring huge networks toward understanding how they really work. One of these is that deeper neural networks are more difficult to train [44]. The most intuitive explanation for this difficulty hinges on the difference in the topology of their loss surface as opposed to those of more shallow networks. An increase in network depth has been observed to result in a relatively

convex loss landscape morphing into a highly chaotic one [9]. Additionally, these larger networks often consist of hundreds of thousands if not millions of nodes and require a substantial amount of memory to store and computational power to use. For our methods of network analysis, we often require the ability to store at least two binning vectors at a time. For a network with ReLU layers, the least demanding of all activation functions in terms of memory, this is equivalent to $2h$ bits of memory where $h$ is the number of nodes in the ReLU layers of the network. The smallest network we consider is ResNet-18 which consists of 1,555,456 ReLU nodes. This high cost grows disproportionately to its benefits beyond a certain point—"each fraction of a percent improved accuracy costs nearly doubling the number of layers" when scaling models up to thousands of layers [115]. Rather than dampening the momentum in the machine learning community, these difficulties presented areas for improvement in the field. As was alluded to in Section 6.3.3, shortcut connections (or skip connections) are among these improvements.

Cautioned by these disadvantages of deep neural networks, but sufficiently motivated by their advantages, we proceed to explore them. In past chapters, we have examined polyhedral decompositions' imposed geometry on the data in their input spaces. Here, we probe neural networks using loss landscapes to see how much we can manipulate their polyhedral decomposition without it proving detrimental to classification performance. A neural network's *loss landscape* is conceptually similar to its input space landscapes introduced in Section 5.3. While a model's input space landscapes' domains are discretized slices of its input space, its loss landscapes' domains are discretized slices of its parameter space. In short, each point of a loss landscape's domain describes a particular configuration of a model and summarizes that neural network's performance. More formally, we first choose a network architecture of interest, without loss of generality say it has $N$ tunable parameters, and define an intrinsically two-dimensional plane $P$ through its parameter space by choosing two basis vectors $v_1$ and $v_2$ with ambient dimension $N$. The points $s_i$ on the loss landscape, an intrinsically three-dimensional surface $S$, satisfy $s_i = (a_i, b_i, c_i)$ where $p_i = (a_i, b_i) = a_i v_1 + b_i v_2 \in P$ where the model prescribed by $p_i$ achieves an average loss value of $c_i$ on the data of interest [9]. We expand the term 'loss' to refer to more than just the average

loss of training data with respect to the objective function used to train the model of interest. Here, we also use a custom loss 'function' which computes the Hamming distance in batches between training points belonging to the same class divided by their Euclidean distance. We train Model 1

$$\mathbb{R}^2 \to \mathbb{R}^{10} \to \mathbb{R}^{10} \to \mathbb{R}^{10} \to \mathbb{R}^{10} \to \mathbb{R}^1 \tag{6.7}$$

and Model 2

$$\mathbb{R}^2 \to \mathbb{R}^{1000} \to \mathbb{R}^{1000} \to \mathbb{R}^{1000} \to \mathbb{R}^{1000} \to \mathbb{R}^1 \tag{6.8}$$

on the interleaving half circles data introduced in Section 4.3 and give their loss landscapes in Figures 6.16 and 6.17, respectively. We define the parameter space of their loss landscapes to go through the initial and final models. By increasing model depth, the two loss functions (with respect to the training objective function and the custom loss function) become more compatible in the sense that a model can be found that reaches an equilibrium between the performance of both loss functions. This motivates us to revisit our bit classifier from Section 5.2.1. Given the size of ResNet-50 and the resulting flexibility of polyhedral resources, can we fine-tune it in such a way that increases the accuracy of our classifier? We believe that the answer is yes. Fine-tuning models such that the polyhedral decomposition has particular desirable properties (like the increased accuracy of bit vector-specific classifiers) seems to be an untouched subject in the field. [116] trains bits of particular layers of a neural network to be 'invariant' among images of the same class, or by encouraging the neural firing pattern corresponding to the layer of interest to be the same for these images. Their experiments provide striking evidence that these invariant neurons help their neural networks generalize to unseen data. This further suggests that fine-tuning ResNet-50 with respect to the special bits of our bit classifier would be a worthwhile endeavor. Namely, the generalization of our classifier that distinguishes between ImageNet images from DAmageNet ones; i.e. we might be able to better detect DAmage-ed images that were not used to build the classifier. We believe that fine-tuning neural networks based on the observations we have made in

previous chapters will become a more active area of research that will continue to improve model generalizability or make them less susceptible to adversarial attacks.

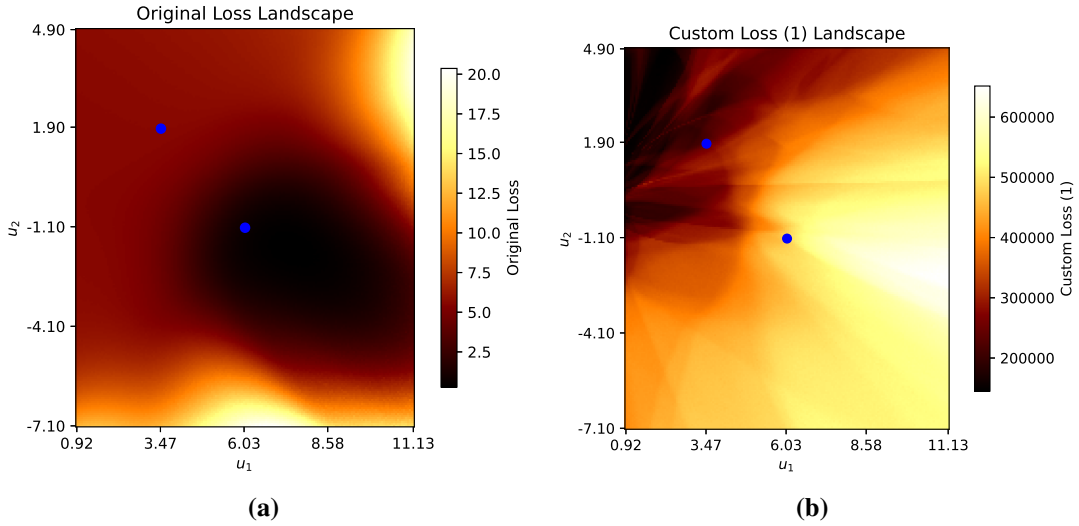



**Figure 6.16:** Loss landscapes of Model 1 (given in (6.7)) with respect to (a) classification loss and (b) the custom loss function with the initial and final models superimposed as blue points, centered about the final model.

## 6.4 Conclusion

In conclusion, Chapter 6 covered the topics of data seriation and neural network generalizability and trainability. We presented a novel data seriation algorithm that uses the curvature from local SVD that we have used to effectively order frames of non-intersecting video sequences. We are curious to experimentally verify the efficacy of our algorithm in different domains including kinesiology and surveillance, and identify unavailability or the proprietary nature of relevant data as an anticipated roadblock in that endeavor. Transitioning to the domain of neural networks, we delved into loss landscapes and their potential role in explaining model generalizability and trainability. We mathematically set the stage for meticulous analysis of training trajectories and level sets of parameter space, which we believe would be a fruitful future direction toward identifying fundamental properties of networks that can perform accurately on new data and can reliably learn to achieve low loss values on an objective function of interest. Then, by examining the effects

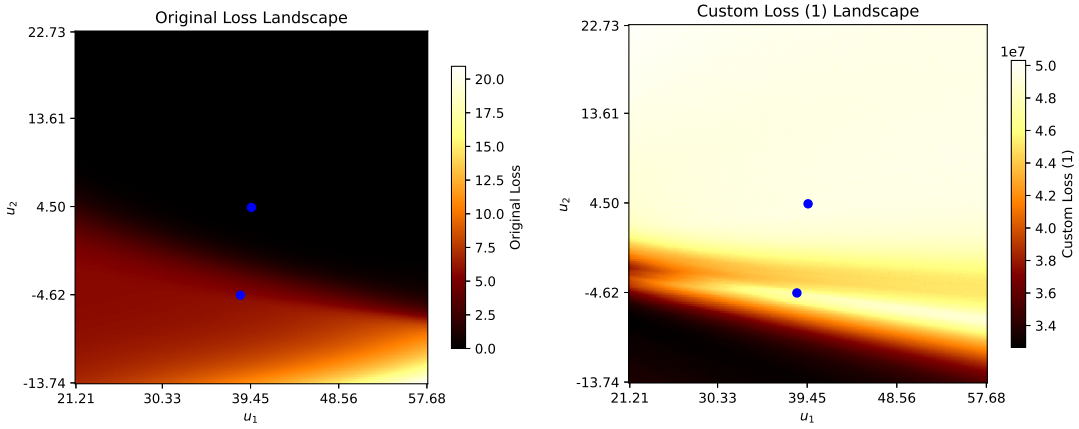



**Figure 6.17:** Loss landscapes of Model 2 (given in (6.8)) with respect to (a) classification loss and (b) the custom loss function with the initial and final models superimposed as blue points, centered about the final model.

of skip connections and model depth, we contributed to the ongoing discussion on architectural considerations in neural network design. Moreover, our investigation into input-output Jacobian norms provided novel perspectives on the influence of skip connections on network smoothness, offering valuable insights into network topology. Lastly, we provide preliminary results that suggest that an increase in model depth enables the manipulation of its polyhedral decomposition to exhibit particular properties of interest that render the network less susceptible to adversarial attack and increase its generalizability while still performing its task of interest accurately. In traversing the interdisciplinary terrain of data science and neural network theory, we have contributed to the existing body of knowledge and hopefully have paved the way for future research endeavors on these topics.

# Chapter 7

# Conclusions

As the technology of machine learning permeates high-risk applications, it becomes imperative to comprehend the inner mechanisms of their components. We note, however, that the rapid proliferation of machine learning technology has outpaced our understanding of its underlying principles. At the center of machine learning is the neural network, which is the focus of our research. In this work, we set out to demystify the intricate workings of neural networks through an analysis of polyhedral decompositions and the geometric and topological implications they have on their respective models. Chapter 2 delved into the mathematical underpinnings of high-dimensional spaces and dimensionality reduction methods, providing essential background for our subsequent analyses. In Chapter 3, we introduced two widely used neural network architectures and datasets whose complexity and size do not appear to have ever been used together in polyhedral-flavored machine learning research. These two chapters together set the stage for Chapter 4 where we provide a detailed formulation of neural networks' polyhedral decompositions and describe their role in defining neural network behavior. We demonstrated how these decompositions evolve during training, particularly with respect to polyhedral size and input-output Jacobian norm. Transitioning from explicit to implicit access to polyhedral decompositions, Chapter 5 showcased innovative tools for probing neural networks' geometry and stability. By harnessing bit vectors and input space landscapes, we uncovered hidden insights into the behavior of these complex systems. Our findings underscore the immense potential of implicit access to polyhedral decompositions in unraveling the mysteries of neural networks. As we push the boundaries of understanding, it becomes evident that we have only scratched the surface of the insights that lie within. Finally, in Chapter 6, we include some loose ends with respect to a data seriation project and interesting preliminary findings relating neural network input-output Jacobians to network smoothness and model depth to a network's capacity to achieve both accurate performance and exhibit particular properties in its polyhedral decomposition. This chapter largely outlined a few interesting future directions. In

conclusion, this work contributes to bridging the gap between theoretical understanding and practical applications of machine learning, paving the way for more robust and interpretable algorithms in the era of high-stakes deployment.

# Bibliography


[1] Huma Jamil, Yajing Liu, Turgay Caglar, Christina Cole, Nathaniel Blanchard, Christopher Peterson, and Michael Kirby. Hamming Similarity and Graph Laplacians for Class Partitioning and Adversarial Image Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 590–599, June 2023.

[2] Jan Kyncl (https://mathoverflow.net/users/24076/jan kyncl). How many vertices can a convex polytope have? MathOverflow. URL:https://mathoverflow.net/q/127426 (version: 2013-04-12).

[3] Yajing Liu, Christina Cole, Chris Peterson, and Michael Kirby. Relu Neural Networks, Polyhedral Decompositions, and Persistent Homology. In *International Conference on Machine Learning 2nd Annual Topology, Algebra, and Geometry in Machine Learning Workshop*, July 2023.

[4] Huma Jamil, Yajing Liu, Christina Cole, Nathaniel Blanchard, Emily J. King, Michael Kirby, and Christopher Peterson. Dual Graphs of Polyhedral Decompositions for the Detection of Adversarial Attacks. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2913–2921, Osaka, Japan, December 2022. IEEE.

[5] *NIST Digital Library of Mathematical Functions*. https://dlmf.nist.gov/, Release 1.2.0 of 2024-03-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds.

[6] John Hopcroft and Ravi Kannan. Computer science theory for the information age, 2012.

[7] Ghaith Makey, Özgün Yavuz, Denizhan K Kesim, Ahmet Turnalı, Parviz Elahi, Serim Ilday, Onur Tokel, and F Ömer Ilday. Breaking crosstalk limits to dynamic holography using orthogonality of high-dimensional random vectors. *Nature Photonics*, 13(4):251–256, 2019.

[8] John Cook. Random projection. *John D. Cook Consulting*. https://www.johndcook.com/blog/2019/04/16/random-projection/, April 2019.

[9] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[10] Ilias Diakonikolas, Daniel M. Kane, and Alistair Stewart. Statistical Query Lower Bounds for Robust Estimation of High-Dimensional Gaussians and Gaussian Mixtures. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 73–84, Berkeley, CA, October 2017. IEEE.

[11] Pentti Kanerva. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation*, 1(2):139–159, June 2009.

[12] Gregory Beylkin and Martin J. Mohlenkamp. Algorithms for Numerical Analysis in High Dimensions. *SIAM Journal on Scientific Computing*, 26(6):2133–2159, January 2005.

[13] Avrim Blum, John E. Hopcroft, and Ravindran Kannan. *Foundations of data science*. Cambridge University Press, New York, NY, first edition edition, 2020.

[14] Paul C Kainen and Věra Kůrková. Quasiorthogonal dimension of euclidean spaces. *Applied mathematics letters*, 6(3):7–10, 1993.

[15] Sanjeev Arora. Lecture 11: High Dimensional Geometry, Curse of Dimensionality, Dimension Reduction. *Princeton University.* https://www.cs.princeton.edu/courses/archive/fall14/cos521/lecnotes/lec11.pdf, 2014.

[16] Sofya Chepushtanova, Elin Farnell, Eric Kehoe, Michael Kirby, and Henry Kvinge. Dimensionality Reduction. In Nathan Carter, editor, *Data Science for Mathematicians*, pages

291–337. Chapman and Hall/CRC, First edition. | Boca Raton, FL : CRC Press, 2020., 1 edition, September 2020.

[17] Elena Facco, Maria d'Errico, Alex Rodriguez, and Alessandro Laio. Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific Reports*, 7(1):12140, September 2017.

[18] Phillip Pope, Chen Zhu, Ahmed Abdelkader, Micah Goldblum, and Tom Goldstein. The intrinsic dimension of images and its impact on learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[19] Jonathan Bac, Evgeny M. Mirkes, Alexander N. Gorban, Ivan Tyukin, and Andrei Zinovyev. Scikit-Dimension: A Python Package for Intrinsic Dimension Estimation. *Entropy*, 23(10), 2021.

[20] Alessio Ansuini, Alessandro Laio, Jakob H Macke, and Davide Zoccolan. Intrinsic dimension of data representations in deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[21] Mark Berman. Improved estimation of the intrinsic dimension of a hyperspectral image using random matrix theory. *Remote Sensing*, 11(9):1049, 2019. Publisher: Multidisciplinary Digital Publishing Institute.

[22] Francesco Camastra and Antonino Staiano. Intrinsic dimension estimation: Advances and open problems. *Information Sciences*, 328:26–41, January 2016.

[23] Daniele Granata and Vincenzo Carnevale. Accurate Estimation of the Intrinsic Dimension Using Graph Distances: Unraveling the Geometric Complexity of Datasets. *Scientific Reports*, 6(1):31377, November 2016.

[24] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E. Houle, Kenichi Kawarabayashi, and Michael Nett. Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 29–38, New York, NY, USA, 2015. Association for Computing Machinery.

[25] Kerstin Johnsson, Charlotte Soneson, and Magnus Fontes. Low Bias Local Intrinsic Dimension Estimation from Expected Simplex Skewness. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(1):196–202, January 2015. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[26] Gabriele Lombardi, Alessandro Rozza, Claudio Ceruti, Elena Casiraghi, and Paola Campadelli. Minimum Neighbor Distance Estimators of Intrinsic Dimension. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 6912, pages 374–389. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[27] Alessandro Rozza, Gabriele Lombardi, Marco Rosa, Elena Casiraghi, and Paola Campadelli. IDEA: Intrinsic Dimension Estimation Algorithm. In Giuseppe Maino and Gian Luca Foresti, editors, *Image Analysis and Processing – ICIAP 2011*, pages 433–442, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[28] Mingyu Fan, Nannan Gu, Hong Qiao, and Bo Zhang. Intrinsic dimension estimation of data by principal component analysis. Technical Report arXiv:1002.2050, arXiv, February 2010. arXiv:1002.2050 [cs] type: article.

[29] Anna Little and Yoon-Mo Jung. Multiscale Estimation of Intrinsic Dimensionality of Data Sets. In *AAAI Fall Symposium - Technical Report*, January 2009.

[30] Kevin M Carter, Raviv Raich, and Alfred O Hero III. On local intrinsic dimension estimation and its applications. *IEEE Transactions on Signal Processing*, 58(2):650–663, 2009.

[31] Mingyu Fan, Hong Qiao, and Bo Zhang. Intrinsic dimension estimation of manifolds by incising balls. *Pattern Recognition*, 42(5):780–787, May 2009.

[32] J.A. Costa, A. Girotra, and A.O. Hero. Estimating local intrinsic dimension with k-nearest neighbor graphs. In *IEEE/SP 13th Workshop on Statistical Signal Processing, 2005*, pages 417–422, 2005.

[33] Matthias Hein and Jean-Yves Audibert. Intrinsic dimensionality estimation of submanifolds in $R^d$. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML 2005, pages 289–296, New York, NY, USA, 2005. Association for Computing Machinery.

[34] Elizaveta Levina and Peter Bickel. Maximum likelihood estimation of intrinsic dimension. *Advances in neural information processing systems*, 17, 2004.

[35] F. Camastra and A. Vinciarelli. Estimating the intrinsic dimension of data with a fractal-based method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(10):1404–1407, October 2002.

[36] Balázs Kégl. Intrinsic dimension estimation using packing numbers. *Advances in neural information processing systems*, 15, 2002.

[37] Francesco Camastra and Alessandro Vinciarelli. Intrinsic dimension estimation of data: An approach based on Grassberger–Procaccia's algorithm. *Neural processing letters*, 14(1):27–34, 2001. Publisher: Springer.

[38] Jörg Bruske and Gerald Sommer. Intrinsic dimensionality estimation with optimally topology preserving maps. *IEEE Transactions on pattern analysis and machine intelligence*, 20(5):572–575, 1998.

[39] James W. Demmel. *Applied numerical linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[40] Paul Richard Halmos. *Finite-dimensional vector spaces*. Undergraduate texts in mathematics. Springer, New York Heidelberg Berlin, 4th ed edition, 1987.

[41] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1556, 2014.

[42] Haris Iqbal. HarisIqbal88/PlotNeuralNet v1.0.0. https://zenodo.org/record/2526395, December 2018.

[43] A. Humayun, R. Balestriero, G. Balakrishnan, and R. Baraniuk. SplineCam: Exact Visualization and Characterization of Deep Network Geometry and Decision Boundaries. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3789–3798, Los Alamitos, CA, USA, June 2023. IEEE Computer Society.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[45] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[46] Marco Gherardi. Solvable Model for the Linear Separability of Structured Data. *Entropy*, 23(3):305, March 2021.

[47] A. Torralba, R. Fergus, and W.T. Freeman. 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, November 2008.

[48] Christiane Fellbaum, editor. *WordNet: an electronic lexical database*. Language, speech, and communication. MIT Press, Cambridge, Mass, 1998.

[49] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*, May 2012.

[50] Theo Lacombe, Yuichi Ike, Mathieu Carriere, Frederic Chazal, Marc Glisse, and Yuhei Umeda. Topological Uncertainty: Monitoring Trained Neural Networks through Persistence of Activation Graphs. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, 2021.

[51] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, Miami, FL, June 2009. IEEE.

[52] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, December 2015.

[53] Sizhe Chen, Xiaolin Huang, Zhengbao He, and Chengjin Sun. DAmageNet: A Universal Adversarial Dataset, December 2019. Number: arXiv:1912.07160 arXiv:1912.07160 [cs, eess, stat].

[54] Sizhe Chen, Zhengbao He, Chengjin Sun, Jie Yang, and Xiaolin Huang. Universal Adversarial Attack on Attention and the Resulting Dataset DAmageNet. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(4):2188–2197, 2022.

[55] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples, March 2015. Number: arXiv:1412.6572 arXiv:1412.6572 [cs, stat].

[56] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the Number of Linear Regions of Deep Neural Networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2924–2932, Cambridge, MA, USA, 2014. MIT Press.

[57] Randall Balestriero and Richard G. Baraniuk. Mad Max: Affine Spline Insights into Deep Learning. *Proceedings of the IEEE*, 109(5):704–727, 2021.

[58] Randall Balestriero, Romain Cosentino, Behnaam Aazhang, and Richard Baraniuk. The Geometry of Deep Networks: Power Diagram Subdivision. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[59] Boris Hanin and David Rolnick. Deep ReLU Networks Have Surprisingly Few Activation Patterns. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[60] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 2847–2854, Sydney, NSW, Australia, August 2017. JMLR.org.

[61] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks, 2017. Publisher: arXiv Version Number: 4.

[62] Boris Hanin and David Rolnick. Complexity of Linear Regions in Deep Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2596–2604. PMLR, May 2019. ISSN: 2640-3498.

[63] Huan Xiong, Lei Huang, Mengyang Yu, Li Liu, Fan Zhu, and Ling Shao. On the Number of Linear Regions of Convolutional Neural Networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10514–10523. PMLR, 13–18 Jul 2020.

[64] Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations, 2013. Publisher: arXiv Version Number: 5.

[65] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and Generalization in Neural Networks: an Empirical Study. In *International Conference on Learning Representations*, 2018.

[66] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[67] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. Series Title: Lecture Notes in Computer Science.

[68] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts, 2016.

[69] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[70] Andrew Maas. Rectifier Nonlinearities Improve Neural Network Acoustic Models, 2013.

[71] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28-3 of *Proceedings of*

*Machine Learning Research*, pages 1319–1327, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[72] Kevin Jarrett, Koray Kavukcuoglu, Marc' Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, Kyoto, September 2009. IEEE.

[73] Imre Bárány and Zoltán Füredi. Computing the volume is difficult. *Discrete & Computational Geometry*, 2(4):319–326, December 1987.

[74] M. E. Dyer and A. M. Frieze. On the Complexity of Computing the Volume of a Polyhedron. *SIAM Journal on Computing*, 17(5):967–974, October 1988.

[75] Martin Dyer, Alan Frieze, and Ravi Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the ACM*, 38(1):1–17, January 1991.

[76] Ravi Kannan, László Lovász, and Miklós Simonovits. Random walks and an $O(N^5)$ volume algorithm for convex bodies. *Random Struct. Algorithms*, 11(1):1–50, aug 1997.

[77] Benno Büeler, Andreas Enge, and Komei Fukuda. Exact Volume Computation for Polytopes: A Practical Study. In Gil Kalai and Günter M. Ziegler, editors, *Polytopes — Combinatorics and Computation*, pages 131–154. Birkhäuser Basel, Basel, 2000.

[78] Alexander Barvinok and Mark Rudelson. A quick estimate for the volume of a polyhedron, 2021. Publisher: arXiv Version Number: 3.

[79] Augustin Chevallier, Frédéric Cazals, and Paul Fearnhead. Efficient computation of the the volume of a polytope in high-dimensions using Piecewise Deterministic Markov Processes. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 10146–10160. PMLR, 28–30 Mar 2022.

[80] Lei Zhao and Wu-Sheng Lu. Chebyshev Centre of a Polyhedron. *University of Victoria.* https://studentweb.uvic.ca/~leizhao/573notes/note11, February 2020.

[81] Amir Beck and Yonina C. Eldar. Regularization in Regression with Bounded Noise: A Chebyshev Center Approach. *SIAM Journal on Matrix Analysis and Applications*, 29(2):606–625, January 2007.

[82] B. Brosowski, F. Deutsch, Ch. Blanc, R. Glowinski, G. Golub, P. Henrici, H. O. Kreiss, A. Ostrowski, and J. Todd, editors. *Parametric Optimization and Approximation*, volume 72 of *International Series of Numerical Mathematics / Internationale Schriftenreihe zur Numerischen Mathematik / Série internationale d'Analyse numérique*. Birkhäuser Basel, Basel, 1985.

[83] School of Mathematics and Physics The University of Queensland, Brisbane, Australia, Robert Salomone, Radislav Vaisman, and Dirk Kroese. Estimating the Number of Vertices in Convex Polytopes. In *4th Annual International Conference on Operations Research and Statistics ( ORS 2016 )*. Global Science & Technology Forum ( GSTF ), January 2016.

[84] Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled Elbassioni, and Vladimir Gurvich. Generating all vertices of a polyhedron is hard. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm - SODA '06*, pages 758–765, Miami, Florida, 2006. ACM Press.

[85] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[86] Yajing Liu, Turgay Caglar, Christopher Peterson, and Michael Kirby. Integrating geometries of ReLU feedforward neural networks. *Frontiers in Big Data*, 6:1274831, November 2023.

[87] Keith Ball. An elementary introduction to modern convex geometry. *Flavors of geometry*, 31:1–58, 1997.

[88] Dan Sloughter. *The Calculus of Functions of Several Variables*. Orange Grove Texts Plus, 2001.

[89] Ali Araabi, Vlad Niculae, and Christof Monz. Joint dropout: Improving generalizability in low-resource neural machine translation through phrase pair variables. In Masao Utiyama and Rui Wang, editors, *Proceedings of Machine Translation Summit XIX, Vol. 1: Research Track*, pages 12–25, Macau SAR, China, September 2023. Asia-Pacific Association for Machine Translation.

[90] Poorya Mianjy and Raman Arora. On Convergence and Generalization of Dropout Training. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21151–21161. Curran Associates, Inc., 2020.

[91] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[92] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, July 2012. Number: arXiv:1207.0580 arXiv:1207.0580 [cs].

[93] Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer, New York, 2006.

[94] Naveed Akhtar and Ajmal Mian. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey, February 2018. Number: arXiv:1801.00553 arXiv:1801.00553 [cs].

[95] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis*. Wiley Series in Probability and Statistics. Wiley, Hoboken, NJ, second edition, 2023.

[96] Trevor Hastie and Werner Stuetzle. Principal Curves. *Journal of the American Statistical Association*, 84(406):502–516, 1989.

[97] Robert Arn. *On the formulation and uses of SVD-based generalized curvatures*. Dissertation, Colorado State University, 2016.

[98] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *International Conference on Learning Representations*, 2017.

[99] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-SGD: Biasing Gradient Descent Into Wide Valleys. In *International Conference on Learning Representations*, 2017.

[100] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp Minima Can Generalize For Deep Nets. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1019–1028. PMLR, 06–11 Aug 2017.

[101] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

[102] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[103] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, Cham, 2018.

[104] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why ReLU Networks Yield High-Confidence Predictions Far Away From the Training Data and How to Mitigate

the Problem. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 41–50, 2019.

[105] Connor Shorten and Taghi M. Khoshgoftaar. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1):60, December 2019.

[106] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Uncertainty Estimation Using a Single Deep Deterministic Neural Network. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9690–9700. PMLR, 13–18 Jul 2020.

[107] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[108] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, Las Vegas, NV, USA, June 2016. IEEE.

[109] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, Los Alamitos, CA, USA, jul 2017. IEEE Computer Society.

[110] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, Venice, October 2017. IEEE.

[111] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning, April 2015. Number: arXiv:1412.6614 arXiv:1412.6614 [cs, stat].

[112] Vladimir Naumovich Vapnik. *Statistical Learning Theory*. Adaptive and learning systems for signal processing, communications, and control. Wiley, New York, 1998.

[113] Peter Bartlett and Shahar Mendelson. Rademacher and Gaussian Complexities: Risk Bounds and Structural Results. *Journal of Machine Learning Research*, 3:463–482, November 2002.

[114] Sayan Mukherjee, Partha Niyogi, Tomaso Poggio, and Ryan Rifkin. Learning theory: stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization. *Advances in Computational Mathematics*, 25(1-3):161–193, July 2006.

[115] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks, June 2017. Number: arXiv:1605.07146 arXiv:1605.07146 [cs].

[116] Akira Sakai, Taro Sunagawa, Spandan Madan, Kanata Suzuki, Takashi Katoh, Hiromichi Kobashi, Hanspeter Pfister, Pawan Sinha, Xavier Boix, and Tomotake Sasaki. Three approaches to facilitate invariant neurons and generalization to out-of-distribution orientations and illuminations. *Neural Networks*, 155:119–143, November 2022.

# Appendix A

# Equivalence of Classical and Modified Gram-Schmidt (CGS/MGS)

The only difference between the two algorithms is the equation for $r_{j,i}$ in the inner for-loop (see Algorithm 1). The base case we consider is the first time the algorithm enters the inner for-loop, which is when $i = 2$ and $j = 1$. In this case, CGS dictates that $r_{1,2} = q_2^\top a_2$ and MGS that $r_{1,2} = q_2^\top q_2$. When $j = 1$, $q_i = a_i$ because $q_i$ has not yet been updated, so $r_{1,2}$ are equal for CGS and MGS. Suppose that after $\xi$ updates, the two algorithms are equal. Then

$$q_\xi^\top a_i = q_\xi^\top q_i.$$

It remains to show that

$$q_{\xi+1}^\top a_i = q_{\xi+1}^\top q_i. \tag{A.1}$$

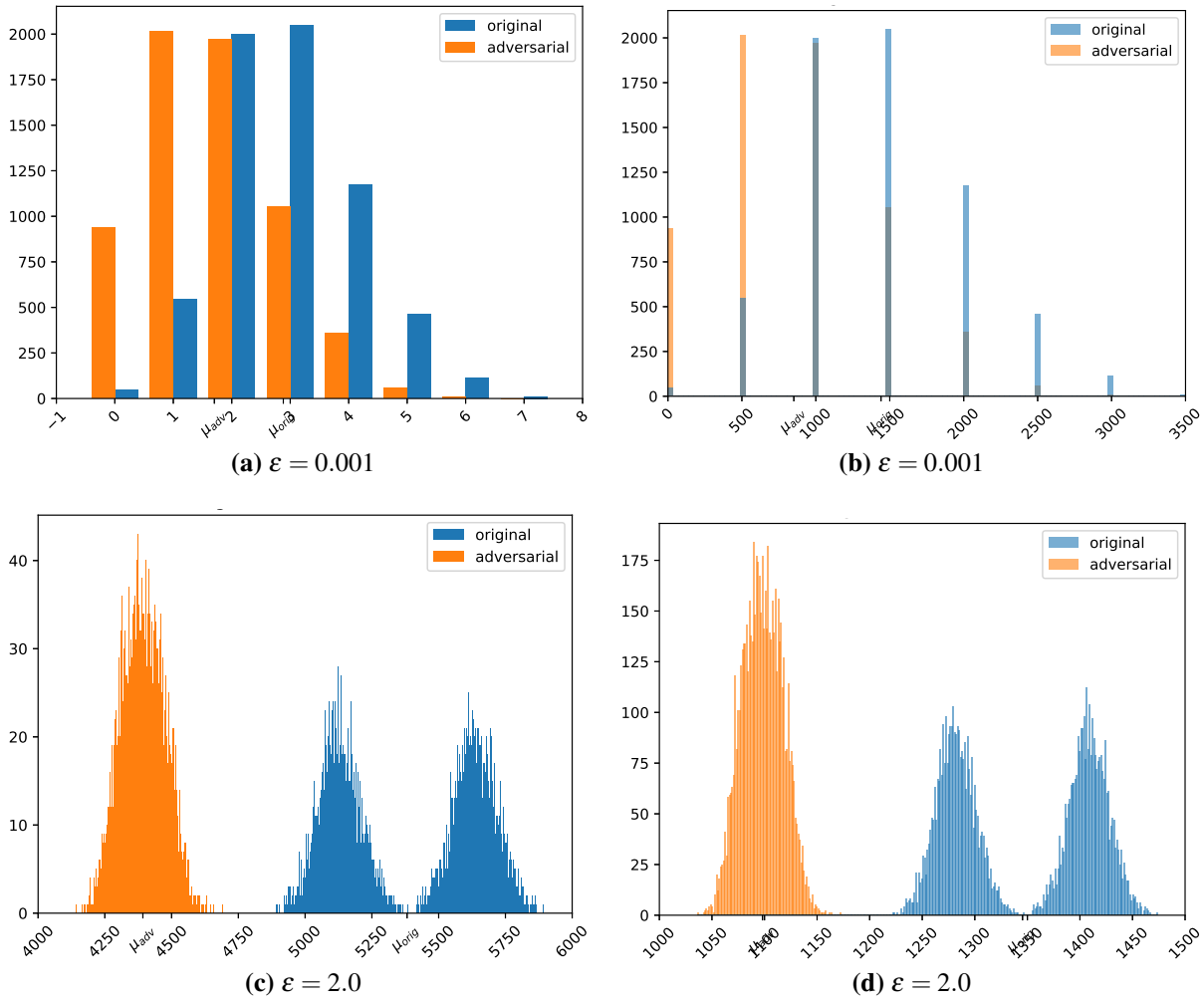
After $j$ updates (where $j \in \mathbb{N}, j < i$) in the inner for-loop, we have that $q_i = a_i - \sum_{k=1}^{j} r_{k,i} q_k$ or that $a_i = q_i + \sum_{k=1}^{j} r_{k,i} q_k$. By substitution, the left-hand side of (A.1) becomes

$$q_{\xi+1}^\top (q_i + \sum_{k=1}^{j} r_{k,i} q_k)$$

where $\sum_{k=1}^{j} r_{k,i} q_k = 0$ because they have been constructed to be mutually orthogonal. Therefore, (A.1) holds and the two algorithms are equal by induction.

# Appendix B

# Polyhedral Size




**(a)** $\varepsilon = 0.001$ **(b)** $\varepsilon = 0.001$

**(c)** $\varepsilon = 2.0$ **(d)** $\varepsilon = 2.0$

**Figure B.1:** Histogram of Hamming distances before (left) and after (right) normalizing by Euclidean distance.

# Appendix C

# Shortest Path Length Plots

## C.1 Circular Training Data

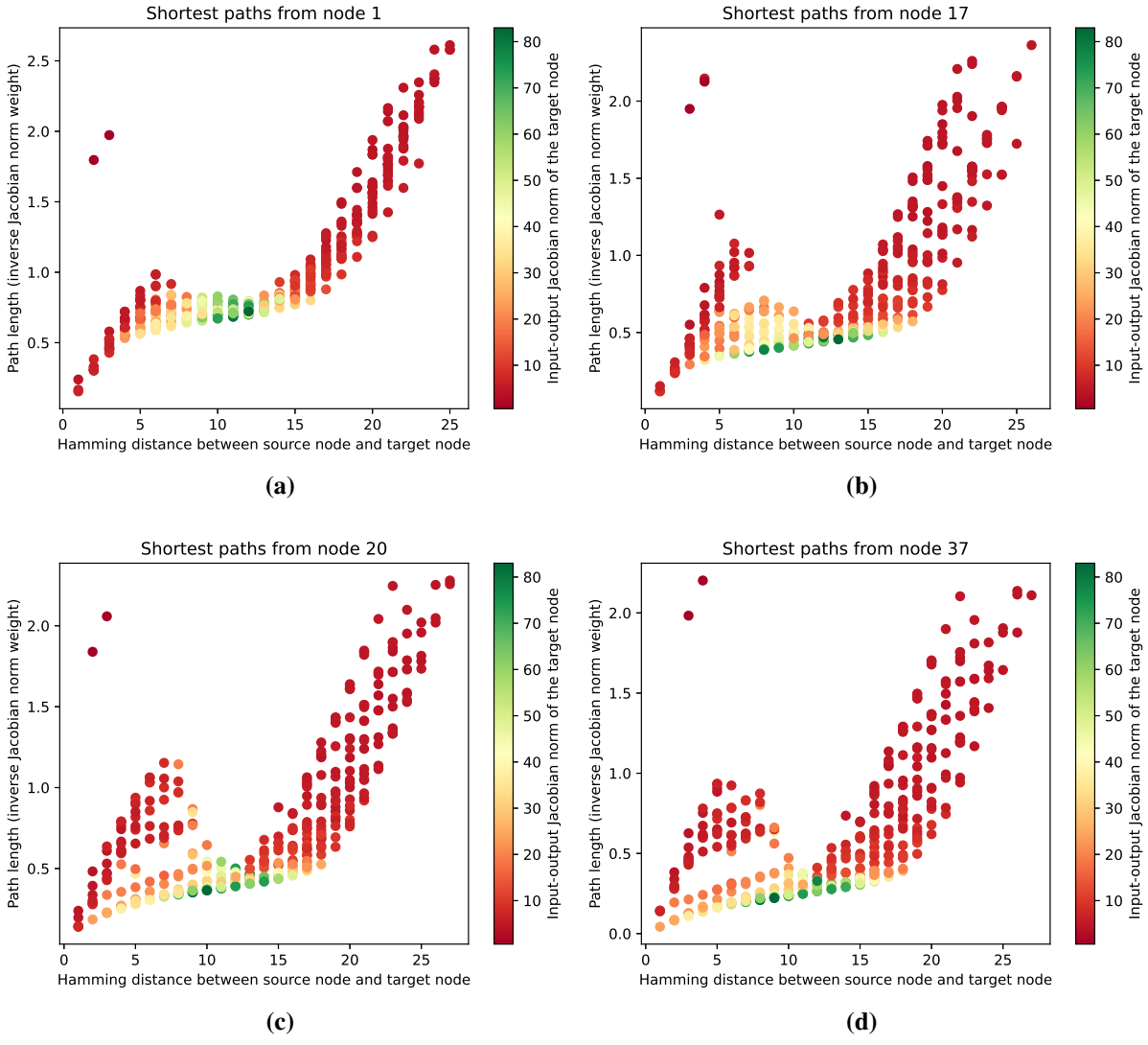



**Figure C.1:** The shortest path in terms of the inverse of the input-output Jacobian norm from $P$ to the polyhedra in $\partial G_\alpha(P)$ where the x-axis gives $\alpha$ and the y-axis gives path lengths. Polyhedron $P$ lies inside the nested circles of training data.

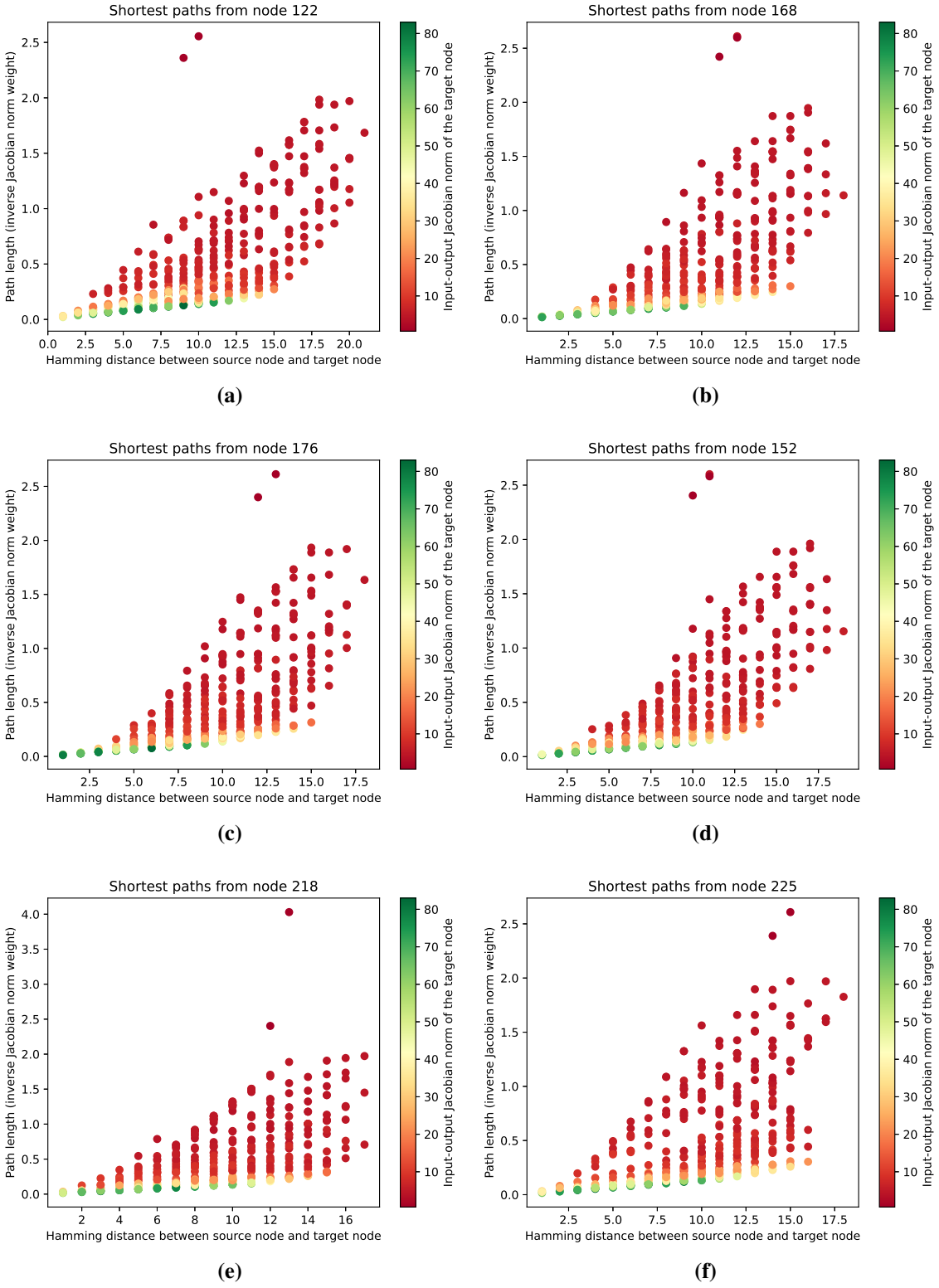



**Figure C.2:** The shortest path in terms of the inverse of the input-output Jacobian norm from $P$ to the polyhedra in $\partial G_\alpha(P)$ where the x-axis gives $\alpha$ and the y-axis gives path lengths. Polyhedron $P$ lies in the trained region of the input space; polyhedra in plots (a)-(c) contain training data while those in plots (d)-(f) do not.

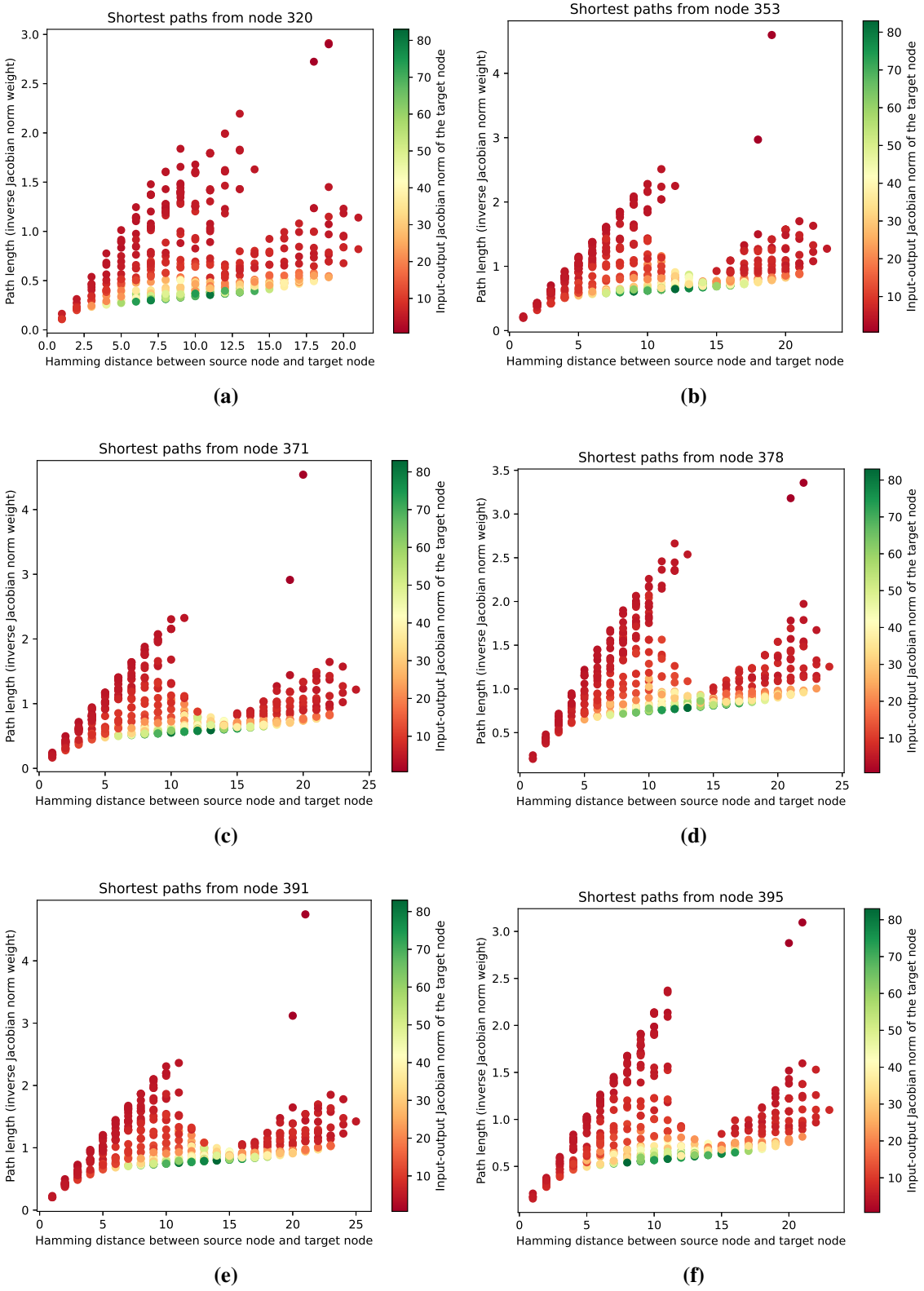



**Figure C.3:** The shortest path in terms of the inverse of the input-output Jacobian norm from $P$ to the polyhedra in $\partial G_\alpha(P)$ where the x-axis gives $\alpha$ and the y-axis gives path lengths. Polyhedron $P$ lies outside of the concentric circles of training data.

## C.2 Interleaving Half Circles

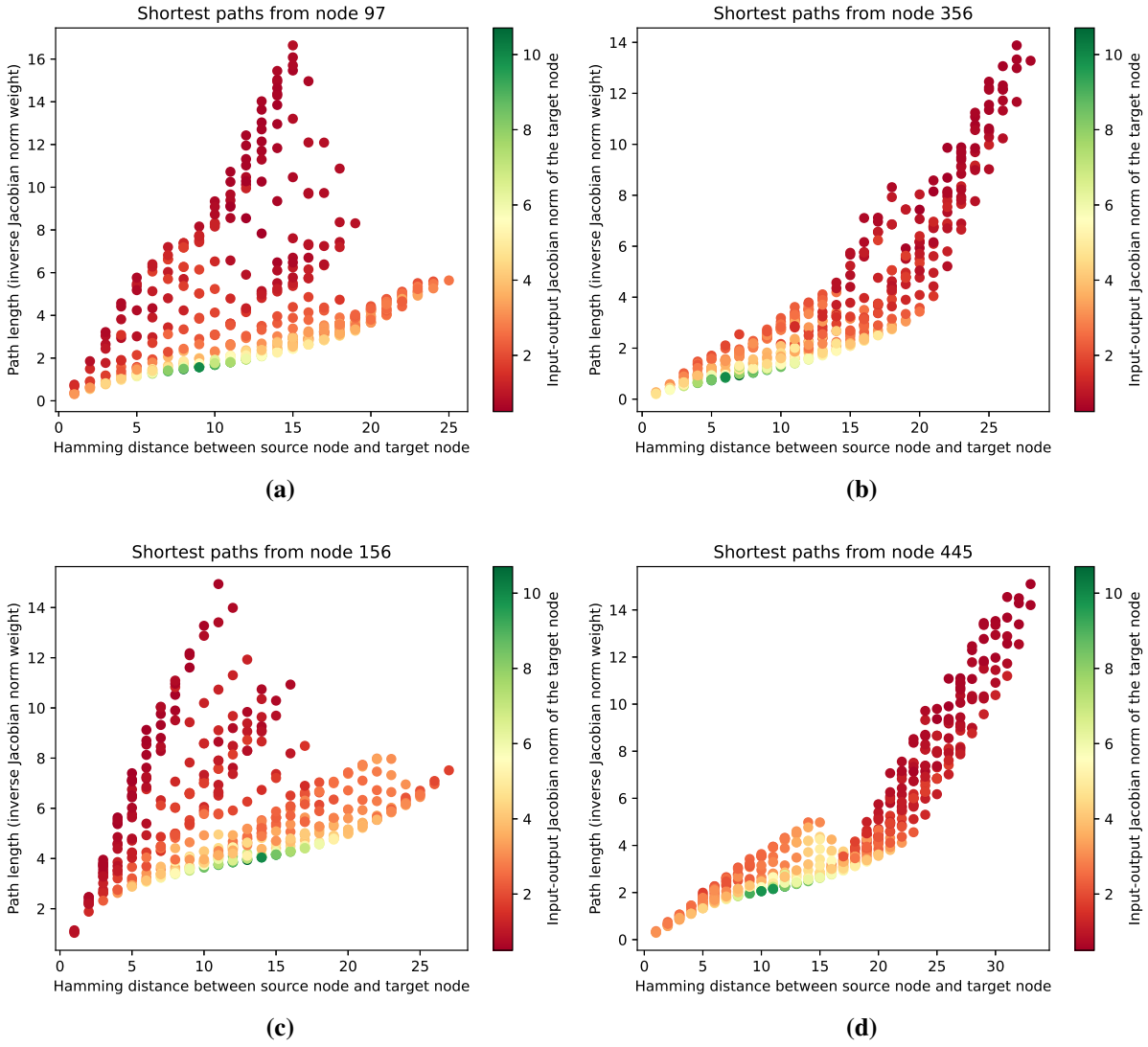



**Figure C.4:** The shortest path in terms of the inverse of the input-output Jacobian norm from $P$ to the polyhedra in $\partial G_\alpha(P)$ where the x-axis gives $\alpha$ and the y-axis gives path lengths. Polyhedron $P$ lies outside of the trained region of the input space.

## C.3 Cluster Training Data

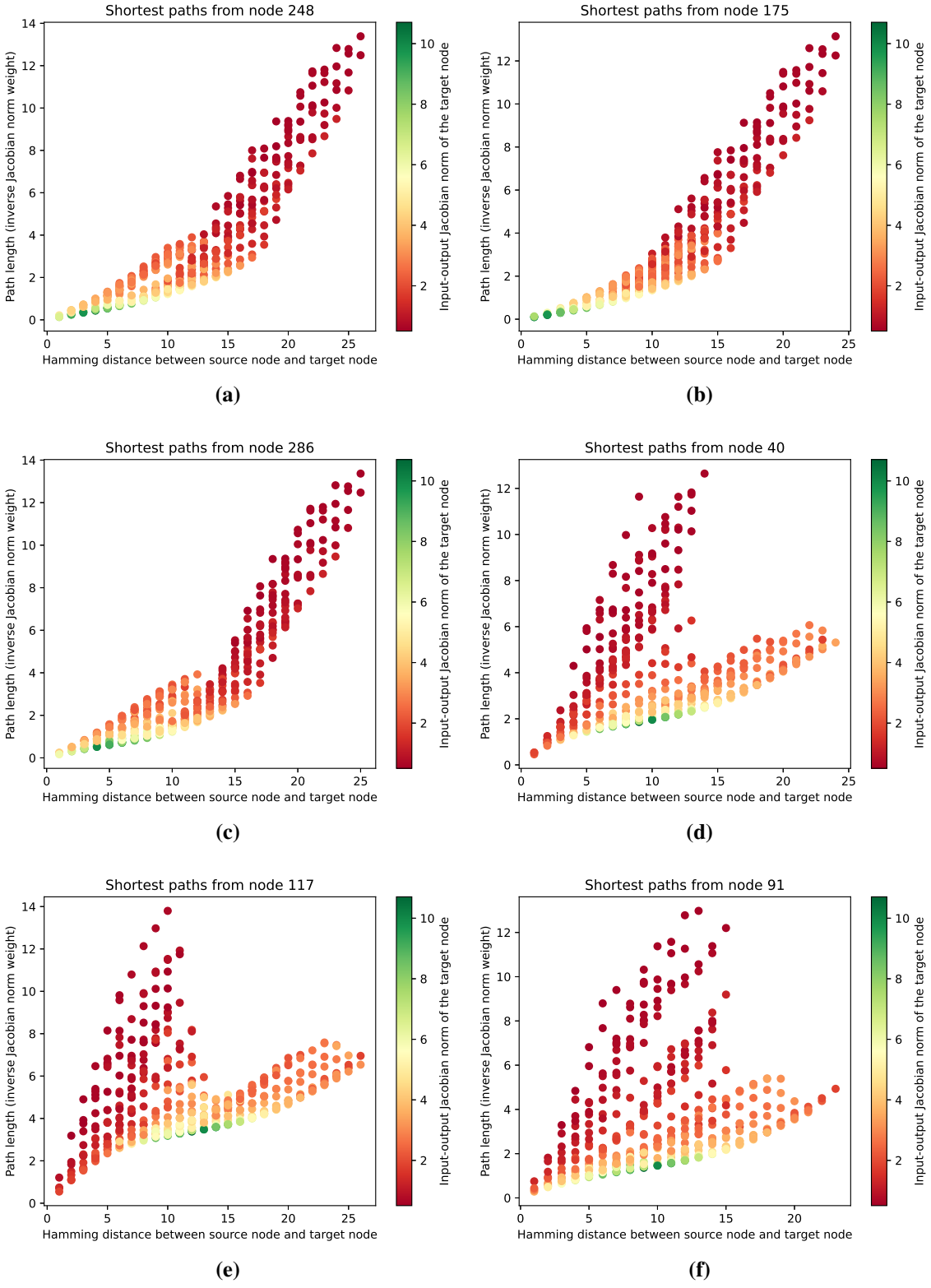



**Figure C.5:** The shortest path in terms of the inverse of the input-output Jacobian norm from $P$ to the polyhedra in $\partial G_\alpha(P)$ where the x-axis gives $\alpha$ and the y-axis gives path lengths. Polyhedron $P$ lies in the trained region of the input space.

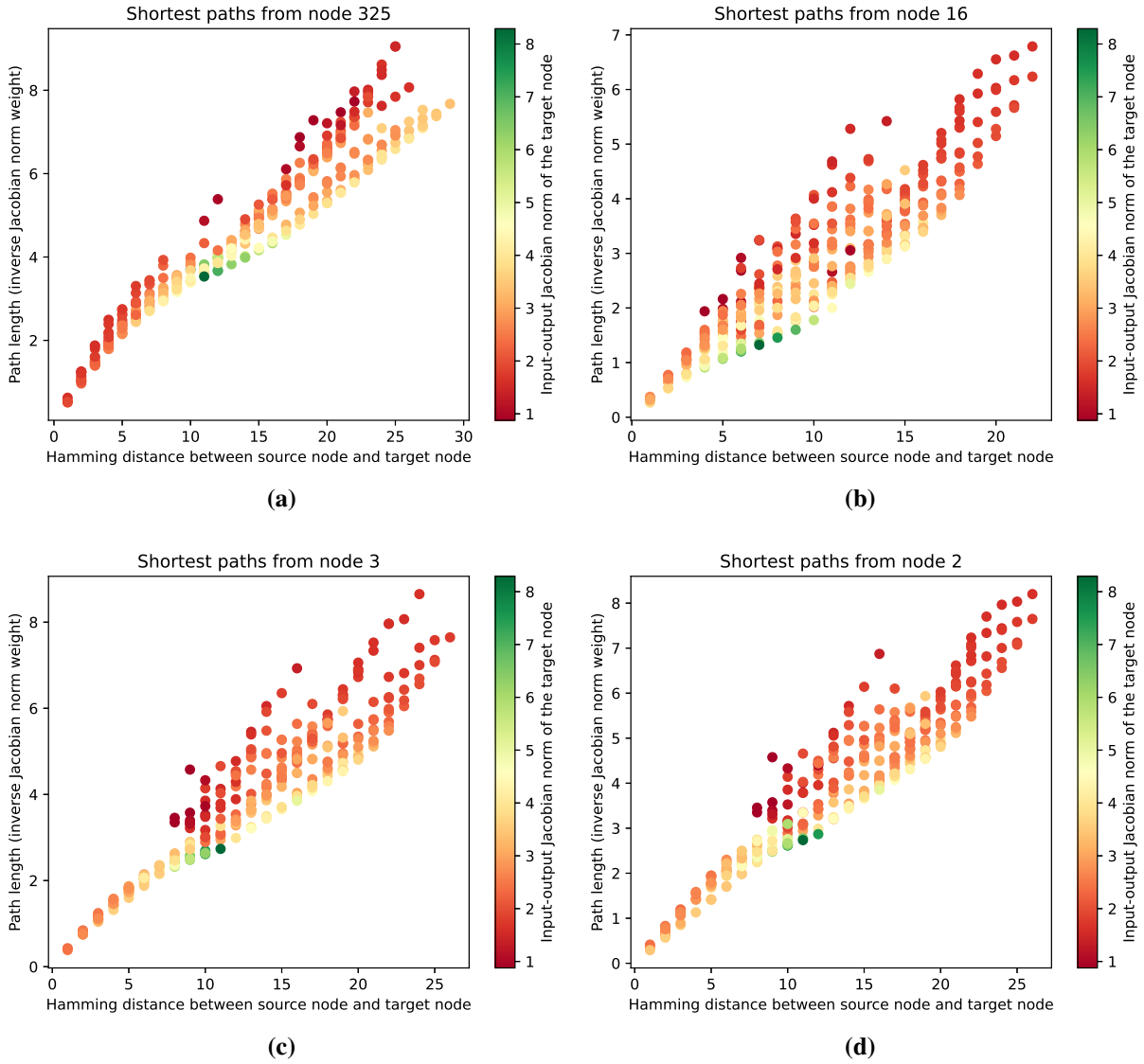



**Figure C.6:** The shortest path in terms of the inverse of the input-output Jacobian norm from $P$ to the polyhedra in $\partial G_\alpha(P)$ where the x-axis gives $\alpha$ and the y-axis gives path lengths. Polyhedron $P$ lies in the trained region of the input space.

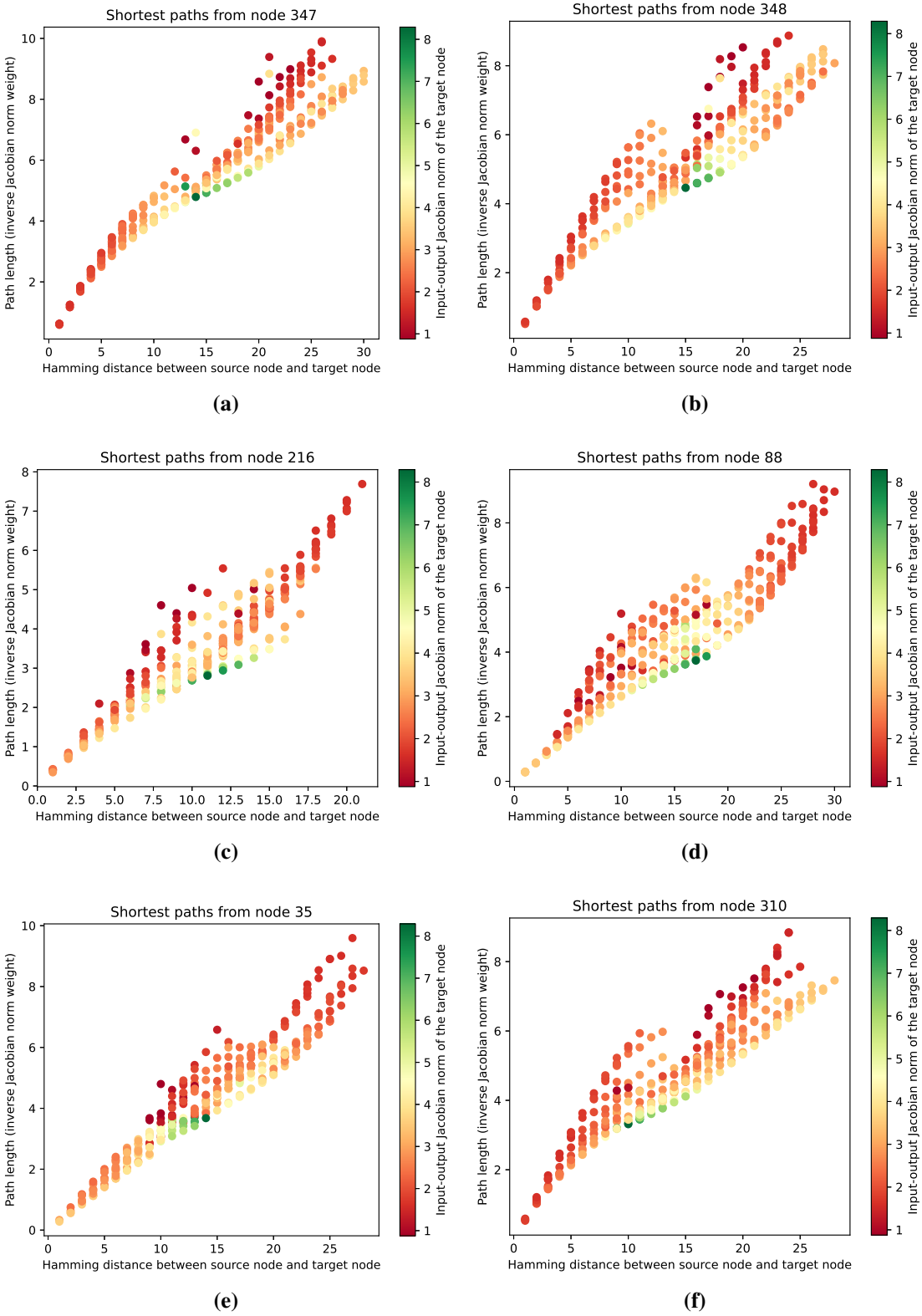
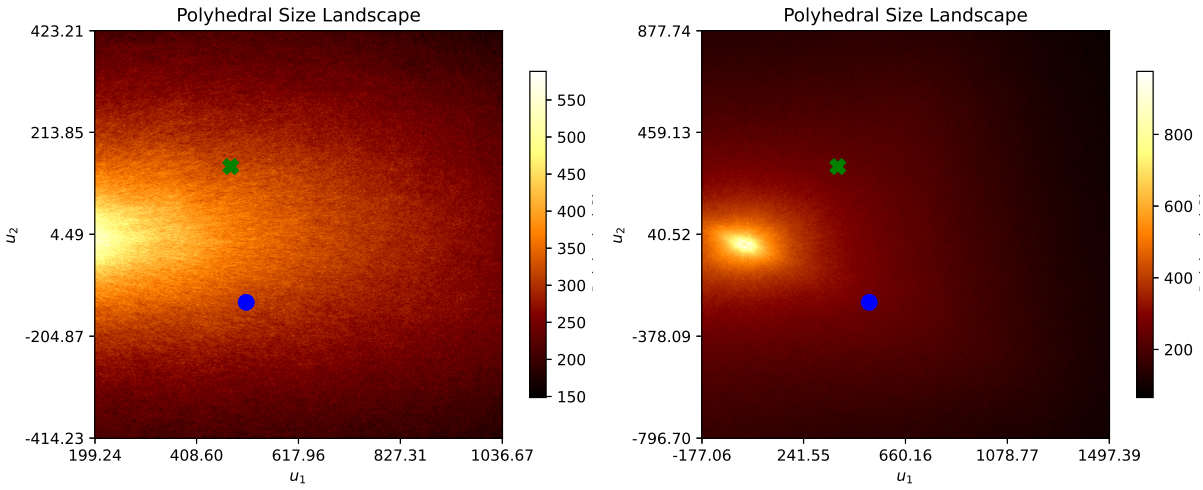



**Figure C.7:** The shortest path in terms of the inverse of the input-output Jacobian norm from $P$ to the polyhedra in $\partial G_\alpha(P)$ where the x-axis gives $\alpha$ and the y-axis gives path lengths. Polyhedron $P$ lies outside the trained region of the input space.

# Appendix D

# Polyhedral Size Landscapes

We include a brief description of polyhedral size landscapes an two examples in Figure D.1. Similar to the other input space landscapes, polyhedral size landscapes were created by first discretizing a region of interest of the neural network domain input. Then, for each point in this grid of samples, the polyhedral size was approximated by performing Algorithm 8 with $D = 10$ and $\varepsilon = 0.001$ and then computing the average of the resulting distribution. The particular discretizations used to produce the landscapes given in Figure D.1 were of size $1200 \times 1200$. Therefore, each of these plots required 14400000 Hamming distance computations, which appears to be insufficient to determine any details in the polyhedral decomposition. Further refinement would be required, which would increase the already significant computational expense of producing these landscapes. We can, however, easily notice high refinement about the origin from these plots.




**(a)** **(b)**

**Figure D.1:** Polyhedral size landscapes through two ImageNet-DAmageNet pairs (where the ImageNet image is denoted by a green 'X' and the DAmageNet image by a blue point) colored by the average Hamming distance computed by Algorithm 8.

# Appendix E

# Loss Landscape Slices

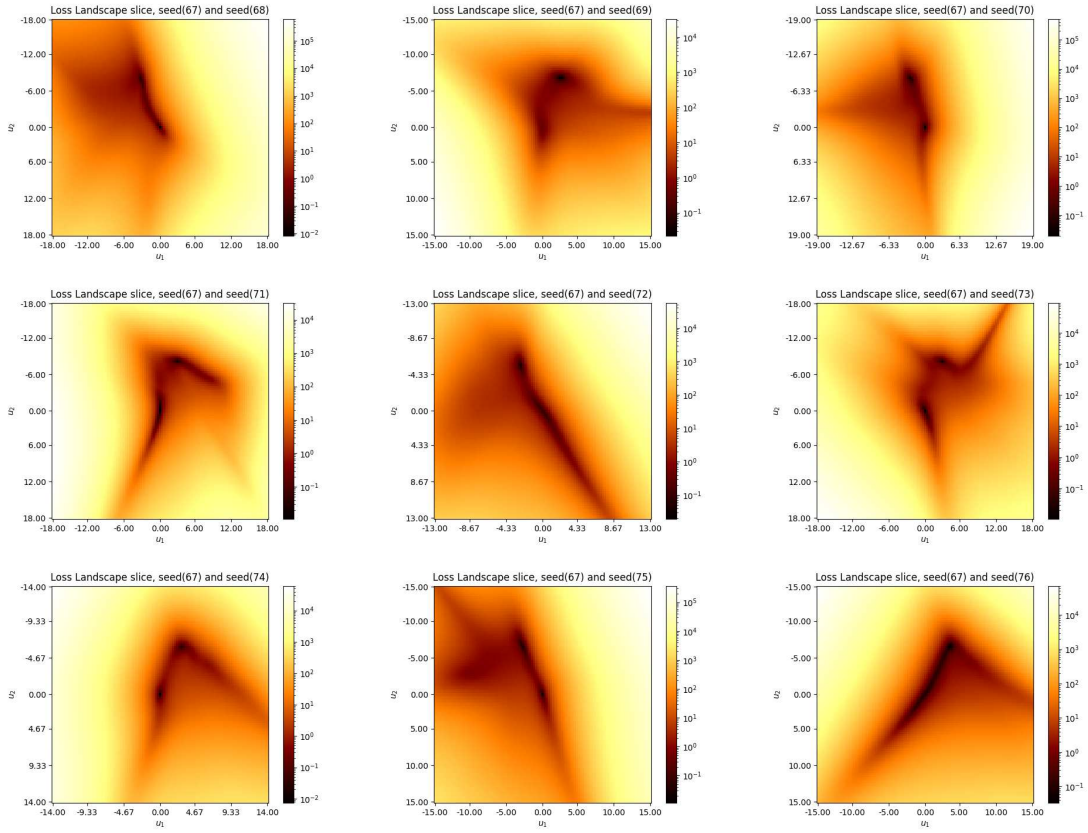



**Figure E.1:** Loss landscapes through two models, the model after training with random initialization given by seed 67 and all other random initializations.