ReLU Neural Networks, Polyhedral Decompositions, and Persistent Homology

Yajing Liu [1] **Christina Cole** [1] **Christopher Peterson** [1] **Michael Kirby** [1]

Abstract

A ReLU neural network leads to a finite polyhedral decomposition of input space and a corresponding finite dual graph. We show that while this dual graph is a coarse quantization of input space, it is sufficiently robust that it can be combined with persistent homology to detect homological signals of manifolds in the input space from samples. This property holds for a variety of networks trained for a wide range of purposes that have nothing to do with this topological application. We found this feature to be surprising and interesting; we hope it will also be useful.

1. Introduction

The rectified linear unit (ReLU) function is the default activation function for many well known, deep, feed forward neural networks (AlexNet, ResNet, Inception, SqueezeNet, etc). These networks frequently use some convolutional layers to decrease the number of parameters being trained and often utilize skips to decrease issues from overfitting. The importance of ReLU feed forward neural networks (FFNNs) has motivated an extensive investigation of their properties from a variety of aspects to demystify their "black-box nature". In this paper, we start from the observation that, regardless of their use of skips and/or convolutions, a ReLU FFNN $F : \mathbb{R}^m \to \mathbb{R}^n$ decomposes the input space ($\mathbb{R}^m$) into convex polyhedra and assigns to each polyhedron a unique binary vector that encodes the ReLU activation pattern of all nodes in the ReLU layers of the network. More precisely, the neural network assigns a binary vector to each input point, assigning identical binary vectors to input points lying in the same polyhedra. Despite this granularity imposed on input space, we found that the homology of manifolds embedded in the input space of the network can still be detected via persistent homology applied solely to distance measures built from the binary vectors of the points in the sampling. After explaining the connection between polyhedra and binary vectors in a ReLU FFNN, we describe methods for finding the binary vectors associated with the nearby neighbors of a given polyhedron in the decomposition. For small networks, one can (in principle) use these methods to determine all the polyhedra in the decomposition and determine their proximity within the decomposition. Toward this goal, we exploit the following observation: *two polyhedra share a facet if and only if their associated binary vectors differ in exactly one bit*. Due to this property, we propose the Hamming distance between binary vectors as a proxy for proximity between polyhedra in the decomposition. More precisely, if $\mathcal{X}$ is a collection of data points in the input space, each $x \in \mathcal{X}$ lies in one of the polyhedra and can be assigned the binary vector, $s(x)$, that labels the polyhedron. Using the Hamming metric, we build a network driven distance matrix between pairs of binary vectors associated to data points in $\mathcal{X}$. Using a software package such as Ripser (see `https://live.ripser.org`), we illustrate how the coarse distance measure corresponding to a polyhedral decomposition of input space can be used to extract homological information about a manifold lying in the input space from sample points on the manifold. It is worth noting that the Hamming distance between polyhedral bit vectors is an approximation for the smallest number of polyhedral steps between two polyhedra (i.e. for the shortest path between vertices on the dual graph of the polyhedral decomposition). (Jamil et al., 2023) has demonstrated the efficacy of the Hamming distance in illuminating the mechanisms underlying neural network functionality.

The paper is organized as follows: Section 2 introduces the basic background on ReLU FFNNs, the polyhedral decomposition associated to a network, the binary vectors assigned to each polyhedron, the linear model that determines each polyhedron, the linear program for filtering out redundant inequalities in the linear model, and the affine linear function that is attached to each polyhedron. Section 3 outlines algorithms for determining the binary vectors that occur for polyhedra in the entire input space and for polyhedra that occur within a bounded region. Section 4 describes how the polyhedral decomposition, realized through binary vectors, can be combined with persistent homology to uncover topological features in data. Section 5 concludes the work and

[1]Department of Mathematics, Colorado State University, Fort Collins, USA. Correspondence to: Yajing Liu <yajing.liu@colostate.edu>.

Proceedings of the 2nd Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML) at the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. 2023. *Copyright 2023 by the author(s).*

sketches directions for future research.

2. Neural Networks and Polyhedral Decompositions

This section gives the basic definitions of a ReLU neural network and describes the connection to polyhedral decompositions and binary vectors.

2.1. Notation for Neural Networks

Consider an $(L+1)-$layer ReLU feedforward neural network:

$$\mathbb{R}^m \xrightarrow[\text{ReLU}]{(W_1,b_1)} \mathbb{R}^{h_1} \xrightarrow[\text{ReLU}]{(W_2,b_2)} \mathbb{R}^{h_2} \to \ldots \to \mathbb{R}^{h_{L-1}} \xrightarrow[\text{ReLU}]{(W_L,b_L)} \mathbb{R}^{h_L} \xrightarrow{(W_{L+1},b_{L+1})} \mathbb{R}^n. \quad (1)$$

In this model, $\mathbb{R}^m$ is the input space, $\mathbb{R}^n$ is the output space, and h_i corresponds to the number of nodes at layer i. Layer 0 corresponds to the input space and layer $L+1$ corresponds to the output space (so $h_0 = m$ and $h_{L+1} = n$). We let $W_i \in \mathbb{R}^{h_i \times h_{i-1}}$ and $b_i \in \mathbb{R}^{h_i}$ denote the weight matrix and bias vector of layer i, respectively. The activation functions for the hidden layers (layers $1, \ldots, L$) are assumed to be ReLU functions (applied coordinate-wise) while the map to the last layer (the output layer) is assumed to be affine linear (without a ReLU function being applied to the image). Recall that the ReLU function is the map $ReLU : \mathbb{R} \to \mathbb{R}$ given by

$$ReLU(a) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{if } a \leq 0. \end{cases} \quad (2)$$

The ReLU map is a map on real numbers that is piecewise linear and continuous. It can be naturally extended to a piecewise linear continuous map on vector spaces (which we also denote as ReLU). More precisely, we define $ReLU : \mathbb{R}^{h_i} \to \mathbb{R}^{h_i}$, by applying the ReLU function to each coordinate of $x \in \mathbb{R}^{h_i}$. Let $w_{i,j}$ denote the jth row of W_i and let $b_{i,j}$ denote the jth entry of b_i. Given an input data point $x \in \mathbb{R}^m$, we denote the output of x in layer i as $F_i(x)$. Thus, with this notation we have $F_i(x) \in \mathbb{R}^{h_i}$, $F_0(x) = x$, and

$$\begin{aligned} F_i(x) &= \text{ReLU}(W_i F_{i-1}(x) + b_i) \\ &= \begin{bmatrix} \max\{0, w_{i,1} F_{i-1}(x) + b_{i,1}\} \\ \vdots \\ \max\{0, w_{i,h_i} F_{i-1}(x) + b_{i,h_i}\} \end{bmatrix}. \end{aligned} \quad (3)$$

2.2. Definitions of Binary Vectors

Consider model (1). Given an input data point $x \in \mathbb{R}^m$, for each hidden layer i (so $1 \leq i \leq L$), we introduce a binary (bit) vector

$$s_i(x) = [s_{i,1}(x) \ \ldots \ s_{i,h_i}(x)]^\top \in \mathbb{R}^{h_i},$$

where $s_{i,j}(x)$ (with $1 \leq j \leq h_i$) is defined as follows:

$$s_{i,j}(x) = \begin{cases} 1 & \text{if } w_{i,j} F_{i-1}(x) + b_{i,j} > 0 \\ 0 & \text{if } w_{i,j} F_{i-1}(x) + b_{i,j} \leq 0. \end{cases} \quad (4)$$

Thus, for each point $x \in \mathbb{R}^m$, we have a sequence of binary vectors $s_1(x), s_2(x), \ldots, s_L(x)$. We can stack the binary vectors associated to x to make a long column vector

$$s(x) = [s_1^\top(x) \ \ldots \ s_L^\top(x)]^\top \in \mathbb{R}^h, \quad (5)$$

where $h = \sum_{i=1}^L h_i$ is the total number of nodes in the hidden layers. We call $s(x)$ the binary vector of x.

Different points from the input space $\mathbb{R}^m$ can have the same binary vector. We next show that the set of points that have the same binary vector, $\{x' : s(x') = s(x), x' \in \mathbb{R}^m\}$, form a convex polyhedron in $\mathbb{R}^m$.

2.3. Linear Model for Binary Vectors

Let $s_1, s_2, \ldots, s_L$ denote a given sequence of binary vectors for model (1). For each layer i $(1 \leq i \leq L)$, we describe inequality constraints that must be satisfied for an input data point to have the same binary vector s_i. To describe the inequalities in a consistent manner, we introduce a sign vector of $1's$ and $-1's$ for each hidden layer. For layer i, define $s'_i = [s'_{i,1} \ \ldots \ s'_{i,h_i}]^\top$ with

$$s'_{i,j} = \begin{cases} 1 & \text{if } s_{i,j} = 0 \\ -1 & \text{if } s_{i,j} = 1. \end{cases} \quad (6)$$

In layer 1, because $F_1(x) = \text{ReLU}(W_1 x + b_1)$, any data point x that has the bit vector s_1 satisfies the following linear inequality:

$$\text{diag}(s'_1)(W_1 x + b_1) \leq 0 \quad (7)$$

where $\text{diag}(v)$ is a square diagonal matrix with the elements of vector v on the main diagonal. The inequality (7) is established by a fundamental principle: the affine output generated by the first hidden layer, in response to input x, must satisfy a greater-than-zero condition for nodes where the corresponding binary vector is set to 1, and a less-than-or-equal-to-zero condition for nodes where the corresponding binary vector is set to 0.

Suppose that x has the bit vector sequence $s_1, s_2, \ldots, s_L$. Let $\hat{W}_j = W_j \text{diag}(s_{j-1}) \hat{W}_{j-1}$ and $\hat{b}_j = W_j \text{diag}(s_{j-1}) \hat{b}_{j-1} + b_j$ for $2 \leq j \leq L$ with $\hat{W}_1 = W_1, \hat{b}_1 = b_1$. By model (1), the following equation holds for $1 \leq j \leq L$:

$$F_j(x) = \text{ReLU}(W_j F_{j-1}(x) + b_j) = \text{diag}(s_j)(\hat{W}_j x + \hat{b}_j)$$

where $F_0(x) = x$.

More generally, any data point x that has s_j as its bit vector for layer j should satisfy the following linear inequalities:

$$\text{diag}(s'_j)\hat{W}_j x \leq \text{diag}(s'_j)(-\hat{b}_j). \tag{8}$$

Let $A_j = \text{diag}(s'_j)\hat{W}_j$ and $c_j = \text{diag}(s'_j)(-\hat{b}_j)$ for $1 \leq j \leq L$. Combining (7) and (8), we have

$$Ax \leq c, \tag{9}$$

where

$$A = [A_1^\top \; A_2^\top \; \ldots \; A_L^\top]^\top \text{ and } c = [c_1^\top \; c_2^\top \; \ldots \; c_L^\top]^\top$$

with $A_i \in \mathbb{R}^{h_i \times m}$ and $c_i \in \mathbb{R}^{h_i}$. Note that the set $P = \{x \in \mathbb{R}^m : Ax \leq c\}$ is a convex polyhedron (potentially empty, potentially unbounded). Considering the totality of points in the input space, the weights of a ReLU neural network lead to a decomposition of the input space into a collection of bounded and unbounded polyhedra and each polyhedron has a corresponding bit vector. Note that a random bit vector may or may not correspond to a non-empty polyhedra.

It is not difficult to see that for a polyhedron P of full dimension (the same dimension as the ambient space), there is a unique smallest subset of inequalities, which we denote by (A', c'), that one can obtain from (A, c) that leaves the polyhedron P unchanged. Thus, A' is built from a subset of rows of A and c' is the corresponding subset of rows of c such that the set of points that satisfy $Ax \leq c$ is the same as the set of points that satisfy $A'x \leq c'$.

Write

$$A = \left[a_1 \; a_2 \; \ldots \; a_h\right]^\top \text{ and } \quad c = \left[c^1 \; c^2 \; \ldots \; c^h\right]^\top$$

with $a_i \in \mathbb{R}^m$ and $c^i \in \mathbb{R}$. To determine if $a_i x \leq c^i$ is a redundant constraint, first define

$$\tilde{A} = \left[a_1 \; a_2 \; \ldots \; a_{i-1} \; a_{i+1} \ldots \; a_h\right]^\top$$

and

$$\tilde{c} = \left[c^1 \; c^2 \; \ldots \; c^{i-1} \; c^{i+1} \; \ldots \; c^h\right]^\top.$$

Next, consider the following linear program

$$\begin{aligned} &\underset{x}{\text{maximize}} \;\; a_i^\top x \\ &\text{s. t. } \; \tilde{A}x \leq \tilde{c}. \end{aligned} \tag{10}$$

If the optimal objective value of (10) is less than or equal to c^i, then the ith linear inequality is redundant, and we can remove $a_i^\top$ and c^i from A and c, respectively. We determine (A', c') by iterating this process to remove all redundant constraints.

Given a bit vector s, the i-th entry of s is called active if the ith row of A is in A', and inactive otherwise. For any input x in the polyhedron determined by the bit vector s, the output of x is determined by the single affine map: $G(x) = W_{L+1}\text{diag}(s_L)\hat{W}_L x + W_{L+1}\text{diag}(s_L)\hat{b}_L + b_{L+1}$.

We summarize some of the points of this section, together with implications (many of which we leave to the reader) in the bullet points that follow. We emphasize that the polyhedral decomposition of input space has an associated dual graph. This graph has vertices corresponding to polyhedra and edges corresponding to polyhedra that share an $m-1$ dimensional face. It is not hard to show that this dual graph is bipartite (refer to Appendix B). We will be using this graph, implicitly, to detect homological signals via persistent homology. For the interested reader, we point out the following references related to topological features and combinatorial features of the type of neural networks we are considering (Grigsby et al., 2022; Masden, 2022b). We would also like to acknowledge the quickly-growing library of work in the area of polyhedral theory, a survey of which can be found in (Huchette et al., 2023).

The following is a summary of the key ideas presented either explicitly or implicitly in this section:

- A ReLU FFNN determines a continuous piecewise linear map $F : \mathbb{R}^m \to \mathbb{R}^n$.
- The neural network leads to a decomposition of $\mathbb{R}^m$ into a collection of bounded and unbounded convex polyhedra.
- A binary vector, supported on the hidden nodes of the network, can be attached to each point in the domain of the network. The value of the binary vector associated to a given input $x \in \mathbb{R}^m$, at a given node, is 1 if ReLU was not applied at the node and 0 if ReLU was applied at the node (the ReLU activation pattern).
- If h denotes the number of hidden nodes in the network, then there are, a priori, 2^h possible binary vectors and thus 2^h possible polyhedra in the decomposition of input space. In reality, the number of realized convex polyhedra in the domain is much smaller.
- If two points lie in the interior of the same convex polyhedron then they determine the same binary vector.
- If two points are in the interior of distinct convex polyhedra then they determine distinct binary vectors.
- From the ReLU activation pattern for points in a convex polyhedron, one can determine an affine linear equation that represents the behavior of the neural network on points in the polyhedron. If the polyhedron is denoted by P, then for points in P, the function $F : \mathbb{R}^m \to \mathbb{R}^n$ can be expressed as $F(x) = A_P x + b_P$ for some matrix A_P and some vector b_P.
- Input values that determine a value of 0 on some node, before applying ReLU, are precisely the input values that lie on the boundary of distinct convex polyhedra. This is due to the fact that applying ReLU to 0 has the same effect as not applying ReLU to 0.

- If two convex polyhedra, P_1, P_2, share an $(m-1)$-dimensional face, then the binary vectors associated to each of the polyhedra differ in one bit. Furthermore, the affine linear functions $A_{P_1} + b_{P_1}$ and $A_{P_2} + b_{P_2}$ agree on this $(m-1)$-dimensional face.
- Any polyhedral decomposition of $\mathbb{R}^m$ has a natural dual graph with vertices corresponding to m-dimensional polyhedra and edges corresponding to polyhedra sharing an $m-1$ dimensional facet.
- The Hamming distance between binary vectors (that represent two polyhedra) can be used as an approximation for the smallest number of polyhedral steps between the two polyhedra (i.e. the length of a minimal geodesic on the dual graph).

3. Algorithms for Bit Vector Search and Examples

The number of polyhedra in the input space or within a bounded region of the input space provides a measure of the network's expressivity and complexity. Upper and lower bounds on the maximal number of polyhedra obtainable from a given ReLU FFNN architecture can be found in (Pascanu et al., 2014), (Montufar et al., 2014), (Raghu et al., 2017), (Arora et al., 2018), and (Serra et al., 2018). Several algorithms ((Xiang et al., 2018), (Yang et al., 2020), and (Xu et al., 2021)) have been developed to compute the exact polyhedra decomposition of the input space through layer-by-layer linear inequality solving. (Masden, 2022a) has given a full encoding of the canonical polyhedral complex across all dimensions. Larger decomposition examples can be computed using a method developed by (Vincent & Schwager, 2021), which enumerates all polyhedra in the input space as follows:

- Start with a random point $x \in \mathbb{R}^m$ and determine its bit vector $s(x)$. This bit vector labels a polyhedron P.
- Find the active bits in $s(x)$ for the polyhedron P.
- Each active bit corresponds to a neighboring polyhedron. Each neighboring polyhedron has a bit vector that can be obtained by "flipping" one of the active bits for P. Thus, one can find all of the neighboring polyhedra for P, in terms of their binary vectors.
- Repeat the process to find the neighbors of each of these newly identified polyhedra.
- The number of active bits for a polyhedron is equal to the number of nearest neighbors of the polyhedron. The previous steps continue until we have a list of polyhedra $\mathcal{P}$ that satisfies the property that for each $P \in \mathcal{P}$, the set of nearest neighbors of P is itself a subset of $\mathcal{P}$.

This process leads to a set $\mathcal{P}$ of convex polyhedra that decompose input space. Such decompositions have been used to define network-imposed distance metrics for model inputs (Balestriero & Baraniuk, 2018). We coarsen these metrics and show they can still be used to detect homological signals. Equating neural networks of various types (convolutional neural networks, residual networks, skip-connected networks, recurrent neural networks) to max-affine spline operators has been carried out by (Balestriero & Baraniuk, 2018). The paper (Sattelberg et al., 2020) investigated the behavior of the ReLU FFNNs based on the structure of the decomposition, together with the affine map attached to each polyhedron.

While there are multiple methods that can be used to determine the polyhedral decomposition, and their associated binary vectors, imposed by the weights of a ReLU neural network, all of these methods are woefully inadequate for the large and deep networks that are commonplace today. This is not a fault of the algorithms, many modern networks contain many millions of hidden nodes and have input spaces with dimension well beyond 100,000. There are just too many polyhedra represented by such networks. However, for small networks, any of the methods will work. We present two straightforward methods. The first is a brute-force method formulated as a Linear programming problem. The second, which is the traversal-and-pruning method outlined above, computationally improves upon the first. We demonstrate our methods with examples. Pseudocode for these two methods are provided in Appendix A.

3.1. Algorithms

A mixed-integer linear program to count the number of polyhedra in a bounded region can be found in (Serra et al., 2018). We present a linear program that can also count polyhedra, determine if they share a facet, and whose implementational simplicity is useful. Our proposed method yields not only the count of polyhedra in the entire input space $\mathbb{R}^m$ but also their respective binary vectors. To count the number of polyhedra in a bounded region, the linear inequality constraints in (15) need to be augmented to include bounds for variables. For example, when the input is 2-dimensional and the bounded region is determined by $a_1 \leq x_1 \leq b_1$, $a_2 \leq x_2 \leq b_2$, the constraints become $\tilde{A}^j x \leq \tilde{c}^j$ where $\tilde{A}^j$ is the concatenation of A^j and B while $\tilde{c}^j$ is the concatenation of c^j and d, with

$$B = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}^T \text{ and } d = \begin{bmatrix} b_1 & -a_1 & b_2 & -a_2 \end{bmatrix}^T.$$

Algorithm 1 is fast for small neural network structure. However, (Hanin & Rolnick, 2019) proved that the number of bit vectors that correspond to polyhedra is approximately h^m which is much smaller than 2^h for large h, highlighting the computational savings that can be procured by only traversing active bit vectors instead of all possibilities as is done by

the brute-force method. These savings are what Algorithm 2 has to offer.

Algorithm 1 is easy to implement but slow. The idea for Algorithm 2 is based on the fact that each polyhedron corresponding to $Ax \leq c$ is determined by $A'x \leq c'$ and two adjacent polyhedra that share one facet differ by one active bit. This active bit corresponds to the hyperplane containing this facet. The key step for Algorithm 2 is to find the active bits for a given bit vector using the method mentioned in Section 2.2. Once identified, these active bits can then be used to find all neighboring polyhedra, starting a scheme that ripples through the desired domain until all polyhedra are found. The idea behind this algorithm also motivated the reachable polyhedral marching algorithm in (Vincent & Schwager, 2021). We make a small modification to enumerate the polyhedra in a bounded region rather than in the entire, unbounded domain. For a bounded region, the bit vector can be expanded by checking whether the corresponding polyhedron hits the boundary or not. For example, when $m = 2$ and the bounded region is defined by $a_1 \leq x_1 \leq b_1, a_2 \leq x_2 \leq b_2$, the bit vector s can be expanded by 1 more bit with 1's or 0's by checking if its $Ax \leq c$ solutions hit any of the domain boundaries defined by $x_1 = a_1$, $x_1 = b_1$, $x_2 = a_2$, or $x_2 = b_2$ (1 if true, 0 otherwise). For the bit vectors that have 1 for their last bit, we update their labels with 1 after they are added to $\mathcal{P}$. The reason is that only if the bit vector of the initial point does not hit the region boundaries, the bit vectors that hit the region boundaries will be derived by flipping the active bits of their neighbors, which means that all its adjacent polyhedra in the bounded region are already in $\mathcal{P}$.

3.2. Examples

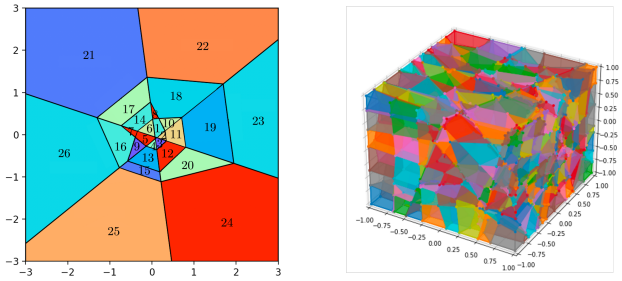
In this section, we visualize the polyhedral decomposition, determined by a ReLU FFNN, on a bounded region for two basic models:

$$\mathbb{R}^2 \xrightarrow[\text{ReLU}]{(W_1,b_1)} \mathbb{R}^3 \xrightarrow[\text{ReLU}]{(W_2,b_2)} \mathbb{R}^3 \xrightarrow{(W_3,b_3)} \mathbb{R} \tag{11}$$

$$\mathbb{R}^3 \xrightarrow[\text{ReLU}]{(W_1,b_1)} \mathbb{R}^{10} \xrightarrow[\text{ReLU}]{(W_2,b_2)} \mathbb{R}^{10} \xrightarrow[\text{ReLU}]{(W_3,b_3)} \mathbb{R}^{10} \xrightarrow{(W_4,b_4)} \mathbb{R} \tag{12}$$

We first applied model (11) to fit $f_1(x_1, x_2) = x_1^2 + x_2^2 - 2/3$ using 10000 points uniformly sampled in $[-1, 1]^2$ and model (12) to fit $f_2(x_1, x_2, x_3) = (x_1 - 1)^2 + 2x_2^2 + x_3^2 + 1$ using 125000 points uniformly sampled in $[-1, 1]^3$. We used TensorFlow to train the above models with batch size of 50 and early stopping criterion based on the convergence of validation loss.

We used Algorithm 1 to enumerate all binary vectors in $\mathbb{R}^2$ and in $[-1, 1]^2$. We used Algorithm 2 to enumerate all binary vectors in $\mathbb{R}^3$ and in $[-1, 1]^3$. We plotted the polyhedra by finding their extremal vertices.



(a) 2-D polyhedral decomposition

(b) 3-D polyhedral decomposition

Figure 1. Visualizations of low-dimensional polyhedral decompositions. A detailed discussion on (a) can be found in Appendix B along with the exact binary vectors that label all of the regions of the decomposition.

Figure 1(a) and Figure 1(b) provide visualizations of the polyhedral decomposition derived from model (11) and model (12), respectively. For Figure 1(a), there are 25 polyhedra in the bounded region $[-1, 1]^2$ and 27 polyhedra in $\mathbb{R}^2$. Of these polyhedra, 21 are bounded and 6 are unbounded. The binary vectors are superimposed onto the polyhedra. Note that the binary vectors associated to different polyhedra differ in exactly one bit if and only if they share a facet. For Figure 1(b), there are 1858 polyhedra in the bounded region $[-1, 1]^3$ and 3331 polyhedra in $\mathbb{R}^3$.

4. Persistent Homology and Polyhedral Decompositions

It has been well documented that the Euclidean distance between sampled points on a manifold in $\mathbb{R}^n$ can be employed to detect the topology of the manifold. In this section, we provide a description of Vietoris-Rips persistent homology and illustrate how it can be effectively combined with the non-Euclidean distance measure, associated to the polyhedral decomposition, to also identify homological features. The exact (not persistent) homology of the decision boundaries of networks were calculated for some randomly initialized networks with up to four input dimensions in (Masden, 2022a). Persistent homology has been a rapidly developing branch of topology largely due to its usefulness in data analysis and machine learning (Ghrist, 2008; Zomorodian & Carlsson, 2004; Carlsson, 2009; Edelsbrunner & Harer, 2008) (a collection of additional resources and videos can be found at `https://www.aatrn.net`). Work linking persistent homology and neural networks has been appearing with increasing frequency, please see the following for a sampling of some recent works in this direction (Rieck et al., 2018; Zhao et al., 2020; Carrière et al., 2020; Birdal et al., 2021). The description of persistent homology given below is extremely condensed. The interested reader is encouraged

to read the survey article (Ghrist, 2008) where additional details can be found.

4.1. Persistent Homology

Consider a metric space, C, with distance function $d : C \times C \to \mathbb{R}$. Let $x^{(1)}, x^{(2)}, \ldots x^{(N)}$ be points in C. We can utilize d to build an $N \times N$ matrix D by setting $D_{i,j} = d(x^{(i)}, x^{(j)})$. D will be hollow (i.e. $D_{i,i} = 0$), symmetric, and non-negative. From D we can build a family, $A(t)$, of $N \times N$ $\{0, 1\}$-matrices parameterized by a real parameter t using the rule

$$A(t)_{i,j} = \begin{cases} 0 & \text{if } D(i,j) > t \text{ or if } i = j \\ 1 & \text{else.} \end{cases} \tag{13}$$

For each t, $A(t)$ can be viewed as an adjacency matrix of a graph $G(t)$. Let $CL(t)$ denote the clique complex of $G(t)$ (Hausmann, 1995). By construction, $CL(t)$ is a simplicial complex and there is a natural inclusion map $CL(t_1) \hookrightarrow CL(t_2)$ whenever $t_1 < t_2$. Let $\mathbb{F}$ be a field. A simplicial complex S has an associated chain complex $S_\bullet$ of $\mathbb{F}$-vector spaces. The failure of the chain complex to be exact at location i is measured by the ith homology $H_i(S_\bullet, \mathbb{F})$ (which itself is an $\mathbb{F}$-vector space). The inclusion map $CL(t_1) \hookrightarrow CL(t_2)$ induces a chain map $CL(t_1)_\bullet \to CL(t_2)_\bullet$. Whenever you have a chain map $F : S_\bullet \to T_\bullet$ between chain complexes, $S_\bullet, T_\bullet$ you get associated linear maps between $H_i(S_\bullet, \mathbb{F})$ and $H_i(T_\bullet, \mathbb{F})$ for each i. Thus, the chain map $CL(t_1)_\bullet \to CL(t_2)_\bullet$ induces, for each i, a linear map

$$H_i(CL(t_1)_\bullet, \mathbb{F}) \to H_i(CL(t_2)_\bullet, \mathbb{F}).$$

If we pick values $t_1 < t_2 < \cdots < t_k$, then we can build a nested sequence of simplicial complexes

$$CL(t_1) \subset CL(t_2) \subset \cdots \subset CL(t_k)$$

which leads to

$$H_i(CL(t_1)_\bullet, \mathbb{F}) \to H_i(CL(t_2)_\bullet, \mathbb{F}) \to \cdots \to H_i(CL(t_k)_\bullet, \mathbb{F}), \tag{14}$$

where the arrows denote linear maps.

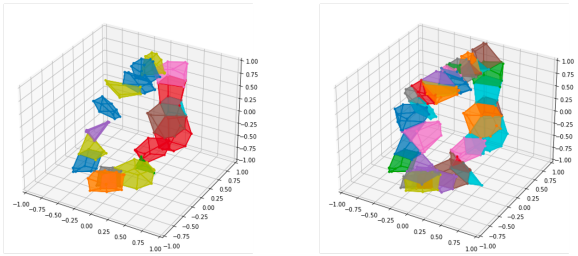
Each of the $H_i(CL(t_j)_\bullet, \mathbb{F})$ are finite dimensional $\mathbb{F}$-vector spaces and we can view (14) as a finitely generated graded $\mathbb{F}[x]$ module, where $\mathbb{F}[x]$ is the ring of polynomials. This module is frequently called the ith persistence module associated to the nested sequence of simplicial complexes. Since $\mathbb{F}[x]$ is a principal ideal domain, the $F[x]$ module has a decomposition into its invariant factors by the well known structure theorem for finitely generated graded modules over a principal ideal domain. Each invariant factor can be viewed as a homology class that has a birth time t_b and a death time t_d (possibly infinite) and this invariant factor can be represented as an interval $[t_b, t_d]$. This collection of intervals corresponds to the barcode representation of the invariant factors of a persistence module.

4.2. Combining Persistence with a Polyhedral Decomposition

First we provide two examples based on the polyhedral decomposition of model (12).

Example 4.1. We consider the circle in $\mathbb{R}^3$ (the input space) with parameterization given by $(0, cos(t), sin(t))$. We sampled the circle at 20 evenly spaced points $x^{(1)}, x^{(2)}, \ldots, x^{(20)}$. Each of these points lie in one of the polyhedra from Figure 1(b). A picture of the polyhedra encountered by the 20 sample points is found in Figure 2(a). We recorded the binary vectors for each of the encountered polyhedra. This gave a total of 20 binary vectors but only 19 were distinct. We labeled these distinct binary vectors $s^{(1)}, s^{(2)}, \ldots s^{(19)}$. We built a 19×19 matrix E by setting $E_{i,j}$ equal to the Hamming distance between $s^{(i)}$ and $s^{(j)}$. The resulting matrix is hollow, symmetric, and has positive integers in entries off the diagonal. We input this matrix into Ripser (see `https://live.ripser.org`) and asked for the H_0 and H_1 barcodes. The result of this experiment can be found in Figure 3(a).

Figure 2. Polyhedra from two sample sizes



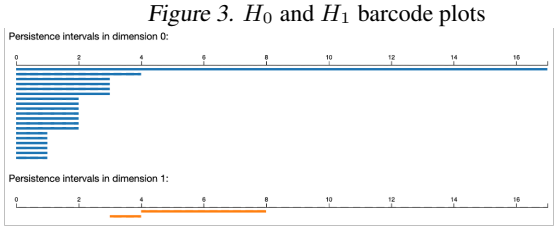
(a) Polyhedra from 20 points

(b) Polyhedra from 500 points

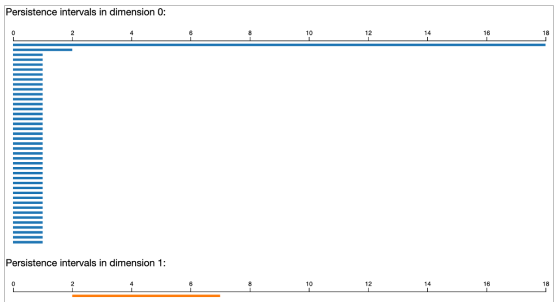
Example 4.2. We considered the same circle in $\mathbb{R}^3$ as in the previous example but sampled at 500 evenly spaced points $x^{(1)}, x^{(2)}, \ldots, x^{(500)}$. A picture of the polyhedra encountered by the 500 sample points is found in Figure 2(b). We recorded the 500 binary vectors for each of the polyhedra that was hit by a data point. Only 41 of the bit vectors were distinct (corresponding to 41 distinct polyhedra) and we labeled these $s^{(1)}, s^{(2)}, \ldots s^{(41)}$. We built a 41×41 matrix F by setting $F_{i,j}$ equal to the Hamming distance between $s^{(i)}$ and $s^{(j)}$. We input this matrix into Ripser (see `https://live.ripser.org`) and asked for the H_0 and H_1 barcodes. The result of this experiment can be found in Figure 3(b).

The H_0 barcode from the first example indicates connectivity at Hamming distance 4. The H_1 barcode indicates a spurious (i.e. short lived) closed loop occurring at Hamming distance 3. The homological signal of the circle appears (and is quite strong) at Hamming distance 4. The H_0 barcode from the second example indicates connectivity at

Figure 3. H_0 and H_1 barcode plots



(a) H_0, H_1 barcodes for Figure 2(a)



(b) H_0, H_1 barcodes for Figure 2(b)

Hamming distance 2. The H_1 barcode indicates a long lived loop beginning at Hamming distance 2.

In the following two examples, we will consider a similar example but utilizing real images from the ImageNet validation dataset (which contains 50K images). We will calculate the Hamming distance between bit vectors of data points via a much deeper neural network. The network we use is known as ResNet-50, it is a 50-layer convolutional neural network and was pre-trained on ImageNet (Deng et al., 2009). The training images are 224x224x3 thus the input space has dimension 150528. It uses ReLU as an activation function on the outputs from the convolutional layers and contains more than 6,000,000 nodes in its many layers. Like in the previous examples, the activation pattern of each ReLU layer is stored as a bit vector (as defined in Section 2.2).

Example 4.3. We started with 3 pictures, denoted by A_1, A_2, A_3, chosen from the ImageNet validation dataset. They are photos of a miniature poodle, a Persian cat, and a Saluki (see Figure 4). Each photo is represented by a 224 x 224 x 3 array. We generated 50 data points using the formulas $\sin\theta A_1 + \cos\theta A_2$ and $A_3 + \sin\theta A_1 + \cos\theta A_2$, respectively, with θ consisting of 50 points uniformly sampled from $[0, 2\pi]$. We calculated the bit vectors via ResNet-50 for the 50 data points and label them as $s^{(1)}, s^{(2)}, \ldots s^{(50)}$. We built a 50×50 distance matrix G by setting $G_{i,j}$ equal to the Hamming distance between $s^{(i)}$ and $s^{(j)}$. We input this matrix into Ripser which returned the H_0 and H_1 barcodes. The result of this experiment can be found in Figure 5.

We note that there is homological noise (or homological

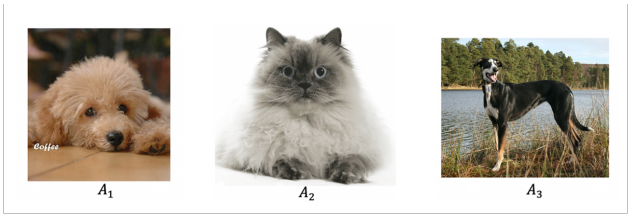


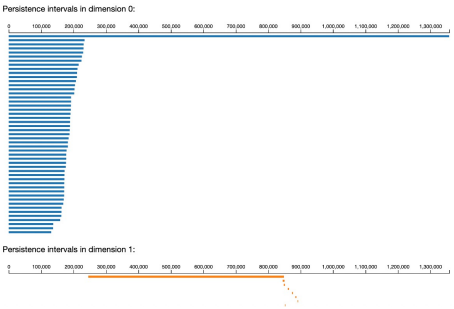
Figure 4. Images of $A_1, A_2,$ and A_3

dust) in Figure 5(a) but none in Figure 5(b). In both examples, there is a strong signal representing the circle. The sample points in the second example have all positive entries while the sample points in the first example do not have this property. We were unsure why (or if) this is related to the homological noise but we found it interesting nevertheless. It is a potentially useful feature that the input space to a neural network simultaneously has two kinds of distance measures. The first derives from the standard Euclidean geometry and the second derives from the coarse geometry implied by the Hamming distance. If one applies an isometry to a data set then its pairwise distance matrix will not change. However, if one applies an isometry with respect to one metric but measure distance via the second metric then one can definitely observe a change. It may be useful to combine information from multiple measurements. One way to carry this out is by using a monotone Boolean function (i.e Boolean functions built using only **and** and **or** operations). The next example illustrates this approach using a small rotation as the isometry in order to produce two distance matrices arising from essentially the same data set.

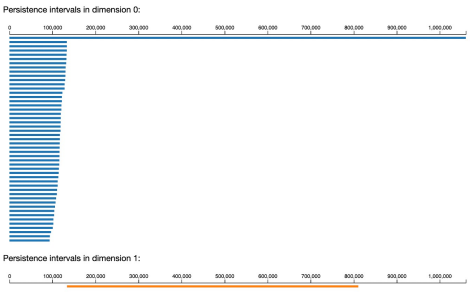
Example 4.4. Using the same A_1, A_2 with Example 4.3, we uniformly generated 50 points using $\sin\theta A_1 + \cos\theta A_2$ from $[0, 2\pi]$ and $[1, 2\pi + 1]$, respectively. We calculated the bit vectors via ResNet-50 for the 50 data points sampled from $[0, 2\pi]$ and labelled them as $s^{(1)}, s^{(2)}, \ldots s^{(50)}$. Similarly, we label the bit vectors of the 50 data points sampled from $[1, 2\pi + 1]$ as $\bar{s}^{(1)}, \bar{s}^{(2)}, \ldots \bar{s}^{(50)}$. We built four distance matrices G^1, G^2, G^3, and G^4 with $G^1_{i,j}$ equal to the Hamming distance between s^i and s^j, $G^2_{i,j}$ equal to the Hamming distance between $\bar{s}^i$ and $\bar{s}^j$, $G^3_{i,j}$ equal to the maximum of the Hamming distance between s^i and s^j and between $\bar{s}^i$ and $\bar{s}^j$, and $G^4_{i,j}$ equal to minimum of the Hamming distance between s^i and s^j and between $\bar{s}^i$ and $\bar{s}^j$. The H_1 barcodes for the four cases are presented in Figure 6

The distance matrix corresponding to the max function corresponds to requiring that distance matrices G_1 and G_2 both have their corresponding entry below some threshold. In other words, this corresponds to applying the **and** Boolean function. Similarly, the min function corresponds to applying the **or** Boolean function. Each seems to improve some feature related to the homological noise but it is hard to

Figure 5. H_0 and H_1 barcode plots for ResNet-50



(a) H_0, H_1 barcodes for $\sin\theta A_1 + \cos\theta A_2$



(b) H_0, H_1 barcodes for $A_3 + \sin\theta A_1 + \cos\theta A_2$

say which of the two is better. In the next example, we see an overall strengthening of the length of the homological signal for the max function and we see an earlier start of the homological signal for the min function. Homological noise did not make an appearance in the next example.

Example 4.5. Using the same A_1, A_2, A_3 with Example 4.3, we uniformly generated 50 points using $A_3 + \sin\theta A_1 + \cos\theta A_2$ from $[0, 2\pi]$ and $[1, 2\pi + 1]$, respectively. As in Example 4.4, we generated 4 different distance matrices $\hat{G}^1$, $\hat{G}^2$, $\hat{G}^3$, and $\hat{G}^4$, respectively. The H_1 barcodes for the four cases are presented in Figure 7.

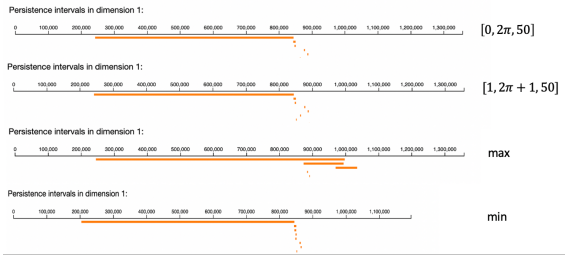


Figure 6. H_1 barcodes for G^1, G^2, G^3, and G^4

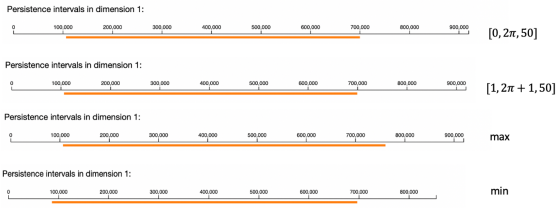


Figure 7. H_1 barcodes for $\hat{G}^1$, $\hat{G}^2$, $\hat{G}^3$, and $\hat{G}^4$

5. Conclusion and Future Work

A ReLU feedforward neural network induces a finite polyhedral decomposition of the input space. The corresponding dual graph represents this decomposition, where vertices correspond to polyhedra and edges represent shared facets. Data in the input space gets mapped to vertices of this graph. The geometry of the graph can be exploited, via persistent homology, to detect homological signals of manifolds from points sampled from the manifold. Many techniques in data analysis build on the premise that data, sharing a collection of common features, tends to be well approximated by a low dimensional geometric object. It is conjectured that the coarseness of the polyhedral decomposition can be helpful in dealing with the noise present in many data sets (maybe by including monotone Boolean functions to combine multiple distance matrices). In future work, we hope to extend the examples in this paper with the goal of detecting manifolds, known to exist in data, such as $SO(3)$, higher dimensional tori, and various fiber bundles involving these manifolds. A torus example can be found in Appendix C.

Neural networks of the kind considered in this paper have a finite number of "polyhedral resources". The training function, the distribution of training data, the network architecture, and the training method are some of the factors in how the network utilizes its polyhedra. In the networks we considered, we observed the general pattern that there was a dense collection of smaller volume polytopes near the training data, larger volume polytopes in the general vicinity but away from the training data, with a shell of unbounded polyhedra surrounding the polytopes. You can see hints of this tendency in Figure 1(a). In future work, we hope to make more precise characterizations in the direction of this observation. In particular, we are eager to extend this work to depictions that more accurately quantify the number of polytopes that emerge from a neural network and their distributions of volumes, the "polytopes landscape".

6. Acknowledgements

This work is partially supported by the United States Air Force under Contract No. FA865020C1121 and the DARPA Geometries of Learning Program under Award No. HR00112290074.

References

Arora, R., Basu, A., Mianjy, P., and Mukherjee, A. Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations*, pp. 1–17, 2018.

Balestriero, R. and Baraniuk, R. A spline theory of deep learning. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 374–383. PMLR, 2018.

Birdal, T., Lou, A., Guibas, L. J., and Simsekli, U. Intrinsic dimension, persistent homology and generalization in neural networks. *Advances in Neural Information Processing Systems*, 34:6776–6789, 2021.

Carlsson, G. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

Carrière, M., Chazal, F., Ike, Y., Lacombe, T., Royer, M., and Umeda, Y. Perslay: A neural network layer for persistence diagrams and new graph topological signatures. In *International Conference on Artificial Intelligence and Statistics*, pp. 2786–2796. PMLR, 2020.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Edelsbrunner, H. and Harer, J. Persistent homology-a survey. *Contemporary Mathematics*, 453:257–282, 2008.

Ghrist, R. Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

Grigsby, J. E., Lindsey, K., and Masden, M. Local and global topological complexity measures of relu neural network functions. *arXiv preprint arXiv:2204.06062*, 2022.

Hanin, B. and Rolnick, D. Deep relu networks have surprisingly few activation patterns. *Advances in neural information processing systems*, 32:361–370, 2019.

Hausmann, J.-C. On the vietoris-rips complexes and a cohomology theory for metric spaces. In *Prospects in Topology (Princeton, NJ, 1994)*, volume 138 of *Annals of Mathematics Studies*, pp. 175–188. Princeton University Press, Princeton, NJ, 1995.

Huchette, J., Muñoz, G., Serra, T., and Tsay, C. When Deep Learning Meets Polyhedral Theory: A Survey, 2023.

Jamil, H., Liu, Y., Caglar, T., Cole, C., Blanchard, N., Peterson, C., and Kirby, M. 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*, pp. 590–599, June 2023.

Masden, M. Algorithmic determination of the combinatorial structure of the linear regions of relu neural networks. *arXiv preprint arXiv:2207.07696*, 2022a.

Masden, M. Algorithmic determination of the combinatorial structure of the linear regions of relu neural networks. *arXiv preprint arXiv:2207.07696*, 2022b.

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 2: 2924–2932, 2014.

Pascanu, R., Montufar, G. F., and Bengio, Y. On the number of inference regions of deep feed forward networks with piece-wise linear activations. *Advances in Neural Information Processing Systems*, 34:1–9, December 2014.

Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Dickstein, J. S. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 2847–2854, August 2017.

Rieck, B., Togninalli, M., Bock, C., Moor, M., Horn, M., Gumbsch, T., and Borgwardt, K. Neural persistence: A complexity measure for deep neural networks using algebraic topology. In *International Conference on Learning Representations*, 2018.

Sattelberg, B., Cavalieri, R., Kirby, M., Peterson, C., and Beveridge, R. Locally linear attributes of relu neural networks. *arXiv preprint arXiv:2012.01940*, 2020.

Serra, T., Tjandraatmadja, C., and Ramalingam, S. Bounding and counting linear regions of deep neural networks. *arXiv preprint arXiv:1711.02114*, 2018.

Vincent, J. A. and Schwager, M. Reachable Polyhedral Marching (RPM): A Safety Verification Algorithm for Robotic Systems with Deep Neural Network Components. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9029–9035, Xi'an, China, May 2021. IEEE.

Xiang, W., Tran, H.-D., Rosenfeld, J. A., and Johnson, T. T. Reachable Set Estimation and Safety Verification for Piecewise Linear Systems with Neural Network Controllers. In *2018 Annual American Control Conference (ACC)*, pp. 1574–1579, June 2018.

Xu, S., Vaughan, J., Chen, J., Zhang, A., and Sudjianto, A. Traversing the Local Polytopes of ReLU Neural Networks: A Unified Approach for Network Verification. *arXiv:2111.08922*, 2021.

Yang, X., Tran, H.-D., Xiang, W., and Johnson, T. Reachability Analysis for Feed-Forward Neural Networks using Face Lattices. *arXiv:2003.01226*, March 2020.

Zhao, Q., Ye, Z., Chen, C., and Wang, Y. Persistence enhanced graph neural network. In *International Conference on Artificial Intelligence and Statistics*, pp. 2896–2906. PMLR, 2020.

Zomorodian, A. and Carlsson, G. Computing persistent homology. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 347–356, 2004.

A. Algorithms

Algorithm 1 Brute-Force Method

Require: A pretrained $(L+1)$-layer ReLU FFNN with weights and biases $\{W_i, b_i\}_{i=1}^{L}$ for all L hidden layers and the number of nodes in all hidden layers equal to h

Ensure: $\mathcal{P}$, the set of bit vectors that determine all polyhedra in the input space, and the size of $\mathcal{P}$, the number of polyhedra in the input space.

1) Initialize an empty set $\mathcal{P}$ to store all bit vectors that determine a polyhedron.

2) For $j = 1$, generate $A^{(j)}x \leq c^{(j)}$ based on $s^{(j)}$ and $\{W_i, b_i\}_{i=1}^{L}$ using (9).

3) Solve the linear program:

$$\begin{aligned} \text{minimize} \quad & 0 \\ \text{subject to} \quad & A^{(j)}x \leq c^{(j)}. \end{aligned} \tag{15}$$

If the optimal value is 0 (as opposed to 'infeasible' if using `scipy.optimize.linprog()` for example), then there is at least one feasible solution to (15), so the bit vector $s^{(j)}$ determines a polyhedron, $p^{(j)}$. If $p^{(j)}$ is full-dimensional, add $s^{(j)}$ to $\mathcal{P}$.

4) Repeat 2) and 3) for $j = 2, \ldots, 2^h$.

Algorithm 2 Traversal-and-Pruning Method

Require: A pretrained $(L+1)$-layer ReLU FFNN with weights and biases $\{W_i, b_i\}_{i=1}^{L}$ for all L hidden layers and a random point $x^{(0)}$ in the input space $\mathbb{R}^m$. Denote by $P(x^{(0)})$ the polytope to which $x^{(0)}$ belongs and by $s(x^{(0)})$ the specific bit string that uniquely identifies $P(x^{(0)})$.

Ensure: $\mathcal{P}$, the set of bit vectors that determine all polyhedra in the input space, and the size of $\mathcal{P}$, the number of polyhedra in the input space.

1) Initialize an empty set $\mathcal{P}$ to store all bit vectors that determine a polyhedron and define a label for each stored bit vector with 1 denoting if we have found all polyhedra adjacent to $P(x^{(0)})$ and 0 otherwise.

2) Calculate the bit vector $s(x^{(0)})$ using (3) and (5), add $s(x^{(0)})$ to the set $\mathcal{P}$, and give a label of 0 to $s(x^{(0)})$.

3) Traverse all the polyhedra adjacent to $P(x^{(0)})$, save their bit vectors to $\mathcal{P}$, and update these bit vectors' labels using the following process:

- Find the active bits of $s(x^{(0)})$ by solving the LP model (13) in (Vincent & Schwager, 2021). Without loss of generality, say there are q active bits.
- Flip each of these q active bits of $s(x^{(0)})$ (one at a time), producing q new bit strings $\{\hat{s}^{(j)} : 1 \leq j \leq q\} =: \hat{\mathcal{S}}$, which all differ from $s(x^{(0)})$ by only one bit. Add the elements of $\hat{\mathcal{S}}$ to $\mathcal{P}$ that are not already in $\mathcal{P}$.
- Label the added bit vectors with 0 and switch the label of $s(x^{(0)})$ from 0 to 1, indicating that we have found all of the neighbors of $P(x^{(0)})$ and have added their bit vectors to $\mathcal{P}$.

4) Repeat 3 for bit vectors in $\mathcal{P}$ with label 0 until all the bit vectors in $\mathcal{P}$ have label 1.

B. 2-D Polyhedral Decomposition Up Close

With Figure 8 and relevant discussion, we demonstrate qualities of the bit vectors that label the distinct regions of the polyhedral decomposition derived from model (11). We can see that the binary vectors produced by model (11) consist of 6 elements or bits, 3 from the first layer of the model and another 3 from the second layer. The collection of the lines of the polyhedral decomposition are defined by the weight matrices and bias vectors of the model.

It can be verified that neighboring polyhedra (polyhedra that share a 1-dimensional face) only differ in one bit. More specifically, they differ in the entry associated with the inequality that defines their shared edge. For example, looking at the top dictionary in Figure 8, we see that the binary vectors from polytopes 22 and 23 differ in their first bit. The first bit in the binary vector in the bottom dictionary is red, so the edge separating polytopes 22 and 23 is red.

Additionally, we clarify the claim that a polyhedral decomposition's dual graph (with vertices cor-

responding to the polyhedra and edges connecting vertices if their corresponding polyhedra share an edge) is bipartite. In our 2-dimensional example, we have two vertex sets $\mathcal{V}_1$ and $\mathcal{V}_2$ where are $\mathcal{V}_1 = \{2, 4, 6, 7, 8, 9, 10, 12, 15, 17, 19, 22, 24, 26,$ and the unlabeled center triangle$\}$ and $\mathcal{V}_2 = \{1, 3, 5, 11, 13, 14, 16, 18, 20, 21, 23,$ and $25\}$ because no two polyhedra in the same set are neighbors.

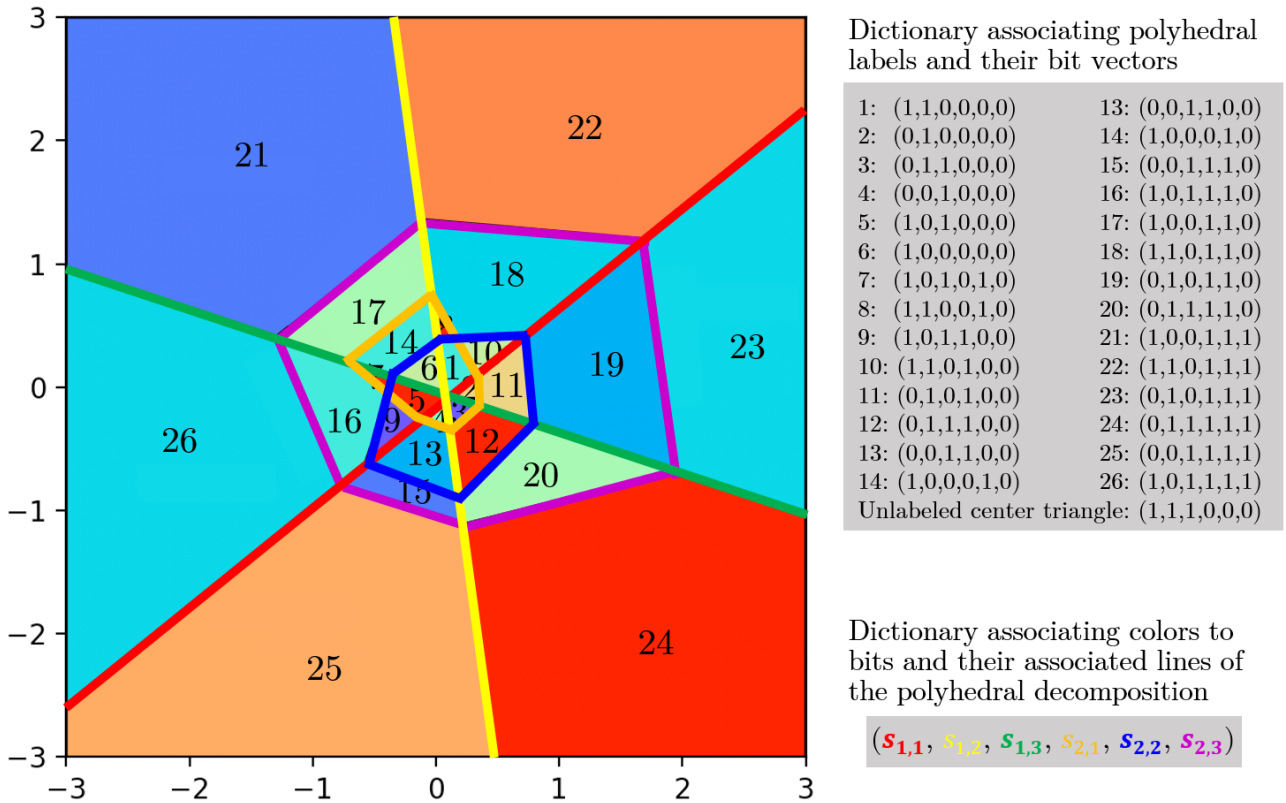


Figure 8. The 2-dimensional polyhedral decomposition with a look-up table giving the binary vectors for each polytope and color-coding that associates each bit to lines in the polyhedral decomposition using the notation introduced in (4).

C. Example

Example C.1. In this example, we started with 5 randomly generated orthogonal tensors of size $224 \times 224 \times 3$ (denoted $A_1, \ldots, A_5$). We made 100 data points generated using the formula $A_5 + \alpha * (\sin\theta_1 A_1 + \cos\theta_1 A_2 + \sin\theta_2 A_3 + \cos\theta_2 A_4)$, for θ_1, θ_2 being 100 points uniformly sampled from $[0, 2\pi] \times [0, 2\pi]$. We input this matrix into Ripser and asked for the H_1 and H_2 barcodes. The result of this experiment can be found in Figure 9.

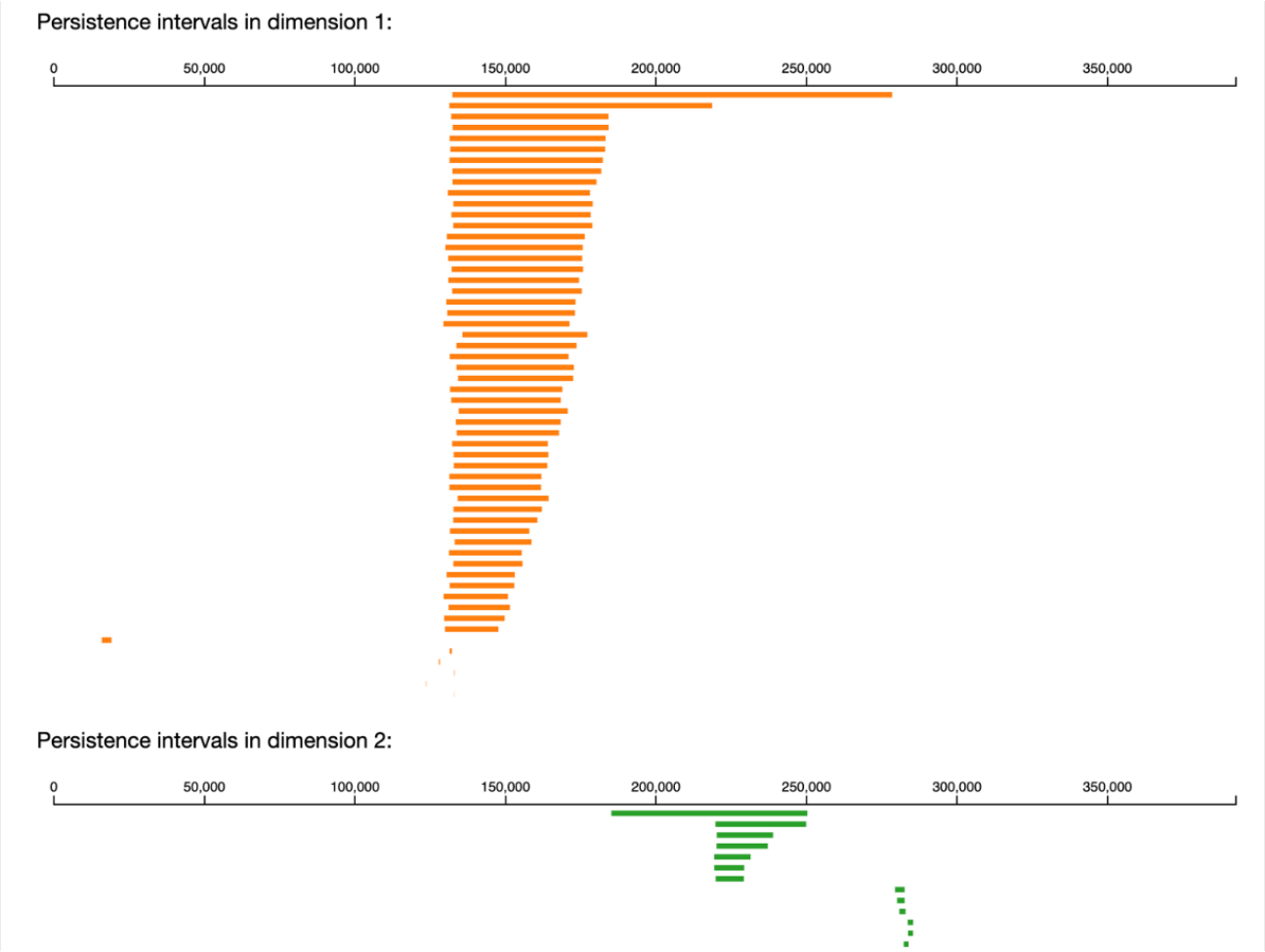


Figure 9. H_1 and H_2 barcode plots of randomly generated orthogonal tensors.