Defining Neural Network Architecture through Polytope Structures of Datasets

Sangmin Lee [1] **Abbas Mammadov** [1 2] **Jong Chul Ye** [1 3]

Abstract

Current theoretical and empirical research in neural networks suggests that complex datasets require large network architectures for thorough classification, yet the precise nature of this relationship remains unclear. This paper tackles this issue by defining upper and lower bounds for neural network widths, which are informed by the polytope structure of the dataset in question. We also delve into the application of these principles to simplicial complexes and specific manifold shapes, explaining how the requirement for network width varies in accordance with the geometric complexity of the dataset. Moreover, we develop an algorithm to investigate a converse situation where the polytope structure of a dataset can be inferred from its corresponding trained neural networks. Through our algorithm, it is established that popular datasets such as MNIST, Fashion-MNIST, and CIFAR10 can be efficiently encapsulated using no more than two polytopes with a small number of faces.

1. Introduction

To comprehend the remarkable performance of deep neural networks (DNNs), extensive research has delved into their architectures and the universal approximation property (UAP). The UAP of two-layer neural networks on compact sets was initially proven by Cybenko (1989), sparking widespread exploration of the UAP in diverse settings for DNNs. Studies have focused on determining the minimal depths and widths of deep ReLU networks required for UAP (Hornik, 1991; Park et al., 2020). These foundational results contribute to unraveling the intricate relationship between approximation power and neural network architectures.

However, a converse problem to address the influence of the characteristics of training datasets necessary to attain the UAP in neural networks has received relatively less attention. For example, when analyzing the swiss roll dataset shown in Figure 1(a), an important practical question emerges: *What are the effective depth and width required for the complete classification of this dataset?* Despite the practical relevance of this inquiry in the context of training neural networks, existing theoretical results only offer basic lower bounds (minimum depth of 2 (Hornik, 1991) and a width of $\max\{d_x + 1, d_y\}$ (Park et al., 2020)), which are often impractical for real applications. While a range of empirical evidence indicates that increasing the depth or width of networks could lead to successful outcomes, there remains an absence of theoretical assurances to foresee these results.

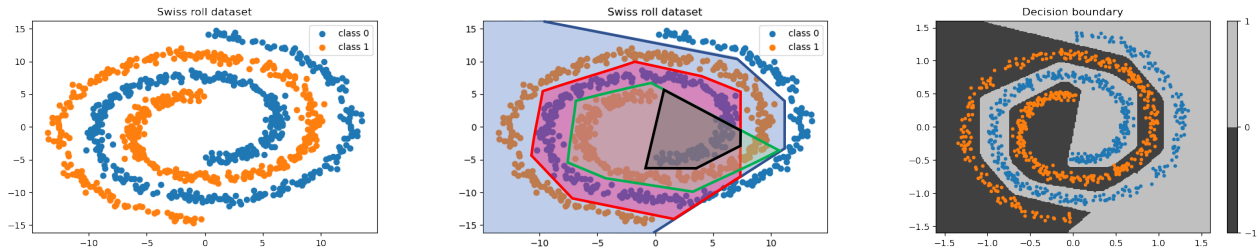
In this paper, we therefore tackle the challenge of identifying the optimal neural network architecture for classifying a given dataset. This task is approached through the lens of the polytope structure of deep ReLU networks, a subject that has garnered considerable attention in recent studies (Black et al., 2022; Grigsby & Lindsey, 2022; Berzins, 2023; Huchette et al., 2023). In fact, our primary theoretical goal is to address the "multiple manifold problem," introduced by Buchanan et al. (2020): *For given two disjoint topological spaces $\mathcal{X}_+$ and $\mathcal{X}_-$, what is the optimal architecture for the neural network $\mathcal{N}$ such that $\mathcal{N}(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \mathcal{X}_+$ and $\mathcal{N}(\boldsymbol{x}) < 0$ otherwise?*

By utilizing the geometric properties of DNNs, here we provide a comprehensive answer to this question. Our approach involves determining both upper and lower bounds for the depth and widths of networks required for dataset classification, based on the polytope covering of the datasets. Specifically, we explicitly construct a neural network with practical applicability. For example, our discovery in Theorem 3.4 reveals that the swiss roll dataset in Figure 1(a) can be efficiently classified using a three-layer ReLU network with 24 neurons, as depicted in Figure 1(c).

Another important contribution of this paper is the investigation into the converse situation, demonstrating that trained neural networks inherently capture the geometric properties of the dataset and enable the extraction of the dataset's polytope structure. As for demonstrating practical use, we uncover and discuss simple geometric traits of real-world

[1]Department of Mathematical Science, KAIST, Daejeon, Korea [2]School of Computing, KAIST, Daejeon, Korea [3]Kim Jaechul Graduate School of AI, KAIST, Daejeon, Korea. Correspondence to: Jong Chul Ye <jong.ye@kaist.ac.kr>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).



(a) (b) (c)

Figure 1. What type of neural network architecture is capable of effectively classifying the swiss roll dataset depicted in (a)? By establishing a collection of covering polytopes to enclose one class, as illustrated in (b), our result demonstrates that a three-layer ReLU network with the architecture $2 \xrightarrow{\sigma} 20 \xrightarrow{\sigma} 4 \to 1$ can successfully achieve this classification task, as exemplified in (c).

datasets such as MNIST, Fashion-MNIST, and CIFAR10, achieved through the training of neural networks.

Importantly, our contributions can be summarized as follows:

- **Explicit construction of networks.** We introduce the novel concept of a *polytope-basis cover* (Definition 3.2), which serves to describe the geometric structure of the dataset in detail. Building on this, we propose the design of a three-layer ReLU network, specifically tailored to efficiently classify the dataset in question, using its polytope-basis cover as a guiding framework (Theorem 3.4).

- **Bounds on network widths.** We define both upper and lower bounds for the width of a neural network necessary to classify a given convex polytope region, taking into account the number of its faces (Proposition 3.1). Furthermore, we derive upper bounds on network widths when the dataset $\mathcal{X}$ is structured as a simplicial complex or can be covered by a difference of prismatic polytopes (Theorem 3.5 and 3.6). These bounds are correlated with the number of facets or the Betti numbers of $\mathcal{X}$, demonstrating an interplay between the dataset's inherent geometry and the required network architecture.

- **Investigating dataset geometry.** Building on our findings, we demonstrate that it is possible to investigate the geometric features of the dataset by training a neural network (Theorem 3.7). Specifically, we develop algorithms that are able to identify a polytope basis-cover for given datasets (Algorithm 1). Our results show that each class within the MNIST, Fashion-MNIST, and CIFAR10 datasets can be effectively distinguished using no more than two convex polytopes, each consisting of fewer than 30 faces (Table 1).

2. Preliminaries

Notation. In this paper, we focus primarily on the binary classification problem, aiming to separate two disjoint topological spaces denoted as $\mathcal{X}_+$ and $\mathcal{X}_-$, or two classes of finite points denoted as $\mathcal{D}_+$ and $\mathcal{D}_-$. The training dataset is represented as $\mathcal{D} = \mathcal{D}_+ \cup \mathcal{D}_- = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$. Throughout the paper, we denote scalars by lowercase letters and vectors by boldface lowercase letters. For a positive integer m, $[m]$ represents the set $\{1, 2, \cdots, m\}$. The ReLU activation function is denoted by $\sigma(x) := \mathrm{ReLU}(x) = \max\{0, x\}$, and it is applied to a vector coordinate-wisely. The sigmoid activation function is denoted as $\mathtt{SIG}(x) = \frac{1}{1+e^{-x}}$. The max pooling operation is represented as $\mathtt{MAX} : \mathbb{R}^d \to \mathbb{R}$, which returns the maximum component of the input vector. The ε neighborhood of a topological space $\mathcal{X} \subset \mathbb{R}^d$ is defined by $\mathcal{B}_\varepsilon(\mathcal{X}) := \{\boldsymbol{x} \in \mathbb{R}^d : \min_{\boldsymbol{y} \in \mathcal{X}} \|\boldsymbol{x} - \boldsymbol{y}\|_2 < \varepsilon\}$. The indicator function is denoted by

$$\mathbb{1}_{\{c\}} := \begin{cases} 1, & \text{if } c \text{ is true,} \\ 0, & \text{otherwise.} \end{cases}$$

Additionally, we define a *convex polytope* as an intersection of hyperspaces, as defined below:

Definition 2.1. A nonempty set $C \subset \mathbb{R}^d$ is called a *convex polytope with m faces* if there exist $\boldsymbol{w}_k \in \mathbb{R}^d$ and $b_k \in \mathbb{R}$ for $k \in [m]$ such that $C = \bigcap_{k=1}^{m} \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{w}_k^\top \boldsymbol{x} + b_k \leq 0\}$.

Network architectures. In this paper, the terminology *architecture* refers to the structure of a neural network, which means the depth and the width of hidden layers, and is often denoted by $\mathcal{A}$. A D-layer neural network $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}$ with hidden layer widths $d_1, d_2, \cdots, d_{D-1}$ and activation functions $\mathtt{ACT}_1, \mathtt{ACT}_2, \cdots, \mathtt{ACT}_D$ is represented by $d \xrightarrow{\mathtt{ACT}_1} d_1 \xrightarrow{\mathtt{ACT}_2} d_2 \xrightarrow{\mathtt{ACT}_3} \cdots \xrightarrow{\mathtt{ACT}_{D-1}} d_{D-1} \xrightarrow{\mathtt{ACT}_D} 1$. When the activation function is the identity, we add nothing on the arrow. For example, $d \xrightarrow{\sigma} m \to 1$ denotes a two-layer

ReLU network with m neurons, presented by

$$\mathcal{N}(\boldsymbol{x}) = v_0 + \sum_{k=1}^{m} v_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k). \tag{1}$$

Definition 2.2. Let $\mathcal{X} := \mathcal{X}_+ \cup \mathcal{X}_- \subset \mathbb{R}^d$ be a union of two disjoint topological spaces. A neural network architecture $\mathcal{A}$ is called a *feasible architecture on* $\mathcal{X}$ if there exists a neural network with the architecture $\mathcal{A}$ such that

$$\begin{aligned} \mathcal{N}(\boldsymbol{x}) > 0 \quad &\text{if} \quad \boldsymbol{x} \in \mathcal{X}_+, \\ \mathcal{N}(\boldsymbol{x}) < 0 \quad &\text{if} \quad \boldsymbol{x} \in \mathcal{X}_-. \end{aligned}$$

In other words, *feasible architecture on* $\mathcal{X}$ refers to a network architecture capable of fully discriminating between the two specified manifolds, $\mathcal{X}_+$ and $\mathcal{X}_-$. This paper aims to explore the connection between feasible architectures and the geometrical properties of the dataset.

3. Main Contributions

In this section, we present our main findings in two forms. Firstly, we establish the upper and lower limits of network width required for classifying a specific dataset. Secondly, we illustrate how trained neural networks inherently capture the geometric characteristics of the dataset they handle.

3.1. Data Geometry-Dependent Bounds on Widths

Let $C \subset \mathbb{R}^d$ be a convex polytope with m faces. Our objective is to establish bounds on the widths of a ReLU neural network necessary for it to be feasible architecture on C. Applying piecewise linearity of ReLU networks and the volume formula of convex polytopes, the following proposition provides the answer.

Proposition 3.1. *Let $C \subset \mathbb{R}^d$ be a convex polytope enclosed by m hyperplanes, and consider $\mathcal{X} = \mathcal{X}_+ \cup \mathcal{X}_-$ where $\mathcal{X}_+ := C, \mathcal{X}_- := \mathcal{B}_\varepsilon(\mathcal{X}_+)^c$. Then, $d \xrightarrow{\sigma} m \to 1$ is a feasible architecture on $\mathcal{X}$ with minimal depth. Conversely, if $d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} d_2 \xrightarrow{\sigma} \cdots \xrightarrow{\sigma} d_k \to 1$ is a feasible architecture on $\mathcal{X}$, then*

$$d_1 \cdot \prod_{j=2}^{k} (2d_j + 1) \geq \begin{cases} \left\lceil \frac{m}{2} \right\rceil + (d-2), & \text{if} \quad m \geq 2d+1, \\ 2d-1, & \text{if} \quad m = 2d-1, 2d, \\ d+1, & \text{if} \quad m < 2d-1. \end{cases}$$

This lower bound is optimal when $k = 1$ and $d = 2$ (i.e., two-layer network on $\mathbb{R}^2$).

Proof sketch. We briefly introduce the main idea here. Let $A_1, \cdots, A_m$ be faces of C, and $\boldsymbol{x}$ be a point in C. Since C is convex, it can be decomposed to m pyramids whose common apex is $\boldsymbol{x}$ (see Figure 2(a)). Then, the volume (in Lebesgue sense) of C is equal to the sum of the volume of m pyramids. Mathematically, it is represented by

$$\mathrm{Vol}_d(C) = \frac{1}{d} \sum_{k=1}^{m} \mathrm{Vol}_{d-1}(A_k) \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)$$

where $\boldsymbol{w}_i$ is a unit vector of the hyperplane A_i, and Vol_d denotes the d-dimensional volume. From this equation, we define a two-layer ReLU network

$$\mathcal{N}(\boldsymbol{x}) := 1 + M(\mathrm{Vol}_d(C) - \frac{1}{d} \sum_{k=1}^{m} \mathrm{Vol}_{d-1}(A_k) \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k))$$

for some constant M. Then $\mathcal{N}(\boldsymbol{x}) = 1$ for all $\boldsymbol{x} \in C$, and we can prove that $d \xrightarrow{\sigma} m \to 1$ is a feasible architecture on the polytope, by adjusting the value of M. The detailed proof can be found in Appendix E.1. □

In the proof of Proposition 3.1, $\sigma(\mathcal{N})$ is a two-layer ReLU network that approximates the indicator function on a convex polytope.[1] Building upon this proposition, we are interested in extending our findings to arbitrary topological spaces, specifically those that can be distinguished by a collection of polytopes. To facilitate this extension, we introduce an additional terminology.

Definition 3.2. Let $\mathcal{X} := \mathcal{X}_+ \cup \mathcal{X}_- \subset \mathbb{R}^d$ be a union of two disjoint topological spaces. A finite collection of polytopes $\mathcal{C} := \{P_1, \cdots, P_{n_P}, Q_1, \cdots, Q_{n_Q}\}$ is called a *polytope-basis cover of* $\mathcal{X}$ if it satisfies

$$\begin{aligned} \sum_{k=1}^{n_P} \mathbb{1}_{\{\boldsymbol{x} \in P_k\}} &> \sum_{k=1}^{n_Q} \mathbb{1}_{\{\boldsymbol{x} \in Q_k\}} \qquad \text{for all} \quad \boldsymbol{x} \in \mathcal{X}_+, \\ \sum_{k=1}^{n_P} \mathbb{1}_{\{\boldsymbol{x} \in P_k\}} &\leq \sum_{k=1}^{n_Q} \mathbb{1}_{\{\boldsymbol{x} \in Q_k\}} \qquad \text{for all} \quad \boldsymbol{x} \in \mathcal{X}_-. \end{aligned}$$

Roughly speaking, a polytope-basis cover of $\mathcal{X}$ is a polytope covering of $\mathcal{X}_+$ and $\mathcal{X}_-$ that admits overlapping, where the difference number of overlapped covers is restricted to be positive or negative with respect to the label. Below, we provide an example of a polytope-basis cover for the swiss roll dataset described in Figure 1(a).

Example 3.3. Let $\mathcal{X}_+$ and $\mathcal{X}_-$ be the orange and blue classes in Figure 1(a), respectively. Figure 2(b) demonstrates a polytope-basis cover of $\mathcal{X}$ consists of four convex polytopes: P_1, P_2, Q_1, Q_2. It is easily checked that $\sum_{k=1}^{2} \mathbb{1}_{\{\boldsymbol{x} \in P_k\}} - \sum_{k=1}^{2} \mathbb{1}_{\{\boldsymbol{x} \in Q_k\}} = 1 > 0$ for $\forall \boldsymbol{x} \in \mathcal{X}_+$, while $\sum_{k=1}^{2} \mathbb{1}_{\{\boldsymbol{x} \in P_k\}} - \sum_{k=1}^{2} \mathbb{1}_{\{\boldsymbol{x} \in Q_k\}} = 0$ for $\forall \boldsymbol{x} \in \mathcal{X}_-$.

The usefulness of polytope-basis covers appears in the following theorem: we can derive an upper bound of feasible architecture on $\mathcal{X}$ from its polytope-basis cover, by applying the constructive proof used in Proposition 3.1.

[1] We also mention that the approximation of indicator functions directly induces UAP of neural networks (Theorem F.3).

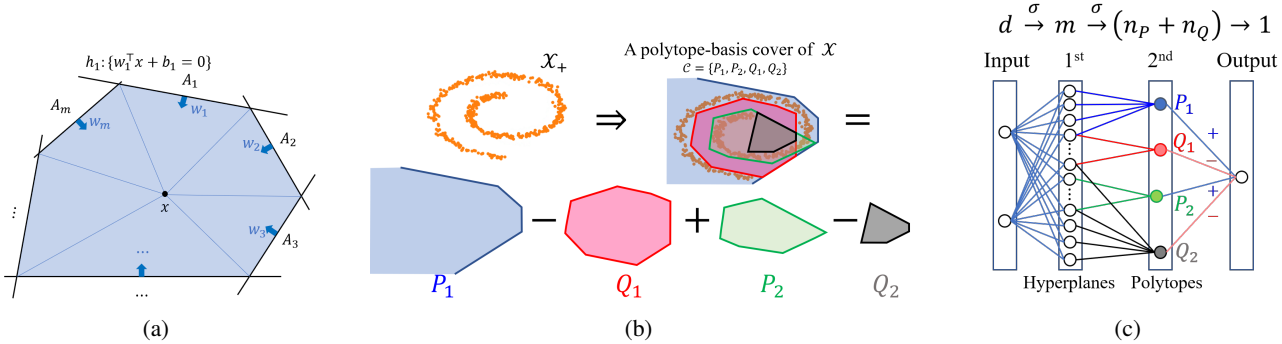


Figure 2. The fundamental ideas in our work. (a) A convex polytope enclosed by m hyperplanes can be decomposed by m small pyramids. (b) For the topological space $\mathcal{X}_+$, the collection of polytopes $\mathcal{C} = \{P_1, P_2, Q_1, Q_2\}$ forms a polytope-basis cover of $\mathcal{X}$. (c) The constructive proof in Theorem 3.4 further exhibits the role of neurons in hidden layers: the width of first layer means the number of total faces in $\mathcal{C}$, and the neurons in the second layer corresponds to the polytopes in $\mathcal{C}$.

Theorem 3.4. *For a given topological space $\mathcal{X} = \mathcal{X}_+ \cup \mathcal{X}_- \subset \mathbb{R}^d$, let $\mathcal{C} = \{P_1, \cdots, P_{n_P}, Q_1, \cdots, Q_{n_Q}\}$ be a polytope-basis cover of $\mathcal{X}$. Let m denote the total number of faces of the convex polytopes in $\mathcal{C}$. Then,*

$$d \xrightarrow{\sigma} m \xrightarrow{\sigma} (n_P + n_Q) \to 1$$

is a feasible architecture on $\mathcal{X}$.

The proof can be found in Appendix E.2. One of the important contributions of Theorem 3.4 is that its construction exhibits the exact role of each neuron in the hidden layers. It demonstrates that for a given polytope-baiss cover, a three-layer ReLU network with widths #(hyperplanes) and #(polytopes) in the first and second hidden layer, respectively, is a feasible architecture. For instance, we recall the polytope-basis cover represented in Figure 2(b). Each neuron in the first hidden layer represents a hyperplane in the input space, where each neuron in the second hidden layer represents a convex polytope (P_i or Q_j) in $\mathcal{C}$ that is formed by connected neurons in the first layer as depicted in Figure 2(c).

Building upon Theorem 3.4, we can further explore the relationship between the topological properties of a dataset and the maximum width achievable by feasible network architectures. Specifically, we concentrate on simplicial complexes and Betti numbers, which are fundamental tools for investigating the topological structure of point cloud datasets in topological data analysis (TDA).

A simplicial j-complex is a specific type of simplicial complex where the highest-dimensional simplex has dimension j. Within a given simplicial complex K, a facet is a simplex with the highest dimension that is not a face (subset) of any larger simplex (Magai & Ayzenberg, 2022). With these definitions established, we proceed by proposing a feasible network architecture and deriving upper bounds on its width when one class $\mathcal{X}_+$ forms a simplicial complex.

Theorem 3.5. *Let $\mathcal{X} = \mathcal{X}_+ \cup \mathcal{X}_-$ be a union of two disjoint topological spaces, where $\mathcal{X}_+ \subset \mathbb{R}^d$ is a simplicial J-complex consists of k facets. Let k_j be the number of j-dimensional facets of $\mathcal{X}_+$ for $j = 1, \cdots, J$. Then, $d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} k \to 1$ is a feasible architecture on $\mathcal{X}$, where d_1 is bounded by*

$$d_1 \leq \min \left\{ k(d+1) - (d-1) \left\lfloor \sum_{j=0}^{\lfloor \frac{d-1}{2} \rfloor} \frac{k_j}{2} \right\rfloor, \right.$$
$$\left. (d+1) \left\lfloor \sum_{j \leq \frac{d}{2}} \left(k_j \frac{j+2}{d-j} + \frac{j+2}{j+1} \right) + \sum_{j > \frac{d}{2}} k_j \right\rfloor \right\}. \quad (2)$$

The proof can be found in Appendix E.3. Theorem 3.5 reveals that the width d_1 is bounded by in terms of the number of facets k of the provided simplicial complex. From a geometric perspective, it is generally intuitive that a smaller number of facets suggests a simpler structure of the simplicial complex. This notion is mathematically expressed in (2), which suggests that the first value in (2) results in $d_1 \lesssim \frac{k}{2}(d+3)$, which magnifies as k increases. Similarly, when the maximal dimension J is smaller than $\frac{d}{2}$ and k is fixed, the summation in the second term in (2) reduces to $d_1 \lesssim (d+1)\left(k\frac{J+2}{d-J} + 2\right)$, which rapidly diminishes as J decreases. This analysis demonstrates that a smaller dimension J demands smaller widths, which aligns with the intuition that *the lower-dimensional manifold could be approximated with the smaller number of neurons.*

Now, we demonstrate how the result in Theorem 3.4 can be further leveraged to ascertain a neural network architecture with width bounds defined in terms of the Betti numbers. The Betti number is a key metric used in TDA to denote the number of k-dimensional 'holes' in a data distribution, which are frequently employed to study the topological characteristics of topological spaces (Naitzat et al., 2020).

Recall that Theorem 3.4 offers an upper bound on widths when $\mathcal{X}$ can be depicted as a difference between two groups of convex sets. Expanding on this, assuming the polytope-basis cover consists of prismatic polytopes[2], we can derive a bound for network architecture in relation to its Betti numbers. The result is concretely explained in the following theorem.

Theorem 3.6. *Let $\mathcal{X} = \mathcal{X}_+ \cup \mathcal{X}_-$ be a union of two disjoint topological spaces, where $\mathcal{X}_-$ can be separated from $\mathcal{X}_+$ by disjoint bounded prismatic polytopes having at most m faces. Let β_k be the k-th Betti number of the polytope-basis cover. Then, the following three-layer architecture*

$$d \xrightarrow{\sigma} \left(m + 2(\beta_0 - 1) + \sum_{k=1}^{d} (m - 2(d-k-1))\, \beta_k \right) \xrightarrow{\sigma} \left(\sum_{k=0}^{d} \beta_k \right) \to 1 \tag{3}$$

is a feasible architecture on $\mathcal{X}$. Conversely, for any such $\mathcal{X}$, suppose $d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} d_2 \xrightarrow{\sigma} \cdots \xrightarrow{\sigma} d_D \to 1$ is a feasible architecture on $\mathcal{X}$. Then, the network widths must satisfy

$$\sum_{i=1}^{D} \prod_{j=i}^{D} d_j \geq 2 \sum_{k=0}^{d} \beta_k - 2. \tag{4}$$

The proof is provided in Appendix E.4. One of the important implications of Theorem 3.6 is the upper and lower bounds on network widths in terms of the Betti numbers of $\mathcal{X}$, which reveals the interplay between the topological characteristics of the dataset and network architectures. In Appendix, we also show in Proposition F.2 that topological property alone cannot determine the feasible architecture, highlighting the significance of prismatic polytopes assumption in Theorem 3.6. In other words, the result in Proposition F.2 implies that the geometrical assumptions in this theorem are indispensable to connecting topological features with bounds on the network widths.

Interestingly, the sum of Betti numbers $\sum_{k=0}^{d} \beta_k$ which appears in the third layer in (3), is often called the *topological complexity* of $\mathcal{X}$. This quantity is recognized as a measure of the complexity of a given topological space in some previous works (Bianchini & Scarselli, 2014; Naitzat et al., 2020), and can be bounded by Morse theory (Milnor et al., 1963) or Gromov's Betti number Theorem (Gromov, 1981).

Furthermore, the lower bound on widths (4) shows that the sum of product of widths should be greater than the sum of Betti numbers. This finding also confirms the increased significance of widths in deeper layers as compared to earlier ones, highlighting the advantageous impact of depth in network architecture. It also verifies that the contribution of the width in deeper layers holds greater significance compared to previous layers, i.e., the positive effect of depth.

[2]For the formal definition of prismatic polytopes, see Appendix E.4

3.2. Polytope-Basis Cover Search Algorithm

So far, we have demonstrated how feasible architecture can be determined from the geometric characteristics of a topological space $\mathcal{X}$, in terms of its polytope-basis cover. In this section, we delve into the converse scenario: given a trained neural network on the dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$, can we obtain a polytope-basis cover of $\mathcal{D}$? We tackle this question by leveraging the convexity of two-layer ReLU networks, which has been studied in a few previous works (Amos et al., 2017; Sivaprasad et al., 2021; Balestriero et al., 2022) (see Appendix A for related works). Our focus also extends to the precise computation of the number of faces.

Theorem 3.7. *Let $\mathcal{T}_j$ and $\mathcal{N}$ be two-layer and three-layer ReLU networks defined by*

$$\mathcal{T}_j(\boldsymbol{x}) := \lambda + \sum_{k=1}^{m_j} v_{jk} \sigma(\boldsymbol{w}_{jk}^\top \boldsymbol{x} + b_{jk}), \quad \forall v_{jk} < 0, \tag{5}$$

$$\mathcal{N}(\boldsymbol{x}) := -\frac{1}{2}\lambda + \sum_{j=1}^{J} a_j \sigma(\mathcal{T}_j(\boldsymbol{x})), \quad \forall a_j \in \{\pm 1\} \tag{6}$$

for a positive constant λ. For a given dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$, suppose $\mathcal{N}$ satisfies

$$\sigma(\mathcal{T}_j(\boldsymbol{x}_i)) = 0 \ \text{ or } \ \lambda, \quad \forall \boldsymbol{x}_i \in \mathcal{D},\ \forall j \in [J]. \tag{7}$$

Then, the collection of polytopes $\{C_j\}_{j \in [J]}$, defined by $C_j := \{\boldsymbol{x} \in \mathbb{R}^d \mid \mathcal{T}_j(\boldsymbol{x}) = \lambda\}$, becomes a polytope-basis cover of $\mathcal{D}$ whose accuracy is same with $\mathcal{N}$.

The proof of this theorem can be found in Appendix E.5. The constant λ in (5) and (7) is a positive scalar value determined from the ratio of labels in the dataset. In experiments, we practically adopted $\lambda = 5$. See Appendix C.3 for further details.

This theorem establishes that if a trained three-layer network $\mathcal{N}$ defined in (6) satisfies the condition outlined in (7), then we can derive a polytope-basis cover of the training dataset $\mathcal{D}$ from $\mathcal{N}$. Below, we outline two strategies we employed to satisfy both (5) and (6) conditions.

Firstly, to meet (5), we utilize the implicit bias of gradient descent established by Du et al. (2018, Theorem 2.1), stated in Proposition F.5. Specifically, we initialize the network weights to satisfy

$$v_{jk} < -\sqrt{\|\boldsymbol{w}_{jk}\|^2 + b_{jk}^2} \quad \forall j \in [J], k \in [m]. \tag{8}$$

Then, the implicit bias preserves the inequality (8), thus ensures $v_{jk} < 0$ for all $j \in [J]$ and $k \in [m]$ on the gradient flow. This satisfies the first condition.

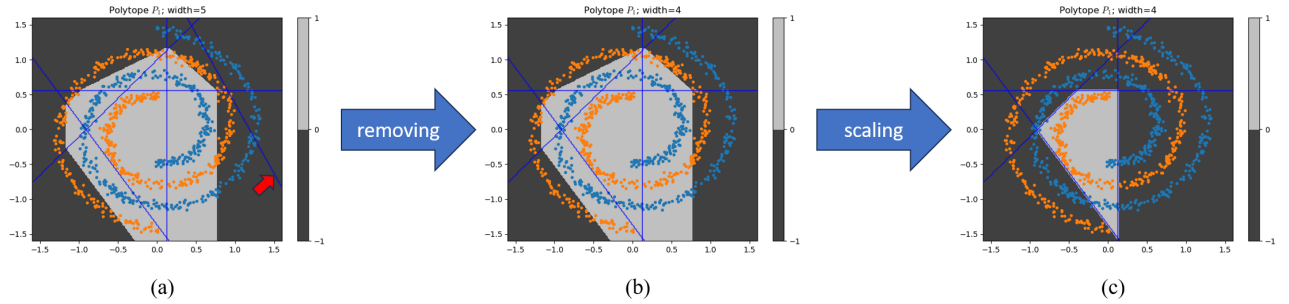


Figure 3. Visualization of Algorithm 1. For a given two-layer network $\mathcal{T}$ defined by (5), it strategically removes and scales specific neurons of $\mathcal{T}$ to encapsulate the characteristics of a single convex polytope. In essence, the algorithm compresses the network to reveal the minimal representation of a polytope structure.

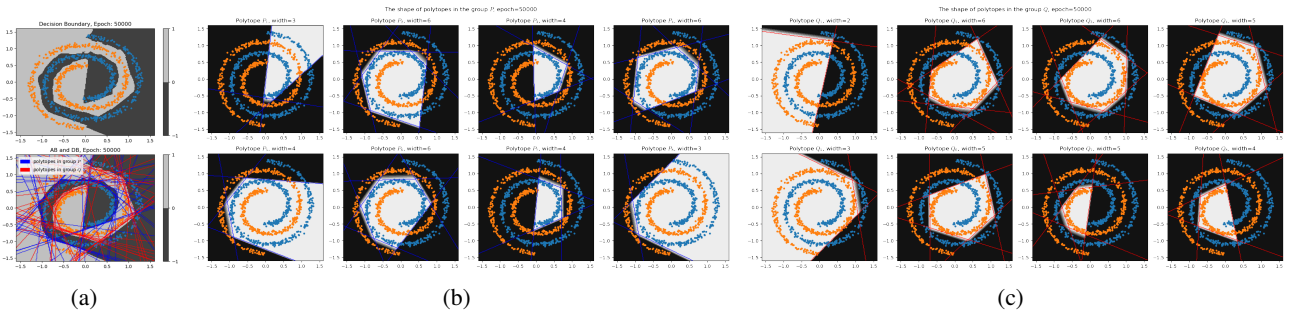


Figure 4. A polytope-basis cover derived from a trained three-layer ReLU network defined in (6), obtained from Algorithm 1. The decision boundary and activation boundaries[4] of the trained network are depicted in (a). Each polytope corresponding to $a_j = +1$ and $a_j = -1$ is illustrated in (b) and (c), respectively. The result constitutes a polytope-basis cover of the swiss roll dataset.

Secondly, to achieve (7), we introduce a novel approach named the "compressing algorithm." This algorithm aims to 'compress' a given two-layer network $\mathcal{T}_j$ defined in (5) to identify a minimal convex polytope representation. The process is precisely outlined in Algorithm 1.

More precisely, the algorithm operates in two main steps: 1) identifying and removing a neuron that do not significantly influence the decision boundary, and 2) amplifying the weights to delineate the faces of the decision boundary polytope. We illustrate the functionality of the algorithm in Figure 3. In Figure 3(a), the red arrow highlights a neuron identified as non-essential for the decision boundary. This neuron is subsequently removed as shown in (b). Subsequently, by scaling the weights $(v_k, \boldsymbol{w}_k, b_k)$ by a factor $\lambda_{scale} > 1$, the decision boundary shrinks into a convex polytope with a number of faces equal to the width of $\mathcal{T}$ (depicted in Figure 3(c)).

Note that a single execution of Algorithm 1 may not immediately yield the network satisfying (7). However, we prove in Proposition C.2 that by iterating this algorithm a sufficient number of times, the output of the algorithm always satisfies both (5) and (7), making it suitable for the application of Theorem 3.7. Therefore, we apply the algorithm once every thousand iterations during the gradient descent optimization process, and the end of whole training. It is precisely described in Algorithm 2 in Appendix C.1. We additionally mention that Algorithm 1 is compatible with non-pretrained networks, although this flexibility may come at the cost of increased training time.

Consequently, we present a polytope-basis cover of the swiss roll dataset derived from a trained three-layer network in Figure 4. The polytopes comprising the resulting cover are visualized in Figure 4 (b) and (c). This outcome demonstrates the polytope-basis cover inherent in the trained network can be identified through Algorithm 1. It is worth noting that the decision boundary and activation boundary[4] displayed in Figure 4(a) is combination of several polytopes.

Lastly, we mention that we further propose two alternative methods for obtaining a polytope-basis cover in Appendix C. Specifically, one approach involves deriving such a cover from any trained two-layer network (Algorithm 3), while the other method entails training several neural networks in order (Algorithm 4). Further details and comparisons of these additional algorithms are provided in Appendix C.

[4] For the formal definition of decision boundary (DB) and activation boundaries (ABs), see Definition C.1 in Appendix C.

Algorithm 1 Compressing algorithm

Input: a pretrained two-layer network $\mathcal{T}$ defined in (5), training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, $\lambda_{scale} > 1$
$m \leftarrow$ the width of $\mathcal{T}$
$K \leftarrow \emptyset$
for $k, l \in [m]$ **do**
  if $\boldsymbol{w}_l^\top \boldsymbol{x}_i + b_l > 0$ implies $\boldsymbol{w}_k^\top \boldsymbol{x}_i + b_k > 0$ for all $i \in [n]$ **then**
    $K \leftarrow K \cup \{k\}$
  end if
end for
if $K \neq \emptyset$ **then**
  $k \leftarrow \arg\min_{k \in K} |v_k| \cdot \|\boldsymbol{w}_k\|$
  remove the k-th neuron $(v_k, \boldsymbol{w}_k, b_k)$ from $\mathcal{T}$
  $m \leftarrow m - 1$
end if
for $k \in [m]$ **do**
  if $\boldsymbol{w}_k^\top \boldsymbol{x}_i + b_k > 0$ and $0 < \mathcal{T}(\boldsymbol{x}_i) < 1$ for some $i \in [n]$ **then**
    $(v_k, \boldsymbol{w}_k, b_k) \leftarrow \lambda_{scale} \times (v_k, \boldsymbol{w}_k, b_k)$
  end if
end for
Output: $\mathcal{T}$

4. Experimental Results

In Section 3, we studied the relationship between the dataset geometry and neural network architectures. In this section, we provide two empirical results: 1) gradient descent indeed converges to the networks we unveil, and 2) we can investigate the geometric features of high-dimensional real-world datasets through our proposed algorithm.

4.1. Convergence on Polytope-Basis Covers

We begin by demonstrating that gradient descent indeed converges to the networks we proposed in the preceding section, without additional regularization terms. We consider two illustrative topological spaces, $\mathcal{X}_1$ and $\mathcal{X}_2$, as depicted in Figure 5. $\mathcal{X}_1$ represents a simplicial 2-complex in $\mathbb{R}^2$, comprising two triangles, while $\mathcal{X}_2$ is a hexagon with a pentagonal hole, possessing a straightforward polytope-basis cover. The objective is to classify points within these spaces in $\mathbb{R}^2$ against others. We evaluate the performance for two loss functions, which are the mean squared error (MSE) loss and the binary cross entropy (BCE) loss functions. For the BCE loss, we applied `SIG` on the last layer.

For the first dataset $\mathcal{X}_1$, Theorem 3.5 suggests that $2 \xrightarrow{\sigma} 6 \xrightarrow{\sigma} 2 \to 1$ is a feasible architecture on $\mathcal{X}_1$. To facilitate a clearer examination of weight vectors in each layer, we plot the activation boundaries[4] in blue (the 1st hidden layer) and red (the 2nd hidden layer) colors, where the gray color denotes the decision boundary of the converged network.

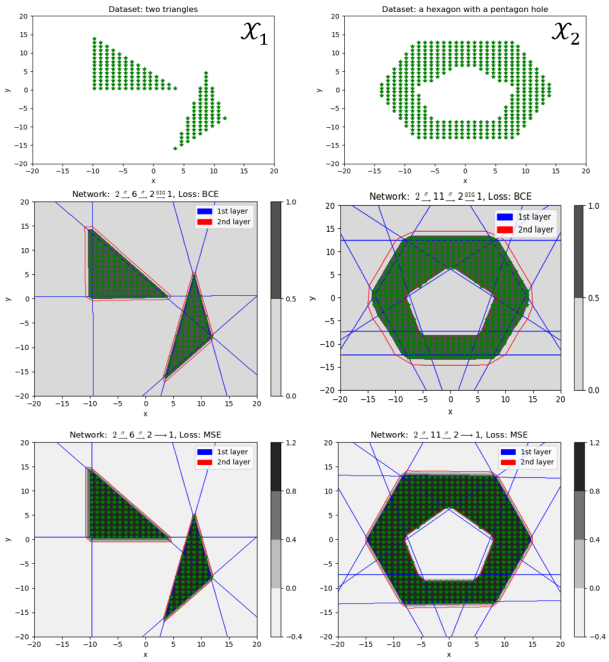
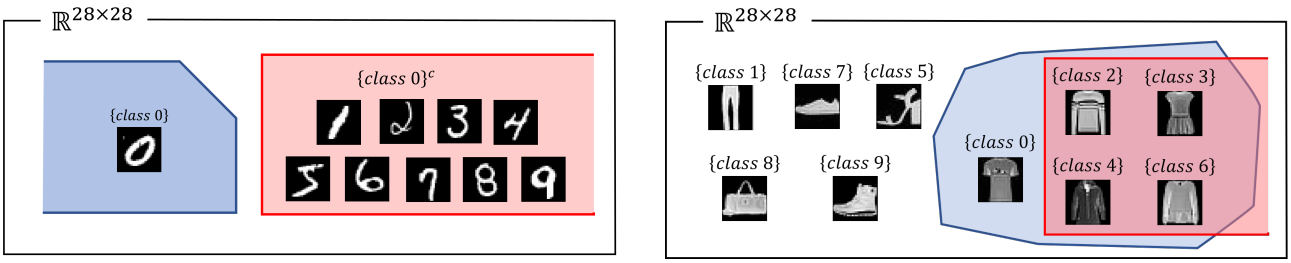


Figure 5. Experimental verification of convergence of gradient descent. Two columns exhibit the shape of two topological spaces, which are 'two triangles' and 'a hexagon with a pentagon hole.' The second and third row show the converged networks by gradient descent under the BCE loss and the MSE loss, respectively. The first layer (blue) and second layer (red) represent the hyperplane and polytopes, respectively, described in Section 3.

The weight vectors in the first layer accurately enclose the two triangles, reflecting the geometrical shape of $\mathcal{X}_1$. Similarly, for the second dataset $\mathcal{X}_2$, Theorem 3.4 suggests that $2 \xrightarrow{\sigma} 11 \xrightarrow{\sigma} 2 \to 1$ is a feasible architecture . Specifically, the eleven neurons in the first layer correspond to the eleven hyperplanes delineating the boundaries of the outer hexagon and the inner pentagon, while two neurons in the second hidden layer align with the two polygons. These outcomes precisely correspond to our network constructions depicted in Figure 2(b, c).

We conclude this section by providing theoretical insights into the convergence behavior of gradient descent. In Appendix D, utilizing our explicit construction of neural networks, we construct an explicit path that loss strictly decreases to zero (the global minima), when the network is initialized close to the target polytope (see Theorem D.3). The specific conditions governing the initialization region are described in terms of the distribution of the dataset along the convex polytope. Although this result does not mean that gradient descent must converge to the global minimum but only guarantees the existence of such a path, it is strong evidence for the convergence to the global minima. For a thorough understanding and precise statements, we refer readers to Appendix D.



(a) (b)

Figure 6. Illustration of a polytope-basis cover of the real datasets. (a) The class $\{0\}$ of MNIST can be separated by a single convex polytope with four faces, while the complement class $\{0\}^c$ can be separated with three faces. (b) The class $\{0\}$ of Fashion-MNIST can be separated by the difference of two polytopes, one of which contains similar images. Other classes also have simple polytope-basis covers as described in Table 1.

4.2. Polytope-Basis Cover for Real Datasets

Here, we delve into the polytope-basis cover analysis of three real-world datasets: MNIST, Fashion-MNIST, and CIFAR10. We focus on binary classification tasks, specifically distinguishing one class from all other classes to obtain a polytope-basis cover of the class. For every class, we also consider its complement, denoted as $\{class\}^c$. We employed Algorithm 2 to get the minimal polytope achieving 99.99% accuracy on the union of the training and test sets.

Our empirical results are presented in Table 1. Each column in the table corresponds to a class in the dataset, where each row presents the type of the class. The values in the table denotes the number of polytopes and their faces (we use notation a+b to denote two polytopes with a and b faces, respectively). For instance, the value in the first row and the first column shows that the $\{0\}$ class images in MNIST dataset can be distinguished from other classes by a single convex polytope with four faces. On the other hand, the second row in the first column shows that the complement of the class, namely $\{0\}^c := \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, can be separated from $\{0\}$ by a convex polytope with three faces, as illustrated in Figure 6(a).

For Fashion-MNIST and CIFAR10 datasets, certain classes that cannot be covered by a convex polytope with less than 30 faces are covered by two polytopes. Figure 6(b) visually illustrates the classification of the class $\{0\}$ in Fashion-MNIST, accomplished through the difference of two polytopes. In the case of CIFAR10 dataset, the number of faces in the polytopes tends to be higher compared to other datasets, consistent with the expectation that CIFAR10 exhibits a more intricate geometric structure than MNIST or Fashion-MNIST.

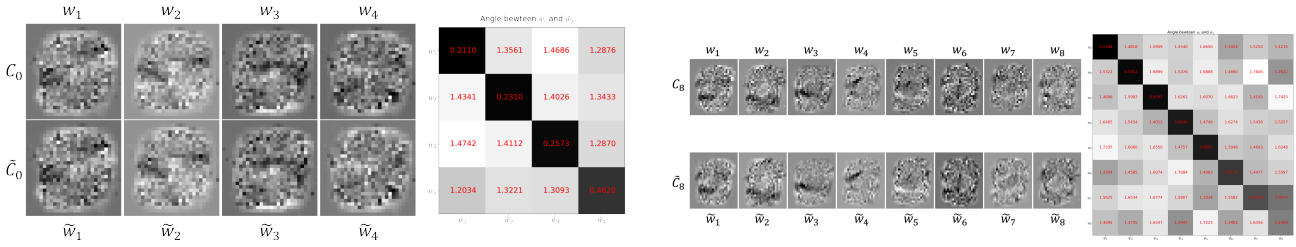
Furthermore, we identify the geometric complexity of each class from Table 1. In Fashion-MNIST, the complement of the class in $\{0, 2, 3, 4, 6\}$ prominently require more faces than other classes, and they all pertain to top clothes and

Datasets		Class 0	1	2	3	4	5	6	7	8	9
MNIST	{class}	4	4	7	8	5	7	4	8	8	7
	$\{class\}^c$	3	3	4	5	4	5	4	4	9	9
Fashion-MNIST	{class}	9+3	4	9+5	9+3	9+6	8	9+7	9+1	6	10
	$\{class\}^c$	16	3	22	11	20	4	28	6	4	5
CIFAR10	{class}	29+3	19	23+3	24+4	19	16+3	21	21	18	21
	$\{class\}^c$	29	7	27+3	26	26+4	17	13	10	20+4	8

Table 1. Polytope-basis covers of each class in MNIST, Fashion-MNIST, and CIFAR10 datasets. Here, a+b denotes two polytopes with a and b faces. For certain classes that cannot be covered by a single convex polytope with less than 30 faces, a second polytope is additionally computed. Overall, each class of real-world datasets can be covered by at most two polytopes, indicating the geometric simplicity of real datasets.

share visual similarities (Figure 6(b)). Furthermore, it fails to find a single convex polytope cover of $\{0\}$ (with less than 30 faces) since the obtained polytope always contains many images in $\{2, 3, 4, 6\}$ classes as illustrated in Figure 6(b). In contrast, the class $\{1\}$ and its complement $\{1\}^c$ are separated by the fewest faces, suggesting they are less entangled with other classes. This observation is consistent with the distinctive, unique shape of the "Trouser" class in Fashion-MNIST dataset. This result highlights *how neural networks can be utilized as a tool for quantifying the geometric complexity of datasets*. We provide additional interesting empirical examples in Appendix B.

We further investigate the uniqueness of the obtained polytope covers in MNIST dataset. When Algorithm 1 is applied to networks initialized with small norm, the obtained covers exhibit noticeable similarity. For MNIST class $\{0\}$, we compute two distinct covering polytopes $\mathcal{C}_0$ and $\tilde{\mathcal{C}}_0$ with four faces. In Figure 7(a), the weight vectors $\boldsymbol{w}_i$ of these covers are visualized, and the angles between the vectors of these two polytopes are displayed. It is easily verified that there is a clear one-to-one correspondence between vectors in the two polytopes, both visually and numerically.



(a) (b)

Figure 7. Visualization of two polytope covers for MNIST classes $\{0\}$ and $\{8\}$. (a) The four faces of two distinct polytope covers for class $\{0\}$ in the MNIST dataset are depicted. (b) The eight faces of two distinct polytope covers for class $\{8\}$ in the MNIST dataset are depicted. The distribution of angles between the vectors in two polytope covers are plotted on the right panel.

For MNIST class $\{8\}$, which has a polytope cover with eight faces, a similar result is provided in Figure 7(b). Although the correspondence is slightly weaker than that of class $\{0\}$ due to the increased number of faces, most vectors still exhibit a strong one-to-one correspondence. From this, it can be seen that Algorithm 1 experimentally provides a unique polytope cover, offering further epexegetic support for the geometric simplicity of MNIST dataset.

Now, we provide feasible architectures for multi-class classifier for real datasets. By combining the results in Table 1 with Theorem 3.4, we can ascertain the feasible architectures of these datasets, based on their geometric characteristics. Note that this is the first result on the minimal network architectures that can completely classify the given datasets, utilizing the geometric features of the datasets. It is provided in the remark below.

Remark 4.1. Adopting the covering polytopes with minimal number in Table 1 for each class, we deduce that

$$\begin{aligned} \text{MNIST}: &\quad 784 \xrightarrow{\sigma} 47 \xrightarrow{\sigma} 10 \to 10 \\ \text{Fashion-MNIST}: &\quad 784 \xrightarrow{\sigma} 90 \xrightarrow{\sigma} 14 \to 10 \\ \text{CIFAR10}: &\quad 3072 \xrightarrow{\sigma} 178 \xrightarrow{\sigma} 12 \to 10 \end{aligned}$$

are feasible architectures for these datasets. Furthermore, the second and third weight matrices in these networks are highly sparse, as demonstrated in the proof of Theorem 3.4 and illustrated in Figure 2(c).

It is worth noting that our findings stem from Algorithm 1, which selectively removes and adjusts neurons within the network. Given the sparse connectivity observed in these networks, we anticipate an inherent connection between our results and the Lottery Ticket Hypothesis (Frankle & Carbin, 2018; Malach et al., 2020). In other words, the sparse pruned neural networks suggested in Remark 4.1 can be understood as an example of the 'winning ticket' in LTH that is explicitly constructed.

Additionally, our results offer another perspective on understanding LTH. For instance, the assumptions in Theorem 3.7 shed light on the significance of masked and unmasked weights, and why maintaining the sign of weight values is important (Zhou et al., 2019). Specifically, the unmasked weights can be associated with the connection of faces to polytopes, and preserving the signs is crucial for maintaining the convex structure of these polytopes, as specified in (5) and (7). We hope our study contributes to future research efforts aimed at elucidating the principles underlying LTH.

5. Conclusion

In this paper, we investigated the geometric characteristics of datasets and neural network architecture. Specifically, we established both upper and lower bounds on the necessary widths of network architectures for classifying given data manifolds, relying on its polytope-basis cover. Furthermore, we extended these insights to simplicial complexes or spaces consisting of prismatic polytopes, shedding light on how the width bound varies in response to the complexity of the dataset. Conversely, we also demonstrated that the polytope structure of datasets can be inspected by training neural networks. We proposed such an algorithm, and our experimental results unveiled that each class within MNIST, Fashion-MNIST, and CIFAR10 datasets can be distinguished by at most two polytopes, implying geometric simplicity of real-world datasets. We further analyzed that the number of faces in the polytope serves as an indicator of the geometric complexity of each class. Our empirical investigations unveil that neural network indeed converges to a polytope-basis cover of dataset, and conversely, it is possible to inspect geometrical features of the dataset from trained neural networks.

Limitations and future work. The optimality of the polytope-basis cover obtained by Algorithm 1 has not been clarified, which remains an interesting avenue for future research. Furthermore, while we only considered fully-connected networks in this work, it is desired to investigate the geometry in other network architectures like convolutional neural networks (CNNs).

Acknowledgements

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT, Ministry of Science and ICT) (No. 2022-0-00984, Development of Artificial Intelligence Technology for Personalized Plug-and-Play Explanation and Verification of Explanation), by National Research Foundation of Korea(NRF) (**RS-2023-00262527**), by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program(KAIST)), and by the National Research Foundation of Korea under Grant RS-2024-00336454

Impact Statement

This paper presents work on describing the polytope structure in deep ReLU networks. This framework provides insights into the geometric roles of neurons and layers, potentially leading to a deeper understanding of high-dimensional dataset geometry and polytope structure of deep ReLU networks. The proposed approach may contribute to investigating data representation, the geometry of feature spaces, and understanding LTH. Lastly, there are no potential societal consequences of this work.

References

Alfarra, M., Bibi, A., Hammoud, H., Gaafar, M., and Ghanem, B. On the decision boundaries of neural networks: A tropical geometry perspective. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45 (4):5027–5037, 2022.

Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In *International Conference on Machine Learning*, pp. 146–155. PMLR, 2017.

Astorino, A. and Gaudioso, M. Polyhedral separability through successive lp. *Journal of Optimization theory and applications*, 112(2):265–293, 2002.

Balestriero, R., Wang, Z., and Baraniuk, R. G. Deephull: Fast convex hull approximation in high dimensions. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3888–3892. IEEE, 2022.

Barannikov, S., Trofimov, I., Sotnikov, G., Trimbach, E., Korotin, A., Filippov, A., and Burnaev, E. Manifold topology divergence: a framework for comparing data manifolds. *Advances in Neural Information Processing Systems*, 34:7294–7305, 2021.

Barannikov, S., Trofimov, I., Balabin, N., and Burnaev, E. Representation topology divergence: A method for comparing neural network representations. *Proceedings of Machine Learning Research*, 162:1607–1626, 2022.

Beise, H.-P., Da Cruz, S. D., and Schröder, U. On decision regions of narrow deep neural networks. *Neural Networks*, 140:121–129, 2021.

Beltran-Royo, C., Llopis-Ibor, L., Pantrigo, J. J., and Ramírez, I. Dc neural networks avoid overfitting in one-dimensional nonlinear regression. *Knowledge-Based Systems*, 283:111154, 2024.

Berzins, A. Polyhedral complex extraction from relu networks using edge subdivision. In *International Conference on Machine Learning*, pp. 2234–2244. PMLR, 2023.

Bianchini, M. and Scarselli, F. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, 25(8):1553–1565, 2014.

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.

Black, S., Sharkey, L., Grinsztajn, L., Winsor, E., Braun, D., Merizian, J., Parker, K., Guevara, C. R., Millidge, B., Alfour, G., et al. Interpreting neural networks through the polytope lens. *arXiv preprint arXiv:2211.12312*, 2022.

Buchanan, S., Gilboa, D., and Wright, J. Deep networks and the multiple manifold problem. In *International Conference on Learning Representations*, 2020.

Bünning, F., Schalbetter, A., Aboudonia, A., de Badyn, M. H., Heer, P., and Lygeros, J. Input convex neural networks for building mpc. In *Learning for Dynamics and Control*, pp. 251–262. PMLR, 2021.

Carlini, N., Liu, C., Erlingsson, Ú., Kos, J., and Song, D. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium (USENIX Security 19)*, pp. 267–284, 2019.

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2633–2650, 2021.

Carlsson, S. Geometry of deep convolutional networks. *arXiv preprint arXiv:1905.08922*, 2019.

Chen, M., Jiang, H., Liao, W., and Zhao, T. Nonparametric regression on low-dimensional manifolds using deep relu networks: Function approximation and statistical recovery. *Information and Inference: A Journal of the IMA*, 11 (4):1203–1253, 2022.

Chen, Y., Shi, Y., and Zhang, B. Input convex neural networks for optimal voltage regulation. *arXiv preprint arXiv:2002.08684*, 2020.

Cohen, U., Chung, S., Lee, D. D., and Sompolinsky, H. Separability and geometry of object manifolds in deep neural networks. *Nature communications*, 11(1):746, 2020.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Dirksen, S., Genzel, M., Jacques, L., and Stollenwerk, A. The separation capacity of random neural networks. *The Journal of Machine Learning Research*, 23(1):13924–13970, 2022.

Du, S. S., Hu, W., and Lee, J. D. Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced. *Advances in neural information processing systems*, 31, 2018.

Fan, F.-L., Huang, W., Zhong, X., Ruan, L., Zeng, T., Xiong, H., and Wang, F. Deep relu networks have surprisingly simple polytopes. *arXiv preprint arXiv:2305.09145*, 2023.

Fan, J., Taghvaei, A., and Chen, Y. Scalable computations of wasserstein barycenter via input convex neural networks. In *International Conference on Machine Learning*, pp. 1571–1581. PMLR, 2021.

Fawzi, A., Moosavi-Dezfooli, S.-M., Frossard, P., and Soatto, S. Empirical study of the topology and geometry of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3762–3770, 2018.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

Goldt, S., Mézard, M., Krzakala, F., and Zdeborová, L. Modeling the influence of data structure on learning in neural networks: The hidden manifold model. *Physical Review X*, 10(4):041044, 2020.

Gorban, A. N. and Tyukin, I. Y. Blessing of dimensionality: mathematical foundations of the statistical physics of data. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376 (2118):20170237, 2018.

Grigsby, J. E. and Lindsey, K. On transversality of bent hyperplane arrangements and the topological expressiveness of relu neural networks. *SIAM Journal on Applied Algebra and Geometry*, 6(2):216–242, 2022.

Gromov, M. Curvature, diameter and betti numbers. *Commentarii Mathematici Helvetici*, 56:179–195, 1981.

Haase, C. A., Hertrich, C., and Loho, G. Lower bounds on the depth of integral relu neural networks via lattice polytopes. In *The Eleventh International Conference on Learning Representations*, 2022.

Haim, N., Vardi, G., Yehudai, G., Shamir, O., and Irani, M. Reconstructing training data from trained neural networks. *Advances in Neural Information Processing Systems*, 35: 22911–22924, 2022.

Hajij, M. and Istvan, K. A topological framework for deep learning. *arXiv preprint arXiv:2008.13697*, 2020.

Hajij, M. and Istvan, K. Topological deep learning: Classification neural networks. *arXiv preprint arXiv:2102.08354*, 2021.

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

Hannouch, K. M. and Chalup, S. Topology estimation of simulated 4d image data by combining downscaling and convolutional neural networks. *arXiv preprint arXiv:2306.14442*, 2023.

Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

Huchette, J., Muñoz, G., Serra, T., and Tsay, C. When deep learning meets polyhedral theory: A survey. *arXiv preprint arXiv:2305.00241*, 2023.

Kantchelian, A., Tschantz, M. C., Huang, L., Bartlett, P. L., Joseph, A. D., and Tygar, J. D. Large-margin convex polytope machine. *Advances in Neural Information Processing Systems*, 27, 2014.

Kim, K., Kim, J., Zaheer, M., Kim, J., Chazal, F., and Wasserman, L. Pllay: Efficient topological layer based on persistent landscapes. *Advances in Neural Information Processing Systems*, 33:15965–15977, 2020.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*, 2018.

Liu, Y., Cole, C. M., Peterson, C., and Kirby, M. Relu neural networks, polyhedral decompositions, and persistent homology. In *Topological, Algebraic and Geometric Learning Workshops 2023*, pp. 455–468. PMLR, 2023.

Magai, G. and Ayzenberg, A. Topology and geometry of data manifold in deep learning. *arXiv preprint arXiv:2204.08624*, 2022.

Makkuva, A., Taghvaei, A., Oh, S., and Lee, J. Optimal transport mapping via input convex neural networks. In *International Conference on Machine Learning*, pp. 6672–6681. PMLR, 2020.

Malach, E., Yehudai, G., Shalev-Schwartz, S., and Shamir, O. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pp. 6682–6691. PMLR, 2020.

Manwani, N. and Sastry, P. Learning polyhedral classifiers using logistic function. In *Proceedings of 2nd Asian Conference on Machine Learning*, pp. 17–30. JMLR Workshop and Conference Proceedings, 2010.

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

Milnor, J. W., Spivak, M., Wells, R., and Wells, R. *Morse theory*. Princeton university press, 1963.

Naitzat, G., Zhitnikov, A., and Lim, L.-H. Topology of deep neural networks. *The Journal of Machine Learning Research*, 21(1):7503–7542, 2020.

Park, S., Yun, C., Lee, J., and Shin, J. Minimum width for universal approximation. In *International Conference on Learning Representations*, 2020.

Paul, R. and Chalup, S. Estimating betti numbers using deep learning. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7. IEEE, 2019.

Pestov, V. Is the k-nn classifier in high dimensions affected by the curse of dimensionality? *Computers & Mathematics with Applications*, 65(10):1427–1437, 2013.

Piwek, P., Klukowski, A., and Hu, T. Exact count of boundary pieces of relu classifiers: Towards the proper complexity measure for classification. In *Uncertainty in Artificial Intelligence*, pp. 1673–1683. PMLR, 2023.

Rolnick, D. and Kording, K. Reverse-engineering deep relu networks. In *International Conference on Machine Learning*, pp. 8178–8187. PMLR, 2020.

Rudin, W. et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1976.

Sankaranarayanan, P. and Rengaswamy, R. Cdinn–convex difference neural networks. *Neurocomputing*, 495:153–168, 2022.

Sivaprasad, S., Singh, A., Manwani, N., and Gandhi, V. The curious case of convex neural networks. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part I 21*, pp. 738–754. Springer, 2021.

Telgarsky, M. Representation benefits of deep feedforward networks. *arXiv preprint arXiv:1509.08101*, 2015.

Tiwari, S. and Konidaris, G. Effects of data geometry in early deep learning. *Advances in Neural Information Processing Systems*, 35:30099–30113, 2022.

Vallin, J., Larsson, K., and Larson, M. G. The geometric structure of fully-connected relu-layers. *arXiv preprint arXiv:2310.03482*, 2023.

Vincent, J. A. and Schwager, M. Reachable polyhedral marching (rpm): An exact analysis tool for deep-learned control systems. *arXiv preprint arXiv:2210.08339*, 2022.

Wang, T., Buchanan, S., Gilboa, D., and Wright, J. Deep networks provably classify data on curves. *Advances in neural information processing systems*, 34:28940–28953, 2021.

Xu, S., Vaughan, J., Chen, J., Zhang, A., and Sudjianto, A. Traversing the local polytopes of relu neural networks. In *The AAAI-22 Workshop on Adversarial Machine Learning and Beyond*, 2021.

Zhou, H., Lan, J., Liu, R., and Yosinski, J. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in neural information processing systems*, 32, 2019.

Appendix

Appendix A. Related Works

Appendix B. Geometric Simplicity of Real-World Datasets

Appendix C. Algorithms for Finding Polytope-Basis Covers
C.1 A Polytope-Basis Cover Derived from a Trained Three-Layer ReLU Network
C.2 A Polytope-Basis Cover Derived from a Trained Two-layer ReLU Network
C.3 An Efficient Algorithm to Find a Simple Polytope-Basis Cover
C.4 Comparison of the Proposed Algorithms.

Appendix D. Convergence on the Proposed Networks

Appendix E. Proofs
E.1 Proof of Proposition 3.1
E.2 Proof of Theorem 3.4
E.3 Proof of Theorem 3.5
E.4 Proof of Theorem 3.6
E.5 Proof of Theorem 3.7
E.6 Proof of Proposition C.2
E.7 Proof of Theorem D.3

Appendix F. Additional Propositions and Lemmas

A. Related Works

Geometric approaches to ReLU networks. Various geometric methodologies have been employed to explore the approximation capabilities of deep ReLU networks. Hanin & Rolnick (2019), for instance, introduced the concept of bent hyperplanes in the input space, assessing its theoretical and empirical complexities. The methodology known as the 'bent hyperplane arrangement' has been applied across various research domains. Notably, it has been utilized in the analysis of decision regions (Beise et al., 2021; Grigsby & Lindsey, 2022; Black et al., 2022) or characterizing linear regions within ReLU networks (Rolnick & Kording, 2020), which is also intricately connected to our proofs in Theorem 3.1 and 3.6.

On the other hand, there has been a burgeoning interest in investigating polytope structures induced by deep ReLU networks (Fawzi et al., 2018; Xu et al., 2021; Alfarra et al., 2022; Vincent & Schwager, 2022; Black et al., 2022; Haase et al., 2022; Liu et al., 2023; Fan et al., 2023; Huchette et al., 2023; Vallin et al., 2023; Piwek et al., 2023). Masden (2022) introduced algorithms capable of extracting the polytope structure inherent in networks and deriving topological properties of the decision boundary. Carlsson (2019) and Vallin et al. (2023) considered the pre-image of ReLU networks, characterizing the geometric shapes of the decision boundary to gain an understanding of the polytope partitions of deep ReLU networks.

Nonetheless, there has been a scarcity of exploration into the explicit construction of neural networks for the purpose of classifying a given dataset, as illustrated in Figure 1. In this study, we address the practical challenge of distinguishing between the two data manifolds, and we introduce practical algorithms for constructing covering polytopes based on the properties of ReLU networks. This approach can be considered a complementary method for investigating the approximation capabilities of neural networks in terms of polytopes - a geometric aspect that has not been extensively explored.

Exploring input convexity in neural networks. Recent years have witnessed a surge in research dedicated to unraveling the convexity of ReLU networks concerning their input. Crucially, it has been established that the ReLU activation function demonstrates convexity concerning its input when composited weights are all positive (Proposition 1 in (Amos et al., 2017), cf. Lemma F.4). This discovery led to the inception of Input Convex Neural Networks (ICNNs) by Amos et al. (2017),

a characteristic that has been effectively leveraged in various applications (Makkuva et al., 2020; Chen et al., 2020; Fan et al., 2021; Bünning et al., 2021; Balestriero et al., 2022). Notably, Balestriero et al. (2022) applied this idea and proposed *DeepHull*, the algorithm to approximate the convex hull by convex deep networks. On the other hand, Sivaprasad et al. (2021) asserted that these convex neural networks exhibit self-regularization effects, demonstrating superior generalization performance in specific tasks. Additionally, research has delved into representing neural networks as the difference of convex (DC) functions (Sankaranarayanan & Rengaswamy, 2022; Piwek et al., 2023; Beltran-Royo et al., 2024). Sankaranarayanan & Rengaswamy (2022) employed Linear Programming to optimize polyhedral DC functions, capitalizing on the piecewise linearity induced by the ReLU activation function. (Beltran-Royo et al., 2024) showed that DC NNs have an implicit bias avoiding overfitting in 1-D nonlinear regularization. (Sankaranarayanan & Rengaswamy, 2022) proposed a new network architecture called Convex Difference Neural Network (CDiNN), and suggested using convex concave procedure for optimization. These findings of previous studies suggest that investigating neural networks through the lens of differences in convex functions holds significant potential for future research.

In this study, we decompose a trained ReLU network into the difference of several convex functions and leverage this decomposition to induce a polytope-basis cover for a given dataset. Specifically, we employ the ICNN architectures to derive a convex polytope cover, revealing how trained neural networks capture the geometric characteristics of the training dataset. While Balestriero et al. (2022) explored a similar approach by minimizing the volume of the polytope as a regularizer, our main focus is on minimizing the number of neurons to reduce a polytope with a small number of faces. Consequently, our empirical results can be considered as an application of ICNNs to extract geometric features of datasets in terms of polytopes.

Moreover, it is noteworthy that previous studies imposed restrictions on the weights, either by enforcing $W_k \geq 0$ (Amos et al., 2017; Sivaprasad et al., 2021) or by substituting them with squared values $W_k^2 \geq 0$ (Sankaranarayanan & Rengaswamy, 2022), to maintain network convexity. In contrast, we achieve this solely by adjusting the initialization conditions, leveraging the implicit bias of gradient descent (Du et al., 2018).

Complexity of datasets and neural network architectures. Several studies have delved into the intricate relationship between the geometric features of datasets and neural network training, often referred to as the *multiple manifold problem* (Goldt et al., 2020; Buchanan et al., 2020; Wang et al., 2021; Chen et al., 2022; Tiwari & Konidaris, 2022). This problem typically involves the binary classification of two low-dimensional manifolds by neural networks. For instance, Buchanan et al. (2020) and Wang et al. (2021) focused on the task of distinguishing between two curves (i.e., one-dimensional manifolds), investigating the convergence speed and generalization concerning the geometric features of the dataset. Similarly, in the context of low-dimensional data manifolds, Tiwari & Konidaris (2022) examined the effects of data geometry on the complexity of trained neural networks, measuring the distance to the manifold. In a related vein, Dirksen et al. (2022) considered the separation problem with randomly initialized ReLU networks, explicitly linking required widths to weight initialization.

Furthermore, there are other numerous empirical studies that have supported the implicit relationship between network architecture and the geometric complexity or topological structure of the data manifold (Fawzi et al., 2018; Kim et al., 2020; Cohen et al., 2020; Birdal et al., 2021; Barannikov et al., 2022; 2021; Naitzat et al., 2020; Hajij & Istvan, 2020; 2021; Magai & Ayzenberg, 2022). Additionally, Li et al. (2018) empirically investigated the loss landscape to measure the intrinsic dimension of datasets, which is deeply related to the minimal neural network architecture. These theoretical and empirical findings suggest a high correlation between neural network architecture and the training dataset: *a more complicated dataset requires a more complex architecture*.

However, a notable gap still exists in the literature when it comes to explicitly constructing a neural network in practice. For example, as introduced in Section 1, it remains unknown which architecture of neural networks can or cannot completely classify a given dataset with explicit construction of neural networks.

Estimating dataset characteristics from a trained network. Once a neural network is trained, it is well-established that the trained network encapsulates information or characteristics of the training dataset (Carlini et al., 2019; 2021; Haim et al., 2022). This phenomenon, implies that neural networks can be employed to extract information about the dataset through the training process. From a topological perspective, Paul & Chalup (2019) introduced a method for estimating the Betti numbers of 2D or 3D datasets using convolutional neural networks, later extended to 4D datasets by (Hannouch & Chalup, 2023). In the geometric realm, Kantchelian et al. (2014) proposed the large-margin convex polytope machine with a training algorithm to find a convex polytope that encloses one class with a large margin. Their empirical results demonstrated that

the digit '2' class of MNIST has a simple convex polytope cover that generalizes well.

However, there are still opportunities for more precise investigations into the dataset geometry in terms of polytopes, which are induced from ReLU networks. In this paper, we propose an algorithm that derives a polytope-basis cover of a given dataset by learning neural networks, building upon our theoretical analysis. Moreover, it reveals the number of faces of the polytopes, which can describe the geometric complexity of datasets.

Our contributions. In this paper, we harmonize diverse perspectives on neural networks, offering insights into the intrinsic relationship between the geometric complexity of datasets and network architectures. Our focus centers on the polytope structure induced by ReLU networks (Theorems 3.4, 3.5, 3.6). The principal contribution of our work lies in delineating lower and upper bounds on widths within deep ReLU networks for the classification of a given dataset, drawing from the polytope structure inherent in the data. Our theoretical results not only elucidate how the geometric complexity of a dataset influences the required widths of neural networks but also illuminate the nuanced role of neurons in deep layers.

Furthermore, we present algorithms aimed at deriving a polytope-basis cover for given datasets, thereby highlighting the inherent link between trained neural networks and the polytope structure of the training datasets. While prior research (Kantchelian et al., 2014; Sivaprasad et al., 2021) merely showcased the polyhedral separability of certain classes in MNIST or CIFAR10, we determine the exact number of faces required for covering polytopes of each class. Essentially, our work introduces a novel methodology for exploring the intricate relationship between dataset geometry and neural networks.

B. Geometric Simplicity of Real-World Datasets

In this section, we present empirical results illustrating how the number of faces of covering polytopes reflects the geometric characteristics of a class. Specifically, we examine a classification task under varying levels of noisy labels. We focus on the class $\{1\}$ in each dataset and manipulate the noise levels (denoted as r) across values of 0, 0.01, 0.1, 0.25, 0.5, and 0.90 while maintaining the total number of data points in the class. The noisy class with noise level r is denoted by $\{1\}_r^{noise}$.

Here we give an example: When $r = 0.00$, the $\{1\}_{r=0.00}^{noise}$ class is identical to the $\{1\}$ class in Fashion-MNIST, comprising 6000 T-shirt images. As r increases, such as $r = 0.01$, the noised class $\{1\}_{r=0.01}^{noise}$ still consists of 6000 images, but only 99% of them are T-shirt images, with the remaining 1% randomly selected from other classes. Consequently, when $r = 0.9$, $\{1\}_{r=0.9}^{noise}$ contains 600 T-shirt images and 5400 randomly selected images from other classes, i.e., totally randomly selected images in the Fashion-MNIST dataset. The task is finding a single covering polytope that contains $\{1\}_{r=0.9}^{noise}$ against the other data.

We utilize Algorithm 1 to identify a single polytope with minimal width, achieving an accuracy greater than 99.9% on the noised dataset [5]. To mitigate the randomness inherent in image selection, we compute the average value over five repetitions. The results are presented in Table 2.

Dataset \ noise level r	0.00	0.01	0.10	0.25	0.50	0.90
MNIST class $\{1\}$	3	5.2	29.2	46.6	64.0	52.4
Fashion-MNIST class $\{1\}$	3	4.6	39.0	57.4	77.0	65.6
CIFAR10 class $\{1\}$	12	12.8	16.2	19.6	28.4	35.6

Table 2. The average number of faces required for a convex polytope to cover the noisy class $\{1\}_r^{noise}$, with varying levels of random labels (r). As the proportion of random labels in the class rises, there is a noticeable corresponding increase in the requisite number of faces.

The results illustrate a positive correlation between the prevalence of noisy labels and the increasing complexity of covering polytopes: as the level of noise rises, the requisite number of faces also increases accordingly. Specifically, Table 2 demonstrates that images within the same class can be effectively distinguished by a convex polytope with a smaller number of faces, highlighting the inherent geometric complexity of real-world datasets.

[5] Note that Tables 1 and 2 employ different accuracy criteria. With a stricter criterion of 99.99% in Table 1, the identified polytopes are likely more precise but potentially require the larger number of faces. Conversely, the looser criterion of 99.9% used in Table 2 may lead to slightly loose polytope, but easier to identify.

Moreover, our result can demonstrate some previous works that classified these datasets using convex polytopes. Notably, for MNIST and CIFAR10 datasets, Kantchelian et al. (2014) and Sivaprasad et al. (2021) observed that convex polytope classifiers exhibit self-regularization effects and robustness to label noise. Our results presented in Table 2 validate this observation based on the geometric features of the datasets: each class in the datasets inherently possesses simple geometry, enabling convex classifiers to generalize effectively.

A particularly intriguing finding emerges when comparing the results with 50% corrupted labels ($r = 0.5$) to those with fully-mixed labels ($r = 0.9$). Notably, for MNIST and Fashion-MNIST datasets, Table 2 suggests that the 50% corrupted dataset is even more challenging to segregate than the fully-mixed counterpart. In other words, a convex polytope with fewer faces must encompass the original images from the uncorrupted class $\{1\}$ - the 'digit 1 images' in MNIST or the 'Trouser images' in Fashion-MNIST-, necessitating a larger number of faces for the polytope to effectively segregate them. This observation emphasizes the geometric simplicity of MNIST and Fashion-MNIST datasets, contrasting with the behavior observed in CIFAR10, which does not exhibit this trend.

Remark B.1. To prevent potential misunderstandings regarding the results in Table 2, we offer two insights. Firstly, in the case of the fully-mixed class ($r = 0.9$), it is not surprising that an arbitrary set of points in high-dimensional space can be separated by a single convex polytope. This phenomenon is rooted in the curse of dimensionality, where in higher dimensions, points are inherently easier to separate (Pestov, 2013; Gorban & Tyukin, 2018). Indeed, Table 2 illustrates that random images in CIFAR10 are separated with fewer faces compared to other datasets. Secondly, it's worth noting that although two manifolds may be polyhedrally separable, this does not necessarily imply that each manifold is convex, as depicted in Figure 8.

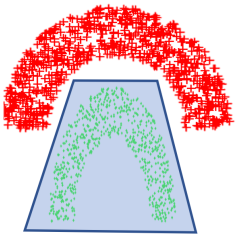


Figure 8. Although two manifolds are polyhedrally separable, both manifolds may not exhibit convexity.

C. Algorithms for Finding Polytope-basis Covers

This section delves into the inner workings of our algorithms. Each algorithm is presented with a breakdown of its steps, the rationale behind its design, and its theoretical underpinnings. Additionally, two novel algorithms dedicated to polytope-basis cover reduction are introduced. For clarity, the section is organized into four sections.

In Section C.1, we delve into the motivation behind Algorithm 1 and provide a comprehensive explanation of its implementation. We introduce Algorithm 3 in Section C.2, which extracts a polytope-basis cover from any trained two-layer ReLU network. Section C.3 introduces Algorithm 4, an efficient algorithm for generating a minimal polytope-basis cover for a given dataset. Finally, Section C.4 presents a comparative analysis of all proposed algorithms.

C.1. A Polytope-Basis Cover Derived From a Three-Layer ReLU Network

In Section 3.2, we introduced the compressing algorithm (Algorithm 1) that deforms a two-layer network to represent a single convex polytope. Before we provide detail descriptions, we introduce some terminologies.

Definition C.1. Let $\mathcal{N}(\boldsymbol{x}) := v_0 + \sum_{k=1}^{m} v_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)$ be a two-layer ReLU network. For each $k \in [m]$, we refer a pair $(v_k, \boldsymbol{w}_k, b_k)$ as a *neuron* of $\mathcal{N}$. We say *a neuron* $\sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)$ *activates (or deactivates)* $\boldsymbol{x}$ if $\boldsymbol{w}_k^\top \boldsymbol{x} + b_k > 0$ (or $\boldsymbol{w}_k^\top \boldsymbol{x} + b_k < 0$, respectively). The *activation boundary (AB)* of a neuron $(v_k, \boldsymbol{w}_k, b_k)$ is defined by $\{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{w}_k^\top \boldsymbol{x} + b_k = 0\}$. Similarly, the *decision boundary (DB)* of $\mathcal{N}$ is defined by the set $\{\boldsymbol{x} \in \mathbb{R}^d \mid \mathcal{N}(\boldsymbol{x}) = 0\}$.

Roughly speaking, the activation boundaries of $\mathcal{T}$ are non-differentiable points of $\mathcal{T}$. For example, the leftmost column in Figure 4 or 9 shows the decision boundary and activation boundaries of trained networks.

Now, we provide in-depth explanations of the algorithms introduced in the main text. Let $\mathcal{T}$ be a two-layer ReLU network defined in (5), thus $v_k < 0$ for all $k \in [m]$. Let S be the region defined by $S := \{\boldsymbol{x} \mid \mathcal{T}(\boldsymbol{x}) = \lambda\}$. If it is nonempty, then it is a convex polytope with m faces by Definition 2.1. However, its decision boundary $R := \{\boldsymbol{x} \mid \mathcal{T}(\boldsymbol{x}) > 0\} \supset S$ may contain more data points than S. For the given dataset $\mathcal{D}$, to achieve (7), the objective of the compressing algorithm is to achieve

$$R \cap \mathcal{D} = S \cap \mathcal{D}. \tag{9}$$

Note that (9) directly implies (7). Below, we demonstrate a detailed examination of the compressing algorithm step-by-step,

which was briefly introduced in Section 3. The algorithm comprises two parts: **1.** eliminating a redundant neuron, and **2.** scaling some neurons.

- **PART 1. Eliminating a redundant neuron.** The first part involves the removal of remaining redundant neurons that may activate some data points but do not contribute significantly to the network output. Specifically, in Figure 3(a), the neuron indicated by the red arrow does not contribute to the change of the decision boundary since the training data points activated by this neuron already exhibit negative outputs, i.e., $\mathcal{T}(\boldsymbol{x}) < 0$ already. In essence, eliminating such neurons does not significantly alter the decision boundary of $\mathcal{T}$. However, although the removal of this neuron maintains the decision boundary of $\mathcal{T}$, it does affect the output value of $\mathcal{T}$, consequently influencing other subnetworks in the three-layer network $\mathcal{N}$. To address this, the algorithm removes only one neuron at once, which has the smallest value of $|v_k| \cdot \|\boldsymbol{w}_k\|$.
- **PART 2. Scaling neurons.** The second part is deforming the given network to satisfy (9) by magnifying neurons. From the definition of $\mathcal{T}(\boldsymbol{x}) = \lambda + \sum_k v_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)$, recall that all $v_k < 0$. If we take $v_k \to -\infty$ for all k, then the region $R = \{\boldsymbol{x} \in \mathbb{R}^d \mid 0 < \mathcal{T}(\boldsymbol{x})\}$ shrinks to the region $S = \{\boldsymbol{x} \in \mathbb{R}^d \mid \mathcal{T}(\boldsymbol{x}) = \lambda\}$ (cf. Figure 15(b)). This is the trick we used to figure out the minimal polytope representation of the decision region.

  In this step, we just increase the magnitude of a neuron $(v_k, \boldsymbol{w}_k, b_k)$ for $k \in [m]$ if it has an activated data $\boldsymbol{x}_i$ such that $\mathcal{T}_j(\boldsymbol{x}) \neq \lambda$. The multiplication constant is referred to λ_{scale} to $(v_k, \boldsymbol{w}_k, b_k)$ in the algorithm. Furthermore, note also that the updated neuron still satisfies (8), thus gradient desecent algorithm can be parallelized with keeping $v_k < 0$. It is noteworthy that in Figure 3(b), the decision boundary demonstrably shrinks to the polytope shown in (c).

However, removing a neuron (PART 1) and adjusting the scaling of some neurons (PART 2) will inevitably alter the network's output, which could potentially decrease its classification accuracy. Therefore, as highlighted in Section 3, it is advisable to apply the compressing algorithm in conjunction with an optimization process, as outlined in Algorithm 2.

Algorithm 2 Extracting a polytope-basis cover from a three-layer ReLU network

Require: a pretrained three-layer network $\mathcal{N}(\boldsymbol{x})$ defined in (6), the training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, Epochs
for $epoch = 1, \cdots, Epochs$ **do**
  for $iteration = 1, 2, \cdots, 1000$ **do**
    one-step gradient descent for $\mathcal{N}$ under the BCE loss (14)
  end for
  for $j = 1, \cdots, J$ **do**
    $\mathcal{T}_j \leftarrow \texttt{COMP}(\mathcal{T}_j)$ ▷ the compressing algorithm (Algorithm 1)
  end for
end for
if there exists $i \in [n]$ and $j \in [J]$ such that $0 < \mathcal{T}_j(\boldsymbol{x}_i) < 1$ **then**
  repeat
    $\mathcal{T}_j \leftarrow \texttt{COMP}(\mathcal{T}_j)$ ▷ the compressing algorithm (Algorithm 1)
  until $\sigma(\mathcal{T}_j(\boldsymbol{x}_i))$ is either 0 or 1 for all $i \in [n]$ and $j \in [J]$
end if
Output: $\mathcal{N}$

Figure 4 illustrates the results obtained by applying Algorithm 2. A three-layer network $\mathcal{N}$ defined in (6) with $J = 16$ polytopes, each consisting of 20 neurons (architecture $2 \xrightarrow{\sigma} 320 \xrightarrow{\sigma} 16 \to 1$), was pre-trained on the swiss roll dataset. Subsequently, Algorithm 1 was applied to compress each $\mathcal{T}_j$ with fine tuning, resulting in a compressed three-layer network with architecture $2 \xrightarrow{\sigma} 72 \xrightarrow{\sigma} 16 \to 1$. If the obtained network still completely classify the dataset, then we can derive the polytope basis cover by Theorem 3.4. More precisely, the polytopes defined by $C_j := \{\boldsymbol{x} \in \mathbb{R}^d \mid \mathcal{T}_j(\boldsymbol{x}) = \lambda\}$, the collection of polytoeps $\mathcal{C} = \{C_j\}_{j \in [J]}$ becomes a polytope-basis cover of the dataset, comprising these 16 polytopes of $\mathcal{N}$. The obtained polytopes are illustrated in Figure 4(b, c).

Furthermore, it is noteworthy to mention the time efficiency of both Algorithm 1 and 2. Despite the presence of multiple **for** loops, these algorithms are implemented efficiently using parallel computing in PyTorch. Empirically, they demonstrate quick performance with practical neural network widths. On a theoretical level, Proposition C.2 ensures that Algorithm 2 terminates within a finite timeframe and provides a polytope-basis cover that maintaining the accuracy of the converged neural network $\mathcal{N}$.

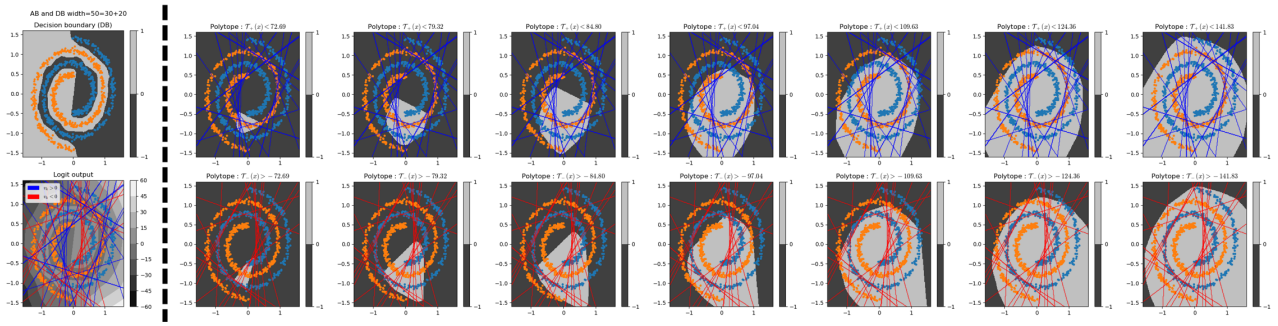


Figure 9. A polytope-basis cover derived by Algorithm 3 from a trained two-layer ReLU network with architecture $2 \xrightarrow{\sigma} 50 \to 1$. The decision boundary and all activation boundaries of the converged network are depicted in the leftmost column. The algorithm provides a polytope-basis cover consists of 68 polytopes, and some of them are illustrated in other columns.

C.2. A Polytope-Basis Covers Derived From a Two-Layer ReLU Network

In this section, we propose another algorithm that extracts a polytope-basis cover from a trained two-layer ReLU network $\mathcal{N}$ defined in (1). First, we decompose $\mathcal{N}$ as the sum of convex and concave functions by aligning it according to the sign of the weight values.

$$\begin{aligned}\mathcal{N}(\boldsymbol{x}) &= v_0 + \sum_{k=1}^{m} v_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}_k + b_k) \\ &= \left(\frac{1}{2}v_0 + \sum_{v_k>0} v_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}_k + b_k)\right) + \left(\frac{1}{2}v_0 + \sum_{v_k<0} v_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}_k + b_k)\right) \\ &=: \mathcal{N}_+(\boldsymbol{x}) + \mathcal{N}_-(\boldsymbol{x}).\end{aligned}$$

Note that both $\mathcal{N}_+$ and $\mathcal{N}_-$ are convex and concave functions, respectively, by Lemma F.4. Now, we consider the network output of each data. For any $\boldsymbol{x}_i \in \mathcal{D}$, we have

$$\begin{aligned}\mathcal{N}(\boldsymbol{x}_i) > 0 \quad &\Leftrightarrow \quad \mathcal{N}_+(\boldsymbol{x}_i) > -\mathcal{N}_-(\boldsymbol{x}_i), \\ \mathcal{N}(\boldsymbol{x}_i) < 0 \quad &\Leftrightarrow \quad \mathcal{N}_+(\boldsymbol{x}_i) < -\mathcal{N}_-(\boldsymbol{x}_i).\end{aligned}$$

Then, we quantize these functions to derive a polytope-basis cover. With the similar idea of Lebesgue integration, we can approximate the convex function $\mathcal{N}_+(\boldsymbol{x})$ by a *simple function*. Here, the *simple function* means a linear combination of indicator functions, basically considered in mathematical field like Lebesgue theory (Rudin et al., 1976). Define $M := \max_{\boldsymbol{x}\in\mathcal{D}} \mathcal{N}_+(\boldsymbol{x})$. Then, for a given $\varepsilon > 0$, we can approximate $\mathcal{N}_+(\boldsymbol{x})$ by

$$\begin{aligned}\mathcal{N}_+(\boldsymbol{x}) &\approx M - \sum_{l=0}^{\infty} l\varepsilon \cdot \mathbb{1}_{\{M-(l+1)\varepsilon < \mathcal{N}_+(\boldsymbol{x}) < M - l\varepsilon\}}(\boldsymbol{x}) \\ &= M - \varepsilon \sum_{C\in\mathcal{C}_Q} \mathbb{1}_{\{\boldsymbol{x}\in C\}}\end{aligned}$$

where $\mathcal{C}_Q$ is the collection of polytopes defined by $C_l := \{\boldsymbol{x} \mid \mathcal{N}_+(\boldsymbol{x}) < M - l\varepsilon\}$ for $l = 0, 1, \cdots$. This decomposition can be understood as quantization of $\mathcal{N}_+(\boldsymbol{x})$ by slices with height ε. Moreover, the above approximation would be accurate as $\varepsilon \to 0$. Similarly, $\mathcal{N}_-(\boldsymbol{x})$ can be approximated in the similar manner. The main idea to obtain a polytope-basis cover $\mathcal{C}$ from $\mathcal{N}$ is selecting sufficiently many ε's to quantize $\mathcal{N}_+$ and $\mathcal{N}_-$.

Empricially, we construct a polytope-basis cover from the values of $\mathcal{N}$. If $\mathcal{C}$ is not a polytope-basis cover yet, we select an incorrectly classified data point $\hat{\boldsymbol{x}}$ with the smallest confidence value, i.e., $\hat{\boldsymbol{x}} := \arg\min_{\boldsymbol{x}_i\in\mathcal{D}} |\mathcal{N}(\boldsymbol{x}_i)|$. Then, we choose an intermediate value c between $\mathcal{N}_+(\hat{\boldsymbol{x}})$ and $\mathcal{N}_-(\hat{\boldsymbol{x}})$ and add two polytopes $C_+ := \{\boldsymbol{x} \mid \mathcal{N}_+(\boldsymbol{x}) < c\}$ and $C_- := \{\boldsymbol{x} \mid \mathcal{N}_-(\boldsymbol{x}) > -c\}$ to the polytope-basis cover $\mathcal{C}$. Then, the value

$$\left| \sum_{C\in\mathcal{C}_P} \mathbb{1}_{\{\boldsymbol{x}\in C\}}(\hat{\boldsymbol{x}}) - \sum_{C\in\mathcal{C}_Q} \mathbb{1}_{\{\boldsymbol{x}\in C\}}(\hat{\boldsymbol{x}}) \right|$$

is decreased by one since $\hat{\boldsymbol{x}}$ is contained in either C_+ or C_-. Therefore, by repeating this process sufficiently many times, $\mathcal{C}$ will correctly classify $\hat{\boldsymbol{x}}$. Based on this idea, we provide Algorithm 3 that extracts a polytope-basis cover from a given trained two-layer ReLU network $\mathcal{N}$.

Algorithm 3 Extracting a polytope-basis cover from a trained two-layer ReLU network

Require: a pretrained two-layer ReLU network $\mathcal{N}$ defined in (1), training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$
Declare the empty collections $\mathcal{C}_P$ and $\mathcal{C}_Q$.
repeat
  Define $o_i := \sum_{C \in \mathcal{C}_P} \mathbb{1}_{\{\boldsymbol{x}_i \in C\}} - \sum_{C \in \mathcal{C}_Q} \mathbb{1}_{\{\boldsymbol{x}_i \in C\}} - \frac{1}{2}$ for all $i \in [n]$.
  if $\mathcal{C}$ is not a polytope-basis cover of $\mathcal{D}$ **then**
    $\hat{\boldsymbol{x}} \leftarrow \underset{\text{sgn}(o_i) \neq \text{sgn}(\mathcal{N}(\boldsymbol{x}_i))}{\arg\min} \mid \mathcal{N}_+(\boldsymbol{x}_i) + \mathcal{N}_-(\boldsymbol{x}_i) \mid$ ▷ $\hat{\boldsymbol{x}}$ is not correctly covered by $\mathcal{C}$, and has the smallest confidence.
    $c \leftarrow \frac{1}{2}(\mathcal{N}_+(\hat{\boldsymbol{x}}) - \mathcal{N}_-(\hat{\boldsymbol{x}}))$
    Add the polytope $C := \{\boldsymbol{x} \mid \mathcal{N}_-(\boldsymbol{x}) > -c\}$ in $\mathcal{C}_P$.
    Add the polytope $C := \{\boldsymbol{x} \mid \mathcal{N}_+(\boldsymbol{x}) < c\}$ in $\mathcal{C}_Q$.
    $\mathcal{C} \leftarrow \mathcal{C}_P \cup \mathcal{C}_Q$ ▷ Now, $\hat{\boldsymbol{x}}$ is correctly covered by $\mathcal{C}$.
  end if
until $\mathcal{C}$ becomes a polytope-basis cover of $\mathcal{D}$
Output: $\mathcal{C}$ ▷ The polytope-basis cover of $\mathcal{D}$ derived from $\mathcal{N}$.

The result of Algorithm 3 on a two-layer ReLU network with architecture $2 \xrightarrow{\sigma} 50 \to 1$, which is trained on the swiss roll dataset, is illustrated in Figure 9. Specifically, the algorithm generates a polytope-basis cover of the dataset consists of 68 polytopes ($\mathcal{C}_P$ and $\mathcal{C}_Q$ consists of 34 polytopes, respectively). 14 polytopes in $\mathcal{C}$ is illustrated in 2nd column to 8th column in Figure 9. Proposition C.2 guarantees that Algorithm 3 must terminate in finite time, and produces a polytope-basis cover of $\mathcal{D}$ which has the same accuracy with the given $\mathcal{N}$. Therefore, training a two-layer ReLU network to 100% accuracy on the dataset $\mathcal{D}$, Algorithm 3 allows to derive a polytope-basis cover of the given dataset.

There are some pros and cons in Algorithm 3. The advantages of the algorithm are 1. it can be applied to arbitrary two-layer ReLU networks, and 2. it does not modify the trained network. Therefore, it unveils the inherent polytope-basis cover and convex polytope structures in trained two-layer ReLU networks. However, two drawbacks of this algorithm are: 1. generally it induces many polytopes in practice, and 2. the number of faces of each polytope is unknown. For a detailed comparison with other algorithms, see Section C.4.

C.3. An Efficient Algorithm to Find a Simple Polytope-Basis Cover

In the preceding subsections, we introduced two algorithms in Sections C.1 and C.2 that extract a polytope-basis cover from trained two-layer or three-layer ReLU networks. However, the results obtained from both algorithms, as illustrated in Figure 4 and 9, still exhibit too many polytopes on the training dataset. As demonstrated for the swiss roll dataset in Figure 1, we have previously shown the existence of a polytope-basis cover comprising only four polytopes (refer to Figure 1 and 2).

In this section, to address the aforementioned issue, we present an efficient algorithm designed to find a polytope-basis cover with a reduced number of polytopes. This algorithm is outlined in Algorithm 4. The key distinction of this algorithm from the previous ones is that it does not derive a polytope cover from a trained network. Instead, it sequentially identifies a convex polytope by training several two-layer ReLU networks defined in (5). Consequently, this algorithm only requires access to the training dataset.

Before we demonstrate the algorithm, we provide modified network and loss functions. Specifically, we consider the following two types of two-layer ReLU networks. Here, $\lambda_{bias} > 0$ is a hyperparameter that enhances the convergences as introduced in (5).

$$\mathcal{T}_+(\boldsymbol{x}) := \lambda_{bias} + \sum_{k=1}^{m} v_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k), \qquad \forall v_k < 0 \tag{10}$$

$$\mathcal{T}_-(\boldsymbol{x}) := -\lambda_{bias} + \sum_{k=1}^{m} v_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k), \qquad \forall v_k > 0 \tag{11}$$

If λ_{bias} is large, then $\mathcal{T}_+$ has a large output value at initialization, and gradient descent optimization is heavily affected by data with 0 labels. A similar situation happens for $\mathcal{T}_-$, and it helps to find a single polytope that contains whole class data. Practically, it is enough to use $\lambda_{bias} = 5$ to find such polytopes.

The modified loss function is defined by

$$L_{BCE,\boldsymbol{\lambda}}(\Theta) := -\frac{1}{|\mathcal{D}_0|}\sum_{y_i=0}\lambda_0 \cdot \ell\left(\texttt{SIG}\circ\mathcal{T}(\boldsymbol{x}_i), y_i\right) - \frac{1}{|\mathcal{D}_1|}\sum_{y_i=1}\lambda_1 \cdot \ell\left(\texttt{SIG}\circ\mathcal{T}(\boldsymbol{x}_i), y_i\right) \tag{12}$$

where $\boldsymbol{\lambda} = (\lambda_0, \lambda_1)$ is the hyperparameter proposed to reinforce to cover a whole data class with specific label. For instance, by using a large value of λ_1, $\mathcal{T}$ can be trained to cover whole data points that have label $y_i = 1$. After successfully configuring the first polytope C_1, we train the second network $\mathcal{T}_2$ to distinguish data points of another data class, $y_i = 0$, in the obtained polytope C_1. Repeating this process alternatively, Algorithm 4 generally provides a polytope-basis cover with a small number of polytopes.

Now, we provide a detailed illustration of the algorithm's functionality with the example displayed in Figure 10. Here we use $\lambda_{bias} = 5$, and $\boldsymbol{\lambda} = (1, 10)$. First, the algorithm trains $\mathcal{T}_1$ on the entire dataset $\mathcal{D}$ using the loss function in (12). After fine-tuning through Algorithm 2, we obtain the first polytope C_1 displayed in Figure 10(b). Note that all orange data points $(\mathcal{D}_1)$ are contained in C_1. Next, the second network $\mathcal{T}_2$ is trained on $(\mathcal{D}_0 \cap C_1) \cup \mathcal{D}_1$ using (12) with $\boldsymbol{\lambda} = (1, 10)$. $\mathcal{T}_2$ is aimed to cover all blue data points $(\mathcal{D}_0)$ within C_1. After training and fine-tuning, we obtain the second polytope C_2 displayed in Figure 10(c). Similarly, we can find the third polytope C_3 by training another network $\mathcal{T}_3$ on $\mathcal{D}_0 \cup (\mathcal{D}_1 \cap C_2)$, displayed in Figure 10(d) and so on. In the example in Figure 10, a total of four polytopes are obtained, and visualized through (b) to (e). Lastly, by Theorem 3.4, we can construct a three-layer ReLU network with architecture $2 \xrightarrow{\sigma} 17 \xrightarrow{\sigma} 4 \to 1$ that can completely classify this swiss roll dataset, based on the obtained polytope-basis cover. The decision boundary of the constructed three-layer ReLU network is illustrated in Figure 10(f).

The key advantage of this algorithm lies in its ability to generate a small number of convex polytopes, in contrast to other algorithms we proposed. As demonstrated in Figure 10, it suggests $2 \xrightarrow{\sigma} 17 \xrightarrow{\sigma} 4 \to 1$ as a feasible architecture on the given swiss roll dataset, which appears to be close to optimal. However, due to the iterative nature of training multiple two-layer networks until achieving a complete polytope-basis cover, it typically requires a longer computation time. A detailed comparison with other algorithms is provided in the subsequent section.

C.4. Comparison of the Proposed Algorithms

Finally, we compare the proposed algorithms (Algorithm 1, 2, 3, and 4). First, we provide theoretical results for the proposed algorithms, where their proofs involve demonstrating how a polytope-basis cover can be explicitly constructed from the results of the algorithms. The detailed proof is available in Appendix E.6.

Proposition C.2. *The following statements hold.*

1. *Let $\mathcal{T}$ be a network produced by repeating Algorithm 1 sufficiently many times. Then, it satisfies both (5) and (7).*
2. *Algorithm 2 must terminate in finite time, and it produces a polytope-basis cover of the training dataset $\mathcal{D}$ that has the same accuracy with the compressed network $\mathcal{N}$.*
3. *Algorithm 3 must terminate in finite time, and it produces a polytope-basis cover of the training dataset $\mathcal{D}$ that has the same accuracy with the given network $\mathcal{N}$.*
4. *If Algorithm 4 terminates in finite time, then it produces a polytope-basis cover of the training dataset $\mathcal{D}$.*

The comparison of proposed algorithms is summarized in Table C.4. Below, we discuss the differences of algorithms for each item in Table C.4.

- **Input network.** Algorithm 2 operates on a pretrained three-layer network outlined in (6). Algorithm 3 necessitates a fully-trained two-layer ReLU network specified by (1). However, Algorithm 4 does not necessitate any pre-existing networks as inputs but generates polytope covers through the training of several two-layer ReLU networks as per (5).

Algorithm 4 An efficient algorithm for finding a polytope-basis cover

Require: training dataset $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_1 = \{(x_i, y_i)\}_{i=1}^n$, hyperparameter $\boldsymbol{\lambda} = (\lambda_0, \lambda_1)$, acc_{th}, λ_{bias}, $width$
$\mathcal{C}_P \leftarrow \emptyset, \mathcal{C}_Q \leftarrow \emptyset$.
$m \leftarrow width$
repeat
  repeat
    Initialize $\mathcal{T}_+$ defined in (10).
    Train $\mathcal{T}_+$ on the dataset $\mathcal{D} \cap \mathcal{C}_P^c$ by gradient descent, under the modified BCE loss (12).
    Fine tune the trained $\mathcal{T}_+$ by Algorithm 2.
    if $\mathcal{T}_+(\boldsymbol{x}) \neq \lambda_{bias}$ for some $\boldsymbol{x} \in \mathcal{D}_1 \cap \mathcal{C}_P^c$ **then**
      $m \leftarrow m + 1$
    end if
  until $\mathcal{T}_+(\boldsymbol{x}) = \lambda_{bias}$ for all $\boldsymbol{x} \in \mathcal{D}_1 \cap \mathcal{C}_P^c$
  $A \leftarrow \{\boldsymbol{x} \in \mathbb{R}^d \mid \mathcal{T}_+(\boldsymbol{x}) = \lambda_{bias}\}$ ▷ This is a polytope covering $\mathcal{D}_1 \cap \mathcal{C}_P^c$
  Add A in $\mathcal{C}_P$
  $m \leftarrow width$
  repeat
    Initialize $\mathcal{T}_-$ defined in (11).
    Train $\mathcal{T}_-$ on $\mathcal{D} \cap \mathcal{C}_Q^c$ by gradient descent, under the modified BCE loss (12).
    Fine tune the trained $\mathcal{T}_-$ by Algorithm 2.
    if $\mathcal{T}_-(\boldsymbol{x}) \neq -\lambda_{bias}$ for some $\boldsymbol{x} \in \mathcal{D}_0 \cap \mathcal{C}_Q^c$ **then**
      $m \leftarrow m + 1$
    end if
  until $\mathcal{T}_-(\boldsymbol{x}) = -\lambda_{bias}$ for all $\boldsymbol{x} \in \mathcal{D}_0 \cap \mathcal{C}_Q^c$
  $A \leftarrow \{\boldsymbol{x} \in \mathbb{R}^d \mid \mathcal{T}_-(\boldsymbol{x}) = -\lambda_{bias}\}$ ▷ This is a polytope covering $\mathcal{D}_0 \cap \mathcal{C}_Q^c$
  Add A in $\mathcal{C}_Q$
  $\mathcal{C} \leftarrow \mathcal{C}_P \cup \mathcal{C}_Q$
until $\mathcal{C}$ becomes a polytope-basis cover of $\mathcal{D}$ with accuracy greater than acc_{th}
Output: $\mathcal{C}$

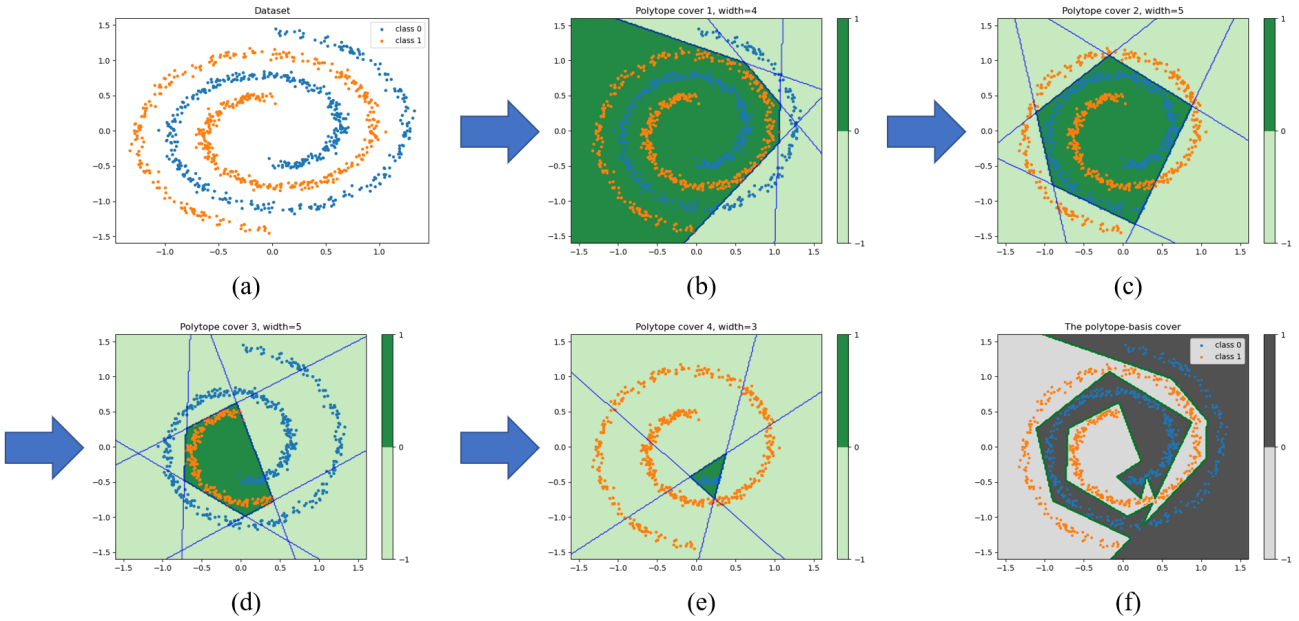


Figure 10. The result of Algorithm 4. For the given dataset (a), the algorithm determines the first polytope C_1 which contains the whole orange class, as shown in (b). In the next step, it obtains the second polytope C_2 which contains the whole blue class inside C_1. Other polytopes are similarly derived and illustrated in (c) to (e). Totally, the algorithm produces a polytope-basis cover consisting of four polytopes. Theorem 3.4 shows that $2 \xrightarrow{\sigma} 17 \xrightarrow{\sigma} 4 \to 1$ is a feasible architecture on this dataset, and the decision boundary of the induced network is drawn in (f).

- **The number of polytopes and faces.** As detailed in Appendix C.2, Algorithm 3 generally yields multiple polytopes to correctly cover all data points in the dataset. Additionally, it does not calculate the precise number of faces for each polytope. In contrast, Algorithm 2 generates a polytope-basis cover by compressing the given three-layer network. Therefore, if the three-layer network $\mathcal{N}$ defined in (6) is the sum of J two-layer networks ($\mathcal{T}_j$), the algorithm is guaranteed to produce a polytope-basis cover consisting of no more than J polytopes. The compressing algorithm and Lemma F.4 provide the exact number of faces for each polytope. Thirdly, Algorithm 4 does not impose any specific lower or upper bounds on the number of polytopes. Similar to Algorithm 2, it also furnishes the exact number of faces for each polytope.

  For instance, we recall the example on the swiss roll dataset: Figures 4, 9, and 10 illustrate that the algorithms yield 16, 68, and four polytopes, respectively. Also note that the last algorithms suggest a feasible architecture of the dataset by $2 \xrightarrow{\sigma} 17 \xrightarrow{\sigma} 4 \to 1$, which looks sufficiently minimal.

- **Theoretical guarantee.** Proposition C.2 guarantees that Algorithm 2 and 3 must terminate in finite time. However, there is no theoretically guarantee for Algorithm 4. Even though, in all our experiments on synthetic and real-world datasets, it always terminates in finite time and produces a complete polytope-basis cover for the given dataset.

- **Time consumption.** Since Algorithm 3 does not require fine-tuning process, it consumes the shortest time among these algorithms. Algorithm 2 requires only fine-tuning process, so it takes a normal amount of time. However, Algorithm 4 tends to spend relatively longer time due to its iterative process of training two-layer networks with fine-tuning until it achieves a complete polytope-basis cover. However, it is essential to note that even for real-world datasets like CIFAR10, the practical execution time is still quite reasonable, typically taking only a few minutes.

- **Accuracy of the obtained cover.** The polytope-basis cover generated by Algorithm 3 preserves the accuracy of the given network (Proposition C.2). In the case of Algorithm 2, the fine-tuning process introduces the possibility of a different accuracy level compared to the original network. Consequently, the final accuracy cannot be determined beforehand. Regarding Algorithm 4, its accuracy is guaranteed to be greater than the given acc_{th} if it terminates within a finite time.

	Algorithm 2	Algorithm 3	Algorithm 4
Input network	a (pretrained) three-layer (6)	any trained two-layer (5)	-
# of obtained polytopes	at most J	generally large	generally small
# of obtained faces	known	unknown	known
Theoretical guarantee for termination	yes	yes	no
Time consumption	normal	short	long
Accuracy of the polytope cover	unknown	same with the given network	$> acc_{th}$ (if it terminates)

Table 3. Comparison of proposed algorithms. All these algorithms generate a polytope-basis cover of the given training dataset, but each of them has its own pros and cons.

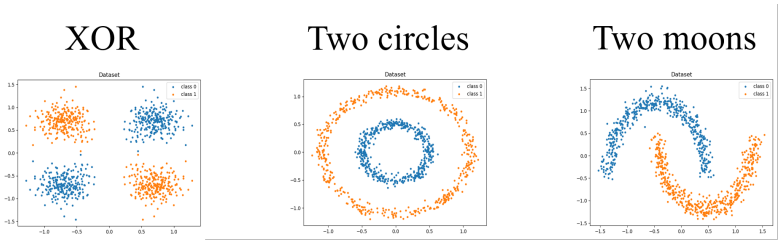


Figure 11. Synthetic datasets - XOR, two circles, and two moons.

Below, we provide additional experimental results of the proposed algorithms on several synthetic datasets, showcasing visual differences among these algorithms. We consider three synthetic datasets: XOR, two circles, and two moons datasets, depicted in Figure 11. The results of algorithms are shown in Figure 12, 13, and 14.

It is easily checked that our proposed algorithms indeed generate polytope-basis covers of the given datasets. In each subfigure, the leftmost column represents the decision boundary and activation boundaries of the obtained networks, and the other columns represent each polytope in the obtained polytope-basis cover. The obtained polytope-basis covers exhibit the geometric characteristics of datasets, and provide feasible architectures of neural networks.

Remark C.3. Figure 14 demonstrates that 'XOR' and 'two circles' datasets have single polytope covers, and 'two moons' dataset can be covered by two polytopes. From the obtained polytope-basis covers, the feasible architectures of these datasets are given by

$$\begin{aligned} \text{XOR}: &\quad 2 \xrightarrow{\sigma} 2 \to 1 \\ \text{Two circles}: &\quad 2 \xrightarrow{\sigma} 4 \to 1 \\ \text{Two moons}: &\quad 2 \xrightarrow{\sigma} 4 \xrightarrow{\sigma} 2 \to 1. \end{aligned}$$

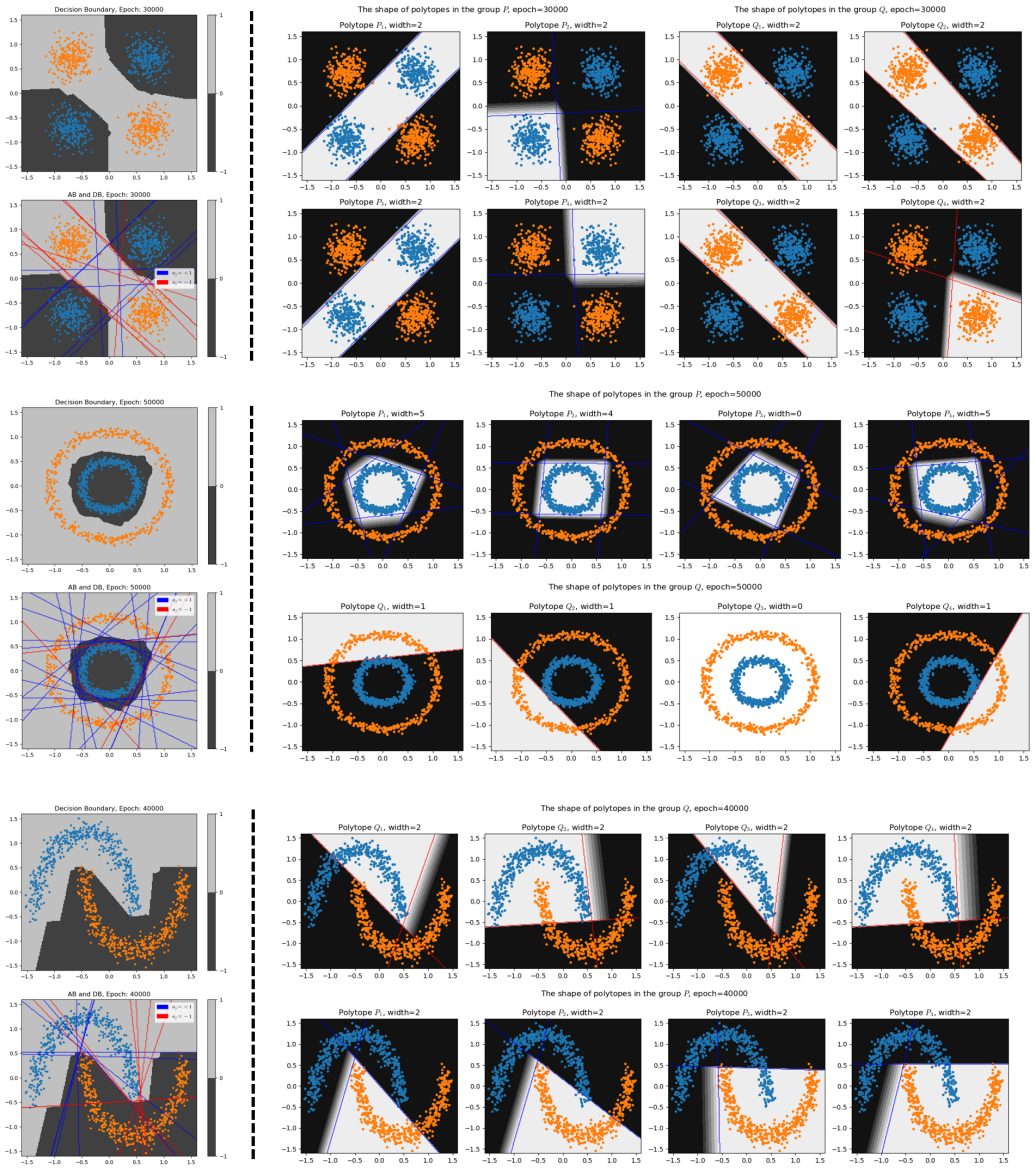


Figure 12. Visualization of Algorithm 2 on the synthetic datasets. These polytope-basis covers are derived from trained three-layer ReLU networks (6) with the architecture $2 \xrightarrow{\sigma} 80 \xrightarrow{\sigma} 8 \to 1$ (i.e., a combination of eight two-layer networks $\mathcal{T}_j$ with $m = 10$ neurons)

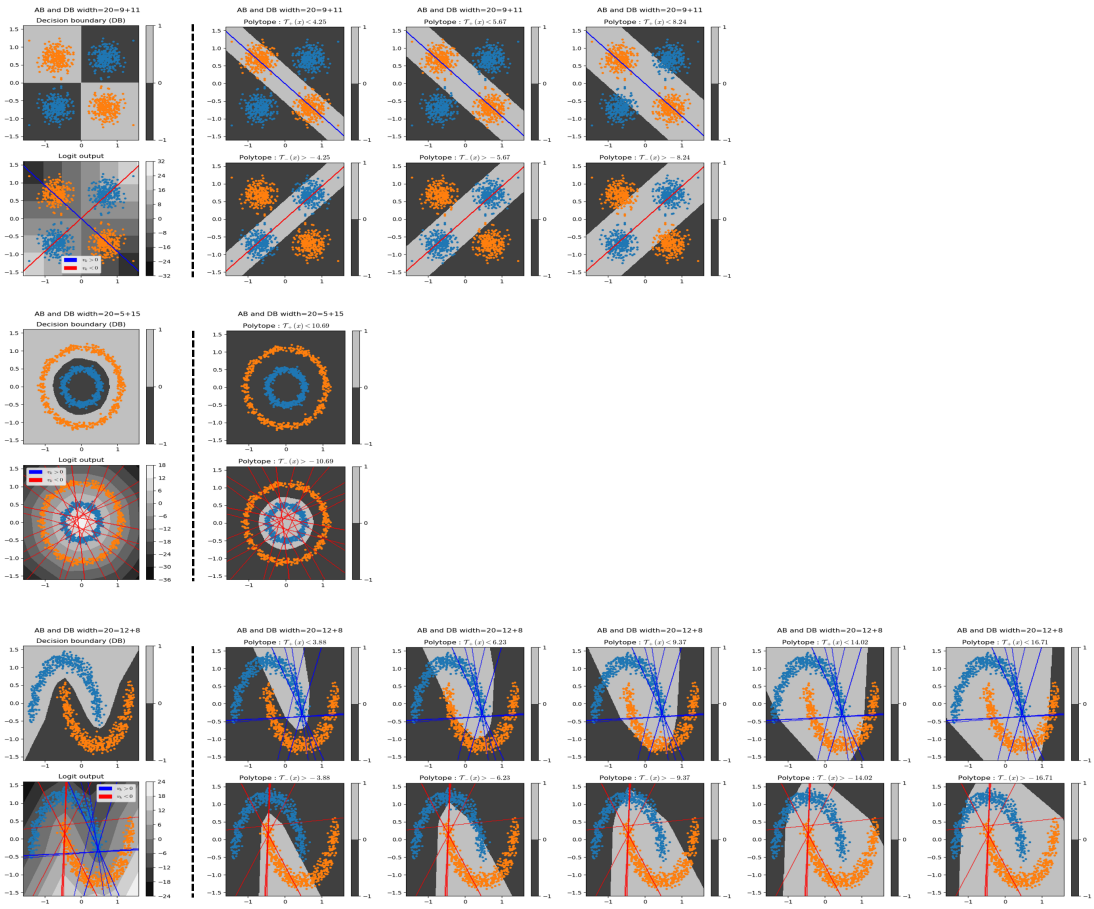


Figure 13. Visualization of Algorithm 3 on the synthetic datasets. These polytope-basis covers are derived from trained two-layer ReLU networks with the architecture $2 \xrightarrow{\sigma} 20 \rightarrow 1$.

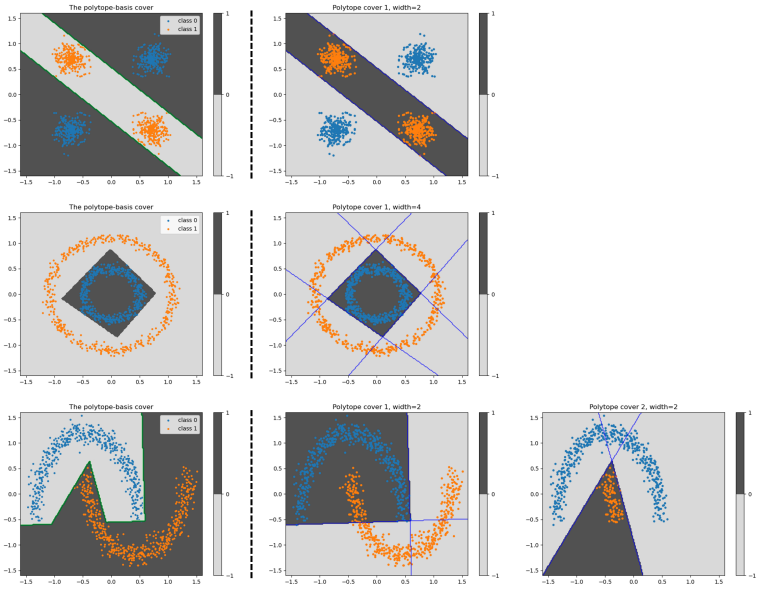
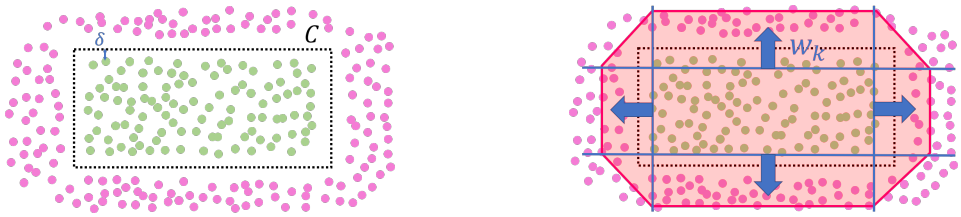


Figure 14. Visualization of Algorithm 4 on the synthetic datasets. Empirically, this algorithm provides the smallest number of polytopes and their faces. The obtained polytope-basis covers can be applied to conclude the feasible architecture of neural networks (Remark C.3).



(a) (b)

Figure 15. Assumptions for the dataset and the network initialization. (a) Dataset $\mathcal{D}$ and a convex polytope C satisfy the Assumption D.2. (b) One example of network initialization satisfying Assumption D.2. The red line displays the decision boundary of $\mathcal{N}$.

D. Convergence on the Proposed Networks

In this section, we investigate whether gradient descent can converge to the networks we proposed in the main text (cf. Theorem 3.4). Specifically, we focus on two-layer ReLU networks, which are the basic building blocks of the constructions. Let $\mathcal{N}$ be a two-layer ReLU neural network defined in (1), where $\Theta := \{v_0\} \cup \{v_k, \boldsymbol{w}_k, b_k\}_{k\in[m]}$ denotes the set of parameters of $\mathcal{N}$. For the given dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, we consider a binary classification problem under the following two loss functions: the mean squared error (MSE) loss and binary cross entropy (BCE) loss. They are defined by

$$L_{MSE}(\Theta) := \frac{1}{2n}\sum_{i=1}^{n}\Big(\sigma\circ\mathcal{N}(\boldsymbol{x}_i) - y_i\Big)^2, \tag{13}$$

$$L_{BCE}(\Theta) := -\frac{1}{n}\sum_{i=1}^{n}\ell\Big(\texttt{SIG}\circ\mathcal{N}(\boldsymbol{x}_i),\ y_i\Big) \tag{14}$$

where $\ell(\mathcal{N}, y) := \mathcal{N}y + (1-\mathcal{N})(1-y)$. Note that we introduce additional activation function σ and $\texttt{SIG}$ to define both loss functions. Specifically, we adopt additional ReLU activation on the output layer to ensure the existence of the zero-loss solution in (13).

We now employ the notion of 'polyhedrally separable' dataset from the learning theory (Astorino & Gaudioso, 2002; Manwani & Sastry, 2010), which is a special case of polytope-basis cover; when a given dataset can be separated by only one convex polytope as depicted in Figure 15 (a).

Definition D.1. We say that the dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i\in[n]}$ is *polyhedrally separable by* C if there exists a convex polytope C such that $\boldsymbol{x}_i \in C$ if and only if $y_i = 1$ for all $i \in [n]$.

We further introduce two notations. First, for a convex polytope C composed of m faces, we denote its k-th face by ∂C_k. Similarly, $\partial^2 C_k$ denotes the boundary of ∂C_k, which refers to the 'edge' part of C. Second, for a set $A \subset \mathbb{R}^d$, $\#(A) := |\{\boldsymbol{x}_i \in \mathcal{D} \mid \boldsymbol{x}_i \in A\}|$ denotes the number of data points $\boldsymbol{x}_i \in \mathcal{D}$ in A. We further need the following assumptions on the dataset $\mathcal{D}$ and network initialization.

Assumption D.2 (Dataset and initialization assumptions)**.** *Suppose the dataset $\mathcal{D}$ is polyhedrally separable by a convex polytope C, which consists of m faces and strictly contains the origin point. Let $\delta > 0$ be the minimum distance between $\boldsymbol{x}_i$ and ∂C, and l_k be the distance between ∂C_k and the origin point. Then, there exist constants $\rho, R > 0$ such that for any $k \in [m]$ and $\delta < r < R$,*

$$\#\Big(\mathcal{B}_{2r}(\partial^2 C_k)\Big) \le \rho\,\#\Big(\mathcal{B}_{r-\delta}(\partial C_k)\Big). \tag{15}$$

Furthermore, the parameters $\{(\boldsymbol{w}_k, b_k, v_k)\}_{k\in[m]}$ of a two-layer ReLU network $\mathcal{N}$ defined in (1) are initialized such that $\boldsymbol{w}_k$ are normal to ∂C_k with outward direction, and satisfying

$$l_k - R \;<\; l_k + \frac{v_0}{v_k\,\|\boldsymbol{w}_k\|} \;<\; -\frac{b_k}{\|\boldsymbol{w}_k\|} \;<\; l_k. \tag{16}$$

The dataset assumption (15) suggests that the data points in the set $\mathcal{B}_r(\partial C)$ for small r are predominantly located in close proximity to the faces of the polytope C, rather than its corners (Figure 15(a)). The network initialization assumption

implies that every neuron $(\boldsymbol{w}_k, b_k)$ of $\mathcal{N}$ is initialized near ∂C_k as described in Figure 15(b). With these assumptions, we can prove the existence of a discrete path that strictly decrease the loss value to zero.

Theorem D.3. *Suppose the dataset $\mathcal{D}$ and the two-layer network $\mathcal{N}$ in (1) satisfy Assumption D.2. Then,*

1. *for the MSE loss defined in (13), suppose v_0 is initialized such that*

$$\frac{\rho}{1-\rho}\frac{4m\rho R^2}{\delta^2} < v_0 < 1. \tag{17}$$

Then, with step size $\eta < \min\left\{\frac{2}{\delta}, \frac{2}{mR}, \frac{4\rho m}{(1-\rho)R}\right\}$, there exists a discrete path that the loss value (13) strictly decreases to zero.

2. *For the BCE loss defined in (14), suppose v_0 is initialized such that*

$$0 < v_0 < \log\left(\frac{(1-\rho)\delta}{4\rho R} - 1\right). \tag{18}$$

Then, with step size $\eta < \min\left\{1, \frac{4\rho R}{(1-\rho)\delta^2}\right\}$, there exists a discrete path that the loss value (14) strictly decreases to zero.

The proof of this theorem can be found in Appendix E.7. Theorem D.3 asserts that the loss landscape has no local minima within this initialization region. If local minima did exist in this region, it would contradict the presence of a loss-decreasing path from the local minima to the global minima. Consequently, this theorem provides strong evidence for the convergence of gradient descent to the global minima.

However, it is important to note that there may still be saddle points where gradient descent could potentially get stuck. In such cases, we believe that stochastic (noisy) gradient descent may help in escaping these saddle points and eventually converging to the global minimum, which has zero error on the training dataset $\mathcal{D}$. Therefore, the initialization conditions described in Assumption D.2, (17), and (18) can be understood as necessary conditions for ensuring that the gradient method converges to the global minimum.

Lastly, we mention that Theorem D.3 can be easily extended to the three-layer network (7) proposed in Theorem 3.4. For such a three-layer network $\mathcal{N}$, Theorem D.3 can be applied to each two-layer subnetwork $\mathcal{T}_j$ to generate the loss-decreasing path. By combining all these paths, a unified loss-decreasing path for $\mathcal{N}$ is formed. This extension underscores the robustness and generality of the convergence properties demonstrated, ensuring that even more complex network architectures retain the desirable characteristics of gradient descent convergence.

E. Proofs

E.1. Proof of Proposition 3.1.

The proof of Proposition 3.1 is divided into two parts. Firstly, we prove the upper bound by constructing the desired neural network. Secondly, we show the lower bound of widths.

E.1.1. The upper bound in Proposition 3.1.

For the given convex polytope $\mathcal{X}$, let $h_1, \cdots, h_m$ be its m hyperplanes enclosing C. Let $\boldsymbol{w}_k$ be the unit normal vector of the k-th hyperplane h_k oriented inside C, as illustrated in Figure 2(a). Then the equation of the k-th hyperplane h_k is given by $h_k : \{\boldsymbol{x} \mid \boldsymbol{w}_k^\top \boldsymbol{x} + b_k = 0\}$ for some $b_k \in \mathbb{R}$. Let A_k be the intersection of the hyperplane h_k and C, which is a face of the polytope C. Let $\boldsymbol{x}$ be any point strictly contained in C. Since $\boldsymbol{w}_k$ is a unit normal vector, $\boldsymbol{w}_k^\top \boldsymbol{x} + b_k$ refers the distance between the hyperplane h_k and the point $\boldsymbol{x}$. Therefore, the d-dimensional Lebesgue measure of C is computed by

$$\mu_d(C) = \frac{1}{d}\sum_{k=1}^{m}(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)\cdot \mu_{d-1}(A_k) \tag{19}$$

where μ_{d-1} and μ_d refer the $(d-1)$ and d-dimensional Lebesgue measures, respectively. Note that (19) comes from the volume formula of a convex polytope, which states that the volume is the sum of volume of m pyramids. Then LHS of

(19) is constant, which does not depend on the choice of $\boldsymbol{x} \in \mathbb{R}^d$. Now, we define a two-layer ReLU network $\mathcal{T}$ with the architecture $d \xrightarrow{\sigma} m \to 1$ by

$$\mathcal{T}(\boldsymbol{x}) := 1 + M\left(\mu_d(C) - \sum_{k=1}^{m} \frac{1}{d}\mu_{d-1}(A_k) \cdot \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)\right) \tag{20}$$

where $M > 0$ is a constant would be determined later. Note that we have $\mathcal{T}(\boldsymbol{x}) = 1$ for $\boldsymbol{x} \in C$ from the construction. However, considering the negative sign, it is worth noting that the equation (19) also holds for $\boldsymbol{x} \notin C$. In particular, for $\boldsymbol{x} \notin C$, (20) deduces

$$\begin{aligned}
\mathcal{T}(\boldsymbol{x}) &= 1 + M\left(\mu_d(C) - \sum_{k=1}^{m} \frac{1}{d}\mu_{d-1}(A_k) \cdot \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)\right) \\
&= 1 + M\left(\mu_d(C) - \sum_{k=1}^{m} \frac{1}{d}\mu_{d-1}(A_k) \cdot (\boldsymbol{w}_k^\top \boldsymbol{x} + b_k) + \sum_{\{k\,:\,\boldsymbol{w}_k^\top \boldsymbol{x} + b_k < 0\}} \frac{1}{d}\mu_{d-1}(A_k) \cdot (\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)\right) \\
&= 1 + M \sum_{\{k\,:\,\boldsymbol{w}_k^\top \boldsymbol{x} + b_k < 0\}} \frac{1}{d}\mu_{d-1}(A_k) \cdot (\boldsymbol{w}_k^\top \boldsymbol{x} + b_k) \\
&< 1.
\end{aligned}$$

Therefore, we conclude that

$$\begin{aligned}
\mathcal{T}(\boldsymbol{x}) &= 1 \qquad \text{if } \boldsymbol{x} \in C, \\
\mathcal{T}(\boldsymbol{x}) &< 1 \qquad \text{otherwise.}
\end{aligned}$$

Lastly, we determine the constant M in $\mathcal{T}$ to satisfy the remained property. For the given $\varepsilon > 0$, consider the closure of complement of the $\frac{\varepsilon}{2}$-neighborhood of C; $D := \overline{\left(\mathcal{B}_{\varepsilon/2}(C)\right)^c}$. Then the previsous result shows that

$$\frac{1}{M}(\mathcal{T}(\boldsymbol{x}) - 1) = \mu_d(C) - \sum_{k=1}^{m} \frac{1}{d}\mu_{d-1}(A_k) \cdot \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k) \tag{21}$$

is bounded above by 0. Furthermore, (21) is continuous piecewise linear, and has the maximum 0 if and only if $\boldsymbol{x} \in C$. Since D is closed and (21) is strictly bounded above by 0 on D, (21) has the finite maximum $M' < 0$ on D.

$$\frac{1}{M}(\mathcal{T}(\boldsymbol{x}) - 1) \le M' < 0 \qquad \text{for } \boldsymbol{x} \in D.$$

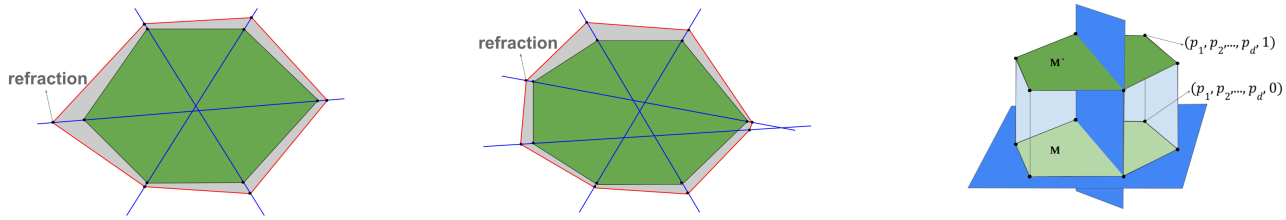
Now, choose M to satisfy $M > -\frac{1}{M'}$. Then if $\boldsymbol{x} \notin B_\varepsilon(C)$, we have $\boldsymbol{x} \in D$, thus

$$\begin{aligned}
\mathcal{T}(\boldsymbol{x}) &= 1 + M\left(\mu_d(C) - \sum_{k=1}^{m} \frac{1}{d}\mu_{d-1}(A_k) \cdot \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)\right) \\
&\le 1 + M \cdot M' \\
&< 0.
\end{aligned}$$

Therefore, we have constructed a two-layer ReLU network $\mathcal{T}$ with the structure $d \xrightarrow{\sigma} m \to 1$ such that

$$\begin{aligned}
\mathcal{T}(\boldsymbol{x}) &= 1 \qquad \text{if } \boldsymbol{x} \in C, \\
\mathcal{T}(\boldsymbol{x}) &< 1 \qquad \text{if } \boldsymbol{x} \in C^c, \\
\mathcal{T}(\boldsymbol{x}) &< 0 \qquad \text{if } \boldsymbol{x} \notin B_\varepsilon(C).
\end{aligned}$$

This completes the proof on the upper bound. Lastly, the minimality of depth comes from the fact that a linear function cannot be a feasible architecture on C. □



(a) (b) (c)

Figure 16. Proof of Proposition 3.1. (a) Given a hexagon which is approximated by three hyperplanes given with blue lines and one polytope in the second layer given with red hexagon. (b) Given a heptagon which is approximated by four hyperplanes given with blue lines and one polytope in the second layer given with red heptagon. (c) Hexagon has been extended to the 3-dimensional polytope by incrementing the number of faces by 2 with some potential first layer neurons (blue hyperplanes).

E.1.2. The lower bound in Proposition 3.1.

Before proving the lower bound, we introduce a definition of *refraction points.* Let $\mathcal{N}(\boldsymbol{x}) := \sigma(v_0 + \sum_{i=1}^{d_1} v_i \sigma(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i))$ be a two-layer network with architecture $d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} 1$. Then the set of refraction points is defined by

$$\{\boldsymbol{x} \in \mathbb{R}^d \mid \mathcal{N}(\boldsymbol{x}) = 0 \quad \text{and} \quad \boldsymbol{w}_i^\top \boldsymbol{x} + b_i = 0 \quad \text{for some } i \in [l].\}$$

In other words, it is the point where the boundary of a linear partition is 'refracted'.

Lower bound for $d = 2$ using only refraction points for $k = 1$. Assume that we are given a convex m-gon to approximate. Considering the fact that we can approximate the neural network arbitrarily close, we can see that the approximated second layer, i.e. neural network should have at least m refraction points in order to get the shape of a polygon. However, if we look from the perspective of first layer neurons, each line has at most 2 intersection with m-gon and it implies that each first layer neuron (or line) can contribute at most 2 meaningful refraction points for the next layer. If we combine above two results, we can obtain that there should be at least $\lceil \frac{m}{2} \rceil$ number of neurons in the first layer, in other words $d_1 \geq \lceil \frac{m}{2} \rceil$. For example, Figure 16(a) and (b) demonstrates the refraction points along with potential first layer hyperplanes (blue lines) and converged polytope at the second layer (red polygon) for hexagon and heptagon, respectively.

Proof of the optimality when $d = 2$ for $k = 1$. First of all, we should note that on $\mathbb{R}^2$, any two convex m-gon's given on the general position (i.e., assume sides are non-parallel mutually) can be approximated by the same neural network (same d_1 value) considering the fact that we can find an approximator for each given error value ϵ. It implies that we can take optimal possible number of neurons in the first layer, which we will denote by $f(m)$ for any given convex m-gon. Let's prove that $f(m) = \lceil \frac{m}{2} \rceil$ for $m \geq 5$ along with $f(3) = f(4) = 3$. The cases $m = 3, 4$ should be handled separately, because we trivially need at least $d + 1 = 3$ hyperplanes for any shape (Lemma F.1), so we start the base case from $m \geq 5$ for $d = 2$.

According to the Lemma F.1, it is appearent that for any m, the inequality $f(m) \geq 3$ should hold trivially. But if we consider the Figure 16(a), we can observe that one can approximate any hexagon with 3 hyperplanes. Apparently, for any pentagon, quadrilateral, and triangle, we can find a corresponding hexagon to include it as a subfigure and rest of the additional vertices of this hexagon can be shrinked to be almost non-exist. It implies that, same number of hyperplanes approximating hexagon can also approximate the polygons with $m \leq 5$. This final result yields that $f(m) \leq 3$ for $m \leq 6$. If we combine these two findings we can get a nice optimality at the fundamental cases, in other words $f(m) = 3$ for $m \in \{3, 4, 5, 6\}$.

Now, assume the contrary that $f(m) \leq \lceil \frac{m}{2} \rceil - 1$, then it is apparent that there is at least one neuron which contributes to the refraction point of at least 2 vertices (i.e. exactly 2 vertices considering previous discussion). Now, if we remove the chosen neuron and the associated 2 vertices and their edges, then the resulting $(m-2)$-gon will be approximated by $f(m) - 1$ number of neurons, which implies that $f(m) - 1 \geq f(m-2)$. Proceeding with the same argument, we can arrive at the conclusion that $f(5)$ or $f(6) \leq 2$; however, we have already proven that $f(5)$ and $f(6)$ are indeed 3. So, the contradiction at the base case yields the result that $f(m) \geq \lceil \frac{m}{2} \rceil$.

For the base cases $m = 5, 6$, we have already demonstrated that $f(5) = f(6) = 3$. Now, take any m-gon which has been approximated well with $f(m) = \lceil \frac{m}{2} \rceil$ neurons. Let's add two new vertices to form a new convex polygon with $(m+2)$ vertices, where the newly added vertices are not adjacent. Then if we add one new neuron which is the line passing through those two points, we can observe that if given $f(m)$ number of neurons approximate m-gon, then $f(m) + 1$ can approximate

$(m+2)$-gon by triggering 2 new refraction points. This inductive argument $f(m+2) \leq f(m)+1$ yields the result that if we start from $f(5) = f(6) = 3$, we can reach a conclusion that $f(m) \leq \lceil \frac{m}{2} \rceil$. However, we have already shown $f(m) \geq \lceil \frac{m}{2} \rceil$ in the proof above. Therefore, the result follows immediately that the optimal number of neurons in the first hidden layer to approximate any convex polygon with m vertices is $\lceil \frac{m}{2} \rceil$ for $m \geq 5$ and $f(3) = f(4) = 3$. □

Lower bound for arbitrary dimension d for $k = 1$. Now we will apply simple induction on the dimensionality to prove the general case for lower bound on the number of first hidden layer neurons. Essentially, we will construct a d-dimensional object for $d \geq 2$ such that, one needs at least $d_1 \geq \lceil \frac{m}{2} \rceil + (d-2)$ number of neurons (hyperplanes) to approximate the convex polytope with l faces. We will proceed with an inductive argument,we have already provided a proof for the base case of $d = 2$ that $d_1 \geq \lceil \frac{m}{2} \rceil$.

Inductive step. Suppose that we have a d-dimensional convex polytope M with v number of vertices and m number of faces such that the following inequality should hold: $d_1 \geq \lceil \frac{m}{2} \rceil + (d-2)$. Let's consider the object on $(d+1)$-dimensional space by adding new entry at the end of each coordinate, i.e. any point $(p_1, p_2, ..., p_d)$ on the object will be replaced by the point $(p_1, p_2, ..., p_d, 0)$. Then consider the new shape M' formed by considering the extension of convex polytope M on $(d+1)$-dimensional space with all the points from $\{p = (p_1, p_2, ..., p_d, x) \mid \forall x = [0, 1] \text{ and } (p_1, p_2, ..., p_d) \in M\}$. Then M' will lie on $(d+1)$-dimensional space and it will have $2v$ vertices and $(m+2)$ number of faces, of which m will be determined by the extensions of faces of polytope M along with two faces from M and its duplicate M'. We can also observe the inductive incrementing idea through the Figure 16(c), in which polytope M at $d = 2$ with 6 faces has been extended to the 3-dimensional polytope with $6 + 2 = 8$ faces.

If we take a closer look at this construction, we can observe that if we take the intersection of each hyperplane from d_1 neurons designed for the approximation of M' and polytope M, then those intersections will be hyperplane for d-dimensional polytope M. It implies that in order to approximate m faces of new polytope, the intersections themselves should approximate the m faces of M. Furthermore, other than those m faces formed by faces of previous polytope M, we should also consider the other 2 faces, namely M and its duplicate M'. Those two hyperplanes will require additional 2 neurons to trigger new refraction points for their approximation. Therefore, there should be at least $d_1 \geq \lceil \frac{m}{2} \rceil + (d-2) + 2$ number of neurons, in which right-hand-side can be equivalently written as $\lceil \frac{m}{2} \rceil + d = \lceil \frac{m+2}{2} \rceil + (d+1-2)$. So, we were able to prove that to have a neural network of the form $d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} 1$ to approximate the convex polytopes with m faces arbitrarily close, then universally the value of d_1 should at least $\lceil \frac{m}{2} \rceil + (d-2)$.

The result can be also stated that for all $m \geq 2d+1$ one can find a d-dimensional convex polytope with l faces such that the minimum required number neurons in the first hidden layer is at least $\lceil \frac{m}{2} \rceil + (d-2)$. For $m = 2d-1$ and $l = 2d$, the lower bound becomes $d_1 \geq 2d-1$ as we have already described that $f(3) = f(4) = 3$. The lower bound on m comes from the fact that the construction has an inductive fashion to create a new object from previous one by adding 2 new faces in each step. For the rest of the values of number of faces m, i.e. $m < 2d-1$, one can consider the trivial bound of $d+1$. More strongly, in case of 2-dimensional space, the statement has been proven for all convex polygons that optimal value is indeed $d_1 = \lceil \frac{m}{2} \rceil$.

Generalization to arbitrary dimension d and depth k. In the context of manifold representations shaped as convex polytopes with varying depths, we employ an inductive approach to establish lower bounds. Leveraging prior findings on two-layer neural networks, we derive insights applicable to arbitrary dimensions d. For any given hyperplane in this setting, a maximum of two distinct refraction points can be identified, a premise that underpins our assumption that each second-layer neuron constitutes a polytope comprised of faces, with no more than twice the number of hyperplanes as the first layer. This result has also been used in the proof of Theorem 3.6 and we can observe the trend from the Figure 18(c).

We transform the general case by considering the facets of second or higher-layer neurons as first-layer neurons (hyperplanes), which represent potential refraction points. This transformation allows us to reduce the problem to a two-layer network by decreasing the depth while augmenting the number of hyperplanes in the first layer. More precisely, for a given feasible architecture of the form $d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} d_2 \xrightarrow{\sigma} \cdots \xrightarrow{\sigma} d_k \to 1$, each of d_2 number of second layer neurons can contribute at most $2d_1$ hyperplanes along with the d_1 hyperplanes in the first layer, which implies total of $d_1 + 2d_1d_2 = d_1(2d_2+1)$ hyperplanes. In other words, we can transform the above network to another network $d \xrightarrow{\sigma} d_1(2d_2+1) \xrightarrow{\sigma} d_3 \xrightarrow{\sigma} \cdots \xrightarrow{\sigma} d_k \to 1$ by reducing the depth by 1. By applying the similar process as above, we assert that initial architecture can be effectively transformed into a more robust architecture, $d \xrightarrow{\sigma} d_1(2d_2+1)(2d_3+1)\ldots(2d_k+1) \xrightarrow{\sigma} 1$.

Consequently, we can generalize lower bounds for convex polytope representations of varying depths, drawing on the

insights gained from our two-layer formulation. The ultimate result yields a powerful lower bound as

$$d_1 \cdot \prod_{j=2}^{k}(2d_j+1) \geq \begin{cases} \lceil \frac{m}{2} \rceil + (d-2), & \text{if} \quad m \geq 2d+1, \\ 2d-1, & \text{if} \quad m = 2d-1, 2d, \\ d+1, & \text{if} \quad m < 2d-1. \end{cases}$$

Moreover, the above result is particularly optimal for the case of convex polygons in two dimensions, where $d = 2$ and $k = 1$, as previously discussed. □

Remark E.1. Rigorously, the lower bound on the network width proposed in Proposition 3.1 can also be understood as the maximum number of faces that a given network can approximate with its polytope. Conversely, to achieve the UAP and approximate any polytope with m faces, the width of the first hidden layer must be greater than or equal to m. This is precisely explained in Proposition F.6, which proves that the upper bound proposed in Proposition 3.1 is tight and sufficient to satisfy the UAP.

E.2. Proof of Theorem 3.4

By Proposition 3.1, for each set $A \in \mathcal{C} = \{P_1, \cdots, P_{n_P}, Q_1, \cdots, Q_{n_Q}\}$, we can construct a two-layer ReLU network $\mathcal{T}_A$ with the architecture $d \xrightarrow{\sigma} m_A \xrightarrow{\sigma} 1$ such that $\mathcal{T}_A(\boldsymbol{x}) = 1$ for $\boldsymbol{x} \in A$ and $\mathcal{T}_A(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \notin B_\varepsilon(A)$, where m_A denotes the number of faces of A. Let $a_i := \mathcal{T}_{P_i}$ for $i \in [n_P]$ and $b_j := \mathcal{T}_{Q_j}$ for $j \in [n_Q]$. Define the output layer by

$$\mathcal{N}(\boldsymbol{x}) = \sum_{i=1}^{n_P} a_i - \sum_{j=1}^{n_Q} b_j - \frac{1}{2}.$$

Then, we obtain the desired network $\mathcal{N}$ which has the architecture $d \xrightarrow{\sigma} m \xrightarrow{\sigma} (n_P + n_Q) \to 1$. □

E.3. Proof of Theorem 3.5

Let $X_1, X_2, \cdots, X_k$ be the k facets of $\mathcal{X}$. For each facet X_i, we can construct a two-layer ReLU network $\mathcal{T}_i$ such that $\mathcal{T}_i(\boldsymbol{x}) = 1$ for $\boldsymbol{x} \in X_i$ and $\mathcal{T}_i(\boldsymbol{x}) < 0$ for $\boldsymbol{x} \notin \mathcal{B}_\varepsilon(X_i)$ by Lemma F.1. Then Theorem 3.4 gives a neural network $\mathcal{N}$ with the architecture $d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} k \to$ with $d_1 = k(d+1)$, therefore, it is a feasible architecture on $\mathcal{X}$. The remaining goal is to reduce the width of the first layer d_1.

From the construction, we recall that $d_1 \leq k(d+1)$ comes from the fact where each simplex X_i is covered by a d-simplex which has $(d+1)$ hyperplanes. Now consider two j-simplices in $\mathbb{R}^d$. If $2j+2 \leq d+1$, then we can connect all points of the two j-simplices in $\mathbb{R}^d$, and it becomes a $(2j+2)$-simplex Δ^{2j+2}. Now construct a d-simplex Δ^{d+1} by choosing $(d+1)-(2j+2)$ points in $\mathcal{B}_\varepsilon(\Delta^{2j+2})$, whose base is this $(2j+2)$-simplex. Then, by adding two distinguishing hyperplanes at last, we totally consume $(d+3)$ hyperplanes to separate two j-simplices.

Now we apply this argument to each pair of two simplices. The above argument shows that two j-simplices separately covered by $2(d+1)$ hyperplanes can be re-covered by $(d+3)$ hyperplanes if $j \leq \lfloor \frac{d-1}{2} \rfloor$, which reduces $(d-1)$ number of hyperplanes. In other words, we can save $(d-1)$ hyperplanes for each pair of two j-simplices whenever $j \leq \lfloor \frac{d-1}{2} \rfloor$. This provides one improved upper bound of d_1:

$$d_1 \quad \leq \quad k(d+1) - (d-1) \left\lfloor \frac{1}{2} \sum_{j=0}^{\lfloor \frac{d-1}{2} \rfloor} k_j \right\rfloor. \tag{22}$$

Now, we consider another pairing. For $0 \leq j \leq J$, $\mathcal{X}$ has k_j j-simplex facets. Since each j-simplex has $(j+1)$ points, in particular, a d-simplex consists of $(d+1)$-points. Therefore, all points in $\left\lfloor \frac{d+1}{j+1} \right\rfloor$ many j-simplices can be contained in one d-simplex. In this case, these j-simplices can be covered by adding $\left\lfloor \frac{d+1}{j+1} \right\rfloor$ hyperplanes more. Thus if we have k_j many j-simplices, then the required number of hyperplanes to separately encapsulate the j-simplices is less than or equal to

$$\#\,(\text{the number of } d\text{-simplices}) \cdot \#\,(\text{the required number of hyperplanes in each } d\text{-simplex})$$

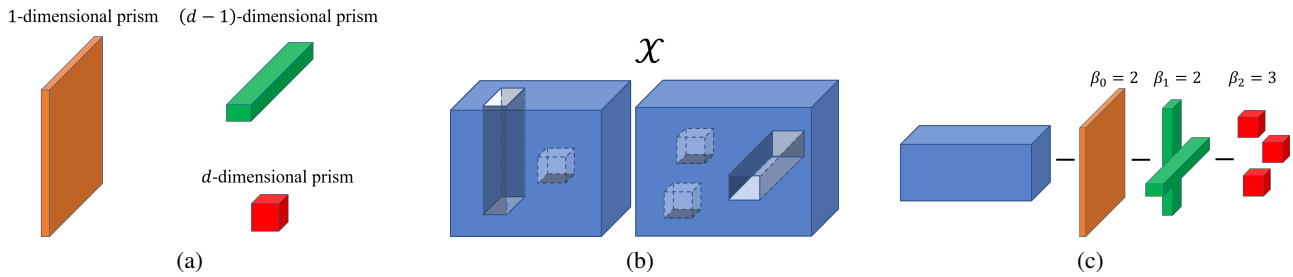


Figure 17. The Proof of upper bounds in Theorem 3.6. (a) Some examples of prismatic polytopes. (b) $\mathcal{X}$ is a topological space satisfying the assumption in Theorem 3.6. (b) The removed prismatic polytopes from $\mathcal{X}$ are displayed. Theorem 3.6 demonstrates that $3 \xrightarrow{\sigma} 34 \xrightarrow{\sigma} 7 \to 1$ is a feasible architecture on $\mathcal{X}$.

$$
\begin{aligned}
&= \left(\left\lfloor \frac{k_j}{\left\lfloor \frac{d+1}{j+1} \right\rfloor} \right\rfloor + 1 \right) \cdot \left(d + 1 + \left\lfloor \frac{d+1}{j+1} \right\rfloor \right) \\
&\leq \left(k_j \frac{j+1}{d-j} + 1 \right) \cdot \left(d + 1 + \frac{d+1}{j+1} \right) \\
&< (d+1) \left(\frac{j+2}{j+1} \right) \left(k_j \frac{j+1}{d-j} + 1 \right) \\
&= (d+1) \left(k_j \frac{j+2}{d-j} + \frac{j+2}{j+1} \right)
\end{aligned} \tag{23}
$$

where the inequality is reduced from the property of the floor function: $a - 1 < \lfloor a \rfloor \leq a < \lfloor a \rfloor + 1$ for any $a \in \mathbb{R}$. Then another upper bound of d_1 is obtained by applying (23) for all $j \leq J$. However, further note that (23) is greater than the known upper bound $k(d+1)$ if $J > \frac{d}{2}$; the sharing of covering simplex is impossible in this case. Therefore, the upper bound of d_1 is given by

$$
\begin{aligned}
d_1 \quad &\leq (d+1) \sum_{j \leq \frac{d}{2}} \left(k_j \frac{j+2}{d-j} + \frac{j+2}{j+1} \right) + (d+1) \sum_{j > \frac{d}{2}} k_j \\
&= (d+1) \left[\sum_{j \leq \frac{d}{2}} \left(k_j \frac{j+2}{d-j} + \frac{j+2}{j+1} \right) + \sum_{j > \frac{d}{2}} k_j \right]
\end{aligned} \tag{24}
$$

To sum up, from (22) and (24), we get the desired result

$$
d_1 \leq \min \left\{ k(d+1) - (d-1) \left\lfloor \frac{1}{2} \sum_{j=0}^{\lfloor \frac{d-1}{2} \rfloor} k_j \right\rfloor, \ (d+1) \left[\sum_{j \leq \frac{d}{2}} \left(k_j \frac{j+2}{d-j} + \frac{j+2}{j+1} \right) + \sum_{j > \frac{d}{2}} k_j \right] \right\}.
$$

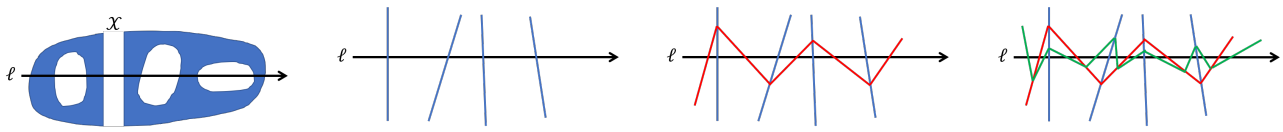
□

E.4. Proof of Theorem 3.6.

The proof consists of two parts: we prove the upper bound first, and second, we show the lower bound.

E.4.1. The upper bound in Theorem 3.6.

We establish a terminology about the shape of prismatic polytopes. A prism in $\mathbb{R}^3$ consists of a 'base' and 'height' dimensions, and we generalize it to high dimensional prisms. We define a **k-dimensional prismatic polytope** in $\mathbb{R}^d$ as a topological space homeomorphic to $K \times \mathbb{R}^{d-k}$, where $\times$ denotes the Cartesian product an $K \subset \mathbb{R}^k$ is a compact set which is the 'base' of the prism. A **bounded k-dimensional prismatic polytope** is a intersection of a k-dimensional prismatic polytope and a bounded convex polytope.



(a) (b) (c) (d)

Figure 18. Proof of lower bounds in Theorem 3.6. (a) Consider a topological space $\mathcal{X}$ whose holes intersect with a straight line ℓ. (b) d_1 neurons in the first hidden layer of $\mathcal{N}$ (blue color) have at most d_1 intersection points with ℓ. (c) A neuron in the second layer (red color) has at most (d_1+1) intersection points with ℓ. (d) Similarly, a neuron in the third layer (green color) has at most $d_2(d_1+1)+1$ intersection points with ℓ.

Roughly speaking, an 1-dimensional prismatic polytope is a thick 'hyperplane' in $\mathbb{R}^d$, $(d-1)$-dimensional prismatic polytope is a long 'rod,' and a d-dimensional prismatic polytope is just a convex polytope, as shown in Figure 17(a). Then, for $k = 1, 2, \cdots, d$, removing a k-dimensional prism from $\mathcal{X}$ generates a k-dimensional hole, which increases the $(k-1)$-th Betti number β_{k-1}. In Figure 17(b), we provide an example of such cover $\mathcal{X}$ in $\mathbb{R}^3$. $\mathcal{X}$ is described by subtracting six prismatic polytopes from a large bunoid in $\mathbb{R}^3$. In this case, the subtracted prismatic polytopes can be understood as a bounded prismatic polytopes with six faces.

Now, we prove the theorem. Recall that the polytope-basis cover $\mathcal{X}$ can be described as a subtraction of $\sum_{k=0}^{d-1}\beta_k$ convex polytopes from a sufficiently large convex polytope with m faces. Applying Theorem 3.4, we get

$$d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} \left(\sum_{k=0}^{d-1}\beta_k\right) \to 1$$

is an upper bound of a feasible architecture , where

$$\begin{aligned} d_1 &\leq m + m\cdot\left(\sum_{k=0}^{d-1}\beta_k - 1\right) \\ &= m\left(\sum_{k=0}^{d-1}\beta_k\right). \end{aligned}$$

This upper bound of d_1 can be further reduced. For $1 \leq k < d$, β_k means the number of k-dimensional holes in $\mathcal{X}$, which was made by punching out a k-dimensional prismatic polytope. Since k-dimensional prisms have $2k$ faces that penetrate $\mathcal{X}$, we can reduce $2(d-k-1)$ number of hyperplanes that cover the hole. When $k=0$, it is easy to check that $2(\beta_0-1)$ hyperplanes are required to separate β_0 connected components. For instance, Figure 17(c) shows this process for a topological space given in Figure 17(b). Then, the required total number of hyperplanes is bounded by

$$d_1 \leq m + 2(\beta_0 - 1) + \sum_{k=1}^{d-1}\left(m - 2(d-k-1)\right)\beta_k$$

which completes the proof. □

E.4.2. The lower bound in Theorem 3.6.

Suppose the given architecture $d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} d_2 \xrightarrow{\sigma} \cdots \xrightarrow{\sigma} d_k \to 1$ is a universally feasible architecture on any topological space $\mathcal{X}$ satisfying the assumptions stated in Theorem 3.6. Then, it is enough to consider the 'worst' case of dataset to prove a lower bound. We will use the same idea in the proof of Proposition F.2. Specifically, for the given Betti numbers β_k, we consider a topological space $\mathcal{X}$ such that every 'hole' intersects with a straight line, say ℓ. Since each hole intersects with ℓ at least two points, we conclude that $\mathcal{N}$ has at least $2\sum_{k=0}^{d}\beta_k$ piecewise linear regions on ℓ (Figure 18(a)).

Now we introduce one terminology: from the piecewise linearity of deep ReLU networks, we define a *linear partition region* to be a maximum connected component where the network is affine on. Note also that the boundary of each linear partition region is non-differentiable points of $\mathcal{N}$ in $\mathbb{R}^d$, which are vanished points of some hidden layers.

We establish the proof by computing the upper bounds of number of linear partition regions on the straight line ℓ made by $\mathcal{N}$. For d_1 neurons in the first hidden layer, the set of vanishing points are d_1 hyperplanes in $\mathbb{R}^d$, thus it can intersect with ℓ at most d_1 times (Figure 18(b)). Then, consider the vanishing points of the second hidden layer. These points form a bent hyperplane in $\mathbb{R}^d$, which is refracted on the intersection with a vanishing hyperplane of the first layer (Figure 18(c)). Therefore, a vanishing hyperplane of the second hidden layer can intersect with ℓ at most (d_1+1) times for each neuron. This concludes that the number of vanishing hyperplanes of the second hidden layers can intersect with ℓ at most $d_2(d_1+1)$ times. By the same argument, after the third layer, the number of maximum partitions on ℓ is bounded by $d_3(d_2(d_1+1)+1)+1$ (Figure 18(d)), and so on. Then, for the given architecture $d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} d_2 \xrightarrow{\sigma} \cdots \xrightarrow{\sigma} d_k \to 1$, the number of linear partition regions on ℓ is bounded by

$$\begin{aligned} &1+d_k + d_k d_{k-1} + d_k d_{k-1} d_{k-2} + \cdots + d_k \cdots d_1 \\ &= 1 + \sum_{i=1}^{k} \prod_{j=i}^{k} d_j. \end{aligned}$$

Therefore, to be a feasible architecture on $\mathcal{X}$, we get

$$1 + \sum_{i=1}^{k} \prod_{j=i}^{k} d_j \geq 2 \sum_{k=0}^{d} \beta_k - 1,$$

which completes the proof. □

E.5. Proof of Theorem 3.7

Recall that $\mathcal{T}_j$ satisfies that $\sigma(\mathcal{T}_j(\boldsymbol{x}_i)) = 0$ or λ for all $\boldsymbol{x}_i \in \mathcal{D}, j \in [J]$. From (5) and Lemma F.4, we know that $C_j := \{\boldsymbol{x} \mid \mathcal{T}_j(\boldsymbol{x}) = \lambda\}$ is a convex polytope for each $j \in [J]$. Then, we get

$$\begin{aligned} \mathcal{N}(\boldsymbol{x}) &= -\frac{1}{2}\lambda + \sum_{j=1}^{J} a_j \sigma(\mathcal{T}_j(\boldsymbol{x})) \\ &= -\frac{1}{2}\lambda + \sum_{j=1}^{J} a_j \mathbb{1}_{\{\boldsymbol{x} \in C_j\}}(\boldsymbol{x}). \end{aligned}$$

Now, we define $\mathcal{C}_P := \{C_j \in \mathcal{C} \mid a_j = +1\}$ and $\mathcal{C}_Q := \{C_j \in \mathcal{C} \mid a_j = -1\}$. Then, we get

$$\mathcal{N}(\boldsymbol{x}_i) > 0 \qquad \Longleftrightarrow \qquad \sum_{C \in \mathcal{C}_P} \mathbb{1}_{\{\boldsymbol{x}_i \in C\}} > \sum_{C \in \mathcal{C}_Q} \mathbb{1}_{\{\boldsymbol{x}_i \in C\}}$$

for all $i \in [n]$. Therefore, Definition 3.2 establishes that $\mathcal{C}$ is a polytope-basis cover of $\mathcal{D}$, ensuring its accuracy matches that of $\mathcal{N}$. □

E.6. Proof of Proposition C.2

Here, we present proofs for each statement.

1. Firstly, we establish the validity of (5), ensuring $v_{jk} < 0$. This condition holds at initialization as the network adheres to (8). Throughout the algorithm, consisting of neuron removal and neuron scaling, neither action compromises (5). Proposition F.5 assures the persistence of (8) under gradient flow. Consequently, (5) remains satisfied throughout.

   Secondly, we scrutinize the condition (7). Suppose there exists $\boldsymbol{x}_i \in \mathcal{D}$ such that $0 < \mathcal{T}(\boldsymbol{x}_i) < \lambda$. From Definition 5, it implies that

$$-\lambda < \mathcal{T}(\boldsymbol{x}_i) - \lambda = \sum_{k \in [m]} v_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}_i + b_k) < 0$$

Recall that all v_k are negative, by the preceded proof, and ReLU is positive homogeneous. Therefore, by scaling neurons $(v_k, \boldsymbol{w}_k, b_k)$ by $(\lambda_{scale} v_k, \lambda_{scale} \boldsymbol{w}_k, \lambda_{scale} b_k)$ such that $\boldsymbol{w}_k^\top \boldsymbol{x}_i + b_k > 0$, the network output changes from

$$\mathcal{T}(\boldsymbol{x}_i) - \lambda \qquad \rightarrow \qquad \lambda_{scale}^2 (\mathcal{T}(\boldsymbol{x}_i) - \lambda).$$

Therefore, by repeating this scaling sufficiently many times, given $\lambda_{scale} > 1$, $(\mathcal{T}(\boldsymbol{x}_i) - \lambda)$ decreases under -1. This process applies to all such $\boldsymbol{x}_i$ in the dataset, eventually leading to $\sigma(\mathcal{T}(\boldsymbol{x}_i)) = 0$ or λ for all $\boldsymbol{x}_i \in \mathcal{D}$.

2. In Algorithm 2, first for loop must terminate after $Epochs$ repetition. Then, the following repeat loop makes all $\mathcal{T}_j$ to satisfy $\sigma(\mathcal{T}_j(\boldsymbol{x}_i))$ is either 0 or 1, for all $\boldsymbol{x}_i \in \mathcal{D}$. However, by the previous proof, we know Algorithm 1 provides a network satisfying both (5) and (7) in finite time. Therefore, this algorithm is guaranteed to terminate in finite time. Lastly, since each $\mathcal{T}_j$ has binary output 0 or 1, convex polytopes defined by $C_j := \{\mathcal{T}_j > 0\}$ forms a polytope-basis cover, which has the same accuracy with the produced network $\mathcal{N}$.

3. To prove finite-time termination of Algorithm 3, it is enough to show that the **repeat** loop in the algorithm must terminate in finite time. Specifically, we prove the following two statements: 1. the incorrectly covered data $\hat{\boldsymbol{x}}$ is correctly covered after one process in the **repeat** loop by adding two polytopes, and 2. these added polytopes do not hurt other correctly covered data.

   First, let $\mathcal{C}$ be a (constructing) polytope-basis cover and let $\hat{\boldsymbol{x}}$ be an incorrectly covered data by $\mathcal{C}$. Then, it means the sign of $\mathcal{N}_+(\hat{\boldsymbol{x}}) - \mathcal{N}_-(\hat{\boldsymbol{x}})$ and the sign of $\hat{o}_i := \sum_{c \in \mathcal{C}_P} \mathbb{1}_{\{\hat{\boldsymbol{x}} \in C\}} - \sum_{c \in \mathcal{C}_Q} \mathbb{1}_{\{\hat{\boldsymbol{x}} \in C\}} + \frac{1}{2}$. Let c be the value between $\mathcal{N}_-(\hat{\boldsymbol{x}})$ and $\mathcal{N}_+(\hat{\boldsymbol{x}})$, and define two convex polytopes

$$C_+ := \{\boldsymbol{x} \mid \mathcal{N}_+(\boldsymbol{x}) < c\}$$
$$C_- := \{\boldsymbol{x} \mid \mathcal{N}_-(\boldsymbol{x}) > -c\}.$$

   Then, by the definition, $\hat{\boldsymbol{x}}$ is only contained in either C_+ or C_-, determined by the sign of $\mathcal{N}(\hat{\boldsymbol{x}})$. Therefore, adding these two polytopes to the polytope-basis cover $\mathcal{C}$, by $C_- \in \mathcal{C}_P$ and $C_+ \in \mathcal{C}_Q$, $\hat{\boldsymbol{x}}$ is now correctly classified by the cover $\mathcal{C}$.

   Second, we claim that adding above two polytopes do not disrupt other correctly covered data. Recall the approximation of convex functions discussed in Appendix C.2. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a convex function, and let $P := \{p_0, p_1, \cdots, p_{J+1}\}$ be a finite partition of real number by

$$-\infty = p_0 \leq p_1 \leq p_2 \leq \cdots \leq p_J \leq p_{J+1} = +\infty$$

   Then, f can be approximated by

$$f(\boldsymbol{x}) \approx p_1 + \sum_{j=1}^{J} (p_{j+1} - p_j) \mathbb{1}_{\{f(\boldsymbol{x}) < p_j\}}.$$

   This approximation can be understood as quantization of the function f by values in P. Then, elementary analysis (Rudin et al., 1976) shows that refinement of the partition P only increases the accuracy of the above approximation. I.e., as polytopes added in the constructing polytope-basis cover $\mathcal{C}$, the number of incorrectly covered data by $\mathcal{C}$ strictly decreases. Since there is finite data points in the training dataset $\mathcal{D}$, Algorithm 3 must terminate in finite time. More precisely, the **repeat** loop in the algorithm must be halted in $n = |\mathcal{D}|$ times.

4. Let $\mathcal{C} = \{C_1, C_2, C_3, \cdots, C_J\}$ be the output of Algorithm 4. Then, from its construction described in the algorithm, it implies that

$$\begin{aligned}
&C_1 \text{ contains all points in } \mathcal{D}_0.\\
&C_2 \text{ contains all points in } \mathcal{D}_1 \cap C_1.\\
&C_3 \text{ contains all points in } \mathcal{D}_0 \cap C_2.\\
&\qquad\qquad \vdots\\
&C_{J-1} \text{ contains all points in } \mathcal{D}_{\frac{1+(-1)^{J-1}}{2}} \cap C_{J-2}.\\
&C_J \text{ contains all points in } \mathcal{D}_{\frac{1+(-1)^{J}}{2}} \cap C_J, \text{ and does not contain the another class.}
\end{aligned}$$

Now, define

$$\mathcal{N}(\boldsymbol{x}) := -\frac{1}{2} + \sum_{j=1}^{J} (-1)^j \sigma(\mathcal{T}(\boldsymbol{x})). \tag{25}$$

Then, $\mathcal{N}$ is a three-layer ReLU network of the form (6). Furthermore, $\mathcal{T}_j(\boldsymbol{x}_i)$ is either 0 or 1 for all $i \in [n]$ and $j \in [J]$, satisfying the condition (7). Therefore, Theorem 3.7 verifies that $\mathcal{C} := \{C_j\}_{j\in[J]}$ becomes a polytope-basis cover of the dataset.

□

E.7. Proof of Theorem D.3.

E.7.1. Proof for the MSE loss (13).

The proof is divided into several steps. First, for $k \in [m]$, we define the following sets:

$$A_k := \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{w}_k^\top \boldsymbol{x} + b_k > 0\} \tag{26}$$

$$B_k := \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{w}_k^\top \boldsymbol{x} + b_k > 0 \text{ and } \boldsymbol{w}_j^\top \boldsymbol{x} + b_j > 0 \text{ for } j \neq k\} \tag{27}$$

I.e., A_k is the region where k-th neuron is alive, and B_k is the region where only k-th neuron is alive (see Figure 19(b,c)). Similarly, we define

$$A_0 := \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{w}_k^\top \boldsymbol{x} + b_k < 0\}$$

which is the region where all neurons are dead, except the last bias term v_0. Now, we define the following values for every $k \in [m]$:

$$\begin{aligned} l_k &:= \text{the distance between } O \text{ and } \partial C_k, \\ s_k &:= -\frac{b_k}{\|\boldsymbol{w}_k\|}, \qquad (28) \\ t_k &:= -\frac{v_0}{v_k \|\boldsymbol{w}_k\|}, \qquad (29) \\ t &:= \max_{k\in[m]} \{t_k, \delta\}. \end{aligned}$$

Then, the network initialization condition (16) gives

$$\begin{aligned} &0 < t_k < R, \\ &0 < s_k < l_k < s_k + t_k. \end{aligned}$$

In other words, s_k is the distance between the origin point O and the hyperplane $\{\boldsymbol{w}_k^\top \boldsymbol{x} + b_k = 0\}$. t_k is the length of 'height' of the region B_k as depicted in Figure 19(c). To be familiar for these notations, we demonstrate the output $\mathcal{N}$ in Figure 19(d) with respect to $\|\boldsymbol{w}_k\|$.

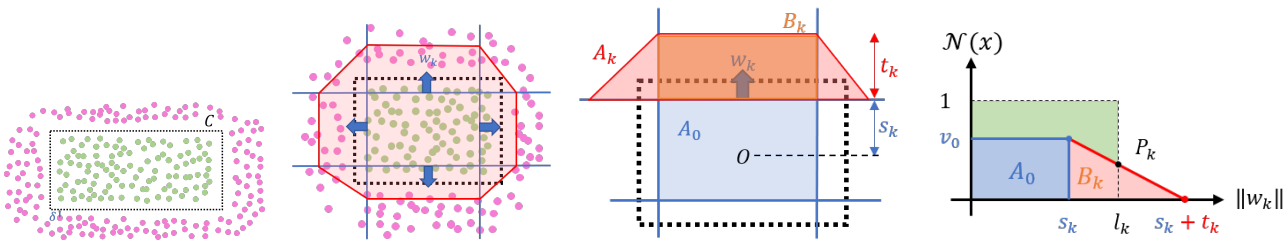
It is clear that $\mathcal{N}(\boldsymbol{x}) = v_0$ if $\boldsymbol{x} \in A_0$, and it linearly decreases to zero for $\boldsymbol{x} \in B_k$. When $\boldsymbol{x}_i \in B_k$ satisfies $\boldsymbol{x}_i^\top \frac{\boldsymbol{w}_k}{\|\boldsymbol{w}_k\|} = s_k + t_k$, $\mathcal{N}(\boldsymbol{x}_i) = 0$. Now we are ready to prove the theorem.

For the previously defined sets A_k and B_k, the MSE loss (13) is computed by

$$\begin{aligned} L_{MSE} &= \frac{1}{2n} \sum_{i=1}^{n} (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2 \\ &= \frac{1}{2n} \sum_{\boldsymbol{x}_i \in A_0} (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2 + \frac{1}{2n} \sum_{\boldsymbol{x}_i \in \cup_k B_k} (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2 + \frac{1}{2n} \sum_{\boldsymbol{x}_i \in \cup_k (A_k \setminus B_k)} (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2 \\ &=: L_1 + L_2 + L_3. \end{aligned} \tag{30}$$

Note that we omitted Θ notation, the set of all learnable parameters. We will observe the change of these loss values with respect to one update of parameters. We add prime ($'$) for the updated parameter. For the given step size η, we explicitly provide the update of parameters by

$$v_0 \quad \to \quad v_0' := v_0 + \Delta v_0,$$



(a) (b) (c) (d)

Figure 19. Proof of Theorem D.3. (a) The given dataset $\mathcal{D}$ is polyhedrally separable by a black dashed rectangle C. (b) Initialization of a two-layer ReLU network $\mathcal{N}$. (c) For $k \in [m]$, sets A_k and B_k defined in (26) and (27) are illustrated. (d) A sideview of the function $\mathcal{N}$ with respect to $\|\boldsymbol{w}_k\|$. s_k and t_k are defined in (28) and (29). Note that the intersection point P_k is invariant after the update of parameters.

$$\begin{aligned} s_k &\rightarrow s_k' := s_k + \Delta s_k, \\ t_k &\rightarrow t_k' := t_k + \Delta t_k \end{aligned}$$

for all $k \in [m]$, where

$$\Delta v_0 := \begin{cases} 0 & \text{if} \quad \#(\cup_{k\in[l]} A_k) > 0, \\ -\frac{1}{2}(v_0 - 1)t\eta, & \text{otherwise}, \end{cases} \tag{31}$$

$$\Delta s_k := \begin{cases} \frac{\eta t_k^2}{v_0 + \eta t_k} \cdot \frac{l_k - s_k}{l_k + t_k - s_k} & \text{if} \quad \#(A_k) > 0, \\ 0 & \text{otherwise}, \end{cases} \tag{32}$$

$$\Delta t_k := \begin{cases} \Delta s_k - \frac{\eta t_k^2}{v_0 + \eta t_k} & \text{if} \quad \#(A_k) > 0, \\ \frac{t_k \Delta v_0 - \eta t_k^2}{v_0 + \eta t_k} & \text{otherwise}. \end{cases} \tag{33}$$

Specifically, v_0 is updated if and only if $\cup_{k\in[l]} A_k$ contains a data point, where s_k and t_k are updated exclusively. The given update terms are proposed to have some invariant quantity. In Figure 19(d), we set P_k to be the output value of $\mathcal{N}$ at l_k, and update equations in (31)$\sim$(33) are determined to keep this value P_k. Furthermore, it satisfies that the change of the slope is exactly $-\eta$, i.e.,

$$\begin{aligned} \Delta\left(-\frac{v_0}{t_k}\right) &:= -\frac{v_0 + \Delta v_0}{t_k + \Delta t_k - \Delta s_k} + \frac{v_0}{t_k} \\ &= -\eta. \end{aligned}$$

Note also that $v_0 < 1$ and $s_k < l_k$ are increasing, where $t_k > 0$ is decreasing.

In the subsequent steps, we examine the change of each loss value. The main idea of the proof is computing lower bounds on the reduction of the loss value resulting from one-step update given by (31)$\sim$(33). It is divided into four steps.

STEP 1. First, we consider when $\#(\cup_{k\in[l]} A_k) = 0$. In this case, since $L_2 = L_3 = 0$ from the definition (30), it is enough to investigate the change of L_1. Recall that

$$L_1 := \frac{1}{2n}\#(A_0)(v_0 - 1)^2.$$

By one-step update of parameters, it becomes $L_1 \rightarrow L_1' := L_1 + \Delta L_1$. Then,

$$\begin{aligned} \Delta L_1 &= L_1' - L_1 \\ &= \frac{1}{2n}\sum_{\boldsymbol{x}_i \in A_0} (v_0 + \Delta v_0 - 1)^2 - \frac{1}{2n}\sum_{\boldsymbol{x}_i \in A_0} (v_0 - 1)^2 \\ &= \frac{1}{2n} \cdot (2v_0 - 2 + \Delta v_0)\Delta v_0 \cdot \#(A_0) \end{aligned}$$

$$
\begin{aligned}
&= -\frac{\#(A_0)}{n}(1-v_0)\Delta v_0 + \frac{\#(A_0)}{2n}(\Delta v_0)^2 \\
&= -\frac{\#(A_0)}{2n}(1-v_0)^2(t\eta - \frac{1}{4}t^2\eta^2) \\
&< -\frac{\#(A_0)}{2n}(1-v_0)^2 \cdot \frac{1}{2}t\eta.
\end{aligned}
$$

Note that we use $\eta < \frac{2}{t} < \frac{2}{\delta}$ on the last inequality. Then, we get

$$
\begin{aligned}
L_1' &= L_1 + \Delta L_1 \\
&= \left(1 + \frac{\Delta L_1}{L_1}\right) L_1 \\
&< \left(1 - \frac{\frac{1}{2n}\#(A_0)(1-v_0)^2 \cdot \frac{1}{2}t\eta}{\frac{1}{2n}\#(A_0)(1-v_0)^2}\right) L_1 \\
&= \left(1 - \frac{1}{2}t\eta\right) L_1 \\
&\leq \left(1 - \frac{1}{2}\delta\eta\right) L_1
\end{aligned} \tag{34}
$$

which states that L_1 strictly decreases after the update.

STEP 2. Now, we consider when $\#(\cup_{k\in[l]} A_k) > 0$. We investigate the second term in (30), defined by

$$
L_2 := \frac{1}{2n} \sum_{k\in[l]} \sum_{\boldsymbol{x}_i \in B_k} (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2.
$$

Recall that the update of parameters given in (31)$\sim$(33) are chosen to keep P_k value and increasing the absolute value of the slope $-\frac{v_0}{t_k}$ by η. Therefore, for any $\boldsymbol{x}_i \in B_k$, it $\mathcal{N}(\boldsymbol{x}_i)$ increases (or decreases) if and only if $\boldsymbol{x}_i \in B_k \cap C$ (or $\boldsymbol{x}_i \in B_k \backslash C$). Therefore, $|\mathcal{N}(\boldsymbol{x}_i) - y_i|$ always strictly decreases after the update, which implies that

$$
\Delta L_2 := L_2' - L_2 < 0. \tag{35}
$$

STEP 3. We observe the last term in (30) when $\#(\cup_{k\in[m]} A_k) > 0$, which is the most technical part in this proof. Recall that

$$
L_3 := \frac{1}{2n} \sum_{\boldsymbol{x}_i \in \cup_k (A_k \backslash B_k)} (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2.
$$

The goal of this step is showing that the absolute change of L_3 is less than it of L_2, i.e., $|\Delta L_3| < \Delta L_2$. The idea is based on the sparsity of the data distribution in $\mathcal{B}_r(\partial^2 C)$; near the neighborhood of 'edge' parts of the polytope C.

Note that for each $k \in [m]$, obviously we have $(A_k \backslash B_k) \subset \mathcal{B}_t(\partial^2 C_k)$ from the linearity of $\mathcal{N}$ (see Figure 19(c) and (d)). It is also worth noting that if $t_k \leq \delta$, then $L_3 = 0$ because there is no $\boldsymbol{x}_i$ in $\cup_{k\in[m]}(A_k \backslash B_k)$, and we have nothing to do. Thus we mostly consider $t_k > \delta$ cases.

Let $\mathcal{N}'$ be the network after the one-step update from $\mathcal{N}$. The difference of output is $\Delta\mathcal{N}(\boldsymbol{x}) := \mathcal{N}'(\boldsymbol{x}) - \mathcal{N}(\boldsymbol{x})$. Recall that parameters v_0, s_k, t_k follow the updated rule (31) $\sim$ (33) such that network have a constant output on $\partial C_k \cap B_k$ (Figure 19(c) and (d)). This implies that both networks $\mathcal{N}$ and $\mathcal{N}'$ have fixed outputs for $\partial C_k \cap B_k$, and then the affine space connecting those fixed points also has the fixed output which comes from the piecewise linearity of $\mathcal{N}$.

STEP 3-1 First, we compute an upper bound of $|\Delta L_3|$. Since $\mathcal{N}(\boldsymbol{x})$ is piecewise linear, we consider the input space partition in $A_k \backslash B_k$ where $\mathcal{N}$ is linear on. Observing the 'corner' parts of the polytope C (see Figure 19(c) and (d)), each partition is intersection of some neurons of $\mathcal{N}$. Choose one partition $P \subset A_k \backslash B_k$, and let $J_P \subset [m]$ be the index set of P that $\boldsymbol{w}_j$ is activated on P if and only if $j \in J_P$, or namely, $P = \bigcap_{j\in J_P} A_j$. Then obviously, the partition P is contained in a ball with radius $\max_{j\in J_P} t_j \leq t < R$. On the contrary, any partition P is contained in t_k-radius ball from ∂_k for some k. Using this, we can disjointly separate the partitions to $\mathcal{Q}_k$ such that

1. $Q_k \subset (A_k \backslash B_k)$
2. Every $P \in \cup_{k\in[m]}(A_k \backslash B_k)$ is exactly contained in one of Q_k.
3. Every $P \in Q_k$ can be bounded by a ball with radius t_k.

Note that $\mathcal{Q}_k$ is a collection of partitions, which can be empty. Using this, we decompose L_3 by the following way. This is just rearranging the terms in L_3.

$$\begin{aligned} L_3 &= \frac{1}{2} \sum_{\boldsymbol{x}_i \in \cup_{k\in[m]}(A_k \backslash B_k)} (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2 \\ &= \frac{1}{2} \sum_{k\in[m]} \sum_{\boldsymbol{x}_i \in \mathcal{Q}_k} (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2 \\ &=: \frac{1}{2} \sum_{k\in[m]} L_{3,k}. \end{aligned}$$

Now, we bound the change of network output $\Delta\mathcal{N}(\boldsymbol{x}_i)$ for $\boldsymbol{x}_i \in P \in \mathcal{Q}_k$.

$$\begin{aligned} |\Delta\mathcal{N}(\boldsymbol{x}_i)| &= |\mathcal{N}(\boldsymbol{x}_i) - \mathcal{N}'(\boldsymbol{x}_i)| \\ &\leq \left| \sum_{j\in J_P} \Delta\left(-\frac{v_0}{t_j}\right) t_k \right| \\ &= \sum_{j\in J_P} \eta R \\ &\leq lR\eta. \end{aligned}$$

Above inequalities come from the fact that, the change of linear value is bounded by product of the change of slope and the maximum diameter of the set. Finally, for a $k \in [m]$, we compute an upper bound of the loss variation of $L_{3,k}$.

$$\begin{aligned} |\Delta L_{3,k}| &= \left| \frac{1}{2n} \sum_{\boldsymbol{x}_i \in (A_k \backslash B_k)} (\mathcal{N}(\boldsymbol{x}_i) + \Delta\mathcal{N}(\boldsymbol{x}_i) - y_i)^2 - \frac{1}{2n} \sum_{\boldsymbol{x}_i \in (A_k \backslash B_k)} (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2 \right| \\ &= \left| \frac{1}{n} \sum_{\boldsymbol{x}_i \in (A_k \backslash B_k)} \left(\mathcal{N}(\boldsymbol{x}_i) - y_i + \frac{1}{2}\Delta\mathcal{N}(\boldsymbol{x}_i) \right) \cdot \Delta\mathcal{N}(\boldsymbol{x}_i) \right| \\ &\leq \frac{1}{n} \sum_{\boldsymbol{x}_i \in (A_k \backslash B_k)} \left(|\mathcal{N}(\boldsymbol{x}_i) - y_i| \cdot |\Delta\mathcal{N}(\boldsymbol{x}_i)| + \frac{1}{2}|\Delta\mathcal{N}(\boldsymbol{x}_i)|^2 \right) \\ &\leq \frac{1}{n} \sum_{\boldsymbol{x}_i \in (A_k \backslash B_k)} \left(1 \cdot |\Delta\mathcal{N}(\boldsymbol{x}_i)| + \frac{1}{2}|\Delta\mathcal{N}(\boldsymbol{x}_i)|^2 \right) \\ &\leq \frac{1}{n} \# \left(\mathcal{B}_{t_k}(\partial^2 C_k) \right) \cdot (mR\eta + \frac{1}{2} m^2 R^2 \eta^2) \\ &\leq \frac{2}{n} \# \Big(\mathcal{B}_{t_k}(\partial^2 C_k) \Big) \cdot mR\eta \\ &\leq \frac{2mR\eta}{n} \cdot \rho \, \#(\mathcal{B}_{t_k}(\partial C_k)). \end{aligned} \tag{36}$$

Note that we used $\eta < \frac{2}{mR}$ to bound the quadratic term η^2.

STEP 3-2. Now, we compute a similar bound for ΔL_2. It can be decomposed to the sum on each B_k.

$$L_2 = \frac{1}{2n} \sum_{k\in[m]} \sum_{\boldsymbol{x}_i \in B_k} (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2$$

$$=: \sum_{k\in[m]} L_{2,k}.$$

We use the fact that each data point $\boldsymbol{x}_i$ is far from ∂C at least δ. From the definition, we get $\Delta\mathcal{N}(\boldsymbol{x}_i) > \delta\eta$. Note that if $t_k < 2\delta$, then $L_{3,k}$ strictly decreases and we have nothing to do. Otherwise, when $t_k > 2\delta$, we have $R > 2\delta$ and there is data far from δ distance from ∂B_k. For such data point $\boldsymbol{x}_i$, we get

$$\begin{aligned}\mathcal{N}(\boldsymbol{x}_i) - 0 &= v_0 - \frac{v_0}{t_k}(t_k - \delta)\\ &= v_0\frac{\delta}{t_k}\\ &> \frac{v_0\delta}{R}\end{aligned}$$

and

$$\begin{aligned}1 - \mathcal{N}(\boldsymbol{x}_i) &= 1 - \left(v_0 - \frac{v_0}{t_k}(t_k - \delta)\right)\\ &= 1 - \frac{v_0}{t_k}\delta\\ &> 1 - \frac{1}{2}v_0\\ &> \frac{1}{2}v_0\\ &> \frac{v_0\delta}{R}.\end{aligned}$$

Therefore, we have shown that

$$\min_{s_k+\delta\leq \|\boldsymbol{x}_i\| \leq s_k+t_k-\delta} |\mathcal{N}(\boldsymbol{x}_i) - y_i| \geq \frac{v_0\delta}{R}.$$

Now we induce the lower bound of $\Delta L_{2,k}$.

$$\begin{aligned}\Delta L_{2,k} &= \frac{1}{2n}\sum_{\boldsymbol{x}_i\in B_k}\left((\mathcal{N}(\boldsymbol{x}_i) + \Delta\mathcal{N}(\boldsymbol{x}_i) - y_i)^2 - (\mathcal{N}(\boldsymbol{x}_i) - y_i)^2\right)\\ &= \frac{1}{n}\sum_{\boldsymbol{x}_i\in B_k}\left(\mathcal{N}(\boldsymbol{x}_i) - y_i + \frac{1}{2}\Delta\mathcal{N}(\boldsymbol{x}_i)\right)\cdot\Delta\mathcal{N}(\boldsymbol{x}_i)\\ &= \frac{1}{n}\sum_{\boldsymbol{x}_i\in B_k}\left(-|(\mathcal{N}(\boldsymbol{x}_i) - y_i)|\cdot|\Delta\mathcal{N}(\boldsymbol{x}_i)| + \frac{1}{2}|\Delta\mathcal{N}(\boldsymbol{x}_i)|^2\right)\\ &\leq \frac{1}{n}\sum_{\boldsymbol{x}_i\in B_k}\left(-\min_{\delta\leq \|\boldsymbol{x}_i\|-s_k \leq t_k-\delta}|\mathcal{N}(\boldsymbol{x}_i) - y_i|\cdot\eta\delta + \frac{1}{2}R^2\eta^2\right)\\ &\leq -\frac{1}{n}\#\left(\mathcal{B}_{\frac{t_k}{2}-\delta}(\partial C_k) - \mathcal{B}_{t_k}(\partial^2 C_k)\right)\cdot\left(\frac{v_0\delta}{R}\cdot\delta\eta - \frac{1}{2}R^2\eta^2\right)\\ &\leq -\frac{1}{n}(1-\rho)\#(\mathcal{B}_{t_k}(\partial C_k))\cdot\left(\frac{v_0\delta^2\eta}{R} - \frac{1}{2}R^2\eta^2\right)\\ &< \frac{1}{n}\frac{(1-\rho)v_0\delta^2\eta}{2R}\,\#(\mathcal{B}_{t_k}(\partial C_k)).\end{aligned} \tag{37}$$

Note that Assumption D.2 on dataset $\mathcal{D}$ is used to induce this inequality. Now, we compare (37) and (36) with initialization condition (17) for v_0. Then, we finally get

$$|\Delta L_{3,k}| \leq \frac{2lR\eta\rho}{n}\,\#(\mathcal{B}_{t_k}(\partial C_k))$$

$$
\begin{aligned}
&< \frac{(1-\rho)v_0\delta^2}{2nR}\eta \cdot \#(\mathcal{B}_{t_k}(\partial C_k)) \\
&< -\Delta L_{2,k}
\end{aligned}
$$

for every $k \in [m]$. By summing up, we conclude $|\Delta L_3| < -\Delta L_2$ or,

$$
\Delta L_2 + \Delta L_3 < 0. \tag{38}
$$

STEP 4. Finally, we combine all results in the previous steps. When $\#(\cup_{k\in[m]} A_k) > 0$, only L_2 and L_3 are changed, then one step update gives

$$
\begin{aligned}
L' &= L + \Delta L \\
&= L_1 + L_2 + L_3 + \Delta L_1 + \Delta L_2 + \Delta L_3 \\
&< L_1 + L_2 + L_3
\end{aligned}
$$

from (35) and (38). Furthermore, since $s_k < l_k$ increases and $t_k > 0$ decreases, the updated parameters satisfy the assumption (16) again. Using mathematical induction, we can repeat above steps until $\#(\cup_{k\in[m]} A_k) = 0$. After achieving $\#(\cup_{k\in[m]} A_k) = 0$, we get $L_2 = L_3 = 0$ from their definition (30). Then, the remained loss L_1 exponentially decreases to zero because

$$
\begin{aligned}
L' &= L_1 + \Delta L_1 \\
&\leq \left(1 - \frac{1}{2}\delta\eta\right) L_1 \\
&\leq \left(1 - \frac{1}{2}\delta\eta\right) L
\end{aligned}
$$

from (34). This completes the proof. □

E.7.2. PROOF FOR THE BCE LOSS (14).

The proof idea is similar with the previous proof. We use the same definitions for A_k, B_k, s_k, t_k, l_k, and other notations. The BCE loss (14) is rearranged by

$$
\begin{aligned}
L_{BCE} &= -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log \texttt{SIG} \circ \mathcal{N}(\boldsymbol{x}_i) + (1-y_i)\log(1 - \texttt{SIG} \circ \mathcal{N}(\boldsymbol{x}_i))\right) \\
&= -\frac{1}{n}\sum_{\boldsymbol{x}_i \in A_0} \log \texttt{SIG} \circ \mathcal{N}(\boldsymbol{x}_i) \\
&\quad - \frac{1}{n}\sum_{k\in[m]}\sum_{\boldsymbol{x}_i \in B_k}\left(y_i \log \texttt{SIG} \circ \mathcal{N}(\boldsymbol{x}_i) + (1-y_i)\log(1 - \texttt{SIG} \circ \mathcal{N}(\boldsymbol{x}_i))\right) \\
&\quad - \frac{1}{n}\sum_{k\in[m]}\sum_{\boldsymbol{x}_i \in (A_k \setminus B_k)}\left(y_i \log \texttt{SIG} \circ \mathcal{N}(\boldsymbol{x}_i) + (1-y_i)\log(1 - \texttt{SIG} \circ \mathcal{N}(\boldsymbol{x}_i))\right) \\
&=: L_1 + L_2 + L_3.
\end{aligned} \tag{39}
$$

Before we start, we compute the derivative and its bound of some functions. For $\zeta \in \mathbb{R}$, define

$$
\begin{aligned}
f(\zeta) &:= \log \texttt{SIG}(\zeta), \\
g(\zeta) &:= \log(1 - \texttt{SIG}(\zeta)).
\end{aligned}
$$

Then their derivatives are given by

$$
\frac{d}{d\zeta} f(\zeta) := 1 - \texttt{SIG}(\zeta),
$$

$$\frac{d}{d\zeta}g(\zeta) := -\texttt{SIG}(\zeta).$$

From the mean value theorem (MVT), we get

$$\begin{aligned} f(\zeta+\Delta\zeta) &= f(\zeta) + f'(\zeta)\Delta\zeta + \frac{1}{2}f''(\tilde{\zeta})(\Delta\zeta)^2 \\ &\geq f(\zeta) + (1-\texttt{SIG}(\zeta))\Delta\zeta - \frac{1}{2}(\Delta\zeta)^2 \end{aligned}$$

and

$$\begin{aligned} f(\zeta+\Delta\zeta) - f(\zeta) &= (1-\texttt{SIG}(\tilde{\zeta}))\Delta\zeta \\ &\leq \Delta\zeta. \end{aligned}$$

Now we defin the update of parameters. For $k \in [l]$, the update of v_0 is given by

$$\Delta v_0 := \begin{cases} 0 & \text{if} \quad \#(\cup_{k\in[l]} A_k) > 0, \\ (1-\texttt{SIG}(v_0))\eta. & \text{otherwise} \end{cases} \tag{40}$$

For Δs_k and Δt_k, we adopt the same update defined in (32) and (33). Namely, the update of parameters preserves the value of $\mathcal{N}$ on l_k and the change of slope is set to $-\eta$. We repeat the analogous arguments in the previous proof.

STEP 1. Firstly, we consider the first loss term L_1 in (39) when $\#(\cup_{k\in[m]} A_k) = 0$. It is changed by

$$\begin{aligned} \Delta L_1 &= L_1' - L_1 \\ &= -\frac{1}{n}\sum_{\boldsymbol{x}_i \in A_0} \log \texttt{SIG}(v_0+\Delta v_0) + \frac{1}{n}\sum_{\boldsymbol{x}_i \in A_0} \log \texttt{SIG}(v_0) \\ &= -\frac{\#(A_0)}{n}(f(v_0+\Delta v_0) - f(v_0)) \\ &\leq -\frac{\#(A_0)}{n}\left((1-\texttt{SIG}(v_0))\Delta v_0 - \frac{1}{2}(\Delta v_0)^2\right) \\ &= -\frac{\#(A_0)}{n}\left((1-\texttt{SIG}(v_0))^2\eta - \frac{1}{2}(1-\texttt{SIG}(v_0))^2\eta^2\right) \\ &< -\frac{\#(A_0)}{n}\frac{1}{2}(1-\texttt{SIG}(v_0))^2\eta. \end{aligned}$$

Therefore, L_1 strictly decreases. Note that we used $\eta < 1$ to bound the η^2 term.

STEP 2. Secondly, we consider when $\#(\cup_{k\in[m]} A_k) > 0$. As discussed in the previous subsection, $\mathcal{N}(\boldsymbol{x}_i)$ strictly increases (or decreases) if and only if $y_i = 1$ (or 0, respectively) because the slope $-\frac{v_0}{t_k}$ changes $-\eta$. This shows that $\Delta L_2 < 0$.

STEP 3. Thirdly, we observe ΔL_2 and $|\Delta L_3|$ when $\#(\cup_{k\in[m]} A_k) > 0$. We compute a bound of ΔL_3 first. For any $k \in [m]$,

$$\begin{aligned} |\Delta L_3| &= \frac{1}{n}\Bigg| \sum_{\boldsymbol{x}_i \in (A_k \setminus B_k)} y_i(f(\mathcal{N}(\boldsymbol{x}_i) + \Delta\mathcal{N}(\boldsymbol{x}_i)) - f(\mathcal{N}(\boldsymbol{x}_i))) \\ &\qquad + (1-y_i)(g(\mathcal{N}(\boldsymbol{x}_i) + \Delta\mathcal{N}(\boldsymbol{x}_i)) - g(\mathcal{N}(\boldsymbol{x}_i))) \Bigg| \\ &\leq \frac{1}{n}\sum_{\boldsymbol{x}_i \in (A_k \setminus B_k)} \Big|(f(\mathcal{N}(\boldsymbol{x}_i) + \Delta\mathcal{N}(\boldsymbol{x}_i)) - f(\mathcal{N}(\boldsymbol{x}_i)))\Big| + \Big|(g(\mathcal{N}(\boldsymbol{x}_i) + \Delta\mathcal{N}(\boldsymbol{x}_i)) - g(\mathcal{N}(\boldsymbol{x}_i)))\Big| \\ &< \frac{1}{n}\sum_{\boldsymbol{x}_i \in (A_k \setminus B_k)} 2|\Delta\mathcal{N}(\boldsymbol{x}_i)| \\ &< \frac{2}{n}\,\#(\mathcal{B}_{t_k}(\partial^2 C_k)) \cdot \max_{\boldsymbol{x}_i \in (A_k \setminus B_k)} |\Delta\mathcal{N}(\boldsymbol{x}_i)| \end{aligned}$$

$$< \frac{2R\eta}{n} \#(\mathcal{B}_{t_k}(\partial^2 C_k)).$$

We obtain a similar bound for $\Delta L_{2,k}$. Let $V_0 := \log\left(\frac{(1-\rho)\delta}{4\rho R} - 1\right)$ be the upper bound of initialization of v_0. Note also that $\texttt{SIG}(V_0) = 1 - \frac{4\rho R}{(1-\rho)\delta}$ and $\eta < \frac{1-\texttt{SIG}(v_0)}{\delta}$. Then,

$$\begin{aligned}
\Delta L_{2,k} &= -\frac{1}{n}\sum_{\boldsymbol{x}_i \in B_k}\Big(y_i(f(\mathcal{N}(\boldsymbol{x}_i) + \Delta\mathcal{N}(\boldsymbol{x}_i)) - f(\mathcal{N}(\boldsymbol{x}_i))) \\
&\qquad + (1-y_i)(g(\mathcal{N}(\boldsymbol{x}_i) + \Delta\mathcal{N}(\boldsymbol{x}_i)) - g(\mathcal{N}(\boldsymbol{x}_i)))\Big) \\
&= -\frac{1}{n}\sum_{\boldsymbol{x}_i \in B_k, y_i=1}\Big((f(\mathcal{N}(\boldsymbol{x}_i) + \Delta\mathcal{N}(\boldsymbol{x}_i)) - f(\mathcal{N}(\boldsymbol{x}_i))) \\
&\qquad - \frac{1}{n}\sum_{\boldsymbol{x}_i \in B_k, y_i=0}(g(\mathcal{N}(\boldsymbol{x}_i) + \Delta\mathcal{N}(\boldsymbol{x}_i)) - g(\mathcal{N}(\boldsymbol{x}_i)))\Big) \\
&< -\frac{1}{n}\sum_{\boldsymbol{x}_i \in B_k, y_i=1}\left((1 - \texttt{SIG}\circ\mathcal{N}(\boldsymbol{x}_i))\Delta\mathcal{N}(\boldsymbol{x}_i) - \frac{1}{2}(\Delta\mathcal{N}(\boldsymbol{x}_i))^2\right) \\
&\qquad - \frac{1}{n}\sum_{\boldsymbol{x}_i \in B_k, y_i=0}\left(-\texttt{SIG}\circ\mathcal{N}(\boldsymbol{x}_i)\cdot\Delta\mathcal{N}(\boldsymbol{x}_i) - \frac{1}{2}(\Delta\mathcal{N}(\boldsymbol{x}_i))^2\right) \\
&< -\frac{1}{n}\sum_{s_k - l_k + \delta < h_i < -\delta}\left((1 - \texttt{SIG}\circ\mathcal{N}(\boldsymbol{x}_i))\Delta\mathcal{N}(\boldsymbol{x}_i) - \frac{1}{2}(\Delta\mathcal{N}(\boldsymbol{x}_i))^2\right) \\
&\qquad - \frac{1}{n}\sum_{\delta < h_i < s_k + t_k - \delta}\left(-\texttt{SIG}\circ\mathcal{N}(\boldsymbol{x}_i)\cdot\Delta\mathcal{N}(\boldsymbol{x}_i) - \frac{1}{2}(\Delta\mathcal{N}(\boldsymbol{x}_i))^2\right) \\
&< -\frac{1}{n}\sum_{s_k - l_k + \delta < h_i < -\delta}\left((1 - \texttt{SIG}(V_0))\delta\eta - \frac{1}{2}\delta^2\eta^2\right) \\
&\qquad - \frac{1}{n}\sum_{\delta < h_i < s_k + t_k - \delta}\left(\texttt{SIG}(0)\cdot\delta\eta - \frac{1}{2}\delta^2\eta^2\right) \\
&< -\frac{1}{n}\,\#\left(\mathcal{B}_{\frac{t_k}{2}-\delta}(\partial C_k) - \mathcal{B}_{t_k}(\partial^2 C_k)\right)\cdot\left((1 - \texttt{SIG}(V_0))\delta\eta - \frac{1}{2}\delta^2\eta^2\right) \\
&< -\frac{1}{n}(1-\rho)\#\left(\mathcal{B}_{t_k}(\partial C_k)\right)\cdot\frac{1}{2}(1 - \texttt{SIG}(V_0))\delta\eta.
\end{aligned}$$

Therefore,

$$\begin{aligned}
|\Delta L_{3,k}| &< \frac{2R\eta}{n}\rho\,\#(\mathcal{B}_{t_k}(\partial C_k)) \\
&< \frac{1}{n}(1-\rho)\#\left(\mathcal{B}_{t_k}(\partial C_k)\right)\cdot\frac{1}{2}(1 - \texttt{SIG}(V_0))\delta\eta \\
&< -L_{2,k}
\end{aligned}$$

and we get $\Delta L_2 + \Delta L_3 < 0$.

STEP 4. Finally, we combine results in the previous steps. When $\#(\cup_{k\in[m]}A_k) > 0$, v_0 is bounded by V_0 and we get $\Delta L_2 + \Delta L_3 < 0$ from **STEP 3**. After update, since $s_k < l_k$ increases and $t_k > 0$ decreases, the updated parameters satisfy Assumption D.2 again. It is repeated with strictly decreasing loss until reaching $\#(\cup_{k\in[m]}A_k) = 0$. After that, v_0 begins to strictly increase, which strictly decreases all L_1, L_2, and L_3. Further, the update equation (40) provides v_0 goes to infinity. Therefore, $\mathcal{N}(\boldsymbol{x}_i) \to \infty$ if and only if it label $y_i = 1$, concludes L_{BCE} converges to zero.

This completes the whole proof of Theorem D.3. □

F. Additional Propositions and Lemmas

Lemma F.1. *Let $0 \leq m \leq d$ be integers, and Δ^m be an m-simplex in $\mathbb{R}^d$. For a given $\varepsilon > 0$, there exists a two-layer ReLU network $\mathcal{T} : \mathbb{R}^d \to \mathbb{R}$ with the architecture $d \xrightarrow{\sigma} (d+1) \to 1$ such that*

$$\begin{aligned} \mathcal{T}(\boldsymbol{x}) &= 1 \qquad \text{if } \boldsymbol{x} \in \Delta^m, \\ \mathcal{T}(\boldsymbol{x}) &\leq 1 \qquad \text{if } \boldsymbol{x} \in B_\varepsilon(\Delta^m), \\ \mathcal{T}(\boldsymbol{x}) &< 0 \qquad \text{if } \boldsymbol{x} \notin B_\varepsilon(\Delta^m). \end{aligned}$$

Furthermore, the minimal width of such two-layer ReLU networks with the architecture $d \xrightarrow{\sigma} d_1 \to 1$ is exactly $d_1 = d+1$.

Proof. We prove the existence part first. For the given m-simplex Δ^m, pick $(d-m)$ distinct points in $\mathcal{B}_\varepsilon(\Delta^m)$. By connecting all these points with the points of Δ^m, we obtain a d-simplex contained in $\mathcal{B}_\varepsilon(\Delta^m)$, which is a convex polytope. By Proposition 3.1, there exists a neural network $\mathcal{T} : \mathbb{R}^d \to \mathbb{R}$ with the architecture $d \xrightarrow{\sigma} d_1 \to 1$ that satisfies the desired properties.

Now, we prove the minimality part. For every $\varepsilon > 0$, suppose there exists a two-layer ReLU network $\mathcal{T}(\boldsymbol{x}) := \sum_{i=1}^{d_1} v_i \sigma(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i) + v_0$ with $d_1 \leq d$ such that $\mathcal{T}(\boldsymbol{x}) = 1$ for $\boldsymbol{x} \in \Delta^m$ and $\mathcal{T}(\boldsymbol{x}) < 0$ for $\boldsymbol{x} \notin \mathcal{B}_\varepsilon(\Delta^m)$. First, we claim that the set of weight vectors $\{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_{d_1}\}$ spans $\mathbb{R}^d$. If the set cannot span $\mathbb{R}^d$, then there exists a nonzero vector $\boldsymbol{u} \in \mathbb{R}^d - \text{span} < \boldsymbol{w}_1, \cdots, \boldsymbol{w}_{d_1} >$. Then, from $\mathcal{T}(\boldsymbol{x}) = 1$ for $\boldsymbol{x} \in \Delta^m$, we get

$$\begin{aligned} \mathcal{T}(\boldsymbol{x} + t\boldsymbol{u}) &= \sum_{i=1}^{d_1} v_i \sigma(\boldsymbol{w}_i^\top (\boldsymbol{x} + t\boldsymbol{u}) + b_i) + v_0 \\ &= \sum_{i=1}^{d_1} v_i \sigma(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i) + v_0 \\ &= \mathcal{T}(\boldsymbol{x}) \\ &= 1 \end{aligned}$$

for any $t \in \mathbb{R}$. This contradicts to the condition $\mathcal{T}(\boldsymbol{x}) < 0$ for $\boldsymbol{x} \notin \mathcal{B}_\varepsilon(\Delta^m)$. Therefore, the set of weight vectors must span $\mathbb{R}^d$.

From the above claim, we further deduce that $d_1 \geq d$. Since we start with the assumption $d_1 \leq d$, thus $d_1 = d$. Then, we conclude that the set of weight vectors $\{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_{d_1}\}$ is a basis of $\mathbb{R}^d$. Now, we focus on the sign of v_0. Suppose $v_0 \geq 0$. Define

$$A := \bigcap_{i=1}^{d_1} \{\boldsymbol{x} \mid \boldsymbol{w}_i^\top \boldsymbol{x} + b_i < 0\},$$

which is an unbounded set since the set $\{\boldsymbol{w}_i\}$ is linearly independent. Then for $\boldsymbol{x} \in A$, we get $\mathcal{T}(\boldsymbol{x}) = v_0 \geq 0$. This contradicts to the assumption $\mathcal{T}(\boldsymbol{x}) < 0$ for all $\boldsymbol{x} \notin \mathcal{B}_\varepsilon(\Delta^m)$. Therefore, $v_0 < 0$.

Lastly, we consider the sign of v_i. Since $\mathcal{T}(\boldsymbol{x}) = 1 > 0$ for $\boldsymbol{x} \in \Delta^m$ and $v_0 < 0$, there exists some positive $v_i > 0$, say, $v_1 > 0$. Similar to the above argument, we define

$$B := \left\{\boldsymbol{x} \mid v_1 \boldsymbol{w}_1^\top \boldsymbol{x} + b_1 + v_0 > 0\right\} \bigcap_{i=2}^{d_1} \left\{\boldsymbol{x} \mid \boldsymbol{w}_i^\top \boldsymbol{x} + b_i < 0\right\},$$

which is also nonempty and unbounded. Then, for $\boldsymbol{x} \in B$, we have

$$\begin{aligned} \mathcal{T}(\boldsymbol{x}) &= \sum_{i=1}^{d_1} v_i \sigma(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i) + v_0 \\ &= v_1 \boldsymbol{w}_1^\top \boldsymbol{x} + b_1 + v_0 \\ &> 0. \end{aligned}$$

Since B is unbounded, this implies that $\mathcal{T}(\boldsymbol{x}) > 0$ over the unbounded subset in $\mathbb{R}^d$, which contradicts to the condition $\mathcal{T}(\boldsymbol{x}) < 0$ for all $\boldsymbol{x} \notin \mathcal{B}_\varepsilon(\Delta^m)$. This completes the whole proof, which shows that the minimum width of two-layer ReLU network is exactly $d+1$. □

Proposition F.2. *Let $\mathcal{X} \subset \mathbb{R}^d$ be a topological space and $\mathcal{A}$ be a neural network architecture that is a feasible architecture on $\mathcal{X}$. Then, there exists a topological space $\mathcal{X}'$ which is homeomorphic to $\mathcal{X}$, but $\mathcal{A}$ is not a feasible architecture on $\mathcal{X}'$.*

Proof. We use the similar technique introduced in (Telgarsky, 2015). Before we start, recall that a network $\mathcal{N}$ with the architecture $\mathcal{A}$ is a piecewise linear function on $\mathbb{R}^d$. Thus $\mathbb{R}^d$ can be partitioned into finitely many regions, where $\mathcal{N}$ is linear on each region. Let M be the maximum number of such regions, that networks with the architecture $\mathcal{A}$ can partition. I.e., any network with the architecture $\mathcal{A}$ has linear regions at most M partitions in $\mathbb{R}^d$.

Now, we consider a contractible topological space $\mathcal{Y}$ which has zig-zag shape as described in Figure 20(b), where the number of sawtooths is greater than $M+2$. We define another topological space $\mathcal{X}' := \mathcal{X}\#\mathcal{Y}$, where $\#$ denotes the connected sum. Note that we can glue $\mathcal{Y}$ to $\mathcal{X}$ preserving the number of sawtooths in $\mathcal{Y}$, because $\mathcal{X}$ is bounded. Then $\mathcal{X}'$ is homeomorphic to $\mathcal{X}$ since $\mathcal{Y}$ is contractible.

Finally, we prove the proposition using contradiction. Suppose there exists a deep ReLU network $\mathcal{N}'$ with the same architecture $\mathcal{A}$, which can approximate $\mathbb{1}_{\{\mathcal{X}'\}}$ under the given error bound $\varepsilon > 0$. Then, by the $\mathcal{Y}$ part in $\mathcal{X}'$, there exists a straight line ℓ that intersects $\mathcal{X}'$ more than $M+3$ times. Therefore, to approximate $\mathbb{1}_{\{\mathcal{X}'\}}$ sufficiently close, $\mathcal{N}'$ must have at least $M+1$ linear regions on ℓ. However, $\mathcal{N}'$ can have at most M linear regions in $\mathbb{R}^d$ from the definition of M. This contradiction completes the proof. □

Theorem F.3. *Let $d_x, d_y \in \mathbb{N}$ and $p \geq 1$. Then, the set of three-layer ReLU networks is dense in $L^p(\mathbb{R}^{d_x}, [0,1]^{d_y})$. Furthermore, let $f : \mathbb{R}^{d_x} \to [0,1]^{d_y}$ be a compactly supported function whose Lipschitz constant is L. Then, for any $\varepsilon > 0$, there exists a three-layer ReLU network $\mathcal{N}$ with the architecture*

$$d_x \xrightarrow{\sigma} (2nd_xd_y) \xrightarrow{\sigma} (nd_y) \to d_y$$

such that $\|\mathcal{N} - f\|_{L^p(\mathbb{R}^{d_x})} < \varepsilon$. Here, $n = \varepsilon^{-d_x}\left(1 + (\sqrt{d_x}L)^p\right)^{d_x/p} = O(\varepsilon^{-d_x})$.

Proof. Fist we recall a result in real analysis: the set of compactly supported continuous functions is dense in $L^p(\mathbb{R}^{d_x})$ for $p \geq 1$ (Rudin et al., 1976, Theorem 3.14). Therefore, it is enough to prove the second statement; which claims that any compactly supported Lipschitz function can be universally approximated by three-layer ReLU networks.

We consider $d_y = 1$ case first. Let $f \in \mathbb{R}^{d_x} \to [0,1]$ be Lipschitz, and let L be its Lipschitz constant. Without loss of generality, suppose the support of f is contained in $[0,1]^{d_x}$. Let $\delta > 0$ be the small number which will be determined. Now we partition $[0,1]^{d_x}$ by regular d_x-dimensional cubes with length δ. Now, consider estimating the definite integral using a Riemann sum over these cubes. The total number of cubes are $n := (\frac{1}{\delta})^{d_x}$, and we number these cubes by $C_1, C_2, \cdots, C_n$. For each cube C_i, by Proposition 3.1, we can define a two-layer ReLU network $\mathcal{T}_i$ with the architecture $d_x \xrightarrow{\sigma} 2d_x \to 1$ such that $\mathcal{T}_i(\boldsymbol{x}) = 1$ in C_i and $\mathcal{T}_i(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \notin B_r(C_i)$ with $r := \frac{1}{2d_x}\frac{\delta^{p+1}}{1+\delta^p}$. Then for any $\boldsymbol{x}_i \in C_i$, we get

$$\begin{aligned}
\int_{B_r(C_i)} |f - f(\boldsymbol{x}_i)\mathcal{T}_i|^p \, d\mu &= \int_{C_i} |f - f(\boldsymbol{x}_i)\mathcal{T}_i|^p \, d\mu + \int_{B_r(C_i)\setminus C_i} |f - f(\boldsymbol{x}_i)\mathcal{T}_i|^p \, d\mu \\
&\leq \int_{C_i} (\sqrt{d_x}L\delta)^p \, d\mu + \int_{B_r(C_i)\setminus C_i} 1^p \, d\mu \\
&\leq (\sqrt{d_x}L\delta)^p \cdot \delta^{d_x} + \left[(\delta + 2r)^{d_x} - \delta^{d_x}\right] \\
&= (\sqrt{d_x}L)^p \cdot \delta^{d_x+p} + \left[\left(1 + \frac{2r}{\delta}\right)^{d_x} - 1\right]\delta^{d_x} \\
&< \left[(\sqrt{d_x}L)^p + 1\right]\delta^{d_x+p}.
\end{aligned}$$

Note that we use two inequalities, $|f(\boldsymbol{x}) - f(\boldsymbol{x}_i)| \leq L\sqrt{d_x}\delta$ for $\boldsymbol{x} \in C_i$ and $(1+a)^k < \frac{1}{1-ak}$ for $0 < a < \frac{1}{k}$. Then, the above equation implies the L^p distance between f and $f(\boldsymbol{x}_i)\mathcal{T}_i$ in C_i is bounded by the above value. Now we define a

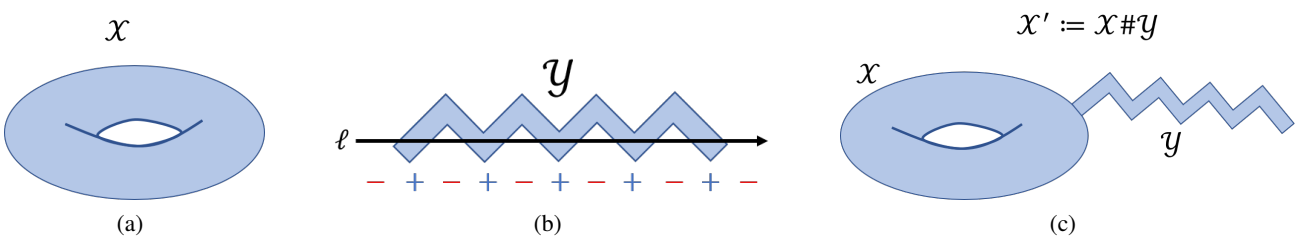


Figure 20. Proof of Proposition F.2. (a) $\mathcal{X}$ is a given topological space, and $\mathcal{A}$ is a feasible architecture on $\mathcal{X}$. (b) $\mathcal{Y}$ is a zig-zag shaped long band, which is a contractible space. There exists a straight line ℓ such that $\mathcal{Y}$ and ℓ has sufficiently many intersection points, so that $\mathcal{A}$ cannot approximate $\mathcal{Y}$. (c) $\mathcal{X}'$ is the connected sum of $\mathcal{X}$ and $\mathcal{Y}$, which is homeomorphic with $\mathcal{X}$. However, $\mathcal{A}$ is not a feasible architecture on $\mathcal{X}'$.

three-layer neural network $\mathcal{N}$ by

$$\mathcal{N}(\boldsymbol{x}) := \sum_{i=1}^{n} f(\boldsymbol{x}_i)\mathcal{T}_i(\boldsymbol{x}),$$

which is a Riemann sum over the n cubes partitions. Then $\mathcal{N}$ has the architecture $d_x \xrightarrow{\sigma} (2nd_x) \xrightarrow{\sigma} n \to 1$ and satisfies

$$\begin{aligned}
\int_{\mathbb{R}^{d_x}} |f - \mathcal{N}|^p d\mu &= \int_{B_r([0,1]^{d_x})} |f - \mathcal{N}|^p \, d\mu \\
&< \sum_{i=1}^{n} \int_{B_r(C_i)} |f - f(\boldsymbol{x}_i)\mathcal{T}_i|^p \, d\mu \\
&\leq \left[(\sqrt{d_x}L)^p + 1 \right] n\delta^{d_x+p}. \\
&= \left[(\sqrt{d_x}L)^p + 1 \right] \delta^p.
\end{aligned}$$

Therefore, take $\delta < \varepsilon(1 + (\sqrt{d_x}L)^p)^{-\frac{1}{p}}$ for given ε, we conclude that $\|f - \mathcal{N}\|_{L^p([0,1]^{d_x})} < \varepsilon$. From this choice of δ, we get

$$n = \delta^{-d_x} > \varepsilon^{-d_x} \left(1 + (\sqrt{d_x}L)^p \right)^{d_x/p} = O(\varepsilon^{-d_x}).$$

If $d_y > 1$, we can obtain the desired network by concatenating d_y networks, thus the architecture is

$$d_x \xrightarrow{\sigma} (2nd_xd_y) \xrightarrow{\sigma} (nd_y) \to d_y.$$

□

Lemma F.4. *Let $\mathcal{T}$ be a two-layer ReLU network defined in (5). Then, the classification region $R := \{\boldsymbol{x} \in \mathbb{R}^d \mid \mathcal{T}(\boldsymbol{x}) > 0\}$ is a convex polytope. Specifically, if the subset $S := \{\boldsymbol{x} \in \mathbb{R}^d \mid \mathcal{T}(\boldsymbol{x}) = \lambda\}$ is nonempty, then it is a convex polytope with m faces.*

Proof. First, we prove that $\mathcal{T}$ is a concave function. Note that σ is convex thus $v_k\sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)$ is a concave function with respect to input $\boldsymbol{x}$, and the sum of concave functions is again concave (we use all $v_k < 0$ here). Therefore, $\mathcal{T}$ is a concave function, and the region $R := \{\boldsymbol{x} \mid \mathcal{T}(\boldsymbol{x}) > 0\}$ is convex. The piecewise linearity of $\mathcal{T}$ implies that R forms a convex polytope.

Now we consider the subset $S := \{\boldsymbol{x} \mid \mathcal{T}(\boldsymbol{x}) = \lambda\}$. Since all $v_k < 0$, $\boldsymbol{x} \in S$ if and only if $\boldsymbol{w}_k^\top \boldsymbol{x} + b_k \leq 0$ for all $k \in [m]$. Then, S is a convex polytope with m faces by Definition 2.1. □

Proposition F.5 (Theorem 2.1 in (Du et al., 2018), two-layer version)**.** *Let $\mathcal{N}(\boldsymbol{x}) := v_0 + \sum_{k=1}^{l} v_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)$ be a two-layer ReLU network, and $L = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathcal{N}(\boldsymbol{x}_i), y_i)$ be the loss function. Then, on the gradient flow, for all $k \in [l]$, the quantity*

$$v_k^2 - \|\boldsymbol{w}_k\|^2 - b_k^2 \tag{41}$$

is invariant.

Proof. The proof is written in (Du et al., 2018), and we provide here for completeness. The gradient of each component is computed by

$$\begin{aligned}
\frac{\partial L}{\partial v_k} &= \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \ell}{\partial \mathcal{N}(\boldsymbol{x}_i)} \cdot \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}_i + b_k), \\
\frac{\partial L}{\partial \boldsymbol{w}_k} &= \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \ell}{\partial \mathcal{N}(\boldsymbol{x}_i)} \cdot v_k \mathbb{1}_{\{\boldsymbol{w}_k^\top \boldsymbol{x}_i + b_k > 0\}} \boldsymbol{x}_i, \\
\frac{\partial L}{\partial b_k} &= \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \ell}{\partial \mathcal{N}(\boldsymbol{x}_i)} \cdot v_k \mathbb{1}_{\{\boldsymbol{w}_k^\top \boldsymbol{x}_i + b_k > 0\}}.
\end{aligned}$$

Then, it is easy to check that

$$v_k \frac{\partial L}{\partial v_k} = \boldsymbol{w}_k^\top (\frac{\partial L}{\partial \boldsymbol{w}_k}) + b_k \cdot \frac{\partial L}{\partial v_k}.$$

Now, we differentiate (41). It gives

$$\begin{aligned}
\frac{d}{dt}(v_k^2 - \|\boldsymbol{w}_k\|^2 - b_k^2) &= 2v_k \frac{dv_k}{dt} - 2\boldsymbol{w}_k^\top (\frac{d\boldsymbol{w}_k}{dt}) - 2b_k \frac{db_k}{dt} \\
&= 2\left(-v_k \frac{\partial L}{\partial v_k} + \boldsymbol{w}_k^\top (\frac{\partial L}{\partial \boldsymbol{w}_k}) + b_k \frac{\partial L}{\partial b_k}\right) \\
&= 0
\end{aligned}$$

for all t. Therefore, (41) is constant. □

Proposition F.6. *Consider the neural network architecture $d \xrightarrow{\sigma} d_1 \xrightarrow{\sigma} d_2 \xrightarrow{\sigma} \cdots \xrightarrow{\sigma} d_D \to 1$ and a convex polytope C with m faces. Then,*

1. *if $d_1 \geq m$ and $d_2 \geq 1$, then it is a feasible architecture on C.*
2. *if $\max_j d_j \leq m - 1$, then it may not be a feasible architecture on some polytope C.*
3. *if $d_1 \leq m - 2$, then it may not be a feasible architecture on some polytope C.*

Proof. The proof is accomplished by two strategies: for a feasible architecture, we explicitly construct such neural networks. For the negative statements, we prove them by providing some counterexamples.

1. Proposition 3.1 shows that $d \xrightarrow{\sigma} m \to 1$ is a feasible architecture . Therefore, for $d_1 \geq m$, then taking the identity for all other layers, it becomes a feasible architecture .
2. When $d_1 \leq m-1$, there is a m-faces convex polytope C that cannot be approximated by the given network architecture. The simplest example is a half-space ($m = 1$).

   Below, we provide another non-trivial example: Let C be a d-simplex in $\mathbb{R}^d$, thus $m = d + 1$. Suppose $\max_j d_j \leq m - 1 = d$. Then, by Lemma F.7, we conclude that the classified regions are always unbounded. Therefore, it cannot approximate a bounded polytope C.

3. From the above proof, recall the d-simplex C (thus $m = d + 1$). If $d_1 \leq m - 2 = d - 1$, we provide a counterexample proving that it cannot be approximated by a ReLU network with the architecture $d \xrightarrow{\sigma} (m-2) \xrightarrow{\sigma} \cdots \xrightarrow{\sigma} d_D \to 1$. Let $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_{m-2}$ be the weight vectors of the first layer. Since the dimension of $span < \boldsymbol{w}_k >$ is equal or less than $m - 2 = d - 1$, there exists a nonzero vector $\hat{\boldsymbol{w}} \in span < \{\boldsymbol{w}_k\}_{k \in [m]} >^{\perp}$. In other words, $\hat{\boldsymbol{w}}^{\top} \boldsymbol{w}_k = 0$ for all $k \in [m-2]$. Then, it implies $\mathcal{N}(\boldsymbol{x} + t\hat{\boldsymbol{w}}) = \mathcal{N}(\boldsymbol{x})$ for all $t \in \mathbb{R}$. Therefore, $\mathcal{N}$ cannot approximate the bounded polytope C.

□

Lemma F.7. *Let $\mathcal{N}$ be a deep ReLU network where all hidden dimension is equal or smaller than the input dimension d. Suppose $\mu(\{\mathcal{N}(\boldsymbol{x}) > 0\}) > 0$. Then, $\mu(\{\mathcal{N}(\boldsymbol{x}) > 0\})$ is either 0 or ∞. In other words, the classification region is either measure-zero or unbounded.*

Proof. Beise et al. (2021, Theorem 2) showed that if all hidden layers have width equal or smaller than the input dimension, then the connected components of every decision region are unbounded. Therefore, $\mu(\{\mathcal{N}(\boldsymbol{x}) > 0\})$ is either 0 or ∞, depends on whether $\{\mathcal{N}(\boldsymbol{x}) > 0\}$ is empty or not. □