ON FUNCTIONAL DIMENSION OF UNIVARIATE RELU NEURAL NETWORKS

Zhen Liang

A dissertation

submitted to the Faculty of

the department of Mathematics

in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Boston College
Morissey College of Arts and Sciences
Graduate School

June 2024

ON FUNCTIONAL DIMENSION OF UNIVARIATE RELU NEURAL NETWORKS

Zhen Liang

Advisor: J. Elisenda Grigsby

The space of parameter vectors for a feedforward ReLU neural networks with any fixed architecture is a high dimensional Euclidean space being used to represent the associated class of functions. However, there exist well-known global symmetries and extra poorly-understood hidden symmetries which do not change the neural network function computed by network with different parameter settings. This makes the true dimension of the space of function to be less than the number of parameters. In this thesis, we are interested in the structure of hidden symmetries for neural networks with various parameter settings, and particular neural networks with architecture $(1, n, 1)$.

For this class of architectures, we fully categorize the insufficiency of local functional dimension coming from activation patterns and give a complete list of combinatorial criteria guaranteeing a parameter setting admits no hidden symmetries coming from slopes of piecewise linear functions in the parameter space. Furthermore, we compute the probability that these hidden symmetries arise, which is rather small compared to the difference between functional dimension and number of parameters. This suggests the existence of other hidden symmetries. We investigate two mechanisms to explain this phenomenon better.

Moreover, we motivate and define the notion of ε-effective activation regions and ε-effective functional dimension. We also experimentally estimate the difference between ε-effective functional dimension and true functional dimension for various parameter settings and different ε.

Contents

ACKNOWLEDGEMENTS

I would like to gratefully thank my advisor, Eli Grigsby, for the academic guidance and support. I would like to thank her for her generosity with her ideas, and for her patience and advice for this work. This thesis would not have appeared without her.

A special thanks to my cat, 银河, for generously offering one year of his life with me abroad during the transition of my life.

Finally, I would like to express my deep gratitude to my parents for their understanding and patience over the past years, but more important, for making my experience in this loka possible.

幽獨自辨。

Chapter 1

Introduction

1.1 Background and motivations

The parametrized family of ReLU neural network functions is a class of piecewise linear functions with finitely many pieces parametrized by $\mathbb{R}^D$, where D is large and determined by the architecture of the neural network. However, the parameter space of a neural network of any fixed architecture has well-known global symmetries, which don't change the function computed by a neural network. In particular, the true dimension of the space of function is less than the number of parameters. [Grigsby et al., 2022] introduced the notion of the *functional dimension* $\dim_{fun}(\theta)$ of a parameter $\theta \in \Theta$. Roughly speaking, for a randomly chosen parameter $\theta_0 \in \Theta$ satisfying certain conditions, the functional dimension is the dimension of subspace $T_{\theta_0}(\Theta)$ spanned by gradients of components of evaluations at an input data point. These are the only directions that one can move when doing gradient descent near θ_0. Functional dimension also serves as an important tool in practice, as deep learning heavily relies on neural networks in expressing functions. Networks of a given architecture are determined by weights and biases parameters, and the space of parameter vectors can be used to study the space of neural network functions of the

given architecture. But the functional dimension changes across parameter space due to hidden symmetries, which are poorly understood. The purpose of this thesis is to investigate the structure of hidden symmetries for a ReLU neural network with architecture $(1, n, 1)$, and have some further discussions on deeper bias-free networks with input dimension 1.

The collection of activation patterns serves as a discrete toy model for neural network. More specifically, we can replace the outputs of the layers with their signs to form a discrete model. If for this discrete model, some sign can be represented by combination of other signs, then sometimes the local functional dimension determined by this discrete model is lower than the theoretical upper bond on functional dimension. We fully categorize the insufficiency of local functional dimension coming from this discrete model for neural networks with architecture $(1, n, 1)$. We investigate two mechanisms that might explain the gap between the theoretical upper bond and the observed behavior of random networks in [Grigsby et al., 2023].

Since we choose bias parameters from a distribution with smaller variance, one natural direction is to study hidden symmetries in a parameter setting by considering only the unbounded activation regions, which is equivalent to considering bias-free neural networks.

In practice, a more natural choice is to only consider activation regions that are large enough. That is, we eliminate those activation regions that are so small that they have a very low probability of containing any input data points, and hence are negligible when computing functional dimension on finite batches of points chosen i.i.d for a fixed probability distribution.

We introduce the notions of ε-effective activation regions, which are activation regions with probability measure greater than ε, and ε-effective functional dimension, which is the functional dimension computed by considering only ε-effective activation regions. This allows us to have a more reasonable calculation since in experiments we

noticed that those activation regions with small probability measure rarely contain any input data point(s) and hence do not contribute to the calculation of functional dimension. We also compare the behavior of ε-effective functional dimension and true functional dimension, and the difference between them.

Key results in this thesis are as follows:

- For $(1, n, 1)$ architecture, we give a complete set of combinatorial criteria on the activation patterns ensuring that a parameter has no hidden symmetries coming from the activation patterns, and prove that the list of criteria is complete. This allows us to compute that for a ReLU neural network function with architecture $(1, n, 1)$, the probability that there exist hidden symmetries among the slopes of piecewise linear functions is $1 - 4^{-n}(n + 1)$.

- We compute the upper bounds on expected functional dimension for a bias-free neural network with architecture $(1, n, 1)$ and corroborate our computation with experiments.

- We establish empirically that the sizes of roughly half of the activation regions follow a power law with respect to the width, and empirically calculate ε-effective sample size, which is the expected number of points landing in ε-heavy regions. These experiments motivate the notion of ε-effective activation regions and ε-effective functional dimension, which are formally defined in Section 3.3.

- For architecture $(1, n, 1)$, we analyze the total measure (out of 1) of the union of ε-effective activation regions as a function of n and ε, and experimentally compute effective functional dimension and its difference compared to the true functional dimension for some setting. Our results suggest that this difference decreases with larger batch sample size of input data points, or with a smaller ε in the definition of ε-effective functional dimension.

1.2 Related work

The functional dimension of a neural network function was defined and studied in [Grigsby et al., 2022], and independently studied by [Bona-Pellissier et al., 2022]. In [Grigsby et al., 2022], the authors study the functional dimension of a network parameter setting, that is, the dimensional complexity of the space of functions that can be achieved by infinitesimally perturbing the parameters. More specifically, for a feedforward ReLU neural network F of architecture $(n_0, n_1, \ldots, n_d)$ parametrized by $\mathbb{R}^D$, where $D = \sum_{i=1}^{d} n_i(n_{i_1} + 1)$, they define a realization map

$$\rho \colon \Theta \to \{\text{Finite PL functions } \mathbb{R}^{n_0} \to \mathbb{R}^{n_d}\}$$

by $\theta \mapsto F_\theta$, where F_θ is a neural network function defined on the parameter setting θ. It is well-known that this realization map ρ is far from injective (cf. [Phuong and Lampert, 2020]), it has a positive dimensional space of redundancies coming from scaling and inverse-scaling the input and output paths to each neuron in all the hidden layers and the output layer. [Rolnick and Kording, 2020] show, by examining the geometry of the decomposition of the domain into linear regions given by the bent hyperplane arrangement of a parameter, for depth-2 neural networks the parameters can be reverse-engineered up to permutation and scaling, so have no hidden symmetries. [Phuong and Lampert, 2020] use similar methods to prove that there exists a parameter in every non-widening ($n_0 \geq n_1 \geq \ldots \geq n_d$) architecture for which reverse-engineering is possible, hence has no hidden symmetries. [Grigsby et al., 2022] prove that the degree to which the realization map ρ fails to be injective for a neural network of a fixed architecture with ReLU activation functions is inhomogeneous; that is, the dimension of the space of redundancies varies with the parameter. In some regions of parameter space, there exists a comparatively high-dimensional set of parameters that define the same function, while in other regions

of parameter space, this set of parameters that correspond to a single function is comparatively low-dimensional.

The goal of this thesis is to understand sources of symmetries in the parameter space to improve the theoretical upper bound on functional dimension determined by [Grigsby et al., 2022]. They prove this upper bound by considering dimensional redundancies coming from well-known global symmetries in the computational graphs, that is, for any activated neuron in a hidden layer n_l, multiply and reverse-multiply a scaling factor λ. It has been observed that for any fully connected neural network with ReLU activation functions, this type of symmetry does not change the function computed by the neural network (cf. [Phuong and Lampert, 2020]; [Rolnick and Kording, 2020]).

Furthermore, the above theoretical upper bound is conjectured in [Grigsby et al., 2023] to be achieved for almost all parameter settings without hidden symmetries. We have a relevant observation showing that empirically, as we are considering more batches of input data points in the process of computing ε-effective functional dimension, the above theoretical upper bound can almost be achieved.

In practice, the main problem of achieving the theoretical upper bound on functional dimension is the existence of those activation regions not containing any input data points. We observed empirically, and theoretically for the single-hidden layer case, that the size of the bounded activation regions, when arranged in descending order, follow a power scaling law for around half of the break points, at which point the sizes of the regions drop precipitously. This neural scaling law phenomenon was originally discovered and studied empirically by [Kaplan et al., 2020].

The empirical behavior of functional dimension was also studied by [Bona-Pellissier et al., 2022]. They defined *computational functional dimension* and proved that as the sample size gets larger, the probability that the computational

functional dimension differs from true functional dimension converges to zero. We also verify and experimentally show that for a fixed batch size, the difference between ε-effective batch functional dimension and batch functional dimension is bounded. Moreover, this difference appears to achieve its maximum for some finite batch size, and then decay and approach 0.

1.3 Notation and background

Let the rectified linear unit function $\mathrm{ReLU}\colon \mathbb{R} \to \mathbb{R}$ be defined by $\mathrm{ReLU}(x) = \max\{0, x\}$, and let $\sigma\colon \mathbb{R}^n \to \mathbb{R}^n$ be the map that applies ReLU to each coordinate.

Definition 1.3.1. *A* feedforward neural network defined on $\mathbb{R}^{n_0}$ with ReLU activation function applied to all hidden layers *is a finite collection of affine maps*

$$A^i\colon \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i},$$

for $i = 1, \ldots, j$ and for $n_i \in \mathbb{N}$ a finite sequence of natural numbers. It has an associated neural network map $F\colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_j}$ *of architecture $(n_0, n_1, \ldots, n_j) \in \mathbb{N}^{j+1}$ given by the composition*

$$F(x) \coloneqq \mathbb{R}^{n_0} \xrightarrow{F_1=\sigma\circ A_1} \mathbb{R}^{n_1} \xrightarrow{F_2=\sigma\circ A_2} \mathbb{R}^{n_2} \xrightarrow{F_3=\sigma\circ A_3} \cdots \xrightarrow{F_j=\sigma\circ A_j} \mathbb{R}^{n_j}.$$

F is said to have depth j and width $\max n_0, n_1, \ldots, n_j$. *The k^{th} layer map of F is the composition $F_k = \sigma \circ A_k$ for $k = 1, \ldots, j$.*

It follows that there exists a natural map from $\mathbb{R}^D = \mathbb{R}^{\sum_{i=1}^{j} n_i(n_{i-1}+1)}$ to the set of neural networks of architecture $n_0, n_1, \ldots, n_j$. This realization map ρ is the map that forms a neural network function by using the coordinates of the point in $\mathbb{R}^D$ to determine affine maps $A^1, \ldots, A^j$.

Now consider the parametrized neural network function. The parameters of a neural network are the weights and biases that are learned from the training data and determine the accuracy of the network's predictions. Let w_{ij}^l be the weight associated to the map connecting the i^{th} neuron of layer $l-1$ and the j^{th} neuron of layer l. Similarly, let b_i^l be the bias associated to the i^{th} neuron in the l^{th} layer. It follows that a neural network map F of architecture $(n_0, \ldots, n_j)$ is defined by

$$D \coloneqq \sum_{i=1}^{j} (n_{i-1} + 1) n_i$$

parameters. Now we can define the parameter space associated to F.

Definition 1.3.2. *The* parameter space *of a neural network of architecture* $(n_0, \ldots, n_j) \in \mathbb{N}^{j+1}$ *is the Euclidean space*

$$\Theta_{n_0 \ldots, n_j} \coloneqq \mathbb{R}^D.$$

Definition 1.3.3. *A* realization map

$$\rho \colon \Theta_{n_0, \ldots, n_j} \to C(\mathbb{R}^{n_0}, \mathbb{R}^{n_j})$$

is defined by sending a parameter $\theta \in \Theta_{n_0, \ldots, n_j}$ *to its associated neural network function* $F_\theta \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_j}$ *given by*

$$F(x) \coloneqq \sigma \circ A^m \circ \cdots \circ \sigma \circ A^1(X).$$

Following the notation in [Grigsby et al., 2022], let $\overline{\rho}$ denote the marked realization map that associates a specific point in the parameter space to its corresponding marked neural network function, i. e., $\overline{\rho}$ records the information of those nested maps with parameter coordinates. Similarly, denote the unmarked realization map by ρ.

We briefly recall some relevant background about hyperplane arrangement, refer-

ring reader to [Grigsby et al., 2022], [Stanley, 2007] for more details.

Definition 1.3.4. *The affine solution set arrangement associated to one layer of a neural network function is the finite ordered set* $S = \{S_1, \dots, S_j\}$. *S is said to be generic if the intersection of all subsets* $\cap_{j=1}^{k} S_{i_j}$ *is an affine linear subspace of* $\mathbb{R}^i$ *of dimension* $i-k$. *In particular, if for* $1 \leq k \leq j$, *every solution set* S_k *has codimension* 1 *in* $\mathbb{R}^i$, *then S is the* hyperplane arrangement *associated to that layer map.*

Definition 1.3.5. *For a canonical polyhedral complex* $\mathcal{C}(\overline{\rho}(\theta))$ *([Grigsby et al., 2022]), the* ternary labeling *of a point* $x \in \mathbb{R}^{n_0}$ *is the sequence of ternary tuples*

$$s_x \coloneqq (s_x^1, \dots, s_x^j) \in \{-1, 0, 1\}^{n_1 + \dots + n_j}$$

representing the sign of the pre-activation output $A^l \circ F^{l-1} \circ \dots \circ F^1(x)$ *of each neuron of* $\overline{\rho}(\theta)$ *at x for the first l layer maps.*

For the purpose of this thesis, we will use binary labeling instead of ternary labeling. We consider both the -1's and 0's in a ternary labeling to be 0.

Definition 1.3.6. *Let* $\theta \in \Theta_{n_0, \dots, n_j}$ *be a parameter. An* activation region *for θ is a maximal connected component of the set of input data* $x \in \mathbb{R}^{n_0}$ *where the ternary labeling has no* 0*s.*

Note that the activation regions coincide with interiors of the n_0-cells of $\mathcal{C}(\overline{\rho}(\theta))$ in this paper, since we will only consider generic and transversal network functions (as proved in [Grigsby and Lindsey, 2021], and see [Grigsby et al., 2022] for definitions for a generic and transversal network). Moreover, the ternary labeling is always constant on the interior of each cell of $\mathcal{C}(\overline{\rho}(\theta))$, and we say the i-th neuron in the l-th layer is *turned off* at a point x if the sign of the pre-activation output of the first l layer maps at x for the i-th neuron is non-positive (with the convention that sgn(0)=0).

Definition 1.3.7. *Fix an ordered set $Z = \{z_1, \ldots, z_k\}$ of finitely many points in $\mathbb{R}^{n_0}$. Define the* evaluation map *$E_Z \colon \Theta_{n_0,\ldots,n_j} \to \mathbb{R}^{k \cdot n_j}$ by*

$$E_Z(\theta) = (\rho_1(\theta)(z_1), \ldots, \rho_{n_j}(\theta)(z_1), \ldots, \rho_1(\theta)(z_k), \ldots, \rho_{n_j}(\theta)(z_k)).$$

Now recall the definition of parametrically smooth points and ordinary parameters:

Definition 1.3.8. *Let $x \in \mathbb{R}^{n_0}$ and $\theta \in \Theta_{n_0,\ldots,n_j}$. The* parametrized family *of a neural network function of architecture $(n_0, \ldots, n_j)$ is the map*

$$\mathcal{F}_{n_0,\ldots,n_j} \colon \Theta_{n_0,\ldots,n_j} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_j}$$

given by

$$\mathcal{F}_{n_0,\ldots,n_j}(\theta, x) = \rho(\theta)(x).$$

Definition 1.3.9. *Fix an architecture $(n_0, \ldots, n_j)$. Let $\theta \in \Theta_{n_0,\ldots,n_j}$ be a point in the parameter space. A point $x \in \mathbb{R}^{n_0}$ is* parametrically smooth *for θ if (θ, x) is a smooth point for $\mathcal{F}_{n_0,\ldots,n_j}$.*

It was shown in [Grigsby and Lindsey, 2021] that the set consisting of all generic and transversal parameters has full Lebesgue measure. Moreover, any generic, transversal parameter has a full measure set of parametrically smooth points. At this time, we assume θ is generic and transversal unless stated explicitly otherwise.

Definition 1.3.10. *A set $Z \subset \mathbb{R}^{n_0}$ is called* decisive *for $\theta \in \Theta_{n_0,\ldots,n_m}$ if for every top-dimensional polyhedron $C \in \mathcal{C}(\theta)$, Z contains precisely $n_0 + 1$ points in the interior of C whose set of pairwise differences contains a basis for $\mathbb{R}^{n_0}$.*

It follows from Lemma 3.12 in [Grigsby et al., 2022] that if θ is generic and transversal, a decisive set Z for θ consists of parametrically smooth points for θ.

Recall that the rank of a smooth map is the rank of its Jacobian matrix of partial derivatives. Choose $z \in \mathbb{R}^{n_0}$ to be a parametrically smooth point for a parameter $\theta \in \Theta_{n_0,\ldots,n_m}$. Then the Jacobian matrix of the evaluation map E_z evaluated at θ is

$$JE_z|_\theta = \left[\frac{\partial (E_z)_i}{\partial \theta_j}\right]\Bigg|_\theta,$$

which is the $n_m \times D$ matrix whose entry in the i-th row and j-th column records the partial derivative of the i-th coordinate of E_z at θ with respect to the j-th parameter.

Definition 1.3.11. *Let $\theta \in \Theta_{n_0,\ldots,n_j}$ be an ordinary parameter. The* batch functional dimension *of θ for a batch of input data points $Z \subset \mathbb{R}^{n_0}$ of parametrically smooth points for θ is*

$$\dim_{fun}(\theta) \coloneqq \text{rank} \boldsymbol{J} E_Z|_\theta.$$

Definition 1.3.12. *Let $\theta \in \Theta_{n_0,\ldots,n_j}$ be an ordinary point. The* (full) functional dimension *at θ is defined to be*

$$\dim_{fun}(\theta) \coloneqq \sup_{Z \subset \mathbb{R}^{n_0} \text{ is finite and parametrically smooth at } \theta} \text{rank} \boldsymbol{J} E_Z|_\theta.$$

For the sake of brevity, we will refer to the full functional dimension as the functional dimension. See [Grigsby et al., 2022] for definitions of stochastic functional dimension and batch functional dimension.

Definition 1.3.13. *(cf. [Grigsby et al., 2023]) The* augmented *computational graph $\widetilde{G}$ for the feedforward ReLU network architecture $(n_0, \ldots, n_m)$ is the graded oriented graph:*

(a) with n_l ordinary vertices and 1 distinguished vertex of grading l for $l = 0, \ldots, m-1$, and n_m ordinary vertices of grading m,

(b) for every $l = 0, \ldots, m-1$, every vertex of grading l is connected by a single

oriented edge to every ordinary vertex of grading $l+1$, oriented toward the vertex of grading $l+1$.

An augmented computational graph for an architecture can be obtained from the standard computational graph for the architecture by isolating the bias variable from each layer and mark the biases as an extra vertex on the top of each layer. Now given a parameter $\theta \in \Theta$, we label the edges of the augmented computational graph in the following way:

- the edge from the i-th ordinary vertex of layer $l-1$ to the j-th ordinary vertex of layer l is labeled with the weight parameter w_{ij}^l,
- the edge from the distinguished vertex of layer $l-1$ to the k-th ordinary vertex of layer l is labeled with the k-th component of the bias vector for F^l, b_k^l.

Associated to every oriented path γ is a corresponding monomial, $m(\gamma)$, in the parameters obtained by taking the product of the weight parameters on the edges traversed along γ.

Definition 1.3.14. *Let $\theta \in \Theta$ be a generic, transversal parameter, and let $x \in \mathbb{R}$ be an input data point with associated binary labeling $s_x = (s_x^1, \ldots, s_x^m)$. A path γ is said to be* open at the point x for parameter θ *if every node along γ has binary labeling* 1.

For illustration, we give an example of an augmented computational graph for architecture $(1, 3, 3, 1)$.

Figure 1.1 shows an augmented computational graph corresponding to the activation pattern associated to an input vector x with binary label

$$s_x = (s_x^1, s_x^2, s_x^3) = ((0, 1, 0), (1, 0, 1), (1)).$$

The ordinary vertices are filled-in circles, with red meaning the neuron is activated and green means the neuron is inactive. The distinguished vertices are solid squares.

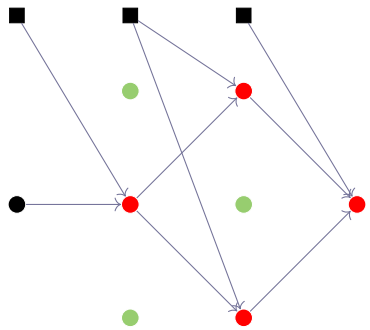

Figure 1.1: An augmented computational graph for architecture $(1, 3, 3, 1)$

Edges connecting ordinary vertices are labeled with weights, and those connecting distinguished and ordinary vertices are labeled with biases. A complete path is one that ends at an output vertex and begins at either an input vertex, or a bias vertex. The above figure draws all the open paths associated to this binary label with solid edges. In particular, there are two open paths γ_1, γ_2 with associated monomials $m(\gamma_1) = w^3_{11}w^2_{21}w^1_{12}$, and $m(\gamma_2) = w^3_{31}w^2_{23}w^1_{12}$.

Chapter 2

$\mathbb{Z}$-linear dependence relations among activation patterns for architecture $(1, n, 1)$

2.1 $\mathbb{Z}$-linear dependence relations in neural networks of architecture $(1, n, 1)$

A neural network with architecture $(1, n, 1)$ has functional dimension at most $2n + 1$. This immediately follows from Theorem 7.1 in [Grigsby et al., 2022].

Theorem 2.1.1. *[Grigsby et al., 2022] For any architecture $(n_0, \dots, n_m)$,*

$$\dim \Theta_{n_0,\dots,n_m} \leq n_m + \sum_{i=0}^{m-1} n_i n_{i+1}.$$

Fix a batch of parametrically smooth points $Z = \{z_1, \dots, z_m\} \subset \mathbb{R}^{n_0}$. If we choose a parameter $\theta \in \Theta$, the batch functional dimension measures locally in $T\Theta_\theta$, the number of directions that one can move to change the value of the function at θ on the batch Z. We can then consider the dimension of the subspace of $T_\theta(\Theta)$

spanned by gradients of components of evaluations at z_i, which are the only possible directions to move when doing gradient descent. The dimension of this subspace of $T_\theta(\Theta)$ is precisely the rank of the Jacobian matrix

$$\text{rank}(\mathbf{J}E_Z|_\theta).$$

Note that the ReLU activation function is not always differentiable, but we need to use the Jacobian matrix $\mathbf{J}E_Z$ to compute partial derivatives of $\rho(\theta)(x)$ for a fixed input data point x in the domain of $\rho(\theta)$ with respect to the coordinates of θ. The discussion in [Grigsby et al., 2022] guarantees the locus of differentiability with respect to both coordinates of θ and x. The idea is, as mentioned in Section 1.3, every generic, transversal parameter has parametrically smooth points away from a Lebesgue measure zero set (in particular, points picked from activation regions are parametrically smooth), and the measure of its parametrically smooth points is full. We refer readers to Theorem 3.16 in [Grigsby et al., 2022] for the proof and more details.

Now consider a single-layer neural network, that is, a neural network function with architecture $(1, n, 1)$. The corresponding parameter space is $\Theta = \mathbb{R}^{3n+1}$, whose dimension is higher than the theoretical upper bound on functional dimension guaranteed by Theorem 2.1.1 (in this case $2n+1$). Hence we are interested in the structure of hidden symmetries for a neural network with architecture $(1, n, 1)$. The columns of its Jacobian matrix record partial derivatives with respect to weight and bias parameters from the two layers, and weights w^2_{ij} from the second layer can be written as a linear combination of weights and biases from the first layer. We motivate the study of the structure of hidden symmetries for a network with architecture $(1, n, 1)$ with the following example.

Example 2.1.2.

Let F be a neural network function of architecture $(1, n, 1)$, where $n \in \mathbb{N}$. Fix a generic, transversal parameter $\theta \in \Theta_{1,n,1}$ with coordinates

$$\theta = (w_{11}^1, b_1^1, w_{12}^1, b_2^1, \ldots, w_{1n}^1, b_n^1, w_{11}^2, w_{21}^2, \ldots, w_{n1}^2, b_1^2).$$

Then the function $\overline{\rho}(\theta)\colon \mathbb{R} \to \mathbb{R}$ has the form

$$\overline{\rho}(\theta)(x) = \sigma(w_{11}^2\sigma(w_{11}^1 x + b_1^1) + w_{21}^2\sigma(w_{12}^1 x + b_2^1) + \cdots + w_{n1}^2\sigma(w_{1n}^1 x + b_n^1) + b_1^2).$$

It follows that the function $\overline{\rho}(\theta)$ is continuous and piecewise linear for each specific parameter $\theta \in \Theta_{n_0,\ldots,n_j}$, with $n+1$ regions since θ is generic and transversal. The bend points (where $\overline{\rho}(\theta)$ is non-linear) are $\frac{-b_i^1}{w_{1i}^1}$. Now suppose w_{ij}^2, $b_j^2 > 0$ for θ, and the binary labeling alternates for the first region (without loss of generality, suppose that the binary labeling starts with 0 and ends with 1): $(0, 1, 0, 1, \ldots, 0, 1)$. It then follows that the binary labeling on the 1-cells associated to the $n+1$ regions are:

$$\begin{aligned}
\text{First region: } s &= (s^1) \\
&= ((0, 1, 0, 1, \ldots, 0, 1)) \\
\text{Second region: } s &= ((1, 1, 0, 1, \ldots, 0, 1)) \\
\text{Third region: } s &= ((1, 0, 0, 1, \ldots, 0, 1)) \\
&\cdots\cdots \\
n^{th} \text{ region: } s &= ((1, 0, 1, 0, \ldots, 1, 1)) \\
n+1^{th} \text{ region: } s &= ((1, 0, 1, 0, \ldots, 1, 0))
\end{aligned}$$

We can then write down the 1-row matrix $\mathbf{J}E_{\{x\}}$ for a generic point x in each

region:

for x in the first region:

$$\begin{bmatrix} 0 & 0 & w_{21}^2 x & w_{21}^2 & \cdots & w_{n1}^2 x & w_{n1}^2 & 0 & w_{12}^1 x + b_2^1 & \cdots & w_{1n}^1 x + b_n^1 & 1 \end{bmatrix}$$

for x in the second region:

$$\begin{bmatrix} w_{11}^2 x & w_{11}^2 & w_{21}^2 x & w_{21}^2 & \cdots & w_{n1}^2 x & w_{n1}^2 & w_{11}^1 x + b_1^1 & w_{12}^1 x + b_2^1 & \cdots & w_{1n}^1 x + b_n^1 & 1 \end{bmatrix}$$

$\cdots \qquad \cdots$

for x in the n^{th} region:

$$\begin{bmatrix} w_{11}^2 x & w_{11}^2 & 0 & 0 & \cdots & w_{n1}^2 x & w_{n1}^2 & w_{11}^1 x + b_1^1 & 0 & \cdots & w_{1n}^1 x + b_n^1 & 1 \end{bmatrix}$$

for x in the $n + 1^{th}$ region:

$$\begin{bmatrix} w_{11}^2 x & w_{11}^2 & 0 & 0 & \cdots & 0 & 0 & w_{11}^1 x + b_1^1 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Now let Z be a set of parametrically smooth points for θ. In particular, denote by

$$\begin{aligned} Z &= \{z_1, \ldots, z_m\} \\ &= \{x^1, \ldots, x_{m_1}^1, \ldots, x_1^{n+1}, \ldots, x_{m_{n+1}}^{n+1}\} \end{aligned}$$

the input data points in the activation regions. Thus in this case, the Jacobian matrix

$\mathbf{J}E_Z|_\theta$ used for computing the batch functional dimension has expression

$$\begin{bmatrix}
0 & 0 & w_{21}^2 x_1^1 & w_{21}^2 & \cdots & w_{n1}^2 x_1^1 & w_{n1}^2 & 0 & w_{12}^1 x_1^1 + b_2^1 & \cdots & w_{1n}^1 x_1^1 + b_n^1 & 1 \\
\vdots & & & & & & & & & & & \vdots \\
0 & 0 & w_{21}^2 x_{z_1}^1 & w_{21}^2 & \cdots & w_{n1}^2 x_{z_1}^1 & w_{n1}^2 & 0 & w_{12}^1 x_{z_1}^1 + b_2^1 & \cdots & w_{1n}^1 x_{z_1}^1 + b_n^1 & 1 \\
w_{11}^2 x_1^2 & w_{11}^2 & w_{21}^2 x_1^2 & w_{21}^2 & \cdots & w_{n1}^2 x_1^2 & w_{n1}^2 & w_{11}^1 x_1^2 + b_1^1 & w_{12}^1 x_1^2 + b_2^1 & \cdots & w_{1n}^1 x_1^2 + b_n^1 & 1 \\
\vdots & & & & & & & & & & & \vdots \\
w_{11}^2 x_{z_2}^2 & w_{11}^2 & w_{21}^2 x_{z_2}^2 & w_{21}^2 & \cdots & w_{n1}^2 x_{z_2}^2 & w_{n1}^2 & w_{11}^1 x_{z_2}^2 + b_1^1 & w_{12}^1 x_{z_2}^2 + b_2^1 & \cdots & w_{1n}^1 x_{z_2}^2 + b_n^1 & 1 \\
\vdots & & & & & & & & & & & \vdots \\
\vdots & & & & & & & & & & & \vdots \\
w_{11}^2 x_1^n & w_{11}^2 & 0 & 0 & \cdots & w_{n1}^2 x_1^n & w_{n1}^2 & w_{11}^1 x_1^n + b_1^1 & 0 & \cdots & w_{1n}^1 x_1^n + b_n^1 & 1 \\
\vdots & & & & & & & & & & & \vdots \\
w_{11}^2 x_{z_n}^n & w_{11}^2 & 0 & 0 & \cdots & w_{n1}^2 x_{z_n}^n & w_{n1}^2 & w_{11}^1 x_{z_n}^n + b_1^1 & 0 & \cdots & w_{1n}^1 x_{z_n}^n + b_n^1 & 1 \\
w_{11}^2 x_1^{n+1} & w_{11}^2 & 0 & 0 & \cdots & 0 & 0 & w_{11}^1 x_1^{n+1} + b_1^1 & 0 & \cdots & 0 & 1 \\
\vdots & & & & & & & & & & & \vdots \\
w_{11}^2 x_{z_{n+1}}^{n+1} & w_{11}^2 & 0 & 0 & \cdots & 0 & 0 & w_{11}^1 x_{z_{n+1}}^{n+1} + b_1^1 & 0 & \cdots & 0 & 1
\end{bmatrix}.$$

Notice that

$$\left(w_{ji}^2 x\right)\left(\frac{w_{ij}^1}{w_{ji}^2}\right) = w_{ij}^1 x$$

and

$$w_{ji}^2\left(\frac{b_j^1}{w_{ji}^2}\right) = b_j^1.$$

If we apply the above operations to columns of weights and biases from the first layer, i. e.,

$$\left(w_{ji}^2 x\right)\left(\frac{w_{ij}^1}{w_{ji}^2}\right) + w_{ji}^2\left(\frac{b_j^1}{w_{ji}^2}\right) = w_{ij}^1 x + b_j^1,$$

which is exactly the expression of the column of w_{ij}^2. This is to say, the column vector of polynomials which are the partial derivatives corresponding to the column associated to w_{ij}^2 is an $\mathbb{R}$-linear combination of the columns associated to w_{ij}^1 and b_j^1,

i. e., $\mathbf{J}E_Z|_\theta$ doesn't need to contain those columns of w_{ij}^2 to compute the functional dimension:

$$\begin{bmatrix}
0 & 0 & w_{21}^2 x_1^1 & w_{21}^2 & \cdots & w_{n1}^2 x_1^1 & w_{n1}^2 & 0 & 0 & \cdots & 0 & 1 \\
\vdots & & & & & & & & & & & \vdots \\
0 & 0 & w_{21}^2 x_{z_1}^1 & w_{21}^2 & \cdots & w_{n1}^2 x_{z_1}^1 & w_{n1}^2 & 0 & 0 & \cdots & 0 & 1 \\
w_{11}^2 x_1^2 & w_{11}^2 & w_{21}^2 x_1^2 & w_{21}^2 & \cdots & w_{n1}^2 x_1^2 & w_{n1}^2 & 0 & 0 & \cdots & 0 & 1 \\
\vdots & & & & & & & & & & & \vdots \\
w_{11}^2 x_{z_2}^2 & w_{11}^2 & w_{21}^2 x_{z_2}^2 & w_{21}^2 & \cdots & w_{n1}^2 x_{z_2}^2 & w_{n1}^2 & 0 & 0 & \cdots & 0 & 1 \\
\vdots & & & & & & & & & & & \vdots \\
\vdots & & & & & & & & & & & \vdots \\
w_{11}^2 x_1^n & w_{11}^2 & 0 & 0 & \cdots & w_{n1}^2 x_1^n & w_{n1}^2 & 0 & 0 & \cdots & 0 & 1 \\
\vdots & & & & & & & & & & & \vdots \\
w_{11}^2 x_{z_n}^n & w_{11}^2 & 0 & 0 & \cdots & w_{n1}^2 x_{z_n}^n & w_{n1}^2 & 0 & 0 & \cdots & 0 & 1 \\
w_{11}^2 x_1^{n+1} & w_{11}^2 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\
\vdots & & & & & & & & & & & \vdots \\
w_{11}^2 x_{z_{n+1}}^{n+1} & w_{11}^2 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 1
\end{bmatrix}$$

We can view this as an alternative way to understand why a neural network function of architecture $(1, n, 1)$ has functional dimension at most $2n + 1$, since the

rank of the column space of $\mathbf{J}E_Z|_\theta$ is at most the rank of

$$rank \begin{bmatrix} 0 & 0 & x_1^1 & 1 & \cdots & x_1^1 & 1 & 1 \\ \vdots & & & & & & & \vdots \\ 0 & 0 & x_{z_1}^1 & 1 & \cdots & x_{z_1}^1 & 1 & 1 \\ x_1^2 & 1 & x_1^2 & 1 & \cdots & x_1^2 & 1 & 1 \\ \vdots & & & & & & & \vdots \\ x_{z_2}^2 & 1 & x_{z_2}^2 & 1 & \cdots & x_{z_2}^2 & 1 & 1 \\ \vdots & & & & & & & \vdots \\ \vdots & & & & & & & \vdots \\ x_1^n & 1 & 0 & 0 & \cdots & x_1^n & 1 & 1 \\ \vdots & & & & & & & \vdots \\ x_{z_n}^n & 1 & 0 & 0 & \cdots & x_{z_n}^n & 1 & 1 \\ x_1^{n+1} & 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ \vdots & & & & & & & \vdots \\ x_{z_{n+1}}^{n+1} & 1 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

which is $2n + 1$, since there are at most $2n + 1$ non-zero columns.

The above example gives an alternative explanation for the upper bound on functional dimension for a neural network with architecture $(1, n, 1)$ by eliminating well-known global symmetries. More hidden symmetries will reduce the functional dimension further, and the degree of this symmetry is inhomogeneous: the functional dimension changes across parameter space due to these hidden symmetries. This can be seen from the upper bound on the expected functional dimension for a neural network with architecture $(1, n, 1)$.

Proposition 2.1.3. *Let F be a bias-free neural network function of architecture $(1, n, 1)$. Let $\theta_0 \in \Theta$ be a parameter. Then the upper bound on expected functional*

dimension of F

$$\mathbb{E}(\dim_{fun}(\theta_0)) \leq n.$$

We refer readers to Appendix 4.1 for a detailed computation on the upper bound on expected functional dimension for an $(1, n, 1)$ architecture and an experimental corroboration. Note that this upper bound is tight if the other direction of lemma 2.1.7 is true.

Each activation region of a neural network function has an associated binary coding, which determines the activation pattern of this region. Recall that for a fixed parameter $\theta \in \Theta$ and a batch of parametrically smooth points Z, each row of its Jacobian matrix $\mathbf{J}E_Z|_\theta$ records the activation status of a generic point x in an activation region. This suggests a connection between the row space of $\mathbf{J}E_Z$ and augmented computational graphs.

In order to deduce some hidden symmetries coming from the activation patterns in a neural network, we will need to consider the linear dependence relations over $\mathbb{Z}$ among the activation patterns for a neural network. Each activation pattern has an associated binary coding illustrating its activation status.

Example 2.1.4.

For illustration, consider the following sequence s of binary labeling for an $(1, 6, 1)$

architecture written from left to right:

$$\begin{aligned} s =& (s_1, \ldots, s_7) \\ =& ((0,1,1,0,1,1), \\ & (1,1,1,0,1,1), \\ & (1,0,1,0,1,1), \\ & (1,0,0,0,1,1), \\ & (1,0,0,1,1,1), \\ & (1,0,0,1,0,1), \\ & (1,0,0,1,0,0)) \end{aligned}$$

Since

$$\begin{aligned} s_1 + (s_7 - (s_5 - s_4)) &= (0,1,1,0,1,1) + (1,0,0,1,0,0) - (1,0,0,1,1,1) + (1,0,0,0,1,1) \\ &= (0+1-1+1, 1, 1, 0+1-1, 1+0-1+1, 1+0-1+1) \\ &= (1,1,1,0,1,1) \\ &= s_2 \end{aligned}$$

We say that there exists a linear dependence relation over $\mathbb{Z}$ among these activation patterns.

We now investigate the role of this $\mathbb{Z}$-linear dependence relation in inducing $\mathbb{R}$-linear dependence relations among the rows of the Jacobian matrix $\mathbf{J}E_Z$ used to compute batch functional dimension.

To do this, we first introduce notation to denote the sum of monomial ending at certain node in layer i, for $i = 1, \ldots, d$.

Given a neural network function F, we have a summand of F associated to each path, which we denote by $F^i_{\textcircled{j}}$, with $\textcircled{j}$ denoting the j-th component of the function F^i

and i denoting the i-th decomposition of the function F. For example, for a neural network of architecture $(1, 3, 3, 1)$, explicitly,

$$\begin{aligned} F = {} & w_{11}^3(w_{11}^2(w_{11}^1 x + b_1^1) + w_{21}^2(w_{12}^1 x + b_2^1) + w_{31}^2(w_{13}^1 x + b_3^1) + b_1^2) \\ & + w_{21}^3(w_{12}^2(w_{11}^1 x + b_1^1) + w_{22}^2(w_{12}^1 x + b_2^1) + w_{32}^2(w_{13}^1 x + b_3^1) + b_2^2) \\ & + w_{31}^3(w_{13}^2(w_{11}^1 x + b_1^1) + w_{23}^2(w_{12}^1 x + b_2^1) + w_{33}^2(w_{13}^1 x + b_3^1) + b_3^2) \\ & + b^3 \end{aligned}$$

Here

$$\begin{aligned} F_{①}^1 &= w_{11}^1 x + b_1^1 \\ F_{②}^1 &= w_{12}^1 x + b_2^1 \\ F_{③}^1 &= w_{13}^1 x + b_3^1 \end{aligned}$$

Then

$$\begin{aligned} F_{①}^2 &= w_{11}^2(F_{①}^1) + w_{21}^2(F_{②}^1) + w_{31}^2(F_{③}^1) + b_1^2 \\ F_{②}^2 &= w_{12}^2(F_{①}^1) + w_{22}^2(F_{②}^1) + w_{32}^2(F_{③}^1) + b_2^2 \\ F_{③}^2 &= w_{13}^2(F_{①}^1) + w_{23}^2(F_{②}^1) + w_{33}^2(F_{③}^1) + b_3^2 \end{aligned}$$

and

$$F = F_{①}^3 = w_{11}^3(F_{①}^2) + w_{21}^3(F_{②}^2) + w_{31}^3(F_{③}^2)$$

Now we are ready to state and prove the following lemma, which can be regarded as a graphical re-interpretation of the well-known backpropagation algorithm.

Lemma 2.1.5. *Let*

$$F = F^d \circ F^{d-1} \circ \cdots \circ F^2 \circ F^1$$

be a neural network function of architecture $(n_0, \ldots, n_d)$*. Denote the ordinary vertices of an augmented graph as* z_i^l*,* $l = 0, \ldots, d$ *if it is the* i*-th vertex in the* l*-th layer. Let* γ_{ij}^l *be the edge connecting* z_i^{l-1} *and* z_j^l*. Then*

$$\frac{\partial F}{\partial w_{ij}^l} = \sum_{\text{path } \gamma \text{ beginning at } z_j^l} \overline{m}(\gamma) \sum_{\text{path } \gamma \text{ ending at } z_j^{l-1}} m(\gamma),$$

where $\overline{m}(\gamma)$ *indicates that the value on the input vertex is not included.*

Proof. Consider the following computational graph with the edge e_{ij}^l corresponding to the weight w_{ij}^l being highlighted in red.

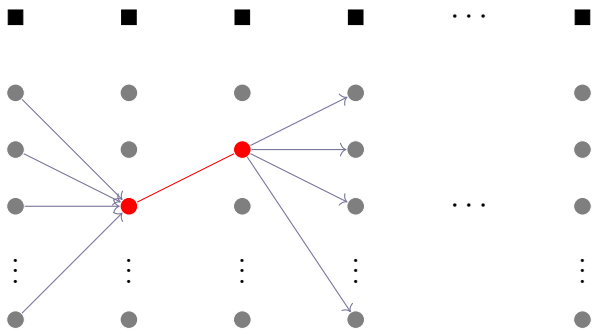

Figure 2.1: Use computational graphs to compute partial derivatives.

As we can see from the computational graph, the edge e_{ij}^l corresponding to the weight w_{ij}^l connects two vertices z_i^{l-1} and z_j^l. We are interested in computing

$$\frac{\partial F}{\partial w_{ij}^l}$$

in terms of the sum of other partial derivatives. Since we can express the neural network function F as the sum of monomial with respect to paths and with their

original values on the input vertices being included, i. e.,

$$F = \sum_{\gamma} m(\gamma),$$

then $\frac{\partial F}{\partial w_{ij}^l}$ depends only on those paths that are starting from or ending at this edge e_{ij}^l.

Now we can use linearity of partial derivatives to break down each partial derivative path by path and apply the chain rule to each path. More specifically, for each such path γ, we have the function F_{ij}^l associated to this path. Then the chain rule tells us:

$$\frac{\partial F}{\partial w_{ij}^l} = (w_{11}^d \dots w_{j1}^{l+1} + \dots + w_{d_{d-1}n_d}^d \dots w_{jn_{l+1}}^{l+1}) F_{\textcircled{i}}^{l-1}$$

But $F_{\textcircled{i}}^{l-1}$ is just the sum of PL functions going into the hidden neuron z_i^{l-1}, therefore

$$\frac{\partial F}{\partial w_{ij}^l} = \sum_{\text{path } \gamma \text{ beginning at } z_j^l} \overline{m}(\gamma) \sum_{\text{path } \gamma \text{ ending at } z_j^{l-1}} m(\gamma).$$

□

Example 2.1.6. (A simple example of broken path interpretation)

Let F be a neural network function of architecture $(1, 3, 3, 3, 1)$. Consider the following augmented computational graph

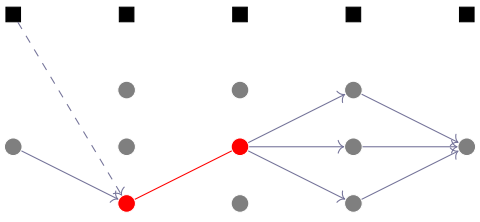

Figure 2.2: An example of broken path interpretation of partial derivatives.

The edge marked in red is e_{32}^2, which corresponds to the weight parameter w_{32}^2. We want to compute the partial derivative of F with respect to w_{32}^2, that can be

explicitly computed can break down into functions along the incoming paths ending at the node z_3^1, and functions along the outgoing paths beginning at the node z_2^2.

$$\begin{aligned}\frac{\partial F}{\partial w_{32}^2} &= w_{11}^4 w_{21}^3 (w_{13}^1 x + b_3^1) + w_{21}^4 w_{22}^3 (w_{13}^1 x + b_3^1) + w_{31}^4 w_{23}^3 (w_{13}^1 x + b_3^1) \\ &= (w_{13}^1 x + b_3^1)(w_{11}^4 w_{21}^3 + w_{21}^4 w_{22}^3 + w_{31}^4 w_{23}^3) \\ &= m(\gamma_{13}^1)(m(\gamma_{211}^{34}) + m(\gamma_{221}^{34}) + m(\gamma_{231}^{34})) \\ &= \sum_{\text{path } \gamma \text{ ending at } z_3^1} m(\gamma) \sum_{\text{path } \gamma \text{ beginning at } z_2^2} \overline{m}(\gamma)\end{aligned}$$

We are now ready to state and prove the following lemma.

Lemma 2.1.7. *Let F be a ReLU neural network function class with architecture $(1, n, 1)$. Choose a generic, transversal parameter $\theta_0 \in \Theta$, let $Z = \{z_1, \ldots, z_m\} \subset \mathbb{R}^{n_0} = \mathbb{R}$ be a set of parametrically smooth points for θ_0, and let $s_{z_i}(\theta_0)$ be the binary sign pattern for z_i at θ_0. If*

$$s_{z_i}(\theta_0) = \sum_{z_j \in Z} s_{z_j}(\theta_0),$$

then

$$\partial\rho(\theta_0)_{z_i} = \sum_{z_j \in Z} \partial\rho(\theta_0)_{z_j},$$

where $\partial\rho(\theta_0)_{z_i}$ is the slope of the realization map $\rho(\theta_0)$ for F at z_i.

Proof. Let Γ_i be the augmented computational graph associated to the activation pattern $s_{\theta_0}(z_i)$. If

$$s_{z_i}(\theta_0) = \sum_{z_j \in Z} s_{z_j}(\theta_0),$$

then the edges being turned on in Γ_i are the sum of activated edges in Γ_j, $j = 1, \ldots, m$, namely,

$$\Gamma_i = \sum_{j \in \{1,\ldots,m\}} \Gamma_j.$$

In particular,

$$m(\gamma_i) = \sum_{j \in \{1,\dots,m\}} m(\gamma_j),$$

the sum of monomials in each Γ_i match. Then

$$\partial\rho(\theta_0)_{z_i} = \sum_{z_j \in Z} \partial\rho(\theta_0)_{z_j}.$$

□

One might wonder whether the opposite direction is true. We conjecture that it is.

Conjecture 2.1.8. *Let F be a ReLU neural network function class with architecture $(1, n, 1)$. Choose a generic, supertransversal parameter $\theta_0 \in \Theta$, let $Z = \{z_1, \dots, z_m\} \subset \mathbb{R}^{n_0} = \mathbb{R}$ be a set of parametrically smooth points for θ_0, and let $s_{z_i}(\theta_0)$ be the sign pattern for z_i at θ_0. If*

$$\partial\rho(\theta_0)_{z_i} = \sum_{z_j \in Z} \partial\rho(\theta_0)_{z_j},$$

then

$$s_{z_i}(\theta_0) = \sum_{z_j \in Z} s_{z_j}(\theta_0),$$

where $\partial\rho(\theta_0)_{z_i}$ is the slope of the realization map $\rho(\theta_0)$ for F at z_i.

For a neural network function of architecture $(1, n, 1)$, Lemma 2.1.7 guarantees that we can transfer linear dependence relations in the data space coming from additive binary triples to the linear dependence relations in the parameter space among the slopes. Namely, if we intend to investigate the linear independence relations of the slopes of realization maps in the parameter space, we believe that it suffices to determine the number of additive triples in the binary labeling associated to an input data point.

Now consider a parametrized neural network function. In particular, when the architecture of a neural network function is $(1, n, 1)$, the graph of the function $\rho(\theta_0)$ at a parametrically smooth point θ_0 has n breakpoints on the axis, and hence n bend points on its graph in the parameter space if the neural network function does not have any hidden neurons being turned off by ReLU activation function. Figure 2.3 shows the picture of a parametrized function $\rho(\theta_0)$ of a neural network function with architecture $(1, 2, 1)$.

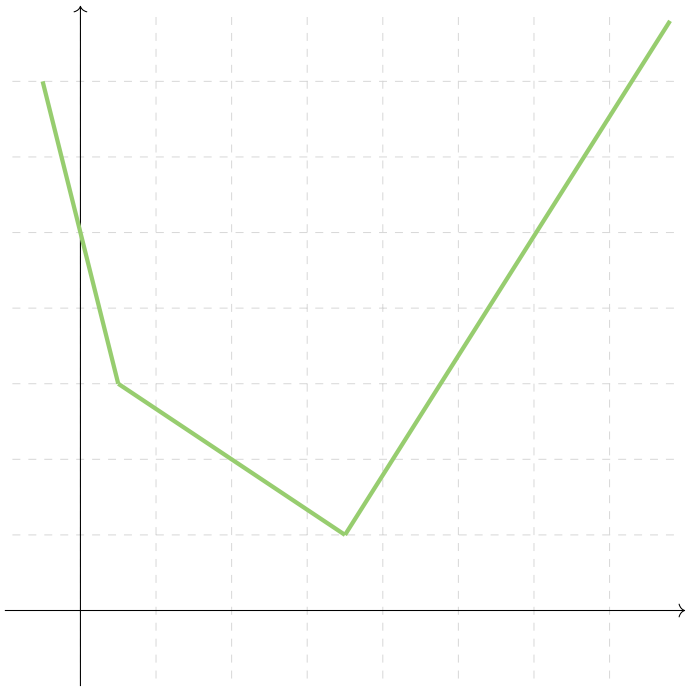

Figure 2.3: The graph of the function $\rho(\theta_0)$ for a parametrically smooth point θ_0.

Let s_x be the binary labeling at an input data point x. Again, we drop x from the subscript since x does not affect the activation pattern. Denote binary labeling of each activation region by

$$s = (s_1, s_2, \ldots, s_{n+1})$$

for an $(1, n, 1)$ architecture. We can then make the following definitions

Definition 2.1.9. *Let s_i, $i = 1, \ldots, n+1$, be the binary labeling of the i-th activation region for an $(1, n, 1)$ architecture. Define index(s_i) to be the ordered tuple positions indicating the 1's in the binary tuple s_i.*

Definition 2.1.10. *Let s_i, $i = 1, \ldots, n+1$, be the binary labeling of the i-th activation region for an $(1, n, 1)$ architecture. Define $|s_i|$ to be the total number of 1's in s_i. We say $|s_i|$ is* ascending *(resp.* descending*) if $|s_i| > |s_{i-1}|$ (resp. $|s_i| < |s_{i-1}|$) for all i.*

For illustration, consider the following example of a binary string

$$s_i = (0, 0, 1, 1, 0, 0, 1).$$

Then

$$\text{index}(s_i) = (3, 4, 7)$$

records the position of 1's in s_i, and

$$|s_i| = 3$$

specifies the total number of 1 in s_i is three.

Now we can investigate the linear dependence relation over $\mathbb{Z}$ among activation patterns for a neural network function with architecture $(1, n, 1)$.

Lemma 2.1.11. *Let F be a neural network function of architecture $(1, n, 1)$. For $i = 1, \ldots, n+1$, let s_i be the binary coding for the i-th activation region. If $|s_i|$ is monotonically ascending (or monotonically descending), or descends first and then only ascends, then there is no $\mathbb{Z}$-linear dependence relations among activation patterns.*

Proof. Since s_i's have constraints coming from the fact that they are a sequence of activation patterns for an $(1, n, 1)$ architecture, it follows that in the first descending

part, 1 will be deleted when going from index(s_1) to index(s_2), and similarly in the last ascending part, $n+1$ in index(s_{n+1}) will never appear in other index(s_i)'s. Therefore, these two parts will be irrelevant of determining the existence of a $\mathbb{Z}$-linear dependence relation among the activation patterns. □

List index(s_i) for all $i = 1, \ldots, n+1$. Start from the first non-descending sequence of index(s) and stop at the end of the last non-ascending index(s). Then the following lemmas hold:

Lemma 2.1.12. *Let F be a neural network function of architecture $(1, n, 1)$. If index(s) is monotonically ascending or monotonically descending for all $i = 1, \ldots, n+1$, i. e., if $|s_i|$ is strictly increasing or strictly decreasing, then there does not exist any linear dependence relation over $\mathbb{Z}$ among activation patterns associated to F.*

Proof. If the binary labelings are monotonically decreasing, then what has appeared in the former index(s_i)'s will never appear later, so there does not exist any additive binary triples. Now suppose that index(s_i)'s are monotonically increasing. Moreover, suppose the first non-descending sign is s_i, then from s_i to s_{i+1}, we are changing the sign at the i-th position. Using index(s_i), this means we are adding $i+1$ to index(s_i). Similarly to what we have in the monotonically decreasing case, n would only appear in the last index(s_{n+1}), so neither can index(s_{n+1}) be written as a linear combination of other binary labelings, nor can it be a base for other binary labelings. This implies that it would be impossible to find any $\mathbb{Z}$-linear dependence relation among activation patterns in this case.

□

Lemma 2.1.13. *Let F be a neural network function of architecture $(1, n, 1)$. If index(s) is not monotonic, then the element $s_i \in s$ satisfying $|s_i| = \arg\max\{|s|\}$ can be expressed as a linear combination of exactly one pair $s_j, s_k \in s$, where $i \neq j \neq k$. Namely, there exists a $\mathbb{Z}$-linear dependence relation among s_j, s_k and s_i.*

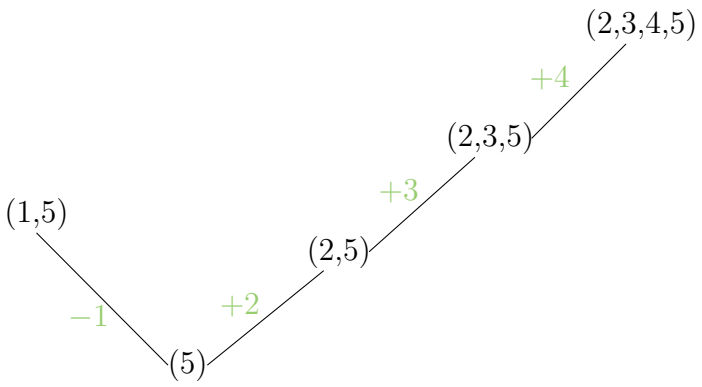

Figure 2.4: A monotonically increasing case for $n = 5$, note that we ignore the first descending part from $(1, 5)$ to (5), and 4 only appears in index$(s_5) = (2, 3, 4, 5)$.

Proof. If index(s) is not monotonic, then index(s) can either increase and decrease thereafter, or increase and decrease for multiple times.

For the first case, assume that we start from index(s_i) and index(s) ascends for j steps, i. e., we reach the maximum at index(s_{i+j}). Then index(s) descends for k steps when it arrives at index(s_{i+j+k}). Denote index(s_{i+j}) as A, and similarly index(s_i) as B, index(s_{i+j+k}) as C. So B has j less 1's compared to A, and C has k less 1's than A. Along the path moving from B to A, we are changing the ϕ-th 0 to 1 at the ϕ-th step. Along the path descending from A to C, we are changing the ψ-th 1 to 0 at the ψ-th step. This implies that the positions of 1's in B and C will match up with the positions of 1's in A. Therefore, A can be represented as a linear combination of B and C, namely, $A = B + C$. In this case, this is the only way to form a $\mathbb{Z}$-linear dependence relation among the activation patterns. Since on the left of the maximum, 1 is added to the maximum one at a time to the corresponding position, and similarly, on the right of the maximum, 1 is subtracted from it one at a time from the corresponding position. Any binary coding in between the maximum and minima will have more 1 needed to form a $\mathbb{Z}$-linear dependence relation.

Now consider the case where index(s) ascends and descends for multiple times, but there is only one maximum. Namely, there exist several local maxima, but there

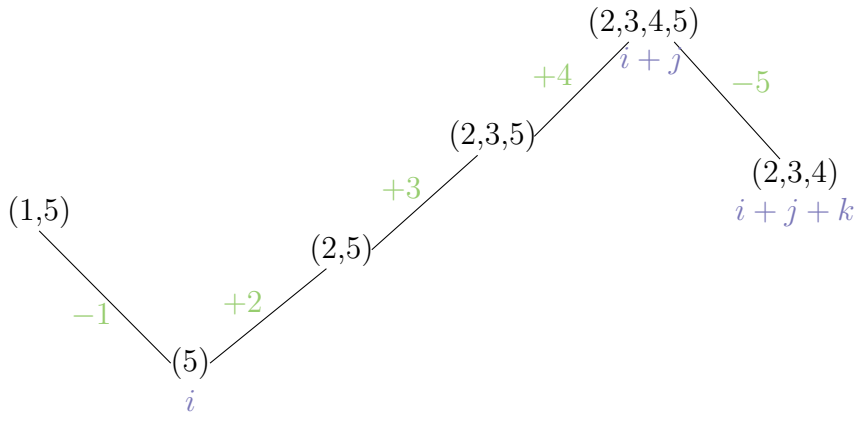

Figure 2.5: Case when there is only one maximum. Here $i = 1$, $j = 3$ and $k = 1$.

is only one global maximum. Assume that starting from index(s_i), index(s) descends for i steps first, then ascends and descends for a_1, d_1, a_2, $d_2, \ldots, a_t$, $d_t$times. Without loss of generality, let s_{i+a_1} be the unique global maximum and denote it as M_G. Denote other local maxima as M_i for $i = 1, \ldots, t-1$, and all the local minima as B_j for $j = 1, \ldots, t+1$.

We can investigate on the positions of 1's in M_G. For those binary labelings appeared before M_G, 1's were appended at positions $i, \ldots, i+a_1$ to form M_G. Therefore M_G must have 1's at its i-th to $(i+a_1)$-th positions. Moreover, M_G must also have 1's at its $(i+a_1+1)$-th to $(i+a_1+d_1)$-th, $(i+a_1+a_2+1)$-th to $(i+a_1+d_1+a_2+d_2)$-th, ..., $(i+\sum_i^t a_i+1)$-th to $(i+\sum_i^t a_i+d_i)$-th positions so that these 1's can be subtracted from M_G to enable the descending pattern for s_x appeared later than M_G. Note that we need to avoid including all the 1's appended in a_i for $i > 1$. This can be achieved by using all the local maximums M_i. Each M_i has two bases binary labeling (recede for a_{i+1} steps to get B_{i+1} and proceed for d_{i+1} steps to get B_{i+2}). Then locally applying how a maximum is formed by its bases, $M_i + B_{i+2} + B_{i+1}$ (here $1+1=0$) will be the same as B_{i+1} minus those 1's diminished from the descending path joining M_i and B_{i+2}. This suggests that we can move the descending path $M_i - B_{i+2}$ after B_{i+1}, i. e., we can paste all the descending paths appeared after M_G together, so that

M_G is the now the only maximum. This modification still will record how 1's in M_G are appended and deleted. In summary, in this case, the sign whose index is M_G can be written as a linear combination of all the B_i's and all the M_i's.

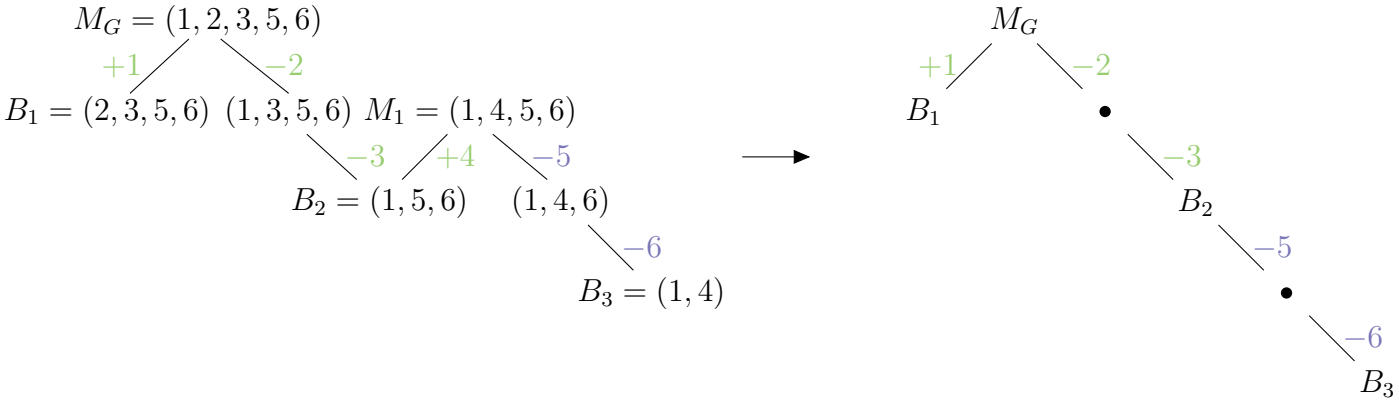

Figure 2.6: Modifying d_{i+1} so that there is only one maximum. The detailed illustration of this specific example is shown in Example 2.1.4.

Finally, if index(s) ascends and descends for multiple times with different global maxima, namely, now there does not exist any single maximum dominating other maximums, then this would be similar to the previous single-global-maximum case, just that we need to pay attention to the total number of steps, which must be enough to guarantee that the maximum has two bases. Thus, if there does not exist sufficiently many steps, then there is no $\mathbb{Z}$-linear dependence relations among activation patterns. If there are enough steps, then we can treat this case as the previous case where there are several maxima. Combining these maxima together by the method shown in figure 2.6 implies that there exists one linear dependence relation over $\mathbb{Z}$ among activation patterns.

In these two cases where there are more than one maximum, a $\mathbb{Z}$-linear dependence relation only exists among the activation patterns M's and B's. Since the ascending paths represent the action of adding 1 to the corresponding position, and the descending paths represent subtracting 1 from the corresponding position, any

middle steps other than M's and B's do not have other binary codings being able to constitute 1 at its corresponding position. □

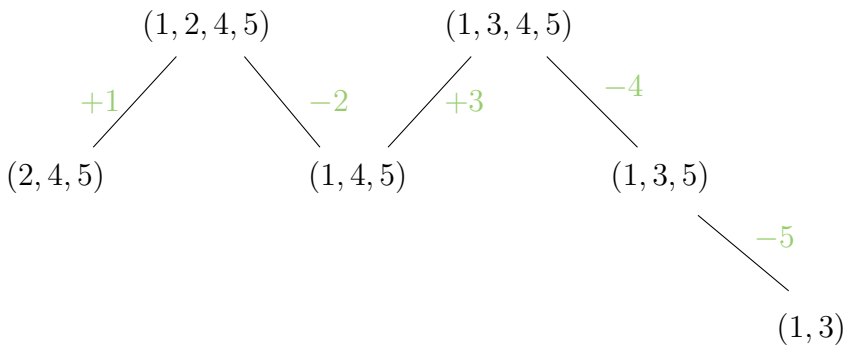

Figure 2.7: A possible situation when there are several global maxima.

Theorem 2.1.14. *Let F be a neural network function of architecture $(1, n, 1)$. Then there exists at most one linear dependence relation over $\mathbb{Z}$ among activation patterns of this network F.*

Proof. Lemma 2.1.12 to lemma 2.1.13 cover all the possibilities for a neural network function F of architecture $(1, n, 1)$ to admit a $\mathbb{Z}$-linear dependence relation among activation patterns of F. Combining the above lemmas prove this theorem.

□

As we have proved above, there exists at most one additive binary triple for a neural network with architecture $(1, n, 1)$. Depending on the binary coding of the first activation region, sometimes there does not exist any additive binary triples among activation regions at all. We are then interested in computing the probability that a neural network function of $(1, n, 1)$ admits a additive binary triple.

Lemma 2.1.15. *Let F be a neural network function of architecture $(1, n, 1)$, $n > 1$. Then with probability $1 - 4^{-n}(n + 1)$, F admits a linear dependence relation over $\mathbb{Z}$ among its activation patterns.*

Proof. Recall that in order for F to have a $\mathbb{Z}$-linear dependence relation among its activation patterns, we shall exclude the following two cases:

Case 1 s is monotonically increasing or monotonically decreasing, which implies that s_1 is all 1's or all 0's, i. e.,

$$s_1 = (0, \dots, 0) \qquad \text{or} \qquad s_1 = (1, \dots, 1).$$

Case 2 s monotonically increases after decreasing first, i. e.,

$$s_1 = (1, 0, \dots, 0) \text{ or } (1, 1, 0, \dots, 0) \text{ or } (1, 1, 1, 0, \dots, 0) \cdots \text{ or } (1, \dots, 1, 0).$$

We can compute that the probability that case 1 can happen is

$$P(\text{Case 1}) = 2 \times \left(\frac{1}{2}\right)^n,$$

and the probability that case 2 happens is

$$P(\text{Case 2}) = (n-1)\left(\frac{1}{2}\right)^n.$$

In total, there are 2^n possible initial binary coding. We can then calculate the probability of F admitting a $\mathbb{Z}$-linear dependence relation is

$$\begin{aligned} 1 - \frac{P(\text{Case 1}) + P(\text{Case 2})}{2^n} &= 1 - \frac{2\left(\frac{1}{2}\right)^n + (n-1)\left(\frac{1}{2}\right)^n}{2^n} \\ &= 1 - 4^{-n}(n+1). \end{aligned}$$

□

In particular, notice that as $n \to \infty$, the above probability approaches 1, meaning, it is more likely for a wider neural network function to admit a linear dependence

relation over $\mathbb{Z}$ among the activation patterns associated to it.

Theorem 2.1.16. *Let F be a neural network function of architecture $(1, n, 1)$. When only considering hidden symmetries coming from collection of activation patterns, the upper bound on its functional dimension is $2n$ with probability $1 - 4^{-n}(n + 1)$.*

Proof. Let F be given. Proposition 2.1.14 summarizes a complete list of combinatorial criteria on the activation patterns ensuring a parameters admits, or does not admit any $\mathbb{Z}$-linear dependence relation. The existence of a $\mathbb{Z}$-linear dependence relation reduced the upper bound on the functional dimension by 1, with probability given by lemma 2.1.15. □

2.2 Z-linear dependence relations and deeper univariate neural networks

The argument of detecting all the $\mathbb{Z}$-linear dependence relations among activation patterns of a neural network cannot be smoothly generalized to a neural network function with a higher input dimension. The main problem is that for a neural network function of architecture $(1, n, 1)$, the binary labelings s_x have an one to one correspondence with the activation regions, whereas for a neural network function of an architecture with a higher input dimension $m > 1$, every binary labeling s_x reflects the activation status provided by m distinct inputs.

Instead, we would also be interested in counting the number of $\mathbb{Z}$-linear dependence relations of a deeper neural network function with input dimension 1. This will provide another perspective of considering the upper bound on functional dimension of a neural network with more layers. In this section, we will only give an example showing the existence of $\mathbb{Z}$-linear dependence relations in a deeper univariate network.

Example 2.2.1. An example of $\mathbb{Z}$-linear dependence relation for architecture $(1, 3, 3, 1)$ with the associated initial binary coding to be $s_x = (s_x^1, s_x^2, s_x^3) = ((0, 1, 0), (1, 0, 1), (1))$.

We first write down all the binary coding from each layer in the χ_l direction. In the direction of χ_1 we have

$$(0, 1, 0), (1, 1, 0), (1, 0, 0), (1, 0, 1),$$

and in the χ_2 direction, we have

$$(1, 0, 1), (0, 0, 1), (0, 1, 1), (0, 1, 0).$$

We can see $\chi_1 = (1, 1, 0)$ with two bases $\chi_1^- = (0, 1, 0)$ and $\chi_1^+ = (1, 0, 0)$, $\chi_2 = (0, 1, 1)$ has bases $\chi_2^- = (0, 0, 1)$ and $\chi_2^+ = (0, 1, 0)$. Then (χ_1, χ_2) is dependent on χ_1^-, χ_1^+, χ_2^- and χ_2^+. We can draw their computational graphs and explicitly write down the open paths:

The open paths for G are:

$$w_{21}^3 w_{12}^2 w_{11}^1 x;$$
$$w_{31}^3 w_{13}^2 w_{11}^1 x;$$
$$w_{21}^3 w_{22}^2 w_{12}^1 x;$$
$$w_{31}^3 w_{23}^2 w_{12}^1 x.$$

The open path for G_{--} is

$$w_{31}^3 w_{23}^2 w_{12}^1 x.$$

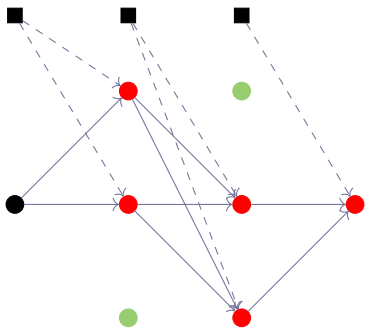

Figure 2.8: The augmented computational graph G for $(\chi_1, \chi_2) = ((1,1,0),(0,1,1),(1))$

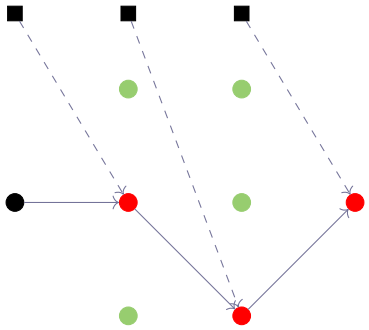

Figure 2.9: The augmented computational graph G_{--} for $(\chi_1^-, \chi_2^-) = ((0,1,0),(0,0,1),(1))$

The open path for G_{-+} is

$$w_{21}^3 w_{22}^2 w_{12}^1 x.$$

The open path for G_{+-} is

$$w_{31}^3 w_{13}^2 w_{11}^1 x.$$

Finally, the open path for G_{++} is

$$w_{21}^3 w_{12}^2 w_{11}^1 x.$$

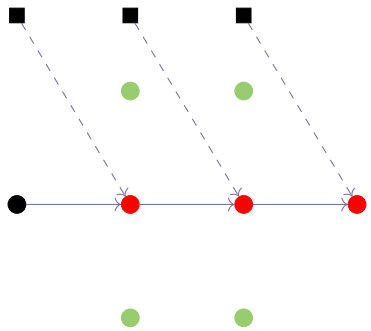

Figure 2.10: The augmented computational graph G_{-+} for $(\chi_1^-, \chi_2^+) = ((0,1,0),(0,1,0),(1))$

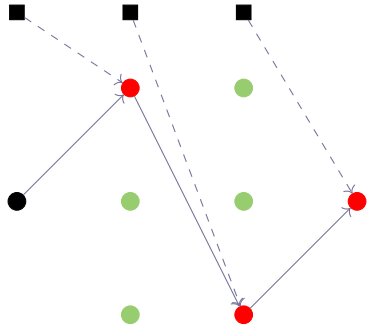

Figure 2.11: The augmented computational graph G_{+-} for $(\chi_1^+, \chi_2^-) = ((1,0,0),(0,0,1),(1))$

From the above figures, we can see that the open paths in G is the sum of the open paths in G_{--}, G_{-+}, G_{+-} and G_{++}, off by a constant given by the bias parameters. In particular, this implies that the partial derivative with respect to weight parameters labeled on the paths of G can be written as a linear combination of the corresponding partial derivatives from all four other graphs.

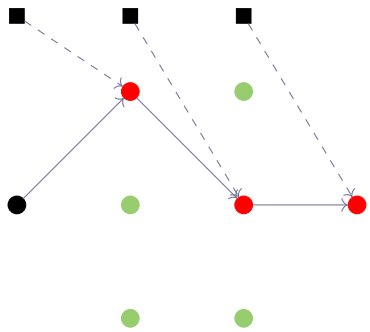

Figure 2.12: The augmented computational graph G_{++} for $(\chi_1^+, \chi_2^+) = ((1,0,0),(0,1,0),(1))$

Chapter 3

Effective functional dimension

3.1 Background

Initialization has a great influence on the speed of the optimization achieved by a neural network. Historically, neural networks are mostly initialized by random weights drawn from either normal distribution or uniform distribution, with normal distributions used more often for deep networks [Krizhevsky et al., 2012]. With uniform standard deviations, neural network models may encounter difficulties converging (e.g., reported in [Karen Simonyan, 2014]). Glorot and Bengio [Glorot and Bengio, 2010] and He [He et al., 2015] proposed different initialization methods to solve this problem. In this section, we have less requirements on initialization for analyzing distribution of breakpoints for a neural network with architecture $(1, n, 1)$. Therefore, we will simplify calculations by using standard normal distribution for weight and bias parameters. That is,

$$w_i \sim N(0,1) \text{ and } b_i \sim N(0,1).$$

Now for a neural network of architecture $(1, n, 1)$ with expression

$$\sigma(\sum_i w_i^{(2)} y_i + b_i^{(2)}),$$

where (j) in the superscript denotes the number of the layer, σ is the ReLU activation function and $y_i = \sigma(w_i^{(1)} x + b_i^{(1)})$, the boundaries of activation regions are given by $-b_i^{(1)}/w_i^{(1)}$. We denote these boundaries as breakpoints of the neural network function. In other words, the *breakpoints* of a neural network function of architecture $(1, n, 1)$ are the boundaries of its activation regions with expression $-b_i^{(1)}/w_i^{(1)}$.

Recall that the Cauchy distribution is the distribution of the ratio of two independent normally distributed random variables, and it has probability density function

$$\begin{aligned} f(x; x_0, \gamma) &= \frac{1}{\pi\gamma \left[1 + \left(\frac{x - x_0}{\gamma}\right)^2\right]} \\ &= \frac{1}{\pi} \left[\frac{\gamma}{(x - x_0)^2 + \gamma^2}\right], \end{aligned}$$

where x_0 is the location parameter and γ is the scale parameter.

For the standard Cauchy distribution, we choose $\gamma = 1$ and $x_0 = 0$. Since the weight and bias parameters were chosen randomly from independent normal distributions, it follows that the breakpoints follow the standard Cauchy distribution.

It turns out that because the breakpoints are drawn from a Cauchy distribution, there is a huge variation in the sizes of the bounded activation regions. To understand this, we first introduce the important notion of the *order statistic* of a random variable.

Definition 3.1.1. *Let X be a random variable with probability density function $f\colon \mathbb{R} \to [0, \infty)$. The* k-th order statistic *on a sample of size n drawn from f is the random variable representing the k-th smallest value among n i.i.d draws of the random variable X. We denote it by $X_{(k:\, n)}$, or just $X_{(k)}$ for convenience.*

Following a common convention, we use upper-case letters to refer to random variables, and lower-case letters (as in the following example) to refer to their actual observed values. For instance, suppose that we take 5 i.i.d random variables X_1, X_2, X_3, X_4, X_5 with the following sample values:

$$x_1 = 6, x_2 = 18, x_3 = 27, x_4 = 9, x_5 = 5.$$

Then the k-th smallest value from the set is considered to be an independent draw of $X_{(k:\,n)}$, which is the k-th order statistic on samples of size 5. Explicitly we have

$$x_{(1)} = 5, x_{(2)} = 6, x_{(3)} = 9, x_{(4)} = 18, x_{(5)} = 27,$$

where the subscript (k) enclosed in parentheses indicates the k-th order statistic of this sample. The probability distribution of these values gives the distribution of the k-th order statistics. We are now ready to state the following.

3.2 ε-effective activation regions

We first motivate the definition of ε-effective activation regions by experimentally and theoretically showing the behavior of activation regions.

Let X be a random variable, following a distribution D, with probability density function $f_X(x)$ such that $f_X(x)$ has a global maximum. In particular, $\arg\max f_X(x)$ is finite. Now sample $\{X_i\}_{i=1}^n$ from D. Define $Y_{(i)}$ to be the i-th order statistics of $\{X_i\}_{i=1}^n$ and let

$$Z_i \coloneqq Y_{(i+1)} - Y_{(i)}.$$

Denote the i-th order statistics of Z_i as $Z_{(i)}$. Then $Y_{(i)}$ has a probability density function transformed from X_i, Z_i has a probability density function transformed

from $Y_{(i)}$, and $Z_{(i)}$ has a probability density function transformed from Z_i. Moreover, the expected value $\mathbb{E}(Z_{(i)})$ would be a number depending on i.

With this setup, we can empirically verify that $\log \mathbb{E}(Z_{(i)})$ and $\log i$ are linearly related for significant portion of the indices i, which equivalently says that $\mathbb{E}(Z_{(i)})$ and i satisfy a power law. Hence, we can make the following conjecture

Conjecture 3.2.1. *Let $Z_{(i)}$ be defined as above. Denote $\mathbb{E}(Z_{(i)})$ as its expected value. Given an architecture $(1, n, 1)$, for $i \leq n/2$,*

$$\mathbb{E}(Z_{(i)}) = \alpha \cdot i^{\beta},$$

for some α, $\beta \in \mathbb{R}$.

We can see from figure 3.1 that the performance of $\log \mathbb{E}(Z_{(i)})$ and $\log i$ behaves predictably as a power law before a turning point, but then gets bottlenecked afterwards. In particular, we can apply linear regression to $\log \mathbb{E}(Z_{(i)})$ and $\log i$ for the points that follow the power law and appear before the turning point, and then verify that $\alpha = -1.905$, $\beta = 7.931$. Therefore,

$$\mathbb{E}(Z_{(i)}) = -1.9 \cdot i^{7.93}. \tag{$*$}$$

We investigate deeper on this turning point appearing in figure 3.1. For an architecture $(1, n, 1)$, we have (figure 3.2, latex doesn't insert the figure correctly)

where the x-axis is n for an $(1, n, 1)$ architecture, and the y-axis is the turning point, i. e., it indicates the place where the power law is violated. The decision boundary for deciding whether the power law stops working or not depends on if the log distance $|\log x - \log y| > \delta$, where x is the point, and y is the fitted line in figure 3.1, δ is a certain threshold, which is chosen to be 0.5 here. Apply linear regression

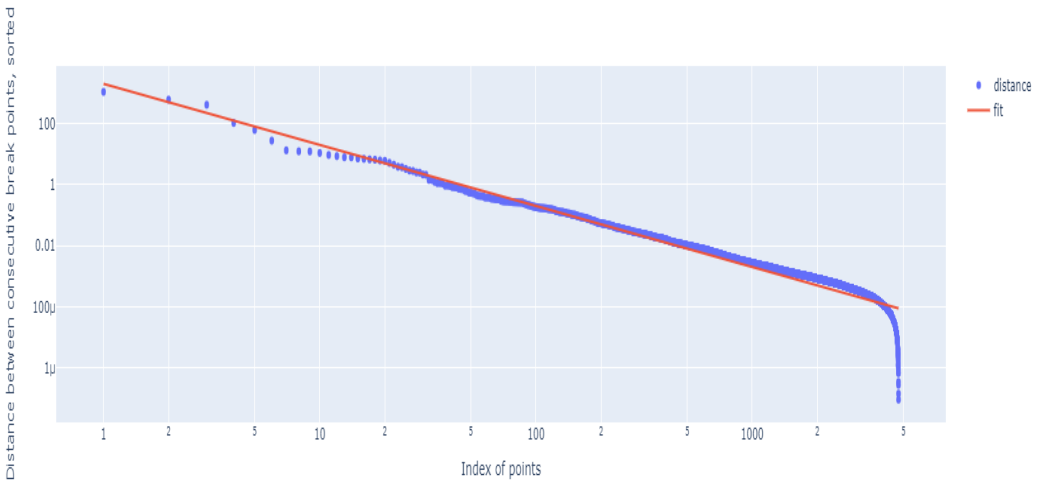

Figure 3.1: $\log \mathbb{E}(Z_{(i)})$ and $\log i$ are linearly related. This figure demonstrates experiments on an architecture $(1, 10000, 1)$, with parameters randomly drawn from $N(0, 1)$. Here on the label of y-axis, $\mu = 10^{-6}$.

to figure 3.2, we get the regression coefficient to be 0.4762, i. e.,

$$y = 0.48n$$

for an $(1, n, 1)$ architecture. In other words, for approximately $i > n/2$, the above power law is violated.

Combined with Figure 3.1, we can see that when n is large enough, there are around half of the breakpoints before and after the turning point. More precisely, approximately half of the activation regions follow the power law $(*)$ above and half the points do not.

The above experiments show that as we have more hidden neurons, the smallest log distance between two consecutive breakpoints is tiny for roughly half of the breakpoints. In other words, half of the activation regions of a neural network function with architecture $(1, n, 1)$ are much smaller than the rest, suggesting a low probability of data points landing in these activation regions. This implies that in practice, due to sampling considerations, achieving the theoretical upper bound on the functional

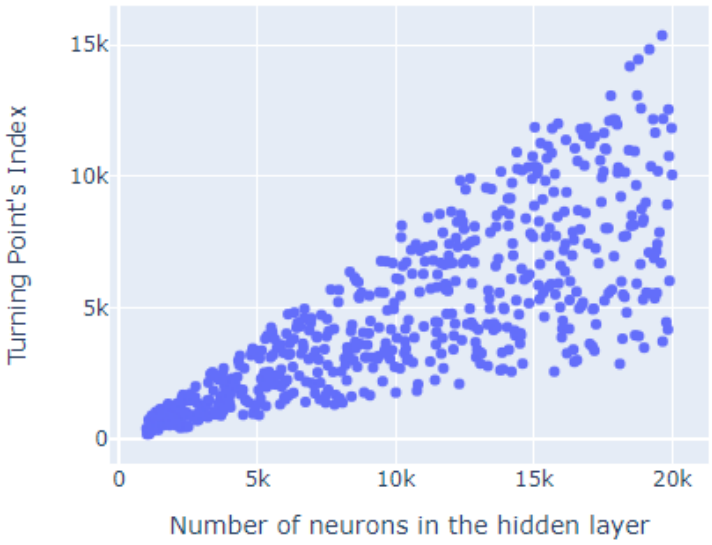

Figure 3.2: Scatter plot of the index where the power law fails when the log distance between the point and the fitted line is greater than 0.5. This figure shows results for architecture $(1, n, 1)$, where $n = 1000 + 20i$ for $i \in [0, 950]$. For each n, $3n + 1$ parameters are randomly drawn from $N(0, 1)$.

dimension is practically impossible.

Note that weights and biases parameters can be sampled from any distribution. There is nothing special about assuming them being independent normally distributed. As long as the quotient distribution of weights and biases has a global maximum, the above statement holds.

In conclusion, the breakpoints of a neural network function of architecture $(1, n, 1)$ whose weights are sampled from $N(0, 1)$ follow a standard Cauchy distribution, resulting in tiny activation regions for roughly half of the points. Moreover, as the neural network function gets wider, i. e., as n gets larger, there will be more and more tiny activation regions clustered together in the center of the bounded regions of the graph of the realization map $\overline{\rho}(\theta)$ (figure 2.3 shows an example of the graph of $\overline{\rho}(\theta)$ for architecture $(1, 2, 1)$). This implies that in practice, when we train a neural network function, many of its activation regions will contain no data points in them, which

will have implications for the functional dimension that can actually be realized on a batch of sample points.

This motivates the following definitions:

Definition 3.2.2. *Let F be a fixed neural network function. Fix $\varepsilon > 0$. Denote $\{x_j\}$ as the set of breakpoints. Let $Y_{(i)}$ be the i-th order statistics of $\{x_j\}$. An activation region R bounded by two consecutive breakpoints $y_{(j-1)}$ and $y_{(j)}$ is said to be* ε-effective for F *if*

$$P(y_{(j-1)} < Z < y_{(j)}) \geq \varepsilon.$$

And an activation region R is said to be ε-ineffective to a neural network F *if*

$$P(y_{(j-1)} < Z < y_{(j)}) < \varepsilon.$$

With this definition in hand, we can also run experiments to estimate the total measure (out of 1) of the union of ε-effective regions as a function of n and ε for a neural network function of architecture $(1, n, 1)$. Our experiments are demonstrated in Figure 3.3.

The effectiveness of an activation region implies that, in practice, usually it would be hard to achieve the theoretical upper bound $2n + 1$ on functional dimension for a neural network function of architecture $(1, n, 1)$. As we have shown by various experiments on "activated" activation regions, if the input data points are sampled from a standard normal distribution, then there are only around 40% of the activation regions are ε-effective.

Insufficient ε-effective activation regions would result in a smaller size of sample data points when computing the functional dimension, and hence, the ε-effective functional dimension will be lower than the full functional dimension. To see this, for m points drawn from $N(0, 1)$, we can run experiments to test the ε-effective sample size, that is, the expected number of points landing in ε-heavy regions. For

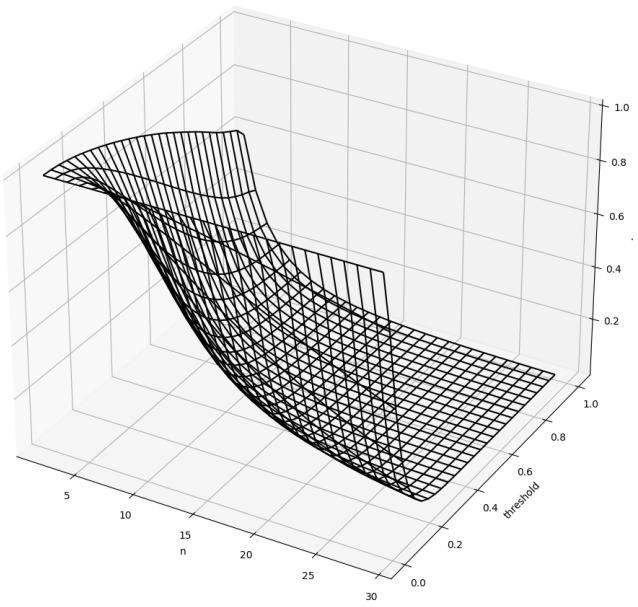

Figure 3.3: Total measure of union of ε-effective regions with threshold axis denoting ε. We can see that when n is large and ε is small, the total measure of ε-effective activation regions is around 40%.

illustration, figure 3.5 shows the ε-effective sample size for $\varepsilon = 0.05$ is at most 82 when $m = 100$ and $n = 1, \ldots, 100$ for a neural network function of architecture $(1, n, 1)$.

The difference in the activation regions and effective activation regions leads to a discussion of effective functional dimension. We should view the ε-effective functional dimension as the functional dimension achieved empirically. We first formally define ε-effective functional dimension.

3.3 ε-effective functional dimension

Consider the set of breakpoints $\{x_j\}_{j=1}^{k}$ following a Cauchy distribution. Note that k is the number of breakpoints and it is fixed for a specific architecture. Let $Y_{(i)}$ be the i-th order statistics of $\{x_j\}$. Locally in the parameter space Θ, the weight and bias parameters are fixed, whence the activation regions are fixed. In particular, this

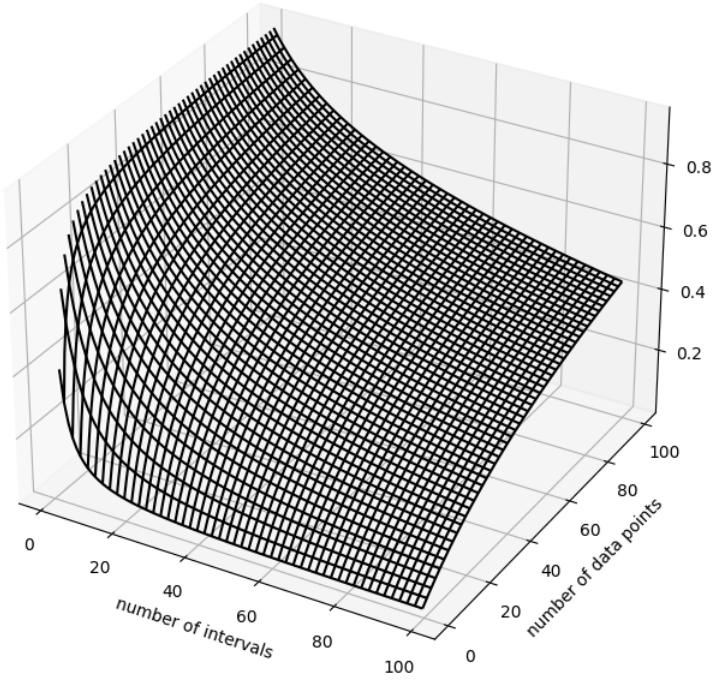

Figure 3.4: For a neural network of architecture $(1, n, 1)$, when n is large and when there are enough amount of input data points, we can see that only about 40% of the activation regions are being ε-effective.

means that $y_{(i)}$'s are fixed. Now let $\varepsilon > 0$, and let z be a random input data point. Define

$$I(i) \coloneqq \begin{cases} 1 & \text{if } P(y_{(i-1)} < z < y_{(i)}) > \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

Let $Z_{\overline{I}}$ be a subset of Z, which excludes points according to this function I. i. e., $Z_{\overline{I}}$ is a smaller batch of sample points, it considers only those input points with weights at least ε.

Definition 3.3.1. *Let the marked realization map $\overline{\rho}(\theta)$ be defined as before. Let $\theta \in \Theta_{n_0,\ldots,n_m}$ be an ordinary parameter. The* batch ε-effective functional dimension *of θ for a batch $Z \subset \mathbb{R}^{n_0}$ of parametrically smooth points for θ is*

$$\dim^{\varepsilon}_{e.ba.fun}(\theta, Z) \coloneqq rank \boldsymbol{J} E_{Z_{\overline{I}}}|_{\theta},$$

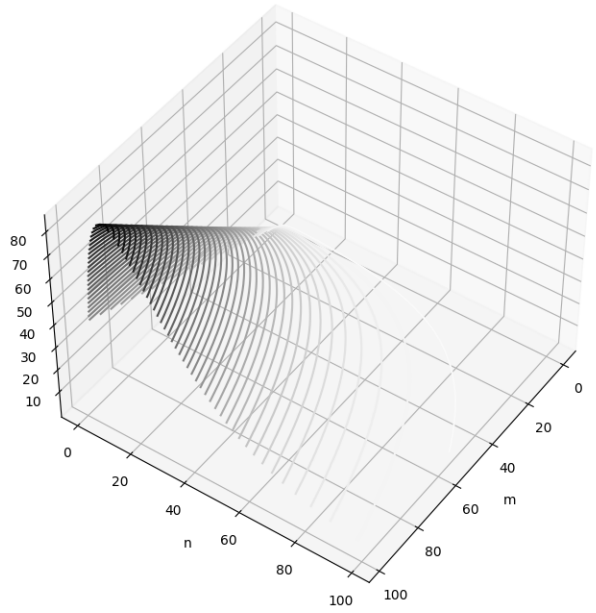

Figure 3.5: The ε-effective sample size for $\varepsilon = 0.05$.

We are now ready to define the (full) ε-effective functional dimension of an ordinary point $\theta \in \Theta_{n_0,\ldots,n_m}$.

Definition 3.3.2. *Fix $\varepsilon > 0$. Consider an architecture $(n_0, \ldots, n_m)$. For any point $\theta \in \Theta_{n_0,\ldots,n_m}$, define the (full) εeffective functional dimension at θ to be*

$$\dim^{\varepsilon}_{e.fun}(\theta) \coloneqq \sup_{Z} \dim^{\varepsilon}_{e.ba.fun}(\theta, Z).$$

Here again, $Z \subset \mathbb{R}^{n_0}$ is finite and parametrically smooth for θ. We can also define the local (in the sense of the parameter space $\Theta_{n_0,\ldots,n_m}$) effective functional dimension.

Definition 3.3.3. *Fix an architecture $(n_0, \ldots, n_m)$. Define the effective functional dimension of the parameter space $\Theta_{n_o,\ldots,n_m}$ to be*

$$\dim^{\varepsilon}_{e.fun} \coloneqq \sup_{\theta} \dim^{\varepsilon}_{e.fun}(\theta).$$

We give an empirical example to illustrate and compare the difference between the (theoretical) functional dimension and the effective functional dimension. Consider

the architecture $(1, 11, 1)$. If $\theta \in \Theta_{1,n,1}$ has coordinates

$$\theta = (w^1_{11}, \ldots, w^1_{1n}, b^1_1, w^1_{21}, \ldots, w^1_{2n}, b^1_2, \ldots, w^2_{1n}, b^2_1),$$

then a similar computation as we did when computing the dimension of the column space of $\mathbf{J}E_z$ shows that for a generic, transversal parameter θ, the functional dimension on decisive sets consisting $k(n_0+1)$ points can achieve the theoretical upper bound $2n+1$, where k is the number of top-dimensional cells of the canonical polyhedral complex $\mathcal{C}(\theta)$. However, in practice, if we sample the weight and bias parameters from a normal distribution and compute the ε-effective functional dimension, with about 40% activation regions being ε-effective, we would only achieve approximately 40% of the true functional dimension.

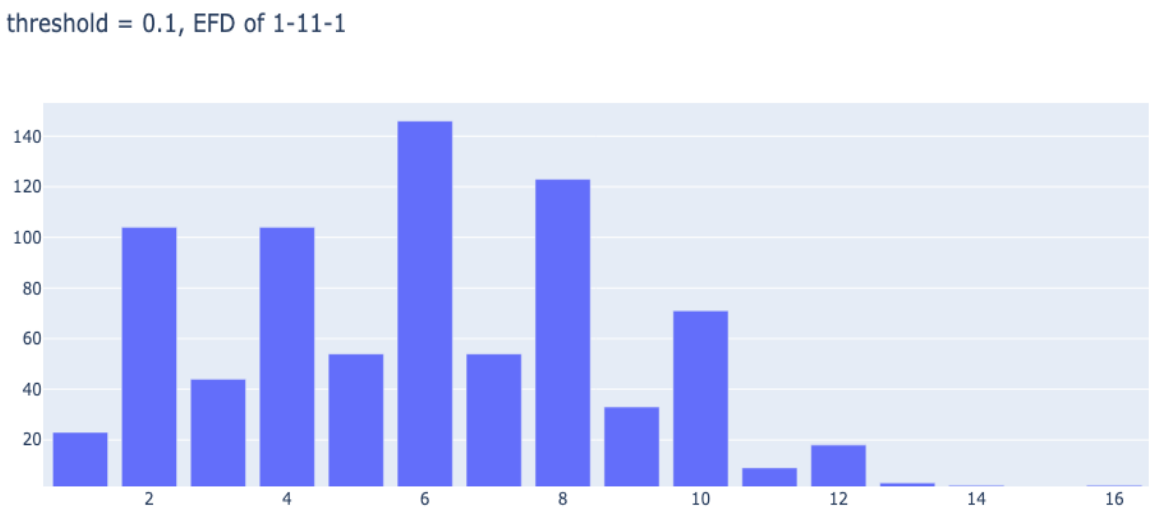

Figure 3.6: ε-effective functional dimension for architecture $(1, 11, 1)$ with $\varepsilon = 0.1$, and no ReLU on the final layer. The x-axis shows the ε-effective functional dimension, and the y-axis is the number of time we get this ε-effective functional dimension. Note that the total number of networks is 2000.

For Figures 3.6 to Figure 3.8, we ran 2000 experiments, where each run has weights and biases drawn from $N(0, 10)$ and 10000 data points randomly generated from

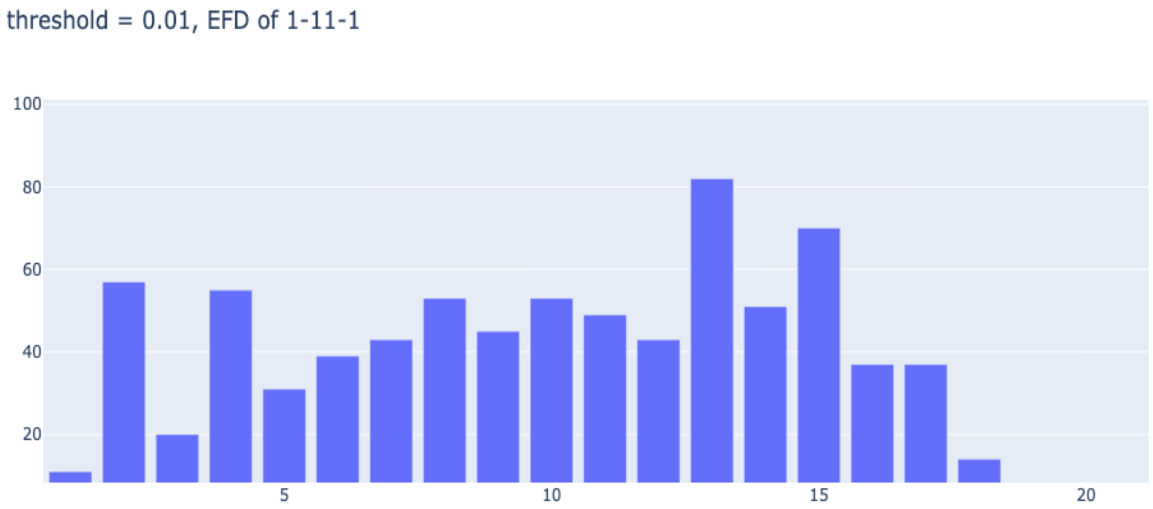

Figure 3.7: ε-effective functional dimension for architecture $(1, 11, 1)$ with $\varepsilon = 0.01$, and no ReLU on the final layer. The x-axis shows the ε-effective functional dimension, and the y-axis is the number of time we get this ε-effective functional dimension. Note that the total number of networks is 2000.

$N(0, 10)$. Here we choose a higher variance for normal distribution on the sample points because otherwise we would need a much larger sample size in order to have enough points in the activation regions so that the ε-effective functional dimension will not always be 0. Additionally, we did not put ReLU activation function on the final layer map, since this will yield many calculations to be 0. These figures show that even though theoretically we should expect $2n+1$ linearly independent degrees of freedom available for perturbing parameters near $\theta = \theta_0$, while remaining in the class of functions realizable by neural network functions of the same architecture, however, in practice, the maximum number of linearly independent degrees of freedom we could actually achieve for varying parameters near θ_0 is much smaller, especially when ε is large. We observe that when $\varepsilon \to 0$, we have more ε-effective activation regions, and the upper bound on effective functional dimension converges to the theoretical upper bound on functional dimension, which is demonstrated by the above figures, with ε

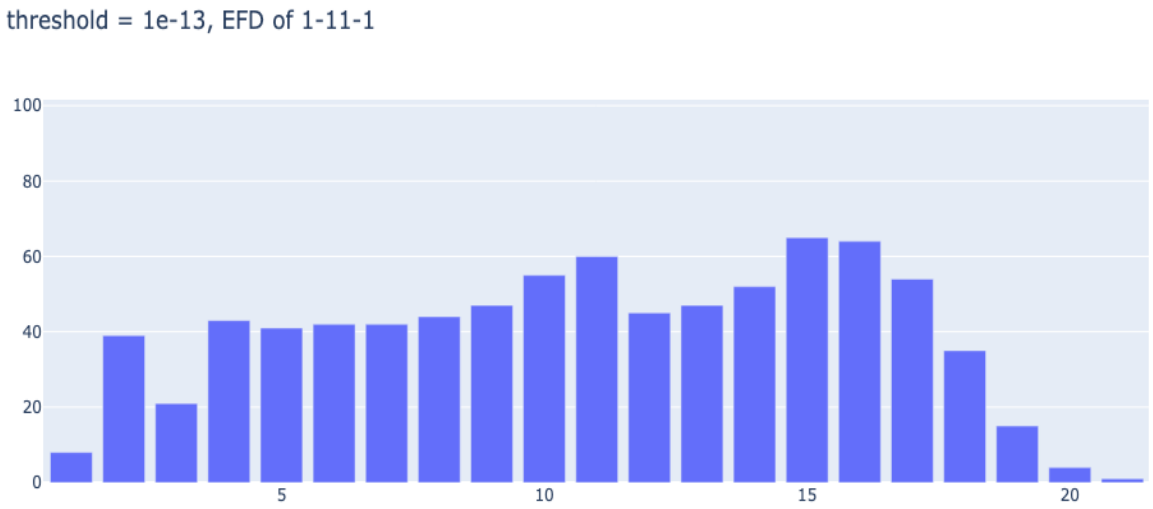

Figure 3.8: ε-effective functional dimension for architecture $(1, 11, 1)$ with $\varepsilon = 10^{-13}$, and no ReLU on the final layer. The x-axis shows the ε-effective functional dimension, and the y-axis is the number of time we get this ε-effective functional dimension. Note that the total number of networks is 2000.

varies from 0.1 to 10^{-13}.

More formally, we can see this from a different perspective: if there are less ε-ineffective regions (i. e., more regions are activated), then the difference between the upper bound on ε-effective functional dimensions and true functional dimension is smaller. We can construct an $\widetilde{\varepsilon}$-net for the set of ε-ineffective regions $\Delta(c)$. That is, we consider only those ε-ineffective regions with weight at least $\widetilde{\varepsilon}$ under normal distribution. Thus, let M be a sample set of size m, and let $\Delta_{\widetilde{\varepsilon}}(c) = \{r \in \Delta(c) \colon P_{x \in N}[x \in r] \geq \widetilde{\varepsilon}\}$. We can make the following definition:

Definition 3.3.4. *For any $\widetilde{\varepsilon} > 0$, we say that a set M is an $\widetilde{\varepsilon}$-net for $\Delta(c)$ if every region in $\Delta_{\widetilde{\varepsilon}}(c)$ contains a point from M, i. e., for every $r \in \Delta_{\widetilde{\varepsilon}(c)}$, we have $M \cap r \neq \emptyset$.*

Now fix an ε-ineffective region $r \in \Delta_{\widetilde{\varepsilon}}(c)$. By construction, we can bound the probability that all the m random sample points miss r, this is at most $(1 - \widetilde{\varepsilon})^m$. In other words, the probability that all but one ε-ineffective regions contain a sam-

ple from M is bounded above by $(1-\widetilde{\varepsilon})^m$. This suggests the difference between the effective batch functional dimension and true functional dimension is bounded. Moreover, this upper bound gets tighter as $\widetilde{\varepsilon}$ gets larger, which means there exists fewer ε-ineffective regions.

Now given a sample batch of size M, denote $E[h_M]$ to be the ε-effective batch functional dimension for M, and h_{FD} to be the batch functional dimension. We would like to investigate more on $|E[h_M]-h_{FD}|$, the difference between the ε-effective batch functional dimension for M and batch functional dimension.

Define η to be the extra rank given by adding rows corresponding to those ineffective sample points (i. e., those data points in activation regions but not ε-effective activation regions) to the Jacobian matrix for computing the ε-effective batch functional dimension. Then by definition, we have

$$E[h_M] \leq h_{FD} \leq E[h_M] + \eta.$$

$|E[h_M] - h_{FD}|$ is thus bounded above by η. Furthermore, we can test $|E[h_M] - h_{FD}|$ for different M and ε.

Figure 3.9 shows that for ε very close to 0 (for instance, $\varepsilon = 10^{-13}$), the difference between ε-effective batch functional dimension and batch functional dimension is constantly zero for any batch size M. When $\varepsilon = 0.01$, $|E[h_M] - h_{FD}| = 0.1$, 0.26, 0.77, 1.51, 1.83, 1.51, 1.24,1, 0.53 for $M = 10$, 46, 215, 1000, 4642, 21544, 100000, 464159, 2154435; respectively for the same M, when $\varepsilon = 0.1$, $|E[h_M] - h_{FD}| = 0.05$, 0.29, 0.77, 1.59, 1.8, 1.26, 1.22, 1, 0.45; for $\varepsilon = 0.26$, $|E[h_M] - h_{FD}| = 0.03$, 0.22, 0.74, 1.45, 1.79, 1.16, 1.17, 1, 0.47.

Let M^* be the batch size such that the difference $|E[h_M] - h_{FD}|$ is maximized. We can see that $M^* = 4642$, which is around 2% of M. For $M > M^*$, $|E[h_M] - h_{FD}|$ decays slowly to 0.

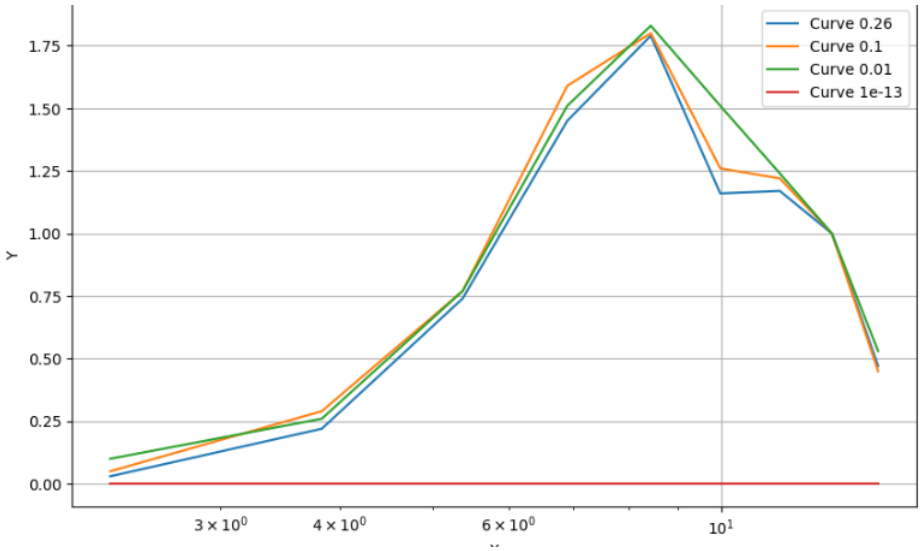

Figure 3.9: Difference between ε-effective batch functional dimension and batch functional dimension for various ε. Each color of the curve shows the behavior of one ε.

Chapter 4

Appendix

4.1 Expected functional dimension

Each activation region has a binary labeling associated to it, therefore, depending on the activation pattern, the functional dimension can vary for a fixed architecture. It is known that the degree of hidden symmetries is inhomogeneous, i. e., many different parameter settings can determine the same neural network function. In particular, the functional dimension changes across the parameter space. This section provides a short discussion on computing the upper bound on expected functional dimension. In particular, we can theoretically compute the upper bound on expected functional dimension for a neural network of architecture $(1, n, 1)$. Note that this upper bound is tight if the opposite direction of lemma 2.1.7 holds.

We start by considering a neural network function of architecture $(1, 2, 1)$.

Lemma 4.1.1. *Let F be a neural network function of architecture $(1, 2, 1)$. Let $\theta_0 \in \Theta$ be a parameter. Then*

$$\mathbb{E}(\dim_{fun}(\theta_0)) \leq \frac{57}{16}.$$

Proof. The activation status of two hidden neurons determine the activation patterns

of three activation regions. In particular, once we know the binary labeling of the first activation region, we can determine the rest. There are four possible sign patterns for the first region: $(0,1),(1,0),(0,0),(1,1)$, each with equal probability $1/4$.

For the first case that the first activation region has binary labeling $(0,1)$, the other two activation regions have binary labeling $(1,1)$ and $(1,0)$. Depends on how the hyperplane from the last layer intersects, with right to be the positive direction, we have the following six cases

$$1.\{(0,1),(1)\},\{(1,1),(1)\},\{(1,0),(1)\};$$

$$2.\{(0,1),(1)\},\{(1,1),(1)\},\{(1,1),(0)\},\{(1,0),(0)\},\{(1,0),(1)\};$$

$$3.\{(0,1),(1)\},\{(0,1),(0)\},\{(1,1),(0)\},\{(1,0),(0)\},\{(1,0),(1)\};$$

$$4.\{(0,1),(1)\},\{(1,1),(0)\},\{(1,0),(0)\},\{(1,0),(1)\};$$

$$5.\{(0,1),(0)\},\{(1,1),(0)\},\{(1,1),(1)\},\{(1,0),(1)\};$$

$$6.\{(0,1),(0)\},\{(0,1),(1)\},\{(1,1),(1)\},\{(1,0),(1)\},$$

and changing the positive direction for the intersecting hyperplane, we will have the other six cases with equal probability. We can compute the functional dimension for each case. Case 1 has functional dimension at most 5, switching the direction for the hyperplane gives functional dimension 0, similarly, case 2 has functional dimension at most 5, 4; case 3 has functional dimension at most 5, 5; case 4 has functional dimension at most 3, 5; case 5 has functional dimension at most 4, 4; and case 6 has functional dimension at most 5, 3. Since bias parameters are chosen from a distribution with smaller variance, it implies that the bounded regions are very small compared to those unbounded ones. Hence we will only consider case 1, 3, 4, 6 for computing the expected functional dimension. Then in this case, the upper bound

on expected functional dimension is

$$\begin{aligned}
&\mathbb{E}(\text{case 1}) + \mathbb{E}(\text{case 3}) + \mathbb{E}(\text{case 4}) + \mathbb{E}(\text{case 6}) \\
\leq &\frac{2.5}{16} + \frac{5}{16} + \frac{4}{16} + \frac{4}{16} \\
= &\frac{15.5}{16}
\end{aligned}$$

Similarly, we can computed the upper bound on expected functional dimensions for the other three cases: $\frac{15.5}{16}$, $\frac{13}{16}$ and $\frac{13}{16}$. Thus, the expected functional dimension for a neural network function of architecture $(1, 2, 1)$ is at most $\frac{57}{16}$. □

We can use the same method to compute the upper bound on expected functional dimension for a general architecture.

Proposition 4.1.2. *Let F be a bias-free neural network function of architecture $(1, n, 1)$. Let $\theta_0 \in \Theta$ be a parameter. Then the upper bound on expected functional dimension of F*

$$\mathbb{E}(\dim_{fun}(\theta_0)) \leq n.$$

Proof. There are three possible cases in the parameter space:

1. All the slopes in the parameter space are preserved (or on the contrary, being zeroed out) by the activation function. The upper bound on expected functional dimension for this case is

$$\mathbb{E} \leq (0 + 2n + 1)\left(\frac{1}{n}\right)\left(\frac{1}{4}\right),$$

where $2n+1$ is the upper bound on functional dimension for an $(1, n, 1)$ network work, $1/n$ is the probability of the intersecting hyperplane can turn in terms of its dimension, and $1/4 = 1/2 \times 1/2$ is the probability that the function achieves the full functional dimension, or it's completely being zeroed out. Then for n

large enough, we have

$$\lim_{n\to\infty} \mathbb{E} \leq \frac{1}{2}.$$

2. Only slopes in the unbounded regions are preserved. Similarly as above, this case has probability

$$\mathbb{E} \leq (5+5)\left(\frac{1}{n}\right)\left(\frac{1}{4}\right),$$

and

$$\lim_{n\to\infty} \mathbb{E} = 0.$$

3. Only one slope in the parameter space is preserved and its symmetric case. This has probability

$$\mathbb{E} \leq (3+2n+1)\left(\frac{1}{n}\right)\left(\frac{1}{4}\right),$$

and

$$\lim_{n\to\infty} \mathbb{E} \leq \frac{1}{2}.$$

Therefore, as each subcase has probability $\left(\frac{1}{2}\right)^n$, we can summarize and compute the expected functional dimension for a bias-free neural network of architecture $(1, n, 1)$:

$$\mathbb{E} \leq \frac{2n\left(\frac{1}{2}\right)\left(\frac{1}{2}\right)^n}{\left(\frac{1}{2}\right)^n} = 2n\left(\frac{1}{2}\right) = n.$$

□

We have also run an experiment to test the expected functional dimension for a neural network function of architecture $(1, n, 1)$, it is 3.74 for $n = 2$, and approaches n when $n \to \infty$, which matches our computations from the upper bound above, leading evidence for Conjecture 2.1.8 (suggesting that the opposite direction for Lemma 2.1.7 is very likely to be true).

Bibliography

[Bona-Pellissier et al., 2022] Bona-Pellissier, J., Malgouyres, F., and Bachoc, F. (2022). Local identifiability of deep relu neural networks: the theory.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. 9:249–256.

[Grigsby and Lindsey, 2021] Grigsby, J. E. and Lindsey, K. (2021). On transversality of bent hyperplane arrangements and the topological expressiveness of relu neural networks.

[Grigsby et al., 2022] Grigsby, J. E., Lindsey, K., Meyerhoff, R., and Wu, C. (2022). Functional dimension of feedforward relu neural networks.

[Grigsby et al., 2023] Grigsby, J. E., Lindsey, K., and Rolnick, D. (2023). Hidden symmetries of relu networks.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

[Kaplan et al., 2020] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models.

[Karen Simonyan, 2014] Karen Simonyan, A. Z. (2014). Very deep convolutional networks for large-scale image recognition.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. 25.

[Phuong and Lampert, 2020] Phuong, M. and Lampert, C. H. (2020). Functional vs. parametric equivalence of re{lu} networks.

[Rolnick and Kording, 2020] Rolnick, D. and Kording, K. P. (2020). Reverse-engineering deep relu networks.

[Stanley, 2007] Stanley, R. P. (2007). An introduction to hyperplane arrangements.