A Simple Geometric Proof for the Benefit of Depth in ReLU Networks

Anonymous authors
Paper under double-blind review

Abstract

We present a simple proof for the benefit of depth in multi-layer feedforward network with rectifed activation ("depth separation"). Specifically we present a sequence of classification problems f_i such that (a) for any fixed depth rectified network we can find an index m such that problems with index $> m$ require exponential network width to fully represent the function f_m; and (b) for any problem f_m in the family, we present a concrete neural network with linear depth and bounded width that fully represents it.

While there are several previous work showing similar results, our proof uses substantially simpler tools and techniques, and should be accessible to undergraduate students in computer science and people with similar backgrounds.

1 Introduction

We present a simple, geometric proof of the benefit of depth in deep neural networks.

We prove that there exist a set of functions indexed by m, each of which can be efficiently represented by a depth m rectified MLP network requiring $O(m)$ parameters. However, for any bounded depth rectified MLP network, there is a function f_m in this set that representing it will require an exponential number of parameters in m.

More formally, let G^d be the set of multi-layer perceptron (MLP) networks with rectified activation and d hidden layers, and let g_Θ be such an MLP with parameters Θ. We will prove the following theorem:

Theorem 1 (Depth Separation)**.** *There exists a set of functions* $f_1, f_2, ..., \quad f_i : \mathbb{R}^2 \mapsto \{-1, 1\}$ *such that:*

a For any d, there exists $m > d$ such that if $g_\Theta \in G^d$ satisfies $g_\Theta(x) = f_m(x)\ \forall x$ then $|\Theta| = \Omega(2^m)$ parameters. (Bounded depth network is exponential in size).

b For any m, there exists a function $g_\Theta \in G^m$ satisfying $g_\Theta(x) = f_m(x)\ \forall x$ and $|\Theta| = O(m)$ parameters. (Utility of depth).

While this is not a novel result, a main characteristic of our proof is its simplicity. In contrast to previous work, our proof uses only basic algebra, geometry and simple combinatorial arguments. As such, it can be easily read and understood by newcomers and practitioners, or taught in an undergraduate class, without requiring extensive background. Tailoring to these crowds, our presentation style is more verbose then is usual in papers of this kind, attempting to spell out all steps explicitly. We also opted to trade generality for proof simplicity, remaining in input space $\mathbb{R}^2$ rather than the more general $\mathbb{R}^n$, thus allowing us to work with lines rather than hyperplanes. Beyond being easy to visualize, it also results in simple proofs of the different lemmas.

2 Related Work

The expressive power gained by depth in multi-layer perceptron (MLP) networks is relatively well studied, with multiple works showing that deep MLPs can represent functions that cannot be represented by similar but shallower networks, unless those have a significantly larger number of units (Delalleau & Bengio, 2011; Pascanu et al., 2013; Bianchini & Scarselli, 2014).

Telgarsky (2015; 2016) show that network depth facilitate fast oscillations in the network response function. Oscillations enabled by a linear growth in depth are shown to require exponential growth in the number of units when approximated well by a shallower network.

Eldan & Shamir (2016) study approximation to the unit sphere in a wide family of activation function. In their construction they show that a 3-layer MLP could first compute the polynomial x^2 for each of the dimensions and use the last layer to threshold the sum of them to model the unit sphere indicator. They analytically show that the same approximation with 2-layer network requires exponentially growth in width with precision.

Yarotsky (2017); Safran & Shamir (2016) show that depth is useful for approximating polynomials by ReLU MLPs. Specifically, that $f(x) = x^2$ could be efficiently approximated with network depth.

While results similar to ours could be derived by a combination of the construction in Eldan & Shamir (2016) and the polynomial approximation of Yarotsky (2017) , we present a different (and to our taste, simpler) proof, using a geometrical interpretation and the number of response regions of ReLU networks, without explicitly modeling the x^2 polynomial.

The ReLU MLP decision space was studied by Pascanu et al. (2013). They show that the input space is sequentially refined by the ReLU and linear operations of the network to form separated convex polytopes in the input space. They call these regions *response regions*. They also establish a lower bound on the maximal number of regions, a bound which is tightened by Montufar et al. (2014); Raghu et al. (2017); Arora et al. (2016); Serra et al. (2017). We rely on the notion of response region in our proof, while attempting to provide an accessible explanation of it. Some of the lemmas we present are simplified versions of results presented in these previous works.

3 Background

3.1 Linearity and piecewise linearity. Convexity.

A *linear function* is a function of the form $f(x) = \mathbf{A}x + \mathbf{b}$. For affine spaces (like the Euclidean space), this is also called an *affine transformation* of the input. In a *piecewise linear* function the input space is split into regions, and each region is associated with a linear function. A composition of linear functions is linear. A composition of piecewise linear functions is piecewise linear.

A $2d$ region is *convex* iff, for any two points in the region, all points on the line connecting the two points is also within the region. A polygon with all internal angles $< 180^o$ is a convex region.

3.2 ReLU MLP with L Layers

A ReLU MLP with L layers parameterized by Θ is a multivariate function defined as the composition:

$$F(X;\Theta) = h^{out} \circ h_L^A \circ \sigma \circ h_{L-1}^A \circ \sigma ... \circ \sigma \circ h_1^A(X)$$

. Where h_i^As are parameterized affine transformations; Θ the set of parameters in them; and σ is the ReLU activation function: a non linear element-wise activation function defined by $\sigma(x) = max\{0, x\}$. We consider ReLU MLPs where all hidden layers have the same width w.[1] Without loss of generality we define the last layer of network, h^{out}, as a weighted sum over its inputs where a sum strictly greater than zero is mapped to the 1 class, and otherwise to the -1 class.

The combination of linear operations and the ReLU function result in a piecewise linear function of the input X.

3.3 ReLU MLP response regions

Piecewise linear activation functions such as ReLU split the input space into convex regions of linear activation. This is asserted formally and visualized in Hanin & Rolnick (2019). The ReLU function has two regions (“pieces”) of linearity $x > 0, x \leq 0$. Within each of these, linearity is maintained. The sequential composition of affine transformations and the ReLU operations created by the MLP

[1]This subsumes networks with layers with width $< w$, as these are equivalent to width w layers with zeroes in specific regions of the parameters.

layers, divides the the input space into convex polytopes (in $2d$ these are convex polygons). Within each such polytope, the function behaves linearly. We call these polytopes *linear response regions*.

The number of these linear response regions, and specifically the effect of MLP depth on the maximal number of regions, was studied in multiple works Montufar et al. (2014); Raghu et al. (2017); Arora et al. (2016); Serra et al. (2017). We focus on the simpler case of 2-class classification ReLU MLP on the Euclidean plane and denote the maximal number of response regions of a network of d layers each with w units as $r(w, d)$. Our presentation of the proof of lemma 4 gives more insight into response regions.

3.4 Folding transformations

Montufar et al. (2014) present the concept of folding transformation and their implementation with ReLUs. Looking at one or more layers as a function $f : \mathbb{R}^2 \to \mathbb{R}^2$, a folding transformation maps a part of the input space to coincide with another. Subsequent operations on the resulting space will apply to both parts, indifferently to their origin in their initial position. As a simple example, consider a ReLU MLP of input dimension 1. A simple folding two-layer transformation could easily model the function $f(x) = |x|$, mapping the negative input values to their positive counterparts. Then, any composed operation in subsequent layers will apply to both the negative values and positive values. This simple mechanism of “code reuse” is key to our constructed deep network and its unit-efficiency.

4 Main Proof

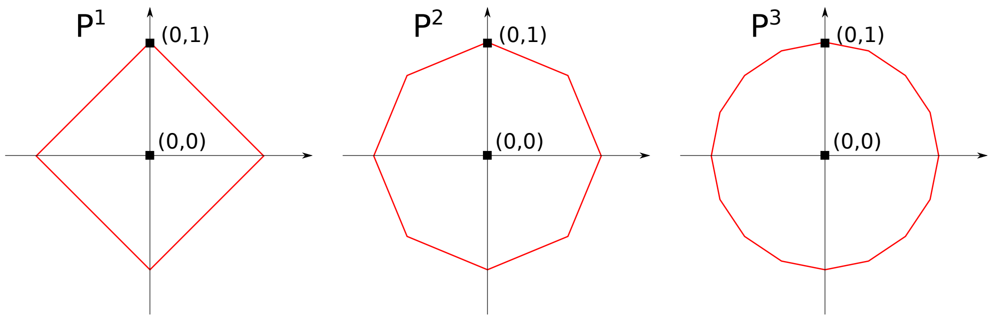

Figure 1: The problem family f_m is characterized by regular polygons, where polygon P^m has 2^{m+1} edges.

4.1 The problems f_m

Let P^m be a regular polygon with 2^{m+1} edges (Figure 1). Without loss of generality, P^m is centered around the origin, bounded by the unit circle, and has a vertex at $(0, 1)$.[2] The set of polygons $P^1, P^2, \ldots$ approaches the unit circle as $m \to \infty$. Let f_m be the function with decision boundary P^m:

$$f_m(x) = \begin{cases} 1 & x \in P^m \\ -1 & \text{otherwise} \end{cases}$$

Points within polygon P^m are of class 1, while other points are of class -1.

4.2 A bounded-depth network representing f_m must be exponentially wide.

We begin with proving (a) of Theorem 1. We will use the following lemmas, with proofs provided later.

[2]Any other regular polygon can be shifted, rotated and scaled to these conditions using affine transformation with $O(1)$ parameters.

Lemma 2. *A rectified MLP is a piecewise linear function.*

Proof: A linear layer followed by a rectifier is piecewise-linear. A composition of piecewise linear functions is itself piecewise linear.

Lemma 3. *Modeling f_m as a piecewise linear function requires at least 2^m response regions.*

Lemma 4. *Rectified MLP with input in $\mathbb{R}^2$, with d hidden layers and where each layer has width of at most w, has at most $2^{2d\log_2 w} = 2^{2d} \cdot 2^{\log_2 w}$ response regions.*

From lemma 4, it is clear that in order to achieve $2^{m-2d} = O(2^m)$ response regions for a network with bounded depth d, we must grow w. To accommodate for the log factor, the growth in w must be exponential. As function f_m requires 2^m response regions (lemma 3), and m can be arbitrarily large, this proves (a).

4.3 EFFICIENT DEPTH-m SOLUTION EXISTS.

We now turn to prove (b). While lemma 4 tells us a network with 2^m response regions requires depth $O(m)$, it does not guarantee such a network exists. To prove (b), we construct such a network, with bounded width and linear depth. The construction is based on folding transformations.

We manually construct the regular polygon decision boundary for polygon P^m through exploitation of symmetry. Intuitively, our construction resembles children paper-cutting, where a sheet of paper is folded multiple times, then cut with scissors. Unfolding the paper reveals a complex pattern with distinctive symmetries. Tracing and cutting the same pattern without any paper folding would require much more effort. Analogously, we'll show how deep networks could implement "folds" through their layers and how ReLU operations, like scissor cuts, are mirrored through the symmetries induced by these folds. Conversely, shallow networks, unable to "fold the paper", must make many more cuts — i.e. must have much more units in order to create the very same pattern.

Formally, our deep network operates as follows: first, it folds across both the X and Y axes, mapping the input space into the first quadrant $(x, y) \mapsto (|x|, |y|)$. It now has to deal only with the positive part of the decision boundary. It then proceeds in steps, in which it first rotates the space around the origin until the remaining decision boundary is symmetric around the X axis, and then folds around the X axis, resulting in half the previous decision boundary, in the first quadrant. This process continues until the decision boundary is a single line, which can be trivially separated. The first step cuts the number of edges in the decision boundary by a factor of four, while each subsequent rotate + fold sequence further cuts the number of polygon edges in half.

This process is depicted in figure 2

More formally, we require four types of transformations:

- $foldXY(\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}) : \mathbb{R}^2 \to \mathbb{R}^2$ — initial mapping of input to the first quadrant.
- $rotate_\Theta(\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}) : \mathbb{R}^2 \to \mathbb{R}^2$ — clockwise rotation around the origin by an angle of Θ.
- $foldX(\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}) : \mathbb{R}^2 \to \mathbb{R}^2$ — folding across the X axis.
- $top(\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}) : \mathbb{R}^2 \to \mathbb{R}^1$ — the final activation layer.

These operations are realized in the network layers, using a combination of linear matrix operations and ReLU activations. The $rotate$ operation is simply a rotation matrix. Rotating by an angle of Θ is realized as:

$$rotate_\Theta(\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}) = \begin{bmatrix} cos(\Theta) & -sin(\Theta) \\ sin(\Theta) & cos(\Theta) \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

The initial folding across both X and Y axes first transforms the input (x, y) to $(x, -x, y, -y)$ using a linear transformation. It then trims the negative values using a ReLU, and sums the first two and last two coordinates using another linear operation, resulting in:

$$foldXY(\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \sigma(\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix})$$

Where σ is the elementwise ReLU activation function. Folding across the X axes is similar, but as all x values are guaranteed to be positive, we do not need to consider $-x$.

$$foldX(\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \sigma(\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix})$$

Finally, the final classification layer is:

$$top(\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}) = \text{sign}(a \cdot x_0 + b \cdot x_1 + c)$$

Composing these operations, the constructed network for problem f_m has the form:

$$f_{MLP}(x) = foldXY \circ rotate_{\pi/4} \circ foldX \circ rotate_{\pi/8} \circ foldX \circ ...rotate_{\pi/2^{m+1}} \circ foldX \circ top$$

Note that the angle of rotation is decreased by a factor of 2 in every subsequent $rotate$. The $rotate$ and $foldX$ transformations pair, folds input space along a symmetry axis and effectively reduce the problem by half. This results in a $foldXY$ operation followed by a sequence of m $rotate \circ foldX$ operations, followed by top.

Marking a $fold$ operation as $\mathbf{F}\sigma\mathbf{C}$ and a rotate operation as $\mathbf{R}$, where $\mathbf{F}, \mathbf{C}, \mathbf{R}$ being matrices, the MLP takes the form: $\mathbf{F}\sigma\mathbf{CRF}\sigma\mathbf{CRF}\sigma\mathbf{CRF}\ldots$ where a sequence $\mathbf{CRF}$ of matrix operations can be collapsed into a single matrix M. This brings us to the familiar MLP form that alternates matrix multiplications and ReLU activations. Overall, the network has $m + 1$ non-linear activations (from m $foldX$ operations and 1 $foldXY$ operation), resulting in $m + 1$ layers.

The response regions produced by the constructed MLP and by a shallow network are depicted in Figure 3.

5 Proofs of Lemmas

5.1 Lemma 3

Modeling P^m as a piecewise linear function requires at least 2^m response regions.

Proof: consider the polygon P_m, and let MLP_m be a ReLU MLP (piecewise-linear function) correctly classifying the problem. Let V_{even} be the set of every *second* vertex along a complete traversal of P_m. For each vertex take an ϵ step away from the origin to create V'_{even} (see Figure 4a for an illustration). Each of the points in V'_{even} are strictly outside P^m and therefore should be classified as class -1.

The response regions produced by MLP_m are both convex and linear. Let p_i, p_j by two arbitrary points in V'_{even}, $p_i \neq p_j$. We will show that p_i, p_j belong in different response regions. Assume by contradiction that p_i, p_j are in the same response region. By convexity all points in a straight line between p_i and p_j are also in the same response region. Also, by linearity these points have an activation value between p_j and p_j and therefore should also be classified as class -1. From the problem construction we know that lines between the even vertices of P_m cross the class boundary as demonstrated in Figure 4b. Therefore, p_i and p_j must lay in different response regions. Since p_i and p_j are arbitrary, MLP_m's number of response regions is at least $|V'_{even}| = 2^m$.

5.2 Lemma 4

Rectified MLP with input in $\mathbb{R}^2$, with d hidden layers and where each layer has width of at most w, has at most $2^{2d \log_2 w}$ response regions.

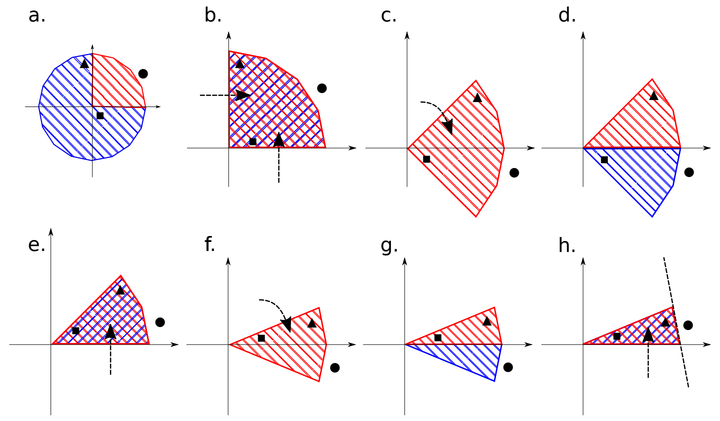

Figure 2: Constructing P^3 using folding and rotation transformations. The 3 blackened markers show how 3 points in the input space are transformed during this process. (a-b) a $foldXY$ operation maps all points to the first quadrant. (c) the slice is rotated clockwise by 45°using a linear transformation. (d-e) the bottom half is mapped into the first quadrant using a $foldX$ operation. (f) rotate by $45/2°$. (g-h) folding. final rotation by $45/4°$ and a final linear decision boundary that correctly classifies the three points.

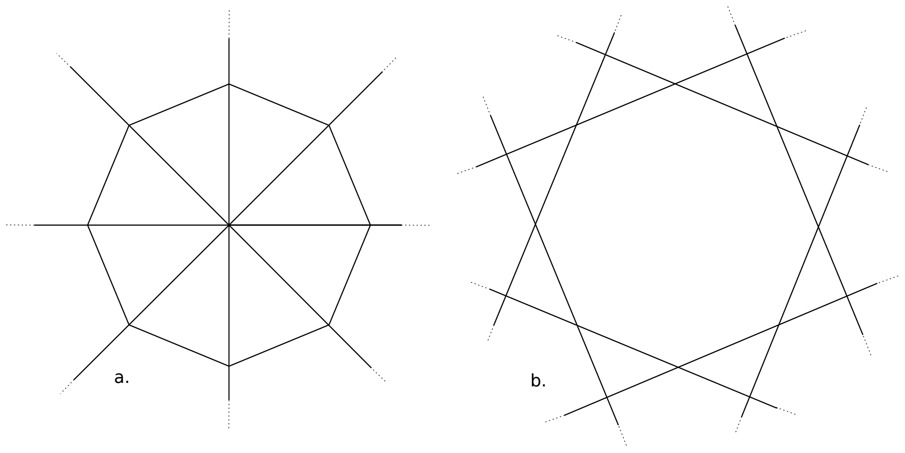

Figure 3: : a.) The response regions of the constructed solution for P_2. b.) A shallow, one layer MLP that solve P_2 - Such an MLP must model each of the regular polygon edges separately .

Proof: Raghu et al. (2017) prove a version of this lemma for input space $\mathbb{R}^n$, which have at most $O(w^{nd}) = O(2^{nd \log_2 w})$ response region. We prove the more restricted case of inputs in $\mathbb{R}^2$, in a similar fashion. We first consider the bound for 1 hidden-layer networks, then extend to d layers. The first part of the proof follows classic and basic results in computational geometry. The argument in the second part (move from 1 to d layers) is essentially the same one of Raghu et al. (2017).

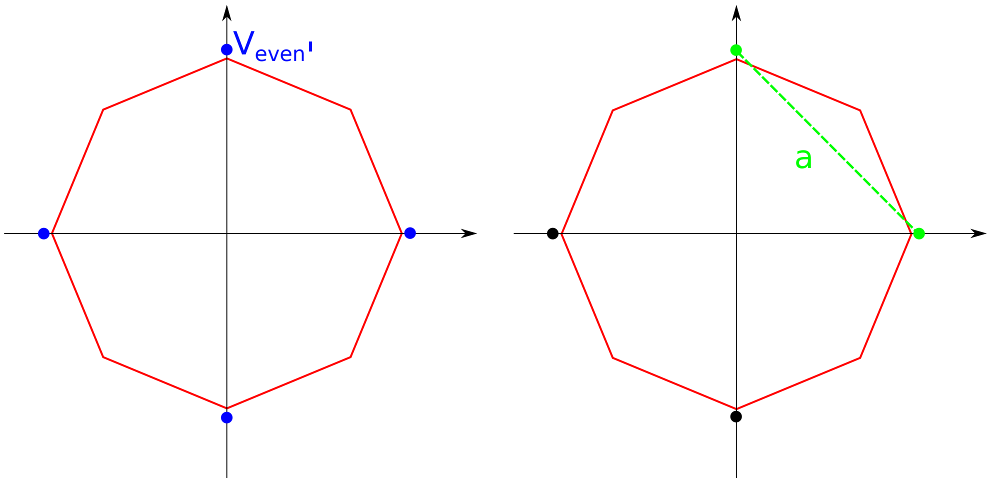

Figure 4: Left: V'_{even} are created by taking every second vertex of P^m then moving them slightly such that they are strictly outside P^m. Right: a chord a in green connecting any two vertices of V'_{even}, must cross P^m. Had both of the chord vertices been in the same response region, by convexity so do all points on a. By linearity, the final network activation of a's points will interpolate the activation of a's endpoints.

Number of regions in a line-arrangement of n lines We start by showing that the maximal number of regions in $\mathbb{R}^2$ created by a line arrangement of n lines, denoted $r(n)$, is $r(n) \leq n^2$. This is based on classic result from computational geometry (Zaslavsky, 1975). Initially, the entire space is a region. A single line divides the space in two, adding one additional region. What happens as we add additional lines? The second line intersects[3] with the first, and splits each of the previous regions in two, adding 2 more regions. The third line intersects with both lines, dividing the line into three sections. Each section splits a region, adding 3 more regions. Continuing this way, the ith line intersects $i-1$ lines, resulting in i sections, each intersecting a region and thus adding a region. Figure 5 shows this for the 4th line. We get:

$$r(n) = 1 + 1 + 2 + 3 + 4 + \ldots + n = 1 + \sum_{i=1}^{n} i = 1 + \frac{n(n+1)}{2} \leq n^2 \text{ (for } n > 2)$$

A 1 hidden-layer ReLU network is a line arrangement Consider a network of the form $y = \mathbf{v}(\mathbf{A}x + \mathbf{b})$ where the matrix $\mathbf{A}$ projects the input x to w dimensions, and the vector $\mathbf{v}$ combines them into a weighted sum. The entire input space is linear under this network: the output is linear in the input.[4] When setting an ReLU activation function after the first layer: $y = \mathbf{v}\sigma(\mathbf{A}x + \mathbf{b})$ we get a 1-hidden layer ReLU network. For a network with a width w hidden layer ($\mathbf{A} \in \mathbb{R}^{w\times 2}$), we get w linear equations, $\mathbf{A}^{(i)}x + \mathbf{b}^(i)$ corresponding to w piecewise linear functions: each function has a section where it behaves according to its corresponding equation (the "active" section), and a section where it is 0 (the "rectified" section). The input transitions between the active and the rectified sections of function i at the boundary given by $\mathbf{A}^{(i)}x + \mathbf{b}^(i) = 0$. Thus, each ReLU neuron corresponds to a line that splits the input space into two: one input region where the neuron is active, and one where it is rectified. Within each region, the behavior of the neuron is linear. For a width w network, we have w such lines — a line arrangement of w lines. The arrangement splits the space into at most $r(w) < w^2$ convex cells, where each cell corresponds to a set of active neurons. Within each cell, the behavior of the input is linear. Such a cell is called a *linear region*.

Additional Layers (Raghu et al., 2017; Pascanu et al., 2013) Additional layers further split the linear regions. Consider the network after $d-1$ layers, and a given linear region R. Within R, the

[3]We assume the added lines are not parallel to any previous line, and do not cross an intersection of previous lines. It is easy to be convinced that such cases will split the space into fewer regions.

[4]We can then set a linear classifier by setting a threshold on y, this will divide the input space in 2, with a single line.

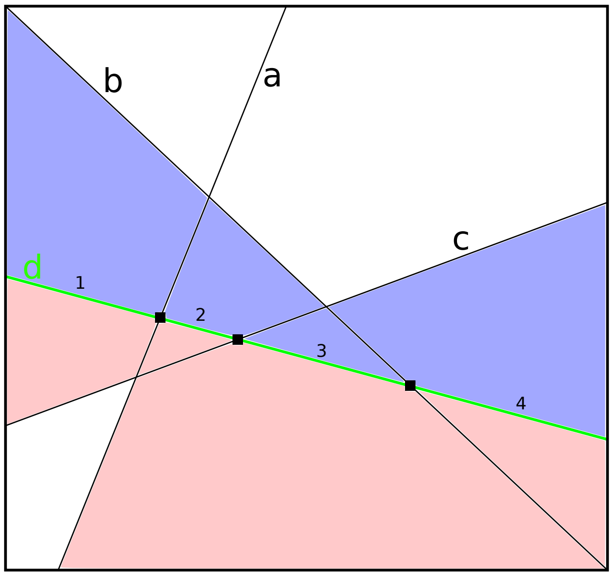

Figure 5: By iteratively introducing lines we can count the maximal number of regions created by k lines. In general positions, the 4th introduced line (d. in greed) will intersect its 3 predecessor in 3 different points. These will create 4 sections, each splitting a region into two (red-blue) hence adding 4 regions to the total count.

set of active neurons in layers $< d-1$ is constant, and so within the region the next layer computes a linear function of the input. As above, the ReLU activation then again gives w line equations, but this time these equations are only valid within R. The next layer than splits R into at most $r(w)$ regions.

Max number of regions in deep networks Raghu et al. (2017) Consider a network with two hidden layers of width w. The first layer introduced at most $r(w) \leq w^2$ convex regions. As we saw above, for the second layer each region can be split again into at most $r(w)$ regions, resulting in at most $w^2 \cdot w^2 = (w^2)^2$ regions. Applying this recursively, we get that the maximal number of regions in a depth d width w ReLU MLP network is $r(w, d) = w^{2d}$. By writing w as $2^{log_2 w}$ we get the bound asserted in the lemma, concluding its proof.

6 Conclusion

We present a depth separation proof for ReLU MLP which is fully self contained and uses only basic mathematical concepts and proof techniques. To the best of our knowledge, the problem construction, the main proof, and lemma 3 are novel. The proof of lemma 4 is a simplified presentation of a more general existing result.

References

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.

Monica Bianchini and Franco Scarselli. On the complexity of shallow and deep neural network classifiers. In *ESANN*, 2014.

Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pp. 666–674, 2011.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pp. 907–940, 2016.

Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. *arXiv preprint arXiv:1901.09021*, 2019.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pp. 2924–2932, 2014.

Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*, 2013.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2847–2854. JMLR. org, 2017.

Itay Safran and Ohad Shamir. Depth separation in relu networks for approximating smooth non-linear functions. *CoRR*, abs/1610.09887, 2016. URL `http://arxiv.org/abs/1610.09887`.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. *arXiv preprint arXiv:1711.02114*, 2017.

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

Matus Telgarsky. Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*, 2016.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94: 103–114, 2017.

Thomas Zaslavsky. *Facing up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes: Face-count Formulas for Partitions of Space by Hyperplanes*, volume 154. American Mathematical Soc., 1975.