# Empirical Bounds on Linear Regions of Deep Rectifier Networks

**Thiago Serra,**[1] **Srikumar Ramalingam**[2]
[1]Bucknell University, USA
[2]The University of Utah, USA
thiago.serra@bucknell.edu, srikumar@cs.utah.edu

## Abstract

We can compare the expressiveness of neural networks that use rectified linear units (ReLUs) by the number of linear regions, which reflect the number of pieces of the piecewise linear functions modeled by such networks. However, enumerating these regions is prohibitive and the known analytical bounds are identical for networks with same dimensions. In this work, we approximate the number of linear regions through empirical bounds based on features of the trained network and probabilistic inference. Our first contribution is a method to sample the activation patterns defined by ReLUs using universal hash functions. This method is based on a Mixed-Integer Linear Programming (MILP) formulation of the network and an algorithm for probabilistic lower bounds of MILP solution sets that we call MIPBound, which is considerably faster than exact counting and reaches values in similar orders of magnitude. Our second contribution is a tighter activation-based bound for the maximum number of linear regions, which is particularly stronger in networks with narrow layers. Combined, these bounds yield a fast proxy for the number of linear regions of a deep neural network.

## 1 Introduction

Neural networks with piecewise linear activations have become increasingly more common along the past decade, in particular since (Nair and Hinton 2010; Glorot, Bordes, and Bengio 2011). The simplest and most commonly used among such forms of activation is the Rectifier Linear Unit (ReLU), which outputs the maximum between 0 and its input argument (Hahnloser et al. 2000; LeCun, Bengio, and Hinton 2015). In the functions modeled by these networks with ReLUs, we can associate each part of the domain in which the network corresponds to an affine function with a particular set of units having positive outputs. We say that those are the active units for that part of the domain. Consequently, over its entire input domain, the network models a piecewise-linear function (Arora et al. 2018). Counting these "pieces" into which the domain is split, which are often denoted as linear regions, is one way to compare the expressiveness of models defined by networks with different configurations or coefficients. The theoretical analysis of the number of input regions in deep learning dates back to at least (Bengio 2009), and recent experiments have shown that the number of regions relates to the accuracy of similar networks (Serra, Tjandraatmadja, and Ramalingam 2018).

The study of linear regions in different network configurations has led to some interesting observations. For example, in a rectifier network with $n$ ReLUs, we learned that not all configurations – and in some cases none – can reach the ceiling of $2^n$ regions (the number of possible sets of active units). On the one hand, we can construct networks where the number of regions is exponential on network depth (Pascanu, Montúfar, and Bengio 2014; Montúfar et al. 2014). On the other hand, there is a bottleneck effect by which the number of active units on each layer affects how the regions are partitioned by subsequent layers due to the dimension of the space containing the image of the function, up to the point that even shallow networks define more linear regions than their deeper counterparts as the input dimension approaches $n$ (Serra, Tjandraatmadja, and Ramalingam 2018). Due to the linear local behavior, the size and shape of linear regions have been explored for provable robustness. In that case, one wants to identify large stable regions and move certain points away from their boundaries (Wong and Kolter 2018; Elsayed et al. 2018; Croce, Andriushchenko, and Hein 2019; Guang-He Lee 2019). To some extent, we may regard the number and the geometry of linear regions as complementary. We can also use linear regions of a network to obtain smaller networks that are equivalent or a global approximation (Kumar, Serra, and Ramalingam 2019). However, we need faster methods to count or reasonably approximate the number of linear regions to make such metric practical.

### Linear Regions and Network Expressiveness

The literature on counting linear regions has mainly focused on bounding their maximum number. Lower bounds are obtained by constructing networks defining increasingly larger number of linear regions (Pascanu, Montúfar, and Bengio 2014; Montúfar et al. 2014; Arora et al. 2018; Serra, Tjandraatmadja, and Ramalingam 2018). Upper bounds are proven using the theory of hyperplane arrangements (Zaslavsky 1975) along with other analytical insights (Raghu et al. 2017; Montúfar 2017; Serra, Tjandraatmadja, and Ramalingam 2018). So far, these bounds are only identical – and thus tight – in the case of one-dimensional inputs (Serra, Tjandraatmadja, and Ramalingam 2018). Both of these lines have explored deepening connections with polyhedral theory, but some of these results have also been recently revisited using tropical algebra (Zhang, Naitzat, and Lim 2018; Charisopoulos and Maragos 2018). The linear regions of a trained network can be enumerated as the projection of a Mixed-Integer Linear Program (MILP) on the binary variables defining if each unit is active (Serra, Tjandraatmadja, and Ramalingam 2018). Another recent line of work focuses on analytical results for the average number of linear regions in practice (Hanin and Rolnick 2019a; 2019b).

Other methods to study neural network expressiveness include universal approximation theory (Cybenko 1989), VC dimension (Bartlett, Maiorov, and Meir 1998), and trajectory length (Raghu et al. 2017). A network architecture can be studied and analyzed based on the largest class of functions that it can approximate. For example, it has been shown that any continuous function can be modeled using a single hidden layer of sigmoid activation functions (Cybenko 1989). The popular ResNet architecture (He et al. 2016) with a single ReLU in every hidden layer can be a universal approximator (Lin and Jegelka 2018). Furthermore, a rectifier network with a single hidden layer can be trained to global optimality in polynomial time on the data size, but exponential on the input dimension (Arora et al. 2018). The use of trajectory length for expressiveness is related to linear regions, i.e., by changing the input along a one dimensional path we study the transition across linear regions. Certain critical network architectures using leaky ReLUs are identified to produce connected decision regions (Nguyen, Mukkamala, and Hein 2018). To avoid such degenerate cases, one needs to use sufficiently wide hidden layers. However, this result is mainly applicable to leaky ReLUs and not to standard ReLUs (Beise, Cruz, and Schröder 2018).

## Counting and Probabilistic Inference

Approximating the size of a set of binary vectors, such as those units that are active on each linear region of a neural network, has been extensively studied in the context of propositional satisfiability (SAT). A SAT formula on a set $V$ of Boolean variables has to satisfy a set of predicates. Counting solutions of SAT formulas is #P-complete (Toda 1989), but one can approximate the number of solutions by making a relatively small number of solver calls to restricted formulas. This line of work relies on hash functions with good statistical properties to partition the set of solutions $S$ into subsets having approximately half of the solutions each. After restricting a given formula $r$ times to either subset, one can test if the subset is empty. Intuitively, $|S| \geq 2^r$ with some probability if these subsets are more often nonempty, or else $|S| < 2^r$. SAT formulas are often restricted with predicates that encode XOR constraints, which can be interpreted in terms of 0–1 variables as restricting the sum of a subset $U \subset V$ of the variables to be either even or odd. XOR constraints are universal hash functions (Carter and Wegman 1979), which enable approximate counting in polynomial time with an NP-oracle (Sipser 1983; Stockmeyer 1985). Interestingly, formulas with a unique solution are as hard as those with multiple solutions (Valiant and Vazirani 1986). From a theoretical standpoint, such approximations are thus not much harder than obtaining a feasible solution.

In the seminal MBound algorithm (Gomes, Sabharwal, and Selman 2006a), XOR constraints on sets of variables with a fixed size $k$ yield the probability that $2^r$ is either a lower or an upper bound. The probabilistic lower bounds are always valid but get better as $k$ increases, whereas the probabilistic upper bounds are only valid if $k = |V|/2$. In practice, these lower bounds can be good for small $k$ (Gomes et al. 2007b). These ideas were later extended to constraint satisfaction problems (Gomes et al. 2007a). Some of the subsequent work has been influenced by uniform sampling results from (Gomes, Sabharwal, and Selman 2006b), where the fixed size $k$ is replaced with an independent probability $p$ of including each variable in each XOR constraint. That includes the ApproxMC and the WISH algorithms (Chakraborty, Meel, and Vardi 2013; Ermon et al. 2013), which rely on finding more solutions of the restricted formulas but generate $(\sigma, \epsilon)$ certificates by which, with probability $1 - \sigma$, the result is within $(1 \pm \epsilon)|S|$. Later work has provided upper bound guarantees when $p < 1/2$, showing that the size of those sets can be $\Theta\big(log(|V|)\big)$ (Ermon et al. 2014; Zhao et al. 2016). Others have tackled this issue differently. One approach limited the counting to any set of variables $I$ for which any assignment leads to at most one solution in $V$, denoting those as minimal independent supports (Chakraborty, Meel, and Vardi 2014; Ivrii et al. 2016). Another approach broke with the independent probability $p$ by using each variable the same number of times across $r$ XOR constraints (Achlioptas and Jiang 2015; Achlioptas, Hammoudeh, and Theodoropoulos 2018). Related work on MILP has only focused on upper bounds based on relaxations (Jain, Kadioglu, and Sellmann 2010).

## Contributions of This Paper

We propose empirical bounds based on the weight and bias coefficients of trained networks, which are the first able to compare networks with same configuration of layers. We also suggest replacing the potential number of linear regions $N$ of an architecture with the *Minimum Activation Pattern Size* (MAPS) $\eta = \log_2 N$. This value can be interpreted as the number of units that any network should have in order to define as many linear regions as another network when adjacent linear regions map to distinct affine functions.

Our main technical contributions are the following:

(i) We introduce a probabilistic lower bound based on sampling the activation patterns of the trained network. More generally, we can approximate solutions of MILP formulations more efficiently than if directly extending SAT-based methods with the `MIPBound` algorithm introduced in Section 4. See results in Figure 2.

(ii) We refine the best known upper bound by further exploring how units partition the input space. With the theory in Section 5, we find that unit activity further contributes to the bottleneck effect caused by narrow layers (those with few units). See results in Table 1.

## 2 Preliminaries and Notations

We consider feedforward Deep Neural Networks (DNNs) with ReLU activations. Each network has $n_0$ input variables given by $\boldsymbol{x} = [x_1\ x_2\ \ldots\ x_{n_0}]^T$ with a bounded domain $\mathbb{X}$ and $m$ output variables given by $\boldsymbol{y} = [y_1\ y_2\ \ldots\ y_m]^T$. Each hidden layer $l \in \mathbb{L} = \{1, 2, \ldots, L\}$ has $n_l$ hidden neurons indexed by $i \in \mathbb{N}_l = \{1, 2, \ldots, n_l\}$ with outputs given by $\boldsymbol{h}^l = [h_1^l\ h_2^l \ldots h_{n_l}^l]^T$. For notation simplicity, we may use $\boldsymbol{h}^0$ for $\boldsymbol{x}$ and $\boldsymbol{h}^{L+1}$ for $\boldsymbol{y}$. Let $\boldsymbol{W}^l$ be the $n_l \times n_{l-1}$ matrix where each row corresponds to the weights of a neuron of layer $l$. Let $\boldsymbol{b}^l$ be the bias vector used to obtain the activation functions of neurons in layer $l$. The output of unit $i$ in layer $l$ consists of an affine transformation $g_i^l = \boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l$ to which we apply the ReLU activation $h_i^l = \max\{0, g_i^l\}$.

We regard the DNN as a piecewise linear function $F : \mathbb{R}^{n_0} \to \mathbb{R}^m$ that maps the input $\boldsymbol{x} \in \mathbb{X} \subset \mathbb{R}^{n_0}$ to $\boldsymbol{y} \in \mathbb{R}^m$. Hence, the domain is partitioned into regions within which $F$ corresponds to an affine function, which we denote as linear regions. Following the literature convention (Raghu et al. 2017; Montúfar 2017; Serra, Tjandraatmadja, and Ramalingam 2018), we characterize each linear region by the set of units that are active in that domain. For each layer $l$, let $\mathbb{S}^l \subseteq \{1, \ldots, n_l\}$ be the activation set in which $i \in S^l$ if and only if $h_i^l > 0$. Let $\mathcal{S} = (\mathbb{S}^1, \ldots, \mathbb{S}^l)$ be the activation pattern aggregating those activation sets. Consequently, the number of linear regions defined by the DNN is the number of nonempty sets in $\boldsymbol{x}$ among all possible activation patterns.

## 3 Counting and MILP Formulations

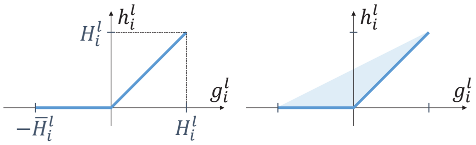
We can represent each linear region defined by a rectifier network with $n$ hidden units on domain $\mathbb{X}$ by a distinct vector in $\{0, 1\}^n$, where each element denotes if a unit is active or not. Such vector can be embedded into an MILP formulation mapping network inputs to outputs (Serra, Tjandraatmadja, and Ramalingam 2018). For a neuron $i$ in layer $l$, we use such binary variable $z_i$, vector $\boldsymbol{h}^{l-1}$ of inputs coming from layer $l-1$, variable $g_i^l$ for the value of the affine transformation $\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l$, variable $h_i^l = \max\{0, g_i^l\}$ denoting the output of the unit, and a variable $\bar{h}_i^l$ denoting the output of a complementary fictitious unit $\bar{h}_i^l = max\left\{0, -g_i^l\right\}$:

$$\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + b_i^l = g_i^l \tag{1}$$

$$g_i^l = h_i^l - \bar{h}_i^l \tag{2}$$

$$h_i^l \leq H_i^l z_i^l, \qquad \bar{h}_i^l \leq \bar{H}_i^l (1 - z_i^l) \tag{3}$$

$$h_i^l \geq 0, \qquad \bar{h}_i^l \geq 0, \qquad z_i^l \in \{0, 1\} \tag{4}$$



(a) (b)

Figure 1: (a) ReLU mapping $h_i^l = \max\{0, g_i^l\}$; and (b) Convex outer approximation on $(g_i^l, h_i^l)$.

For correctness, constants $H_i^l$ and $\bar{H}_i^l$ should be positive and as large as $h_i^l$ and $\bar{h}_i^l$ can be. In such case, the value of $g_i^l$ determines if the unit or its fictitious complement is active. However, constraints (1)–(4) allow $z_i^l = 1$ when $g_i^l = 0$. To count the number of linear regions, we consider the set of binary variables in the solutions where all active units have positive outputs, i.e., $h_i^l > 0$ if $z_i^l = 1$ (Serra, Tjandraatmadja, and Ramalingam 2018), thereby counting the positive solutions with respect to $f$ on the binary variables of

$$\max\ f \tag{5}$$

$$\text{s.t. } (1) - (4), f \leq h_i^l + (1 - z_i^l) H_i^l \quad l \in \mathbb{L};\ i \in \mathbb{N}_l \tag{6}$$

$$\boldsymbol{x} \in \mathbb{X} \tag{7}$$

The solutions on $z$ can be enumerated using the one-tree algorithm (Danna et al. 2007), in which the branch-and-bound tree used to obtain the optimal solution of the formulation above is further expanded to collect near-optimal solutions up to a given limit. In general, finding a feasible solution to a MILP is NP-complete (Cook 1971) and thus optimization is NP-hard. However, a feasible solution in our case can be obtained from any valid input (Fischetti and Jo 2018). While that does not directly imply that optimization over neural networks is easy, it hints at good properties to explore.

MILP formulations have been used for network verification (Lomuscio and Maganti 2017; Dutta et al. 2018) and evaluation of adversarial perturbations (Cheng, Nührenberg, and Ruess 2017; Fischetti and Jo 2018; Tjeng, Xiao, and Tedrake 2019; Xiao et al. 2018; Anderson et al. 2019). Other applications relax the binary variables as continuous variables in $[0, 1]$ or use the Linear Programming (LP) formulation of a particular linear region (Bastani et al. 2016; Ehlers 2017; Wong and Kolter 2018), which is defined using $\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + b_i^l \geq 0$ for active units and the complement for inactive units. Although equivalent, these MILP formulations may differ in strength (Fischetti and Jo 2018; Tjeng, Xiao, and Tedrake 2019; Huchette 2018; Anderson et al. 2019). When the binary variables are relaxed, their linear relaxation may be different. We say that an MILP formulation $A$ is stronger than a formulation $B$ if, when projected on common sets of variables, the linear relaxation of $A$ is a subset of the linear relaxation of $B$. Formulation strength is often used as a proxy for solver performance.

Differences in strength may be due to smaller values for constants such as $H_i^l$ and $\bar{H}_i^l$, additional valid inequalities to remove fractional values of $\boldsymbol{z}$, or an extended formulation with more variables. Let us consider the strength of the formulation for each ReLU activation $h_i^l = \max\{0, g_i^l\}$. Ideally, we want the projection on $g_i^l$ and $h_i^l$ to be the convex outer approximation of all possible combined values of those variables (Wong and Kolter 2018), as illustrated in Figure 1.

**Lemma 1.** *If $H_i^l = \arg\max_{g^{l-1}}\{g_i^l\} \geq 0$ and $\bar{H}_i^l = \arg\max_{g^{l-1}}\{-g_i^l\} \geq 0$, then the linear relaxation of (2)–(4) defines the convex outer approximation on $(g_i^l, h_i^l)$.*

Lemma 1 shows that the smallest possible values of $H_i^l$ and $\bar{H}_i^l$ are necessary to obtain a stronger formulation. The proof can be found in Appendix A. A similar claim without proof is found in (Huchette 2018).

When $\mathbb{X}$ is defined by a box, and thus the domain of each input variable $x_i$ is an independent continuous interval, then the smallest possible values for $H_i^1$ and $\bar{H}_i^1$ can be computed with interval arithmetic by taking element-wise maxima (Cheng, Nührenberg, and Ruess 2017; Serra, Tjandraatmadja, and Ramalingam 2018). When extended to subsequent layers, this approach may overestimate $H_i^l$ and $\bar{H}_i^l$ as the maximum value of the outputs are not necessarily independent. More generally, if $\mathbb{X}$ is polyhedral, we can obtain the smallest values for these constants by solving a sequence of MILPs (Fischetti and Jo 2018; Tjeng, Xiao, and Tedrake 2019). We can use $H_i^{l'} = \max \{g_i^{l'} : (1) - (4)\ \forall l \in \{1, \ldots, l'-1\},\ i \in \mathbb{N}_l, x \in \mathbb{X}\}$. If $H_i^{l'} \leq 0$, the unit is always inactive, denoted as *stably inactive*, and we can remove such units from the formulation. Similarly, we can use $\bar{H}_i^{l'} = -\min \{g_i^{l'} : (1) - (4)\ \forall l \in \{1, \ldots, l'-1\},\ i \in \mathbb{N}_l, x \in \mathbb{X}\}$. If $\bar{H}_i^{l'} < 0$, the unit is always active, denoted as *stably active*, and we can simply replace constraints (1)-(4) with $h_i^l = g_i^l$. In certain large networks, many units are stable (Tjeng, Xiao, and Tedrake 2019). The remaining units, where activity depends on the input, are denoted *unstable*.

We propose some valid inequalities involving consecutive layers of the network. For unit $i$ to be active when $b_i^l \leq 0$, there must be a positive contribution, and thus some unit $j$ in layer $l-1$ such that $\boldsymbol{W}_{ij}^l > 0$ is also active. Hence, for each layer $l \in \{2, \ldots, L\}$ and unit $i \in \mathbb{N}_l$ such that $b_i^l \leq 0$,

$$z_i^l \leq \sum_{j \in \{1, \ldots, n_{l-1}\} : \boldsymbol{W}_{ij}^l > 0} z_j^{l-1}. \tag{8}$$

Similarly, unit $i$ is only inactive when $b_i^l > 0$ if some unit $j$ in layer $l-1$ such that $\boldsymbol{W}_{ij}^l < 0$ is active. Likewise, for each layer $l \in \{2, \ldots, L\}$ and unit $i \in \mathbb{N}_l$ such that $b_i^l > 0$,

$$(1 - z_i^l) \leq \sum_{j \in \{1, \ldots, n_{l-1}\} : \boldsymbol{W}_{ij}^l < 0} z_j^{l-1}. \tag{9}$$

Let us denote unstable units in which $b_i^l \leq 0$, and thus (8) applies, as *inactive leaning*; and those in which $b_i^l > 0$, and thus (9) applies, as *active leaning*. Within linear regions where none among the units of the previous layer in the corresponding inequalities is active, these units can be regarded as stably inactive and stably active, respectively. We will use the same ideas to obtain better bounds in Section 5.

## 4 Approximate Lower Bound

We can use approximate model counting to estimate the size of the set of binary vectors corresponding to the activation patterns of a neural network. Essentially, we restrict the set of solutions by iteratively adding constraints with good sampling properties, such as XOR, until the problem becomes infeasible. Based on how many constraints it takes to make the formulation infeasible across many runs, we obtain bounds on the number of solutions with a certain probability. We describe in Section 1 how this type of approach has been extensively studied in the SAT literature.

The same ideas have not yet been extended to MILP formulations, where we exploit the specificity of MILP solvers to devise a more efficient algorithm. The assumption in SAT-based approaches is that each restricted formula entails a new call to the solver. Hence, obtaining a data point for each number of restrictions takes a linear number of calls. That has been improved to a logarithmic number of calls by orderly applying the same sequence of constraints up to each number of $r$ of XOR constraints, with which one can apply binary search to find the smallest $r$ that makes the formula unsatisfiable (Chakraborty, Meel, and Vardi 2016). In MILP solvers, we can test for all values of $r$ with a single call to the solver by generating parity constraints as lazy cuts, which can be implemented through callbacks. When a new solution is found, a callback is invoked to generate parity constraints. Each constraint may or may not remove the solution just found, since we preserve the independence between the solutions found and the constraints generated, and thus we may need to generate multiple parity constraints before yielding the control back to the solver. Algorithm 1 does that based on MBound (Gomes, Sabharwal, and Selman 2006a).

We denote Algorithm 1 as `MIPBound`. For each repetition of the outer `while` loop, parity constraints are added to a copy $F'$ of the formulation until it becomes infeasible. Appendix B discusses how to represent parity constraints in MILP using unit hypercube cuts from (Balas and Jeroslow 1972). The inner `while` loop corresponds to the solver call and the block between `repeat` and `until` is implemented as a lazy cut callback, which is invoked when an incumbent solution $s$ is found. After each solver call, the number of constraints $r$ to make $F'$ infeasible is used to increase $f[j]$ for all $j < r$, which counts the number of times that $F'$ remained feasible after adding $j$ constraints. If $F$ often remained feasible with $j$ constraints, we compute the probability $P_{j-1}$ that $|S| > 2^{j-1}$, which is explained in Appendix C.

## 5 Analytical Upper Bound

In order to bound the number of linear regions, we use activation hyperplanes and the theory of hyperplane arrangements. For each unit, the activation hyperplane $\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + b_i^l = 0$ splits the input space $\boldsymbol{h}^{l-1}$ into the regions where the unit is active ($\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + b_i^l > 0$) or inactive ($\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + b_i^l \leq 0$). The number of full-dimensional regions defined by the arrangement of $n_l$ hyperplanes in an $n_{l-1}$-dimensional space is at most $\sum_{j=0}^{n_{l-1}} \binom{n_l}{j}$ (Zaslavsky 1975). See Appendix D for related bounds (Raghu et al. 2017; Montúfar 2017; Serra, Tjandraatmadja, and Ramalingam 2018).

We can improve on these previous bounds by leveraging that some units of the trained network are stable for some or all possible inputs. First, note that only units in layer $l$ that can be active in a given linear region produced by layers 1 to $l-1$ affect the dimension of the space in which the linear region can be further partitioned by layers $l+1$ to $L$. Second, only the subset of these units that can also be inactive

**Algorithm 1** MIPBound computes the probability of some lower bounds on the solutions of a formulation $F$ with $n$ binary variables by adding parity constraints of size $k$

```
$i \leftarrow 0$
for $j \leftarrow 0 \rightarrow n$ do
    $f[j] \leftarrow 0$
end for
while Termination criterion not satisfied do
    $F' \leftarrow F$
    $i \leftarrow i + 1$
    $r \leftarrow 0$
    while $F'$ has some solution $s$ do
        repeat
            Generate parity constraint $C$ with $k$ of the variables
            $F' \leftarrow F' \cap C$
            $r \leftarrow r + 1$
        until $C$ removes $s$
    end while
    for $j \leftarrow 0 \rightarrow r - 1$ do
        $f[j] \leftarrow f[j] + 1$
    end for
end while
for $j \leftarrow 0 \rightarrow n - 1$ do
    $\delta \leftarrow f[j+1]/i - 1/2$
    if $\delta > 0$ then
        $P_j \leftarrow 1 - \left(\frac{e^{2.\delta}}{(1+2.\delta)^{1+2.\delta}}\right)^{i/2}$
    else
        break
    end if
end for
return Probabilities $P$
```

within that region, i.e., the unstable ones, counts toward the number of hyperplanes partitioning the linear region at layer $l$. Every linear region is contained in the same side of the hyperplane defined by each stable unit. Hence, let $\mathcal{A}_l(k)$ be the maximum number of units that can be active in layer $l$ if $k$ units are active in layer $l-1$; and $\mathcal{I}_l(k)$ be the corresponding maximum number of units that are unstable.

**Theorem 2.** *Consider a deep rectifier network with L layers with input dimension $n_0$ and at most $\mathcal{A}_l(k)$ active units and $\mathcal{I}_l(k)$ unstable units in layer l for every linear region defined by layers $1$ to $l-1$ when $k$ units are active in layer $l-1$. Then the maximum number of linear regions is at most*

$$\sum_{(j_1,\ldots,j_L)\in J} \prod_{l=1}^{L} \binom{\mathcal{I}_l(k_{l-1})}{j_l}$$

*where $J = \{(j_1,\ldots,j_L) \in \mathbb{Z}^L : 0 \le j_l \le \zeta_l, \zeta_l = \min\{n_0, k_1, \ldots, k_{l-1}, \mathcal{I}_l(k_{l-1})\}\ \}$ with $k_0 = n_0$ and $k_l = \mathcal{A}_l(k_{l-1}) - j_{l-1}$ for $l \in \mathbb{L}$.*

*Proof.* We define a recursive recurrence to bound the number of subregions within a region. Let $R(l,k,d)$ bound the maximum number of regions attainable from partitioning a region with dimension at most $d$ among those defined by layers $1$ to $l-1$ in which at most $k$ units are active in layer $l-1$. For the base case $l = L$, we have $R(L,k,d) = \sum_{j=0}^{\min\{\mathcal{I}_L(k),d\}} \binom{\mathcal{I}_L(k)}{j}$ since $\mathcal{I}_l(k) \le \mathcal{A}_l(k)$. The recurrence groups regions with same number of active units in layer $l$ as $R(l,k,d) = \sum_{j=0}^{\mathcal{A}_l(k)} N^l_{\mathcal{I}_l(k),d,j} R(l+1, j, \min\{j,d\})$ for $l = 1$ to $L-1$, where $N^l_{p,d,j}$ is the maximum number of regions with $j$ active units in layer $l$ from partitioning a space of dimension $d$ using $p$ hyperplanes.

Note that there are at most $\binom{\mathcal{I}_l(k)}{j}$ regions defined by layer $l$ when $j$ unstable units are active and there are $k$ active units in layer $l-1$, which can be regarded as the subsets of $\mathcal{I}_l(k)$ units of size $j$. Since layer $l$ defines at most $\sum_{j=0}^{\min\{\mathcal{I}_l(k),d\}} \binom{\mathcal{I}_l(k)}{j}$ regions with an input dimension $d$ and $k$ active units in the layer above, by assuming the largest number of active hyperplanes among the unstable units and also using $\binom{\mathcal{I}_l(k)}{\mathcal{I}_l(k)-j} = \binom{\mathcal{I}_l(k)}{j}$, then we define $R(l,k,d)$ for $1 \le l \le L-1$ as the following expression:

$$\sum_{j=0}^{\min\{\mathcal{I}_l(k),d\}} \binom{\mathcal{I}_l(k)}{j} R(l+1, \mathcal{A}_l(k)-j, \min\{\mathcal{A}_l(k)-j, d\}).$$

Without loss of generality, we assume that the input is generated by $n_0$ active units feeding the network, hence implying that the bound can be evaluated as $R(1, n_0, n_0)$:

$$\sum_{j_1=0}^{\min\{\mathcal{I}_1(k_0),d_1\}} \binom{\mathcal{I}_l(k_0)}{j_1} \cdots \sum_{j_L=0}^{\min\{\mathcal{I}_L(k_{L-1}),d_L\}} \binom{\mathcal{I}_L(k_{L-1})}{j_L},$$

where $k_0 = n_0$ and $k_l = \mathcal{A}_l(k_{l-1}) - j_{l-1}$ for $l \in \mathbb{L}$, whereas $d_l = \min\{n_0, k_1, \ldots, k_{l-1}\}$. We obtain the final expression by nesting the values of $j_1, \ldots, j_L$. □

Theorem 2 improves the result in (Serra, Tjandraatmadja, and Ramalingam 2018) when not all hyperplanes partition every region from previous layers ($\mathcal{I}_l(k_{l-1}) < n_l$) or not all units can be active (smaller intervals for $j_l$ due to $\mathcal{A}_l(k_{l-1})$).

Now we discuss how the parameters that we introduced in this section can be computed exactly, or else approximated. We first bound the value of $\mathcal{I}_l(k)$. Let $U_l^-$ and $U_l^+$ denote the sets of inactive leaning and active leaning units in layer $l$, and $U_l = U_l^+ \cup U_l^-$. For a given unit $i \in U_l^-$, we can define a set $J^-(l,i)$ of units from layer $l-1$ that, if active, can potentially make $i$ active. In fact, we can use the set in the summation of inequality (8), and therefore let $J^-(l,i) := \{j : 1 \le j \le n_{l-1}, \boldsymbol{W}^l_{ij} > 0\}$. For a given unit $i \in U_l^+$, we can similarly use the set in inequality (9), and let $J^+(l,i) := \{j : 1 \le j \le n_{l-1}, \boldsymbol{W}^l_{ij} < 0\}$. Conversely, let $I(l,j) := \{i : i \in U^+_{l+1}, j \in J^+(l+1,i)\} \cup \{i : i \in U^-_{l+1}, j \in J^-(l+1,i)\}$ be the units in layer $l+1$ that may be locally unstable if unit $j$ in layer $l$ is active.

**Proposition 3.** $\mathcal{I}_l(k) \le \max_S \left\{ \left| \bigcup_{j\in S} I(l-1,j) \right| : S \in \mathbb{S}^k \right\},$ *where $\mathbb{S}^k = \{S : S \subseteq \{1,\ldots,n_{l-1}\}, |S| \le k\}$.*

Next we bound the value of $\mathcal{A}_l(k)$ by considering a larger subset of the units in layer $l$ that only excludes locally inactive units. Let $n_l^+$ denote the number of stably active units in layer $l$, which is such that $n_l^+ \leq n_l - |U_l|$, and let $I^-(l,j) := \{i : i \in U_{l+1}^-, j \in J^-(l+1,i)\}$ be the inactive leaning units in layer $l+1$ that can be activated if unit $j$ in layer $l$ is active.

**Proposition 4.** *For $\mathbb{S}^k$ defined as before,* $\mathcal{A}_l(k) \leq n_l^+ + |U_l^+| + \max_S \left\{ \left| \bigcup_{j \in S} I^-(l-1,j) \right| : S \in \mathbb{S}^k \right\}$.

In practice, however, we may only need to inspect a small number of such subsets. In the average-case analysis presented in Appendix E, only $O(n_{l-1})$ subsets are needed. We also observed in the experiments that the minimum value of $k$ to maximize $\mathcal{I}_l(k)$ and $\mathcal{A}_l(k)$ is rather small. If not, we can approximate $\mathcal{I}_l(k)$ and $\mathcal{A}_l(k)$ with strong optimality guarantees $(1 - \frac{1}{e})$ using simple greedy algorithms for submodular function maximization (Nemhauser, Wolsey, and Fisher 1978). We discuss that possibility in Appendix F.

## 6 Experiments

We tested on rectifier networks trained on the MNIST benchmark dataset (LeCun et al. 1998), consisting of 22 units distributed in two hidden layers and 10 output units, with 10 distinct networks for each distribution of units between the hidden layers. See Appendix G for more details about the networks and the implementation. For each size of parity constraints $k$, which we denote as XOR-$k$, we measure the time to find the smallest coefficients $H_i^l$ and $\bar{H}_i^l$ for each unit along with the subsequent time of `MIPBound` (Algorithm 1). We let `MIPBound` run for enough steps to obtain a probability of 99.5% in case all tested constraints of a given size preserve the formulation feasible, and we report the largest lower bound with probability at least 95%. We also use the approach in (Serra, Tjandraatmadja, and Ramalingam 2018) to count the exact number of regions for benchmarking. Since counting can be faster than sampling for smaller sets, we define a DNN with $\eta < 12$ as small and large otherwise. We use Configuration Upper Bound (Configuration UB) for the bound in (Serra, Tjandraatmadja, and Ramalingam 2018). The upper bound from Theorem 2, which we denote as Empirical Upper Bound (Empirical UB), is computed at a small fraction of the time to obtain coefficients $H_i^l$ and $\bar{H}_i^l$ for the lower bound. We denote as APP-$k$ the average between the XOR-$k$ Lower Bound (LB) and Empirical UB.

Table 1 shows the gap closed by Empirical UB. Figure 2 (top) compares the bounds with the number of linear regions. Figure 2 (bottom) compares the time for exact counting and approximation. Figure 3 compares APP-$k$ with the accuracy of networks not having particularly narrow layers, in which case the number of regions relates to network accuracy (Serra, Tjandraatmadja, and Ramalingam 2018).

## 7 Conclusion

This paper introduced methods to obtain tighter bounds on the number of linear regions. These methods are considerably faster than direct enumeration, entail a probabilistic lower bound algorithm to count MILP solutions, and help understanding how ReLUs partition the input space.

Table 1: Gap (%) closed by Empirical UB between Configuration UB and the number of regions for widths $n_1; n_2; n_3$.

| Widths | Gap | Widths | Gap | Widths | Gap |
|---|---|---|---|---|---|
| 1;21;10 | 73.1 | 8;14;10 | 0 | 15;7;10 | 0.5 |
| 2;20;10 | 17.8 | 9;13;10 | 0 | 16;6;10 | 1 |
| 3;19;10 | 10.4 | 10;12;10 | 1 | 17;5;10 | 1.8 |
| 4;18;10 | 3.1 | 11;11;10 | 0 | 18;4;10 | 3.4 |
| 5;17;10 | 3.9 | 12;10;10 | 0 | 19;3;10 | 9.5 |
| 6;16;10 | 2 | 13;9;10 | 0.1 | 20;2;10 | 44.5 |
| 7;15;10 | 1.1 | 14;8;10 | 0.2 | 21;1;10 | 98.3 |

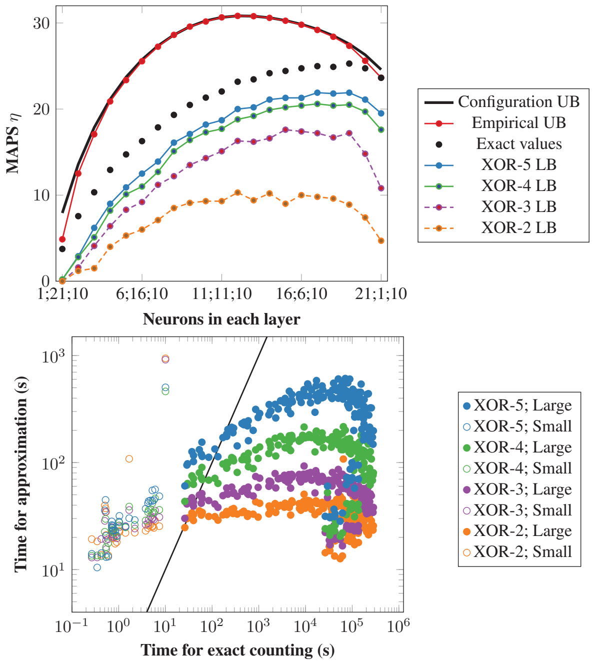


Figure 2: Top: Averages of the Empirical UB and XOR-$k$ LBs with probability 95% compared with the Configuration UB and the exact number of regions for 10 networks of each type. Bottom: Comparison of approximation vs. exact counting times by XOR size and number of regions.

Prior work on bounding the number of linear regions has first focused on the benefit of depth (Pascanu, Montúfar, and Bengio 2014; Montúfar et al. 2014), and then on the bottleneck effect that is caused by a hidden layer that is too narrow (Montúfar 2017) and more generally by small activation sets (Serra, Tjandraatmadja, and Ramalingam 2018). In our work, we looked further into how many units can possibly be active or not by taking into account the weights and the bias of each ReLU, which allows us to identify stable units. Stable units do not contribute as significantly to the number of linear regions. Consequently, we found that the bottleneck effect in the upper bound is even stronger in narrow layers, as evidenced in both extremes of Table 1.

The probabilistic lower bound is based on universal hashing functions to sample activation patterns, and more generally allows us to estimate the number of solutions on binary variables of MILP formulations. By exploiting call-

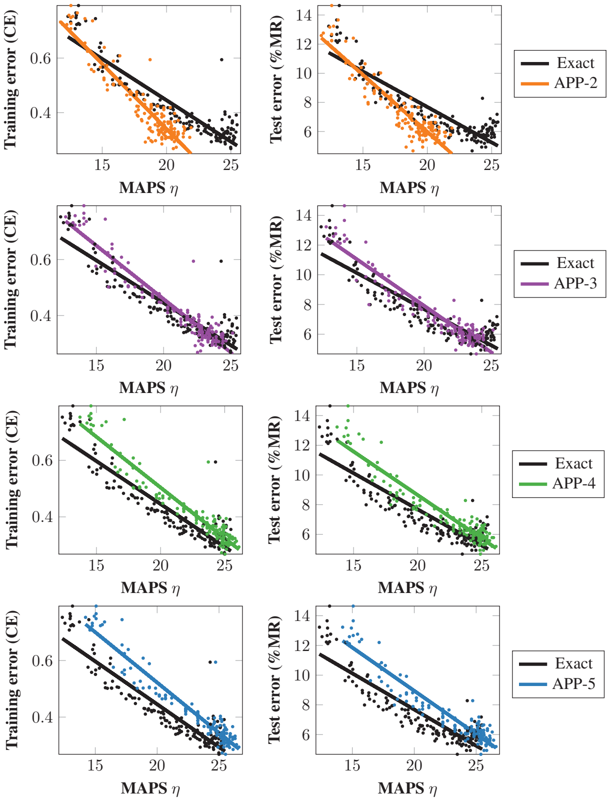


Figure 3: Regression on network accuracy for exact and approximated count in black and color, respectively. The approximated count averages lower and upper bounds. As the XOR size increases, these regressions become more parallel, and thus the approximate count regression is more accurate.

backs that are typical of MILP solvers, the number of solver calls in the proposed algorithm `MIPBound` does not depend on the number of sizes for which we evaluate the probabilistic bounds like in related work. The algorithm is orders of magnitude faster than exact counting on networks with a large number of linear regions. These bounds can be parameterized for a balance between precision and speed. Nevertheless, we noted that lower bounds from XOR constraints of size 2, which are faster to compute but not as accurate, can be used to compare relative expressiveness since the curves from XOR-2 to XOR-5 have a very similar shape.

When upper and lower bounds are combined, the weakest approximation still preserves a negative correlation with accuracy, hence indicating that it may suffice to compare networks for relative expressiveness. It is important to note that we do not need the exact count to compare two networks in terms of expressiveness or performance. Nevertheless, the stronger approximations produce more precise correlations, which is evidenced by the more parallel regressions and thus more stable gaps across network sizes.

**Appendices** An extended version of this paper with appendices can be found at https://arxiv.org/abs/1810.03370.

**Acknowledgements** We would like to thank Christian Tjandraatmadja, Michele Lombardi, and Tiago F. Tavares for their helpful suggestions and comments on this work.

## References

Achlioptas, D., and Jiang, P. 2015. Stochastic integration via error-correcting codes. In *UAI*.

Achlioptas, D.; Hammoudeh, Z.; and Theodoropoulos, P. 2018. Fast and flexible probabilistic model counting. In *SAT*.

Anderson, R.; Huchette, J.; Tjandraatmadja, C.; and Vielma, J. P. 2019. Strong mixed-integer programming formulations for trained neural networks. In *IPCO*.

Arora, R.; Basu, A.; Mianjy, P.; and Mukherjee, A. 2018. Understanding deep neural networks with rectified linear units. In *ICLR*.

Balas, E., and Jeroslow, R. G. 1972. Canonical cuts on the unit hypercube. *SIAM J. Appl. Math.* 23:61–69.

Bartlett, P.; Maiorov, V.; and Meir, R. 1998. Almost linear VC-dimension bounds for piecewise polynomial networks. *Neural computation*.

Bastani, O.; Ioannou, Y.; Lampropoulos, L.; Vytiniotis, D.; Nori, A.; and Criminisi, A. 2016. Measuring neural net robustness with constraints. In *NeurIPS*.

Beise, H.-P.; Cruz, S. D. D.; and Schröder, U. 2018. On decision regions of narrow deep neural networks. *CoRR* abs/1807.01194.

Bengio, Y. 2009. Learning deep architectures for AI. *Foundations and Trends®in Machine Learning* 2(1):1–127.

Carter, J. L., and Wegman, M. N. 1979. Universal classes of hash functions. *J. Comput. Syst. Sci.* 18:143–154.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2013. A scalable approximate model counter. In *CP*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2014. Balancing scalability and uniformity in SAT witness generator. In *DAC*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2016. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *IJCAI*.

Charisopoulos, V., and Maragos, P. 2018. A tropical approach to neural networks with piecewise linear activations. *CoRR* abs/1805.08749.

Cheng, C.-H.; Nührenberg, G.; and Ruess, H. 2017. Maximum resilience of artificial neural networks. In *ATVA*.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *STOC*.

Croce, F.; Andriushchenko, M.; and Hein, M. 2019. Provable robustness of ReLU networks via maximization of linear regions. In *UAI*.

Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.* 2(4):303–314.

Danna, E.; Fenelon, M.; Gu, Z.; and Wunderling, R. 2007. Generating multiple solutions for mixed integer programming problems. In *IPCO*.

Dutta, S.; Jha, S.; Sankaranarayanan, S.; and Tiwari, A. 2018. Output range analysis for deep feedforward networks. In *NFM*.

Ehlers, R. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *ATVA*.

Elsayed, G. F.; Krishnan, D.; Mobahi, H.; Regan, K.; and Bengio, S. 2018. Large margin deep networks for classification. In *NeurIPS*.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013. Taming the curse of dimensionality:discrete integration by hashing and optimization. In *ICML*.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2014. Low-density parity constraints for hashing-based discrete integration. In *ICML*.

Fischetti, M., and Jo, J. 2018. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. In *CPAIOR*.

Glorot, X.; Bordes, A.; and Bengio, Y. 2011. Deep sparse rectifier neural networks. In *AISTATS*.

Gomes, C. P.; Hoeve, W.-J. V.; Sabharwal, A.; and Selman, B. 2007a. Counting CSP solutions using generalized XOR constraints. In *AAAI*.

Gomes, C. P.; Hoffmann, J.; Sabharwal, A.; and Selman, B. 2007b. Short XORs for model counting: From theory to practice. In *SAT*.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006a. Model counting: A new strategy for obtaining good bounds. In *AAAI*.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006b. Near-uniform sampling of combinatorial spaces using XOR constraints. In *NeurIPS*.

Guang-He Lee, David Alvarez-Melis, T. S. J. 2019. Towards robust, locally linear deep networks. In *ICLR*.

Hahnloser, R. H. R.; Sarpeshkar, R.; Mahowald, M. A.; Douglas, R. J.; and Seung, H. S. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* 405.

Hanin, B., and Rolnick, D. 2019a. Complexity of Linear Regions in Deep Networks. In *ICML*.

Hanin, B., and Rolnick, D. 2019b. Deep ReLU Networks Have Surprisingly Few Activation Patterns. In *NeurIPS*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

Huchette, J. 2018. *Advanced mixed-integer programming formulations: Methodology, computation, and application*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Ivrii, A.; Malik, S.; Meel, K. S.; and Vardi, M. Y. 2016. On computing minimal independent support and its applications to sampling and counting. *Constraints* 21:41–58.

Jain, S.; Kadioglu, S.; and Sellmann, M. 2010. Upper bounds on the number of solutions of binary integer programs. In *CPAIOR*.

Kumar, A.; Serra, T.; and Ramalingam, S. 2019. Equivalent and approximate transformations of deep neural networks. *CoRR* abs/1905.11428.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.

Lin, H., and Jegelka, S. 2018. ResNet with one-neuron hidden layers is a universal approximator. In *NeurIPS*.

Lomuscio, A., and Maganti, L. 2017. An approach to reachability analysis for feed-forward ReLU neural networks. *CoRR* abs/1706.07351.

Montúfar, G.; Pascanu, R.; Cho, K.; and Bengio, Y. 2014. On the number of linear regions of deep neural networks. In *NeurIPS*.

Montúfar, G. 2017. Notes on the number of linear regions of deep neural networks. In *SampTA*.

Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted Boltzmann machines. In *ICML*.

Nemhauser, G. L.; Wolsey, L. A.; and Fisher, M. L. 1978. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming* 14:265–294.

Nguyen, Q.; Mukkamala, M. C.; and Hein, M. 2018. Neural networks should be wide enough to learn disconnected decision regions. *CoRR* abs/1803.00094.

Pascanu, R.; Montúfar, G.; and Bengio, Y. 2014. On the number of response regions of deep feed forward networks with piece-wise linear activations. In *ICLR*.

Raghu, M.; Poole, B.; Kleinberg, J.; Ganguli, S.; and Sohl-Dickstein, J. 2017. On the expressive power of deep neural networks. In *ICML*.

Serra, T.; Tjandraatmadja, C.; and Ramalingam, S. 2018. Bounding and counting linear regions of deep neural networks. In *ICML*.

Sipser, M. 1983. A complexity theoretic approach to randomness. In *STOC*.

Stockmeyer, L. 1985. On approximation algorithms for #P. *SIAM Journal on Computing* 14(4):849–861.

Tjeng, V.; Xiao, K.; and Tedrake, R. 2019. Evaluating robustness of neural networks with mixed integer programming. In *ICRL*.

Toda, S. 1989. On the computational power of PP and (+)P. In *FOCS*.

Valiant, L., and Vazirani, V. 1986. NP is as easy as detecting unique solutions. *Theoretical Computer Science* 47:85–93.

Wong, E., and Kolter, J. Z. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*.

Xiao, K. Y.; Tjeng, V.; Shafiullah, N. M.; and Madry, A. 2018. Training for faster adversarial robustness verification via inducing ReLU stability. *CoRR* abs/1809.03008.

Zaslavsky, T. 1975. *Facing up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes*. AMS.

Zhang, L.; Naitzat, G.; and Lim, L.-H. 2018. Tropical geometry of deep neural networks. In *ICML*.

Zhao, S.; Chaturapruek, S.; Sabharwal, A.; and Ermon, S. 2016. Closing the gap between short and long XORs for model counting. In *AAAI*.