Unboundedness of Linear Regions of Deep ReLU Neural Networks

A. Ponomarchuk, C. Koutschan, B. Moser

RICAM-Report 2022-09

Unboundedness of Linear Regions of Deep ReLU Neural Networks[⋆]

Anton Ponomarchuk[1], Christoph Koutschan[1], and Bernhard Moser[2]

[1] Johann Radon Institute for Computational and Applied Mathematics (RICAM), ÖAW, Austria
[2] Software Competence Center Hagenberg (SCCH), Austria

Abstract. Recent work concerning adversarial attacks on ReLU neural networks has shown that unbounded regions and regions with a sufficiently large volume can be prone to containing adversarial samples. Finding the representation of linear regions and identifying their properties are challenging tasks. In practice, one works with deep neural networks and high-dimensional input data that leads to polytopes represented by an extensive number of inequalities, and hence demanding high computational resources. The approach should be scalable, feasible and numerically stable. We discuss an algorithm that finds the H-representation of each region of a neural network and identifies if the region is bounded or not.

Keywords: Neural network, unbounded polytope, linear programming, ReLU activation function

1 Introduction

In recent years, neural networks have become the dominant approach to solving tasks in domains like speech recognition, object detection, image generation, and classification. Despite their high prediction performance, there still exist undesirable network properties that are not fully understood. We investigate phenomena related to unbounded regions in the network's input space: the network should not provide high confidence predictions for data far away from training data. Concrete examples show how unbounded regions can be used to produce fooling images or out-of-distribution images that lead to misclassification of the network [4,7,8,10,13]. Robustness against such adversarial attacks is of utmost importance in critical applications like autonomous driving or medical diagnosis. There are still open questions concerning the correct processing and the meaning of unbounded linear regions for neural network accuracy.

A neural network F can be viewed as a function that maps an input vector $\mathbf{x} \in \mathbb{R}^{n_0}$ to an output vector in $\mathbb{R}^{n_L}$, by propagating it through the L hidden layers of the network. Each layer performs an affine-linear transformation, followed by a nonlinear activation function. Here we use the ReLU-activation function $\sigma(x) := \max(0, x)$, and therefore $F \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ is a continuous piecewise linear function. It splits the input space $\mathbb{R}^{n_0}$

[⋆] The research reported in this paper has been partly funded by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI.

into a finite set of linear regions (polytopes), on each of which the function F is linear, i.e., it can be described as $\mathbf{x} \mapsto \mathbf{A} \cdot \mathbf{x} + \mathbf{b}$ for some $A \in \mathbb{R}^{n_L \times n_0}$, $\mathbf{b} \in \mathbb{R}^{n_L}$.

It has been shown [6] that neural networks with ReLU activation function and softmax as output (the network is then called a classifier) achieve overconfident predictions in all unbounded linear regions. Moreover, it is not clear what impact unbounded regions have on the neural network's calibration [12]. It is beneficial to know that the trained model provides accurate predictions and the ones that describe the confidence in the output class correctness. This leads us to the task of deciding which linear regions are bounded and which are unbounded. For the bounded regions the next question is if their volume is sufficiently large to contain adversarial examples. Determining the volume of a convex polytope is a polynomial problem. It can be done either by triangulation [1] or by volume approximation using Monte Carlo sampling or random walk methods [3,11]. The recent approximation approaches achieve the complexity of $\mathcal{O}(Nn_0^4)$ steps, where n_0 is the dimension of the input space and N is the number of inequalities defining the given polytope. As a result, it is challenging to apply them in practice for high-dimensional input spaces and deep neural networks.

We discuss an algorithm for checking the boundedness of a polytope $\mathbf{H} \subset \mathbb{R}^{n_0}$. In order to do so, we first revise the algorithm from [16] that for any point $\mathbf{x} \in \mathbb{R}^{n_0}$ calculates a maximal linear region $\mathbf{H}$ such that $\mathbf{x} \in \mathbf{H}$. Then by using the H-representation of the polytope $\mathbf{H}$ we provide an algorithm to check whether $\mathbf{H}$ is bounded or not.

2 Preliminaries

A function $F \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ defined by a neural network propagates an input vector $\mathbf{x} \in \mathbb{R}^{n_0}$ through L hidden layers to some output vector in $\mathbb{R}^{n_L}$, where n_i denotes the number of neurons in the i-th layer. More precisely, the i-th hidden layer, $i \in \{1, \ldots, L\}$, performs the mapping $\mathbf{a}_i$, which is described by the following composition of functions:

$$\mathbf{a}_i(\mathbf{x}) \coloneqq \sigma \circ f_i(\mathbf{x}),$$

where $f_i \colon \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ is an affine mapping and $\sigma \colon \mathbb{R} \to \mathbb{R}$ is a non-linear function that acts componentwise on vectors. Each affine mapping f_i is represented by a linear function $f_i(\mathbf{x}) \coloneqq \mathbf{A}_i\mathbf{x} + \mathbf{b}_i$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and $\mathbf{b}_i \in \mathbb{R}^{n_i}$, and where the non-linear part is the ReLU activation function $\sigma(x) \coloneqq \max(x, 0)$. Hence, a neural network $F \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ has the form

$$F(\mathbf{x}) \coloneqq f_L \circ \sigma \circ f_{L-1} \circ \ldots \circ \sigma \circ f_1(\mathbf{x}).$$

Since $F(\mathbf{x})$ is a composition of affine and ReLU activation functions, it follows that it is piecewise linear. This property of $F(\mathbf{x})$ implies that the input space $\mathbb{R}^{n_0}$ is split into a finite set of linear regions $\{\mathbf{H}_i\}_{i=1}^J$, such that the function $F(\mathbf{x})$ is linear in each $\mathbf{H}_i$ and $\mathbb{R}^{n_0} = \bigcup_{i=1}^J \mathbf{H}_i$. It can been shown [16] that all the regions are polytopes and for each one its so-called H-representation can be determined: every polytope $\mathbf{H} \subseteq \mathbb{R}^{n_0}$ can be represented as a finite intersection of halfspaces, i.e., as the set of all points that satisfy a finite list of linear inequalities:

$$\mathbf{H} \coloneqq \left\{\mathbf{x} \in \mathbb{R}^{n_0} \mid \mathbf{W}\mathbf{x} \leq \mathbf{v}\right\},$$

where $\mathbf{W} \in \mathbb{R}^{N \times n_0}$, $\mathbf{v} \in \mathbb{R}^N$, and $N = n_1 + \cdots + n_L$ denotes the total number of neurons of the network F. Moreover, an algorithm was proposed in [16] that computes for a given point $\mathbf{x} \in \mathbb{R}^{n_0}$ the corresponding polytope $\mathbf{H}(\mathbf{x}) \subseteq \mathbb{R}^{n_0}$ (see Appendix A).

3 Representation and analysis via code space

With the preparations introduced above, we can now present an algorithm that checks whether the polytope $\mathbf{H}(\mathbf{x})$ is bounded or not, for a given point $\mathbf{x} \in \mathbb{R}^{n_0}$. Without loss of generality, we can assume that the H-representation of $\mathbf{H}(\mathbf{x})$ does not have any duplicated inequalities, i.e., the augmented matrix $(\mathbf{W}|\mathbf{v}) \in \mathbb{R}^{N \times (n_0+1)}$ does not have any repeated rows. Let us denote by $\ker(\mathbf{W})$ the null space of the matrix $\mathbf{W}$. The following lemma enables us to check the boundedness of $\mathbf{H}(\mathbf{x})$.

Lemma 1. *A non-empty polytope* $\mathbf{H} = \{\mathbf{x} \in \mathbb{R}^{n_0} \mid \mathbf{W}\mathbf{x} \leq \mathbf{b}\}$ *is bounded if and only if* $\ker(\mathbf{W}) = \{\mathbf{0}\}$ *and the following linear program admits a feasible solution:*

$$\min \|\mathbf{y}\|_1 \quad \textit{subject to} \quad \mathbf{W}^T\mathbf{y} = \mathbf{0} \textit{ and } \mathbf{y} \geq \mathbf{1}. \tag{1}$$

Proof. ("$\Rightarrow$") Assume, that $\mathbf{H}$ is bounded and non-empty. Then it follows that the null space of $\mathbf{W}$ is trivial, because otherwise, there would exist a non-zero vector $\mathbf{v} \in \ker(\mathbf{W})$ such that for all scalars $\lambda \in \mathbb{R}$ and any point $\mathbf{x} \in \mathbf{H}$ the following holds:

$$\mathbf{W}(\mathbf{x} + \lambda\mathbf{v}) \leq \mathbf{b} \;\Rightarrow\; \mathbf{x} + \lambda\mathbf{v} \in \mathbf{H}.$$

As a result, the polytope $\mathbf{H}$ is unbounded, contradicting our assumption.

Now, let us assume that the linear program (1) does not have a solution. It can only be the case when the set of feasible solutions that is described by the restrictions $\mathbf{W}^T\mathbf{y} = \mathbf{0}$ and $\mathbf{y} \geq \mathbf{1}$ is empty. By Stiemke's Lemma, see Appendix C, it follows that there exists a vector $\mathbf{v} \in \mathbb{R}^{n_0}$ such that $\mathbf{W}\mathbf{v} > \mathbf{0}$. Thereby, for all vectors $\mathbf{x} \in \mathbf{H}$ and for all non-positive scalars $\lambda \in \mathbb{R}$ we have:

$$\mathbf{W}(\mathbf{x} + \lambda\mathbf{v}) = \mathbf{W}\mathbf{x} + \lambda\mathbf{W}\mathbf{v} \leq \mathbf{b} \;\Rightarrow\; \mathbf{x} + \lambda\mathbf{v} \in \mathbf{H}.$$

As a result, the polytope $\mathbf{H}$ is unbounded, that is contradiction. Thus, if the given polytope $\mathbf{H}$ is bounded then the given linear program has a solution and the null space of the matrix $\mathbf{W}$ is trivial.

("$\Leftarrow$") Assume that the null space of the matrix $\mathbf{W}$ is trivial and the linear program (1) has a solution. If the given (non-empty) polytope $\mathbf{H}$ was unbounded, then for all vectors $\mathbf{x} \in \mathbf{H}$ there would exist a nonzero vector $\mathbf{v} \in \mathbb{R}^{n_0}$, such that for all non-negative $\lambda \in \mathbb{R}$ holds $\mathbf{x} + \lambda\mathbf{v} \in \mathbf{H}$. It follows that

$$\lambda\mathbf{W}\mathbf{v} \leq \mathbf{b} - \mathbf{W}\mathbf{x}, \tag{2}$$

where $\mathbf{b} - \mathbf{W}\mathbf{x} \geq \mathbf{0}$. If at least one of the entries of $\mathbf{W}\mathbf{v}$ is positive, then there exists $\lambda' > 0$ such that the inequality (2) breaks. Thus $\mathbf{W}\mathbf{v} \leq \mathbf{0}$, and Stiemke's Lemma, together with the trivial null space of $\mathbf{W}$, leads to a contradiction. $\square$

Concerning the complexity, note that in the worst case, it is necessary to compute the null space of the matrix $\mathbf{W}$ and to solve the linear program described in Lemma 1. The basic algorithm to solve the linear programming problem is an interior-point algorithm [15]. It has the worst-case complexity $\mathcal{O}(\sqrt{n_0}L)$, where L is the length of the binary encoding of the input data:

$$L \coloneqq \sum_{i=0}^{N} \sum_{j=0}^{n_0} \lceil \log_2(|w_{ij}| + 1) + 1 \rceil,$$

where $w_{i0} = b_i$ and $w_{0j} = 1$. The complexity of the solver depends on the input dimension n_0 and the number N of inequalities. The HiGHS [9] implementation of the interior-point method finds an optimal solution even faster in practice.

On the other hand, the complexity for computing the null space of the matrix $\mathbf{W} \in \mathbb{R}^{n_0 \times N}$ is at most $\mathcal{O}(n_0 N \min(n_0, N))$. Hence, the total-worst case time complexity for the given approach is at most $\mathcal{O}(n_0 N \min(n_0, N) + \sqrt{n_0}L)$. In practice, modern implementations interior-point algorithms, like HiGHS, use techniques for speeding up computation, for instance, parallelization.

We implemented Lemma 1 and the polytope computation algorithm, see Appendix A, in Python with libraries for scientific and deep learning computing: SciPy [18], NumPy [5], PyTorch [14]. For time execution measurements we used the basic Python library timeit. For creating the graphics we used the matplotlib library [2].

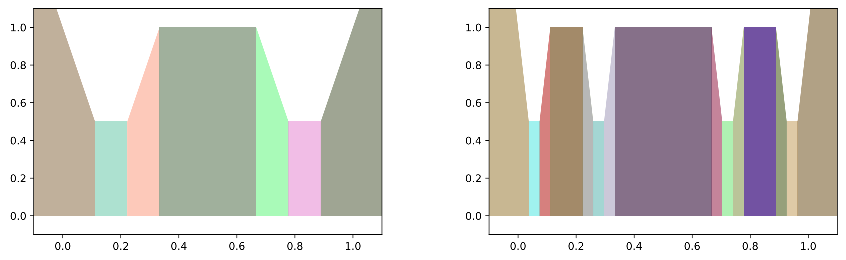

Fig. 1. Decision surfaces splitted into polytopes for depths $L = 2, 3$. In each polytope a neural network is linear. The leftmost and rightmost regions are unbounded.

4 Experiments

In this section we are going to discuss some preliminary experiments that we carried out with our implementation.

The first experiment is based on a neural network that is defined recursively. On the unit interval $[0, 1]$ we define the function $F(x) \coloneqq \max\{-3x + 1, 0, 3x - 2\}$. Its nested

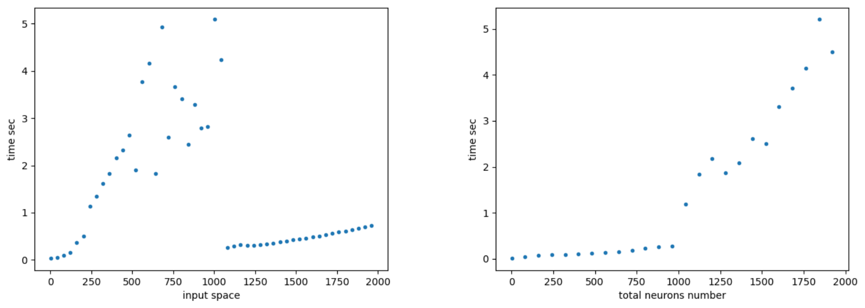

Fig. 2. Time needed for checking whether the calculated polytope is bounded. The left plot shows timings for different input spaces and a fixed neural network with $N = 1074$ neurons. In the right plot, the input space is fixed ($n_0 = 1024$) and the total number of neurons varies.

composition is the following: $F^{l+1}(x) := F\big(F^l(x)\big)$ and $F^1(x) := F(x)$, see Fig. 1. The decision boundary is defined as the upper border of the regions:

$$\mathbf{H}_L := \big\{(x,y) \in [0,1]^2 \bigm| y \leq \tfrac{1}{2}(F_L(x)+1)\big\}.$$

The network $F^L(x)$ is used to test the algorithm. The reason is that this network could generate any number of explicit linear regions. The greater recursion L, the greater is the number of linear regions the network generates. While experimenting with this network, we found several problems that could occur during the execution of the algorithm. Firstly, after computing the H-representation of a polytope, all identical rows should be removed from the corresponding matrix $\mathbf{W}$. If the matrix contains identical rows, it can provide incorrect results. Secondly, not all linear problem solvers can handle problems with floating point numbers in practice. For example, the basic interior-point linear program solver in SciPy failed to solve such problems due to numerical problems. On the other hand, the HiGHS implementation succeeded.

The second experiment evaluates the time dependency on the dimension n_0 of the input space and the number N of neurons. For this experiment, fully-connected neural networks were generated with random weights. Suppose that the number of neurons is smaller than the dimension of the input space $N < n_0$. In this case, any region $\mathbf{H}$ is unbounded because the corresponding matrix $\mathbf{W}$ will have a non-trivial null space. As a result, there is no need for solving the linear program task in such a situation. Also, one can see the "drop" in the time measurements in the left plot of Fig. 2 because of it. Otherwise, if $n_0 \leq N$, then there are cases where one needs to compute the null space and the linear program for the given $\mathbf{W}$. As a result, in the right plot, the jump at $N = 1024$ is explained because not all null spaces are non-trivial beyond this point.

5 Outlook and future work

The algorithms presented in the paper are the first step in further understanding the properties of the linear regions that correspond to a neural network. There is an open question of detecting linear regions prone to containing adversarial examples. Is there a relation between the region's volume and this problem? What should one do to bypass the problem in such networks? Also, the same questions relate to unbounded regions.

Furthermore, there is an open question of how geometric knowledge about the tessellation of the input space can help us create better validation and test sets? Using these algorithms, one can work with polytopes that correspond to training and validation points and compare them.

References

1. Büeler, B., Enge, A., Fukuda, K.: Exact volume computation for polytopes: a practical study. In: Polytopes—combinatorics and computation. pp. 131–154. Springer (2000)
2. Caswell, T.A., Droettboom, M., Lee, A., de Andrade, E.S., Hoffmann, T., Hunter, J., Klymak, J., Firing, E., Stansby, D., Varoquaux, N., Nielsen, J.H., Root, B., May, R., Elson, P., Seppänen, J.K., Dale, D., Lee, J.J., McDougall, D., Straw, A., Hobson, P., hannah, Gohlke, C., Vincent, A.F., Yu, T.S., Ma, E., Silvester, S., Moad, C., Kniazev, N., Ernest, E., Ivanov, P.: matplotlib/matplotlib: Rel: v3.5.1 (Dec 2021)
3. Emiris, I.Z., Fisikopoulos, V.: Efficient random-walk methods for approximating polytope volume. In: Proceedings of the thirtieth annual symposium on computational geometry. pp. 318–327 (2014)
4. Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. In: International Conference on Machine Learning. pp. 1321–1330. PMLR (2017)
5. Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature **585**(7825), 357–362 (Sep 2020)
6. Hein, M., Andriushchenko, M., Bitterwolf, J.: Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 41–50 (2019)
7. Hendrycks, D., Gimpel, K.: A baseline for detecting misclassified and out-of-distribution examples in neural networks. arXiv preprint arXiv:1610.02136 (2016)
8. Hsu, Y.C., Shen, Y., Jin, H., Kira, Z.: Generalized ODIN: Detecting out-of-distribution image without learning from out-of-distribution data. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
9. Huangfu, Q., Hall, J.J.: Parallelizing the dual revised simplex method. Mathematical Programming Computation **10**(1), 119–142 (2018)
10. Leibig, C., Allken, V., Ayhan, M.S., Berens, P., Wahl, S.: Leveraging uncertainty information from deep neural networks for disease detection. Scientific reports **7**(1), 1–14 (2017)
11. Mangoubi, O., Vishnoi, N.K.: Faster polytope rounding, sampling, and volume computation via a sub-linear ball walk. In: 2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS). pp. 1338–1357. IEEE (2019)
12. Minderer, M., Djolonga, J., Romijnders, R., Hubis, F., Zhai, X., Houlsby, N., Tran, D., Lucic, M.: Revisiting the calibration of modern neural networks. Advances in Neural Information Processing Systems **34** (2021)
13. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)

14. Paszke, A., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019)
15. Potra, F.A., Wright, S.J.: Interior-point methods. Journal of computational and applied mathematics **124**(1-2), 281–302 (2000)
16. Shepeleva, N., Zellinger, W., Lewandowski, M., Moser, B.: ReLU code space: A basis for rating network quality besides accuracy. ICLR 2020 Workshop on Neural Architecture Search (NAS 2020) (2020)
17. Stiemke, E.: Über positive Lösungen homogener linearer Gleichungen. Mathematische Annalen **76**(2), 340–342 (1915)
18. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al.: Scipy 1.0: fundamental algorithms for scientific computing in python. Nature methods **17**(3), 261–272 (2020)

Appendix A: Polytope calculation for an input point $\mathbf{x}$

Let us remind that the ReLU neural network $F(\mathbf{x}) = f_L \circ \sigma \circ f_{L-1} \circ \ldots \circ \sigma \circ f_1(\mathbf{x})$ is a composition of L affine functions $f_i(\mathbf{x}) = \mathbf{A}_i\mathbf{x} + \mathbf{b}_i$, where $\mathbf{A}_i \in \mathbb{R}^{n_i \times n_{i-1}}$ and $\mathbf{b}_i \in \mathbb{R}^{n_i}$ for all $i \in \{1, \ldots, L\}$, with a point-wise non-linear function $\sigma(x) = \max(x, 0)$. We denote the i-th hidden layer of the network $F(\mathbf{x})$ by $\mathbf{a}_i(\mathbf{x}) = \sigma \circ f_i(\mathbf{x})$.

A **binary activation state** for an input vector $\mathbf{x} \in \mathbb{R}^{n_0}$ is the function

$$\beta_k^{i_k}(\mathbf{x}) := \begin{cases} 1, & a_k^{i_k}(\mathbf{x}) > 0, \\ 0, & \text{otherwise}, \end{cases}$$

where $a_k^{i_k}(\mathbf{x})$ is the i_k-th output of the k-th hidden layer $\mathbf{a}_k$, for all $k \in \{1, \ldots, L\}$ and $i_k \in \{1, \ldots, n_k\}$.

A **polar activation state** for an input vector $\mathbf{x} \in \mathbb{R}^{n_0}$ is the function

$$\pi_k^{i_k}(\mathbf{x}) := 2\beta_k^{i_k}(\mathbf{x}) - 1,$$

for all $k \in \{1, \ldots, L\}$ and $i_k \in \{1, \ldots, n_k\}$. Note that we defined two binary functions which have the sets $\{0, 1\}$ and $\{-1, 1\}$ as codomains, respectively. By using $\beta_k^{i_k}(\mathbf{x})$ and $\pi_k^{i_k}(\mathbf{x})$, we now collect all states of a layer into a diagonal matrix form:

$$\mathbf{Q}_k^{\pi}(\mathbf{x}) := \operatorname{diag}\big(\pi_k^1(\mathbf{x}), \ldots, \pi_k^{n_k}(\mathbf{x})\big),$$
$$\mathbf{Q}_k^{\beta}(\mathbf{x}) := \operatorname{diag}\big(\beta_k^1(\mathbf{x}), \ldots, \beta_k^{n_k}(\mathbf{x})\big),$$

where $k \in \{1, \ldots, L\}$. We will use the matrix $\mathbf{Q}_k^{\beta}(\mathbf{x})$ to model the behavior of the activation function in the k-th layer. For each input vector $\mathbf{x} \in \mathbb{R}^{n_0}$, the matrices $\mathbf{Q}_k^{\pi}(\mathbf{x})$ and $\mathbf{Q}_k^{\beta}(\mathbf{x})$ allow us to derive an H-representation of the corresponding polytope $\mathbf{H}(\mathbf{x}) \in \{\mathbf{H}_i\}_{i=1}^{J}$ in explicit form. More precisely, the H-representation is given as a set of inequalities in the following way:

$$\mathbf{H}(\mathbf{x}) := \big\{\mathbf{x}' \in \mathbb{R}^{n_0} \mid \mathbf{W}_k(\mathbf{x}) \cdot \mathbf{x}' + \mathbf{v}_k(\mathbf{x}) \geq \mathbf{0},\ k \in \{1, \ldots, L\}\big\}, \tag{3}$$

where

$$\mathbf{W}_k(\mathbf{x}) \coloneqq \mathbf{Q}_k^{\pi}(\mathbf{x})\mathbf{A}_k \prod_{j=1}^{k=1} \mathbf{Q}_{k-j}^{\beta}(\mathbf{x})\mathbf{A}_{k-j}, \tag{4}$$

$$\mathbf{v}_k(\mathbf{x}) \coloneqq \mathbf{Q}_k^{\pi}(\mathbf{x}) \sum_{i=1}^{k} \left(\prod_{j=1}^{k-i} \mathbf{A}_{k-j+1}\mathbf{Q}_{k-j}^{\beta}(\mathbf{x}) \right) \mathbf{b}_i, \tag{5}$$

such that $\mathbf{W}_k(\mathbf{x}) \in \mathbb{R}^{n_k \times n_0}$ and $\mathbf{v}_k(\mathbf{x}) \in \mathbb{R}^{n_k}$. According to (3), the polytope $\mathbf{H}(\mathbf{x})$ is defined by exactly $N = n_1 + \ldots + n_L$ inequalities. However, in practice, the number of half-spaces whose intersection yields the polytope $\mathbf{H}(\mathbf{x})$ is typically smaller than N, so that the above representation is not minimal in general.

Appendix B: Unbounded linear region problem

As mentioned in Appendix A, a ReLU neural network F splits the input space $\mathbb{R}^{n_0}$ into a set of linear regions $\mathbb{R}^{n_0} = \bigcup_{i=1}^{J} \mathbf{H}_i$. On each such linear region the network realizes some affine function

$$F_{\mathbf{H}_i}(\mathbf{x}) \coloneqq \mathbf{A}_i\mathbf{x} + \mathbf{b}_i,$$

where $\mathbf{A}_i \in \mathbb{R}^{n_L \times n_0}$, $\mathbf{b} \in \mathbb{R}^{n_L}$, $\mathbf{x}_i \in \mathbf{H}_i$ and $i \in \{1, \ldots, J\}$. So the given network F is represented by a set of affine functions $F_{\mathbf{H}_i}$, each of which corresponds to some linear region $\mathbf{H}_i$ for all $i \in \{1, \ldots, J\}$.

Assume that $\mathbf{A}_i$ does not contain identical rows for all $i \in \{1, \ldots, J\}$, then for almost all $\mathbf{x} \in \mathbb{R}^{n_0}$ and $\varepsilon > 0$, there exists an $\alpha > 0$ and a class $k \in \{1, \ldots, n_L\}$ such that for $\mathbf{z} = \alpha\mathbf{x}$ the following holds:

$$\frac{\exp(F_k(\mathbf{z}))}{\sum_{j=1}^{n_L} \exp(F_j(\mathbf{z}))} \geq 1 - \varepsilon. \tag{6}$$

Inequality (6) shows that almost any point from the input space $\mathbb{R}^{n_0}$ can be scaled such that the transformed input value will get an overconfident output for some class $k \in \{1, \ldots, n_L\}$. See reference [6] for a proof of the above statement.

Appendix C: Stiemke's Lemma

Stiemke's Lemma states the following:

Lemma 2. *Let $\mathbf{W} \in \mathbb{R}^{m \times n}$ and $\mathbf{W}\mathbf{x} = 0$ be a homogeneous system of linear inequalities. Then only one of the following is true:*

1. *There exists a vector $\mathbf{v} \in \mathbb{R}^m$ such that the vector $\mathbf{W}^T\mathbf{v} \geq 0$ with at least one non-zero element.*
2. *There exists a vector $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{x} > 0$ and $\mathbf{W}\mathbf{x} = 0$.*

This lemma has variants with different sign constraints that can be found in [17].