Property Inference in ReLU nets using Linear Interpolants

Divyansh Pareek[*], Saket Dingliwal[*], and Jatin Arora[*]

dpareek@cs.cmu.edu, sdingliw@cs.cmu.edu, jatina@cs.cmu.edu

Abstract

Deep Neural Networks have emerged as powerful tools and are being increasingly deployed in real-world safety critical domains. Despite their widespread success, their complex architecture makes proving any formal guarantees about them difficult. In this paper, we introduce a way to bridge the gap between the architecture of the network and the high level properties on its output. Our key insight is that instead of directly proving the safety properties that are required, we first prove intermediate properties that relate closely to the structure of the neural net and use them to reason about the safety properties. We build theoretical foundations for our approach, and provide empirical evidence of its advantages.

1 Introduction

There has been an ever increasing trend of using Deep Neural Networks for vast variety of machine learning applications [12]. Their widespread success can be attributed to their complex architecture that makes them powerful function approximators, however this also makes proving formal guarantees about them difficult [11]; which is crucial for their use in many real world safety critical systems [2]. Present-day networks suffer from a range of problems, including lack of explainability [6] and susceptibility to adversarial attacks [13]. These caveats highlight the need of formal guarantees on the correctness on these networks, especially in safety/security sensitive areas like autonomous driving, health-care and banking. Identifying how logical notions of high level correctness relate to the complex low-level network architecture is a significant challenge [9]; but one that makes advances towards actually understanding the input-output relation of networks, and subsequently deriving provable guarantees on them.

One such type of guarantee that takes a high-level correctness property and relates it to the network architecture and weights, is as follows: *Given a property P on the output of the neural network, find a region R (as large as possible) in the input space for which that property is guaranteed to hold.* Identifying such 'confidence'-regions is useful in motion planning and collision-prediction networks, since absence of certain physically possible trajectories in the guaranteed region helps to identify potential collision courses. Furthermore, the property P could be designed such that it expresses undesired outputs. This will help in identifying the regions of input space in which the network is making an error (and therefore requires more data for these identified parts of the input space).

This problem has been studied via the lens of *activation patterns* in [7]. Activation patterns are a natural tool for this because they describe how a ReLU network breaks up the input space into piecewise-linear components. The idea of their approach is to start with the activation pattern of a point in the input space that satisfies the property P, and then repeatedly relax the pattern by eliminating constraints that are non-essential for property P; enlarging the region captured. Our main insight, motivated by interpolants [10], is that instead of directly proving

[*]Equal Contributions

the safety properties that are required, we first prove intermediate properties that relate closely to the structure of the neural net and use them to reason about the safety properties.

We flesh out the theory and build the methodology for our approach, and empirically evaluate the performance through a set of exploratory experiments, achieving promising results. By identifying a larger region of the input space that guarantees the property on the output; we outperform the existing approach, validating the foundations of our work and encouraging large-scale experiments.

2 Background

Let $l \in \{0, 1, \cdots L\} = [0, L]$ denote layer numbers in the network; where $l = 0$ is input layer, $l \in [1, L-1]$ are hidden layers, and $l = L$ is the output layer. ReLU activation is applied only on the hidden layers. Let N_l denote the number of neurons in layer l, for $l \in [0, L]$. Denote $\mathcal{X} := \mathbb{R}^{N_0} \equiv$ the input space. And $\mathcal{Y} := \mathbb{R}^{N_L} \equiv$ the output/logit space. Let $F : \mathcal{X} \to \mathcal{Y}$ denote the *trained/static* ReLU network in consideration. Then, our goal is : we want to identify a region R (as large as possible) on the input space, such that the property P is guaranteed to hold on the output space. That is, $R \subseteq \mathcal{X}$ such that $X \in R \implies P(F(X))$.

Note that the region R will be characterized by a logic formula on the input space $\mathcal{X}$ which will be satisfied only by points belonging to R. Here $P(Y)$ is a *boolean* property and the argument $Y \in \mathcal{Y} = \mathbb{R}^{N_L}$ denotes a placeholder in the output space. The most commonly sought-after property is 'class-guarantee' in classification networks, which is what we'll focus on. This property guarantees that the output of the network is a certain class. For instance, $P(Y) = (Y_0 > Y_1) \wedge (Y_0 > Y_2) \wedge \cdots \wedge (Y_0 > Y_{N_L - 1})$ guarantees that the output is class-0. Note that this is property is linear in its argument Y (placeholder for the network's output).

Before briefly describing the existing approach, we review relevant notation and properties about activation patterns. A ReLU neuron is considered off when it's output is zero and on when it's output is positive. An activation pattern (denoted σ) specifies an activation status $\{\mathsf{on}/\mathsf{off}/\mathsf{dc}\}$ for *all* ReLU neurons. dc denotes "don't-care". A dc neuron is unconstrained (can be on or off). Note that an activation pattern σ holds meaning only in the context of the network F we're analyzing. So, the dependence on F will be implicit in our discussion.

For any input $X \in \mathcal{X}$, let σ_X denote the activation pattern dictated by the forward pass of X on the network F. Note that the notation σ_X refers to a particular activation pattern with X as the input to the network, while the notation $\sigma(X)$ refers to a formula on the input space with X as the variable/placeholder. Let tuple (l, i) refer to a neuron of the network in layer $l \in [1, L]$ and numbered $i \in [0, N_l)$. Let $A_{(l,i)}(X)$ denote the feed-in (before applying ReLU) to this neuron on applying input $X \in \mathcal{X}$ to the network. For a pattern σ, let $\sigma(l, i) \in \{\mathsf{on}/\mathsf{off}/\mathsf{dc}\}$ denote what the pattern σ dictates on neuron (l, i). Now we can define a formula as

$$\sigma(X) := \left(\bigwedge_{\sigma(l,i)=\mathsf{on}} A_{(l,i)}(X) > 0 \right) \bigwedge \left(\bigwedge_{\sigma(l,i)=\mathsf{off}} A_{(l,i)}(X) \leq 0 \right)$$

This formula exactly characterizes all points $X \in \mathcal{X}$ such that σ_X agrees with σ. This means that $\forall (l, i),\ \ \sigma(l, i) \neq \mathsf{dc} \implies \sigma(l, i) = \sigma_X(l, i)$. Basically, all neurons that are not "don't-care" in σ must have the same signature in σ_X also. Denote $\mathsf{support}(\sigma) = \{X \in \mathcal{X} : \sigma(X) \text{ is true}\}$. We say that $\sigma \implies P$, if $\forall X \in \mathcal{X},\ \sigma(X) \implies P(F(X))$. Consider a function $\mathsf{checkSAT}(\sigma, P)$ that returns the truth value of $\sigma \implies P$. We say that σ is minimal w.r.to property P when $\sigma \implies P$ and unconstraining (converting to dc) any neuron in σ invalidates P. Minimality helps in getting rid of unnecessary constraints, and ensuring that more input points can satisfy the property, by covering a larger region in the input space. Consider a function $\mathsf{FindMinimal}(\sigma, P)$ which returns a heuristically found minimal σ' such that $(\sigma \implies P) \implies (\sigma' \implies P)$.

A pattern σ is said to be *complete* if $\forall(l,i), \sigma(l,i) \in \{\mathsf{on}, \mathsf{off}\}$. That is, there are no "don't-cares". Note that σ_X (activation pattern from forward pass of X on F) is complete. It can be easily argued ([7] shows this formally) that for a complete pattern σ, the output of the network is an affine function of the input for all support points. That is, there exist $W \in \mathbb{R}^{N_L \times N_0}, b \in \mathbb{R}^{N_L}$ such that $\forall X \in \mathsf{support}(\sigma), F(X) = W \cdot X + b$. Consider a function $\mathsf{getWeights}(\sigma)$ which precisely returns these linear weights W, b corresponding the output of the network for a complete σ.

Now with this setup, we describe the approach of [7]. Call this procedure $\mathsf{getRegion}(P, X_0)$ that takes in the property P and a point $X_0 \in \mathcal{X}$ such that $P(F(X_0))$ is true. Initializing $\sigma \leftarrow \sigma_{X_0}$, the procedure calls $\mathsf{checkSAT}(\sigma, P)$. If true, it relaxes σ by calling $\mathsf{FindMinimal}(\sigma, P)$. Else if false, it returns the intersection of $\sigma(X)$ with $P(F(X)) = P(W \cdot X + b)$, since $F(X) = W \cdot X + b$ for all $X \in \mathsf{support}(\sigma)$. This is described below.

Algorithm 1: getRegion(P, X_0)

```
input  : Property $P$ and input $X_0$ such that $P(F(X_0))$ holds
output: Formula $Q$ on $\mathcal{X} : Q(X) \Rightarrow P(F(X))$
$\sigma \leftarrow \sigma_{X_0}$;
if checkSAT$(\sigma, P)$ then
    return FindMinimal$(\sigma, P)$
else
    $W, b$ = getWeights$(\sigma)$;
    return $\sigma(X) \wedge P(W.X + b)$;
end
```

The justification is as follows. When $\sigma \not\Longrightarrow P$, then we know there are points in $\mathsf{support}(\sigma)$ itself that don't satisfy P. So the procedure returns the intersection of $\sigma(X)$ and $P(F(X)) = P(W \cdot X + b)$, since $F(X) = W \cdot X + b$ for all $X \in \mathsf{support}(\sigma)$. On the other hand if $\sigma \implies P$, then we can call the $\mathsf{FindMinimal}(\sigma, P)$ procedure to 'relax' the activation pattern by removing constraints that don't affect P.

There are various heuristics to implement $\mathsf{FindMinimal}$. The most immediate strategy is a greedy one [7]: Beginning with a complete activation pattern, try to relax layers one by one starting from the last hidden layer. Once we reach a layer that cannot be relaxed as a whole, try and relax neurons in this layer individually. Note that our approach is generic and serves as an improvement over the baseline over any heuristic implementation of the function $\mathsf{FindMinimal}$.

3 Methodology

Now we explain a potential drawback in the existing approach and our solution to alleviate it. Note that here and throughout this section, we refer to W, b as the result of $\mathsf{getWeights}(\sigma)$ where σ is the complete activation pattern with which we start the minimalization procedure. In the procedure above, if the $\mathsf{checkSAT}(\sigma, P)$ call returns false, then the method returns $\sigma(X) \wedge P(W \cdot X + b)$. In this case no minimisation of σ takes place. Note that the property P is high level and does not consider the architecture of the neural network. There is no general intuition that the activation pattern corresponds to the property that we are trying to check. Hence, it is expected that this $\mathsf{checkSAT}(\sigma, P)$ call will often return false, and the entire method returns $\sigma(X) \wedge P(W \cdot X + b)$ (without any minimalization of the activation pattern). This observation is backed by our experiments, where this procedure enters the else branch frequently.

We propose that instead of finding the minimal activation pattern that implies $P(Y)$, we should find one that implies an intermediate property $I(X, Y)$. The condition on $I(X, Y)$ being that using I, we should be able to represent/check the property P. In this section consider the verification of the property $P(Y) = Y_0 > Y_1$ ('class-guarantee' in binary classification). We can simply extend the ideas presented here to multi-class classification by a logical-and of $Y_0 > Y_1$, $Y_0 > Y_2$ and so on.

As a motivating example, consider $I(X, Y) := \big(Y_0 - Y_1 = (W_0 \cdot X + b_0 - W_1 \cdot X - b_1)\big)$. Here W_0, b_0 and W_1, b_1 are the first and second rows of W, b; and they denote the linear relation of $F(X)_0$ and $F(X)_1$ respectively with the input X. We know that $\sigma \Rightarrow I$ (because $F(X) = W \cdot X + b$ for $X \in \mathsf{support}(\sigma)$). Therefore, the procedure $\mathsf{getRegion}$ when called with I will never go to the else branch. The algorithm then returns $\sigma' = \mathsf{FindMinimal}(\sigma, I)$. Now using the fact that $\sigma' \Rightarrow I$, we can express a convex formula that implies P as follows:

$$\mathcal{G}(X) := \sigma'(X) \wedge (W_0 \cdot X + b_0 - W_1 \cdot X - b_1 > 0)$$

Note that $\forall X \in \mathcal{X},\ \mathcal{G}(X) \Rightarrow P(F(X))$, which is crucial and ensures that the property P is checkable using the intermediate property I. This approach is similar to the concept of interpolants introduced in [10]. The idea there is that in order to prove $A \Rightarrow B$, we prove $A \Rightarrow C$ and $C \Rightarrow B$. This has been immensely successful in the context of model checking and verification of safety properties [1]. The benefit of this approach in this context is that we can leverage the low level architecture of the neural net to construct such I.

The above illustrates one choice of I. This however was a motivating example, and we can improve upon it to create another I that captures a larger region of the input space. The reason is that the above I is a strict equality on the weights and biases, and the differences $W_0 - W_1$, $b_0 - b_1$ are highly likely to change as one moves outside the initial activation pattern. To create the improved I, we update the equality to an inequality and introduce a free parameter ϵ while still ensuring that the original property is check-able using the intermediate property. This is done as follows: Let $dW = W_0 - W_1$ and $db = b_0 - b_1$. Consider $I(X, Y) := \big(Y_0 - Y_1 > dW \cdot X + db - \epsilon\big)$. Note that for this I as well $\sigma \Rightarrow I$ for any $\epsilon > 0$. Therefore, $\mathsf{getRegion}$ will return $\sigma' = \mathsf{FindMinimal}(\sigma, I)$. Now we can express the resultant formula as $\sigma'(X) \wedge (dW \cdot X + db \geq \epsilon)$, which implies the property $P(Y) = Y_0 > Y_1$; ensuring that P is checkable using I.

The intuition behind why this particular I is a good way to capture a larger input region is as follows. We know that for $X \in \mathsf{support}(\sigma)$, $Y_0 - Y_1 = dW \cdot X + db$. From the lottery ticket hypothesis [5], it is clear that there are a lot of neurons in the network that have a very small effect on the output. Therefore, those neurons will have a small effect on $Y_0 - Y_1$. We want to quantify this "small" change by introducing an ϵ relaxation. It might be possible to turn those neurons from on/off to dc while remaining within the ϵ approximation of $Y_0 - Y_1$. Hence, it is expected that $\mathsf{FindMinimal}(\sigma, I)$ will be able to find more relaxed activation patterns capturing larger regions of the input space. Note that this does not guarantee that we will find larger regions overall, because the final output involves the intersection with the 'critical hyperplane' $(dW \cdot X + db \geq \epsilon)$. If we keep ϵ too large, we will find a large σ' but the final region might be small because the 'critical hyperplane' becomes more stringent as ϵ increases. On the other hand, if we keep ϵ too small, we won't get considerable relaxation in the activation pattern σ'. This trade-off on ϵ is revealed in our experiments also. More importantly, we are able to show that for a tuned value of the parameter ϵ, our method is able to capture larger regions in the input space. We illustrate some points of the theory and the advantage of our approach on a small toy ReLU net in appendix 6.1.

4 Experiments

Goal We conduct various experiments under different settings with the following objectives in mind (i) Compare our approach against baseline under different evaluation metrics (as defined in [7] and described below) for a practical dataset, network and its properties. (ii) Argue about the robustness of our approach in choice of initial activation pattern for a given network. (iii) Validate our hypothesis regarding the trade off in the choice of ϵ by conducting all experiments for a wide range of ϵ's.

Evaluation Metrics The verification algorithms output an intersection of halfspaces in $\mathcal{X}$ (conjunction of linear inequalities) for which the property to be verified holds. This makes it difficult to directly measure and compare the performance of algorithms in this domain. Note that along with the bigger regions in the input space, we are also interested in finding those regions which carry the maximum probability mass of underlying data distribution. Based on this, we use two different metrics for evaluations as proposed in [7] : (1) support on training data, and (2) volume of $\mathsf{UA-boxes}$. The former refers to number of training data points that lie in the outputted input region. The latter is defined as bounds on each dimension of input (interval $[lo_i, hi_i]$ for dimension i) that lies inside the output region and estimate its size.

Dataset and Network Firstly, note that our approach can be used to verify any neural network with ReLU activations trained on any available dataset. We use a classification dataset: PIMA Indian Diabetes Dataset from UCI repository [4]. The data consists of 9 input features comprising of skin thickness, glucose level, blood pressure, etc and an output variable for whether or not the person had diabetes. There are 768 training points. We trained a neural network consisting of 2 hidden layers with 12 and 10 neurons respectively and ReLU activations. This network is able to achieve 81% accuracy on using a train-test (85-15) split. This network has a total of 2^{22} (since $12 + 10$ hidden neurons) possible activation patterns however, only 127 of them are realized by the training dataset. Given an initial point in the input space, the goal of verification is to find the maximum surrounding input region such that the 'class-guarantee' property (defined in 2) holds. Given an initial point, we are interested in finding a region such that "For every point in the region, the predicted class is same as the given initial point" .

Experimental setting For our implementation of the algorithm, we used Marabou [8] as the decision procedure in the $\mathsf{checkSAT}$ calls. We implemented $\mathsf{FindMinimal}$ using a greedy approach as done in [7], explained towards the end of section 2. We also used cvxpy [3] to compute under approximation boxes ($\mathsf{UA-boxes}$) to evaluate the identified region in the input space. The code for the implementation is available here. [1] We run the baseline [7] and our algorithm with different ϵ's starting from an initial training point in each of the 127 realized activation patterns of the network. For each activation pattern, we try to find the maximum input region for the property that the predicted class of the given input is the chosen class for that region and compute the metrics for it. Figure 1 summarizes our experiments for comparison of our method with the baseline. It consists of four scatter plots where each red point corresponds to metric value for the baseline on one of the 127 initial input points while blue is for our approach. More concretely, if for a particular initial point, if blue point lies above the red point, this shows that our approach was able to achieve a higher metric value and hence a better performance than the baseline on that initial point. Since the goal for property inference is defined for a given random initial point, we want our approach to have larger metric values(and hence lie above in plot) for as many different input points as possible. The first two plots correspond to the support in training data metric and the remaining two are for the log-volume for the $\mathsf{UA-boxes}$. The first plot for each metric shows the performance of

[1]https://github.com/typerSniper/NNInfer/tree/master/Code

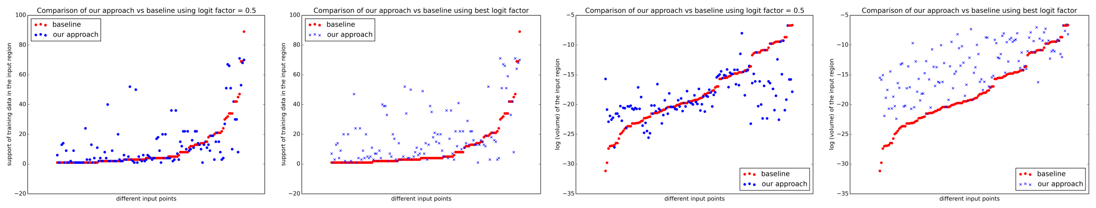

Figure 1: Comparison of our approach and baseline on different metrics for different initial points from training data. Higher value on any input point indicates better performance (larger input regions) starting from that point. Blue points (our approach) lie above red points (baseline) on large number of initial points in both the metrics

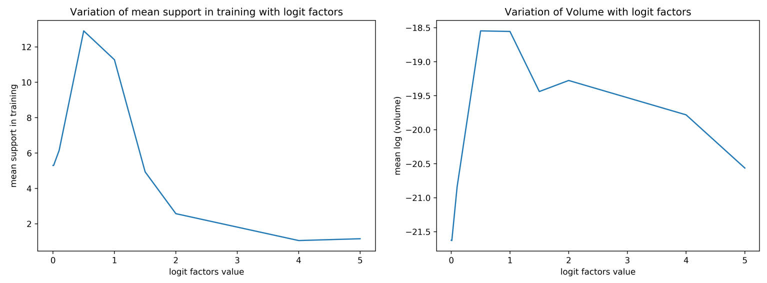

Figure 2: Trends in metric values with change in value of ϵ averaged over different initial points in the training data

our approach on a fixed hyper-parameter while the second one demonstrates the effectiveness of our approach if we chose the best hyper-parameter ϵ for each input point. We also plot the average support and volume over all the points for different values of the $\mathsf{logit-factor}$ in 2. The $\mathsf{logit-factor}$ is just a scaled version of ϵ where scale depends on difference in logits of the initial input point (X_0 in $\mathsf{getRegion}$). This is done to make the free parameter more robust to the actual values weights in the network. In our experiments, $\mathsf{logit-factor}$ is what we vary.

Results Our methods show gains over the baseline as shown in Figure 1. Concretely, in 50 out of 127 initial points we begin with, we are able to achieve more training data support of the identified region, while an equal train support in 41 of the remaining points. This is achieved at $\mathsf{logit-factor} = 0.5$. This indicates that we are able to identify bigger regions and hence can reason about the properties for larger number of points from the data distribution. This effect is compounded significantly if we choose the best ϵ for each input point as visible in the second and fourth plot in the figure, where for almost all initial inputs, we can achieve larger volume. We also verify experimentally that in 53 initial points, the baseline is even not able to call $\mathsf{FindMininimal}$ as the property fails to hold true in the initial activation pattern. Since we guarantee call to $\mathsf{FindMininimal}$ in our approach, we demonstrate that we are more robust to the choice of the initial activation pattern. Figure 2 validates our hypothesis about trends in choice of ϵ. As mentioned earlier, a large ϵ allows more relaxation of the activation pattern but reduces the intersection region with the additional constraint of $dW \cdot X + db \geq \epsilon$. We observe this trend for all the inputs of the diabetes dataset. Both the plots show a region of increase followed by a decrease with the best performance achieved at $\mathsf{logit-factor} = 0.5$.

5 Conclusion and Future Work

In this work, we highlight the significance of the use of interpolating properties in verifying safety properties for neural networks. We build up a methodology by defining an intermediate property closely related to neural network structure, which subsumes the actual property to be proven and hence acts as a bridge between the user-desired high-level specification and the low-level architecture of the network. By conducting preliminary experiments on a small real-world classification dataset, a ReLU network and with the class dominance property (ie, $Y_0 > Y_1$), we demonstrate the merit in this direction of research by outperforming the existing approach. Note that the theory of interpolants is fairly general and can be extended further to different safety properties, complex datasets and advanced architectures. These initial encouraging results motivate us to develop broader theory and do large-scale experiments on different datasets and benchmarks, networks with convolution and pooling layers, and other safety-critical properties other than class dominance as future work.

References

[1] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[2] Markus Borg, Cristofer Englund, Krzysztof Wnuk, Boris Duran, Christoffer Levandowski, Shenjian Gao, Yanwen Tan, Henrik Kaijser, Henrik Lönn, and Jonas Törnqvist. Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry. *arXiv preprint arXiv:1812.05389*, 2018.

[3] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[4] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[5] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018.

[6] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018.

[7] Divya Gopinath, Hayes Converse, Corina Pasareanu, and Ankur Taly. Property inference for deep neural networks. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 797–809. IEEE, 2019.

[8] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.

[9] Xuankang Lin, He Zhu, Roopsha Samanta, and Suresh Jagannathan. ART: abstraction refinement-guided training for provably correct neural networks. *CoRR*, abs/1907.10662, 2019.

[10] Kenneth L. McMillan. Interpolation and sat-based model checking. In *CAV*, 2003.

[11] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

[12] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[13] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

6 Appendix

6.1 Illustration

In this section we take a very small ReLU neural network and illustrate the theoretical structures and the advantage of our approach. The input $X \in \mathbb{R}^2$ and the output $F(X) \in \mathbb{R}^2$. Staying in $\mathbb{R}^2$ makes visualization easier. There is one hidden layer with four neurons. We arbitrarily fix the weights and biases of this network. We're focusing on the 'class-guarantee' property, namely $P(Y) = Y_0 > Y_1$.

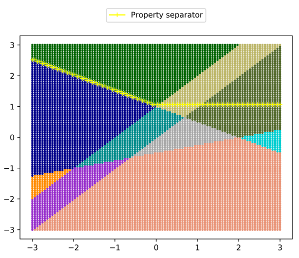

Figure 3: Honeycomb structure of Activation patterns

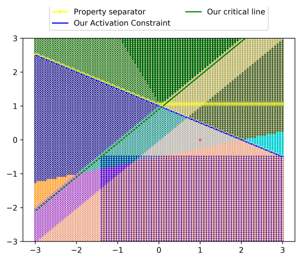

Figure 4: Our approach with $\epsilon = 0.1$

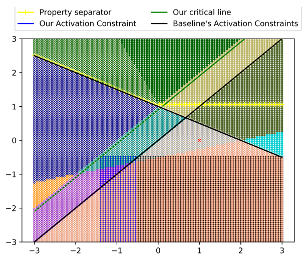

Figure 5: Baseline approach overlayed on our approach

First we illustrate the "honeycomb" structure of the various activation patterns. Each point $X \in \mathbb{R}^2$ lies in its activation pattern σ_X. Note that multiple X's can have the same σ_X. There are a total of 2^H such patterns possible where H is the number of hidden neurons. $H = \sum_{l=1}^{L-1} N_l = 4$ in this case. Of course, not all of them will be realizable because the weights of the network constraint what all activation patterns have a non-empty support. Because each activation pattern is an intersection of halfspaces, the entire structure of activation patterns overlayed on the input space ($\mathbb{R}^2$ here) forms a "honeycomb"-like structure, shown in figure 3.

There will be a separator in this input space for the two classes that we predict. The separator is the line $F(X)_0 = F(X)_1$ and demarcates the regions of each class label. That separator will pass through some activation patterns, but inside an activation pattern the separator will be linear (since $F(X)$ is linear). So the class separator is overall piecewise linear in the space $\mathcal{X}$. The yellow line in figure 3 plots this separator of the original property, ie, the line at which $\mathsf{P}(Y)$ goes from True to False.

We fix the starting point $X_0 = (1, 0)^T \in \mathbb{R}^2$ to be the same for both the approaches (marked red cross in figures 4 and 5). The activation pattern σ_{X_0} has all 4 hidden neurons fixed. Our approach is able to relax this to σ'_{our} that has only 1 neuron constrained (and other 3 are dc). We then take the intersection of this region with the 'critical hyperplane' defined by $dW \cdot X + db \geq \epsilon$. This is illustrated in the figure 4. The blue box is the axis-aligned $\mathsf{UA} - \mathsf{box}$ of this region.

The baseline approach is able to relax only 2 neurons out of the initial four to dc. So σ'_{base} has two neurons constrained, shown in figure 5. Its axis-aligned $\mathsf{UA} - \mathsf{box}$ is shown in black in figure 5. We can see that our approach captures a bigger region of the input space.