Final Project Report

Abstract interpretation-based verification of neural networks

Using case disjunctions and junctions for ReLU activation function

Category

Implementation of a verification method for reactive/hybrid systems

Names & Titles

Biswajeet Kar Master of Technology in IoT

Tzu-yi Chiu X2017

Contents

1 Overview **2**

2 Concrete example **3**

2.1 Case disjunctions 3

2.2 Case junctions 5

2.3 Bounds computed with our abstraction 6

3 Implementation **7**

4 Conclusion **8**

1 Overview

Each time we approximate ReLU by a linear function using an abstract Zonotope transformer, the output generates a noise symbol and loses precision at every hidden layer. One alternative way is to consider the conditional statement:

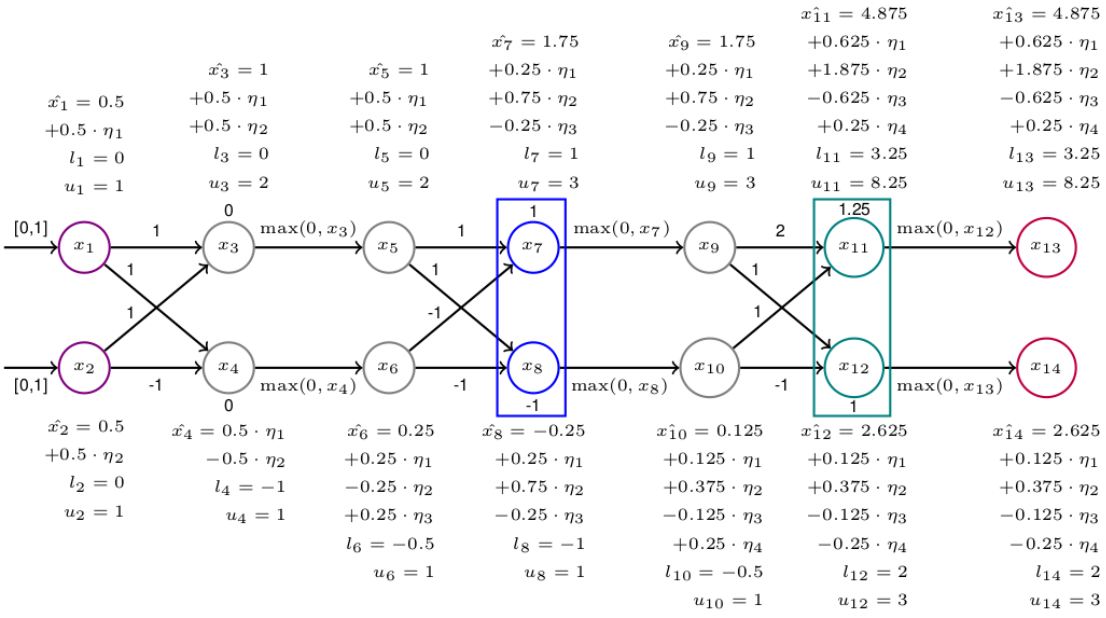
$$\text{ReLU}(x) = \max(x, 0) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

In our approach, we propose to create case disjunctions whenever we encounter a ReLU function for which the input variable can range from negative values to positive values. Indeed, the number of cases at the output layer can increase exponentially with the number of layers, but the precision is exact in every single case, and every output will only depend on the intial input values.

Though a lot of cases may be computed, some expressions of final output values can be considered "closed enough" to be joined, by simply abstracting the minor different parts, in order to obtain only one expression at the end. The number of cases can thus be reduced when propogating inside the neural network, without losing too much precision. Especially, we will show that the constraints used for case disjunctions may lead to a refinement of the abstraction we apply for the junctions.

We will now show how the junctions and disjunctions are concretely computed, with the example provided by [4] in Figure 1, step by step. In [4], the authors were able to prove that $x\hat{}_{13}$ is always greater than $x\hat{}_{14}$ by refining the bounds while using an abstract Zonotope transformer. In our abstraction using disjunctions and junctions, the bounds computed for the output variables will be the same as those computed in the paper. We can actually prove that they are the optimal ones.

Figure 1: Example and analysis with DeepZ (see [4] for details)



2 Concrete example

2.1 Case disjunctions

(First Layer.) We start by writing:

$$\begin{cases} \hat{x_1} = 0.5 + 0.5\eta_1, & l_1 = 0,\ u_1 = 1 \\ \hat{x_2} = 0.5 + 0.5\eta_1, & l_2 = 0,\ u_2 = 1 \end{cases}$$

We apply the affine transformation on $\hat{x_1}$ and $\hat{x_2}$ to get:

$$\begin{cases} \hat{x_3} = \hat{x_1} + \hat{x_2} = 0.5(\eta_1 + \eta_2) + 1, & l_3 = 0,\ u_3 = 2 \\ \hat{x_4} = \hat{x_1} - \hat{x_2} = 0.5(\eta_1 - \eta_2), & l_4 = -1,\ u_4 = 1 \end{cases}$$

Here we encounter for the first time the ReLU function. Since $\hat{x_3}$ is always positive, $\hat{x_5}$ remains the same. Two cases have to be considered while computing $\hat{x_6}$ depending on the sign of $\hat{x_4}$, where:

$$\hat{x_4} \geq 0 \Leftrightarrow \eta_1 \geq \eta_2$$

(Second Layer.)

Case 1: $\eta_1 \geq \eta_2$

$$\begin{cases} \hat{x_5} = \hat{x_3} = 0.5(\eta_1 + \eta_2) + 1 \\ \hat{x_6} = \hat{x_4} = 0.5(\eta_1 - \eta_2) \end{cases}$$

Affine transformation:

$$\begin{cases} \hat{x_7} = \hat{x_5} - \hat{x_6} + 1 = \eta_2 + 2 \\ \hat{x_8} = \hat{x_5} - \hat{x_6} - 1 = \eta_2 \end{cases}$$

Here again we encounter another ReLU function. Since $\hat{x_7}$ is always positive, $\hat{x_9}$ remains the same. However we have two subcases depending on the sign of $\hat{x_8}$, where:

$$\hat{x_8} \geq 0 \Leftrightarrow \eta_2 \geq 0$$

(Third Layer.)

Case 1.1: $\eta_2 \geq 0$

$$\begin{cases} \hat{x_9} = \hat{x_7} = \eta_2 + 2 \\ \hat{x_{10}} = \hat{x_8} = \eta_2 \end{cases}$$

And thus:

$$\begin{cases} \hat{x_{11}} = 2\hat{x_9} + \hat{x_{10}} & = 3\eta_2 + 5.25 \\ \hat{x_{12}} = \hat{x_9} - \hat{x_{10}} + 1 & = 3 \end{cases}$$

Since everything is positive:

$$\boxed{\begin{cases} \hat{x_{13}} = \hat{x_{11}} & = 3\eta_2 + 5.25 \\ \hat{x_{14}} = \hat{x_{12}} & = 3 \end{cases} \quad \text{under constraint} \quad \begin{cases} \eta_1 \geq \eta_2 \\ \eta_2 \geq 0 \end{cases}} \tag{1}$$

Case 1.2: $\eta_2 < 0$

$$\begin{cases} \hat{x_9} = \hat{x_7} = \eta_2 + 2 \\ \hat{x_{10}} = 0 \end{cases}$$

Thus:

$$\begin{cases} \hat{x_{11}} = 2\hat{x_9} + \hat{x_{10}} & = 2\eta_2 + 5.25 \\ \hat{x_{12}} = \hat{x_9} - \hat{x_{10}} + 1 & = \eta_2 + 3 \end{cases}$$

At the end:

$$\boxed{\begin{cases} \hat{x_{13}} = \hat{x_{11}} & = 2\eta_2 + 5.25 \\ \hat{x_{14}} = \hat{x_{12}} & = \eta_2 + 3 \end{cases} \text{ under constraint } \begin{cases} \eta_1 \geq \eta_2 \\ \eta_2 < 0 \end{cases}} \tag{2}$$

Case 2: $\eta_1 < \eta_2$

$$\begin{cases} \hat{x_5} = \hat{x_3} = 0.5 \cdot (\eta_1 + \eta_2) + 1 \\ \hat{x_6} = 0 \end{cases}$$

Affine transformation:

$$\begin{cases} \hat{x_7} = \hat{x_5} - \hat{x_6} + 1 = 0.5(\eta_1 + \eta_2) + 2 \\ \hat{x_8} = \hat{x_5} - \hat{x_6} - 1 = 0.5(\eta_1 + \eta_2) \end{cases}$$

Here again we encounter another ReLU function, so two subcases depending on the sign of $\hat{x_8}$, where:

$$\hat{x_8} \geq 0 \Leftrightarrow \eta_1 + \eta_2 \geq 0$$

(Third Layer.)

Case 2.1: $\eta_1 + \eta_2 \geq 0$

$$\begin{cases} \hat{x_9} = \hat{x_7} & = 0.5(\eta_1 + \eta_2) + 2 \\ \hat{x_{10}} = \hat{x_{10}} & = 0.5(\eta_1 + \eta_2) \end{cases}$$

Thus:

$$\begin{cases} \hat{x_{11}} = 2\hat{x_9} + \hat{x_{10}} + 1.25 & = 1.5(\eta_1 + \eta_2) + 5.25 \\ \hat{x_{12}} = \hat{x_9} - \hat{x_{10}} + 1 & = 3 \end{cases}$$

At the end:

$$\boxed{\begin{cases} \hat{x_{13}} = \hat{x_{11}} & = 1.5(\eta_1 + \eta_2) + 5.25 \\ \hat{x_{14}} = \hat{x_{12}} & = 3 \end{cases} \text{ under constraint } \begin{cases} \eta_1 < \eta_2 \\ \eta_1 + \eta_2 \geq 0 \end{cases}} \tag{3}$$

Case 2.2: $\eta_1 + \eta_2 < 0$

$$\begin{cases} \hat{x_9} = \hat{x_7} = 0.5(\eta_1 + \eta_2) + 2 \\ \hat{x_{10}} = 0 \end{cases}$$

Thus:

$$\begin{cases} \hat{x_{11}} = 2\hat{x_9} + \hat{x_{10}} + 1.25 & = (\eta_1 + \eta_2) + 5.25 \\ \hat{x_{12}} = \hat{x_{10}} & = 0.5(\eta_1 + \eta_2) + 3 \end{cases}$$

At the end:

$$\boxed{\begin{cases} \hat{x_{13}} = \hat{x_{11}} & = (\eta_1 + \eta_2) + 5.25 \\ \hat{x_{14}} = \hat{x_{12}} & = 0.5(\eta_1 + \eta_2) + 3 \end{cases} \text{ under constraint } \begin{cases} \eta_1 < \eta_2 \\ \eta_1 + \eta_2 < 0 \end{cases}} \tag{4}$$

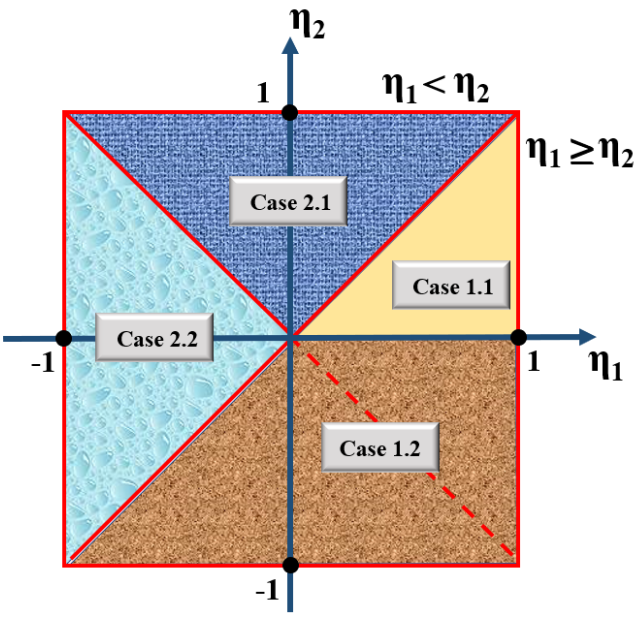


Figure 2: Case disjunction on the 2D input plan for our example

2.2 Case junctions

Notice the similarity we can observe between the equations (1) and (3), and between the equations (2) and (4). They still remain similar because we didn't go deep inside the neural network. Imagine we were in a larger network which includes our example, then at this moment, we would like to start combining these similar expressions in order to reduce the output possibilities as soon as possible, by abstracting the minor difference using another noise symbol. More precisely, consider the two following junctions in our example:

(1)&(3) We introduce $\eta_3 \in [-1; 1]$ such that:

$$\eta_3 = \begin{cases} \eta_2 & \text{in Case 1.1} \\ (\eta_1 + \eta_2)/2 & \text{in Case 2.1} \end{cases}$$

The junction of these two subcases will result in the following constraint:

$$\eta_2 \geq 0 \ \wedge \ \eta_1 + \eta_2 \geq 0$$

Notice that under this constraint, we even won a significant refinement for η_3 which will only range inside $[0; 1]$. We can thus combine (1) & (3) by writing:

$$\boxed{\begin{cases} \hat{x_{13}} = 3\eta_3 + 5.25 \\ \hat{x_{14}} = 3 \end{cases}, \eta_3 \in [0; 1]} \tag{5}$$

(2)&(4) As same as the previous example, we introduce $\eta_4 \in [-1;1]$ such that:

$$\eta_4 = \begin{cases} \eta_2 & \text{in Case 1.2} \\ (\eta_1 + \eta_2)/2 & \text{in Case 2.2} \end{cases}$$

The junction of these two subcases will result in the following constraint:

$$\eta_2 < 0 \ \vee \ \eta_1 + \eta_2 < 0$$

Notice that under this constraint, η_4 will only range inside $[-1;0]$. We then combine (2) & (4) by writing:

$$\boxed{\begin{cases} \hat{x_{13}} = 2\eta_4 + 5.25 \\ \hat{x_{14}} = \eta_4 + 3 \end{cases}, \eta_4 \in [-1;0]} \tag{6}$$

Of course, the similarity between the equations (5) and (6) naturally leads us to considering another noise symbol $\varepsilon \in [-2;3]$ such that:

$$\varepsilon = \begin{cases} 3\eta_3 & \text{in Case 1.1} \cup \text{Case 2.1} \\ 2\eta_4 & \text{in Case 1.2} \cup \text{Case 2.2} \end{cases}$$

and thus we can combine these two remaining cases (5) and (6) to obtain:

$$\begin{cases} \hat{x_{13}} = \varepsilon + 5.25 \\ \hat{x_{14}} = \eta_4 + 3 \end{cases} \quad \text{where:} \quad \begin{cases} \varepsilon \in [-2;3] \\ \eta_4 \in [-1;0] \end{cases}$$

At the end:

$$\boxed{\begin{cases} \hat{x_{13}} = 2.5\eta_5 + 5.75 \\ \hat{x_{14}} = \eta_6 + 2.5 \end{cases} \text{with } \eta_5, \eta_6 \in [-1;1]} \tag{7}$$

2.3 Bounds computed with our abstraction

The lower bounds l_{13}, l_{14} and upper bounds u_{13}, u_{14} computed by our abstraction take the following values:

$$\begin{cases} u_{13} = 8.25 \\ l_{13} = 3.25 \end{cases} \quad \& \quad \begin{cases} u_{14} = 3 \\ l_{14} = 2 \end{cases}$$

Notice that these bounds are the same as obtained in [4]. We can actually prove that, in our example, these are the optimized bounds by observing:

$$\begin{cases} (\eta_1, \eta_2) = (1,1) & \Rightarrow \ (\hat{x_{13}}, \hat{x_{14}}) = (8.25, 3) \\ \eta_2 = -1 & \Rightarrow \ (\hat{x_{13}}, \hat{x_{14}}) = (3.25, 2) \end{cases}$$

3 Implementation

We implemented a simple "empirical verifier", which tests the network by inputing random values in a particular range, and visualize the bounds of output values. This empirical method can, after a sufficient number of tests, produce the distribution of output values between the theoretical lower and upper bounds in order to validate the bounds we have computed with our abstraction.

It consists of a Python program that verifies a neural network, which takes our network architecture and input values as arguments, and calculates the output values with our network. Here we are using only the ReLU function inside the intermediate layers.

The network architecture includes a list of individual layers inside a neural network with information such as number of input and output nodes, biases, weights, and the activation function in each node. Find in Table 1 the functions used for our implementation.

We have a simplest version of a verifier with 1000 input data points (random inputs of $\hat{x_1}, \hat{x_2} \in [0; 1]^2$) and tried to plot the two outputs $\hat{x_{13}}$ and $\hat{x_{14}}$ seperately on a 3D model with the software *Plot-ly*. After then, we found all output values indeed between the expected lower and upper bounds:

$$\begin{cases} u_{13} = 8.25 \\ l_{13} = 3.25 \end{cases} \quad \& \quad \begin{cases} u_{14} = 3 \\ l_{14} = 2 \end{cases}$$

The CSV file is joined with this report. Notice that the boundaries between input case disjunctions are easy to see.

Function name	**Brief description**
init_layers	Initiate the program with our network architecture and return an array of dictionary containing the bias and weights of individual layers.
relu	The infamous ReLU function.
single_step	Take input from the previous layer and calculate the output until the next.
run_simulation	Run an end-to-end simulation over the network to get the final output.
main	Take random inputs and store the input and output values inside a csv file.

Table 1: The four functions used for our implementation

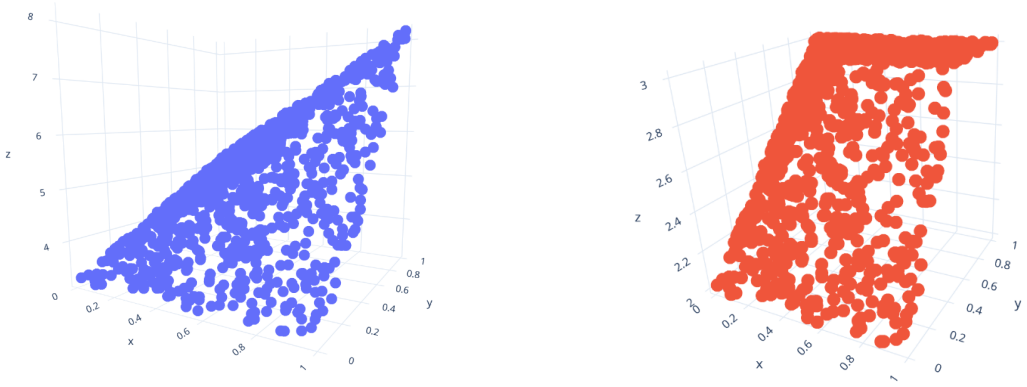


Figure 3: The 3D plot of $x\hat{}_{13}$ (left) and $x\hat{}_{14}$ (right) with 1000 random inputs.

4 Conclusion

As we are able to reduce the number of noises and capted to the minimum, we have indeed optimized the precision while keeping the lower and upper bounds fixed as for the previous work done in [4].

However, this abstraction must be mathematically formalized to facilitate the implemention of the associated verifier so that we can prove properties efficiently when working in a larger and deeper network. Especially, we have to define properly a sort of *distance* between output expressions to determine which of them may be considered "close enough", and thus the computer can easily join these cases and abstract them into a final expression. This is undoubtedly a tough task because even for human, sometimes it is not easy to visualize the similarity of certain expressions. As a result, the implementation we have done only consisted in empirical verification.

References

[1] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[2] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. A logical product approach to zonotope intersection. In *Proceedings of the 22Nd International Conference on Computer Aided Verification*, CAV'10, 2010.

[3] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Puschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems 31*. 2018.

[4] Gagandeep Singh, Timon Gehr, Markus Puschel, and Martin Vechev. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*. URL: https://files.sri.inf. ethz.ch/website/papers/RefineZono.pdf.