# Reliable and Interpretable AI Project Write-Up

Remo Geissbühler, Jakob von Kalle, Laurin Paech[§]

## I. Introduction, Notation

The goal of this project is to implement a Neural Network verifier based on the DeepPoly relaxation proposed in [1]. Namely, consider a classification network $N : \mathbb{R}^d \to \mathbb{R}^k$, where $k$ is the number of classes. Assume that $N$ consists of $L$ layers, each one representing a mapping $\mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$. Now, for every layer $\ell$ we introduce a symbolic variable $\mathbf{x}^{(\ell)}$ and weights $\mathbf{W}_{\leqslant}^{(\ell)}$, $\mathbf{W}_{\geqslant}^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ as well as biases $\mathbf{v}_{\leqslant}^{(\ell)}$, $\mathbf{v}_{\geqslant}^{(\ell)} \in \mathbb{R}^{n_\ell}$ such that we can bound it symbolically as

$$\mathbf{x}^{(\ell)} \leqslant \mathbf{W}_{\leqslant}^{(\ell)} \mathbf{x}^{(\ell-1)} + \mathbf{v}_{\leqslant}^{(\ell)}, \tag{1a}$$

$$\mathbf{x}^{(\ell)} \geqslant \mathbf{W}_{\geqslant}^{(\ell)} \mathbf{x}^{(\ell-1)} + \mathbf{v}_{\geqslant}^{(\ell)}. \tag{1b}$$

Note that in the above equations, we use the relations $\leqslant, \geqslant$ component-wise. In contrast to [1], the symbolic equations only depend on the previous layer.

By backsubstitution as introduced in [1], we can use the symbolic bounds to find concrete bounds $\mathbf{l}^{(\ell)}$, $\mathbf{u}^{(\ell)} \in \mathbb{R}^{n_\ell}$ such that $\mathbf{l}^{(\ell)} \leqslant \mathbf{x}^{(\ell)} \leqslant \mathbf{u}^{(\ell)}$. Assume now that we want to certify $N$ for a target class $T$, i.e. we want to prove $\mathbf{x}_T^{(L)} - \mathbf{x}_j^{(L)} \geqslant 0$ for all classes $j \in [k], j \neq T$. To that end, pursuant to [1], we introduce an additional layer with $k-1$ symbolic variables that represent $\mathbf{x}_T^{(L)} - \mathbf{x}_j^{(L)}$. Clearly, certifying $N$ is now equivalent to $\mathbf{l}^{(L+1)} > \mathbf{0}$ after backsubstitution.

The problem at hand concerns the symbolic weights for the ReLU Layer. In particular, we can use $\mathbf{W}_{\geqslant}^{(\ell)} = \operatorname{diag}(\boldsymbol{\lambda})$ for any $\boldsymbol{\lambda} \in [0,1]^{n_\ell}$. We now aim at finding a heuristic that given $\mathbf{l}^{(\ell-1)}$, $\mathbf{u}^{(\ell-1)}$ chooses such a $\boldsymbol{\lambda}^{(\ell)}$.

## II. DeepPoly Implementation

We solved backsubstitution, the main ingredient to the DeepPoly algorithm, recursively. Assume that we want to find $\mathbf{l}^{(\ell)}$and suppose that we already have an equation of the form:

$$\mathbf{x}^{(\ell)} \geqslant \mathbf{W}_{\geqslant}^{(\ell \to m)} \mathbf{x}^{(m)} + \mathbf{v}_{\geqslant}^{(\ell \to m)}, \tag{2}$$

that is, we already backsubstituted up to layer $m+1$. Every layer now implements a `backsubstitute` method that, given $\mathbf{W}_{\geqslant}^{(\ell \to m)}$, $\mathbf{v}_{\geqslant}^{(\ell \to m)}$ computes new weights $\mathbf{W}_{\geqslant}^{(\ell \to m-1)}$, $\mathbf{v}_{\geqslant}^{(\ell \to m-1)}$ with respect to $\mathbf{x}^{(m-1)}$:

$$\mathbf{x}^{(\ell)} \geqslant \mathbf{W}_{\geqslant}^{(\ell \to m-1)} \mathbf{x}^{(m-1)} + \mathbf{v}_{\geqslant}^{(\ell \to m-1)}. \tag{3}$$

[§]Equal contribution

An overview of the implementations for different layers can be found below:

- **Affine:** Matrix multiplication with the layer's weights and bias
- **ReLU:** Element-wise multiplication with upper or lower slope (cf. [1]), depending on the sign of $\mathbf{W}_{\geqslant}^{(\ell \to m)}$.
- **Convolution:** Compute the transpose convolution of $\mathbf{W}_{\geqslant}^{(\ell \to m)}$ matching the layer's input shape

## III. Heuristic

Our approach is relatively straight-forward. For every ReLu layer $\ell$, we keep a variable $\boldsymbol{\lambda}^{(\ell)} \in \mathbb{R}^{n_\ell}$, our current choice for the lower bound. Initially, we choose $\boldsymbol{\lambda}^{(\ell)}$ that minimizes the area of the polygon, as explained in [1]:

$$\boldsymbol{\lambda}_0^{(\ell)} := \mathbb{1}_{\{-\mathbf{l}^{(\ell-1)} < \mathbf{u}^{(\ell-1)}\}}. \tag{4}$$

After the "forward pass", we compute the loss function as

$$\mathcal{L}(\Lambda) := -\sum_{i \leqslant k-1} \mathbb{1}_{\{l_i^{(L+1)}(\Lambda) \leqslant 0\}} l_i^{(L+1)}(\Lambda), \tag{5}$$

where we use $\Lambda$ to denote the set of all $\boldsymbol{\lambda}^{(\ell)}$. We write $\mathbf{l}^{(L+1)}(\Lambda)$ to highlight that this lower bound depends on our choice of $\Lambda$. Since computing $\mathbf{l}^{(L+1)}(\Lambda)$ consists of backsubstitution (which is essentially a linear operation) and since the weights depend differentiably on $\Lambda$, we can (sub-)differentiate $\mathcal{L}$ and use Projected Gradient Descent (PGD) to update our current choice. Namely:

$$\boldsymbol{\lambda}_{t+1}^{(\ell)} := \operatorname{clamp}\left(\boldsymbol{\lambda}_t^{(\ell)} - \alpha \nabla_{\boldsymbol{\lambda}^{(\ell)}} \mathcal{L}(\Lambda_t)\right). \tag{6}$$

Here, $\operatorname{clamp} : \mathbb{R} \to [0,1]$ is applied component-wise and represents the projection step of PGD.

Finally, we repeat the update step in (6) indefinitely. If at any point $\mathbf{l}^{(L+1)} > \mathbf{0}$, we conclude that the network is verifiable.

## IV. Implementation Details

In our implementation, we use the automatic differentiation functionality of PyTorch [2] to find the gradients with respect to $\Lambda$. Furthermore, we use the Adam optimizer [3] to perform the update step. In our experiments, we used an initial learning rate of 0.5 that decays by a factor of 0.97 every step. All the other parameters of the optimizer were left on default.

## REFERENCES


[1] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3290354

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.