Reliable and Interpretable AI Project Write-Up

Kenan Bešić, Laurin Paech, Pouya Pourjafar

I. Introduction

Since the advent of Deep Learning, robustness and interpretability of models have been an important issue. By this it can be ensured that a model is reliable, safe and even fair. In this project, we build an automated verifier for proving robustness of fully connected and convolutional neural networks with rectified linear activations (ReLU) against adversarial attacks, which is sound with respect to floating point arithmetic. Such an attack generates an adversarial image that is inside an L_∞-norm based ϵ-ball around the original image such that the network classifies the adversarial example to a different output class.

II. Initialization

For this we build up-on the DeepZ relaxations introduced in the lecture. First, we generate the initial zonotope. To do so, we create the center matrix, our initial image, and a matrix with the given epsilon values in the diagonal as error-magnitude matrix. Since only pixel intensities are between zero and one in the images, we can clip the initial zonotope such the intervals stay in the range of 0 to 1. This makes our analysis more precise. After clipping, we adjust the center exactly to the middle of the new lower and the new upper bound. After that we set the magnitude of the error-term such that we get the correct clipped interval.

After having prepared the data, we iterate over all layers of the network. Based on the provided networks, the verifier needs to handle the ReLU activation function and support normalization, linear, convolutional layers. As the zonotope affine transformer is exact and all layers but the ReLU layers are affine, we only introduce new error terms for ReLU. Furthermore, it is important to note, that the bias contained in some layers is not added twice, i.e. to the center and to the magnitudes. Next, we will go over each layer and explain our approach.

III. Normalization

Every network starts with a normalization layer. As this layer is affine, we just need to normalize the zonotope representation. We do this as follows (m represents the magnitude):

$$y = \frac{x - \mu}{\sigma} = \frac{a_0 + m - \mu}{\sigma} = \frac{a_0 - \mu}{\sigma} + \frac{m}{\sigma}$$

IV. Conv

Convolutional layers consist of multiplication with the neuron weights and addition over the kernel. This is again affine. The center matrix gets passed through the layer. To ensure that the magnitudes are passed correctly, a matrix for every error term is constructed such that the magnitudes of the same neuron are in the same position. By that the convolutional layers are now an affine transformation on the magnitudes.

V. ReLU

The ReLU transformer introduced in DeepZ is a zonotope transformer with minimum area. First, we calculate the slopes introduced in DeepZ. We initialize the trainable parameters with the calculated slopes from the first iteration. We then define a loss function, which captures how distinct the lower box bound of the true label is to the upper bounds of the other labels. Using this loss function and the SGD optimization, we improve the slopes to verify with higher precision. Since we make sure that our slopes are always between zero and one by a clipping module, the verifier is sound. Some of the computations were detached in the bigger networks (conv3, conv4), to make the runtime faster. In those cases, only limited local dependencies are considered for the gradient computation.

VI. Verification

After over-approximating the input ball, we now need to verify the result, that is $x_{target} > x_{other}$ for all other labels. We have seen the basic verification using the box bounds per neuron. However, this can be imprecise and we need to analyze the structure of the zonotope for further precision. For this, we need to factorize the common error terms for each neuron. Now, error terms are shared between the output neurons and verification can be done more precisely.

VII. Evaluation

In the evaluation of our verifier, we used the initially provided test cases and preliminary submission test cases. Additionally, to ensure robustness, we generated several counter examples using PGD attacks on sampled MNIST data. These are not used in the verifier and are only used for testing on our local branches.