Alex Hägele
Florian Barkmann
Simon Heinke

Verifying deep neural networks with DeepPoly

Verifier implementation

Our approach to the verification task at hand was to build for each given network a corresponding verification network: for each layer in the original network we add a matching transformer layer in the verification network that takes a DeepPoly object as input and returns the transformed DeepPoly object. To the verification network we finally also append a verification layer.
First, we implemented the DeepPoly class into which we save all the constraints for each layer and which also holds the method used for the backsubstitution. A DeepPoly object thus holds two vectors lb and ub which represent the concrete bounds, as well as two matrices that hold the upper and lower symbolic constraints, which are expressed in terms of the outputs of the preceding layer.
Then we built our transformer classes, where we implemented one class for each layer type. For the convolutional layers, we transform the convolution operation into a simple matrix-matrix multiplication, so we can treat it as a linear layer. In each convolutional and linear layer we do a full backsubstitution to the input space in order to calculate the concrete lb and ub vectors.
The validation layer is basically a linear layer with the following weight matrix (i.e. symbolic constraints) $W \in \mathbb{R}^{9\times 10}$:

$$W := \begin{pmatrix} -1 & 0 & \dots & 1 & \dots & 0 & 0 \\ 0 & -1 & \dots & 1 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 1 & \dots & -1 & 0 \\ 0 & 0 & \dots & 1 & \dots & 0 & -1 \end{pmatrix}$$

where the column filled with ones corresponds to the true label. With this approach, we can simply use backsubstitution: If all the concrete lower bounds of the validation layer are greater than zero, the input is verified.

Choosing lambdas

We decided to neglect static heuristics and went with an optimizational approach, in which we adapt all our lambda values simultaneously according to a loss function. To choose the λ in the ReLU layers, we defined the following loss function:

$$loss_\lambda = \max(\log(-results_{<0})),$$

where $results_{<0}$ denotes all negative entries of the obtained lower bounds of the validation layer, and where log is the element-wise logarithm. The loss function penalizes the largest negative value of the lower bounds, which prevents us from verifying the input. Thus, by minimizing the loss we obtain λ that maximize the lower bound which results in a more accurate approximation. We found that the log improves convergence speed. We initialize all lambdas with one.
Furthermore, we parameterized λ with $1/(1+\exp(u))$. This allowed us to use Adam to minimize the $loss_{\lambda(u)}$ without having to worry about parameter clamping. We let Adam run until the input was verified or the process times out, which (under testing conditions) is equivalent to "not verified". With this approach we were able to classify all the published testcases (i.e. the original 20 as well as the preliminary 10 testcases) correctly.