RIAI 2020 Course Project

Theresa Blümlein, Edward Günther, and Carina Schnuck
{tbluemlein, gedward, cschnuck}@student.ethz.ch

18th of December 2020

The task of this year's RIAI course project was to use DeepPoly [2] in order to build a verifier that is as precise and scalable as possible while keeping the soundness property. Specifically, the task was to construct a heuristic to find a parameter set of λ's that would allow to verify a given network on a given input (a picture in our case). Preciseness is achieved by correctly verifying as many input-network pairs as possible and scalability is achieved by finding a set of λ's that can verify the given input-network pairs in the short amount of time given for the verification process. Soundness is of minor importance here since accurately implementing DeepPoly will ensure soundness on its own (generally of course, soundness is of great importance). In the following, we will describe our approach. We will do this in a chronological way, meaning we will lay out the process, including those ideas that did not work out in the end, towards the final version of our heuristic.

Our first step was to understand the DeepPoly system introduced in [2] [1] and implement a functioning and correct version in Python using PyTorch. As proposed in the papers and the lecture, we used the set of λ's that would minimize the area of the pointwise transformers within ReLU layers. We also implemented a pairwise difference layer as the last layer. We hoped to achieve some additional tightness of output bounds by calculating bounds in two ways, first by subtracting the predicted class probability of the true label minus all other labels individually and vice versa. However, since we did not see considerable differences in the bounds when comparing the first and second difference, we decided to only do one full back-substitution for the second. With this method we could already verify some of the test images for fully-connected and convolutional networks.

Our idea then was to use either Bayesian optimization or gradient descent (GD) to update the set of λ's. Since the first of these approaches did not work as expected, we focused on the latter. As a loss for GD, we chose the sum of all positive upper bounds. In this way, the algorithm would try to find sets of λ's that would achieve upper bounds all below zero, in which case the input-network pair would be verified (see differences mentioned in the paragraph before). This approach already showed some improvements compared to the smallest-area-heuristic; however, the GD algorithm sometimes got stuck in local minima, which we assume was due to the low signal-noise-ration of the λ's in the first layers to the loss after the last layer. We tried out some minor adjustments such as back-substituting from each layer back to the previous layer (rather than always back-substitute to the first layer) and checking which number of back-steps would achieve the smallest bounds. In the end, we only kept the idea to check whether bounds have improved whenever we update them. Our final and most crucial idea was to start GD not from the last layer for all λ's at the same time but from every layer individually. The idea was to begin after the first ReLU layer, update the λ's associated with this layer according to interval size after the layer until convergence and then repeat this step for the next ReLU layer until we arrive at the last one. After we optimized all layers individually, we used the remaining time to update all λ's simultaneously using the loss after the last layer. Since this did not work too well, we again tried a different approach, now training λ's again layerwise (meaning λ's of all other layers were frozen) but with respect to the loss after the last layer. Again, after optimizing λ's layerwise, GD was used to update all λ's simultaneously for the remaining time. This proved to be a successful strategy and hence became our final solution.

Last, a quick note on the optimization algorithm: The time limit of 180 seconds for the verification process meant that convergence of the GD algorithm had to be fast. Thus, we used the learning rate scheduler proposed in [3] and set optimization hyperparameters in a trial and error manner.

References

[1] Christoph Müller, Gagandeep Singh, Markus Püschel, and Martin Vechev. Neural network robustness verification on gpus, 2020.

[2] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), January 2019.

[3] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates, 2018.