CS 781: Formal Methods in Machine Learning
Project Proposal

Yash Jonjale (22B0990)
Krishna N Agaram (210051003)

October 17, 2024

Contents

1 Proposal

Image recognition is invaluable for interaction with computers; human-like robustness of recognition to small perturbations in the input, however, is still a concern. Oftentimes, the perturbation is due to a mistake in the image collection, e.g. a cat with a tree branch covering its face, or the number 8 with a small ink blot. In this project, we propose to investigate the *verification* of robustness of image recognition models to such *locally spatial* perturbations. In particular, we consider the following *spatially-local* perturbation model to image I parameterized by integer k and real number ϵ: an arbitrary center of perturbation O is chosen within the image, and a square of side length k pixels centred at O is perturbed by adding a random noise of magnitude at most ϵ in $\mathcal{L}^p$ norm. We aim to develop a method to verify, given a model, the robustness to such perturbations when only k and ϵ (not the center O) are known. Our method will build on the state-of-the-art `auto_LirPA` codebase, finetuning approximations and inequalities to the locally spatial perturbation model. Finally, time permitting, we will evaluate the method on a variety of standard image recognition models and popular datasets.

2 Challenges

A non-exhaustive list of challenges that we anticipate follow.

- The key challenge. It is trivial, of course, to simply try every possible center O and verify robustness to the perturbation. However, this yields a quadratic (in image dimensions) blow-up, which is not feasible - even for 100×100-sized images and $30s$ per LiRPA verification, this takes 24 hours. The challenge is to **consider all centers simultaneously during verification, with little loss below the worst-case center**.
- The only approximate quantities in CROWN are the bounds on the ReLU outputs. Our challenge, then, is to **tighten the ReLU bounds for the particular perturbation model**.
- The last step in the LiRPA process is to bound the quantity $A\boldsymbol{\varepsilon}$ with x being the perturbation applied to the image. CROWN uses Hölder's inequality; we will need to **find tighter bounds given the spatial perturbation model**. Actually, this is not sufficient: we also need to make our bounds **differentiable** with respect to the matrix A (and hence the αs).
- These were the theoretical concerns; the devil is in the implementation details. We need to **find and replace the old bounds with tighter versions, making sure the tighter bounds are efficiently computable**. We aim for only a constant-factor slowdown in the verification process.

- Finally, it would be nice to be able, if the verification fails, to be able to **find a center O whose perturbation fails to be recognized**. We intend to try this, time permitting.

Possible solutions to these challenges are discussed in Section 4.

3 Tools and Codebase

We shall build on top of the existing `auto_LiRPA` framework (github) for LiRPA-based robustness verification. Here is a fork of the same that we intend to develop on top of. Of course, we shall be programming with standard machine learning frameworks (`PyTorch`, `numpy` etc) as well. For testing, we shall use (part of) the MNIST and CIFAR datasets, standard image manipulation libraries (`imageio`, `PIL`) and small sequential neural networks. It is unlikely, but if we can get hold of a GPU and are able to program the same changes to the code for running on GPU, we could use much larger models and datasets.

4 Approach

We propose the following approach to solve the challenges outlined in Section 2. We notice that due to the convolution layers preceding the ReLU nodes, it is highly likely that the input x to a ReLU node depends (linearly) on much of the input pixels. This suggests that the lower and upper bounds on the ReLU node input (and hence output) can be tightened by considering the spatial perturbation model: if the input x is written

$$x = \sum_{i,j} A_{ij} I_{i,j} + b = \sum_{i,j} A_{ij} \varepsilon_{i,j} + b',$$

where I is the input image and ε is the perturbation, it is clear that an upper bound on the last summation (suppose that ε is bounded in $\mathcal{L}^\infty$ norm by 1) is the sum of the absolute values of *the top k^2 elements of A* - which can be significantly smaller than the sum of the absolute values of all elements of A. Similarly, a lower bound on the sum is simply the negation of the sum of the absolute values of those same elements. This observation tightens the bounds on the ReLU nodes. Note that this is efficiently computable, as it only requires sorting the elements of A (to compute the linear combination representing the input x itself takes $|I|$ time, so we are only a factor $\log |I|$ slower).

A similar idea holds for tightening the final expression $A\varepsilon$, instead of using the typical Hölder's inequality. We can use the same idea as above: the sum of the absolute values of the top k^2 elements of A is a tighter bound than the sum of the absolute values of all elements of A.

Note that in both cases, we can slightly improve the bounds by only considering the k^2-sized subsets of elements of A that make a square centered at some center O of the image. In particular, we need only consider $|I|$ collections of k^2 elements, and take the largest sum of absolute values. This is in fact only a constant factor slower than the original LiRPA algorithm.

Finally, notice that we must ensure that our tighter bounds remain differentiable w.r.t the matrix A, so that we can use gradient-based optimization to find the optimal αs.

5 Implementing the proposed approach

Implementing the proposed solution would require changes in two places: the complete-verifier of the alpha-beta crown tool and the autolirpa tool.

Files in the α, β-CROWN tool:

- beta_crown_solver.py - input output has to be tweaked a little to allow the optimization of the changed final function
- abcrown.py - has to be changed to consider the new perturbation format.

Files in the `auto_LiRPA` tool:

- perturbation.py - to encode the new perturbation model.
- backward_bound.py, forward_bound.py, bound_general.py, and bound_ops.py must be changed to calculate the right bounds on the new objective function.

This list is not exhaustive. These are simply those found after a first inspection. In due progress of the project we might end up modifying other files as well.