UNIVERSITÉ DU LUXEMBOURG

Final Project: Neural Network Verification

Lattice Theory for Parallel Programming

Goals

- ✯ Understatnding neural network verification with abstract interpretation.
- ✯ Implementing bound propagation with different abstract domains on CPU/GPU.
- ✯ Send the code and report in "name1-name2.zip" before 14th of December 2025 by email to `yi-nung.tsao@uni.lu` and `pierre.talbot@uni.lu` with the subject "[MHPC][LTPP] Neural Network Verification Project".
- ✯ Don't share your source code
- ✯ This is a solo project or in a team of 3 students maximum.

Nowadays, neural networks are widely used in various applications, such as autonomous systems, healthcare, and many others. However, neural networks are still unable to make perfect decisions. For example, they remain highly vulnerable to adversarial examples. Adversarial examples refer to inputs that are initially classified correctly but lead to incorrect predictions when perturbed by slight, human-imperceptible noise. For instance, in an autonomous driving system, a neural network may correctly recognize a clean stop sign but fail to classify it accurately if a specific sticker pattern is added or if the brightness differs from the training dataset. Such misclassifications can result in critical errors, potentially causing accidents that endanger both the driver and other road users. Therefore, verifying if a neural network consistently produces robust outputs within a given input space is a crucial and necessary step before deploying it in safety-critical applications.

In recent years, several methods have been proposed for neural network verification. Most of these approaches are based on *abstract interpretation*, they are overapproximating the output bounds of each neuron in the network to make the verification process both efficient and practically feasible. In addition, there is a neural network verification competition in every year since 2020. This competition uses two standard input files with `vnnlib` and `onnx` formats. `vnnlib` is used to define the preconditions and postconditions and `onnx` provides network architecture and its trained parameters (weights/biases).

Exercise 1 – Warm-up

To have better understanding about verifying neural networks, we first build the state-of-the-art verifier, $\alpha - \beta-$`CROWN`, to know how it works.

- Install $\alpha - \beta-$`CRON` from https://github.com/Verified-Intelligence/alpha-beta-CROWN.
- Test $\alpha - \beta-$`CRON` with their provided examples.

Exercise 2 – Bound Propagation

As we have mentioned in the lecture, there are different abstract domains can be used in the bound propagation to verify neural networks. Your tasks are:

1. Implement 3 bound propagations with interval abstract domain, symbolic interval abstract domain, and deeppoly abstract domain, respectively, for verifying feed-forward ReLU neural networks. (You can use any C++ library for numerical operations such as OpenBLAS.)
2. Check the results from your implementation and $\alpha-\beta-$`CROWN` (we have provided a configuration file for `CROWN`, see `config.yaml`).
3. As we know, deeppoly abstract domain is more precise than others in terms of overapproximated area in relu function, does it mean that deeppoly abstract domain is always dominating other abstract domains? If not, please show an example from datasets to explain.
4. By using abstract interpretation to verify neural networks, we are still suffering from the imprecision of the abstract domains (false negative). How can you make those abstract domains be more precise? (Show the results to demonstrate the improvements.)
5. In CUDA/C++, implement parallel bound propagation with 3 abstract domains on GPU.
6. Define and formalize a new abstract domain for verifying feedforward relu neural networks or other network architectures. Prove that all abstract transformations are sound.
7. If we have known that a neural network is not robust, what can we do to improve its robustness?

Deliverable.
In the repository, we have provided 2 input parsers for loading `vnnlib` and `onnx` files.

- If you do not present this project in-class, please produce a short video (maximum 10 minutes), and give the linke of the upload in the README of the project. Present the main ideas and the formalization.
- Prove the soundness of 3 abstract domains for verifying neural networks in PDF file formatted using LaTeX.
- The parallel implementation of the algorithm on GPU, with benchmarking analysis and any other information you find relevant.
- Your program should be usable as follows:
  `./nnv [your_instance.csv] -o output.csv`

  You can add different options to enable/disable some optimizations, or propose different variants of the algorithm.

Input Parser.

- Vnnlib Input format.
  In `vnnlib` file, there are 3 components: declare variables, define preconditions, and define postconditions.
  Let the input layer has 2 neurons `X_0,X_1` and the output layer contains 2 neurons `Y_0,Y_1`. The preconditions define the bounds of input neurons from 1 to 2. The postconditions are either `Y_0` is greater than `Y_1` or `Y_1` is greater than `Y_0`. In this instance, the `vnnlib` file will be defined as follows:

```
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const Y_0 Real)
(declare-const Y_1 Real)

(assert (<= X_0 2))
(assert (>= X_0 1))
(assert (<= X_1 2))
(assert (>= X_1 1))

(assert (or
(and (<= Y_0 Y_1))
(and (<= Y_1 Y_0))))
```

- Onnx Input format.
  In the final project, we only consider feedforward relu neural networks. To describe such networks, we need to use those operators in `onnx` file in the following:
  - `Sub` and `Div`: They provide constant values for normalization on input vector.
  - `MatMul`: It provides a weight matrix $\boldsymbol{W}$: $\mathbb{R}^{\ell} \times \mathbb{R}^{\ell+1}$ for affine function.
  - `Add`: It provides a biase vector $\boldsymbol{b}$: $\mathbb{R}^{\ell+1}$ for affine function, this operator is usually used right after `MatMul`.
  - `Gemm`: It provides a weight matrix $\boldsymbol{W}$: $\mathbb{R}^{\ell+1} \times \mathbb{R}^{\ell}$ and a biase vector $\boldsymbol{b}$: $\mathbb{R}^{\ell}$ for affine function.
  - `Conv`: It will give us all parameters used in convolutional operation, this also can be viewed as `Gemm`.
  - `Relu`: It specifies the nonlinear activation function as relu.

  We have provided all necessary functions and cmake file for building both input parsers. Feel free to use it directly to implement your bound propagation. Note that for the vnnlib parser, we did the negation on postconditions, which means that the verification result is determined by checking if all lower bounds of output neurons are greater than 0. If so, return `UNSAT`. Otherwise return `SAT` and its adversarial example.

The Format of Output CSV File.
vnnlib file name, onnx file name, abstract domain, `UNSAT` or `SAT`, at least 1 adversarial example if any.
Please collect all results into a single `csv` file.

Reference.

- Naive Interval Domain: Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, ChongliQin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models, 2019.
- Symbolic Interval Domain: Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In 27th USENIX Security Symposium (USENIX Security 18), pages 1599–1614, Baltimore, MD, aug 2018. USENIX Association.
- DeepPoly: Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. Proc. ACM Program. Lang., 3(POPL), jan 2019.
- CROWN: Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, page 4944–4953, Red Hook, NY, USA, 2018. Curran Associates Inc.
- Neural Network Visualization Tool: https://netron.app/