CS433: Assignment 4

Evuri Mohana Sreedhara Reddy (23b1017)
Shaik Awez Mehtab (23b1080)

April 2025

1 Introduction

In this assignment, we explored how to use an SMT solver to analyze the robustness of a neural network. The main idea was to encode a trained network as a set of mathematical constraints. Given an input image, we aimed to find the maximum L2 perturbation allowed on the input while ensuring that the model's output label stays the same. This can help us understand how sensitive or stable the network is to small changes in the input.

2 Dataset and Preprocessing

We used the CIFAR-10 dataset for this task. Since working with full-size images in an SMT solver would be too slow, we preprocessed the data to reduce its size. We converted all images to grayscale and resized them to 8×8. This gave us a manageable input size of 64 pixels per image.

We used the following preprocessing steps with torchvision:

- `Grayscale()` to remove color channels,
- `Resize((8, 8))` to shrink the image,
- `ToTensor()` to turn the image into a PyTorch tensor.

3 Model Architecture and Training

We built a small fully connected feedforward neural network using the torch library. The architecture had three layers:

- Input layer with 64 neurons (one for each pixel),
- Two hidden layers with 8 neurons each,
- Output layer with 10 neurons, one for each class in CIFAR-10.

We used ReLU after each hidden layer and ArgMax to decide the final class. This satisfies the requirement that the network only use ReLU and ArgMax as nonlinear components.

We trained the model using the Adam optimizer and cross-entropy loss. The training ran for 5 epochs with a batch size of 64. Training was quick because of the small model and reduced image size. After training, we picked one test image to verify.

4 Encoding the Neural Network for SMT

We encoded the trained network into constraints using the Z3 solver to verify robustness. We defined real-valued variables for both the original input and the perturbed version. We added constraints for each pixel so that the perturbed pixel must stay in the range [0, 1].

We used the L2 norm to measure how much the image is perturbed. We added the constraint:

$$\sum_i (\text{perturbed}_i - \text{original}_i)^2 \leq \delta^2$$

This means that the total squared difference between the original and perturbed images must be less than or equal to the chosen bound δ^2.

The network's layers were encoded as matrix operations using the trained weights and biases. We computed the weighted sum of the inputs for each neuron and applied the ReLU using the `If` function provided by Z3. At the output layer, we added constraints to make sure that the predicted class (the one the network gives on the original image) still has the highest score after perturbation:

$$\text{logit}_{\text{original}} \geq \text{logit}_j \quad \text{for all } j \neq \text{original}$$

This SMT encoding was written to a file called `nn_encoding.smt2`, which we then ran through Z3 to check satisfiability.

5 Experiments and Observations

We tested this setup using a few examples from the test set. For each image, we changed the L2 perturbation bound δ and recorded whether Z3 returned `SAT`, `UNSAT`, or `UNKNOWN`.

For a sample image with label 9, we observed the following:

- For small values like $\delta = 0.1$ or 0.3, Z3 quickly returned `SAT`, confirming the network's prediction stayed the same under small changes.
- As we increased δ, the solver continued to return `SAT` but started taking longer. Around $\delta = 0.9$, the time jumped noticeably.
- After that, the solver either took too long or returned `UNKNOWN`.

This tells us the model is fairly robust to small changes in the input, but the solver struggles with larger bounds. One reason is that many more perturbed inputs are possible with a large enough bound, making the search space harder for Z3 to handle.

Another detail we noticed was that if we only add the upper bound on the perturbation, the solver can just pick the original input again (zero perturbation), which still satisfies the constraint. To avoid this, we sometimes also added a small lower bound:

$$\sum_i (\text{perturbed}_i - \text{original}_i)^2 \geq \epsilon$$

with $\epsilon = 10^{-6}$, just to ensure the perturbation isn't exactly zero. But for the main robustness test, we focused only on the upper bound, since the goal was to check if any input within that limit can still be classified the same.

6 Conclusion

We used the torch library to train a small neural network on a preprocessed CIFAR-10 dataset and encoded the trained model using Z3. We tested whether the model's predictions remain unchanged for different levels of L2 perturbation on the input.

For small perturbations, the SMT solver could return `SAT` quickly, showing that the model is robust in those regions. As the allowed perturbation increased, the solver started taking more time and eventually couldn't give a clear answer.

This experiment shows how SMT solvers can help with formal verification of neural networks, especially for small models. However, they still have trouble scaling to more complex inputs or larger perturbation bounds.

Files Submitted

- `nn_encoding.smt2`: SMT-LIB encoding of the model.
- `A4.ipynb`: Code used to train the network and generate the SMT encoding.
- `report.pdf`: This report.