ECE/CS 584: Embedded and cyberphysical system verification	Spring 2024
Your name, netid	*Homework 2*
	Due Mar 10th 11:59pm

Typeset your solutions using LaTeX zip your solutions in a single file called `netid-584-S24.zip` *and upload this file through Canvas.*

Problem 1: Verification Requirements for Multi-class classification (20 pts)

Consider a classification neural network $\boldsymbol{y} = f(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{y} \in \mathbb{R}^m$. The network predicts m scores for m labels (e.g., in a traffic sign recognition neural network, the labels can be "stop sign", "speed limit sign", "yield sign", etc), and the label with the greatest score (top-1 score) is the final predicted class of the model.

In our lecture, we want to verify that *the prediction score of the "stop sign" is greater than that of the "speed limit sign" for all* $\boldsymbol{x} \in \mathcal{S}$. If we denote the score of "stop sign" to be y_i and the score of "speed limit sign" is y_j, we essentially want to prove that $y_i - y_j > 0$ for all $\boldsymbol{x} \in \mathcal{S}$ (here $y_i := [f(x)]_i$ and $y_j := [f(\boldsymbol{x})]_j$). In this example, we formulate this problem as a satisfiability problem, which may be checked using an SMT solver:

$\downarrow$
可满足性

$$\exists \boldsymbol{x},\ \boldsymbol{x} \in \mathcal{S} \wedge (\boldsymbol{y} = f(\boldsymbol{x})) \wedge (y_i - y_j \leq 0) \tag{1}$$

We verify our requirement if an SMT solver reports "unsatisfiable" for the above feasibility problem. If an SMT solver finds a satisfiable solution $\boldsymbol{x}_{\text{sat}}$, then $\boldsymbol{x}_{\text{sat}}$ is a counterexample where the requirement $y_i - y_j > 0$ for $\boldsymbol{x}_{\text{sat}} \in \mathcal{S}$ does not hold.

In our lecture, we discussed how to use optimization-based formulation to solve the verification problem. For example, the satisfiability problem in (1) can be solved with the following optimization problem:

$$\min_{\boldsymbol{x} \in \mathcal{S}} g(\boldsymbol{x}) := y_i - y_j \tag{2}$$

If the objective function $g(\boldsymbol{x})$ becomes non-positive at the global optimal solution $\boldsymbol{x}^*$, the requirement does not hold since $\boldsymbol{x}^*$ is a counter-example where $y_i - y_j \leq 0$. Otherwise, the requirement is verified.

Now, you are asked to write the satisfiability problem (in a similar manner as in (1)) for verifying the following requirements:

$\exists x, x \in S \wedge (y = f(x)) \wedge (y_i - y_j \leq 0)$

1. The prediction score for label i is always the top-1 score for all $x \in \mathcal{S}$ (**5pts**)

2. The prediction score for label j can never be the top-1 score for all $x \in \mathcal{S}$ (**5pts**)

$\exists x, x \in S \wedge (y = f(x)) \wedge (y_i - y_j \leq 0)$

In addition, write the optimization problem that can be used to solve the above two verification requirements by redefining $g(\boldsymbol{x})$ (**10pts**). $\min_{x \in S} g(x) = y_i - y_j$

Problem 2: CROWN Bound propagation on Non-feedforward Neural Network (25pts)

In this problem, we consider a simple neural network with a "skip" connection:

$$\begin{aligned}
y &= \boldsymbol{W}^{(3)}(\hat{\boldsymbol{z}}^{(2)} + \boldsymbol{z}^{(1)}) \\
\hat{\boldsymbol{z}}^{(2)} &= \text{ReLU}(\boldsymbol{z}^{(2)}) \\
\boldsymbol{z}^{(2)} &= \boldsymbol{W}^{(2)}\hat{\boldsymbol{z}}^{(1)} + \boldsymbol{b}^{(2)} \\
\hat{\boldsymbol{z}}^{(1)} &= \text{ReLU}(\boldsymbol{z}^{(1)}) \\
\boldsymbol{z}^{(1)} &= \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}
\end{aligned}$$

Here the input $\boldsymbol{x} \in \mathbb{R}^2$ and $y \in \mathbb{R}$. The input set $\mathcal{S} = \{\boldsymbol{x} | -1 \leq x_1 \leq 1, -1 \leq x_2 \leq 1\}$. The network weights as:

$$\boldsymbol{W}^{(1)} = \boldsymbol{W}^{(2)} = \begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}, \ \boldsymbol{W}^{(3)} = \begin{bmatrix} -1 & 1 \end{bmatrix}, \ \boldsymbol{b}^{(1)} = \begin{bmatrix} 1 & 1 \end{bmatrix}, \ \boldsymbol{b}^{(2)} = \begin{bmatrix} 2 & 2 \end{bmatrix} \tag{3}$$

In CROWN, since the choice of α (slope of the linear lower bound of ReLU: $\hat{z}_j^{(i)} \geq \alpha_j^{(i)} z_j^{(i)}$) will lead to different answers, in this problem, we will use the following "adaptive" rule to choose $\alpha_j^{(i)}$ for the linear lower bound of an unstable neuron j in the i-th layer:

$$\alpha_j^{(i)} = \begin{cases} 1, & \text{if } |u_j^{(i)}| \geq |l_j^{(i)}| \\ 0, & \text{if } |u_j^{(i)}| < |l_j^{(i)}| \end{cases} \tag{4}$$

Here $l_j^{(i)}$ and $u_j^{(i)}$ are the preactivation bounds (i.e., $l_j^{(i)} \leq z_j^{(i)} \leq u_j^{(i)}$ for $\boldsymbol{x} \in \mathcal{S}$). Now, answer the following questions. To get full credit, please show intermediate steps and linear equations (if applicable) that lead to your answer instead of just the final results.

1. Use interval bound propagation (IBP) to calculate the lower bound of y (**5pts**)
2. Use CROWN to calculate the intermediate layer lower and upper bounds of $\boldsymbol{z}^{(2)}$. You will need these bounds to obtain linear relaxations for $\hat{\boldsymbol{z}}^{(2)} = \text{ReLU}(\boldsymbol{z}^{(2)})$ in your next step. (**5pts**)
3. Using the intermediate layer bounds you obtained in step 2, calculate CROWN lower bound on the output y. (**10pts**)
4. Now we consider optimizing α to tighten the lower bound of y for this neural network. Assuming all ReLU neurons are unstable, how many αs can we use to get the tightest possible

lower bound of y? (hint 1: tighter intermediate layer bounds can lead to a tighter bound on y, and intermediate layer bounds can be optimized independently; hint 2: to obtain the tightest lower bounds and upper bounds, two sets of αs can be used, one for the lower bounds and one for the upper bounds). (**5pts**)

Problem 3: Implementing the Mixed Integer Programming and Linear Programming Formulations for Neural Network Verification (30 pts)

Code repository: `https://github.com/huanzhang12/ECE584-SP24-assignment2`

In our lecture, we discussed the linear programming formulation and mixed integer linear programming formulation for neural network verification. In this problem, you will implement these formulations on a simple neural network.

The code loads a simple pretrained model with three linear layers, and one test image (a handwritten digit, 0 - 9) from the MNIST dataset. The test image $\boldsymbol{t} \in \mathbb{R}^{784}$ contains 28×28 pixel values. Element-wise perturbation with a given perturbation size is added to each pixel: e.g., when the perturbation size is s, the input set $\mathcal{S} := \{\boldsymbol{x} \mid t_i - s \leq x_i \leq t_i + s, \forall i \in [1, 784]\}$. The neural network predicts scores $\boldsymbol{y} \in \mathbb{R}^{10}$, and y_i represents the score for the image being recognized as a digit i. If the label of test image $\boldsymbol{t}$ is c, we want to verify that y_c is always the top-1 score for all $\boldsymbol{x} \in \mathcal{S}$ by solving 9 optimization objectives.

You can run code template `mip_assigment.py` with an input data `data1.pth` by running `python3 mip_assigment.py data1.pth`. The README document in the repository provides additional instructions for running the code.

You will use the Gurobi optimizer to solve MILP and LP problems. Your code needs to formulate the verification problem using the Python interface of Gurobi. You can refer to the Gurobi Python documentation available at this link.

Note that MILP and LP formulations require the pre-activation intermediate layer bounds (upper bound $u_j^{(i)}$ and lower bound $l_j^{(i)}$). In the given implementation, these bounds need to be computed using the MILP/LP formulation as well, by treating $z_j^{(i)}$ as the optimization objective.

Now, complete the following tasks:

1. Complete the code for the MILP formulation using the given template. The parts that you need to finish have been marked with TODOs in source files. Using the MILP formulation, you should obtain an exact solution (global minimum) for optimization problems. Submit your completed program `mip.py`. (**10pts**)

2. Change the code and allow the integer variables ($p_j^{(i)}$ using the notation in our lecture) to take continuous values from 0 to 1, which gives the LP formulation. Using the LP formulation, you can only obtain a lower bound for the verification objective. Submit your completed program as `lp.py`. (**10pts**)

3. Using the data *data1.pth*, collect the exact solution found by the MILP formulation and the lower bound found by the LP formulate for perturbation size $s = \{0, 0.001, 0.003, 0.01, 0.03, 0.1\}$.

Plot or list the results showing the gap between the exact solution and the lower bound at different perturbation sizes for all 9 optimization objectives. If program does not finish within 10 minutes, you can report "timeout". (**10pts**)

Problem 4: Implementing CROWN for a New Activation Function (25pts)

Code repository: `https://github.com/huanzhang12/ECE584-SP24-assignment2`

In our lecture, we discussed how to propagate CROWN linear bounds through a non-linear function in a neural network, such as a ReLU function. In this problem, you will work on develop bound propagation over a network with `hardtanh` activation function defined as:

$$\text{hardtanh}(z) = \begin{cases} -1, & z < -1 \\ z, & -1 \le z \le 1 \\ 1, & z > 1 \end{cases} \tag{5}$$

To help you, we provide a reference implementation of the CROWN algorithm for feedforward ReLU networks. The hardtanh network you will be working on has the same architecture as the ReLU network, so you just need to focus on changing the activation function part.

You can run the ReLU reference code using `python3 crown.py data1.pth`. The network used is the same as the previous question, and the definition of the input set $\mathcal{S}$ is also the same. `data1.pth` gives the test input $\boldsymbol{t}$. The program, by default, will print the lower and upper bounds for all 10 output neurons given this input with perturbation size $s = 0.01$ (changeable in the code).

1. Please carefully read the provided code files, and understand how CROWN is implemented. **(0pts)**
2. Derive the linear lower and upper bounds for hardtanh(z) given the preactivation bounds $l \le z \le u$. Use the tightest linear bounds when possible, and when there are multiple possible solutions (like the linear lower bound for ReLU), you can make the lower and upper bounds use the same slope. (Hint: you will need to consider different segments of the function, see Table 2, Table 3, Figure 1 and the case studies in the CROWN paper). **(10pts)**
3. Implement CROWN for hardtanh network using the provided code template. **(15pts)**
4. (Bonus) Discuss how to allow bound optimization (α-CROWN) for a hardtanh neural network. **(bonus 5pts)**
5. (Bonus) Implement α-CROWN with optimized bounds for hardtanh and report its improvements. **(bonus 15pts)**