ADM - Adversarial Instances

Davide Davoli

January 15, 2022

1 Introduction

The goal of this project is to implement a MILP model to generate adversarial instances that can confuse a NN to report wrong results. The model is being used to test the robustness of neural networks trained to solve an image classification task. The code of the project can be found at https://github.com/davidedavo/MILP-Adversarial-Instances.

2 PyTorch to MILP conversion

The MILP model is created converting PyTorch model into a MILP one. The MILP model can support several NN layers as: Conv2d (+ ReLU), Linear (+ ReLU), MaxPool2D, Flatten.

2.1 2D Convolution

Constraints:

- $padding = 0$
- $stride = 1$

Let $y \in \mathbb{R}^{KC \times OH \times OW}$ be the output of the layer. The Conv2d can be implemented adding the following constraints to the MILP model:

$$Conv2d + ReLU: \begin{cases} \sum_{ic=1}^{IC} \sum_{kh=1}^{KH} \sum_{kw=1}^{KW} k^c[ic, kh, kw] \cdot x[ic, h+kh, w+kw] = y[c, h, w] - s[c, h, w] \\ z[c, h, w] = 0 \rightarrow s[c, h, w] \leq 0 \\ z[c, h, w] = 1 \rightarrow y[c, h, w] \leq 0 \\ z[c, h, w] \in \{0, 1\} \end{cases}$$

$$Conv2d: \quad \sum_{ic=1}^{IC} \sum_{kh=1}^{KH} \sum_{kw=1}^{KW} k^c[ic, kh, kw] \cdot x[ic, h+kh, w+kw] = y[c, h, w] \tag{1}$$

where $k^c \in \mathbb{R}^{IC \times KH \times KW}$ is the kernel tensor of the c^{th} output channel; $c = 1, ..., KC$; $h = 1, ..., OH$; $w = 1, ..., OW$. $x \in \mathbb{R}^{IC \times IH \times IW}$ is the input of the layer. If the current layer is the first one, x is the input tensor, otherwise is the activation map of the previous layer. Output dimensions are calculated as follows:

$$OH = IH - KH + 1 \qquad\qquad OW = IW - KW + 1. \tag{2}$$

2.2 2D Max Pooling

Constraint: $padding = 0$

Let $x \in \mathbb{R}^{C \times IH \times IW}$; $y \in \mathbb{R}^{C \times OH \times OW}$ be the input and output of the layer respectively. The MaxPooling2d of kernel size k and stride s can be implemented adding the following constraint to the MILP model:

$$MaxPool2d: \max_{i,j \in N_k(h,w)} x[c, i, j] = y[c, h, w] \tag{3}$$

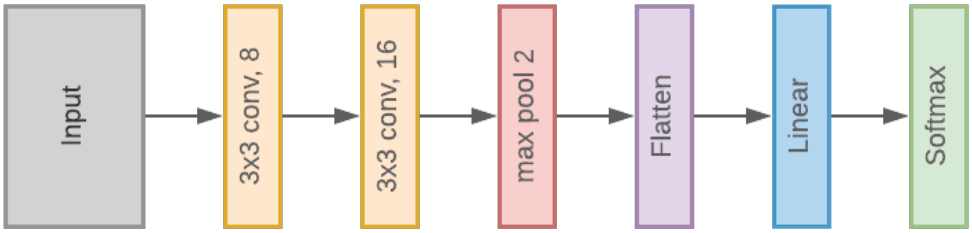

Figure 1: CNN classifier architecture. The ReLU after each Conv layer is not shown in figure.

where $c = 1, ..., C$; $h = 1, ..., OH$; $w = 1, ..., OW$; $N_k(h, w) = \{(x, y) : x \in [h \cdot s, h \cdot s + k], y \in [w \cdot s, w \cdot s + k]\}$. The output dimensions are calculated as follows:

$$OH = \left\lfloor \frac{IH - k}{s} \right\rfloor + 1 \qquad OW = \left\lfloor \frac{IW - k}{s} \right\rfloor + 1. \tag{4}$$

This constraint is implemented using the `max_()` Gurobi function

2.3 Linear

Let $x \in \mathbb{R}^{IF}$ be the input of the linear layer, where IF is the number of input features[1].$y \in \mathbb{R}^{OF}$ is the output of the layer, where OF is the number of otuput features of the layer. The Linear layer can be implemented adding the following constraints to the MILP model:

$$Linear + ReLU: \begin{cases} \sum_{i=1}^{IF} w[i,j] \cdot x[i] = y[j] - s[j] \\ z[j] = 0 \rightarrow s[j] \leq 0 \\ z[j] = 1 \rightarrow y[j] \leq 0 \\ z[j] \in \{0, 1\} \end{cases} \tag{5}$$

$$Linear: \qquad \sum_{i=1}^{IF} w[i,j] \cdot x[i] = y[j]$$

where $j = 1, ..., OF$; $w \in \mathbb{R}^{IF \times OF}$ is the weights matrix of the layer.

2.4 Flatten

The flatten layer is supported only for Linear layers whose input are multi-dimensional activation maps. This is accomplished by automatically converting one-dimensional indices to multi-dimensional indices. This process is completely "transparent" to the Linear layer which will always use mono-dimensional indices to access at its input.

3 The PyTorch model

In figure 1 is shown the architecture of the CNN classifier. The model is composed by: (i) two 3x3 Convolutional layer followed by ReLU activations (ii) Max Pooling layer of kernel_size=2 and stride=2 (iii) Linear layer followed by a Softmax activation.

4 Setting up the MILP model

The MILP model is implemented converting the PyTorch model as shwon in section 2. The output of this process is a set of variables and constraints as illustrated in section 2. We represent the activation

[1]

variables as x^l, where $l = 0$ denotes the input of the network and $l = 1, ..., L$ is the index of the current linear or convolutional layer. x^l can have different shapes:

- $x^l \in \mathbb{R}^{N_l}$ if the l^{th} layer is a **linear** one. N_l is the number of neurons in layer l.
- $x^l \in \mathbb{R}^{C_l \times H_l \times W_l}$ if the l^{th} layer is a **convolutional, max pooling or input** layer. C_l, H_l, W_l, are respectively the number of channels, height and width of the output of layer l (or of the input image if $l = 0$).

4.1 Inference model

The inference model is implemented adding the following objective function:

$$min \sum_{l=1}^{L} \sum_{z \in x^l} z \tag{6}$$

5 Results

Three different solutions have been tried:

1. MILP model **without last linear layer**: Finds the correct solution (compared with the solution obtained using the PyTorch model). The output of this execution can be found here
2. MILP model **with last linear layer**: Starts performing the branch and bounds technique without finding any results in more than 4h of execution. The output of this execution can be found here
3. MILP model **with last linear layer and additional ReLU** [2]: Starts performing the branch and bounds technique finding a feasible solution in few shots. The output of this execution can be found here

6 Conclusions

Adding the linear layer (without the ReLU activation constraints) to the MILP model, makes the model unfeasible or very difficult to be solved by Gurobi.

[2]The additional ReLU is not included in the PyTorch model, but used for debug purposes