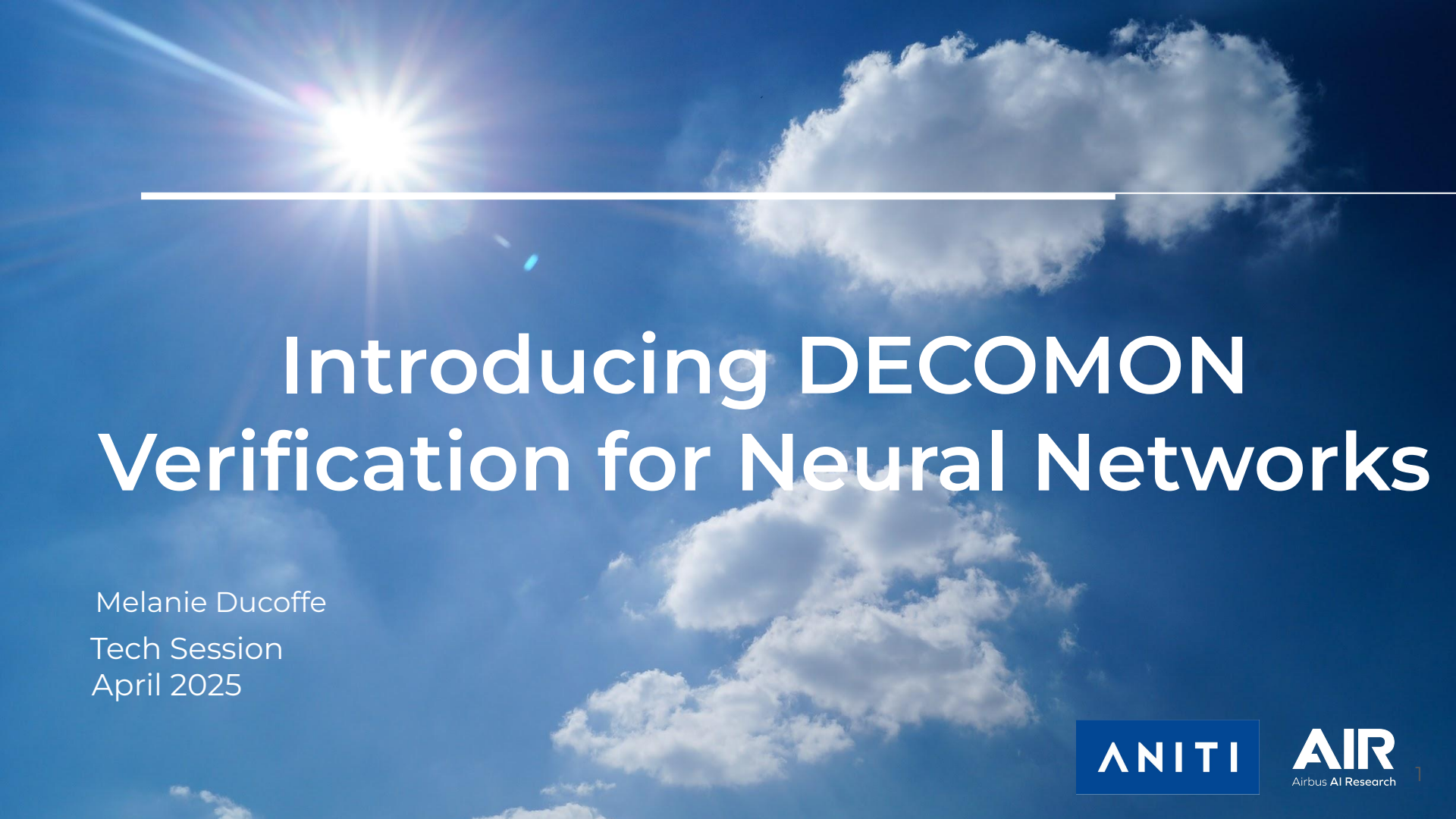


# Introducing DECOMON Verification for Neural Networks

Melanie Ducoffe

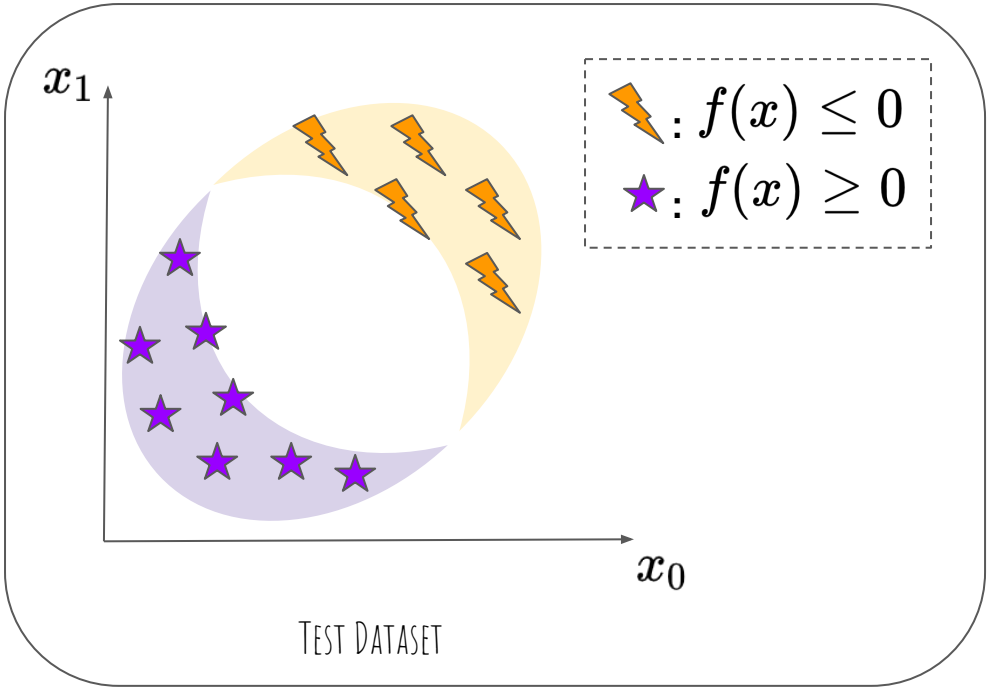
Tech Session
April 2025

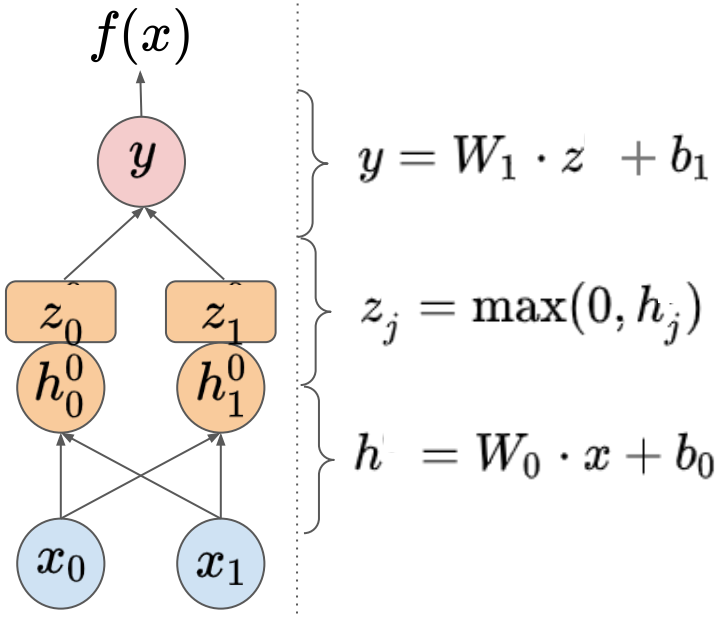
ANITI

AIR Airbus AI Research

AIRBUS

## Let's start with a naive example

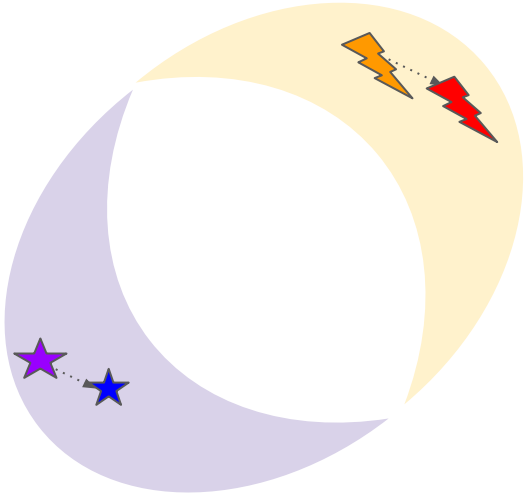


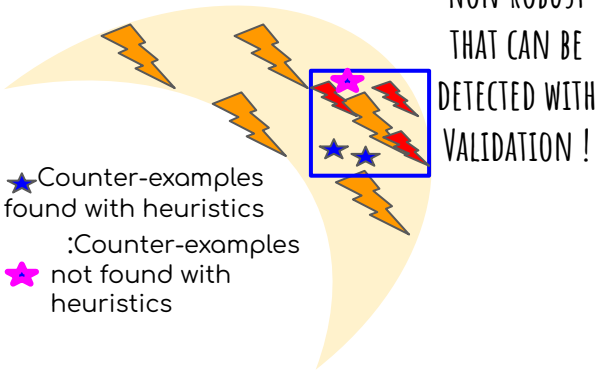
# Define a property: Local Robustness

AIRBUS

Human decisions are locally stable… so should be the decision of a machine learning model

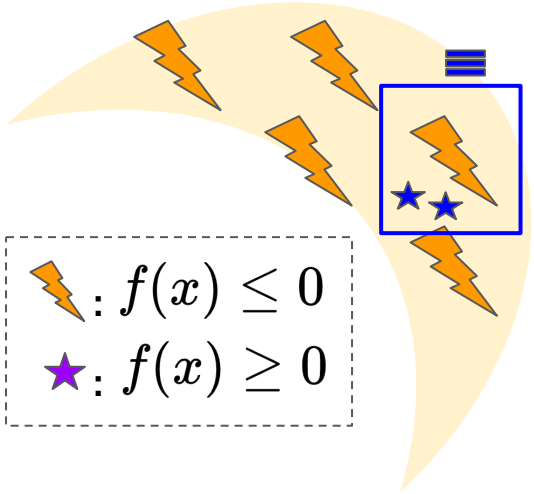



**Local Robustness** is one property among others used in this context to illustrate verification of AI models.
<u>All the existing methods can be adapted to other specifications.</u>




- Finding a counter-example is easier than demonstrating a false property
- Adversarial examples (White box attack) is a way to test your property

# Local Robustness as a Relaxed Optimization Problem



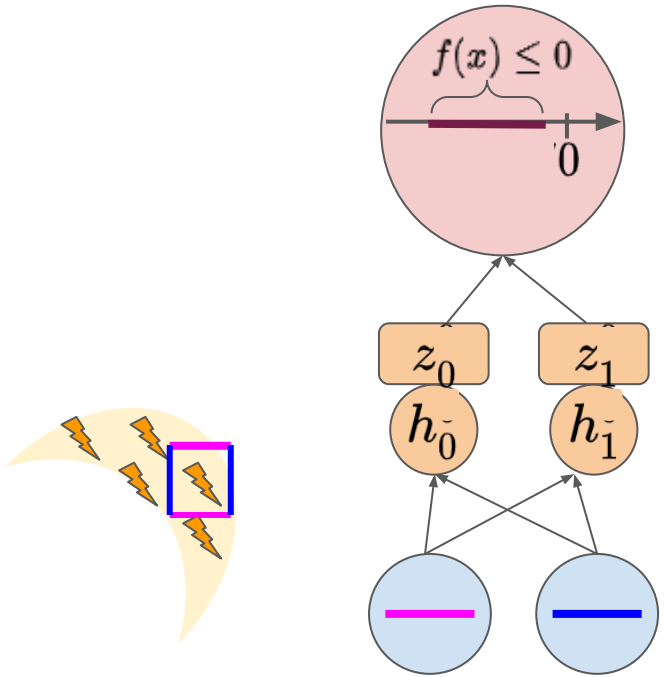

Robustness

Given an input domain $\Omega$ $\nexists x \in \Omega$ s.t $f(x) > 0$

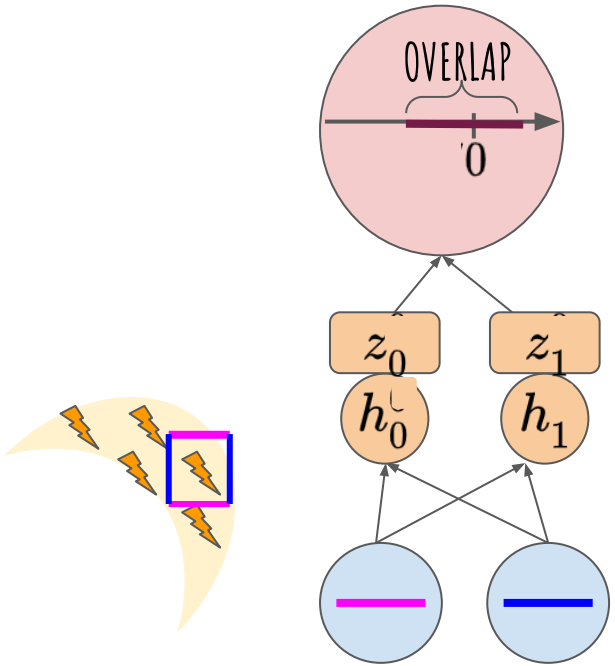
$$\max_{x \in \Omega} \hat{f}(x) \leq 0 \;\&\; \forall x \in \Omega \; f(x) \leq \hat{f}(x)$$

outer-approximations that only implies robustness:
**Linear Relaxation** [CROWN]
= Abstract Interpretation [DeepPOLY]

# Incomplete Formal Verification

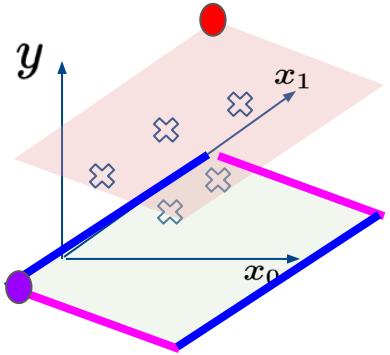



CASE 1: ROBUST



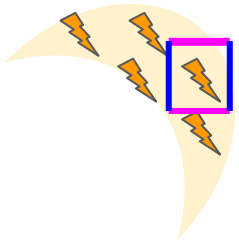

CASE 2: MAYBE ROBUST

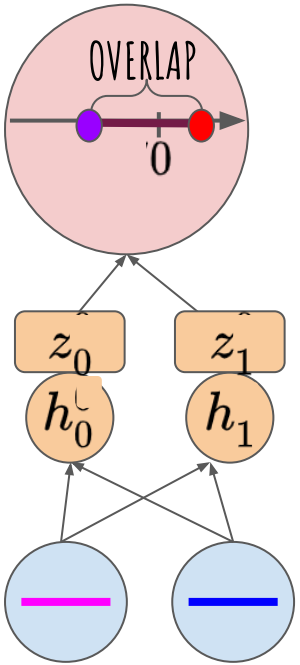
# Linear Relaxation




two hyperplanes that bound the output

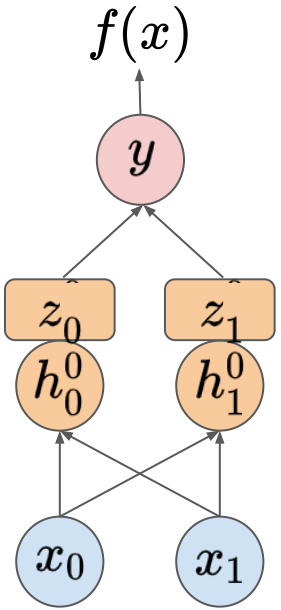
Different ‘recipes’ in the litterature that balance efficiency and scalability






CASE 2: MAYBE ROBUST

AIRBUS




$$y = W_1 \cdot z + b_1$$

$$z_j = \max(0, h_j^i)$$

$$h = W_0 \cdot x + b_0$$

**Pre-requisite: Bound the output of a layer by two affine bounds given the layer's input**

$$W_1 \cdot z + b_1 \leq y \leq W_1 \cdot z + b_1$$

$$\underline{A} \cdot h + \underline{a} \leq z \leq \bar{A} \cdot h + \bar{a}$$

$$W_0 \cdot x + b_0 \leq h \leq W_0 \cdot x + b_0$$

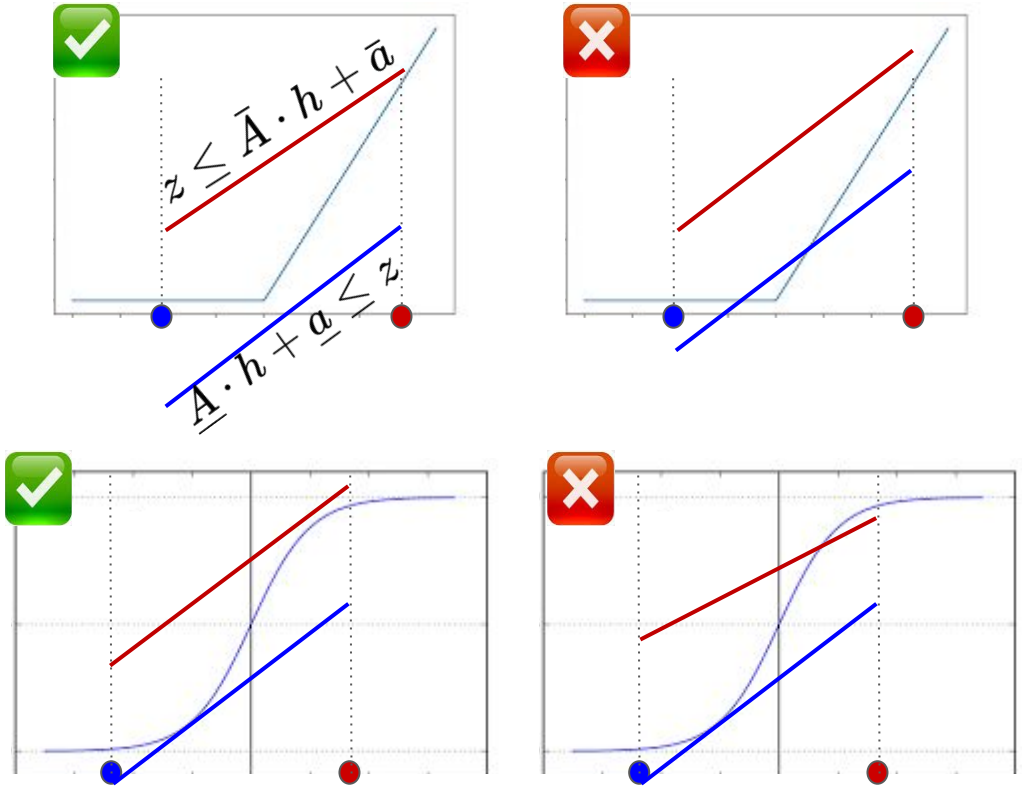
*Immediate* for Linear Layers (Dense, BatchNorm, Conv, Reshape…)

More tricky for NON Linear (ReLU, MaxPooling,…)

AIRBUS

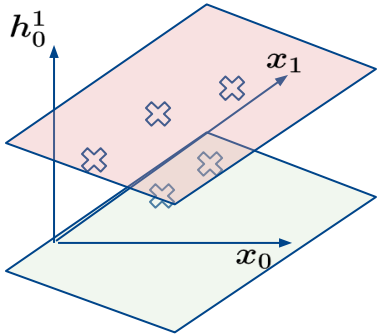
Pre-requisite: an interval for the layer's input [ ● , ● ]

Some affine bounds are more useful (closer to the layer's output)
Existing heuristics/tuning scheme exist



# Forward propagation rules

AIRBUS

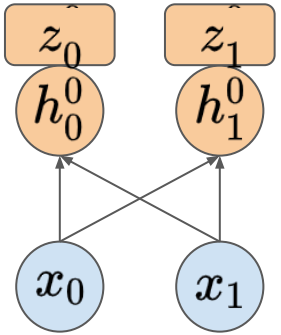



**FORWARD methods:**

- **IBP:** forward propagation of CONSTANT bounds
- **FORWARD:** forward propagation of AFFINE bounds
- **HYBRID:** forward propagation of CONSTANT bounds

TIGHTER

**Golden rule: Multiplying both side by a negative number reverses the inequality**

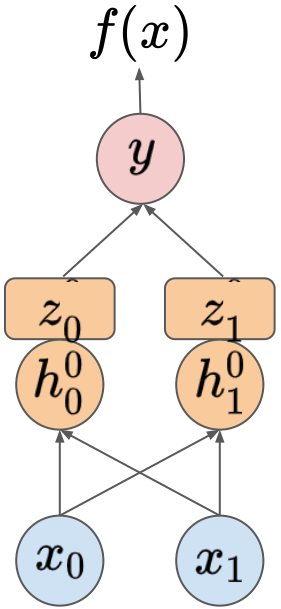



$$\underline{A} \cdot h + \underline{a} \leq z \leq \bar{A} \cdot h + \bar{a}$$

$$\underline{W}_0 \cdot x + \underline{b}_0 \leq h \leq \bar{W}_0 \cdot x + \bar{b}_0$$

$$z \leq \bar{A}^{\geq 0}\left(\bar{W}_0 \cdot x + \bar{b}_0\right) + \bar{A}^{\leq 0}\left(\underline{W}_0 \cdot x + \underline{b}_0\right) + \bar{a}$$

# Backward (CROWN/DEEP POLY) propagation rules

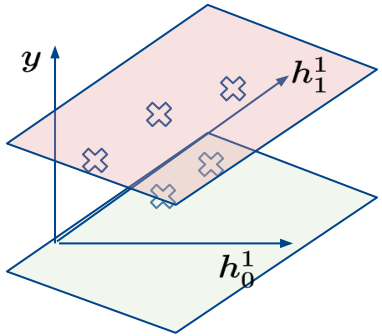



$$\underline{W}_1 \cdot z + \underline{b}_1 \leq y \leq \bar{W}_1 \cdot z + \bar{b}_1$$

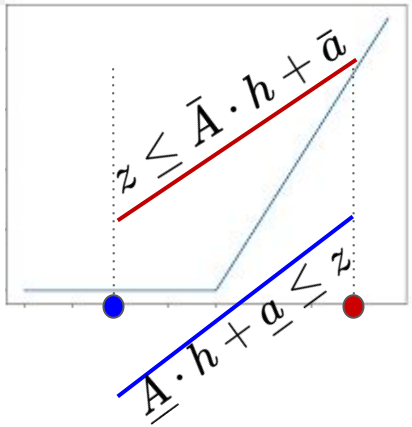
$$\underline{A} \cdot h + \underline{a} \leq z \leq \bar{A} \cdot h + \bar{a}$$

$$y \leq \bar{W}_1^{\geq 0}\left(\bar{A} \cdot h + \bar{a}\right) + \bar{W}_1^{\leq 0}\left(\underline{A} \cdot h + \underline{a}\right) + \bar{b}_1$$

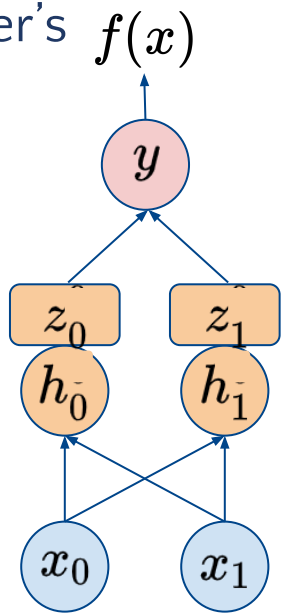
a simple change of variable for linear layers $h = W_0 \cdot x + b_0$



# Oracle for NON-linear layers




Pre-requisite: an interval for the layer's input [ , ]

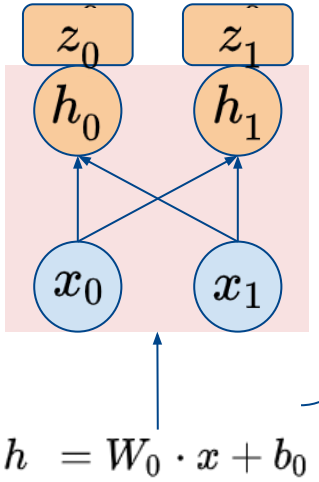



Upper and Lower bounds are required beforehand
With forward propagation, they are computed on the fly

For backward propagation, we need an ORACLE:

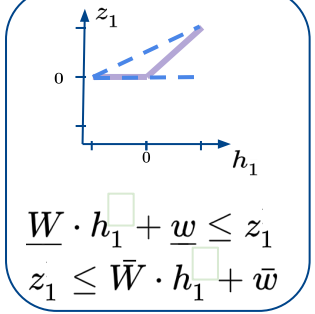
- a forward propagation: CROWN-{IBP, FORWARD, HYBRID}
- Recursive calls of backward prop on sub networks: CROWN

$$f(x) \leq W_1 \cdot z + b_1$$
$$W_1 \cdot z + b_1 \leq f(x)$$




CROWN :

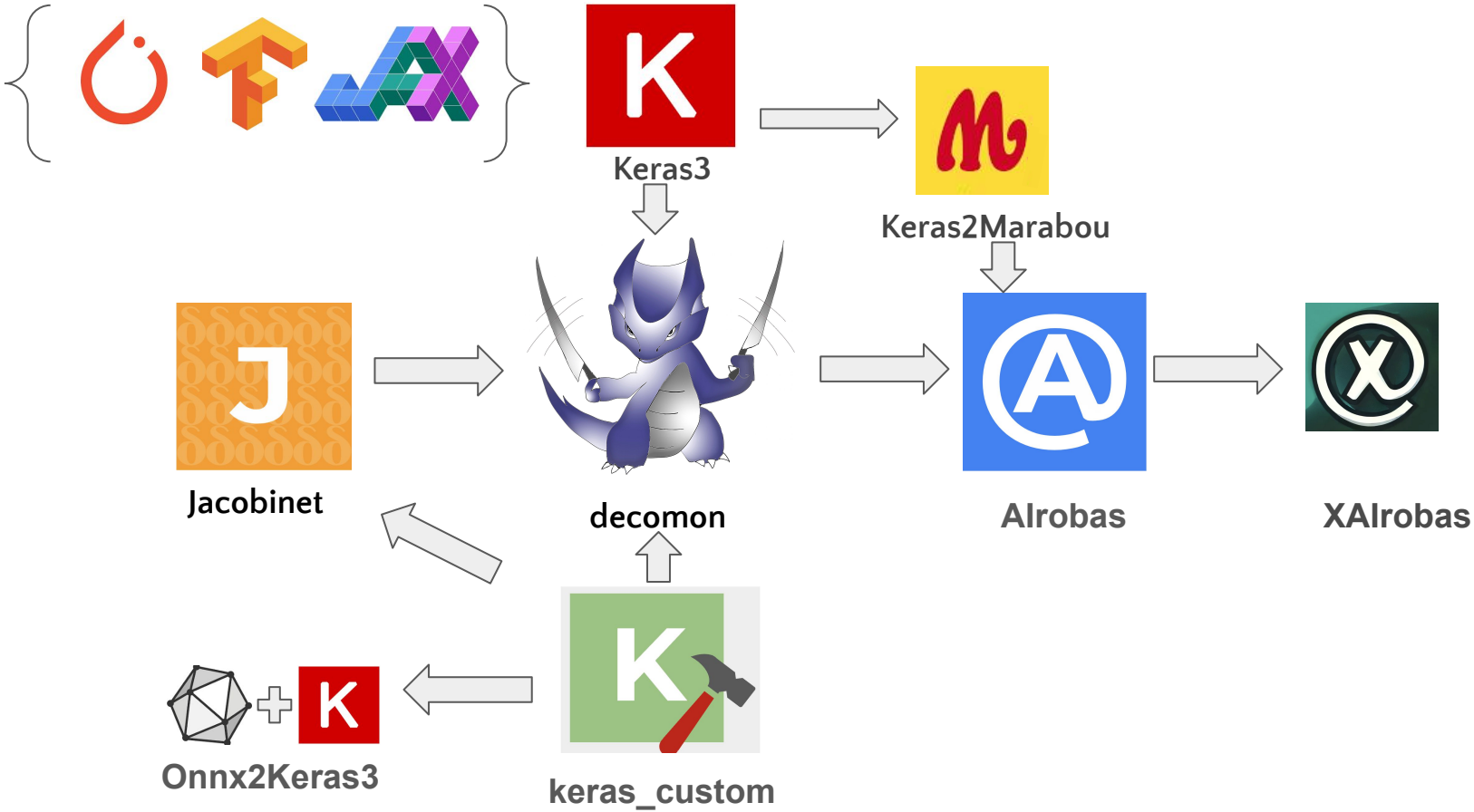
$$l_h \leq h \leq u_h$$



From theory to practice ...

# A Verification Ecosystem in ANITI

AIRBUS



# Getting Started

AIRBUS

GOTO: https://github.com/airbus/decomon@refactor

Requirements: Keras3, Jacobinet, KerasCustom

Installation:

```
pip install
git+https://github.com/airbus/decomon@refactor
```

Tutorials on jupyter notebook (tutorials) or google colab

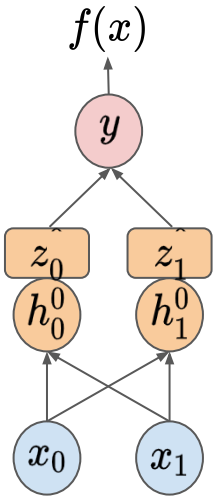
AIRBUS

**TUTORIAL 1**

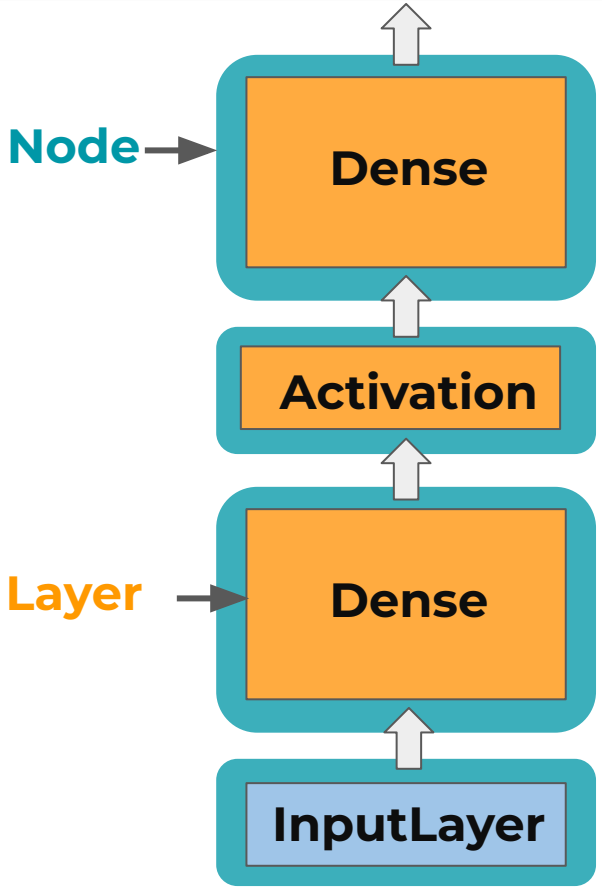
Bounding the output of a Neural Network trained on a sinusoidal function

- Apply LiRPA

# Computational Graph

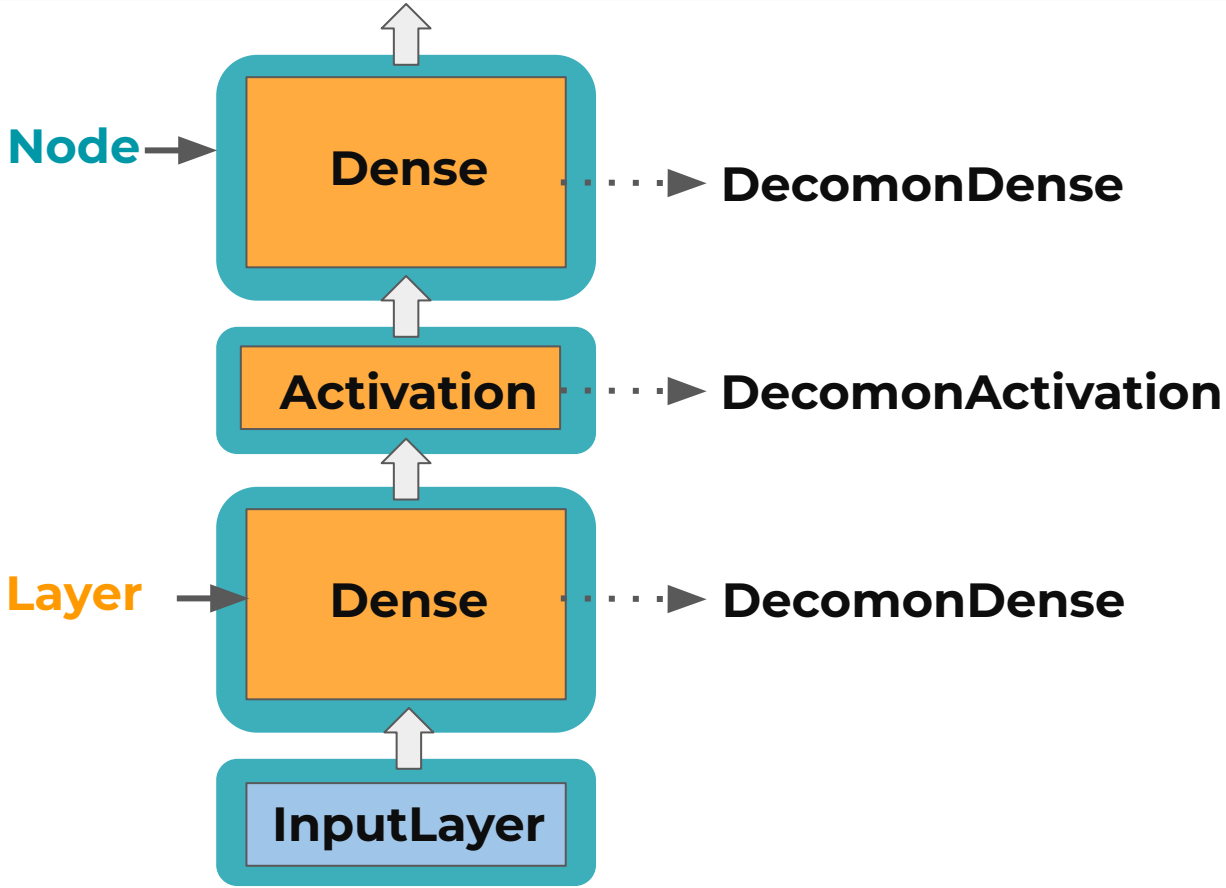



```keras
from keras.layers import Input, Dense, Activation
from keras.models import Model

x = Input((2,))
h = Dense(2)(x)
z = Activation('relu')(h)
y = Dense(1)(z)

f = Model(x, y)
```



# From Computational Graph to Decomon

AIRBUS



```python
class DecomonLayer(Wrapper):

    linear: bool = False

    diagonal: bool = False

    increasing: bool = False

    decreasing: bool = False

    def get_affine_representation(self):
      ...

    def get_affine_bounds(self, lower: Tensor, upper: Tensor, **kwargs):
      ...

    def forward_ibp_propagate(self, lower: Tensor, upper: Tensor):
      ...

    def forward_affine_propagate(
        self, input_affine_bounds: list[Tensor], input_constant_bounds: list[Tensor]
    ):
      ...

    def backward_affine_propagate(
        self, output_affine_bounds: list[Tensor], input_constant_bounds: list[Tensor], **kwargs
    ):
      ...
```

AIRBUS

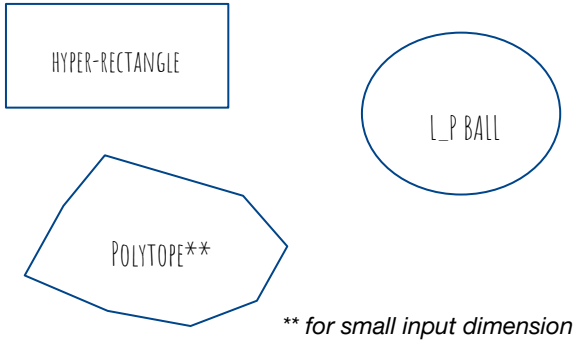
# TUTORIAL 2

Graph Computation and
Custom Operators

# Perturbation Domain

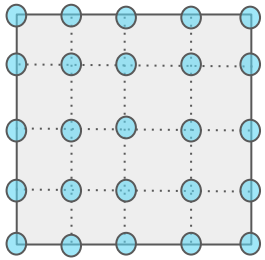
AIRBUS

## Convex hull
## Worst case perturbation




** *for small input dimension*

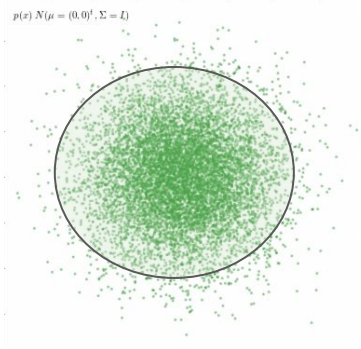
## Discrete Convex hull



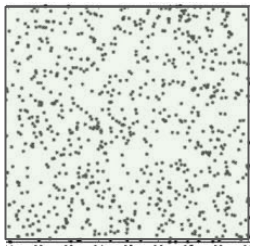
## Distribution over convex hull
## Gaussian, uniform... [proven]

*probabilistic approach can tighten up robustness certificate to around 1.8× and 3.5× with at least a 99.99% confidence compared with the worst-case robustness*

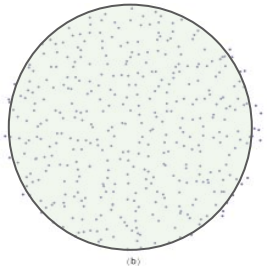
$$\gamma_L \leq \mathbb{P}\left[g_t(X) > a\right] \leq \gamma_U$$



ball+gaussian



box+uniform



ball+uniform

```python
class PerturbationDomain(ABC):

    def get_upper(self, x: Tensor, w: Tensor, b: Tensor, missing_batchsize: bool = False, **kwargs:
Any):
        """Merge upper affine bounds with perturbation domain input to get upper constant bound."""
        ...

    def get_lower(self, x: Tensor, w: Tensor, b: Tensor, missing_batchsize: bool = False, **kwargs:
Any):
        """Merge lower affine bounds with perturbation domain input to get lower constant bound."""
        ...
```

AIRBUS

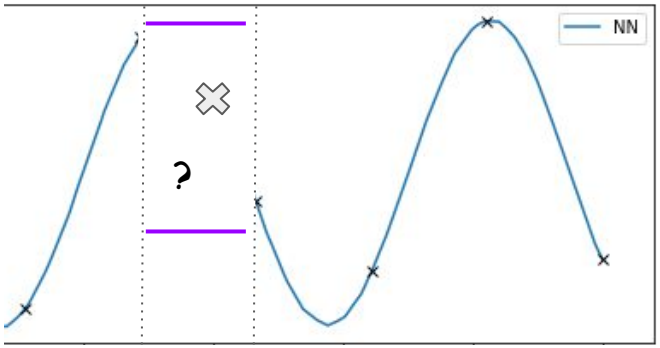
**TUTORIAL 3**

Local Robustness to Sensor Noise for Regression

- Use different perturbation domain

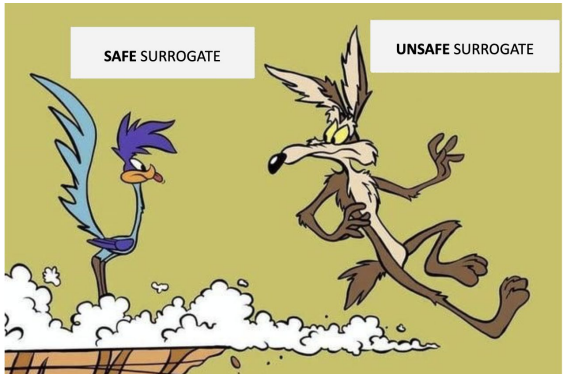
# Some properties for Regression

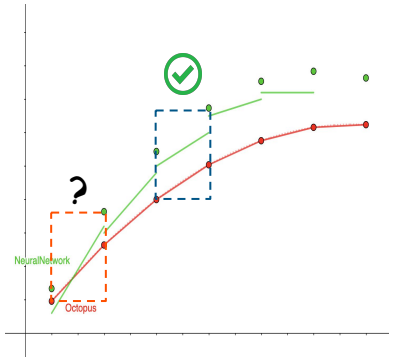
## Worst case variation




$$\max_{x \in \Omega} \mid f(x) - f(x_0) \mid$$

## Over-estimation




$$\min_{x \in \Omega} f(x) - f(x_0) \geq 0$$

## Partial Input Monotony

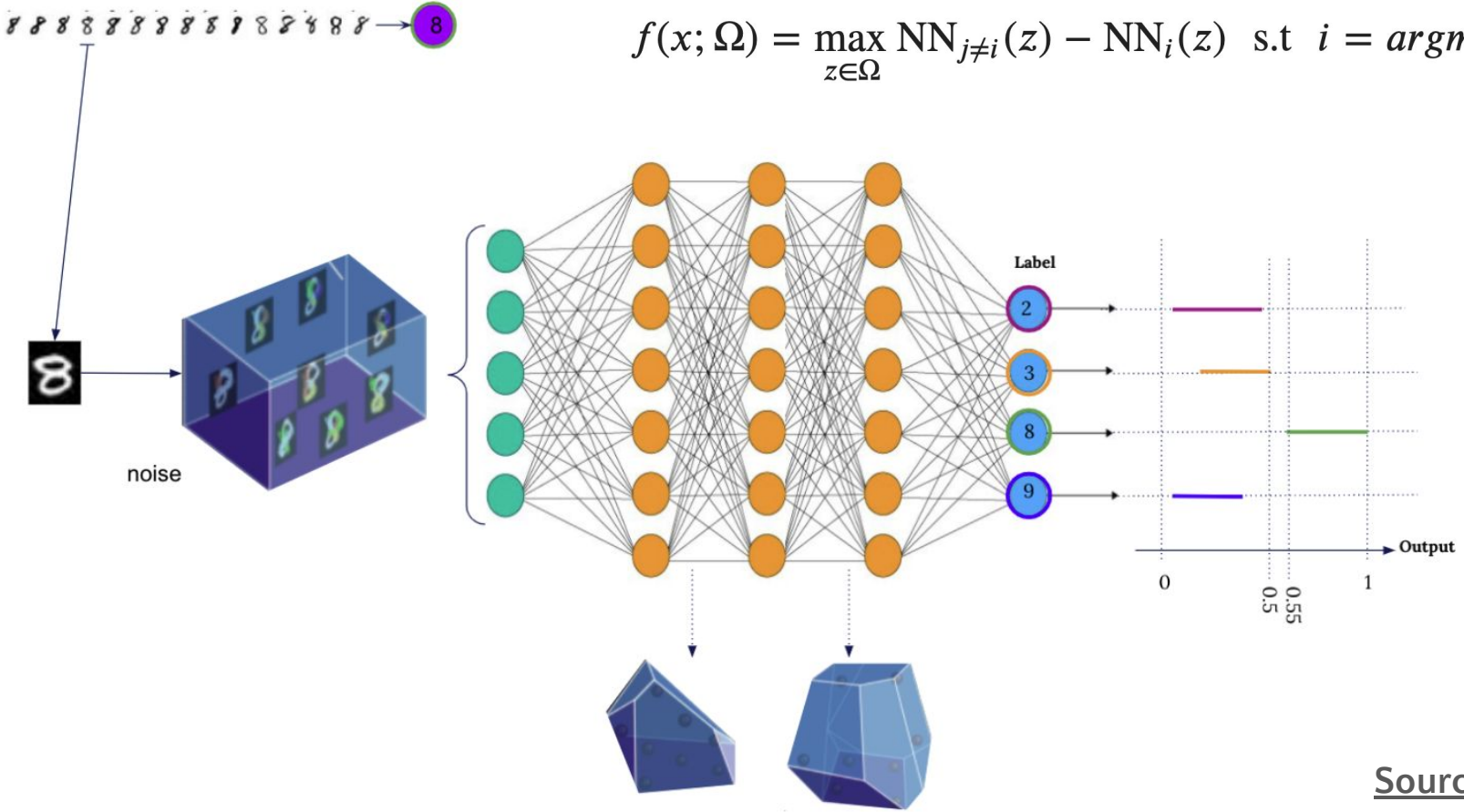
$$x_1 \qquad\qquad x_2 \qquad\qquad f(x_1) \qquad f(x_2)$$

$$\begin{pmatrix} \text{speed} \\ \text{weight} \\ \text{dry runway} \end{pmatrix} \begin{matrix} = \\ = \\ < \end{matrix} \begin{pmatrix} \text{speed} \\ \text{weight} \\ \text{wet runway} \end{pmatrix} \Longrightarrow \text{BDE}_1 < \text{BDE}_2$$

Source [1, 2, 3, 4, 5]

# Some properties for Classification

$$f(x;\Omega) = \max_{z\in\Omega} \mathrm{NN}_{j\neq i}(z) - \mathrm{NN}_i(z) \quad \text{s.t} \quad i = argmax\ \mathrm{NN}(x)$$

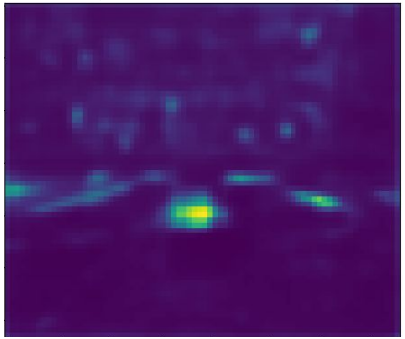



Source [6, 7, 8, 9]

# Some properties for (Single) Object detection

$x =$ 

Objectness map

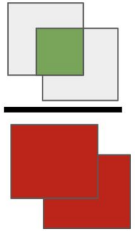
$f_0(x) =$ 

Local stability for Runway Detection

$$\min_{x \in \Omega} IOU\left(f(x), b_{gt}\right) \geq T$$

$$d = \text{argmax}\, f_0(x);$$

$$IoU(b_{gt}, b_d) = \frac{|b_{gt} \cap b_d|}{|b_{gt} \cup b_d|} =$$



**Verification for Object Detection -- IBP IoU**

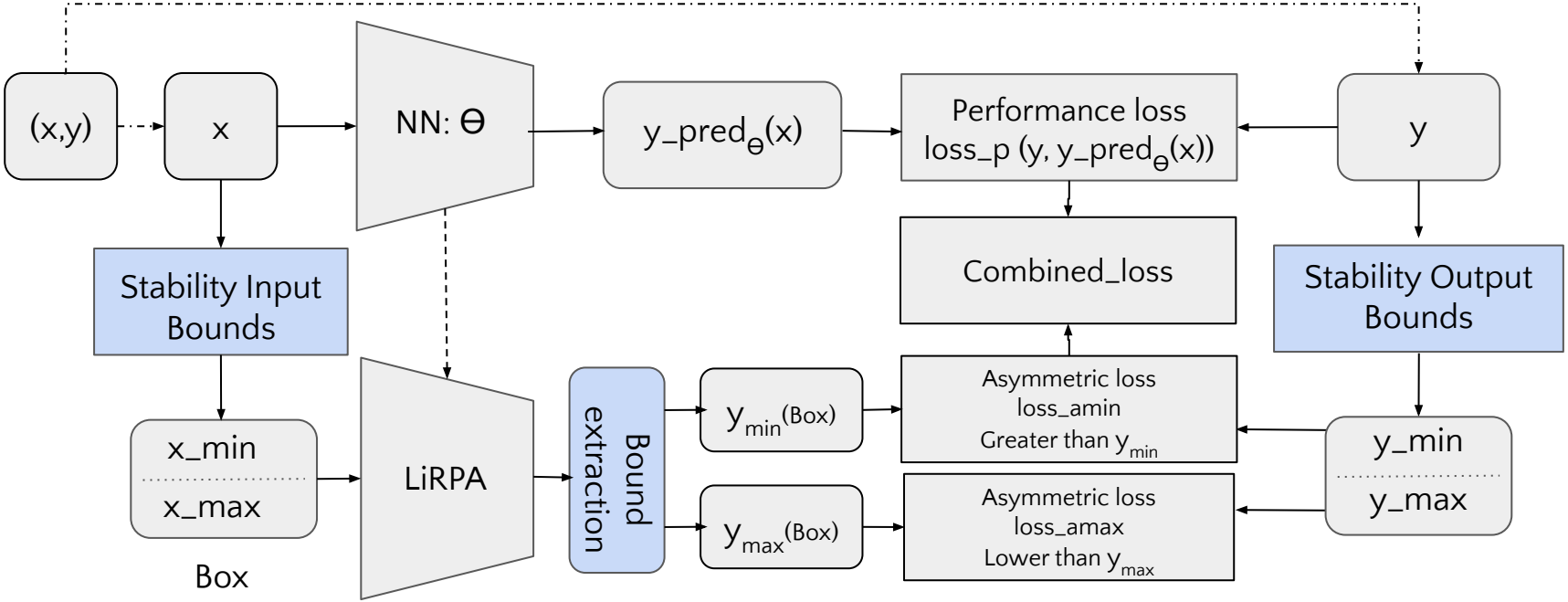
https://arxiv.org/abs/2403.08788

AIRBUS

## TUTORIAL 4

Local Robustness to misclassification

# Certified training



# Certified training: use IBP or CROWN-IBP

AIRBUS