Machine Learning for Graphs and Sequential Data

Robustness of Machine Learning – Exact Certification

Lecturer: Prof. Dr. Stephan Günnemann

www.daml.in.tum.de

Summer Term 2020

Data Analytics and
Machine Learning

Roadmap

1. Introduction
2. Construction of adversarial examples
3. Improving robustness
4. **Certifiable robustness**
   - **Exact certification**
   - Convex relaxations
   - Lipschitz-continuity
   - Randomized smoothing

Motivation

- Adversarial Training improves robustness, but how can we make sure that the user can really rely on the results?
  - There could still exist some cases where the model behaves in an undesired way

- As discussed, detection of adversarial examples does not seem to work

- **Better approach: Robustness certification**.
  - Idea: try to prove that the classifier's prediction does not change within a radius (measured by some norm)
  - If the proof is successful, we know there cannot be an adversarial example within that radius. We get a guarantee!
  - If the proof is not successful, the prediction could change. Therefore the sample might be an adversarial example (or it might be possible to adversarially change it).
    - In a very conservative approach, we could now refuse our model's prediction and/or consult an expert for manual inspection.

Exact Verification

Goal: Develop an algorithm that answers the question:

"Is the classifier f_θ around the sample $\mathbf{x}$ adversarial-free
(within an ϵ-ball measured by some norm)?"

The algorithm should return **YES** <u>if and only if</u> there are no adversarial example within an ϵ ball around the input sample (i.e. **NO** iff there is an adv. example).

Exact verification methods are typically designed for neural networks with **ReLU activation function**. ReLU networks are very prevalent in deep learning and are well-suited for **combinatorial** exact verification methods.

Exact Verification

- We view the neural network as a **sequence of functions** (i.e. the layers).
- Each **layer** is defined as $f_i(x) = \sigma(\boldsymbol{W}_i\, x + b_i)$, where $\boldsymbol{W}_i$ and b_i are the weight matrix and the bias of layer i, respectively.
- The **ReLU activation** function is defined as $\sigma(x) = \max(0, x)$ and is applied entry-wise to the input.
- The **overall network** is a function F: $\mathbb{R}^d \to \mathbb{R}^{|\mathcal{Y}|}$ given by:
$$F(x) = \boldsymbol{W}_L\ f_{L-1} \circ f_{L-2} \circ \ldots \circ f_1(x) + \boldsymbol{b}_L$$
- The output of F are the **logits** which are subsequently fed into the softmax function to obtain a categorical distribution.
  - We can omit the softmax for certification since the operation is order-preserving (i.e., the 'winning' class does not change).

Exact Certification: Complexity

Exact certification is a very powerful method for a defending system: we know exactly when a sample could be an adversarial example and can potentially even use this knowledge to get the worst-case perturbation for adversarial training.

Unfortunately, [Katz et al. 2017] report the following result:

Theorem: **Exact certification** of neural networks with ReLU activation function and L_∞-bounded perturbations is **NP-complete**.

Nevertheless, solvers for NP-complete problems have made significant progress, so certifying small to medium-size neural networks is sometimes possible.

Mixed Integer Linear Programming

- One approach for exact certification of ReLU networks is to use mixed integer linear programming (**MILP**).
- Recall **linear programs** (LPs):

$$\begin{array}{rl} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \textit{subject to} & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array}$$

- Integer linear programs: we have the additional constraints $\mathbf{x}_i \in \mathbb{Z}$, i.e. the variables are integer-valued
- Mixed integer linear programs (**MILP**): *some* variables constrained to be integers, others not.

Mixed Integer Linear Programming: Complexity

- One approach for exact certification of ReLU networks is to use mixed integer linear programming (**MILP**).
- Recall **linear programs** (LPs):

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T \mathbf{x} \\ \textit{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

Can be solved **efficiently** (in polynomial time)

- Integer linear programs: we have the additional constraints $\mathbf{x}_i \in \mathbb{Z}$, i.e. the variables are integer-valued
- Mixed integer linear programs (**MILP**): *some* variables constrained to be integers, others not.

NP-complete

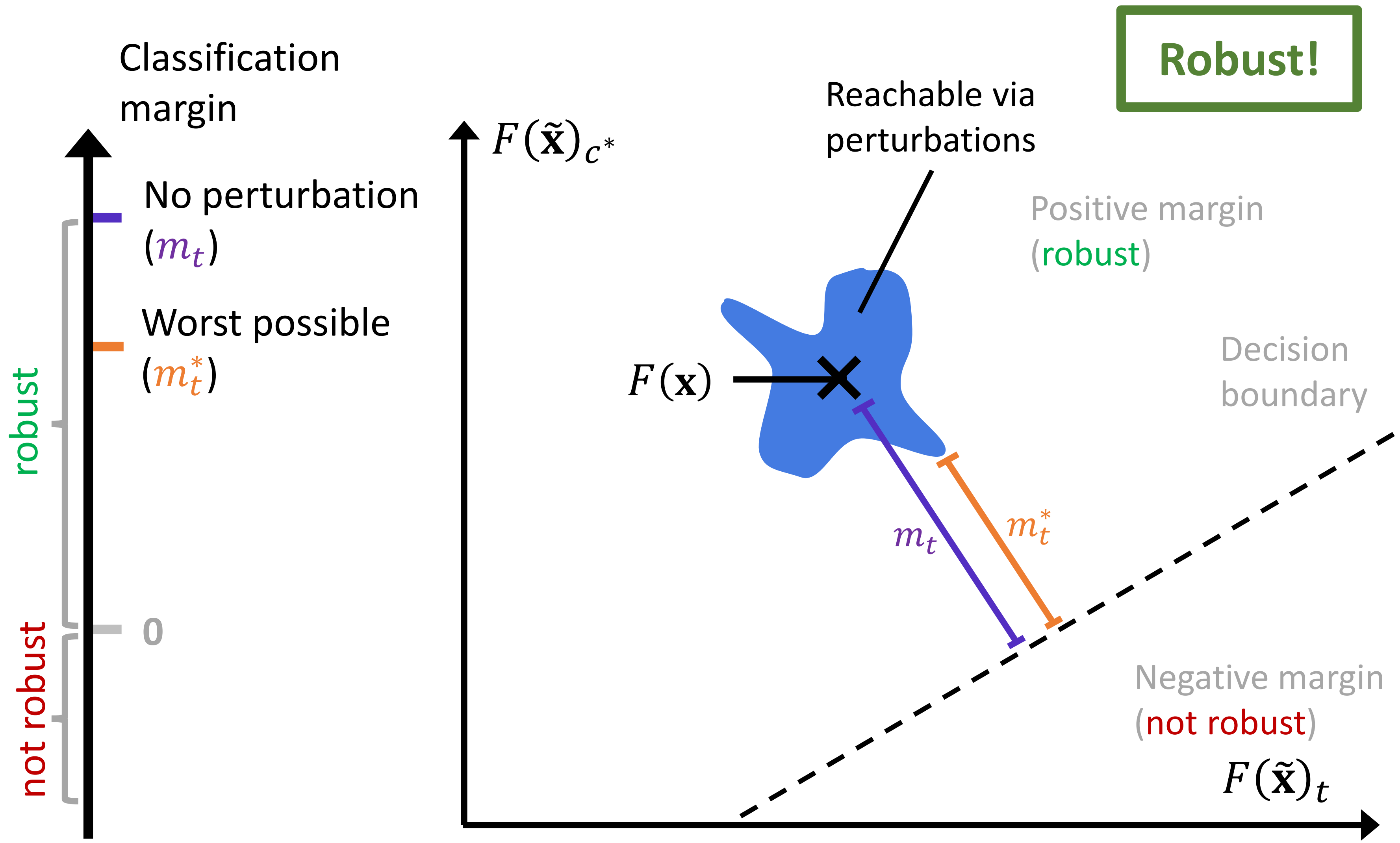
Expressing Exact Certification as MILP

- Suppose our classifier predicts class c^* for $\mathbf{x}$, i.e. $\mathrm{c}^* = \arg\max_c F(\mathbf{x})_c$
- We call $\mathrm{m_t} = F(\mathbf{x})_{c^*} - F(\mathbf{x})_t$ the classification margin of classes c^* and t.
- **Worst-case margin:**

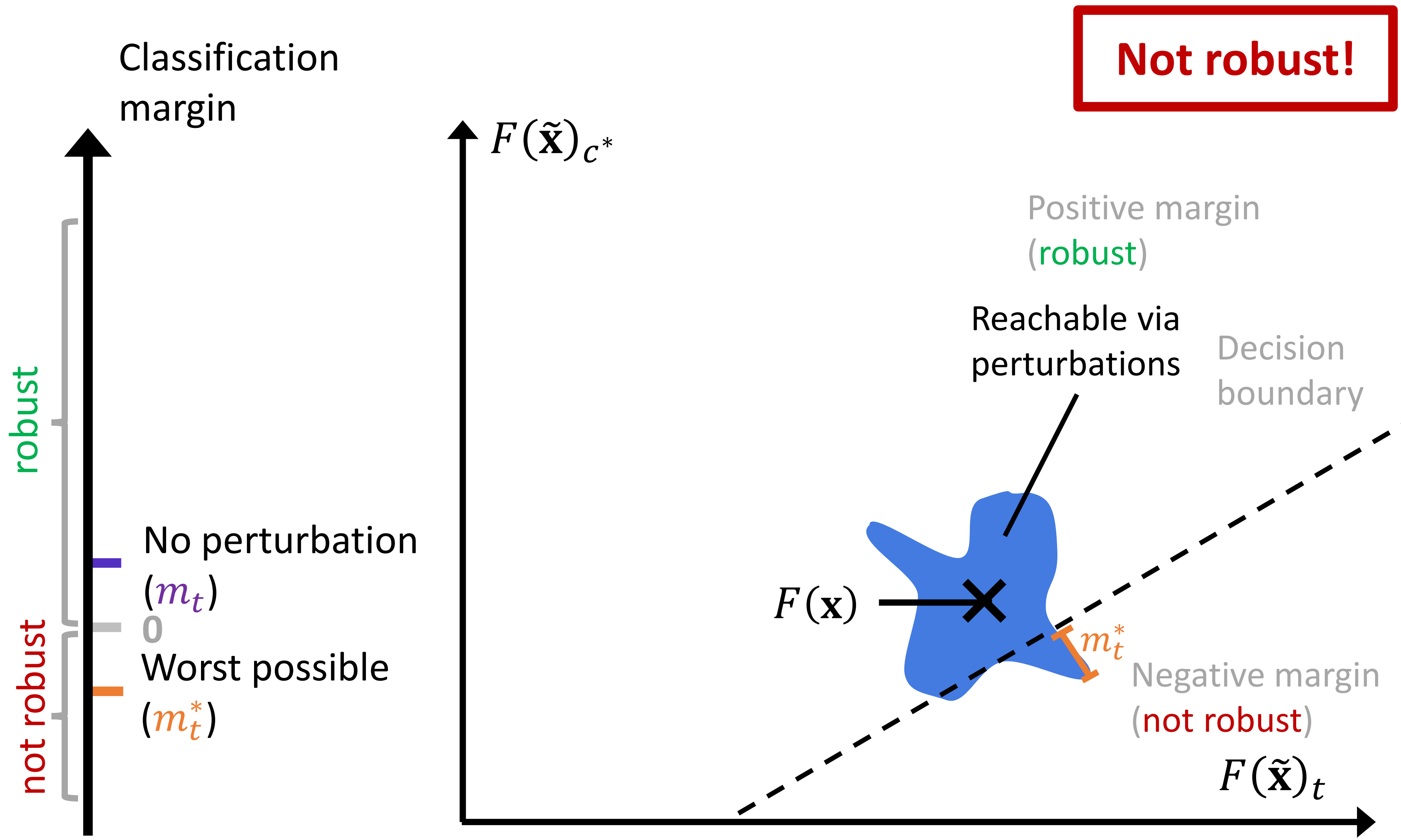
$$m_t^* = \min_{\tilde{\mathbf{x}}} \; F(\tilde{\mathbf{x}})_{c^*} - F(\tilde{\mathbf{x}})_t$$
$$subject\ to\ \|\tilde{\mathbf{x}} - \mathbf{x}\|_p \le \epsilon$$

- $m_t^* > 0$: the classifier's prediction cannot be changed from class c^* to t
- If for **all classes** $t \neq c^*$ we have $m_t^* > 0$ → we can certify robustness
- If for **any class** $t \neq c^*$ we have $m_t^* < 0$ → there exists an adversarial example $\tilde{\mathbf{x}} \in \mathcal{P}_{\epsilon,p}(\mathbf{x})$

Exact Robustness Certification: Illustration



Exact Robustness Certification: Illustration



Exact Certification: Optimization Problem

- We can write the optimization problem:

$$
\begin{aligned}
m_t^* = \min_{\tilde{\mathbf{x}}} \quad & [\hat{\mathbf{x}}^{(L)}]_{c^*} - [\hat{\mathbf{x}}^{(L)}]_t \\
\textit{subject to} \quad & \|\tilde{\mathbf{x}} - \mathbf{x}\|_p \le \epsilon \\
& \boldsymbol{y}^{(0)} = \tilde{\mathbf{x}} \\
& \hat{\mathbf{x}}^{(l)} = \boldsymbol{W}_l \boldsymbol{y}^{(l-1)} + \boldsymbol{b}_l \qquad \forall l = 1 \dots L \\
& \boldsymbol{y}^{(l)} = \mathrm{ReLU}\left(\hat{\mathbf{x}}^{(l)}\right) \qquad \forall l = 1 \dots L-1
\end{aligned}
$$

- Here, $\hat{\mathbf{x}}^{(l)}$ denotes the pre-ReLU activation at layer l.

- To express this as a MILP, we need to encode
  - The L_p constraints on the adversarial perturbation
  - The nonlinear ReLU constraints, which is where most of the difficulty comes from

Expressing Exact Certification as MILP: L_p Constraints

- L_∞ constraints are straightforward to include in linear programs
- We add the following constraints to the optimization:

$$\begin{aligned} \mathbf{x}_i - \tilde{\mathbf{x}}_i &\leq \epsilon \quad \forall i \\ \tilde{\mathbf{x}}_i - \mathbf{x}_i &\leq \epsilon \quad \forall i \end{aligned}$$

- L_1 constraints are also straightforward to encode
- L_2 constraints can be captured using mixed integer quadratic programming

Expressing Exact Certification as MILP: ReLU (1)

- The ReLU activation function is where most of the difficulty comes from.
- We want to encode $\mathbf{y} = \mathrm{ReLU}(\mathbf{x})$
- Naively, we can encode the ReLU activation by introducing a binary variable (vector of variables) $\boldsymbol{a}$:

$$\mathbf{y}_i \leq \boldsymbol{a}_i \cdot \mathbf{x}_i \text{ and } \mathbf{y}_i \geq 0 \text{ and } \mathbf{y}_i \geq \mathbf{x}_i$$

- However, now we have a constraint with a product of two variables, hence, is not linear and not convex.

Expressing Exact Certification as MILP: ReLU (2)

- Suppose we have lower and upper bounds $[\boldsymbol{l}, \boldsymbol{u}]$ on the input $\mathbf{x}$ to the ReLU activation.
- Then, we can encode the ReLU activation using linear and integer constraints [Tjeng et al. 2019]):

$$\mathbf{y}_i = \mathrm{ReLU}(\mathbf{x}_i) \Leftrightarrow \begin{array}{l} \quad (\mathbf{y}_i \leq \mathbf{x}_i - \boldsymbol{l}_i(1 - \boldsymbol{a}_i)) \\ \wedge (\mathbf{y}_i \leq \boldsymbol{a}_i \cdot \boldsymbol{u}_i) \\ \wedge (\mathbf{y}_i \geq \mathbf{x}_i) \\ \wedge (\mathbf{y}_i \geq 0) \\ \wedge (\boldsymbol{a}_i \in \{0, 1\}) \end{array} \qquad \forall i$$

Overall MILP

- Note that the overall MILP optimizes over multiple variables
  - Efficient solvers will remove some of them due to the equality constraints

$$m_t^* = \min_{\tilde{\mathbf{x}}, \boldsymbol{y}^{(l)}, \hat{\mathbf{x}}^{(l)}, \boldsymbol{a}^{(l)}} [\hat{\mathbf{x}}^{(L)}]_{c^*} - [\hat{\mathbf{x}}^{(L)}]_t$$

$$\begin{aligned}
\textit{subject to} \quad & \mathbf{x}_i - \tilde{\mathbf{x}}_i \leq \epsilon \quad \forall i \\
& \tilde{\mathbf{x}}_i - \mathbf{x}_i \leq \epsilon \quad \forall i \\
& \boldsymbol{y}^{(0)} = \tilde{\mathbf{x}} \\
& \hat{\mathbf{x}}^{(l)} = \boldsymbol{W}_l \boldsymbol{y}^{(l-1)} + \boldsymbol{b}_l \qquad \forall l = 1 \dots L \\
& \boldsymbol{y}_i^{(l)} \leq \hat{\mathbf{x}}_i^{(l)} - \boldsymbol{l}_i^{(l)} \left(1 - \boldsymbol{a}_i^{(l)}\right) \\
& \boldsymbol{y}_i^{(l)} \geq \hat{\mathbf{x}}_i^{(l)} \qquad \forall l = 1 \dots L-1 \\
& \boldsymbol{y}_i^{(l)} \leq \boldsymbol{u}_i^{(l)} \cdot \boldsymbol{a}_i^{(l)} \qquad \forall i \\
& \boldsymbol{y}_i^{(l)} \geq 0 \\
& \boldsymbol{a}_i^{(l)} \in \{0, 1\}
\end{aligned}$$

- Remark: We can also handle all classes $t \neq c^*$ at the same time in a single MILP

On the Lower and Upper Bounds

- The MILP formulation relies on being able to compute lower and upper bounds $[\boldsymbol{l}^{(l)}, \boldsymbol{u}^{(l)}]$ on the input $\hat{\mathbf{x}}^{(l)}$ to the ReLU activation (for every layer l)
- One simple way to get (loose) lower and upper bounds is interval arithmetic:

$$\boldsymbol{u}^{(l)} = [\boldsymbol{W}_l]_+ \boldsymbol{u}^{(l-1)} - [\boldsymbol{W}_l]_- \boldsymbol{l}^{(l-1)} + \boldsymbol{b}_l$$
$$\boldsymbol{l}^{(l)} = [\boldsymbol{W}_l]_+ \boldsymbol{l}^{(l-1)} - [\boldsymbol{W}_l]_- \boldsymbol{u}^{(l-1)} + \boldsymbol{b}_l$$

L_p constraints

$$\boldsymbol{u}_i^{(0)} = \boldsymbol{x}_i + \epsilon$$
$$\boldsymbol{l}_i^{(0)} = \boldsymbol{x}_i - \epsilon$$

- Here, $[\boldsymbol{W}]_+ = \max\{\boldsymbol{W}, 0\}, [\boldsymbol{W}]_- = \max\{-\boldsymbol{W}, 0\}$

Stable and Unstable Units

- Units for which $\boldsymbol{u}_i^{(l)} \geq \boldsymbol{l}_i^{(l)} \geq 0$ are called *stably active*
- Units for which $0 \geq \boldsymbol{u}_i^{(l)} \geq \boldsymbol{l}_i^{(l)}$ are called *stably inactive*

Can be removed from the optimization

- Units for which $\boldsymbol{u}_i^{(l)} \geq 0 \geq \boldsymbol{l}_i^{(l)}$ are called *unstable*

- While the tightness of the upper and lower bounds has no influence on the correctness of the result, **tighter bounds** lead to **more stable units** and greatly **speed up** the optimization.

Summary

- Exact verification is possible for ReLU-Networks
  - However, expensive for large neural networks
- Can we find more efficient certificates?
  - Unfortunately not if we focus on exact certificates (”if and only if”) due to the NP-hardness (assuming P!=NP)

- However, we can change the allowed answers to our question

“Is the classifier f_θ around the sample $\mathbf{x}$ adversarial-free (within an ϵ-ball measured by some norm)?”

➢ If the algorithm returns **`YES`**, there are no adversarial examples within an ϵ ball around the input sample; if the algorithm returns **`POTENTIALLY NOT`** there might be adversarial examples or it is adversarial-free

- This is a conservative (careful) answer, i.e. in cases where the algorithms says ”yes” we can rely on the prediction → i.e. we still have a guarantee
- We discuss such principles in the next sections!

Recommended Reading

- Lecture 12: Certified defenses I: Exact certification of Jerry Li‘s course on Robustness in Machine Learning (CSE 599-M), https://jerryzli.github.io/robust-ml-fall19.html

References – Exact Verification

- Katz, Guy, et al. "Reluplex: An efficient SMT solver for verifying deep neural networks." *International Conference on Computer Aided Verification*. Springer, Cham, 2017.
- Tjeng, Vincent, et al. “Evaluating Robustness of Neural Networks with Mixed Integer Programming.” *International Conference on Learning Representations*, 2019