Robustness Verification for Deep Neural Networks

Guanqin Zhang
Guanqin.Zhang@student.uts.edu.au

June 23, 2023

Agenda

1. Introduction
2. DNN Robustness Issues
3. Robustness Verification
4. Research Plan & Progress

Success of Deep Learning Models

Deep learning has achieved great success in a wide range of domains:

- **Computer vision tasks**, such as handwriting recognition, image classification, and object detection.
- **Natural language tasks,** such as speech recognition and text translation.
- **Recommendation tasks**, such as online shopping recommendations and news recommendations.

A Need for Verification

Deep learning techniques only rely on data so **unexpected outcomes** can happen. And deep learning techniques have been raised as **unreliable concerns**, given misclassification errors and misbehavior.

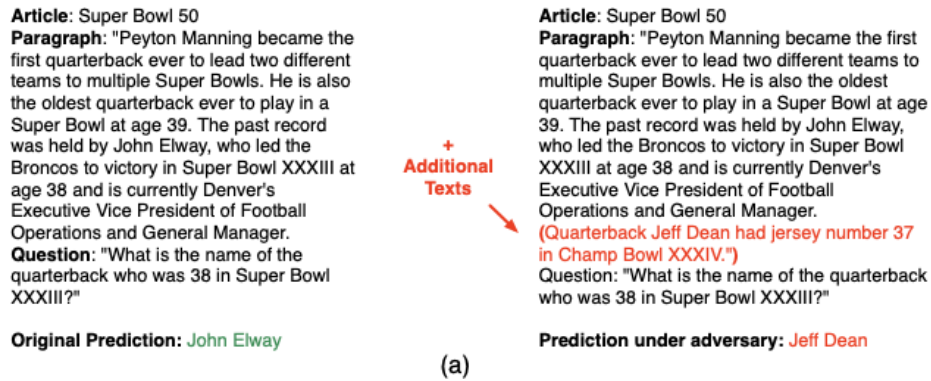
A Need for Verification

Deep learning techniques only rely on data so **unexpected outcomes** can happen. And deep learning techniques have been raised as **unreliable concerns**, given misclassification errors and misbehavior.

Verification is needed and important for deep learning practitioners to pass the certification procedures, particularly if the applied deep learning models are interfacing with humans and might have **life-safety-critical** implications.

Robustness Issues (a)

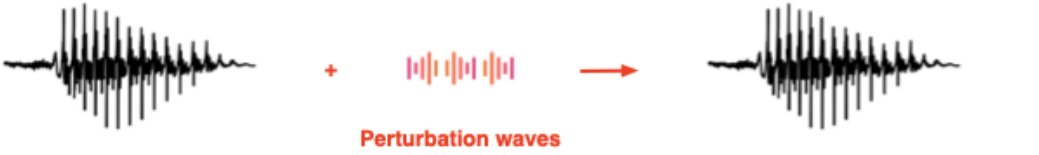
Distraction in a Q/A model.[1]



[1] Morris, John X., et al. "Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp." preprint arXiv:2005.05909 (2020).

Robustness Issues (b)

Unexpected recognition in a language model[2]



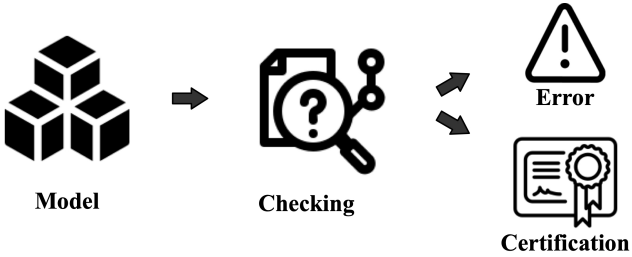
[2]Carlini, Nicholas, and David Wagner. "Audio adversarial examples: Targeted attacks on speech-to-text." 2018 IEEE security and privacy workshops (SPW). 2018.

Model's misbehavior: The stop sign with some illumination in the dark is **incorrectly** recognized as a "speed 60". Demo [3]

Under incorrect recognition, a **higher** probability of crashes may occur.

[3] Gnanasambandam, A., Sherman, A. M., & Chan, S. H. (2021). Optical adversarial attack. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 92-101).

Aim



Robustness verification aims to certify "specification guarantees" on model behaviors or find the errors of the model.

1. **Testing and verification** approaches can improve **dependability**, reduce **misbehavior**, detect outlier **counterexamples** of deep learning models:
   - **Testing discovers as many misbehaviors as possible** by using crafted testing samples.
   - **Verification proves the absence of misbehaviors** of deep learning models from the users' specifications.

Robustness Verification of Deep Learning Models

1. **Testing and verification** approaches can improve **dependability**, reduce **misbehavior**, detect outlier **counterexamples** of deep learning models:
   - **Testing discovers as many misbehaviors as possible** by using crafted testing samples.
   - **Verification proves the absence of misbehaviors** of deep learning models from the users' specifications.
2. Why **verification of deep learning models**?

1. **Testing and verification** approaches can improve **dependability**, reduce **misbehavior**, detect outlier **counterexamples** of deep learning models:
   - **Testing discovers as many misbehaviors as possible** by using crafted testing samples.
   - **Verification proves the absence of misbehaviors** of deep learning models from the users' specifications.
2. Why **verification of deep learning models**?
   - **Sample-based** testing or validations cannot **enumerate** all possible inputs and are time-consuming, especially in domains like image and video.
   - Verification can mathematically **certify** models with provably guaranteed robustness or supply a **counterexample** that violates the expected behaviors of models.

3. Challenges of applying existing verification techniques to neural networks:

3. Challenges of applying existing verification techniques to neural networks:
   - Deep learning models are **non-convex** or **nondeterministic**. They cannot be absolutely assumed as deterministic computer programs.
   - **Misbehaviors** of deep learning models are **not well-specified**.
   - Verification of different neural networks (e.g., recurrent neural networks and convolutional neural networks) and their applications (e.g., image classification and speech recognition) can vary greatly.

Deep Feedforward Networks

We demonstrate verification cases using a tiny **deep feedforward network** as a verification example because they form the basis of many other applications.

Deep Feedforward Networks

We demonstrate verification cases using a tiny **deep feedforward network** as a verification example because they form the basis of many other applications.

A feedforward network defines a mapping $\boldsymbol{y} = f_\theta(\boldsymbol{x})$, and learns the value of the parameters θ that result in the best function approximation.

We demonstrate verification cases using a tiny **deep feedforward network** as a verification example because they form the basis of many other applications.

A feedforward network defines a mapping $\boldsymbol{y} = f_\theta(\boldsymbol{x})$, and learns the value of the parameters θ that result in the best function approximation.

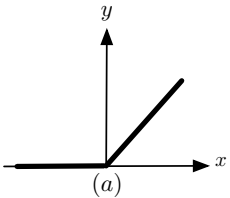
It accepts the input $\boldsymbol{x}$ which contains n features as $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, through the intermediate computation layers used to define f_θ, and finally to the output as $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$.

$$f_\theta : \mathbb{R}^n \to \mathbb{R}^m, \tag{1}$$

where $\mathbb{R}$ is the real domain set. At a high level, f_θ maps an n-dimensional input domain $\mathcal{I} \subseteq \mathbb{R}^n (n > 0)$ to an m-dimensional output domain $\mathcal{O} \subseteq \mathbb{R}^m (m > 0)$.

An activation function is a function used by an artificial neuron in the hidden layer of neural networks, which produces some output given one or more inputs to the neuron.

ReLU is called a rectified linear unit, if the value is greater than 0, then it will give away the same value as the output. otherwise, it will give 0 as output.



$\sigma = ReLU(x) = max(x, 0)$ is a piecewise activation function.

An Example

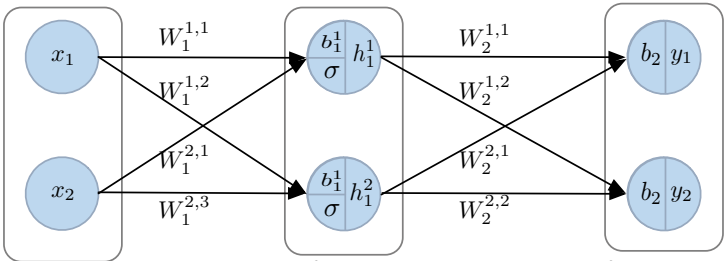


Figure 1: A tiny feed-forward neural network contains an input layer with 2 input nodes, an output layer with 2 output nodes, and a hidden layer with 2 hidden nodes.

x_i is the input, and b_i is the bias, and σ is the activation function, and W is the weights matrix.

Robustness Formulation

In response to the perturbations, a **robust** DNN model (f_θ) can be formalized as:

$$\forall \boldsymbol{x}, \hat{\boldsymbol{x}} \in \mathcal{I}, \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_{\mathrm{p}} < \epsilon \Rightarrow \mathit{argmax}\, f_\theta(\hat{\boldsymbol{x}}) = \mathit{argmax}\, f_\theta(\boldsymbol{x}) \tag{2}$$

where $\hat{\boldsymbol{x}}$ denotes the perturbed input under $\|\cdot\|_{\mathrm{p}}$ normalization with $\epsilon > 0$ distance (the degree of perturbations) to the original input $\boldsymbol{x}$ for the input space $\mathcal{I}$.

Robustness Formulation

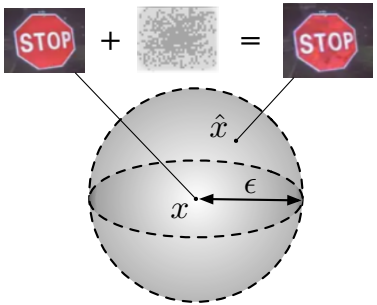


Figure 2: Within $\epsilon -$ **bounded** perturbated input should be classified as the similar result expected by humans.

A **well-trained robust** model can be viewed to maintain:

$$\forall \boldsymbol{x}, \hat{\boldsymbol{x}} \in \mathcal{I}, \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_{\mathrm{p}} < \epsilon \Rightarrow \mathit{argmax}\, f_\theta(\hat{\boldsymbol{x}}) = \mathit{argmax}\, f_\theta(\boldsymbol{x})$$

Robustness Verification Problem Formulation

Formally, a well-trained neural network function $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^m$, is verified as a certain mapping from inputs to outputs, as well as a property ϕ to verify. ϕ represents the specific behavior that we aim to ensure the network has.

Robustness Verification Problem Formulation

Formally, a well-trained neural network function $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^m$, is verified as a certain mapping from inputs to outputs, as well as a property ϕ to verify. ϕ represents the specific behavior that we aim to ensure the network has.

It is defined in two parts:

1. It contains a set of constraints on input to define a domain of desired robust region $\mathcal{R}(\epsilon - \mathit{bounded})$.
2. It comprises a Boolean function of network output that indicates if the prediction made by the network is compatible with the desired outcome.

Robustness Verification Problem Formulation

Formally, a well-trained neural network function $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^m$, is verified as a certain mapping from inputs to outputs, as well as a property ϕ to verify. ϕ represents the specific behavior that we aim to ensure the network has.

It is defined in two parts:

1. It contains a set of constraints on input to define a domain of desired robust region $\mathcal{R}(\epsilon - \mathit{bounded})$.
2. It comprises a Boolean function of network output that indicates if the prediction made by the network is compatible with the desired outcome.

Then, the property to verify is that for any samples belong to the robust region, the prediction made for it should satisfy the Boolean function. Formally, we write:

$$\forall \hat{\boldsymbol{x}} \in \mathcal{R}(\boldsymbol{x}), f_\theta(\hat{\boldsymbol{x}}) \vDash \phi, \tag{3}$$

where the robust region is formed as $\mathcal{R}(\boldsymbol{x}) := \{\hat{\boldsymbol{x}} | \ \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_\mathrm{p} \leq \epsilon\}$.

Verification Techniques

Many techniques designed for traditional program verification, such as **constraint solving (SMT-solving)**, **reachability analysis**, and **abstract interpretation**, are adopted in reasoning network models.

However, the verification of neural networks still has a long way to go.

SMT-based Approaches

The representative approaches [4] [5] [6] are using logical formulas in a set of constraints in the form of the **Hoare logic**, which contain the **input conditions** (C_{in}) and expected **output conditions** (C_{out}), among which are **conjugated** by the encoded networks (C_{f_θ}) as

$$C_{in} \wedge C_{f_\theta} \wedge C_{out} \tag{4}$$

The encoded network is expressed in the *Boolean* expressions. We verify the non-existence of the violations $\neg C_{out}$ as encoded networks C_{f_θ} below

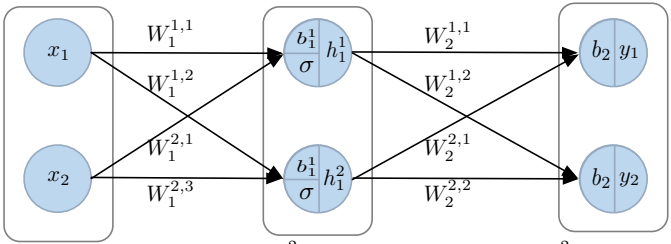
$$C_{in} \wedge C_{f_\theta} \wedge \neg C_{out}(violations) \tag{5}$$

[4] Huang, X., Kwiatkowska, M., Wang, S., & Wu, M. (2017, July). Safety verification of deep neural networks. In International conference on computer aided verification (pp. 3-29). Springer, Cham.

[5] Katz, G., Barrett, C., Dill, D. L., Julian, K., & Kochenderfer, M. J. (2017, July). Reluplex: An efficient SMT solver for verifying deep neural networks. In International conference on computer aided verification (pp. 97-117). Springer, Cham.

[6] Scheibler, K., Winterer, L., Wimmer, R., & Becker, B. (2015, March). Towards Verification of Artificial Neural Networks. In MBMV (pp. 30-40).

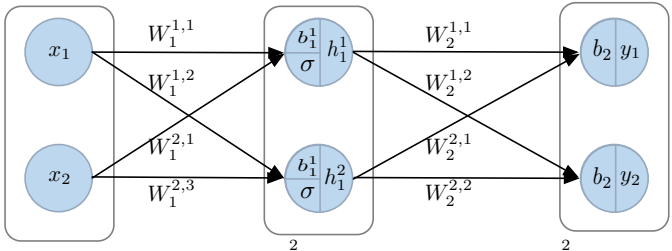
An Example



$$h_1^i = \sigma(\sum_{j=1}^{2} W_1^{j,i} x_j + b_1) \qquad y_i = \sum_{j=1}^{2} W_2^{j,i} h_1^j + b_2$$

- Take a specific input $(x_1 = 0.2, x_2 = 0.5)$.
- Generate a robust region as a constraint $C_{in}(0.1 < x_1 < 0.3 \wedge 0.4 < x_2 < 0.6)$ with $\epsilon = 0.1$, under ∞ norm.
- Assume user-desired output (the expected classification result), which can be expressed as $C_{out}(y_1 > y_2)$.

An Example (cont.)

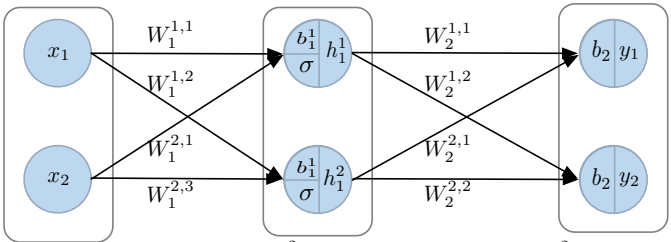


The neural network can be represented as $C_{f_\theta} = \bigwedge_{i=1}^{2} \sigma(\sum_{j=1}^{2} W_1^{j,i} h_1^j + b_1)$

Then, a conjunction of constraints $C_{in} \wedge C_{f_\theta} \wedge \neg C_{out}$ is formulated as

$0.1 < x_1 < 0.3 \wedge 0.4 < x_2 < 0.6 \wedge \bigwedge_{i=1}^{2} \sigma(\sum_{j=1}^{2} W_1^{j,i} h_1^j + b_1) \wedge y_1 \leq y_2$

An Example (cont.)



$$h_1^i = \sigma(\sum_{j=1}^{2} W_1^{j,i} x_j + b_1) \qquad y_i = \sum_{j=1}^{2} W_2^{j,i} h_1^j + b_2$$

The constraints can be resolved using an SMT solver (e.g., Z3 [7]) to return a satisfactory (certified) result if no counterexample that violates the constraints are found or an unsatisfied result otherwise.

[7] Moura, L. D., & Bjørner, N. (2008, March). Z3: An efficient SMT solver. In International conference on Tools and Algorithms for the Construction and Analysis of Systems (pp. 337-340). Springer, Berlin, Heidelberg.

Exact Method (Reluplex)

A specific input $(x_1 = 0.2, x_2 = 0.5)$ can be expressed as a conjunct clause in the constraint $C_{in}(0.1 < x_1 < 0.3 \wedge 0.4 < x_2 < 0.6)$, and the neural network is a legitimate model if it can always yield user-desired output (the expected classification result) which can be expressed as $C_{out}(y_1 > y_2)$. The neural network can be represented as

$$C_f = \bigwedge_{i=1}^{2} \sigma(\sum_{j=1}^{2} W_1^{j,i} h_1^j + b_1) \tag{6}$$

Then, a conjunction of constraints $C_{in} \wedge C_f \wedge \neg C_{out}$ is used to search one violated example (counterexample).

However, this is 'NP-hard', when solving by SAT or SMT solver.

In a mathematical optimization problem or numerical optimization, the form is usually expressed as

$$\begin{aligned} \text{minimize} \quad & f_0(\boldsymbol{x}), \\ \text{s.t.} \quad & f_i(\boldsymbol{x}) \leq c_i, i \in \{1, \ldots, n\}, \end{aligned} \tag{7}$$

In a mathematical optimization problem or numerical optimization, the form is usually expressed as

$$\begin{aligned} \text{minimize} \quad & f_0(\boldsymbol{x}), \\ \text{s.t.} \quad & f_i(\boldsymbol{x}) \leq c_i, i \in \{1, \ldots, n\}, \end{aligned} \tag{7}$$

Then we can transform the properties of the symbolic constraints into linear programming verification properties:

$$\begin{aligned} &\text{minimize } y_1 - y_2 \\ &\text{s.t. } \boldsymbol{C_{in}} \wedge \boldsymbol{C_f} \end{aligned} \tag{8}$$

In a mathematical optimization problem or numerical optimization, the form is usually expressed as

$$\begin{aligned} \text{minimize} \quad & f_0(\boldsymbol{x}), \\ \text{s.t.} \quad & f_i(\boldsymbol{x}) \leq c_i, i \in \{1, \ldots, n\}, \end{aligned} \tag{7}$$

Then we can transform the properties of the symbolic constraints into linear programming verification properties:

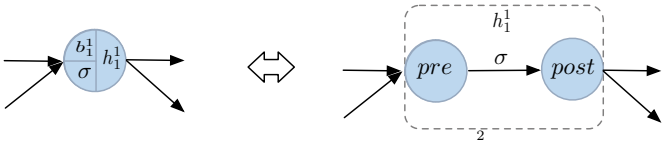
$$\begin{aligned} &\text{minimize } y_1 - y_2 \\ &\text{s.t. } \boldsymbol{C_{in}} \wedge \boldsymbol{C_f} \\ &\textbf{as} \end{aligned} \tag{8}$$

$$\begin{aligned} &\text{minimize } y_1 - y_2 \\ &\text{s.t. } \{0.1 - x_1 < 0\} \wedge \{x_1 - 0.3 < 0\} \wedge \cdots \wedge \boldsymbol{C_f} \end{aligned} \tag{9}$$

Linear Programming (LP)

ReLU function is **piece-wise**, and directly verifying DNN model is **non-convex** problem.

ReLU function is **piece-wise**, and directly verifying DNN model is **non-convex** problem.

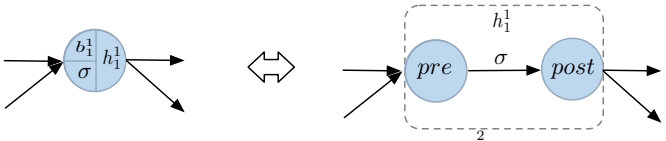


$$h_1^i = \sigma(\sum_{j=1}^{2} W_1^{j,i} x_j + b_1)$$

$$pre = \sum_{j=1}^{2} W_1^{j,i} x_j + b_1$$

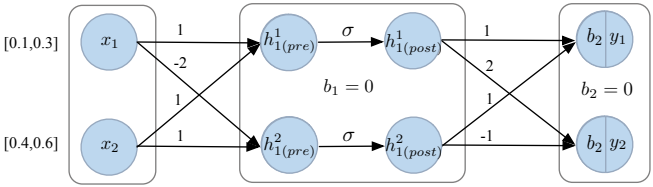
$$h_1^i = post = \sigma(pre)$$

ReLU function is **piece-wise**, and directly verifying DNN model is **non-convex** problem.



Each *ReLU* function can produce different outputs from two inputs that introduce a disjunctive ($\vee$) formula

LP Example



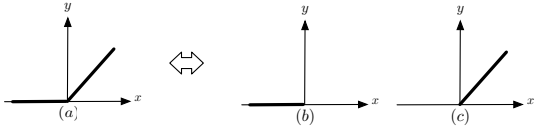
Then, we reformulate the previous constraint $C_{in} \wedge C_f \wedge \neg C_{out}$ as

$$
\begin{aligned}
\boldsymbol{C_{in}} &= \{(0.1 < x_1 < 0.3) \wedge (0.4 < x_2 < 0.6)\} \\
\boldsymbol{C_f} &= \{h^1_{1(pre)} = x_1 + x_2, h^2_{1(pre)} = x_1 - x_2, \\
&((h^1_{1(pre)} \leq 0) \wedge (h^1_{1(post)} = 0)) \vee ((h^1_{1(pre)} > 0) \wedge (h^1_{1(post)} = h^1_{1(pre)})), \\
&((h^2_{1(pre)} \leq 0) \wedge (h^2_{1(post)} = 0)) \vee ((h^2_{1(pre)} > 0) \wedge (h^2_{1(post)} = h^2_{1(pre)})), \\
&y_1 = h^1_{1(post)} + h^2_{1(post)}, y_2 = 2h^1_{1(post)} - h^2_{1(post)}\} \\
\neg \boldsymbol{C_{out}} &= \{y_1 \leq y_2\}.
\end{aligned}
\tag{10}
$$

Such that

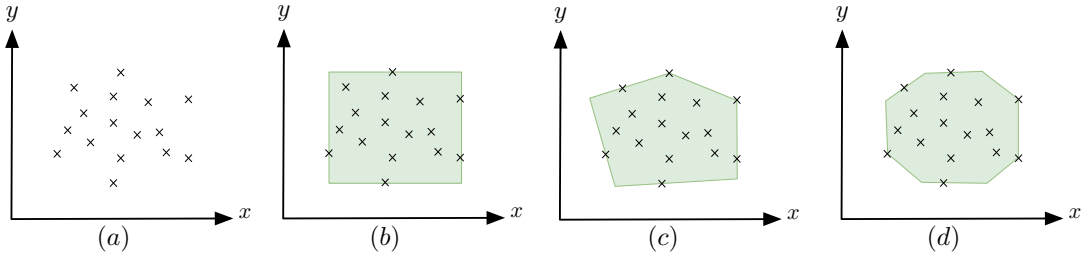
$$h_1 \rightarrow ((pre \leq 0) \wedge (post = 0)) \vee ((pre > 0) \wedge (post = pre)), \tag{11}$$

bisects the *ReLU* function in the verification problem.



However, only using LP solver, can not solve this verification problem because *ReLU* is a non-linear function.

Abstraction to Bound



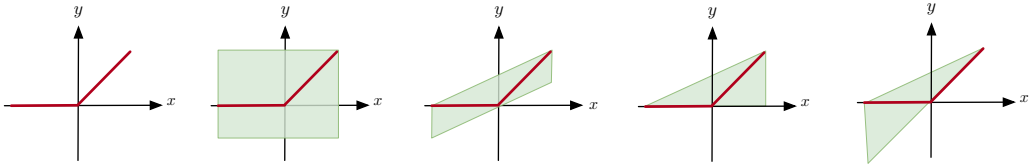
In literature, ReLU can be bound by abstraction methods, such as **Interval, Polyhedron, Zonotop**

Abstract Domaim

In abstract interpretation, it is essential to choose a suitable abstract domain because it determines the efficiency and precision of the abstract interpretation.

- **Interval** contains a lower bound and an upper bound for each variable, which can be viewed as $l \leq x_i \leq u$ or we defined as $[x] = [l, u]$.
- **Zonotop** contains a set of constraints as $x_i = a_i + \sum_{j=1}^{m} b_j \gamma_j$, where a_i and b_j are real scalars, and γ_j is an interval with $[l, u]$, m number of input features.
- **PolyHedron** contains constraints in the form of linear (in)equality (i.e., $\sum_{i=1}^{m} a_i x_i \leq b$).

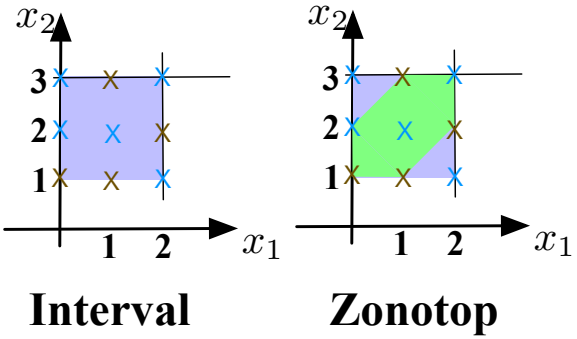
Linear Abstraction



Then, we can reformulate the neural network functions by the linear constraints.

$$f_\theta(\boldsymbol{x}) = f^k \circ f^{k-1} \circ \cdots \circ f^1(\boldsymbol{x}), \tag{12}$$

It's **over-approximated**.



- $x_1, x_2 \in \mathbb{Z}$
- Interval space takes 9 integer points, while Zonotope takes 7 points.

- Brown points are from the concrete domain.
- Blue points are the approximated points from the abstract domain.

Other Verification Methods

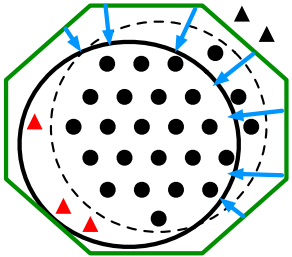
Mixed integer programming (MIP)[8] is another mathematical optimization technique that involves optimizing an objective function subject to a set of constraints, where some variables are required to be **integers**.

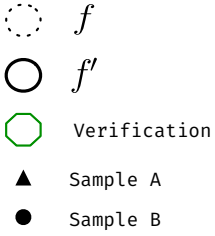
Branch and bound [9] is another verification approach that begins by utilizing an approximate verification method to obtain a lower and upper bound. If the upper bound is negative or zero, this neuron verification will be terminated.

[8] Tjeng, V., Xiao, K., & Tedrake, R. (2017). Evaluating robustness of neural networks with mixed integer programming. preprint arXiv:1711.07356.

[9] Zhang, H., Wang, S., Xu, K., Li, L., Li, B., Jana, S., ... & Kolter, J. Z. (2022). General cutting planes for bound-propagation-based neural network verification. arXiv:2208.05740.

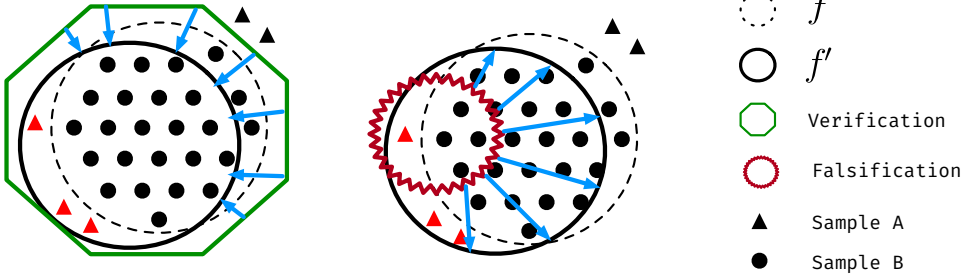
Verification





- over-approximated
- slow
- incomplete

Recently Proposed Work (in progress)



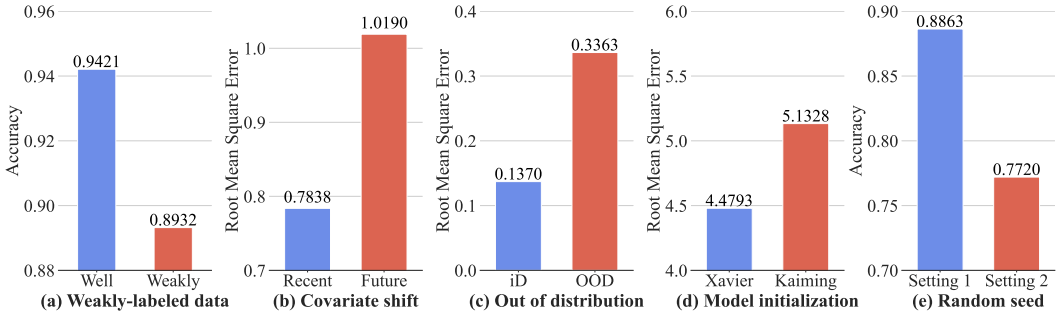
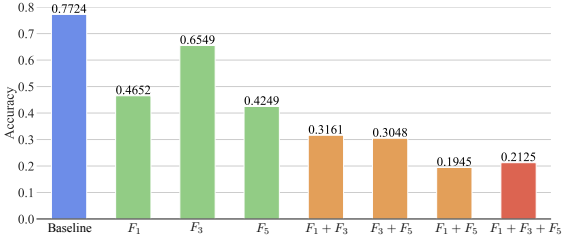


Figure 3: A range of applications in DNN projects with corresponding data and configurations. Accuracy on classification and root mean square error (RMSE) on regression tasks

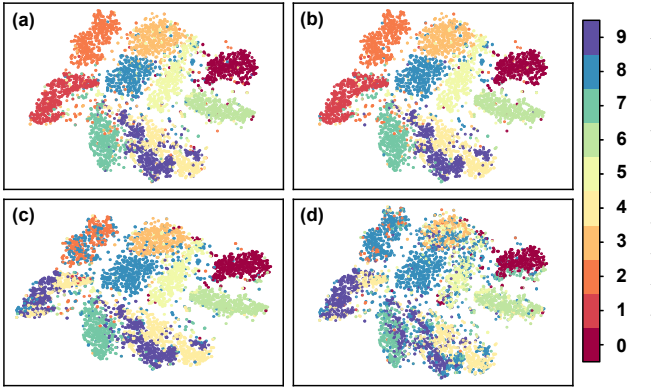
Combinatorial factors



Impacts of combinatorial factors on an image classification DNN model.

Surface	Factor	Target
Data (F_D)	F_1 Adversarial attack	Input $\mathbb{X}$
	F_2 Out of distribution	
	F_3 Label flipping attack	Output $\mathbb{Y}$
	F_4 Label noise injection	
Configuration (F_C)	F_5 Weight modification	Model f_θ
	F_6 Bias modification	
	F_7 Conv layer modification	
	F_8 FC layer modification	
	F_9 Number of threadings	
	F_{10} Pseudorandom seeds	
	...	...

Case Study 2 (submitted)



Predictive distribution of the MNIST testing set and adversarial set from machine learning and unlearning models. Here, the samples are clustered according to the predicted labels using t-distributed stochastic neighbor embedding (t-SNE) visualization

Research Plan & Progress

Months 0–6 (first stage)	Summarize and elaborate more on previous work; Data collection and comparative study; Expand literature review in DNNs and verification;	Finished a case-study[10]
Months 7–12 (second stage)	Specify my research target and plan; Collect methods that could aid my specific project; Conduct the desirable DNNs case study;	Conduct machine unlearning case study (in submission progress) Propose a falsification method speed up the verification processes
Months 13–24 (third stage)	Propose idea and design the corresponding method; Validate the proposed method; Mid-term Reports; Prepare & Submit journal, peer-reviewed conferences;	
Months 25-32 (third stage)	Data collections; Extend & Form new ideas; Submit journal, peer-reviewed conferences;	
Months 33-36	Prepare & Submission thesis;	

[10] Zhang, Guanqin, Jiankun Sun, Feng Xu, H. M. N. Bandara, Shiping Chen, Yulei Sui, and Tim Menzies. "A Tale of Two Cities: Data and Configuration Variances in Robust Deep Learning." arXiv preprint arXiv:2211.10012 (2022).