



Vehicle Tutorial Chapter 4: Property-Driven Training

Today's presentors: Ekaterina Komendantskaya and Luca Arnaboldi (live),
Matthew Daggitt (online), on behalf of the Vehicle team

In this chapter...



We will discuss:

- ▶ ... why and how training is a part of verification of neural networks

In this chapter...



We will discuss:

- ▶ ... why and how training is a part of verification of neural networks
- ▶ ... what choices exist for implementing this, generally

In this chapter...



We will discuss:

- ▶ ... why and how training is a part of verification of neural networks
- ▶ ... what choices exist for implementing this, generally
- ▶ ... what choices **Vehicle** makes in this respect

In this chapter...



We will discuss:

- ▶ ... why and how training is a part of verification of neural networks
- ▶ ... what choices exist for implementing this, generally
- ▶ ... what choices **Vehicle** makes in this respect
- ▶ ... theoretical and practical issues with the chosen methods, and **Vehicle's** take on them

Recap: four PL problems



- I^O Interoperability – properties are not portable between training/counter-example search/ verification.
- I^P Interpretability – code is not easy to understand.
- I^f Integration – properties of networks cannot be linked to larger control system properties.
- E^G Embedding gap – little support for translation between problem space and input space.

Why Training is a part of Verification?



```
vehicle verify \  
  --specification acasXu.vcl \  
  --verifier Marabou \  
  --network acasXu:acasXu_1_7.onnx \  
  --property property3
```

Verifying properties:

```
property3 [=====] 1/1 queries  
  result: counterexample found  
  x: [1799.9886669999978, 1.9509286320000003e-2,  
      3.09999732192, 980.0, 1017.6036]
```

Why Training is a part of Verification?



Verifying a Fashion MNIST network on 500 samples we get:


	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.5$
Success rate:	82.6 % (413/500)	29.8 % (149/500)	3.8 % (19/500)	0 % (0/500)

A few words on the context



1943 Perceptron by McCulloch and Pitts

90-2000 Rise of machine learning applications

2013  C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. 2013. (10000+ citations on GS)

2013-.. Tens of thousands of papers on adversarial training
(in the attack-defence style)


 A. C. Serban, E. Poll, J. Visser. Adversarial Examples - A Complete Characterisation of the Phenomenon. 2019.

A few words on the context



1943 Perceptron by McCulloch and Pitts

90-2000 Rise of machine learning applications


2013  C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. 2013. (10000+ citations on GS)

2013-.. Tens of thousands of papers on adversarial training
(in the attack-defence style)

 A. C. Serban, E. Poll, J. Visser. Adversarial Examples - A Complete Characterisation of the Phenomenon. 2019.

2017 First Neural network verification attempts

 G. Katz, C.W. Barrett, D.L. Dill, K. Julian, M.J. Kochenderfer: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. CAV (1) 2017: 97-117.

 X. Huang, M. Kwiatkowska, S. Wang, M. Wu. Safety Verification of Deep Neural Networks. CAV (1) 2017: 3-29.

2017-.. Hundreds of papers on neural network verification



Training for Robustness



Training for Robustness

Training generally:

1. depends on data
2. depends on loss functions
3. some other parameters like shape of the functions

1. Data Augmentation

Suppose we are given a data set $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$.

Prior to training, generate new training data samples close to existing data and label them with the same output as the original data.



C. Shorten, T.M. Khoshgoftaar: A survey on image data augmentation for deep learning. J. Big Data 6, 60 (2019)



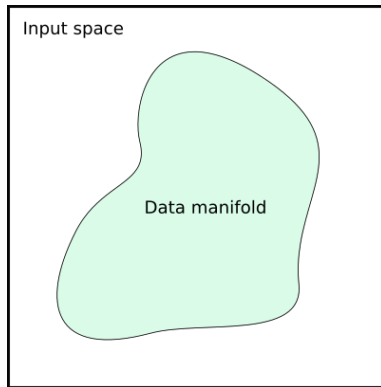
1. Data Augmentation

Suppose we are given a data set $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$.

Prior to training, generate new training data samples close to existing data and label them with the same output as the original data.



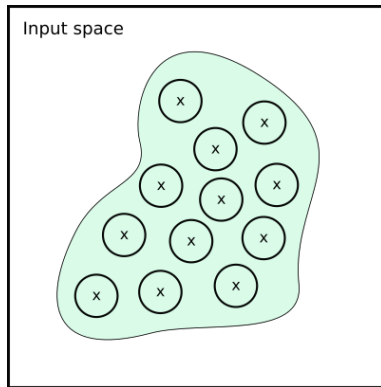
C. Shorten, T.M. Khoshgoftaar: A survey on image data augmentation for deep learning. J. Big Data 6, 60 (2019)



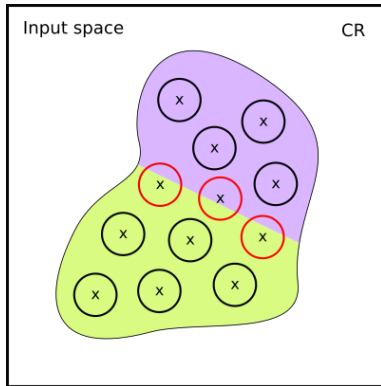
1. Data Augmentation

Suppose we are given a data set $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$. Prior to training, generate new training data samples close to existing data and label them with the same output as the original data.

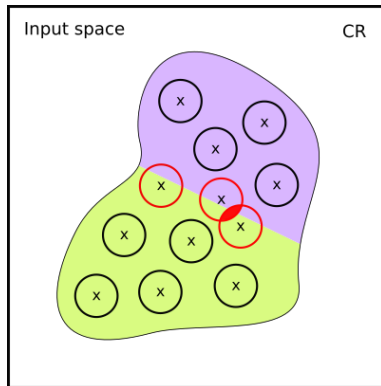
 C. Shorten, T.M. Khoshgoftaar: A survey on image data augmentation for deep learning. J. Big Data 6, 60 (2019)



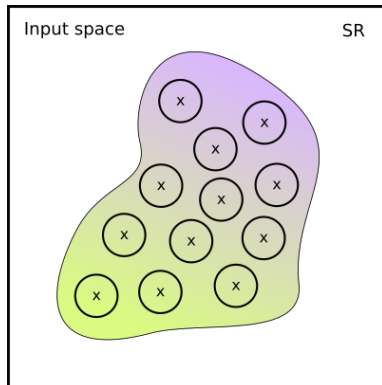
However,



However,



Adversarial Training



I.J. Goodfellow, J. Shlens, C. Szegedy: Explaining and harnessing adversarial examples. 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)

2. Solutions Involving Loss Functions



Given a data set \mathcal{D} ,

a function $f_{\theta} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ (with optimisation parameters θ),

a loss function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ computes a penalty proportional to the difference between the output of f_{θ} on a training input $\hat{\mathbf{x}}$ and a desired output \mathbf{y} .

2. Solutions Involving Loss Functions



Given a data set \mathcal{D} ,
a function $f_{\theta} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ (with optimisation parameters θ),
a loss function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ computes a penalty proportional to the difference between the output of f_{θ} on a training input $\hat{\mathbf{x}}$ and a desired output \mathbf{y} .

Example (Cross Entropy Loss Function)

Given a function $f_{\theta} : \mathbb{R}^n \rightarrow [0, 1]^m$, the cross-entropy loss is defined as

$$\mathcal{L}_{ce}(\hat{\mathbf{x}}, \mathbf{y}) = - \sum_{i=1}^m \mathbf{y}_i \log(f_{\theta}(\hat{\mathbf{x}})_i) \quad (1)$$

where \mathbf{y}_i is the true probability for class i and $f_{\theta}(\hat{\mathbf{x}})_i$ is the probability for class i as predicted by f_{θ} when applied to $\hat{\mathbf{x}}$.

2. Adversarial Training for Robustness

- ▶ gradient descent minimises loss $\mathcal{L}(\hat{\mathbf{x}}, \mathbf{y})$ between the predicted value $f_{\theta}(\hat{\mathbf{x}})$ and the true value \mathbf{y} , for each entry $(\hat{\mathbf{x}}, \mathbf{y})$ in \mathcal{D} :

$$\min_{\theta} \mathcal{L}(\hat{\mathbf{x}}, \mathbf{y})$$



2. Adversarial Training for Robustness



- ▶ gradient descent minimises loss $\mathcal{L}(\hat{\mathbf{x}}, \mathbf{y})$ between the predicted value $f_{\theta}(\hat{\mathbf{x}})$ and the true value \mathbf{y} , for each entry $(\hat{\mathbf{x}}, \mathbf{y})$ in \mathcal{D} :

$$\min_{\theta} \mathcal{L}(\hat{\mathbf{x}}, \mathbf{y})$$

- ▶ For adversarial training, we instead minimise the loss with respect to the worst-case perturbation of each sample in \mathcal{D} .
 - ▶ Replace the standard training objective with:

$$\min_{\theta} \left[\max_{\mathbf{x}: |\mathbf{x} - \hat{\mathbf{x}}| \leq \epsilon} \mathcal{L}(\mathbf{x}, \mathbf{y}) \right]$$

- ▶ the inner maximisation is done by “projected gradient descent” (PGD), that “projects” the gradient of \mathcal{L} on $\hat{\mathbf{x}}$ in order to perturb it and get the worst \mathbf{x} .




3. Lipschitz Continuity



Optimise for:

$$\forall \mathbf{x} : |\mathbf{x} - \hat{\mathbf{x}}| \leq \epsilon \Rightarrow |f(\mathbf{x}) - f(\hat{\mathbf{x}})| \leq L|\mathbf{x} - \hat{\mathbf{x}}|$$

 P. Pauli, A. Koch, J. Berberich, P. Kohler, F. Allgower: Training robust neural networks using Lipschitz bounds. IEEE Control Systems Letters (2021)

 H. Gouk, E. Frank, B. Pfahringer, M.J. Cree: Regularisation of neural networks by enforcing Lipschitz continuity. Machine Learning 110(2), 393–416 (2021)

and much more...



Ok, great!

Machine Learning Community knows how to make our networks more robust, and maybe even verifiable!



Ok, great!

Machine Learning Community knows how to make our networks more robust, and maybe even verifiable!

But remember:

I^O Interoperability – properties are not portable between training/counter-example search/ verification.

I^P Interpretability ...

I^f Integration ...

E^G Embedding gap ...

Interpretation of adversarial training:



Recall the epsilon ball robustness:

$$\forall \mathbf{x} \in \mathbb{B}(\hat{\mathbf{x}}, \epsilon). \text{ *robust*}(f(\mathbf{x}))$$

We can map different kinds of adversarial training to formal properties:

Training style	Definition of <i>robust</i>
Data Augmentation	$\operatorname{argmax} f(\mathbf{x}) = i$
Adversarial Training	$ f(\mathbf{x}) - f(\hat{\mathbf{x}}) \leq \delta$
Lipschitz Continuity	$ f(\mathbf{x}) - f(\hat{\mathbf{x}}) \leq L \mathbf{x} - \hat{\mathbf{x}} $
...	...



M. Casadio, E. Komendantskaya, M. L. Daggitt, W. Kokke, G. Katz, G. Amir, and I. Rafaeli. 2022. Neural Network Robustness as a Verification Property: A Principled Case Study. CAV'22.

Consequences of this finding:



- ▶ one kind of robustness does not necessarily imply another;

Consequences of this finding:



- ▶ one kind of robustness does not necessarily imply another;
- ▶ It is easy to get it wrong, and, while optimising for **one kind** of robustness, achieve little in verification success rates

Consequences of this finding:



- ▶ one kind of robustness does not necessarily imply another;
- ▶ It is easy to get it wrong, and, while optimising for **one kind** of robustness, achieve little in verification success rates

Example

In majority of ML + verification papers, adversarial robustness is used for training (it encourages standard robustness of networks), while verification is done for classification robustness. We show that these two types of robustness are not in any relation: i.e. increasing one does not generally increase the other.

Consequences of this finding:



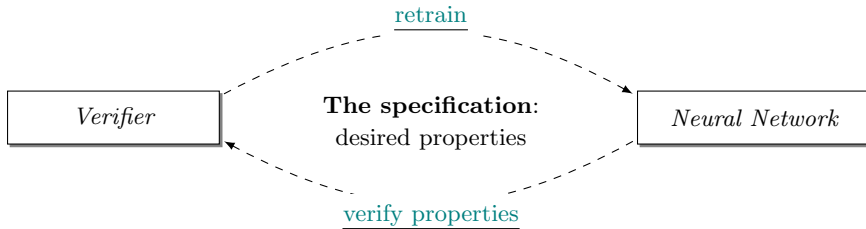
- ▶ one kind of robustness does not necessarily imply another;
- ▶ It is easy to get it wrong, and, while optimising for **one kind** of robustness, achieve little in verification success rates

Example

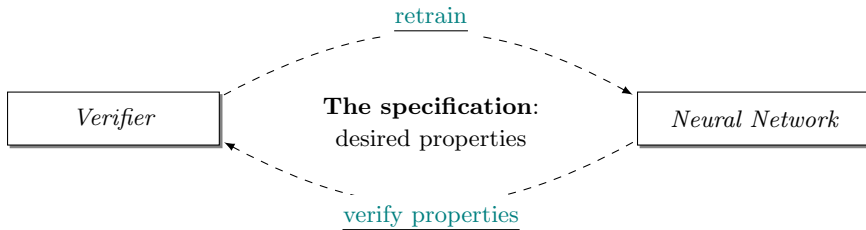
In majority of ML + verification papers, adversarial robustness is used for training (it encourages standard robustness of networks), while verification is done for classification robustness. We show that these two types of robustness are not in any relation: i.e. increasing one does not generally increase the other.

- ▶ And what to do with properties that are not ϵ -ball robustness?
Out-of-the-box PGD training only works with ϵ -balls around data points.

The solution we are looking for



The solution we are looking for



NB

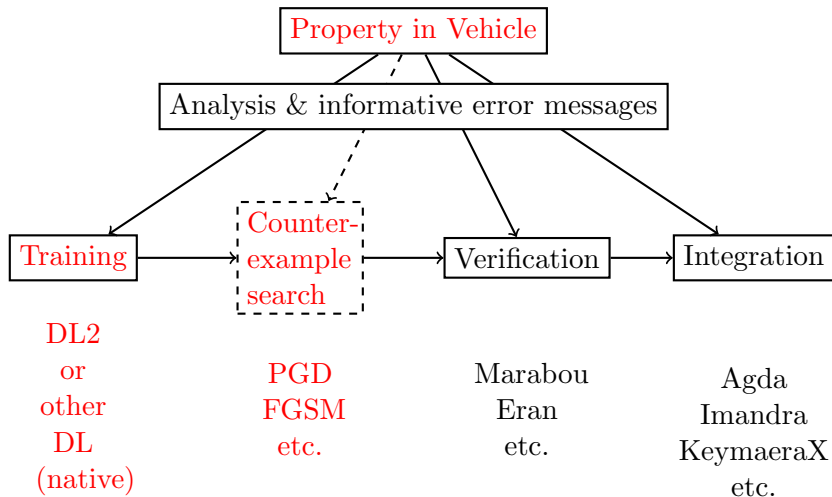
I^O Interoperability ...

I^P Interpretability ...

I^f Integration ...

E^G Embedding gap ...

In Vehicle terms,



Vehicle's formula



Property-driven training =

Differentiable Logic + (PGD) optimisation

Example - differentiable logic



We define a very simple differentiable logic on a toy propositional language

$$a := a \mid p \leq p \mid a \wedge a \mid a \Rightarrow a$$

based on Gödel fuzzy logic [van Krieken 2022].

$$\mathbf{T}(a_1 \leq a_2) := 1 - \max\left(\frac{\mathbf{T}(a_1) - \mathbf{T}(a_2)}{\mathbf{T}(a_1) + \mathbf{T}(a_2)}, 0\right)$$

$$\mathbf{T}(a_1 \wedge a_2) := \min(\mathbf{T}(a_1), \mathbf{T}(a_2))$$

$$\mathbf{T}(a_1 \Rightarrow a_2) := \max(1 - \mathbf{T}(a_1), \mathbf{T}(a_2))$$

Example - translation



$$\mathbf{T}(\|f(\mathbf{x}) - f(\hat{\mathbf{x}})\| \leq \delta) =$$

Example - translation



$$\mathbf{T}(\|f(\mathbf{x}) - f(\hat{\mathbf{x}})\| \leq \delta) = 1 - \max\left(\frac{\mathbf{T}(\|f(\mathbf{x}) - f(\hat{\mathbf{x}})\| - \delta)}{\mathbf{T}(\|f(\mathbf{x}) - f(\hat{\mathbf{x}})\| + \delta)}, 0\right)$$

Different existing DLs



► DL2



Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. DL2: Training and querying neural networks with logic. In ICML'19, pp. 1931–1941.

► fuzzy DLs such as: Godel, Łukasiewicz, Yager, product and others



Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. Artificial Intelligence, 302:103602, 2022.

► Signal Temporal Logic DL



Peter Varnai and Dimos V. Dimarogonas. On robustness metrics for learning stl tasks. In 2020 American Control Conference (ACC), pp. 5394–5399, 2020.

► Formalising and comparing them all



Natalia Slusarz, Ekaterina Komendantskaya, Matthew L. Daggitt, Robert J. Stewart, Kathrin Stark: Logic of Differentiable Logics: Towards a Uniform Semantics of DL. LPAR 2023: 473-493

Design Space for Differentiable Logics



Properties:	DL2	Gödel	Łukasiewicz	Yager	Product	STL
Geometric:						
Weak Smoothness	yes*	no	no	no	yes*	yes
Shadow-lifting	yes	no	no	no	yes	yes
Scale invariance	yes	yes	no	no	no	yes
Logical:						
Idempotence	no	yes	no	no	no	yes
Commutativity	yes	yes	yes	yes	yes	yes
Associativity	yes	yes	yes	yes	yes	no
Quantifier commutativity	no	yes	no	no	no	no
Soundness	yes	yes	yes	yes	yes	no



Recall: With “projected gradient descent” (PGD),

we minimise the loss wrt the worst-case perturbation of each sample in \mathcal{D} .

- Replace the standard training objective with:

$$\min_{\theta} \left[\max_{\mathbf{x}: |\mathbf{x} - \hat{\mathbf{x}}| \leq \epsilon} \mathcal{L}(\mathbf{x}, \mathbf{y}) \right]$$



Recall: With “projected gradient descent” (PGD),

we minimise the loss wrt the worst-case perturbation of each sample in \mathcal{D} .

- Replace the standard training objective with:

$$\min_{\theta} \left[\max_{\mathbf{x}: |\mathbf{x} - \hat{\mathbf{x}}| \leq \epsilon} \mathcal{L}(\mathbf{x}, \mathbf{y}) \right]$$

- in Vehicle, we replace the above training objective with

$$\min_{\theta} \left[\max_{\mathbf{x} \in \mathbb{H}_{\mathcal{P}}} \mathcal{L}_{\mathcal{S}}(\mathbf{x}, \mathbf{y}) \right]$$

where $\mathbb{H}_{\mathcal{P}}$ is a hyper-shape (usually a hyper-rectangle) that corresponds to the pre-condition \mathcal{P} of the property: $\forall \mathbf{x}. \mathcal{P} \Rightarrow \mathcal{S}$; and $\mathcal{L}_{\mathcal{S}}$ is obtained by DL-translation of \mathcal{S} .

Examples: AcasXu



- In AcasXu, $\mathbb{H}_{\mathcal{P}}$ is given by (the normalised) vector bounds given by the property:

$$1500 \leq \rho \leq 1800$$

$$-0.06 \leq \theta \leq 0.06$$

$$\psi \geq 3.10$$

$$v_{own} \geq 980$$

$$v_{int} \geq 960$$

- and $\mathcal{L}_{\mathcal{S}} = \mathbf{T}(\text{not}(\text{minimalScoreclearOfConflict}x))$.

Recall:

```
@property
```

```
property3 : Bool
```

```
property3 = forall x . validInput x and  
                directlyAhead x and  
                movingTowards x =>  
                not (minimalScore clearOfConflict x)
```

Examples: Degenerate case



$$\min_{\theta} [\max_{\mathbf{x} \in \mathbb{H}_{\mathcal{P}}} \mathcal{L}_{\mathcal{S}}(\mathbf{x}, \mathbf{y})]$$

- When we have

$$\forall \mathbf{x}. \mathcal{S}$$

instead of $\forall \mathbf{x}. \mathcal{P} \Rightarrow \mathcal{S}$

- $\mathbb{H}_{\mathcal{P}}$ is only constrained by normalisation boundaries, e.g. by $[0, 0, \dots, 0]$ and $[1, 1, \dots, 1]$ if we normalise values between 0 and 1.



$$\min_{\theta} [\max_{\mathbf{x} \in \mathbb{H}_{\mathcal{P}}} \mathcal{L}_{\mathcal{S}}(\mathbf{x}, \mathbf{y})]$$

- ▶ When we have

$$\forall \mathbf{x}. |\mathbf{x} - \hat{\mathbf{x}}| \leq \epsilon \Rightarrow |f(\mathbf{x}) - f(\hat{\mathbf{x}})| \leq \delta$$

- ▶ $\mathbb{H}_{\mathcal{P}}$ is the ϵ -cube around $\hat{\mathbf{x}}$
- ▶ and, in case we use the Gödel DL, we get the

$$\mathcal{L}_{\mathcal{S}} = \mathbf{T}(\|f(\mathbf{x}) - f(\hat{\mathbf{x}})\| \leq \delta) = 1 - \max\left(\frac{\mathbf{T}(\|f(\mathbf{x}) - f(\hat{\mathbf{x}})\| - \delta)}{\mathbf{T}(\|f(\mathbf{x}) - f(\hat{\mathbf{x}})\| + \delta)}, 0\right)$$

Vehicle's way to implement this



- ▶ You use the same specification (“VCL file”) as for verification

See the Demo next

Vehicle's way to implement this



- ▶ You use the same specification (“VCL file”) as for verification
- ▶ You need a Tensorflow version of the network you wish to train

See the Demo next

Vehicle's way to implement this



- ▶ You use the same specification (“VCL file”) as for verification
- ▶ You need a Tensorflow version of the network you wish to train
- ▶ Using provided Python template, you call the specification and the network when running the Python Script:

See the Demo next

Vehicle's way to implement this



- ▶ You use the same specification (“VCL file”) as for verification
- ▶ You need a Tensorflow version of the network you wish to train
- ▶ Using provided Python template, you call the specification and the network when running the Python Script:
 - ▶ the loss function \mathcal{L}_S is automatically generated from the spec

See the Demo next

Vehicle's way to implement this



- ▶ You use the same specification (“VCL file”) as for verification
- ▶ You need a Tensorflow version of the network you wish to train
- ▶ Using provided Python template, you call the specification and the network when running the Python Script:
 - ▶ the loss function $\mathcal{L}_{\mathcal{S}}$ is automatically generated from the spec
 - ▶ the hyper-shape $\mathbb{H}_{\mathcal{P}}$ is currently provided by the user;

See the Demo next

Vehicle's way to implement this



- ▶ You use the same specification (“VCL file”) as for verification
- ▶ You need a Tensorflow version of the network you wish to train
- ▶ Using provided Python template, you call the specification and the network when running the Python Script:
 - ▶ the loss function $\mathcal{L}_{\mathcal{S}}$ is automatically generated from the spec
 - ▶ the hyper-shape $\mathbb{H}_{\mathcal{P}}$ is currently provided by the user;
 - ▶ PGD finds counter-examples within $\mathbb{H}_{\mathcal{P}}$

See the Demo next

Vehicle's way to implement this



- ▶ You use the same specification (“VCL file”) as for verification
- ▶ You need a Tensorflow version of the network you wish to train
- ▶ Using provided Python template, you call the specification and the network when running the Python Script:
 - ▶ the loss function $\mathcal{L}_{\mathcal{S}}$ is automatically generated from the spec
 - ▶ the hyper-shape $\mathbb{H}_{\mathcal{P}}$ is currently provided by the user;
 - ▶ PGD finds counter-examples within $\mathbb{H}_{\mathcal{P}}$
 - ▶ Native Python training is used, given the loss function $\mathcal{L}_{\mathcal{S}}$.

See the Demo next

In Conclusion,

