Neural Network Verification With Vehicle: Chapter 4 - Property-Driven Training

ICFP'23 Tutorial

Matthew Daggitt [1] Wen Kokke (online) [2] Ekaterina Komendantskaya[3]

[1]Heriot-Watt University · [2]University of Strathclyde · [3]University of Southampton

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.

$I^{\int}$ 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 McCullogh 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 McCullogh 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 generally:

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

1. Data Augmentation

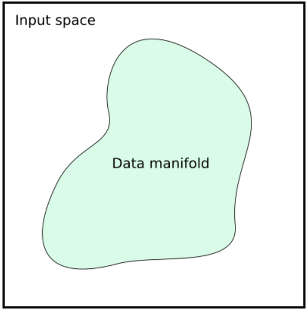
Suppose we are given a data set $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$.
Prior to training, generate new training data samples close to existing data and label them with the same output as the original data.

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

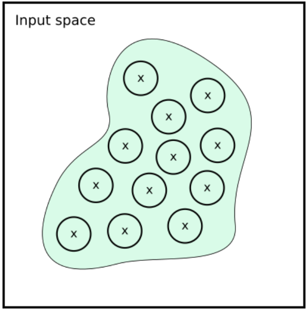
C. Shorten, T.M. Khoshgoftaar: A survey on image data augmentation for deep learning. J. Big Data 6, 60 (2019)



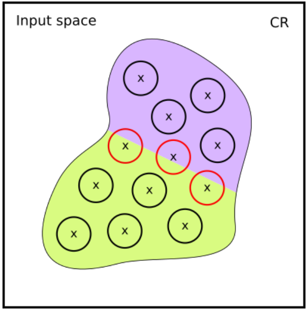
1. Data Augmentation

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

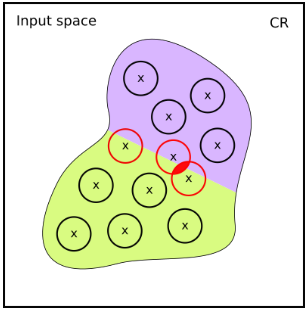
C. Shorten, T.M. Khoshgoftaar: A survey on image data augmentation for deep learning. J. Big Data 6, 60 (2019)



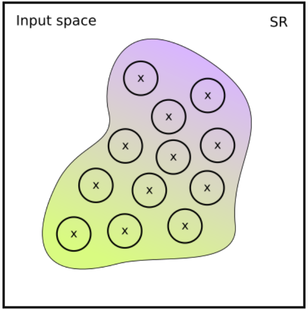
However,



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 $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ (with optimisation parameters θ),
a loss function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ 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 $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ (with optimisation parameters θ),
a loss function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ 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 $f_\theta : \mathbb{R}^n \to [0,1]^m$, the cross-entropy loss is defined as

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

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 $\mathcal{D}$. It thus solves the optimisation problem:

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

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 $\mathcal{D}$. It thus solves the optimisation problem:

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

- 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:

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

  - 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 $\mathbf{x}$.

Zico Kolter and Aleksander Madry. Adversarial Robustness - Theory and Practice. NeurIPS 2018 tutorial.

Lipshitz 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...

3. Other ways to do property-driven training

More sophisticated approaches to property driven training include:

- Tailoring the neural network architectures
- Tailoring the activation functions
- Including probabilsitic or deterministic solvers into neural network layers
- ...

Eleonora Giunchiglia, Mihaela Catalina Stoian, and Thomas Lukasiewicz. Deep learning with logical constraints. IJCAI 2022, pages 5478–5485. ijcai.org, 2022

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 ...
- $I^\int$ Integration ...
- E^G Embedding gap ...

Interpretation of adversarial training:

Recall the epsilon ball robustness:

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

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

Training style	Definition of *robust*
Data Augmentation	$\mathit{argmax}\ [f(\mathbf{x})] = i$
DL2 training	$f(\mathbf{x})_i \geq \eta$
Adversarial Training	$\lvert f(\mathbf{x}) - f(\hat{\mathbf{x}})\rvert \leq \delta$
Lipschitz Continuity	$\lvert f(\mathbf{x}) - f(\hat{\mathbf{x}})\rvert \leq L\lvert \mathbf{x} - \hat{\mathbf{x}}\rvert$
...	...

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

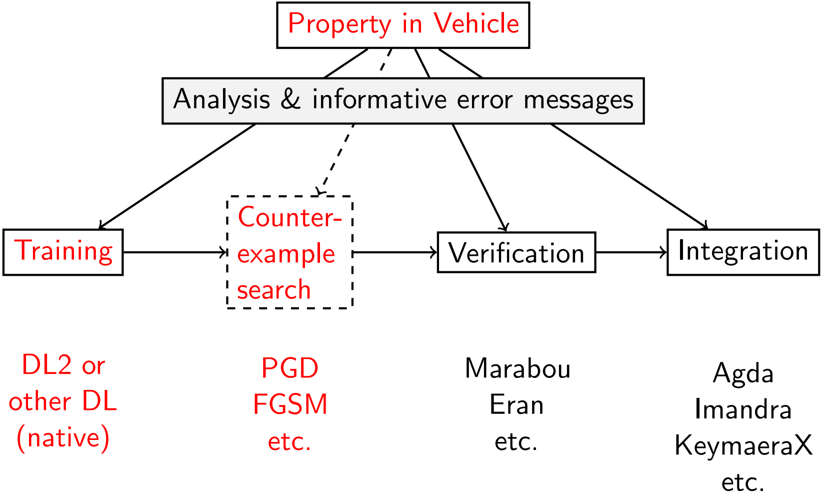
I^O Interoperability . . .

I^P Interpretability . . .

$I^{\int}$ 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 language

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

based on Gödel fuzzy logic [van Krieken 2022].

$$\begin{aligned}
\mathbf{T}(a_1 \leq a_2) &:= 1 - \max(\frac{a_1 - a_2}{a_1 + a_2}, 0) \\
\mathbf{T}(p_1 \wedge p_2) &:= \min(\mathbf{T}(p_1), \mathbf{T}(p_2)) \\
\mathbf{T}(p_1 \Rightarrow p_2) &:= \max(1 - \mathbf{T}(p_1), \mathbf{T}(p_2))
\end{aligned}$$

Example - translation

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

Example - translation

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

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

Vehicle's formula

Property-driven training =

Differentiable Logic + (PGD) optimisation

Optimisation

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:

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

Optimisation

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:

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

- in Vehicle, given a property $\forall \mathbf{x}.\mathcal{P}(\mathbf{x}) \Rightarrow \mathcal{S}(\mathbf{x})$, we replace the above training objective with

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

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

Examples: AcasXU

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

- $\mathbb{H}_{\mathcal{P}(\mathbf{x})}$ is given by (the normalised) vector bounds given by:

$$
\begin{aligned}
&1500 \leq \rho \leq 1800 \\
&-0.06 \leq \theta \leq 0.06 \\
&\psi \geq 3.10 \\
&v_{own} \geq 980 \\
&v_{int} \geq 960
\end{aligned}
$$

- Recall that for Acas Xu networks, $\mathbf{x} = [\rho, \theta, \psi, v_{own}, v_{int}]^{norm}$

Examples: AcasXU

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

- $\mathbb{H}_{\mathcal{P}(\mathbf{x})}$ is given by (the normalised) vector bounds given by:

$$\begin{aligned} &1500 \leq \rho \leq 1800 \\ &-0.06 \leq \theta \leq 0.06 \\ &\psi \geq 3.10 \\ &v_{own} \geq 980 \\ &v_{int} \geq 960 \end{aligned}$$

- Recall that for Acas Xu networks, $\mathbf{x} = [\rho, \theta, \psi, v_{own}, v_{int}]^{norm}$
- and $\mathcal{L}_{\mathcal{S}(\mathbf{x})} = \mathbf{T}(\texttt{not (minimalScore clearOfConflict } \mathbf{x}\texttt{)})$.

Examples: Degenerate case

- When we have

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

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

Recall we solve the optimisation problem

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

- $\mathbb{H}_{\mathcal{P}(\mathbf{x})} = \mathbb{H}_{\mathbf{x}}$ is the domain of $\mathbf{x}$ (usually constrained by normalisation boundaries, e.g. by $[0, 0, \ldots, 0]$ and $[1, 1, \ldots, 1]$.)

Examples: standard robustness

- When we have

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

Recall we solve the optimisation problem

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

- $\mathbb{H}_{\mathcal{P}(x)}$ is the ϵ-cube around $\hat{\mathbf{x}}$
- and, in case we use the Gödel DL, we get $\mathcal{L}_{\mathcal{S}(\mathbf{x})} = \mathbf{T}(||f(\mathbf{x}) - f(\hat{\mathbf{x}})|| \leq \delta) =$ $1 - \max(\dfrac{\mathbf{T}(||f(\mathbf{x}) - f(\hat{\mathbf{x}})|| - \delta)}{\mathbf{T}(||f(\mathbf{x}) - f(\hat{\mathbf{x}})|| + \delta)}, 0)$

Vehicle's way to implement this

- You use the same specification ("VCL file") as for verification

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

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:

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 $\mathcal{L}_{\mathcal{S}(\mathbf{x})}$ is automatically generated from the spec

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 $\mathcal{L}_{\mathcal{S}(\mathbf{x})}$ is automatically generated from the spec
  - the hyper-shape $\mathbb{H}_{\mathcal{P}(\mathbf{x})}$ is currently provided by the user;

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 $\mathcal{L}_{\mathcal{S}(\mathbf{x})}$ is automatically generated from the spec
  - the hyper-shape $\mathbb{H}_{\mathcal{P}(\mathbf{x})}$ is currently provided by the user;
  - PGD finds counter-examples within $\mathbb{H}_{\mathcal{P}(\mathbf{x})}$

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 $\mathcal{L}_{\mathcal{S}(\mathbf{x})}$ is automatically generated from the spec
  - the hyper-shape $\mathbb{H}_{\mathcal{P}(\mathbf{x})}$ is currently provided by the user;
  - PGD finds counter-examples within $\mathbb{H}_{\mathcal{P}(\mathbf{x})}$
  - native (SGD) training (in Tensorflow) is used, given the loss function $\mathcal{L}_{\mathcal{S}(\mathbf{x})}$,

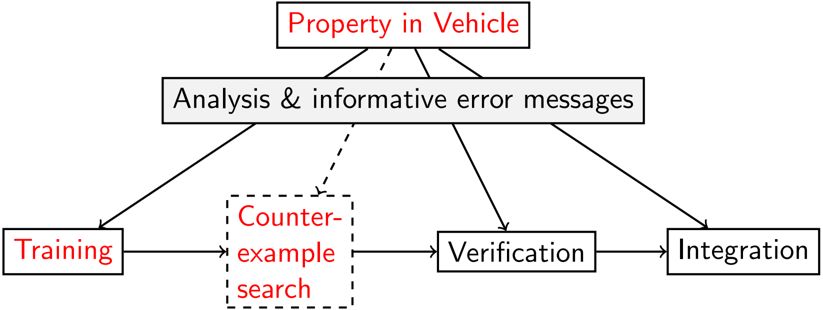
Vehicle's way to implement this

- 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 $\mathcal{L}_{\mathcal{S}(\mathbf{x})}$ is automatically generated from the spec
  - the hyper-shape $\mathbb{H}_{\mathcal{P}(\mathbf{x})}$ is currently provided by the user;
  - PGD finds counter-examples within $\mathbb{H}_{\mathcal{P}(\mathbf{x})}$
  - native (SGD) training (in Tensorflow) is used, given the loss function $\mathcal{L}_{\mathcal{S}(\mathbf{x})}$,

...to solve the optimisation problem

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

In Conclusion,



DL2 or other DL (native)	PGD FGSM etc.	Marabou Eran etc.	Agda Imandra KeymaeraX etc.

We covered:

- I^O Interoperability – properties are not portable between training/counter-example search/ verification.
- I^P Interpretability – code is not easy to understand.
- $I^\int$ 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.

Next

- I^O Interoperability – properties are not portable between training/counter-example search/ verification.
- I^P Interpretability – code is not easy to understand.
- $I^\int$ 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.