Vehicle Tutorial Chapter 1: Getting Started

Ekaterina Komendantskaya and Matthew Daggitt and Wen Kokke and ...

Thank you for your attention! (Vehicle Team)

Matthew Daggitt

Rob Stewart

Natalia Slusarz

Wen Kokke

Luca Arnaboldi

Kathrin Stark

Bob Atkey

Marco Casadio

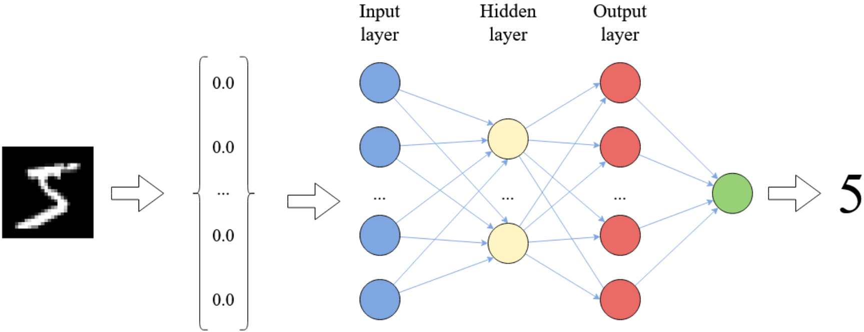
Table of Contents

Neural Network Verification: overview of the new domain

The lifecycle of neural network verification

Introduction to Vehicle (a mini tutorial)

Neural nets for classification



Formally,

a neural network is a function $N : R^n \rightarrow R^m$.

Neural networks

… are ideal for “perception” tasks:

- approximate functions when exact solution is hard to get
- tolerant to noisy and incomplete data

Neural networks

… are ideal for "perception" tasks:

- approximate functions when exact solution is hard to get
- tolerant to noisy and incomplete data

Neural networks

... are ideal for "perception" tasks:

- approximate functions when exact solution is hard to get
- tolerant to noisy and incomplete data

Neural networks

… are ideal for "perception" tasks:

- approximate functions when exact solution is hard to get
- tolerant to noisy and incomplete data

BUT

- solutions not easily conceptualised (lack of explainability)
- prone to a new range of safety and security problems:

Neural networks

… are ideal for “perception” tasks:

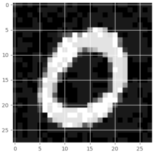
- approximate functions when exact solution is hard to get
- tolerant to noisy and incomplete data

BUT

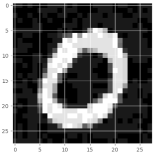
- solutions not easily conceptualised (lack of explainability)
- prone to a new range of safety and security problems:

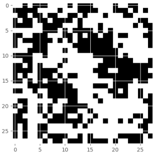
  adversarial attacks
  data poisoning
  catastrophic forgetting

One example: Adversarial Attacks

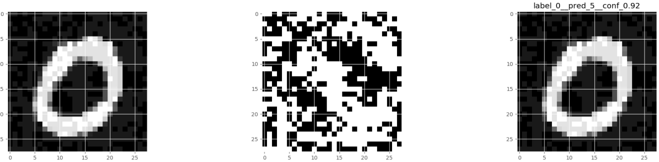


One example: Adversarial Attacks

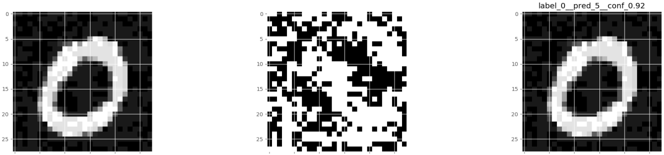




One example: Adversarial Attacks

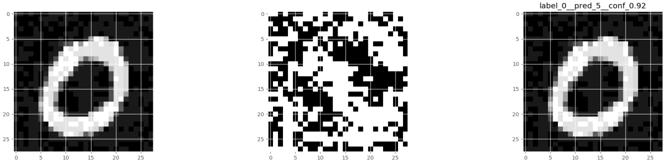


One example: Adversarial Attacks



the perturbations are imperceptible to human eye

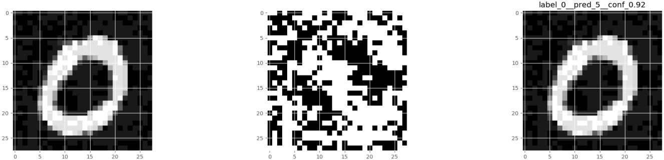
One example: Adversarial Attacks



the perturbations are imperceptible to human eye

attacks transfer from one neural network to another

One example: Adversarial Attacks

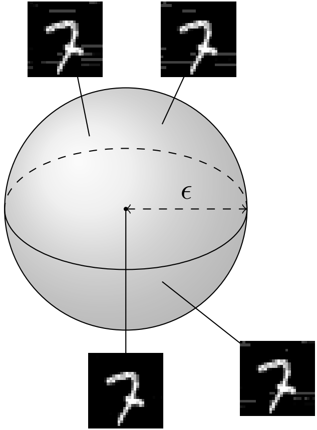


the perturbations are imperceptible to human eye

attacks transfer from one neural network to another

affect any domain where neural networks are applied

Verification Property: "ϵ-ball robustness"



An ϵ-ball $\mathbb{B}(\hat{\mathbf{x}}, \epsilon) = \{\mathbf{x} \in \mathbb{R}^n : |\hat{\mathbf{x}} - \mathbf{x}| \leq \epsilon\}$

Classify all points in $\mathbb{B}(\hat{\mathbf{x}}, \epsilon)$ "robustly".

Another example property: ACAS Xu

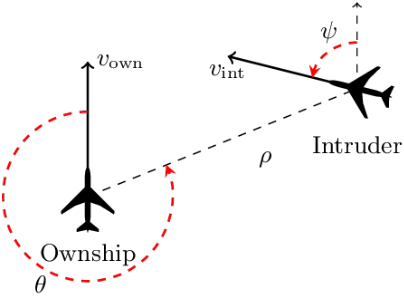
A collision avoidance system for unmanned autonomous aircraft.

Inputs:

- Distance to intruder, ρ
- Angle to intruder, θ
- Intruder heading, φ
- Speed, v_{own}
- Intruder speed, v_{int}

Outputs:

- Clear of conflict
- Strong left
- Weak left
- Weak right
- Strong right



ACAS Xu

The system was originally implemented as a 2Gb lookup table but was replaced with a neural network in order to improve size and latency requirements.

ACAS Xu

The system was originally implemented as a 2Gb lookup table but was replaced with a neural network in order to improve size and latency requirements.

10 different specified properties in total.

ACAS Xu

The system was originally implemented as a 2Gb lookup table but was replaced with a neural network in order to improve size and latency requirements.

10 different specified properties in total.

Definition (ACAS Xu: Property 1)

If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.

ACAS Xu

The system was originally implemented as a 2Gb lookup table but was replaced with a neural network in order to improve size and latency requirements.
10 different specified properties in total.

Definition (ACAS Xu: Property 1)

If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.

$$(\rho \geq 55947.691) \wedge (v_{own} \geq 1145) \wedge (v_{int} \leq 60)$$
$$\Rightarrow \text{the score for COC is at most } 1500$$

More Generally

Given $N : R^n \to R^m$

Verification of such functions most commonly boils down to specifying admissible intervals for the function's output given an interval for its inputs.

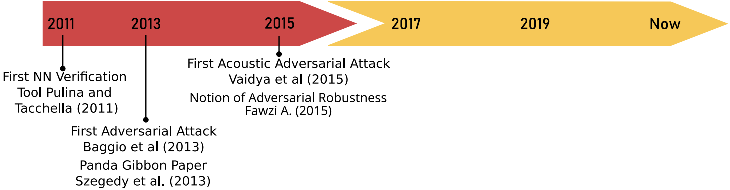
More Generally

Given $N : R^n \to R^m$

Verification of such functions most commonly boils down to specifying admissible intervals for the function's output given an interval for its inputs.

Casadio, M., Komendantskaya, E., Daggitt, M.L., Kokke, W., Katz, G., Amir, G., Refaeli, I.: Neural network robustness as a verification property: A principled case study. In: Computer Aided Verification (CAV 2022).

Overview of The Verification Landscape



2011 2013 2015 2017 2019 Now

First NN Verification Tool Pulina and Tacchella (2011)

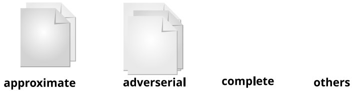
First Adversarial Attack Baggio et al (2013)

Panda Gibbon Paper Szegedy et al. (2013)

First Acoustic Adversarial Attack Vaidya et al (2015)

Notion of Adversarial Robustness Fawzi A. (2015)

What tools are available?
2015



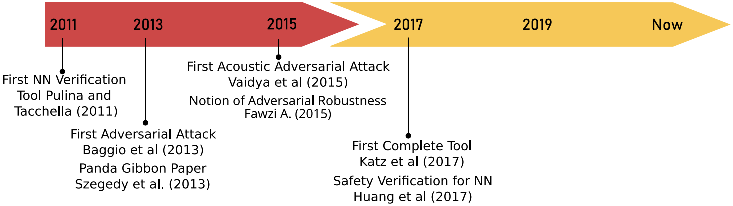
approximate

adverserial

complete

others

Overview of The Verification Landscape



2011 — First NN Verification Tool Pulina and Tacchella (2011)

2013 — First Adversarial Attack Baggio et al (2013)

Panda Gibbon Paper Szegedy et al. (2013)

2015 — First Acoustic Adversarial Attack Vaidya et al (2015)

Notion of Adversarial Robustness Fawzi A. (2015)

2017 — First Complete Tool Katz et al (2017)

Safety Verification for NN Huang et al (2017)

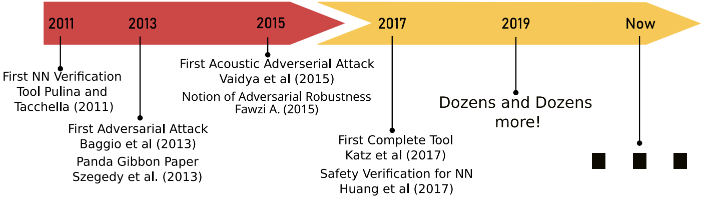
2019

Now

What tools are available?

2017

Overview of The Verification Landscape



2011 2013 2015 2017 2019 Now

First NN Verification Tool Pulina and Tacchella (2011)

First Adversarial Attack Baggio et al (2013)

Panda Gibbon Paper Szegedy et al. (2013)

First Acoustic Adverserial Attack Vaidya et al (2015)

Notion of Adversarial Robustness Fawzi A. (2015)

First Complete Tool Katz et al (2017)

Safety Verification for NN Huang et al (2017)

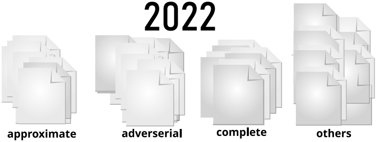
Dozens and Dozens more!

■ ■ ■

I have this specification
I want to verify!

What tools are available?

2022



Current Verifier Landscape

A whole range of domain-specific verifiers exist:

Current Verifier Landscape

A whole range of domain-specific verifiers exist:

- Marabou (SMT technology)

Current Verifier Landscape

A whole range of domain-specific verifiers exist:

- Marabou (SMT technology)
- ERAN (abstract interpretation + MILP)

Current Verifier Landscape

A whole range of domain-specific verifiers exist:

- Marabou (SMT technology)
- ERAN (abstract interpretation + MILP)
- Verisig (interval arithmetic)
- AlphaBetaCROWN (linear bound propagation)
- ...

International Standards and Competitions

https://www.vnnlib.org/

Current Verifier Landscape

A whole range of domain-specific verifiers exist:

- Marabou (SMT technology)
- ERAN (abstract interpretation + MILP)
- Verisig (interval arithmetic)
- AlphaBetaCROWN (linear bound propagation)
- ...

International Standards and Competitions

https://www.vnnlib.org/

Marabou is our choice as it is complete, and the set of expressible queries is large!

Guy Katz, Clarke Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In CAV, 2017.

Table of Contents

Neural Network Verification: overview of the new domain

The lifecycle of neural network verification

Introduction to Vehicle (a mini tutorial)

The lifecycle of neural network verification

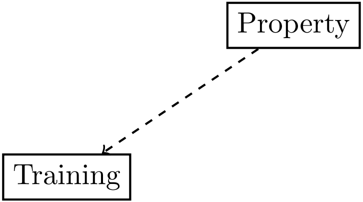
Property

The lifecycle of neural network verification

Property

Training

The lifecycle of neural network verification

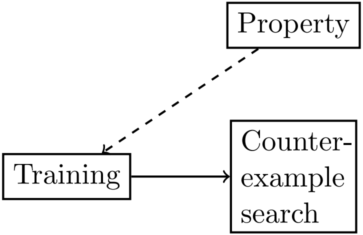


Property

Training

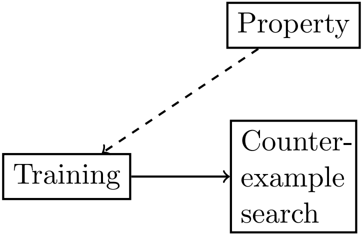
DL2
ACT
etc.

The lifecycle of neural network verification



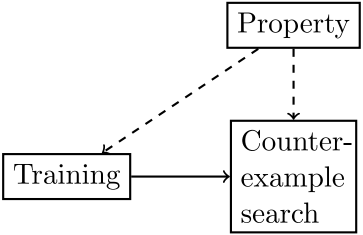
DL2
ACT
etc.

The lifecycle of neural network verification



DL2
ACT
etc.

The lifecycle of neural network verification



Property

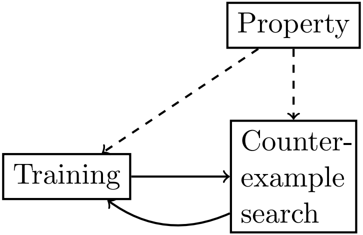
Training

Counter-example search

DL2
ACT
etc.

PGD
FGSM
etc.

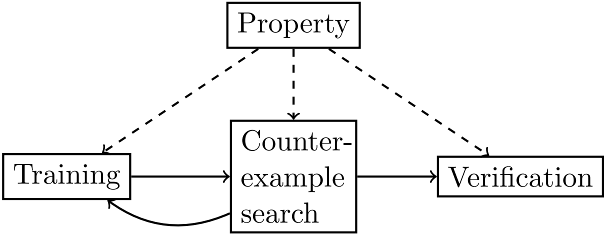
The lifecycle of neural network verification



DL2
ACT
etc.

PGD
FGSM
etc.

The lifecycle of neural network verification

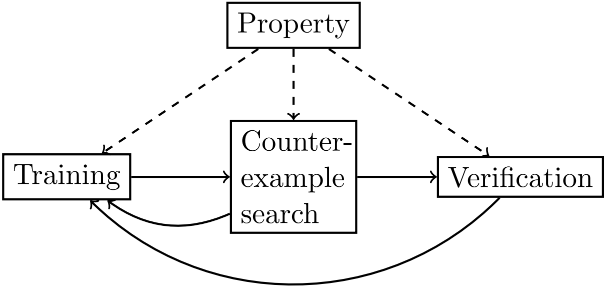


DL2
ACT
etc.

PGD
FGSM
etc.

Marabou
Eran
etc.

The lifecycle of neural network verification

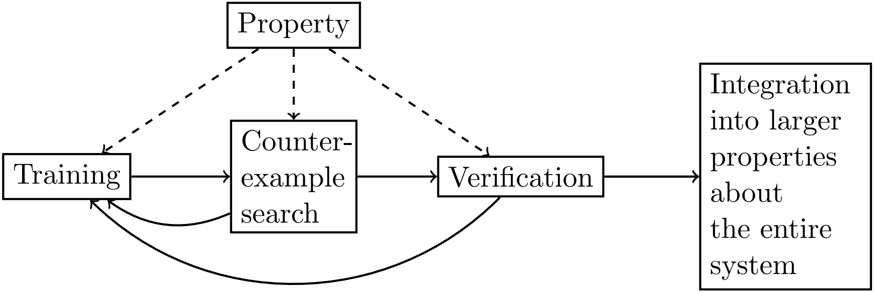


DL2
ACT
etc.

PGD
FGSM
etc.

Marabou
Eran
etc.

The lifecycle of neural network verification



Property

Training

Counter-example search

Verification

Integration into larger properties about the entire system

DL2
ACT
etc.

PGD
FGSM
etc.

Marabou
Eran
etc.

Challenges the area faces

- Theory: finding appropriate verification properties

Challenges the area faces

- Theory: finding appropriate verification properties
- Solvers: undecidability of non-linear real arithmetic and scalability of neural network verifiers

Challenges the area faces

- Theory: finding appropriate verification properties
- Solvers: undecidability of non-linear real arithmetic and scalability of neural network verifiers
- ML: understanding and integrating property-driven training

Challenges the area faces

- Theory: finding appropriate verification properties
- Solvers: undecidability of non-linear real arithmetic and scalability of neural network verifiers
- ML: understanding and integrating property-driven training
- Programming: finding the right languages to support these developments

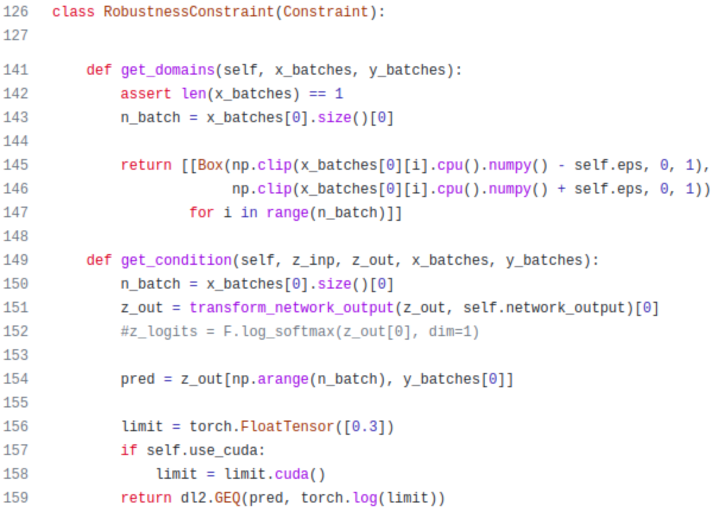
Challenges the area faces

- Theory: finding appropriate verification properties
- Solvers: undecidability of non-linear real arithmetic and scalability of neural network verifiers
- ML: understanding and integrating property-driven training
- Programming: finding the right languages to support these developments
- Complex systems: integration of neural net verification into complex systems

How BIG is the problem?

Lets look under the hood...

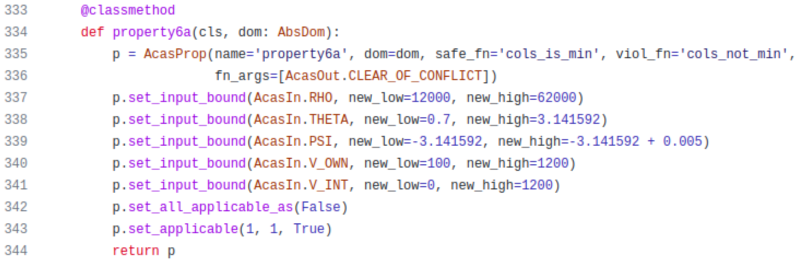
Training framework: DL2



```
class RobustnessConstraint(Constraint):


    def get_domains(self, x_batches, y_batches):
        assert len(x_batches) == 1
        n_batch = x_batches[0].size()[0]

        return [[Box(np.clip(x_batches[0][i].cpu().numpy() - self.eps, 0, 1),
                     np.clip(x_batches[0][i].cpu().numpy() + self.eps, 0, 1))
                for i in range(n_batch)]]

    def get_condition(self, z_inp, z_out, x_batches, y_batches):
        n_batch = x_batches[0].size()[0]
        z_out = transform_network_output(z_out, self.network_output)[0]
        #z_logits = F.log_softmax(z_out[0], dim=1)

        pred = z_out[np.arange(n_batch), y_batches[0]]

        limit = torch.FloatTensor([0.3])
        if self.use_cuda:
            limit = limit.cuda()
        return dl2.GEQ(pred, torch.log(limit))
```

Fischer, M., Balunovic, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., and Vechev, M. T. DL2: training and querying neural networks with logic. In Proc. of the 36th Int. Conf. Machine Learning, ICML 2019

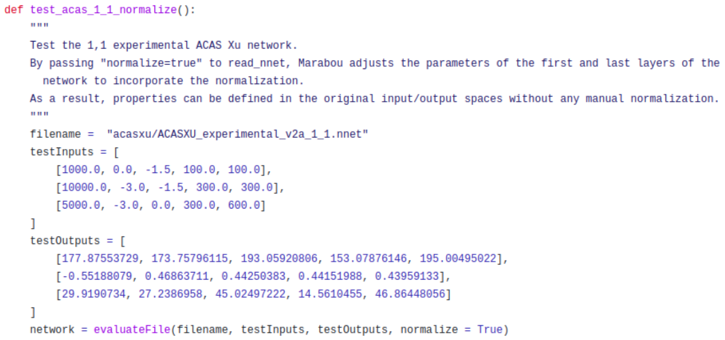
Training framework: ART



```
    @classmethod
    def property6a(cls, dom: AbsDom):
        p = AcasProp(name='property6a', dom=dom, safe_fn='cols_is_min', viol_fn='cols_not_min',
                     fn_args=[AcasOut.CLEAR_OF_CONFLICT])
        p.set_input_bound(AcasIn.RHO, new_low=12000, new_high=62000)
        p.set_input_bound(AcasIn.THETA, new_low=0.7, new_high=3.141592)
        p.set_input_bound(AcasIn.PSI, new_low=-3.141592, new_high=-3.141592 + 0.005)
        p.set_input_bound(AcasIn.V_OWN, new_low=100, new_high=1200)
        p.set_input_bound(AcasIn.V_INT, new_low=0, new_high=1200)
        p.set_all_applicable_as(False)
        p.set_applicable(1, 1, True)
        return p
```

Lin, X., Zhu, H., Samanta, R., and Jagannathan, S. (2020). Art: Abstraction refinement-guided training for provably correct neural networks. In FMCAD 2020

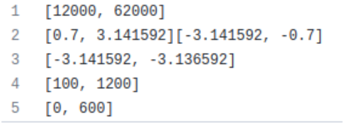
Verification framework: Marabou



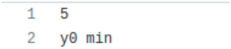
```
def test_acas_1_1_normalize():
    """
    Test the 1,1 experimental ACAS Xu network.
    By passing "normalize=true" to read_nnet, Marabou adjusts the parameters of the first and last layers of the
      network to incorporate the normalization.
    As a result, properties can be defined in the original input/output spaces without any manual normalization.
    """
    filename =  "acasxu/ACASXU_experimental_v2a_1_1.nnet"
    testInputs = [
        [1000.0, 0.0, -1.5, 100.0, 100.0],
        [10000.0, -3.0, -1.5, 300.0, 300.0],
        [5000.0, -3.0, 0.0, 300.0, 600.0]
    ]
    testOutputs = [
        [177.87553729, 173.75796115, 193.05920806, 153.07876146, 195.00495022],
        [-0.55188079, 0.46863711, 0.44250383, 0.44151988, 0.43959133],
        [29.9190734, 27.2386958, 45.02497222, 14.5610455, 46.86448056]
    ]
    network = evaluateFile(filename, testInputs, testOutputs, normalize = True)
```

Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D. L., Kochenderfer, M. J., and Barrett, C. W. (2019). The Marabou framework for verification and analysis of deep neural networks. In CAV 2019

Verification framework: ERAN



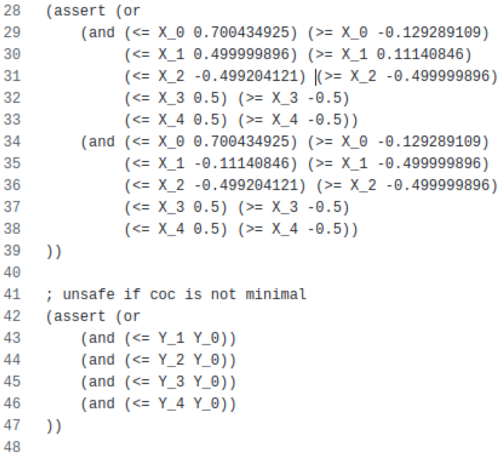
```
[12000, 62000]
[0.7, 3.141592][-3.141592, -0.7]
[-3.141592, -3.136592]
[100, 1200]
[0, 600]
```



```
5
y0 min
```

Singh, G., Gehr, T., Püschel, M., and Vechev, M. T. (2019). An abstract domain for certifying neural networks. PACMPL, 3(POPL):41:1–41:30.

Verification property language: VNNLIB



```
(assert (or
    (and (<= X_0 0.700434925) (>= X_0 -0.129289109)
         (<= X_1 0.499999896) (>= X_1 0.11140846)
         (<= X_2 -0.499204121) (>= X_2 -0.499999896)
         (<= X_3 0.5) (>= X_3 -0.5)
         (<= X_4 0.5) (>= X_4 -0.5))
    (and (<= X_0 0.700434925) (>= X_0 -0.129289109)
         (<= X_1 -0.11140846) (>= X_1 -0.499999896)
         (<= X_2 -0.499204121) (>= X_2 -0.499999896)
         (<= X_3 0.5) (>= X_3 -0.5)
         (<= X_4 0.5) (>= X_4 -0.5))
))

; unsafe if coc is not minimal
(assert (or
    (and (<= Y_1 Y_0))
    (and (<= Y_2 Y_0))
    (and (<= Y_3 Y_0))
    (and (<= Y_4 Y_0))
))
```

Recap: What are the problems?

Recap: What are the problems?

- Interoperability - properties are not portable between training/counter-example search/ verification.

Recap: What are the problems?

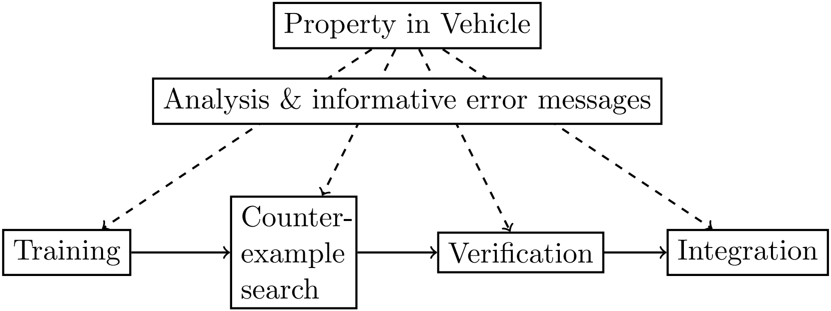
- Interoperability - properties are not portable between training/counter-example search/ verification.
- Interpretability - code is not easy to understand.

Recap: What are the problems?

- Interoperability - properties are not portable between training/counter-example search/ verification.
- Interpretability - code is not easy to understand.
- Integration - properties of networks cannot be linked to larger control system properties.

Vehicle ...

is a domain-specific functional language for writing high-level property specifications for neural networks



Property in Vehicle

Analysis & informative error messages

Training

Counter-example search

Verification

Integration

DL2
ACT
etc.

PGD
FGSM
etc.

Marabou
Eran
etc.

Agda
Imandra
KeymaeraX
etc.

Table of Contents

Neural Network Verification: overview of the new domain

The lifecycle of neural network verification

Introduction to Vehicle (a mini tutorial)

Types

Let us build the ACAS Xu specification.
We start with types of input and output vectors, as well as types of ACAS Xu networks

```
type InputVector = Vector Rat 5
type OutputVector = Vector Rat 5

@network
acasXu : InputVector -> OutputVector
```

The Vector type represents a mathematical vector, or in programming terms can be thought of as a fixed-length array.

Values

Types for values are automatically inferred by **Vehicle**. For example, we can declare the number π and its type will be inferred as rational:

```
pi = 3.141592
```

Working with Vectors

- some input or output pre-processing maybe expected when defining a neural network.

Example

It is assumed that the ACAS Xu inputs and outputs are normalised, i.e. the network does not work directly with units like m/s. However, the specifications we want to write should ideally concern the original units.

Working with Vectors

- some input or output pre-processing maybe expected when defining a neural network.

Example

It is assumed that the ACAS Xu inputs and outputs are normalised, i.e. the network does not work directly with units like m/s. However, the specifications we want to write should ideally concern the original units.

- This is an instance of “ problem space / input space mismatch”
- ... that is very common in neural net verification
- Being able to reason about problem space (alongside the input space) is a feature that distinguishes **Vehicle** from majority of the mainstream neural network verifiers

Vector Normalisation

For clarity, we define a new type synonym for unnormalised input vectors which are in the problem space.

```
type UnnormalisedInputVector = Vector Rat 5
```

Next we define the range of the inputs that the network is designed to work over.

```
minimumInputValues : UnnormalisedInputVector
minimumInputValues = [0,0,0,0,0]

maximumInputValues : UnnormalisedInputVector
maximumInputValues = [60261.0, 2*pi, 2*pi, 1100.0, 1200.0]

meanScalingValues : UnnormalisedInputVector
meanScalingValues = [19791.091, 0.0, 0.0, 650.0, 600.0]
```

Vector manipulation

An alternative method to vector definition is to use the ‘foreach‘ constructor, which is used to provide a value for each ‘index i‘.

```
minimumInputValues : UnnormalisedInputVector
minimumInputValues = foreach i . 0
```

Let us see how ‘foreach‘ works with vector indexing.
We can now define the normalisation function that takes an input vector and returns the unnormalised version.

```
normalise : UnnormalisedInputVector -> InputVector
normalise x = foreach i .
  (x ! i - meanScalingValues ! i) / (maximumInputValues ! i)
```

Vector manipulation

An alternative method to vector definition is to use the ‘foreach‘ constructor, which is used to provide a value for each ‘index i‘.

```
minimumInputValues : UnnormalisedInputVector
minimumInputValues = foreach i . 0
```

Let us see how ‘foreach‘ works with vector indexing.
We can now define the normalisation function that takes an input vector and returns the unnormalised version.

```
normalise : UnnormalisedInputVector -> InputVector
normalise x = foreach i .
  (x ! i - meanScalingValues ! i) / (maximumInputValues ! i)
```

... our first acquaintance with functions!

Functions and Types

```
<name> : <type>
<name> [<args>] = <expr>
```

Functions make up the backbone of the **Vehicle** language.

Functions and Types

```
<name> : <type>
<name> [<args>] = <expr>
```

Functions make up the backbone of the **Vehicle** language.

```
validInput : UnnormalisedInputVector -> Bool
validInput x = forall i .
  minimumInputValues ! i <= x ! i <= maximumInputValues ! i
```

Functions and Types

```
<name> : <type>
<name> [<args>] = <expr>
```

Functions make up the backbone of the **Vehicle** language.

```
validInput : UnnormalisedInputVector -> Bool
validInput x = forall i .
  minimumInputValues ! i <= x ! i <= maximumInputValues ! i
```

Our first acquaintance with quantifiers!

One of the main advantages of **Vehicle** is that it can be used to state and prove specifications that describe the network's behaviour over an infinite set of values.

Functions and Types

Function Composition: Exercise!

```
normAcasXu : UnnormalisedInputVector -> OutputVector
normAcasXu x = acasXu (normalise x)
```

Pre-defined functions and predicates

We have already used:

```
*
/
!
<=
```

Exercise

What do they stand for?

Lets verify ACAS Xu!

```
distanceToIntruder = 0   -- measured in metres
angleToIntruder    = 1   -- measured in radians
intruderHeading    = 2   -- measured in radians
speed              = 3   -- measured in metres/second
intruderSpeed      = 4   -- measured in meters/second

clearOfConflict = 0
weakLeft        = 1
weakRight       = 2
strongLeft      = 3
strongRight     = 4
```

The fact that all vector types come annotated with their size means that it is impossible to mess up indexing into vectors, e.g. if you changed ‘distanceToIntruder = 0‘ to ‘distanceToIntruder = 5‘ the specification would fail to type-check.

Property 1

If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.

Property 1

If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.

```
intruderDistantAndSlower : UnnormalisedInputVector -> Bool
intruderDistantAndSlower x =
  x ! distanceToIntruder >= 55947.691 and
  x ! speed              >= 1145      and
  x ! intruderSpeed      <= 60
```

Property 1

If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.

```
intruderDistantAndSlower : UnnormalisedInputVector -> Bool
intruderDistantAndSlower x =
  x ! distanceToIntruder >= 55947.691 and
  x ! speed              >= 1145      and
  x ! intruderSpeed      <= 60
```

Exercise!

1. Can you identify whether the specification is written in terms of input space or problem space? How do you know?
2. Can you spot more pre-defined **Vehicle** functions? What are they?

There is little left to do!

If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.

There is little left to do!

If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.

```
@property
property1 : Bool
property1 = forall x .
   validInput x and intruderDistantAndSlower x =>
           normAcasXu x ! clearOfConflict <= 1500
```

There is little left to do!

If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.

```
@property
property1 : Bool
property1 = forall x .
   validInput x and intruderDistantAndSlower x =>
           normAcasXu x ! clearOfConflict <= 1500
```

Exercise!

1. Can you guess the purpose of the syntax

   ```
   @property
   ```

   ?
2. What kind of domain ‘forall‘ ranges over? Is it finite or infinite?

How to run Vehicle

Checklist

1. a verifier installed (Marabou);
2. the actual network is supplied in an ONNX format
3. **Vehicle** is installed.

How to run Vehicle

Checklist

1. a verifier installed (Marabou);
2. the actual network is supplied in an ONNX format
3. **Vehicle** is installed.

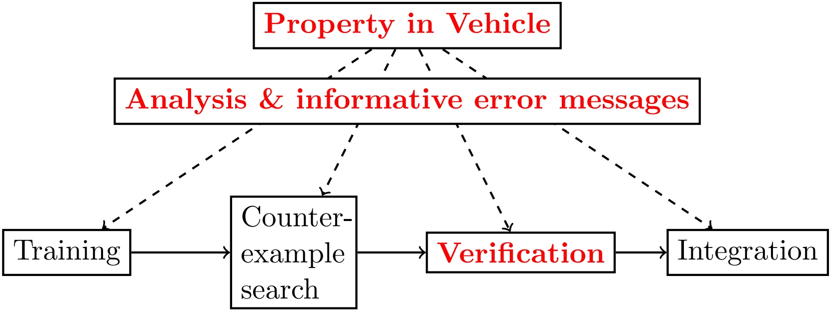
```
vehicle --compileAndVerify
  --specification acasXu.vcl
  --verifier Marabou
  --network acasXu:acasXu_1_7.onnx
  --property property1

--------------------------------------------------
Verifying properties:
  property1 [===========================] 1/1 queries complete
Result: true
   property1
```

Exercise: ϵ-ball Robustness

```
type Image = Tensor Rat [28, 28]
type Label = Index 10
validImage : Image -> Bool
validImage x = forall i j . 0 <= x ! i ! j <= 1

@network
classifier : Image -> Vector Rat 10

advises : Image -> Label -> Bool
advises x i = forall j . j != i => classifier x ! i > classifier x ! j

@parameter
epsilon : Rat

boundedByEpsilon : Image -> Bool
boundedByEpsilon x = forall i j . -epsilon <= x ! i ! j <= epsilon

robustAround : Image -> Label -> Bool
robustAround image label = forall pertubation .
  let perturbedImage = image - pertubation in
  boundedByEpsilon pertubation and validImage perturbedImage =>
    advises perturbedImage label

@dataset
trainingImages : Vector Image n

@dataset
trainingLabels : Vector Label n

@property
robust : Vector Bool n
robust = foreach i . robustAround (trainingImages ! i) (trainingLabels ! i)
```

Vehicle ...

is a domain-specific functional language for writing high-level property specifications for neural networks



Property in Vehicle

Analysis & informative error messages

Training

Counter-example search

Verification

Integration

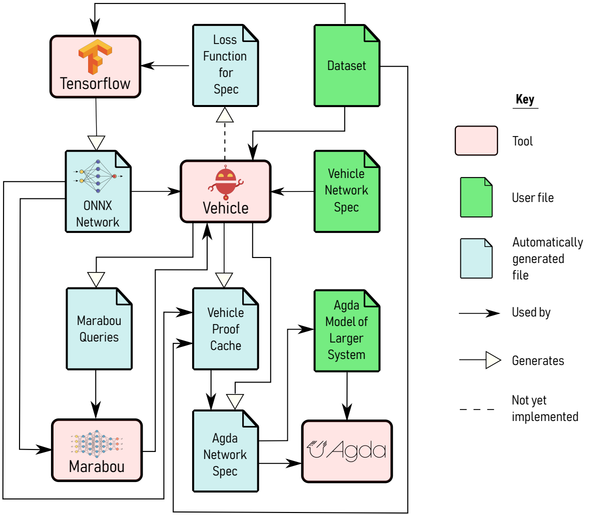
DL2
ACT
etc.

PGD
FGSM
etc.

Marabou
Eran
etc.

Agda
Imandra
KeymaeraX
etc.

Vehicle Architecture



Tensorflow

Loss Function for Spec

Dataset

ONNX Network

Vehicle

Vehicle Network Spec

Marabou Queries

Vehicle Proof Cache

Agda Model of Larger System

Marabou

Agda Network Spec

Agda

Key

Tool

User file

Automatically generated file

Used by

Generates

Not yet implemented

Sources

M. Daggitt, R. Atkey, W. Kokke, E. Komendantskaya, L. Arnaboldi: Compiling Higher-Order Specifications to SMT Solvers: How to Deal with Rejection Constructively. CPP 2023

N. Slusarz, E. Komendantskaya, M. Daggitt, R. Stewart, K. Stark: Logic of Differentiable Logics: Towards a Uniform Semantics of DL. LPAR 2023.

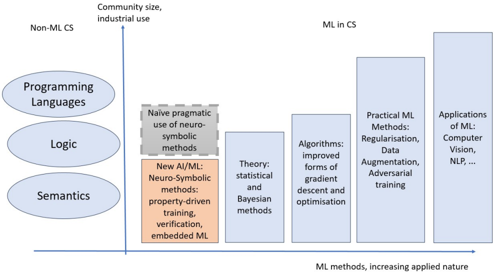
Matthew L. Daggitt, Wen Kokke, Robert Atkey, Luca Arnaboldi, Ekaterina Komendantskaya: Vehicle: Interfacing Neural Network Verifiers with Interactive Theorem Provers. FOMLAS

Vehicle Team: The Vehicle language: `https://github.com/vehicle-lang` 2023.

M.Daggitt and W.Kokke: Vehicle User Manual. 2023.

Which challenges Vehicle addresses

- Theory: finding appropriate verification properties
- Solvers: undecidability of non-linear real arithmetic and scalability of neural network verifiers
- ML: understanding and integrating property-driven training
- Programming: finding the right languages to support these developments
- Complex systems: integration of neural net verification into complex systems


Community size, industrial use
Non-ML CS
ML in CS
Programming Languages
Logic
Semantics
Naïve pragmatic use of neuro-symbolic methods
New AI/ML: Neuro-Symbolic methods: property-driven training, verification, embedded ML
Theory: statistical and Bayesian methods
Algorithms: improved forms of gradient descent and optimisation
Practical ML Methods: Regularisation, Data Augmentation, Adversarial training
Applications of ML: Computer Vision, NLP, ...
ML methods, increasing applied nature