Continuous Verification of Machine Learning

a Declarative Programming Approach

Ekaterina Komendantskaya,
joint work with Daniel Kienitz and Wen Kokke

Lab for AI and Verification, Heriot-Watt University, Scotland

Invited talk at PPDP 2020

Outline

Verification of AI: Overview and Motivation

Outline

Verification of AI: Overview and Motivation

Why Verifying Neural Networks?

Outline

Verification of AI: Overview and Motivation

Why Verifying Neural Networks?

Challenges of Neural Network Verification

Outline

Verification of AI: Overview and Motivation

Why Verifying Neural Networks?

Challenges of Neural Network Verification

Continuous Verification

Table of Contents

Pervasive AI...

Pervasive AI...

Autonomous cars

Pervasive AI...

Autonomous cars

Smart Homes

Pervasive AI...

Autonomous cars

Robotics

Smart Homes

Pervasive AI...

Autonomous cars

Smart Homes

Robotics

Chat Bots

Pervasive AI...

Autonomous cars

Smart Homes

Robotics

Chat Bots

...and many more ...

AI is in urgent need of verification: safety, security, robustness to changing conditions and adversarial attacks, ...

Lab for AI and Verification

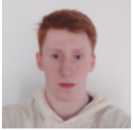
- LAIV launched in March 2019
- ...in order to accumulate local expertise in AI, programming languages, verification
- ... and respond to demand in Edinburgh Robotarium and Edinburgh Center for Robotics

LAIV members:










Perception and Reasoning

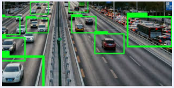
AI methods divide into:

Perception and Reasoning

AI methods divide into:

Perception tasks:

Computer Vision



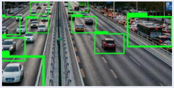
Natural language understanding

Perception and Reasoning

AI methods divide into:

Perception tasks:

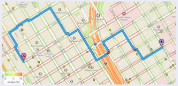
Computer Vision



Natural language understanding

Reasoning tasks:

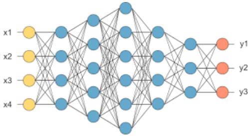
Planning



(Logical) reasoning

A.Hill, E.K. and R.Petrick: Proof-Carrying Plans: a Resource Logic for AI Planning. PPDP'20.

Neural Networks...



take care of
perception tasks:

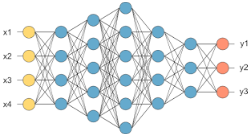
computer vision

speech recognition

pattern recognition

...

Neural Networks...



take care of
perception tasks:

computer vision

speech recognition

pattern recognition

...

In:

autonomous cars

robots

medical applications

chatbots

mobile phone apps

...

Table of Contents

Neural network is

… a function

$$N : \mathbb{R}^n \to \mathbb{R}^m$$

Neural network is

... a function that separate inputs (data points) into classes

Neural network is

... a function that separate inputs (data points) into classes

Suppose we have four data points

	x_1	x_2	y
1	1	1	1
2	1	0	0
3	0	1	0
4	0	0	0

Neural network is

... a function that separate inputs (data points) into classes

Suppose we have four data points

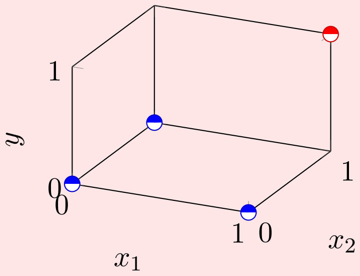
	x_1	x_2	y
1	1	1	1
2	1	0	0
3	0	1	0
4	0	0	0

We may look for a linear function:

$$\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R})$$
$$\text{neuron}\ \ x_1\ x_2 = b + w_{x_1} \times x_1 + w_{x_2} \times x_2$$

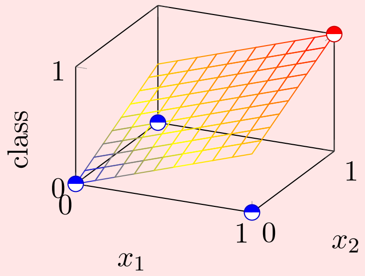
Neural networks are

Plotting these four data points in 3-dimensional space:



Neural network is

... a separating linear function:

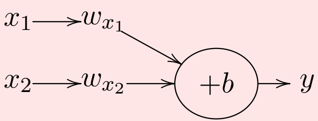


Neural networks are

Taking

$$\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R})$$
$$\text{neuron}\ \ x_1\ x_2 = b + w_{x_1} \times x_1 + w_{x_2} \times x_2$$

here is its neuron view:

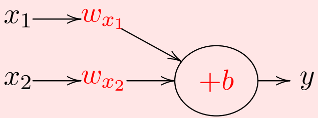


Neural networks are

Taking

$$\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R})$$
$$\text{neuron}\ \ x_1\ x_2 = b + w_{x_1} \times x_1 + w_{x_2} \times x_2$$

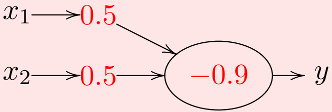
here is its neuron view:



Neural networks are

After running the training algorithm:

$$\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R})$$
$$\text{neuron}\ \ x_1\ x_2 = -0.9 + 0.5 \times x_1 + 0.5 \times x_2$$



(This is one of infinitely many solutions)

Neural networks

Or may be we want to constrain the outputs:

$$\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R}\ \{y = 0 \vee y = 1\})$$
$$\text{neuron}\ \ x_1\ x_2 = S\ (-0.9 + 0.5x_1 + 0.5x_2)$$

Neural networks

Or may be we want to constrain the outputs:

$$\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R}\ \{y = 0 \lor y = 1\})$$
$$\text{neuron}\ \ x_1\ x_2 = S\ (-0.9 + 0.5x_1 + 0.5x_2)$$

Neural networks

Or may be we want to constrain the outputs:

$$\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R}\ \{y = 0 \vee y = 1\})$$
$$\text{neuron}\ \ x_1\ x_2 = S\ (-0.9 + 0.5x_1 + 0.5x_2)$$

where

$$S\ x = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

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:

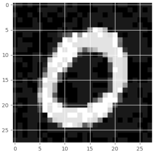
- approximate functions when exact solution is hard to get
- tolerant to noisy and incomplete data

BUT

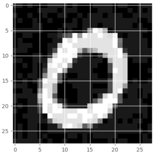
- solutions not easily conceptualised (lack of explainability)
- prone to a new range of safety and security problems:

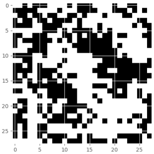
  adversarial attacks
  data poisoning
  catastrophic forgetting

Adversarial Attacks

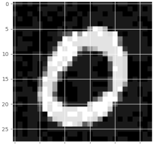


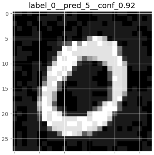
Adversarial Attacks



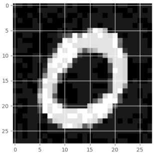


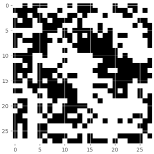
Adversarial Attacks

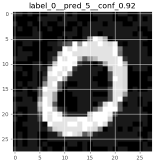




Adversarial Attacks

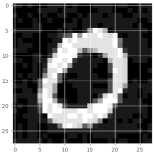


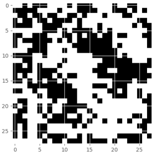


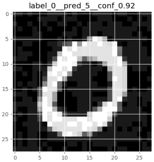


the perturbations are imperceptible to human eye

Adversarial Attacks



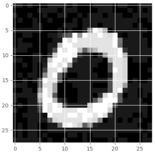


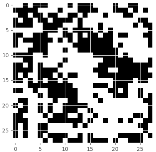


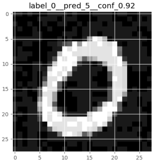
the perturbations are imperceptible to human eye

attacks transfer from one neural network to another

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

A few words on the context

1943 Perceptron by McCullogh and Pitts

A few words on the context

1943 Perceptron by McCullogh and Pitts

90s – Rise of machine learning applications

A few words on the context

1943 Perceptron by McCullogh and Pitts

90s – 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. (5000+ citations)

“ The existence of the adversarial negatives appears to be in contradiction with the network’s ability to achieve high generalization performance. Indeed, if the network can generalize well, how can it be confused by these adversarial negatives, which are indistinguishable from the regular examples? “

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. (5000+ citations)

2013-.. 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. (5000+ citations)

2013-.. 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

Table of Contents

Programs are functions...

$Program : \mathcal{A} \rightarrow \mathcal{B}$

and so are neural networks:

$$NeuralNet : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

Neural Network Verification

… could be like any other verification task

Neural Network Verification

... could be like any other verification task

if not for the following four problems:

I Semantics of function components is mostly opaque
II Number of verification parameters is huge
III Undecidable verification for non-linear functions
IV Finding verifiable neural networks may be difficult

Neural Network Verification

... could be like any other verification task

if not for the following four problems:

I Semantics of function components is mostly opaque

II Number of verification parameters is huge

III Undecidable verification for non-linear functions

IV Finding verifiable neural networks may be difficult

Neural Network Verification

... could be like any other verification task

if not for the following four problems:

I Semantics of function components is mostly opaque

II Number of verification parameters is huge

III Undecidable verification for non-linear functions

IV Finding verifiable neural networks may be difficult

Neural Network Verification

... could be like any other verification task

if not for the following four problems:

I Semantics of function components is mostly opaque
II Number of verification parameters is huge
III Undecidable verification for non-linear functions
IV Finding verifiable neural networks may be difficult

I. The problem of opaque semantics

$Program : \mathcal{A} \to \mathcal{B}$

$$NeuralNet : \mathbb{R}^n \to \mathbb{R}^m$$

But normally, programs

have semantically meaningful parts

which allows us to verify components that matter

I. The problem of opaque semantics

$Program : \mathcal{A} \to \mathcal{B}$

$$NeuralNet : \mathbb{R}^n \to \mathbb{R}^m$$

For neural nets:

~~have semantically meaningful parts~~

~~which allows us to verify components that matter~~

I. The problem of opaque semantics

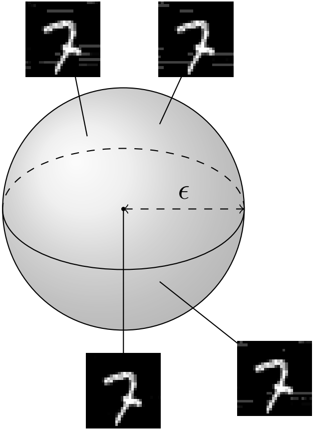
$Program : \mathcal{A} \to \mathcal{B}$

$$NeuralNet : \mathbb{R}^n \to \mathbb{R}^m$$

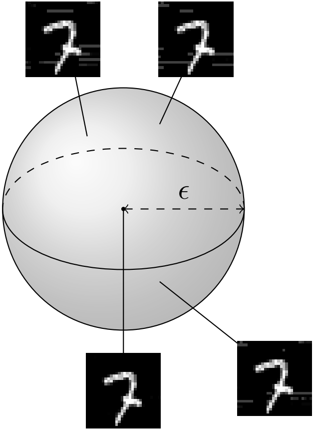
For neural nets:

input and output are the only semantically meaningful parts (and even that is somewhat blurry)

The "ϵ-ball verification"



The "ϵ-ball verification"



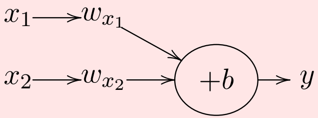
An ϵ-ball $\mathbb{B}(\hat{x}, \epsilon) = \{x \in \mathbb{R}^n : ||\hat{x} - x|| \leq \epsilon\}$

Classify all points in $\mathbb{B}(\hat{x}, \epsilon)$ in the "same class" as $\hat{x}$.

For example,

Take

$$\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R}\ \{y = 0 \vee y = 1\})$$
$$\text{neuron}\ \ x_1\ x_2 = S\ (-0.9 + 0.5x_1 + 0.5x_2)$$



For example,

Take

$$\begin{aligned}&\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R}\ \{y = 0 \vee y = 1\})\\&\text{neuron}\ \ x_1\ x_2 = S\left(-0.9 + 0.5x_1 + 0.5x_2\right)\end{aligned}$$

Define

$$\begin{aligned}\text{truthy}\ \ x &= |1 - x| \leq \epsilon\\ \text{falsey}\ \ x &= |0 - x| \leq \epsilon\end{aligned}$$

For example,

Take

$$\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R}\ \{y = 0 \vee y = 1\})$$
$$\text{neuron}\ \ x_1\ x_2 = S\ (-0.9 + 0.5x_1 + 0.5x_2)$$

Verify

$$\text{test}\ \ :\ \ (x_1 : \mathbb{R}\ \{\text{truthy}\ \ x_1\}) \to (x_2 : \mathbb{R}\ \{\text{truthy}\ \ x_2\}) \to (y : \mathbb{R}\ \{y = 1\})$$
$$\text{test}\ = \ \text{neuron}$$

For example,

Take

$$\text{neuron} : (x_1 : \mathbb{R}) \to (x_2 : \mathbb{R}) \to (y : \mathbb{R}\ \{y = 0 \lor y = 1\})$$
$$\text{neuron}\ \ x_1\ x_2 = S\ (-0.9 + 0.5x_1 + 0.5x_2)$$

Verify

$$\text{test}\ \ :\ \ (x_1 : \mathbb{R}\ \{\text{truthy}\ \ x_1\}) \to (x_2 : \mathbb{R}\ \{\text{truthy}\ \ x_2\}) \to (y : \mathbb{R}\ \{y = 1\})$$
$$\text{test}\ = \text{neuron}$$

Wen Kokke, E.K., Daniel Kienitz, Robert Atkey and David Aspinall. 2020. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS'20.

Refinement type library for Neural Net Verification

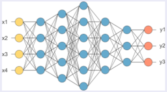
data

Refinement type library for Neural Net Verification

Python (Keras)

data

a neural network



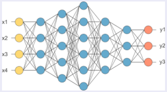
Refinement type library for Neural Net Verification

data

Python (Keras)

F^*

a neural network



NN as function; verification conditions as types

Refinement type library for Neural Net Verification

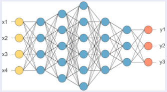
Python (Keras)

F^*

Z3

data

a neural network



NN as function; verification conditions as types

Except you only see:

F^*

data

NN as function; verification conditions as types

Refinement type library for Neural Net Verification

1 Let Python process the data and find a suitable network

Refinement type library for Neural Net Verification

1. Let Python process the data and find a suitable network
2. Export Python neural net to F* automatically:

```
val model : network (*with*) 2 (*inputs*) 1 (*output*) 1
(*layer*)
let model = NLast // ← makes single-layer network
  { weights    = [[0.5R]; [0.5R]]
  ; biases     = [−0.9R]
  ; activation = Threshold }
```

Refinement type library for Neural Net Verification

1. Let Python process the data and find a suitable network
2. Export Python neural net to F* automatically:

```
val model : network (*with*) 2 (*inputs*) 1 (*output*) 1
(*layer*)
let model = NLast // ← makes single-layer network
  { weights    = [[0.5R]; [0.5R]]
  ; biases     = [−0.9R]
  ; activation = Threshold }
```

NB

Uniform syntax for all networks we obtain from Python code!

Refinement type library for Neural Net Verification

1. Let Python process the data and find a suitable network
2. Export Python neural net to F* automatically:

```
val model : network (*with*) 2 (*inputs*) 1 (*output*)
1 (*layer*)
let model = NLast // ← makes single-layer network
  { weights    = [[0.5R]; [0.5R]]
  ; biases     = [−0.9R]
  ; activation = Threshold }
```

NB

Uniform syntax for all networks we obtain from Python code!

Refinement type library for Neural Net Verification

1. Let Python process the data and find a suitable network
2. Export Python neural net to F* automatically:
3. Define your verification conditions:

```
let eps      = 0.1R
let truthy x = 1.0R - eps ≤ x && x ≤ 1.0R + eps
let falsey x = 0.0R - eps ≤ x && x ≤ 0.0R + eps

val verify : (x1 : ℝ{truthy x1}) → (x2 : ℝ{truthy x2})
        → (y  : vector ℝ 1 {y ≡ [1.0R]})
let verify x1 x2 = run model [x1; x2]
```

Refinement type library for Neural Net Verification

1. Let Python process the data and find a suitable network
2. Export Python neural net to F* automatically:
3. Define your verification conditions:

```
let eps       = 0.1R
let truthy x = 1.0R - eps ≤ x && x ≤ 1.0R + eps
let falsey x = 0.0R - eps ≤ x && x ≤ 0.0R + eps

val verify : (x₁ : ℝ  {truthy x1} )
     → (x₂ : ℝ { truthy x2} )
        → (y  : vector ℝ 1 {y == [1.0R]} )
let verify x₁ x₂ = run model [x₁; x₂]
```

Refinement type library for Neural Net Verification

1. Let Python process the data and find a suitable network
2. Export Python neural net to F* automatically:
3. Define your verification conditions:

```
let eps      = 0.1R
let truthy x = 1.0R - eps ≤ x && x ≤ 1.0R + eps
let falsey x = 0.0R - eps ≤ x && x ≤ 0.0R + eps

val verify : (x₁ : ℝ  {truthy x1} )
     → (x₂ : ℝ { truthy x2} )
        → (y  : vector ℝ 1 {y == [1.0R]} )
let verify x₁ x₂ = run model [x₁; x₂]
```

Note: it is a universal property.

Refinement type library for Neural Net Verification

1. Let Python process the data and find a suitable network
2. Export Python neural net to F* automatically:
3. Define your verification conditions:

```
let eps      = 0.1R
let truthy x = 1.0R - eps ≤ x && x ≤ 1.0R + eps
let falsey x = 0.0R - eps ≤ x && x ≤ 0.0R + eps

val verify : (x1 : ℝ{truthy x1}) → (x2 : ℝ{truthy x2})
        → (y  : vector ℝ 1 {y ≡ [1.0R]})
let verify x1 x2 = run model [x1; x2]
```

4. Type check and relax!

Neural net robustness as refinement type

Wen Kokke, E.K., Daniel Kienitz, Robert Atkey and David Aspinall. 2020. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS'20.

- Builds on the real number library in F*;
- Concise Linear Algebra module;
- Straightforward definitions of neural nets as composed functions;
- with linear or "non-linear" activation functions
- A Python wrapper.

Neural net robustness as refinement type

Wen Kokke, E.K., Daniel Kienitz, Robert Atkey and David Aspinall. 2020. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS'20.

- Builds on the real number library in F*;
- Concise Linear Algebra module;
- Straightforward definitions of neural nets as composed functions;
- with linear or "non-linear" activation functions
- A Python wrapper.

Neural net robustness as refinement type

Wen Kokke, E.K., Daniel Kienitz, Robert Atkey and David Aspinall. 2020. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS'20.

- Builds on the real number library in F*;
- Concise Linear Algebra module;
- Straightforward definitions of neural nets as composed functions;
- with linear or "non-linear" activation functions
- A Python wrapper.

Neural net robustness as refinement type

Wen Kokke, E.K., Daniel Kienitz, Robert Atkey and David Aspinall. 2020. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS'20.

- Builds on the real number library in F*;
- Concise Linear Algebra module;
- Straightforward definitions of neural nets as composed functions;
- with linear or "non-linear" activation functions
- A Python wrapper.

Neural net robustness as refinement type

Wen Kokke, E.K., Daniel Kienitz, Robert Atkey and David Aspinall. 2020. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS'20.

- Builds on the real number library in F*;
- Concise Linear Algebra module;
- Straightforward definitions of neural nets as composed functions;
- with linear or "non-linear" activation functions
- A Python wrapper.

Neural Network Verification

Recall the 4 problems:

I Semantics of function components is opaque

II Number of verification parameters is huge

III Undecidable verification for non-linear functions

IV Finding verifiable neural networks is difficult

Neural Network Verification

Recall the 4 problems:

I. Semantics of function components is opaque

Use refinement types (for [functional] elegance),

... or SMT solvers directly.

Wen Kokke. 2020. Sapphire: a Neural Net Verification Library for Z3 in Python. https://github.com/wenkokke/sapphire

II. Number of verification parameters is huge

III. Undecidable verification for non-linear functions

IV. Finding verifiable neural networks

We get

Semantic Opacity

We get

Semantic Opacity $\rightarrow$ ϵ-ball verification

We get

Semantic Opacity → ϵ-ball verification → overwhelming No of parameters

Neural Network Verification

Recall

I Semantics of function components is opaque

Use refinement types (for [functional] elegance)

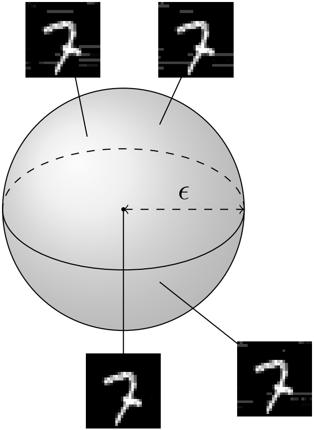
... or SMT solvers directly.

II Number of verification parameters is huge

III Undecidable verification for non-linear functions

IV Finding verifiable neural networks may be difficult

The "ϵ-ball verification"



Scale in Neural network verification

MNIST data set

2828 images of the handwritten digits “0” to “9”
784 pixels each

Scale in Neural network verification

MNIST data set

2828 images of the handwritten digits “0” to “9”
784 pixels each

The smallest network

input layer of 784 weights

Scale in Neural network verification

MNIST data set

2828 images of the handwritten digits "0" to "9"
784 pixels each

The smallest network

input layer of 784 weights

hidden layer of (say) 128 ReLU nodes

Scale in Neural network verification

MNIST data set

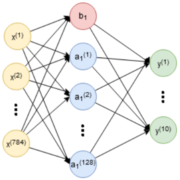
2828 images of the handwritten digits “0” to “9”
784 pixels each

The smallest network

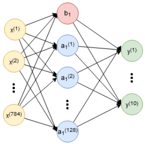
input layer of 784 weights

hidden layer of (say) 128 ReLU nodes

output layer of 10 softmax neurons



Scaling Neural network verification

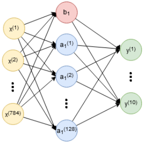


We want to say:

```
val sample_in : vector ℝ 784
let sample_in = let v = [7.394R; −0.451R; …; 0.199R]

val sample_out: vector ℝ 10
let sample_out = let v = [0.998R; 0.000R; …; 0.000R]
```

Scaling Neural network verification



We want to say:

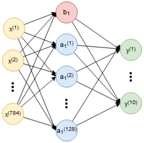
```
val sample_in : vector ℝ 784
let sample_in = let v = [7.394R; −0.451R; ...; 0.199R]

val sample_out: vector ℝ 10
let sample_out = let v = [0.998R; 0.000R; ...; 0.000R]
```

And prove the ϵ-ball property:

```
let _ =  ∀ (x:vector ℝ 784). (|sample_in - x| < 0.01R)
⟹ (|sample_out - (run network x)| < 0.1R))
```

Scaling Neural network verification



We want to say:

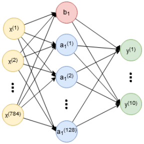
```
val sample_in : vector ℝ 784
let sample_in = let v = [7.394R; −0.451R; …; 0.199R]

val sample_out: vector ℝ 10
let sample_out = let v = [0.998R; 0.000R; …; 0.000R]
```

And prove the ϵ-ball property:

```
let _ = ∀ (x:vector ℝ 784). (|sample_in - x| < 0.01R)
⟹ (|sample_out - (run network x)| < 0.1R))
```

Scaling Neural network verification



We want to say:

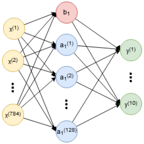
```
val sample_in : vector ℝ 784
let sample_in = let v = [7.394R; −0.451R; …; 0.199R]

val sample_out: vector ℝ 10
let sample_out = let v = [0.998R; 0.000R; …; 0.000R]
```

And prove the ϵ-ball property:

```
let _ =  ∀ (x:vector ℝ 784).  (|sample_in - x| < 0.01R
 ⟹ (|sample_out - (run network x)| < 0.1R))
```

Scaling Neural network verification



We want to say:

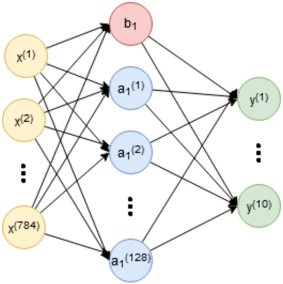
```
val sample_in : vector ℝ 784
let sample_in = let v = [7.394R; −0.451R; …; 0.199R]

val sample_out: vector ℝ 10
let sample_out = let v = [0.998R; 0.000R; …; 0.000R]
```

And prove the ϵ-ball property:

```
let _ =  ∀ (x:vector ℝ 784). (|sample_in - x| < 0.01R)
⟹ (|sample_out -  (run network x)  | < 1.0R))
```

Scaling Neural network verification



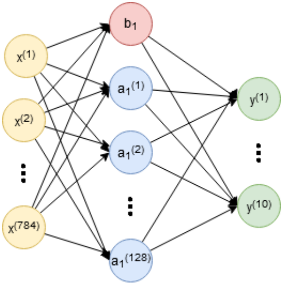
The smallest reasonable neural network

input layer of 784 weights

hidden layer of (say) 128 ReLU nodes

output layer of 10 softmax neurons

Scaling Neural network verification



The smallest reasonable neural network

input layer of 784 weights

hidden layer of (say) 128 ReLU nodes

output layer of 10 softmax neurons

total $784 \times 128 + 128 + 128 \times 10 + 10 = 101770$ parameters

Neural Network Verification

Recall

- I Semantics of function components is opaque
- II Number of verification parameters is huge
- III Undecidable verification for non-linear functions
- IV Finding verifiable neural networks may be difficult

Neural Network Verification

Recall

I Semantics of function components is mostly opaque

II Number of verification parameters is huge

Reduce the number of parameters (to scale)

either reduce network size and re-train
or reduce the network to a provably equivalent
or use over-approximation (in the style of abstract interpretation)

III Undecidable verification for non-linear functions

IV Finding verifiable neural networks may be difficult

Neural Network Verification

Recall

I Semantics of function components is mostly opaque

II Number of verification parameters is huge

Reduce the number of parameters (to scale)

either reduce network size and re-train

W.Kokke, E.K., D.Kienitz, R.Atkey and D.Aspinall. 2020. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS'20.

or reduce the network to a provably equivalent

or use over-approximation (in the style of abstract interpretation)

III Undecidable verification for non-linear functions

IV Finding verifiable neural networks may be difficult

Neural Network Verification

Recall

I Semantics of function components is mostly opaque

II Number of verification parameters is huge

Reduce the number of parameters (to scale)

either reduce network size and re-train
or reduce the network to a provably equivalent

S.Gokulanathan, A.Feldsher, A.Malca, C.W. Barrett, G. Katz: Simplifying Neural Networks Using Formal Verification. NFM 2020: 85-93

or use over-approximation (in the style of abstract interpretation)

III Undecidable verification for non-linear functions

IV Finding verifiable neural networks may be difficult

Neural Network Verification

Recall

I Semantics of function components is mostly opaque

II Number of verification parameters is huge

Reduce the number of parameters (to scale)

either reduce network size and re-train
or reduce the network to a provably equivalent
or use over-approximation (in the style of abstract interpretation)

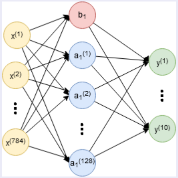
G.Singh, T.Gehr, M.Püschel, M.Vechev: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. 3(POPL): 41:1-41:30 (2019)

III Undecidable verification for non-linear functions

IV Finding verifiable neural networks may be difficult

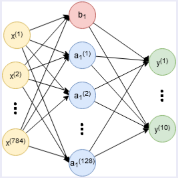
So far...

Semantic Opacity



So far...

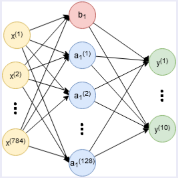
Semantic Opacity



ϵ-ball verification

So far...

Semantic Opacity



ϵ-ball verification

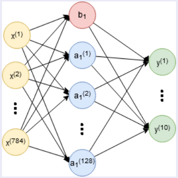
overwhelming No of parameters

So far...

overwhelming No of parameters

ϵ-ball verification

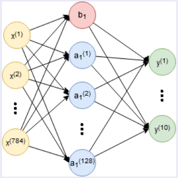
Semantic Opacity



can verify a different object!

So far...

Semantic Opacity

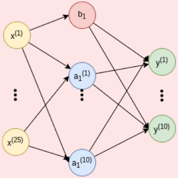


ϵ-ball verification

overwhelming No of parameters

can verify a different object!

verify a smaller network



Neural Network Verification

Recall

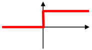
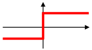
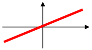
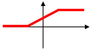
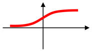
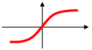
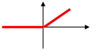
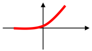
I Semantics of function components is mostly opaque

II Number of verification parameters is huge

III Undecidable verification for non-linear functions

IV Finding verifiable neural networks may be difficult

Activation functions

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

On the solvers side...

The SMT solver Z3:

- uses Dual Simplex to solve linear real arithmetic;
- and a fragment of non-linear real arithmetic – multiplications
- uses conflict resolution procedure
- based on cylindrical algebraic decomposition

On the solvers side...

The SMT solver Z3:

- uses Dual Simplex to solve linear real arithmetic;
- and a fragment of non-linear real arithmetic – multiplications
- uses conflict resolution procedure
- based on cylindrical algebraic decomposition

Jovanović, D., de Moura, L.: Solving non-linear arithmetic. ACM Communications in Computer Algebra 46(3/4), 104 (Jan 2013).

On the solvers side...

The SMT solver Z3:

- uses Dual Simplex to solve linear real arithmetic;
- and a fragment of non-linear real arithmetic – multiplications
- uses conflict resolution procedure
- based on cylindrical algebraic decomposition

We need:

- exponents
- logarithms
- trigonometric functions

Jovanović, D., de Moura, L.: Solving non-linear arithmetic. ACM Communications in Computer Algebra 46(3/4), 104 (Jan 2013).

On the solvers side...

The solver MetiTarski:

Supports:

- exponents
- logarithms
- trigonometric functions

for 4-5 variables

Akbarpour, B., Paulson, L.C.: MetiTarski: An automatic theorem prover for real- valued special functions. Journal of Automated Reasoning 44(3), 175–205 (Aug 2009).

On the solvers side...

The solver MetiTarski:

Supports:

- exponents
- logarithms
- trigonometric functions

for 4-5 variables

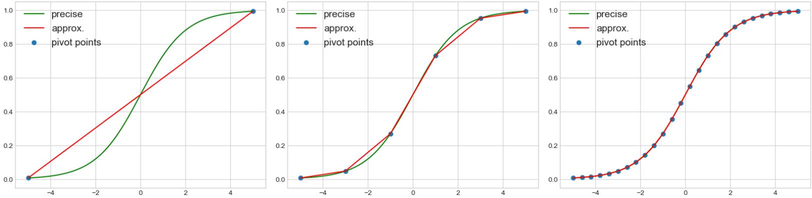
We need:

hundreds of variables

Akbarpour, B., Paulson, L.C.: MetiTarski: An automatic theorem prover for real- valued special functions. Journal of Automated Reasoning 44(3), 175–205 (Aug 2009).

Solutions?!

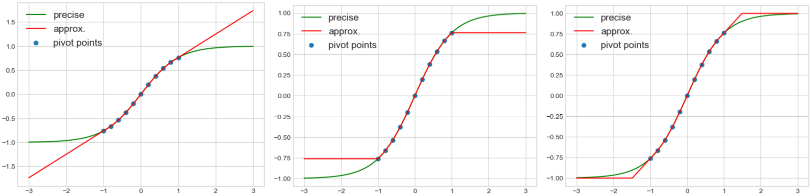
Linearise effectively!



Wen Kokke, E.K., Daniel Kienitz, Robert Atkey and David Aspinall. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS'20.

Solutions?!

Linearise effectively!



Wen Kokke, E.K., Daniel Kienitz, Robert Atkey and David Aspinall. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS'20.

Neural Network Verification

Recall

- Semantics of function components is mostly opaque

Use refinement types for functional elegance

Neural Network Verification

Recall

I Semantics of function components is mostly opaque

Use refinement types for functional elegance

II Number of verification parameters is huge

Reduce the number of parameters for scale

Neural Network Verification

Recall

I Semantics of function components is mostly opaque

Use refinement types for functional elegance

II Number of verification parameters is huge

Reduce the number of parameters for scale

III Undecidable verification for non-linear functions

Linearise effectively for automation

Neural Network Verification

Recall

I Semantics of function components is mostly opaque

Use refinement types for functional elegance

II Number of verification parameters is huge

Reduce the number of parameters for scale

III Undecidable verification for non-linear functions

Linearise effectively for automation

IV Finding verifiable neural networks may be difficult

"Opaque Verification"

overwhelming No of parameters

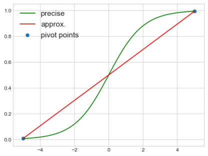
can verify a different object!

verify a smaller network

“Opaque Verification”

overwhelming No of parameters

Non-linearity



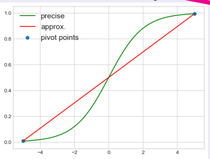
can verify a different object!

verify a smaller network

“Opaque Verification”

overwhelming No of parameters

Non-linearity



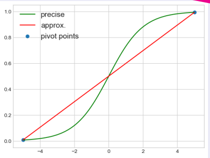
can verify a different object!

verify a smaller network

"Opaque Verification"

overwhelming No of parameters

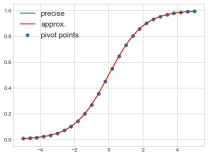
Non-linearity



can verify a different object!

verify a smaller network

verify a linearised network



Neural Network Verification

Recall

I Semantics of function components is mostly opaque

Use refinement types for functional elegance

II Number of verification parameters is huge

Reduce the number of parameters for scale

III Undecidable verification for non-linear functions

Linearise effectively for automation

IV Finding verifiable neural networks may be difficult

Neural Network Verification

Recall

I Semantics of function components is mostly opaque

Use refinement types for functional elegance

II Number of verification parameters is huge

Reduce the number of parameters for scale

III Undecidable verification for non-linear functions

Linearise effectively for automation

IV Finding verifiable neural networks may be difficult

(Re-)Train your network correct

Constraint-driven training

Train your network correct!

- augment loss functions with logical constraints

M.Fischer, M.Balunovic, D.Drachsler-Cohen, T.Gehr, C.Zhang, and M.Vechev. 2019. DL2: Training and Querying Neural Networks with Logic. ICML 2019, Vol. 97. PMLR, 1931–1941.

Constraint-driven training

Train your network correct!

- augment loss functions with logical constraints
  - M.Fischer, M.Balunovic, D.Drachsler-Cohen, T.Gehr, C.Zhang, and M.Vechev. 2019. DL2: Training and Querying Neural Networks with Logic. ICML 2019, Vol. 97. PMLR, 1931–1941.
- augment loss functions with abstract interpretation constraints
  - E.Ayers, F.Eiras, M.Hawasly, I.Whiteside: PaRoT: A Practical Framework for Robust Deep Neural Network Training. NFM 2020: 63-84
  - M.Balunovic and M.Vechev. 2020. Adversarial Training and Provable Defenses: Bridging the Gap. ICLR 2020.

Constraint-driven training

Train your network correct!

- augment loss functions with logical constraints
  - M.Fischer, M.Balunovic, D.Drachsler-Cohen, T.Gehr, C.Zhang, and M.Vechev. 2019. DL2: Training and Querying Neural Networks with Logic. ICML 2019, Vol. 97. PMLR, 1931–1941.
- augment loss functions with abstract interpretation constraints
  - E.Ayers, F.Eiras, M.Hawasly, I.Whiteside: PaRoT: A Practical Framework for Robust Deep Neural Network Training. NFM 2020: 63-84
  - M.Balunovic and M.Vechev. 2020. Adversarial Training and Provable Defenses: Bridging the Gap. ICLR 2020.

A word of caution

- Marco Casadio. Generative versus logical training against adversarial attacks. MSc Thesis at HWU. 2020.

“Opaque Verification”

overwhelming No of parameters

Non-linearity

can verify a different object!

verify a smaller network

verify a linearised network

“Opaque Verification”

overwhelming No of parameters

Non-linearity

Non-verifiable network

can verify a different object!

verify a smaller network

verify a linearised network

“Opaque Verification”

overwhelming No of parameters

Non-linearity

Non-verifiable network

can verify a different object!

verify a smaller network

verify a linearised network

“Opaque Verification”

overwhelming No of parameters

Non-linearity

Non-verifiable network

can verify a different object!

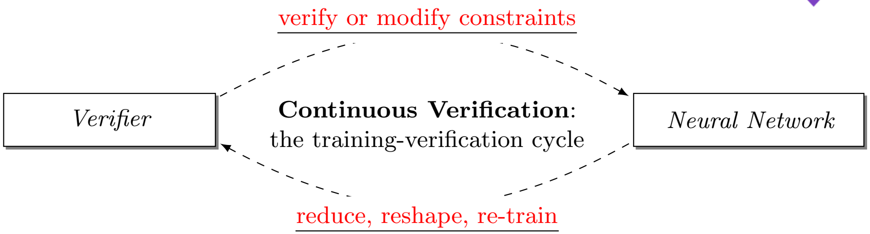
verify a smaller network

verify a linearised network

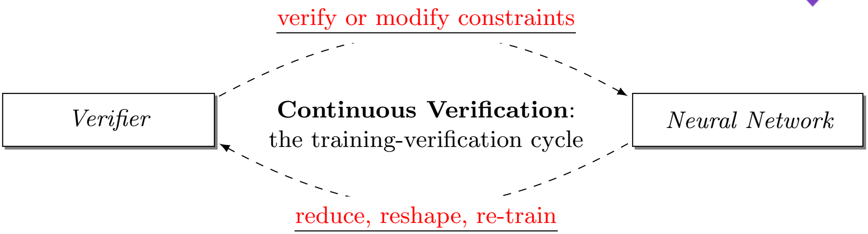
re-train your network correct

Table of Contents

Continuous Verification



Continuous Verification

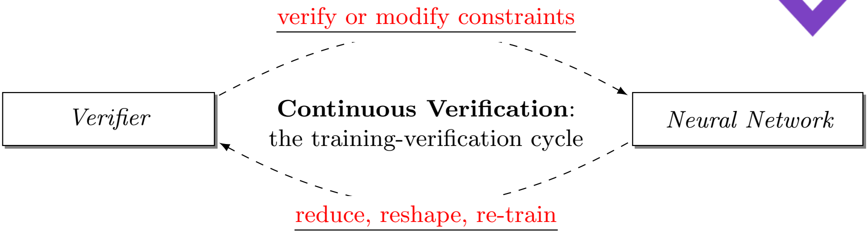


We have seen "continuous verification"

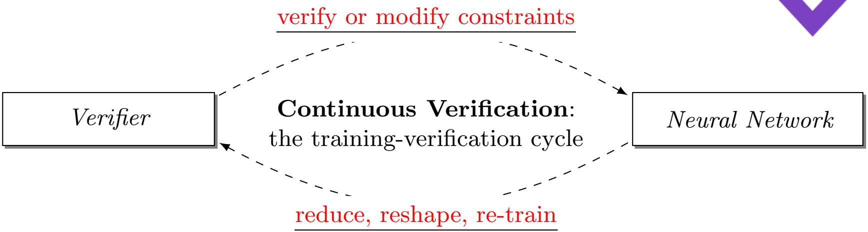
as a trend that arises everywhere in neural network verification

for a variety of different reasons!

Continuous Verification



Continuous Verification



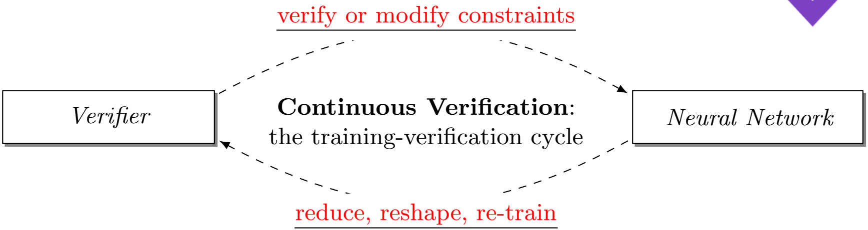
Role of declarative programming:

ϵ-ball verification is an instance of refinement type checking

$\texttt{verify}\ x :\ \ x : \mathbb{R}^n\{|\texttt{sample\_in} - x| < \epsilon\} \Longrightarrow y : \mathbb{R}^m\{|\texttt{sample\_out} - y| < \epsilon'\}$
$\texttt{verify}\ x = \texttt{run network}\ x$

Wen Kokke, E.K., Daniel Kienitz, Robert Atkey and David Aspinall. 2020. Neural Networks, Secure by Construction: An Exploration of Refinement Types. APLAS'20.

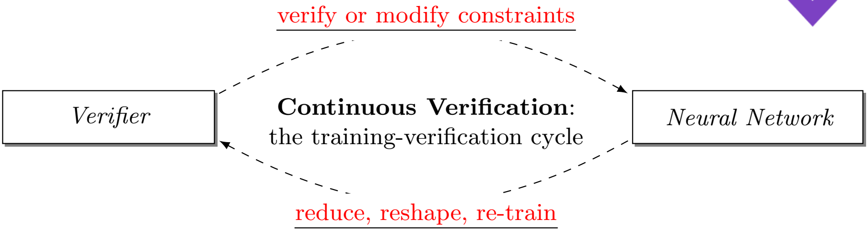
Continuous Verification



Role of declarative programming:

Solvers are increasingly important for automation

Continuous Verification



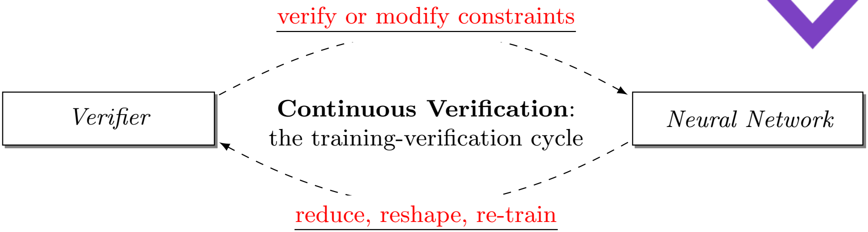
Role of declarative programming:

Solvers are increasingly important for automation

Rise of domain-specific solvers for neural networks:

Katz, G., et al.: The Marabou framework for verification and analysis of deep neural networks. In: CAV 2019, Part I. LNCS, vol. 11561, pp. 443–452. Springer (2019)

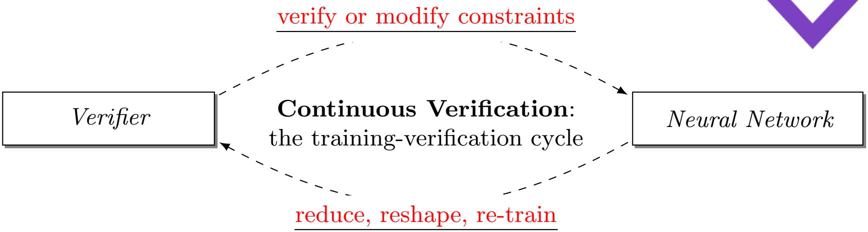
Continuous Verification



Role of declarative programming in continuous verification?

- provide a sound and elegant PL infrastructure
- that bootstraps solvers and machine learning algorithms
- to ensure transparency, modularity,
- invariant and safety checks

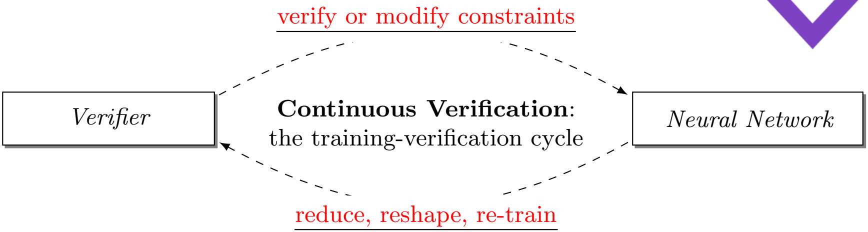
Continuous Verification



Role of declarative programming in continuous verification?

- provide a sound and elegant PL infrastructure
- that bootstraps solvers and machine learning algorithms
- to ensure transparency, modularity,
- invariant and safety checks

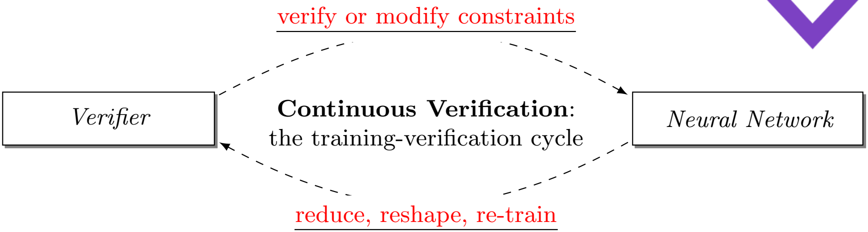
Continuous Verification



Role of declarative programming in continuous verification?

- provide a sound and elegant PL infrastructure
- that bootstraps solvers and machine learning algorithms
- to ensure transparency, modularity,
- invariant and safety checks

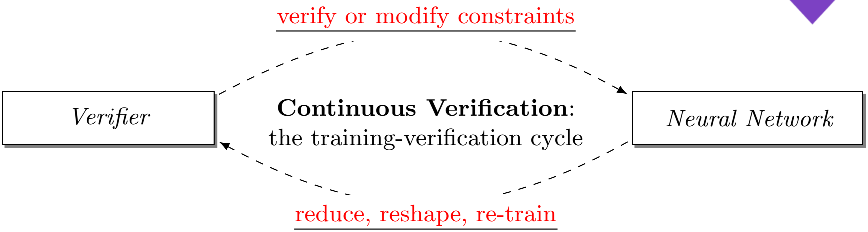
Continuous Verification



Role of declarative programming in continuous verification?

- provide a sound and elegant PL infrastructure
- that bootstraps solvers and machine learning algorithms
- to ensure transparency, modularity,
- invariant and safety checks

Continuous Verification

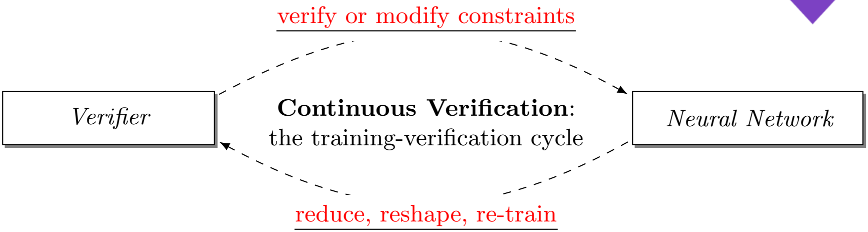


Role of declarative programming in continuous verification?

- Verification as refinement type checking
- Training as program synthesis

$\texttt{verify}\ x :\ \ x : \mathbb{R}^n\{|\texttt{sample\_in} - x| < \epsilon\} \Longrightarrow y : \mathbb{R}^m\{|\texttt{sample\_out} - y| < \epsilon'\}$
$\texttt{verify}\ x = \texttt{run network}\ x$

Continuous Verification



Role of declarative programming in continuous verification?

- Verification as refinement type checking
- Training as program synthesis

$$\texttt{verify}\ x :\ \ x : \mathbb{R}^n\{|\texttt{sample\_in} - x| < \epsilon\} \Longrightarrow y : \mathbb{R}^m\{|\texttt{sample\_out} - y| < \epsilon'\}$$
$$\texttt{verify}\ x = \texttt{run} \qquad\qquad ?NETWORK? \quad x$$

Thanks for your attention!

.

.

Thanks for your attention!

Open PhD position at LAIV

on verification of recurrent neural networks