Reliable and Interpretable Artificial Intelligence

Martin Vechev
Department of Computer Science, ETH Zurich
How good (robust) is your neural net?

Neural networks are *not* robust to input perturbations (e.g., image rotation / change of lighting)

Misclassifications in neural networks deployed in self-driving cars [1]
In each picture one of the 3 networks makes a mistake...

Wanted: Automated and scalable analysis to verify realistic NNs

Useful in:

• Ensuring correctness of a larger cyber-physical system that uses the NN
• Proving robustness of the NN (beyond just finding adversarial examples)
• Learning interpretable specs of the NN
• Comparing NNs
Problem Statement and Challenges

Neural Network Analysis Problem

Given
- a neural network $N$
- a property over inputs $\varphi$
- a property over outputs $\psi$

check whether $\forall i \in I. i \models \varphi \implies N(i) \models \psi$ holds

Challenges:
- The property $\varphi$ over inputs usually captures an unbounded set of inputs
- Existing symbolic solutions do not scale to large networks (e.g. conv nets)
Key Technical Insight: AI for AI

Deep Neural Nets:
Affine transforms + Restricted non-linearity

Abstract Interpretation:
Scalable and Precise Numerical Domains
AI$^2$: Abstract Interpretation for NNs

- Concrete
- Inputs
  - Neuron values: (0.01, 0.03,..), (0.94, 0.02,..), (0.23, 0.12,..), (0.12, 0.54,..)
- Neuron values: (0.01, 0.03,..), (0.94, 0.02,..), (0.23, 0.12,..), (0.12, 0.54,..)
- Outputs: (0.01, 0.03,..), (0.94, 0.02,..), (0.23, 0.12,..), (0.12, 0.54,..)
AI²: Abstract Interpretation for NNs

Concrete layer transformer
AI²: Abstract Interpretation for NNs

Concrete

Abstract

Concrete layer transformer

Inputs → Neuron values → Neuron values → Outputs

Concrete layer transformer

Outputs
AI²: Abstract Interpretation for NNs
Why Abstractions? (high-level view)

Minor issue 😊: general problem is undecidable
Hence: approximation
What is Abstract Interpretation?

We will see a glimpse of A.I., and only the parts we need for our problem. However, this should be enough to get a working intuition of it, to know the libraries and to know how to apply it.
Abstract Interpretation: a primer

The theory of abstract interpretation is a theory of approximation
Probably one of the most elegant theories in computer science

- an elegant theoretical framework
- systematic way to build automated analyzers
- a way to think about approximation
- theory invented in late 70s
- started gaining popularity in the 90s
- all commercial tools use some form of A.I.

The principles of approximation are fundamental to reasoning about computation with infinite state spaces.
Abstract Interpretation: a primer

A.I. concerns itself with questions such as:

• What is it that we are approximating?

• What does it mean for the approximation to be optimal (or to approximate)?

• What does it mean for an approximation to be correct?

• How do we actually build a correct and optimal system?

• Can the process of building an analyzer be automated?
Abstract Interpretation Recipe
for building an A.I. engine

1. come up with an abstract domain
   • select based on the type of properties you want to prove

2. define abstract semantics for the programming language w.r.t. to the abstract domain from step 1.
   • we need to define the abstract transformers, that is, the effect of statement / expression on the abstract domain
   • we need to prove that the abstract semantics are sound w.r.t concrete semantics of the programming language

3. iterate abstract transformers over the abstract domain
   • until we reach a fixed point

The fixed point is the over-approximation of the program
A.I. cheat sheet

• \((C, \subseteq_c)\) is the **concrete lattice**. An element \(x\) in \(C\) is a set of concrete program states.

• \((A, \subseteq_A)\) is the **abstract lattice**. An element \(z\) in \(A\) is an abstract element that represents a set of concrete states.

• \(F\) is your program. \(F(x)\) applies it on set of states \(x\). \(F\) is **monotone**. Least fixed point of \(F\) (LFP \(F\)) is an element in \(C\) that captures all reachable states of \(F\) – the set may be infinite or unbounded so we typically cannot compute it.

• \(F^\#\) is the abstract transformer. \(F^\#(z)\) applies \(F^\#\) to abstract element \(z\). \(F^\#\) should be **monotone** (see Tarski’s theorem)

• \(\gamma\) is the **concretization**: it defines to which concrete states an abstract element maps to. \(\gamma\) is monotone. It is key to defining what it means for \(F^\#\) to approximate \(F\).

• We iterate \(F^\#\) to a fixed point. If \(F^\#\) approximates \(F\), then its least fixed point (LFP \(F^\#\)) approximates LFP \(F\)! We can compute LFP \(F^\#\)!

We just need a little bit of theory to know when we reach a fixed point and what approximation means. Simply a fixed point is not enough! We need to know \(F^\#\) actually approximates \(F\).
Tarski’s fixed point theorem (part of it)

if \((L, \sqsubseteq, \sqcup, \sqcap, \perp, T)\) is a complete lattice and 
f: \(L \rightarrow L\) is a monotone function,

then: \(\text{lfp} \sqsubseteq f\) exists

Note: the complete lattice can be of infinite height
Practically: a useful fixed point theorem

Given a poset of finite height, a least element $\bot$, a monotone $f$.

Then the iterates $f^0(\bot), f^1(\bot), f^2(\bot)$... form an increasing sequence which eventually stabilizes from some $n \in \mathbb{N}$, that is: $f^n(\bot) = f^{n+1}(\bot)$ and:

$$\text{lfp} \sqsubseteq f = f^n(\bot)$$

This leads to a simple iterative algorithm for computing $\text{lfp} \sqsubseteq f$
Approximating a Function

So we have 2 functions:

\[ F: C \rightarrow C \]
\[ F^\#: A \rightarrow A \]

Here is a definition of function approximation:

\[ \forall z \in A : F(\gamma(z)) \sqsubseteq_c \gamma(F^\#(z)) \]
Visualizing the Definition

\[ F(x) \subseteq C \]

\[ F(x) \rightarrow x \]

\[ F \]

\[ (C, \subseteq_C) \]

\[ \gamma \]

\[ (A, \subseteq_A) \]

\[ F^#(z) \]

\[ F^# \]

\[ z \]
Key Theorem of A.I.: Least Fixed Point Approximation

If we have:

1. monotone functions $F: C \rightarrow C$ and $F^#: A \rightarrow A$
2. $\gamma: A \rightarrow C$ is monotone
3. $\forall z \in A: F(\gamma(z)) \sqsubseteq_c \gamma(F^#(z))$ (that is, $F^#$ approximates $F$)

then:

$$\text{Ifp}(F) \sqsubseteq_c \gamma(\text{Ifp}(F^#))$$

This is important as it goes from local function approximation to global approximation.
Now, let us see how to instantiate the theory.
Abstract Interpretation Recipe
for building an A.I. engine

1. come up with an abstract domain
   • select based on the type of properties you want to prove

2. define abstract semantics for the programming language w.r.t. to the abstract domain from step 1.
   • we need to define the abstract transformers, that is, the effect of statement / expression on the abstract domain
   • we need to prove that the abstract semantics are sound w.r.t concrete semantics of the programming language

3. iterate abstract transformers over the abstract domain
   • until we reach a fixed point

The fixed point is the over-approximation of the program
Each variable in the program takes a value from the following domain (a complete lattice):
Abstract Interpretation Recipe
for building an A.I. engine

1. come up with an abstract domain
   • select based on the type of properties you want to prove

2. define abstract semantics for the programming language w.r.t. to the abstract domain from step 1.
   • we need to define the abstract transformers, that is, the effect of statement / expression on the abstract domain
   • we need to prove that the abstract semantics are sound w.r.t concrete semantics of the programming language

3. iterate abstract transformers over the abstract domain
   • until we reach a fixed point

The fixed point is the over-approximation of the program
If we add $\bot_i$ to any other element, we get $\bot_i$.

If both operands are not $\bot_i$, we get:

$$[x, y] + [z, q] = [x + z, y + q]$$

what about $\ast$?

is $[x, y] \ast [z, q] = [x \ast z, y \ast q]$ sound?
\leq (av_1, av_2)

what should \leq([0,4], [3,7]) produce?

one answer is: ([0,3], [3,7]). Is it sound?

another non-comparable answer is: ([0,4], [4,7]). Is it sound?

Can you find a more precise answer?
\[ \preceq (av_1, av_2) \]

\[ \preceq ([l_1,u_1], [l_2,u_2]) = ([l_1,u_1] \cap_i [-\infty,u_2], [l_1, \infty] \cap_i [l_2,u_2]) \]

\[ \preceq ( [0,4], [3,7]) = ( [0,4] \cap_i [-\infty,7], [0, \infty] \cap_i [3, 7]) = ( [0,4], [3,7]) \]
Abstract Interpretation Recipe
for building a static analyzer

1. come up with an abstract domain
   • select based on the type of properties you want to prove

2. define abstract semantics for the programming language w.r.t. to the abstract domain from step 1.
   • we need to define the abstract transformers, that is, the effect of statement / expression on the abstract domain
   • we need to prove that the abstract semantics are sound w.r.t concrete semantics of the programming language

3. iterate abstract transformers over the abstract domain
   • until we reach a fixed point

The fixed point is the over-approximation of the program
We typically start the iteration from \(\bot\) meaning nothing is reachable.
Iterate 0

foo (int i) {
  1: int x := 5;
  2: int y := 7;
  3: if (i ≥ 0) {
     4: y := y + 1;
     5: i := i - 1;
     6: goto 3;
  }
  7: }

1: x → ⊥i, y → ⊥i, i → ⊥i
2: x → ⊥i, y → ⊥i, i → ⊥i
3: x → ⊥i, y → ⊥i, i → ⊥i
4: x → ⊥i, y → ⊥i, i → ⊥i
5: x → ⊥i, y → ⊥i, i → ⊥i
6: x → ⊥i, y → ⊥i, i → ⊥i
7: x → ⊥i, y → ⊥i, i → ⊥i
Iterate 1

Initially, the values are T as the variables can be anything

```c
foo (int i) {
    int x := 5;
    int y := 7;
    if (i ≥ 0) {
        y := y + 1;
        i := i - 1;
    }
}
```

1: \( x \rightarrow [-\infty,\infty], \ y \rightarrow [-\infty,\infty], \ i \rightarrow [-\infty,\infty] \)
2: \( x \rightarrow \bot_i, \ y \rightarrow \bot_i, \ i \rightarrow \bot_i \)
3: \( x \rightarrow \bot_i, \ y \rightarrow \bot_i, \ i \rightarrow \bot_i \)
4: \( x \rightarrow \bot_i, \ y \rightarrow \bot_i, \ i \rightarrow \bot_i \)
5: \( x \rightarrow \bot_i, \ y \rightarrow \bot_i, \ i \rightarrow \bot_i \)
6: \( x \rightarrow \bot_i, \ y \rightarrow \bot_i, \ i \rightarrow \bot_i \)
7: \( x \rightarrow \bot_i, \ y \rightarrow \bot_i, \ i \rightarrow \bot_i \)
foo (int i) {
1: int x := 5;
2: int y := 7;
3: if (i ≥ 0) {
4: y := y + 1;
5: i := i - 1;
6: goto 3;
7: }
}
foo (int i) {
  1: int x := 5;
  2: int y := 7;
  3: if (i ≥ 0) {
      4: y := y + 1;
      5: i := i - 1;
      6: goto 3;
  }
  7: }

1: x → [-∞,∞], y → [-∞,∞], i → [-∞,∞]
2: x → [5,5], y → [-∞,∞], i → [-∞,∞]
3: x → [5,5], y → [7,7], i → [-∞,∞]
4: x → ⊥i, y → ⊥i, i → ⊥i
5: x → ⊥i, y → ⊥i, i → ⊥i
6: x → ⊥i, y → ⊥i, i → ⊥i
7: x → ⊥i, y → ⊥i, i → ⊥i
foo (int i) {
    1: int x := 5;
    2: int y := 7;
    3: if (i ≥ 0) {
       4: y := y + 1;
       5: i := i - 1;
       6: goto 3;
    }
    7: }

1: x → [-∞,∞], y → [-∞,∞], i → [-∞,∞]
2: x → [5,5], y → [-∞,∞], i → [-∞,∞]
3: x → [5,5], y → [7,7], i → [-∞,∞]
4: x → [5,5], y → [7,7], i → [0,∞]
5: x → ⊥_i, y → ⊥_i, i → ⊥_i
6: x → ⊥_i, y → ⊥_i, i → ⊥_i
7: x → [5,5], y → [7,7], i → [-∞, -1]
Iterate 5

```c
foo (int i) {
    int x := 5;
    int y := 7;
    if (i ≥ 0) {
        y := y + 1;
        i := i - 1;
        goto 3;
    }
    7:
}
```

<table>
<thead>
<tr>
<th>State</th>
<th>x</th>
<th>y</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>x → [−∞, ∞], y → [−∞, ∞], i → [−∞, ∞]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:</td>
<td>x → [5, 5], y → [−∞, ∞], i → [−∞, ∞]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3:</td>
<td>x → [5, 5], y → [7, 7], i → [−∞, ∞]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4:</td>
<td>x → [5, 5], y → [7, 7], i → [0, ∞]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>x → [5, 5], y → [8, 8], i → [0, ∞]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6:</td>
<td>x → ⊥, y → ⊥, i → ⊥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>x → [5, 5], y → [7, 7], i → [−∞, −1]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
foo (int i) {
1: int x := 5;
2: int y := 7;
3: if (i ≥ 0) {
   4: y := y + 1;
   5: i := i - 1;
   6: goto 3;
}
7: }

1: x → [-∞,∞], y → [-∞,∞], i → [-∞,∞]
2: x → [5,5], y → [-∞,∞], i → [-∞,∞]
3: x → [5,5], y → [7,7], i → [-∞,∞]
4: x → [5,5], y → [7,7], i → [0,∞]
5: x → [5,5], y → [8,8], i → [0,∞]
6: x → [5,5], y → [8,8], i → [-1,∞]
7: x → [5,5], y → [7,7], i → [-∞, -1]
Iterate 7: we jumped to $\infty$

Step 1: we did a join: $[7,7] \sqcup [8,8] = [7,8]$  
Step 2: we saw that end point of interval increased, so we bumped it to $\infty$

foo (int i) {
    1: int x := 5;
    2: int y := 7;
    3: if (i $\geq$ 0) {
        4: y := y + 1;
        5: i := i - 1;
        6: goto 3;
    }
    7: }

1: $x \rightarrow [-\infty, \infty], \; y \rightarrow [-\infty, \infty], \; i \rightarrow [-\infty, \infty]$
2: $x \rightarrow [5,5], \; y \rightarrow [-\infty, \infty], \; i \rightarrow [-\infty, \infty]$
3: $x \rightarrow [5,5], \; y \rightarrow [7, \infty], \; i \rightarrow [-\infty, \infty]$
4: $x \rightarrow [5,5], \; y \rightarrow [7,7], \; i \rightarrow [0, \infty]$
5: $x \rightarrow [5,5], \; y \rightarrow [8,8], \; i \rightarrow [0, \infty]$
6: $x \rightarrow [5,5], \; y \rightarrow [8,8], \; i \rightarrow [-1, \infty]$
7: $x \rightarrow [5,5], \; y \rightarrow [7,7], \; i \rightarrow [-\infty, -1]$
foo (int i) {
    1: int x := 5;
    2: int y := 7;
    3: if (i ≥ 0) {
        4: y := y + 1;
        5: i := i - 1;
        6: goto 3;
    }
    7: }

1: x → [-∞, ∞], y → [-∞, ∞], i → [-∞, ∞]
2: x → [5, 5], y → [-∞, ∞], i → [-∞, ∞]
3: x → [5, 5], y → [7, ∞], i → [-∞, ∞]
4: x → [5, 5], y → [7, ∞], i → [0, ∞]
5: x → [5, 5], y → [8, 8], i → [0, ∞]
6: x → [5, 5], y → [8, 8], i → [-1, ∞]
7: x → [5, 5], y → [7, ∞], i → [-∞, -1]
### Iterate 9

```c
foo (int i) {
    int x := 5;
    int y := 7;
    if (i >= 0) {
        y := y + 1;
        i := i - 1;
        goto 3;
    }
}
```

<table>
<thead>
<tr>
<th>Step</th>
<th>Value of x</th>
<th>Value of y</th>
<th>Value of i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[-∞, ∞]</td>
<td>[-∞, ∞]</td>
<td>[-∞, ∞]</td>
</tr>
<tr>
<td>2</td>
<td>[5, 5]</td>
<td>[-∞, ∞]</td>
<td>[-∞, ∞]</td>
</tr>
<tr>
<td>3</td>
<td>[5, 5]</td>
<td>[7, ∞]</td>
<td>[-∞, ∞]</td>
</tr>
<tr>
<td>4</td>
<td>[5, 5]</td>
<td>[7, ∞]</td>
<td>[0, ∞]</td>
</tr>
<tr>
<td>5</td>
<td>[5, 5]</td>
<td>[8, ∞]</td>
<td>[0, ∞]</td>
</tr>
<tr>
<td>6</td>
<td>[5, 5]</td>
<td>[8, 8]</td>
<td>[-1, ∞]</td>
</tr>
<tr>
<td>7</td>
<td>[5, 5]</td>
<td>[7, ∞]</td>
<td>[-∞, -1]</td>
</tr>
</tbody>
</table>
foo (int i) {
1: int x := 5;
2: int y := 7;
3: if (i ≥ 0) {
4:   y := y + 1;
5:   i := i - 1;
6:   goto 3;
3:   }  
7:  
}
Iterate 11: a fixed point is reached

foo (int i) {
    int x := 5;
    int y := 7;
    if (i ≥ 0) {
        y := y + 1;
        i := i - 1;
        goto 3;
    }
}

1: x → [-∞,∞], y → [-∞,∞], i → [-∞,∞]
2: x → [5,5], y → [-∞,∞], i → [-∞,∞]
3: x → [5,5], y → [7,∞], i → [-∞,∞]
4: x → [5,5], y → [7,∞], i → [0,∞]
5: x → [5,5], y → [8,∞], i → [0,∞]
6: x → [5,5], y → [8,∞], i → [-1,∞]
7: x → [5,5], y → [7,∞], i → [-∞, -1]
We saw a glimpse of A.I., but enough to get a working intuition with it.

Abstract Interpretation is a rich area with many branches and applications.

A particular branch we use when analyzing neural networks is numerical domains (relating variables).

Transformers of these domains can be very tricky to implement efficiently and correctly!

Good analyzer is a combination of careful Math (e.g., polyhedral computation) + Efficient algorithms and coding.

To use A.I., we typically use existing libraries which implement all transformers and operators we need, e.g.:

→ ELINA (ETH): http://elina.ethz.ch/
→ Apron (ENS): http://apron.cri.ensmp.fr/library/

We will use these libraries when analyzing robustness of neural networks...
AI²: Abstract Interpretation for NNs

Concrete layer transformer

Abstract layer transformer

Concrete

Abstract numerical element

Inputs

Neuron values

Outputs

(0.01, 0.03,..)
(0.94, 0.02,..)
(0.23, 0.12,..)
(0.12, 0.54,..)

...
A numerical domain that is exact for linear operations. Used to analyze numerical programs.

Here, each variable (in our case, abstract neuron) is captured in an affine form.

We can think of this domain as a more extensive version of the Interval domain, we still talk about a single variable, but the variables can also be related a little bit through parameters.
Zonotope Abstract Domain

If we have two (concrete) neurons $n$ and $m$, then the abstract neurons will look like:

\[
\hat{n} = a_0^n + \sum_{i=1}^{k} a_i^n \epsilon_i
\]

\[
\hat{m} = a_0^m + \sum_{i=1}^{k} a_i^m \epsilon_i
\]

The **meaning** (γ) is a polytope centered around $a_0^n$ and $a_0^m$. 

Example of a concretization:
Zonotope Abstract Domain

If we have two (concrete) neurons \( n \) and \( m \), then the abstract neurons will look like:

\[
\hat{n} = a_0^n + \sum_{i=1}^{k} a_i^n \varepsilon_i \\
\hat{m} = a_0^m + \sum_{i=1}^{k} a_i^m \varepsilon_i
\]

\( \varepsilon_i \) : noise terms ranging \([-1,1]\) shared between abstract neurons
\( a_i^n \) : real number that controls magnitude of noise

The meaning \( \gamma \) is a polytope centered around \( a_0^n \) and \( a_0^m \)

Closed under affine transforms, e.g., \( \hat{n} + \hat{m} \)
Not closed under joins and meets, e.g., \( \hat{n} \sqcup \hat{m} \), \( \hat{n} \not\preceq \hat{m} \)

Example of a concretization:
How concretization looks like?

\[ \hat{n} = 3 + \epsilon_1 + 2\epsilon_2 \]
\[ \hat{m} = 0 + \epsilon_1 + \epsilon_2 \]
What is the concretization of this one?

\[
\hat{n} = 1 - 2\epsilon_1 + \epsilon_2 \\
\hat{m} = 0 + \epsilon_1 + \epsilon_2
\]
Zonotope Abstract Domain Operations

Multiplication by a constant real-valued constant $C$:

\[
(a_0^n + \sum_{i=1}^{k} a_i^n \epsilon_i) \ast C = (C \ast a_0^n + \sum_{i=1}^{k} C \ast a_i^n \epsilon_i)
\]

Adding two variables is done component-wise (abstract transformer is exact):

\[
(a_0^n + \sum_{i=1}^{k} a_i^n \epsilon_i) + (a_0^m + \sum_{i=1}^{k} a_i^m \epsilon_i) = (a_0^n + a_0^m) + \sum_{i=1}^{k} (a_i^n + a_i^m) \ast \epsilon_i
\]
Zonotope Abstract Domain Operations

Multiplication of two variables is non-linear, so an approximation is computed. Here is one abstract transformer (not very precise, but often implemented):

\[
(a_0^n + \sum_{i=1}^{k} a_i^n \epsilon_i) \times (a_0^m + \sum_{i=1}^{k} a_i^m \epsilon_i) = \\
\]

This becomes a fresh variable \( \epsilon_{i,j} \)

\[
(a_0^n \times a_0^m) + \sum_{i=1}^{k} (a_i^n \times a_0^m + a_i^m \times a_0^n) \times \epsilon_i + \sum_{i=1}^{k} \sum_{j=1}^{k} a_i^m \times a_j^n \times \epsilon_i \times \epsilon_j
\]

\( \epsilon_{i,j} \in [-1,1] \) if \( i \neq j \)

\( \epsilon_{i,j} \in [0,1] \) if \( i = j \)

Join and Meet are a little more elaborate, and the domain is not closed under these.
AI²: Abstracting Neurons

Abstract

\[ \varphi_0 \] \rightarrow \varphi_1 \rightarrow \cdots \rightarrow \varphi_{n-1} \rightarrow \varphi_n \]

bounded powerset of zonotopes
AI²: Abstracting Neurons

Robustness specification $\varphi_0$

$$x_0 = 0$$
$$x_1 = 0.975 + 0.025\epsilon_1$$
$$x_2 = 0.125$$
$$\ldots$$
$$x_{784} = 0.938 + 0.062\epsilon_{784}$$
$$\forall i, \epsilon_i \in [0,1]$$

bounded powerset of zonotopes
AI²: Abstracting Neurons

Robustness specification ϕ₀

\[ x_0 = 0 \]
\[ x_1 = 0.975 + 0.025\epsilon_1 \]
\[ x_2 = 0.125 \]

\[ \cdots \]
\[ x_{784} = 0.938 + 0.062\epsilon_{784} \]
\[ \forall i, \epsilon_i \in [0,1] \]

Captures a set of images

Bounded powerset of zonotopes
AI²: Abstracting Neurons

Robustness specification $\varphi_0$

$\begin{align*}
x_0 &= 0 \\
x_1 &= 0.975 + 0.025\varepsilon_1 \\
x_2 &= 0.125 \\
&\vdots \\
x_{784} &= 0.938 + 0.062\varepsilon_{784} \\
\forall i. \varepsilon_i &\in [0,1]
\end{align*}$

Output constraint $\varphi_n$

$\begin{align*}
x_0 &= 0 \\
x_1 &= 2.60 + 0.015\varepsilon_0 + 0.023\varepsilon_1 + 5.181\varepsilon_2 + \cdots \\
x_2 &= 4.63 - 0.005\varepsilon_0 - 0.006\varepsilon_1 + 0.023\varepsilon_1 + \cdots \\
&\vdots \\
x_9 &= 0.12 - 0.125\varepsilon_0 + 0.102\varepsilon_1 + 3.012\varepsilon_1 + \cdots \\
\forall i. \varepsilon_i &\in [0,1]
\end{align*}$

Captures a set of images

Captures all possible output vectors

bounded powerset of zonotopes
AI²: Abstracting Neurons

Robustness specification $\varphi_0$

$x_0 = 0$
$x_1 = 0.975 + 0.025\varepsilon_1$
$x_2 = 0.125$

$\ldots$

$x_{784} = 0.938 + 0.062\varepsilon_{784}$
$\forall i. \varepsilon_i \in [0,1]$

Output constraint $\varphi_n$

$x_0 = 0$
$x_1 = 2.60 + 0.015\varepsilon_0 + 0.023\varepsilon_1 + 5.181\varepsilon_2 + \ldots$
$x_2 = 4.63 - 0.005\varepsilon_0 - 0.006\varepsilon_1 + 0.023\varepsilon_2 + \ldots$

$\ldots$

$x_9 = 0.12 - 0.125\varepsilon_0 + 0.102\varepsilon_1 + 3.012\varepsilon_2 + \ldots$
$\forall i. \varepsilon_i \in [0,1]$

Label $i$ is possible iff: $\varphi_n \cap \{\forall j. x_i \geq x_j\} \neq \perp$

(board)
Abstract Neuron Transformers

\[ \hat{a} = 0.2\hat{n} + 0.4\hat{m} \]

\[ \hat{b} = 0.1\hat{n} + 0.5\hat{m} \]

\[ \hat{z} = ReLU(\hat{a}) \]

\[ \hat{q} = ReLU(\hat{b}) \]
Abstract Neuron Transformers

\[ \hat{a} = 0.2\hat{n} + 0.4\hat{m} \]

\[ \hat{b} = 0.1\hat{n} + 0.5\hat{m} \]

Activation function:

\[ z = \text{ReLU}(a) = \max(0, a) \]
Abstract Neuron Transformers

Activation function:  \[ z = \text{ReLU}(a) = \max(0, a) \]

ReLU abstract transformer:

\[ f^\#_{\text{ReLU}} = f^\#_k \circ \cdots \circ f^\#_1 \]

\[ f^\#_i(\psi) = (\psi \cap \{x_i \geq 0\}) \cup \psi_0 \]

\[ \psi_0 = \begin{cases} \lceil x_i = 0 \rceil(\psi) & \text{if } (\psi \cap \{x_i < 0\}) \neq \perp \\ \perp & \text{otherwise} \end{cases} \]
The AI² System

Supports neural networks with:

Layers: Fully-connected, convolutional, max-pooling, flattening
Activation functions: ReLU

Supported numerical domains:

Intervals, Zonotopes, Polyhedra, Bounded powerset domain
Experimental Results

MNIST ConvNet 6 layers, 15K neurons

CIFAR-10 ConvNet 6 layers, 57K neurons
Summary

• Brief introduction to abstract interpretation (approximating functions)

• Example of A.I. on intervals

• Proving robustness of neural networks with A.I. and Zonotope domain

• Open problems: addition activation functions (sigmoid), different kinds of networks, faster Zonotope implementation