Verification of Machine Learning Programs

Guy Katz

The Hebrew University of Jerusalem

Summer School on Foundations of Programming and Software Systems
July 4, 2018
Table of Contents

1 Introduction

2 Neural Networks

3 The Neural Network Verification Problem

4 State-of-the-Art Verification Techniques

5 Reluplex

6 Summary
Software systems are everywhere. Phones, airplanes, hospitals... Complexity is increasing. Autonomous driving... Manually creating software is very difficult.
Software systems are everywhere
- Phones, airplanes, hospitals
Background

- Software systems are everywhere
  - Phones, airplanes, hospitals
- Complexity is increasing
  - Autonomous driving
Background

- Software systems are everywhere
  - Phones, airplanes, hospitals

- Complexity is increasing
  - Autonomous driving

- Manually creating software is *very* difficult
Machine Learning to the Rescue

Image recognition, game playing, autonomous driving, etc.
Machine Learning to the Rescue

Requirements → Machine Learning Algorithm → Artifact

Input/Output Pairs
Image recognition, game playing, autonomous driving, etc.
Can Things go Wrong?

Black-box artifacts are useful
Technology is accessible to non-experts
But their opaqueness can be dangerous
Traditional quality-assurance techniques do not apply
Code reviews? Refactoring? Invariants?
How do we know what is going on inside the black box?
Can Things go Wrong?

- Black-box artifacts are useful
Can Things go Wrong?

- Black-box artifacts are useful
  - Technology is accessible to non-experts
Can Things go Wrong?

- Black-box artifacts are useful
  - Technology is accessible to non-experts
- But their opaqueness can be dangerous
Can Things go Wrong?

- Black-box artifacts are useful
  - Technology is accessible to non-experts
- But their opaqueness can be dangerous
- Traditional quality-assurance techniques do not apply
Can Things go Wrong?

- Black-box artifacts are useful
  - Technology is accessible to non-experts
- But their opaqueness can be dangerous
- Traditional quality-assurance techniques do not apply
  - Code reviews? Refactoring? Invariants?
Can Things go Wrong?

- Black-box artifacts are useful
  - Technology is accessible to non-experts
- But their opaqueness can be dangerous
- Traditional quality-assurance techniques do not apply
  - Code reviews? Refactoring? Invariants?
- How do we know what is going on inside the black box?
When Things go Wrong...
The ACAS Xu System

An Airborne Collision-Avoidance System, for drones
Being developed by the US Federal Aviation Administration (FAA)

Produce an advisory:
Clear-of-conflict (COC)
Strong left
Weak left
Strong right
Weak right

Ownship
ν
Intruder
ν
ρ
ψ
θ
The ACAS Xu System

- An *Airborne Collision-Avoidance System*, for drones
The ACAS Xu System

- An *Airborne Collision-Avoidance System*, for drones
- Being developed by the US Federal Aviation Administration (FAA)
The ACAS Xu System

- An *Airborne Collision-Avoidance System*, for drones
- Being developed by the US Federal Aviation Administration (FAA)
- Produce an advisory:
  - *Clear-of-conflict (COC)*
  - *Strong left*
  - *Weak left*
  - *Strong right*
  - *Weak right*
The ACAS Xu System (cnt’d)

ACAS Xu logic too complex for manual implementation.

Previous approach: large lookup table (size: 2GB)
Interpolate if needed
Switched to neural networks for compression (size: 3MB)
Also smoother than interpolation
But this requires a new certification
Especially because this is a new approach

Guy Katz (HUJI)
ACAS Xu logic *too complex* for manual implementation
ACAS Xu logic *too complex* for manual implementation

Previous approach: large lookup table (size: 2GB)
ACAS Xu logic **too complex** for manual implementation

Previous approach: large lookup table (size: 2GB)
- Interpolate if needed

Switched to neural networks for compression (size: 3MB)
- Also smoother than interpolation

But this requires a new certification especially because this is a new approach
The ACAS Xu System (cnt’d)

- ACAS Xu logic *too complex* for manual implementation
- Previous approach: large lookup table (size: 2GB)
  - Interpolate if needed
- Switched to neural networks for *compression* (size: 3MB)
ACAS Xu logic too complex for manual implementation

Previous approach: large lookup table (size: 2GB)
  - Interpolate if needed

Switched to neural networks for compression (size: 3MB)
  - Also smoother than interpolation
ACAS Xu logic too complex for manual implementation

Previous approach: large lookup table (size: 2GB)
  - Interpolate if needed

Switched to neural networks for compression (size: 3MB)
  - Also smoother than interpolation

But this requires a new certification procedure
ACAS Xu logic *too complex* for manual implementation

- Previous approach: large lookup table (size: 2GB)
  - Interpolate if needed

- Switched to neural networks for *compression* (size: 3MB)
  - Also smoother than interpolation

- But this requires a new *certification* procedure
  - Especially because this is a new approach
The ACAS Xu System (cnt’d)

Certification via testing and simulation

Encounter plots

But these only cover a finite set of inputs

Verification can help
Certification via testing and simulation
The ACAS Xu System (cnt’d)

- Certification via testing and simulation
- *Encounter plots*
Certification via testing and simulation

*Encounter plots*
The ACAS Xu System (cnt’d)

- Certification via testing and simulation
- *Encounter plots*

- But these only cover a finite set of inputs
Certification via testing and simulation

**Encounter plots**

But these only cover a finite set of inputs

- Verification can help
Verification

Given program $P$ and property $\phi$, does $P$ satisfy $\phi$?

Option 1: prove that property $\phi$ holds
Option 2: provide a counter-example showing that it does not

Stronger guarantees than testing: holds for any possible input
Not just a finite set that was tested
But, computational cost much higher

Guy Katz (HUJI)

Verification of ML
Given program $P$ and property $\varphi$, does $P$ satisfy $\varphi$?
Given program $P$ and property $\varphi$, does $P$ satisfy $\varphi$?

- Option 1: prove that property $\varphi$ holds
Verification

- Given program $P$ and property $\varphi$, does $P$ satisfy $\varphi$?
  - Option 1: *prove* that property $\varphi$ holds
  - Option 2: provide a *counter-example* showing that it does not

Stronger guarantees than testing: holds for any possible input. Not just a finite set that was tested. But, computational cost much higher.
Verification

Given program $P$ and property $\varphi$, does $P$ satisfy $\varphi$?
- Option 1: *prove* that property $\varphi$ holds
- Option 2: provide a *counter-example* showing that it does not

Stronger guarantees than testing: holds for *any* possible input
Verification

- Given program $P$ and property $\varphi$, does $P$ satisfy $\varphi$?
  - Option 1: prove that property $\varphi$ holds
  - Option 2: provide a counter-example showing that it does not

- Stronger guarantees than testing: holds for any possible input
  - Not just a finite set that was tested
Verification

Given program $P$ and property $\varphi$, does $P$ satisfy $\varphi$?

- Option 1: prove that property $\varphi$ holds
- Option 2: provide a counter-example showing that it does not

Stronger guarantees than testing: holds for any possible input
- Not just a finite set that was tested

But, computational cost much higher
Verification (cnt’d)

A lot of work on “traditional” systems
Handling common software constructs (e.g., loops, conditions)
Figuring out the properties to check (e.g., no array overflows)
Also, plenty of work on improving scalability
Need to figure this things out for ML-generated software
Is it worth the effort?
Yes, especially for safety-critical systems (like ACAS Xu)
A lot of work on “traditional” systems
A lot of work on “traditional” systems
- Handling common software constructs (e.g., loops, conditions)

Also, plenty of work on improving scalability

Need to figure this things out for ML-generated software

Is it worth the effort?

Yes, especially for safety-critical systems (like ACAS Xu)
A lot of work on “traditional” systems

- Handling common software constructs (e.g., loops, conditions)
- Figuring out the properties to check (e.g., no array overflows)

Also, plenty of work on improving scalability
Need to figure this things out for ML-generated software
Is it worth the effort? 

Yes, especially for safety-critical systems (like ACAS Xu)
A lot of work on “traditional” systems
  - Handling common software constructs (e.g., loops, conditions)
  - Figuring out the properties to check (e.g., no array overflows)

Also, plenty of work on improving scalability
A lot of work on “traditional” systems
- Handling common software constructs (e.g., loops, conditions)
- Figuring out the properties to check (e.g., no array overflows)

Also, plenty of work on improving scalability

Need to figure this things out for ML-generated software
A lot of work on “traditional” systems
  - Handling common software constructs (e.g., loops, conditions)
  - Figuring out the properties to check (e.g., no array overflows)

Also, plenty of work on improving scalability

Need to figure this things out for ML-generated software

Is it worth the effort?
A lot of work on “traditional” systems

- Handling common software constructs (e.g., loops, conditions)
- Figuring out the properties to check (e.g., no array overflows)

Also, plenty of work on improving scalability

Need to figure this things out for ML-generated software

Is it worth the effort?

- Yes, especially for safety-critical systems (like ACAS Xu)
Adversarial Inputs

In 2014, an intriguing property was observed: Small perturbations of inputs lead to misclassification. Can usually find such inputs very easily.
In 2014, an intriguing property was observed:

- Small perturbations of inputs lead to misclassification.
- Can usually find such inputs very easily.
In 2014, an intriguing property was observed:

![Image of panda and gibbon with mathematical expression]

\[ \text{“panda” } \quad 57.7\% \text{ confidence } + \epsilon \times \quad = \quad \text{“gibbon” } \quad 99.3\% \text{ confidence} \]

Goodfellow et al., 2015
In 2014, an intriguing property was observed:

Small perturbations of inputs lead to misclassification.

Goodfellow et al., 2015
In 2014, an intriguing property was observed: small perturbations of inputs lead to misclassification. Can usually find such inputs very easily.

*Small perturbations* of inputs lead to misclassification.

Can usually find such inputs *very* easily.
Adversarial Inputs (cnt’d)

Even worse: can cause misclassification to a specific (targeted) input.

Attacks can be carried out in the real world.

Dangers:
- Natural malformation of input
- Adversary changes “stop” sign into a “entering highway” sign?

Guy Katz (HUJI)
Adversarial Inputs (cnt’d)

- Even worse: can cause misclassification to a specific (targeted) input
Adversarial Inputs (cnt’d)

- Even worse: can cause misclassification to a specific (targeted) input
- Attacks can be carried out in the real world
Adversarial Inputs (cnt’d)

- Even worse: can cause misclassification to a specific (targeted) input
- Attacks can be carried out in the real world
- Dangers:
Adversarial Inputs (cnt’d)

- Even worse: can cause misclassification to a specific \textit{(targeted)} input
- Attacks can be carried out in the \textit{real world}
- Dangers:
  - Natural malformation of input
Adversarial Inputs (cnt’d)

- Even worse: can cause misclassification to a specific \((\textit{targeted})\) input

- Attacks can be carried out in the \(\textit{real world}\)

- Dangers:
  - Natural malformation of input
  - Adversary changes “stop” sign into a “entering highway” sign?
A network’s resilience to adversarial attacks is called adversarial robustness. There exist hardening techniques for increasing robustness, but these usually defend against existing attacks, and then a new attack breaks them. Verification can be used to establish robustness guarantees.
A network’s resilience to adversarial attacks is called \textit{adversarial robustness}. 
Adversarial Robustness

- A network’s resilience to adversarial attacks is called *adversarial robustness*.

- There exist hardening techniques for increasing robustness.
Adversarial Robustness

- A network’s resilience to adversarial attacks is called \textit{adversarial robustness}.
- There exist hardening techniques for increasing robustness.
- But...
A network’s resilience to adversarial attacks is called adversarial robustness.

There exist hardening techniques for increasing robustness.

But...

These usually defend against existing attacks.
Adversarial Robustness

- A network’s resilience to adversarial attacks is called adversarial robustness.
- There exist hardening techniques for increasing robustness.
- But...
  - These usually defend against existing attacks.
  - And then a new attack breaks them.
A network’s resilience to adversarial attacks is called \textit{adversarial robustness}.

There exist hardening techniques for increasing robustness.

But...
- These usually defend against \textit{existing} attacks.
- And then a \textit{new} attack breaks them.

Verification can be used to establish robustness \textit{guarantees}.
Roadmap

Machine-learned software becoming widespread
Problems with these systems already observed
Certification is a new and significant challenge

Up next:
1. See why neural network verification is hard
2. Survey state-of-the-art verification techniques
3. Discuss one technique (Reluplex) in more detail
Roadmap

- Machine-learned software becoming widespread
Roadmap

- Machine-learned software becoming widespread
- Problems with these systems already observed
Roadmap

- Machine-learned software becoming widespread
- Problems with these systems already observed
- Certification is a new and significant challenge
Roadmap

- Machine-learned software becoming widespread
- Problems with these systems already observed
- Certification is a new and significant challenge

Up next:
Roadmap

- Machine-learned software becoming widespread
- Problems with these systems already observed
- Certification is a new and significant challenge

Up next:

- We will focus on neural networks, and will:
Roadmap

- Machine-learned software becoming widespread
- Problems with these systems already observed
- Certification is a new and significant challenge

Up next:

We will focus on neural networks, and will:

1. See why neural network verification is hard
Roadmap

- Machine-learned software becoming widespread
- Problems with these systems already observed
- Certification is a new and significant challenge

Up next:

We will focus on neural networks, and will:

1. See why neural network verification is hard
2. Survey state-of-the-art verification techniques
Roadmap

- Machine-learned software becoming widespread
- Problems with these systems already observed
- Certification is a new and significant challenge

Up next:

We will focus on neural networks, and will:

1. See why neural network verification is hard
2. Survey state-of-the-art verification techniques
3. Discuss one technique (Reluplex) in more detail
Neural Networks

Typical sizes (number of neurons): between few hundreds and millions
Typical sizes (number of neurons): between few hundreds and millions
Neural Networks (cnt’d)

First layer is the input layer. In the ACAS Xu example: sensor readings.

Final layer is the output layer. In the ACAS Xu example: scores for possible advisories.

All other layers are called hidden layers.

Each edge is assigned a weight, and these define the network’s behavior.
Neural Networks (cnt’d)

- First layer is the *input* layer
First layer is the *input* layer
- In ACAS Xu example: sensor readings
Neural Networks (cnt’d)

- First layer is the *input* layer
  - In ACAS Xu example: sensor readings
- Final layer is the *output* layer
First layer is the *input* layer
- In ACAS Xu example: sensor readings

Final layer is the *output* layer
- In ACAS Xu example: scores for possible advisories
First layer is the *input* layer
- In ACAS Xu example: sensor readings

Final layer is the *output* layer
- In ACAS Xu example: scores for possible advisories

All other layers are called *hidden* layers
First layer is the *input* layer
- In ACAS Xu example: sensor readings

Final layer is the *output* layer
- In ACAS Xu example: scores for possible advisories

All other layers are called *hidden* layers

Each edge is assigned a *weight*, and these define the network’s behavior
Training Neural Networks

Weights are determined during the training phase: A network is trained on a finite set of inputs... and then expected to generalize to other inputs. Training is about picking good weights: If the network errs, change weights to correct that behavior. Topic of much research, well beyond our scope. We assume that the network has already been trained.
Weights are determined during the *training* phase:
Weights are determined during the *training* phase:
- A network is trained on a *finite* set of inputs
Weights are determined during the *training* phase:
- A network is trained on a *finite* set of inputs
- ... and then expected to generalize to other inputs
Training Neural Networks

- Weights are determined during the *training* phase:
  - A network is trained on a *finite* set of inputs
  - ... and then expected to generalize to other inputs

- Training is about picking good weights:
Weights are determined during the *training* phase:
- A network is trained on a *finite* set of inputs
- ... and then expected to generalize to other inputs

Training is about picking good weights:
- If the network errs, change weights to correct that behavior
Weights are determined during the *training* phase:
- A network is trained on a *finite* set of inputs
- ... and then expected to generalize to other inputs

Training is about picking good weights:
- If the network errs, change weights to correct that behavior
- Topic of much research, well beyond our scope
Weights are determined during the *training* phase:
- A network is trained on a *finite* set of inputs
- ... and then expected to generalize to other inputs

Training is about picking good weights:
- If the network errs, change weights to correct that behavior
- Topic of much research, well beyond our scope

We assume that the network has already been trained
Evaluating Neural Networks

Nodes evaluated layer by layer:
- Input layer is given
- Every layer computed from its predecessor, according to weights and activation functions
Evaluating Neural Networks

- Nodes evaluated layer by layer:
Evaluating Neural Networks

- Nodes evaluated layer by layer:
  - Input layer is given
Evaluating Neural Networks

- Nodes evaluated layer by layer:
  - Input layer is given
  - Every layer computed from its predecessor, according to *weights* and *activation functions*
Evaluating Neural Networks

- Nodes evaluated layer by layer:
  - Input layer is given
  - Every layer computed from its predecessor, according to \textit{weights} and \textit{activation functions}

\begin{tikzpicture}[node distance=1.5cm]
  \node[shape=circle,draw=blue] (v1) {$v_1$};
  \node[shape=circle,draw=blue,below of=v1] (v2) {$v_2$};
  \node[shape=circle,draw=blue,below of=v2] (v3) {$v_3$};
  \node[shape=circle,draw=blue,above of=v2] (v4) {$v_4$};
  \draw[-stealth,black] (v1) -- (v4) node [midway, above] {$w_1$};
  \draw[-stealth,black] (v2) -- (v4) node [midway, above] {$w_2$};
  \draw[-stealth,black] (v3) -- (v4) node [midway, above] {$w_3$};
\end{tikzpicture}
Evaluating Neural Networks

- Nodes evaluated layer by layer:
  - Input layer is given
  - Every layer computed from its predecessor, according to weights and activation functions

\[ v_4 = \left( \sum_{i=1}^{3} w_i \cdot v_i \right) \]
Nodes evaluated layer by layer:

- Input layer is given
- Every layer computed from its predecessor, according to weights and activation functions

\[ v_4 = f\left(\sum_{i=1}^{3} w_i \cdot v_i\right) \]
Activation Functions

Rectified Linear Unit (ReLU):
\[ f(x) = \max(x, 0) \]
- Active phase: \( x \geq 0 \), output is \( x \)
- Inactive phase: \( x < 0 \), output is 0.
Activation Functions

Rectified Linear Unit (ReLU):
\[ f(x) = \max(x, 0) \]

Active phase: \( x \geq 0 \), output is \( x \).
Inactive phase: \( x < 0 \), output is 0.
Rectified Linear Unit (ReLU): $f(x) = \max(x, 0)$
Activation Functions

- Rectified Linear Unit (ReLU): \( f(x) = \max(x, 0) \)
  - \textit{Active} phase: \( x \geq 0 \), output is \( x \)
Rectified Linear Unit (ReLU): $f(x) = \max(x, 0)$

- **Active phase:** $x \geq 0$, output is $x$
Rectified Linear Unit (ReLU): $f(x) = \max(x, 0)$

- **Active** phase: $x \geq 0$, output is $x$

The equation $0 \cdot 1 + 2 \cdot 2 + 3 \cdot (-1) = 1$ is shown in the diagram.
Rectified Linear Unit (ReLU): $f(x) = \max(x, 0)$

- **Active** phase: $x \geq 0$, output is $x$
Rectified Linear Unit (ReLU): $f(x) = \max(x, 0)$

- Active phase: $x \geq 0$, output is $x$
Rectified Linear Unit (ReLU): $f(x) = \max(x, 0)$

- **Active** phase: $x \geq 0$, output is $x$
- **Inactive** phase: $x < 0$, output is 0.
**Rectified Linear Unit (ReLU):** $f(x) = \max(x, 0)$

- *Active* phase: $x \geq 0$, output is $x$
- *Inactive* phase: $x < 0$, output is 0.
Rectified Linear Unit (ReLU): $f(x) = \max(x, 0)$

- **Active** phase: $x \geq 0$, output is $x$
- **Inactive** phase: $x < 0$, output is 0.

$$1 \cdot 1 + 0 \cdot 2 + 3 \cdot (-1) = -2$$
activation functions

Rectified Linear Unit (ReLU): \( f(x) = \max(x, 0) \)

- **Active** phase: \( x \geq 0 \), output is \( x \)
- **Inactive** phase: \( x < 0 \), output is 0.

\[
1 \cdot 1 + 0 \cdot 2 + 3 \cdot (-1) = -2
\]
Activation Functions (cnt’d)

Pooling layers:
- **Max pooling:**
  \[ f(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n) \]
- **Average pooling:**
  \[ f(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} x_i \]

**Sigmoid function:**
\[ f(x) = \frac{1}{1 + e^{-x}} \]

**Hyperbolic tangent function:**
\[ f(x) = \tanh(x) \]
Pooling layers:

- **Max pooling:**
  \[ f(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n) \]

- **Average pooling:**
  \[ f(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} x_i \]

- **Sigmoid function:**
  \[ f(x) = \frac{1}{1 + e^{-x}} \]

- **Hyperbolic tangent function:**
  \[ f(x) = \tanh(x) \]
Pooling layers:

- Max pooling: \( f(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n) \)
Pooling layers:

- Max pooling: \( f(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n) \)
- Average pooling: \( f(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} x_i \)
Activation Functions (cnt’d)

- Pooling layers:
  - Max pooling: \( f(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n) \)
  - Average pooling: \( f(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} x_i \)

- Sigmoid function: \( f(x) = \frac{1}{1+e^{-x}} \)
Activation Functions (cnt’d)

Pooling layers:
- Max pooling: \( f(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n) \)
- Average pooling: \( f(x_1, \ldots, x_n) = \frac{1}{n} \sum_{i=1}^{n} x_i \)

Sigmoid function: \( f(x) = \frac{1}{1+e^{-x}} \)

Hyperbolic tangent function: \( f(x) = \tanh(x) \)
Table of Contents

1 Introduction

2 Neural Networks

3 The Neural Network Verification Problem

4 State-of-the-Art Verification Techniques

5 Reluplex

6 Summary
Neural Network Verification

Definition (The Neural Network Verification Problem)

For a neural network $N : \bar{x} \rightarrow \bar{y}$, an input property $P(\bar{x})$ and an output property $Q(\bar{y})$, does there exist an input $\bar{x}_0$ with output $\bar{y}_0 = N(\bar{x}_0)$, such that $\bar{x}_0$ satisfies $P$ and $\bar{y}_0$ satisfies $Q$?

$P(\bar{x})$ characterizes the inputs we are checking
$Q(\bar{y})$ characterizes undesired behavior for those inputs

Negative answer (UNSAT) means property holds
Positive answer (SAT) includes a counterexample
Neural Network Verification

Definition (The Neural Network Verification Problem)

For a neural network \( N : \bar{x} \rightarrow \bar{y} \), an input property \( P(\bar{x}) \) and an output property \( Q(\bar{y}) \), does there exist an input \( \bar{x}_0 \) with output \( \bar{y}_0 = N(\bar{x}_0) \), such that \( \bar{x}_0 \) satisfies \( P \) and \( \bar{y}_0 \) satisfies \( Q \)?
Neural Network Verification

Definition (The Neural Network Verification Problem)

For a neural network $N : \bar{x} \rightarrow \bar{y}$, an input property $P(\bar{x})$ and an output property $Q(\bar{y})$, does there exist an input $\bar{x}_0$ with output $\bar{y}_0 = N(\bar{x}_0)$, such that $\bar{x}_0$ satisfies $P$ and $\bar{y}_0$ satisfies $Q$?

- $P(\bar{x})$ characterizes the inputs we are checking

- $Q(\bar{y})$ characterizes undesired behavior for those inputs

Negative answer (UNSAT) means property holds
Positive answer (SAT) includes a counterexample
### Definition (The Neural Network Verification Problem)

For a neural network $N : \bar{x} \rightarrow \bar{y}$, an input property $P(\bar{x})$ and an output property $Q(\bar{y})$, does there exist an input $\bar{x}_0$ with output $\bar{y}_0 = N(\bar{x}_0)$, such that $\bar{x}_0$ satisfies $P$ and $\bar{y}_0$ satisfies $Q$?

- $P(\bar{x})$ characterizes the inputs we are checking
- $Q(\bar{y})$ characterizes *undesired* behavior for those inputs
Definition (The Neural Network Verification Problem)

For a neural network $N : \bar{x} \rightarrow \bar{y}$, an input property $P(\bar{x})$ and an output property $Q(\bar{y})$, does there exist an input $\bar{x}_0$ with output $\bar{y}_0 = N(\bar{x}_0)$, such that $\bar{x}_0$ satisfies $P$ and $\bar{y}_0$ satisfies $Q$?

- $P(\bar{x})$ characterizes the inputs we are checking
- $Q(\bar{y})$ characterizes undesired behavior for those inputs
- Negative answer (UNSAT) means property holds
Neural Network Verification

**Definition (The Neural Network Verification Problem)**

For a neural network $N: \bar{x} \rightarrow \bar{y}$, an input property $P(\bar{x})$ and an output property $Q(\bar{y})$, does there exist an input $\bar{x}_0$ with output $\bar{y}_0 = N(\bar{x}_0)$, such that $\bar{x}_0$ satisfies $P$ and $\bar{y}_0$ satisfies $Q$?

- $P(\bar{x})$ characterizes the inputs we are checking
- $Q(\bar{y})$ characterizes *undesired* behavior for those inputs
- Negative answer (UNSAT) means property holds
- Positive answer (SAT) includes a *counterexample*
Example: ACAS Xu

\[ P(\bar{x}) : \bar{x}[0] \geq 40000 \]

\[ Q(\bar{y}) : \begin{align*}
\bar{y}[0] & \leq \bar{y}[1] \\
\bar{y}[0] & \leq \bar{y}[2] \\
\bar{y}[0] & \leq \bar{y}[3] \\
\bar{y}[0] & \leq \bar{y}[4]
\end{align*} \]

UNSAT means the system behaves as expected.
Example: ACAS Xu

- Want to ensure: whenever intruder is distant, network always answers *clear-of-conflict*
Example: ACAS Xu

- Want to ensure: whenever intruder is distant, network always answers *clear-of-conflict*

- $P(\bar{x})$: 

\[
\begin{align*}
\bar{x}_0 & \geq 40000 \\
\bar{y}_0 & \leq \bar{y}_1 \\
\bar{y}_0 & \leq \bar{y}_2 \\
\bar{y}_0 & \leq \bar{y}_3 \\
\bar{y}_0 & \leq \bar{y}_4
\end{align*}
\]

*UNSAT* means the system behaves as expected
Example: ACAS Xu

- Want to ensure: whenever intruder is distant, network always answers *clear-of-conflict*

- $P(\bar{x})$:
  - $\bar{x}[0] \geq 40000$
Example: ACAS Xu

- Want to ensure: whenever intruder is distant, network always answers *clear-of-conflict*

  - $P(\bar{x})$:
    - $\bar{x}[0] \geq 40000$
  
  - $Q(\bar{y})$:
Example: ACAS Xu

- Want to ensure: whenever intruder is distant, network always answers *clear-of-conflict*

- \( P(\bar{x}):\)
  - \( \bar{x}[0] \geq 40000 \)

- \( Q(\bar{y}):\)
  - \( (\bar{y}[0] \leq \bar{y}[1]) \lor (\bar{y}[0] \leq \bar{y}[2]) \lor (\bar{y}[0] \leq \bar{y}[3]) \lor (\bar{y}[0] \leq \bar{y}[4]) \)
Example: ACAS Xu

- Want to ensure: whenever intruder is distant, network always answers *clear-of-conflict*

  \[ P(\bar{x}): \]
  \[ \bar{x}[0] \geq 40000 \]

  \[ Q(\bar{y}): \]
  \[ (\bar{y}[0] \leq \bar{y}[1]) \lor (\bar{y}[0] \leq \bar{y}[2]) \lor (\bar{y}[0] \leq \bar{y}[3]) \lor (\bar{y}[0] \leq \bar{y}[4]) \]

- UNSAT means the system behaves as expected
Example: Adversarial Robustness

\[ \text{Want to ensure: for a given input } \bar{x}_0 \text{ and a given amount of noise } \delta, \text{ classification remains the same} \]

\[ P(\bar{x}) : \| \bar{x} - \bar{x}_0 \|_{\infty} \leq \delta \]

Equivalent to:

\[ \bigwedge_i (-\delta \leq \bar{x}[i] - \bar{x}_0[i] \leq \delta) \]

\[ Q(\bar{y}) : \bigvee_i (\bar{y}[i] \leq \bar{y}_0[i]), \text{ where } \bar{y}_0[i] \text{ is the desired label} \]

UNSAT means the system behaves as expected.
Example: Adversarial Robustness

- Want to ensure: for a given input $\bar{x}_0$ and a given amount of noise $\delta$, classification remains the same

$$\|\bar{x} - \bar{x}_0\|_{L_\infty} \leq \delta$$

Equivalent to:

$$\bigwedge_i (-\delta \leq \bar{x}[i] - \bar{x}_0[i] \leq \delta)$$

$$\bigvee_i (\bar{y}[i]_0 \leq \bar{y}[i])$$

where $\bar{y}[i]_0$ is the desired label

UNSAT means the system behaves as expected
Example: Adversarial Robustness

- Want to ensure: for a given input $\bar{x}_0$ and a given amount of noise $\delta$, classification remains the same

- $P(\bar{x})$: 

\[
\|\bar{x} - \bar{x}_0\|_{\infty} \leq \delta
\]

Equivalent to:

\[
\bigwedge_i (-\delta \leq \bar{x}_i - \bar{x}_0_i \leq \delta)
\]

$Q(\bar{y})$: 

\[
\bigvee_i (\bar{y}_i \leq \bar{y}_i^0)
\]

where $\bar{y}_i^0$ is the desired label

UNSAT means the system behaves as expected
Example: Adversarial Robustness

- Want to ensure: for a given input $\bar{x}_0$ and a given amount of noise $\delta$, classification remains the same

- $P(\bar{x})$:
  - $\|\bar{x} - \bar{x}_0\|_{L_\infty} \leq \delta$
Example: Adversarial Robustness

- Want to ensure: for a given input $\bar{x}_0$ and a given amount of noise $\delta$, classification remains the same

- $P(\bar{x})$:
  - $\|\bar{x} - \bar{x}_0\|_{L_\infty} \leq \delta$
  - Equivalent to: $\land_i (-\delta \leq \bar{x}[i] - \bar{x}_0[i] \leq \delta)$
Example: Adversarial Robustness

- Want to ensure: for a given input $\bar{x}_0$ and a given amount of noise $\delta$, classification remains the same

- $P(\bar{x})$:
  - $\|\bar{x} - \bar{x}_0\|_{L_{\infty}} \leq \delta$
  - Equivalent to: $\bigwedge_i (-\delta \leq \bar{x}[i] - \bar{x}_0[i] \leq \delta)$

- $Q(\bar{y})$:
Example: Adversarial Robustness

- Want to ensure: for a given input $\bar{x}_0$ and a given amount of noise $\delta$, classification remains the same

  \[ P(\bar{x}) : \]
  - $\| \bar{x} - \bar{x}_0 \|_{L_\infty} \leq \delta$
  - Equivalent to: $\bigwedge_i (-\delta \leq \bar{x}[i] - \bar{x}_0[i] \leq \delta)$

  \[ Q(\bar{y}) : \]
  - $\bigvee_i (\bar{y}[i_0] \leq \bar{y}[i])$, where $\bar{y}[i_0]$ is the desired label
Example: Adversarial Robustness

- Want to ensure: for a given input $\bar{x}_0$ and a given amount of noise $\delta$, classification remains the same

$P(\bar{x})$:
- $\|\bar{x} - \bar{x}_0\|_{L_\infty} \leq \delta$
- Equivalent to: $\bigwedge_i (-\delta \leq \bar{x}[i] - \bar{x}_0[i] \leq \delta)$

$Q(\bar{y})$:
- $\bigvee_i (\bar{y}[i_0] \leq \bar{y}[i])$, where $\bar{y}[i_0]$ is the desired label

UNSAT means the system behaves as expected
Theorem (Neural Network Verification Complexity)

For a neural network with ReLU activation functions, and for properties $P()$ and $Q()$ that are conjunctions of linear constraints, the verification problem is NP-complete in the number of ReLU nodes.
Verification Complexity

**Theorem (Neural Network Verification Complexity)**

For a neural network with ReLU activation functions, and for properties $P()$ and $Q()$ that are conjunctions of linear constraints, the verification problem is NP-complete in the number of ReLU nodes.

- Membership in NP: can check in polynomial time that a given $x$ satisfies $P(x)$ and $Q(N(x))$. 

Guy Katz (HUJI)
Theorem (Neural Network Verification Complexity)

For a neural network with ReLU activation functions, and for properties $P()$ and $Q()$ that are conjunctions of linear constraints, the verification problem is NP-complete in the number of ReLU nodes.

- **Membership in NP**: can check in polynomial time that a given $x$ satisfies $P(x)$ and $Q(N(x))$

- **NP-Hardness**: by reduction from 3-SAT
Verification Complexity (cnt’d)

Boolean variables: $x_1, \ldots, x_n$

Input to 3-SAT:

$$C_1 \land C_2 \land \ldots \land C_k$$

Each clause $C_i$ is $q_{1i} \lor q_{2i} \lor q_{3i}$

$q_i$'s are variables or their negations

Goal: find a variable assignment that satisfies the formula

We will construct an input to the verification problem that is satisfiable iff the formula is satisfiable
Verification Complexity (cnt’d)

- Boolean variables: $x_1, \ldots, x_n$

Input to 3-SAT:

Each clause $C_i$ is $q_{1i} \lor q_{2i} \lor q_{3i}$'s are variables or their negations

Goal: find a variable assignment that satisfies the formula
Verification Complexity (cnt’d)

- Boolean variables: $x_1, \ldots, x_n$
- Input to 3-SAT: $C_1 \land C_2 \land \ldots \land C_k$

Each clause $C_i$ is $q_{1i} \lor q_{2i} \lor q_{3i}$ where $q_{1i}$'s are variables or their negations.
Verification Complexity (cnt’d)

- Boolean variables: $x_1, \ldots, x_n$
- Input to 3-SAT: $C_1 \land C_2 \land \ldots \land C_k$
- Each clause $C_i$ is $q^1_i \lor q^2_i \lor q^3_i$
Boolean variables: \( x_1, \ldots, x_n \)

Input to 3-SAT: \( C_1 \land C_2 \land \ldots \land C_k \)

Each clause \( C_i \) is \( q_i^1 \lor q_i^2 \lor q_i^3 \)
- \( q \)'s are variables or their negations
Boolean variables: $x_1, \ldots, x_n$

Input to 3-SAT: $C_1 \land C_2 \land \ldots \land C_k$

Each clause $C_i$ is $q_i^1 \lor q_i^2 \lor q_i^3$
  - $q$'s are variables or their negations

Goal: find a variable assignment that satisfies the formula
Verification Complexity (cnt’d)

- Boolean variables: $x_1, \ldots, x_n$
- Input to 3-SAT: $C_1 \land C_2 \land \ldots \land C_k$
- Each clause $C_i$ is $q_i^1 \lor q_i^2 \lor q_i^3$
  - $q$’s are variables or their negations
- Goal: find a variable assignment that satisfies the formula
- We will construct an input to the verification problem that is satisfiable iff the formula is satisfiable
Reduction: Handling Negations

\[ q_{j_i} \geq 1 - x_{j_i}, \text{i.e.} \quad q_{j_i} = \neg x_{j_i} \]
Reduction: Handling Negations

\[ q^i_j = \neg x^j \]

Guy Katz (HUJI)
Reduction: Handling Negations

$q^j_i$ gets $1 - x_j$, i.e. $q^j_i = \neg x_j$
Reduction: Handling Disjunctions

At least one input is 1:
- If $t_i$ is 0, $y_i$ is 1

All inputs are 0:
- If $t_i$ is 1, $y_i$ is 0

In other words:
$$y_i = q_1^i \lor q_2^i \lor q_3^i$$
Reduction: Handling Disjunctions

At least one input is 1:
- If $t_i$ is 0, $y_i$ is 1.
- All inputs are 0:
  - $t_i$ is 1,
  - $y_i$ is 0.

In other words:
$$y_i = q^1_i \lor q^2_i \lor q^3_i$$
At least one input is 1: $t_i$ is 0, $y_i$ is 1
Reduction: Handling Disjunctions

At least one input is 1: \( t_i \) is 0, \( y_i \) is 1

All inputs are 0: \( t_i \) is 1, \( y_i \) is 0
At least one input is 1: $t_i$ is 0, $y_i$ is 1

All inputs are 0: $t_i$ is 1, $y_i$ is 0

In other words: $y_i = q_i^1 \lor q_i^2 \lor q_i^3$
Reduction: Handling Conjunctions

\[ y \]

\[ y \neg \]

\[ \ldots \]

\[ y \]

\[ 1 \]

\[ 1 \]

\[ y \]

\[ \text{is the final output of the network} \]

We define the output property, \( Q(y) \), to be

\[ y = \neg \]

This is satisfied only if all conjuncts are 1.
Reduction: Handling Conjunctions

We define the output property, $Q(y)$, to be $y = n$.

This is satisfied only if all conjuncts are 1.
Reduction: Handling Conjunctions

\( y \) is the final output of the network

\[ y = \prod_{i=1}^{n} y_i \]

- \( \prod \) denotes the conjunction (AND) operation.
- \( y_i \) are the inputs to the network.
- \( y \) is the output of the network.

This is satisfied only if all conjuncts are 1.
$y$ is the final output of the network

We define the output property, $Q(y)$, to be $y = n$
Reduction: Handling Conjunctions

- $y_1$ 1
- $y_n$ 1
- $y$

- $y$ is the final output of the network
- We define the output property, $Q(y)$, to be $y = n$
- This is satisfied only if all conjuncts are 1
Reduction: Putting it all Together

Input property $P(x)$: $\forall i. x_i \in \{0, 1\}$

Output property $Q(y)$: $y = n$

Verification property $\text{SAT}$ iff original formula is $\text{SAT}$
Reduction: Putting it all Together

Input property $P(x)$: $\forall i. x_i \in \{0, 1\}$

Output property $Q(y)$: $y = n$

Verification property SAT iff original formula is SAT
Input property $P(x)$: $\forall i. \ x_i \in \{0, 1\}$
Input property $P(x)$: $\forall i. \ x_i \in \{0, 1\}$

Output property $Q(y)$: $y = n$
Input property $P(x): \forall i. \ x_i \in \{0, 1\}$

Output property $Q(y): y = n$

Verification property SAT iff original formula is SAT
Extending the Definition for P() and Q()

Corollary

The verification problem remains NP-complete if we allow P() and Q() to have arbitrary Boolean structure.

Proof: we add (polynomially many) nodes to handle disjunctions and negations.

So, it is enough to solve just for conjunctions.
Extending the Definition for \(P()\) and \(Q()\)

**Corollary**

*The verification problem remains \(NP\)-complete if we allow \(P()\) and \(Q()\) to have arbitrary Boolean structure*
Extending the Definition for $P()$ and $Q()$

**Corollary**

*The verification problem remains NP-complete if we allow $P()$ and $Q()$ to have arbitrary Boolean structure*

- Proof: we add (polynomially many) nodes to handle disjunctions and negations
Corollary

The verification problem remains NP-complete if we allow \( P() \) and \( Q() \) to have arbitrary Boolean structure.

- Proof: we add (polynomially many) nodes to handle disjunctions and negations.
- So, it is enough to solve just for *conjunctions*.
Another Extension: Max-Pooling

ReLU is a piece-wise linear function. Max-Pooling is also piece-wise linear. It is enough to solve just for ReLUs. Other piece-wise linear functions? Non piece-wise linear functions?
Another Extension: Max-Pooling

- ReLU is a piece-wise linear function
Another Extension: Max-Pooling

- ReLU is a piece-wise linear function
- Max-Pooling is also piece-wise linear
Another Extension: Max-Pooling

- ReLU is a piece-wise linear function
- Max-Pooling is also piece-wise linear
- Can express one in terms of the other:

\[
\text{ReLU}(x) = \max(x, 0)
\]

\[
\max(x, y) = \text{ReLU}(x - y) + y
\]
Another Extension: Max-Pooling

- ReLU is a piece-wise linear function
- Max-Pooling is also piece-wise linear
- Can express one in terms of the other:
  - ReLU\( (x) = \max(x, 0) \)
Another Extension: Max-Pooling

- ReLU is a piece-wise linear function
- Max-Pooling is also piece-wise linear
- Can express one in terms of the other:
  - $\text{ReLU}(x) = \max(x, 0)$
  - $\max(x, y) = \text{ReLU}(x - y) + y$
Another Extension: Max-Pooling

- ReLU is a piece-wise linear function
- Max-Pooling is also piece-wise linear
- Can express one in terms of the other:
  - $\text{ReLU}(x) = \max(x, 0)$
  - $\max(x, y) = \text{ReLU}(x - y) + y$
- It is enough to solve just for ReLUs
Another Extension: Max-Pooling

- ReLU is a piece-wise linear function
- Max-Pooling is also piece-wise linear
- Can express one in terms of the other:
  - \( \text{ReLU}(x) = \max(x, 0) \)
  - \( \max(x, y) = \text{ReLU}(x - y) + y \)
- It is enough to solve just for ReLUs
- Other piece-wise linear functions?
Another Extension: Max-Pooling

- ReLU is a piece-wise linear function
- Max-Pooling is also piece-wise linear
- Can express one in terms of the other:
  - ReLU($x$) = $\max(x, 0)$
  - $\max(x, y) = \text{ReLU}(x - y) + y$
- It is enough to solve just for ReLUs
- Other piece-wise linear functions?
- Non piece-wise linear functions?
Roadmap

Neural network verification is hard
NP-complete even for simple networks and properties
Real networks can be quite large

So what can we do?

Next, we will:
1. Survey state-of-the-art verification techniques
2. Discuss one such technique (Reluplex) in more detail
Neural network verification is hard
Roadmap

- Neural network verification is *hard*
  - NP-complete even for simple networks and properties
Neural network verification is *hard*
- NP-complete even for simple networks and properties
- Real networks can be quite large
Neural network verification is *hard*
  - NP-complete even for simple networks and properties
  - Real networks can be quite large

So what can we do?
Roadmap

- Neural network verification is *hard*
  - NP-complete even for simple networks and properties
  - Real networks can be quite large

- So what can we do?

- Next, we will:
Roadmap

- Neural network verification is *hard*
  - NP-complete even for simple networks and properties
  - Real networks can be quite large

- So what can we do?

- Next, we will:
  1. Survey state-of-the-art verification techniques
Roadmap

- Neural network verification is **hard**
  - NP-complete even for simple networks and properties
  - Real networks can be quite large

- So what can we do?

- Next, we will:
  1. Survey state-of-the-art verification techniques
  2. Discuss one such technique (Reluplex) in more detail
# Table of Contents

1. **Introduction**

2. **Neural Networks**

3. **The Neural Network Verification Problem**

4. **State-of-the-Art Verification Techniques**

5. **Reluplex**

6. **Summary**
Disclaimer: The literature on neural network verification is growing rapidly. The work mentioned here is just a sample. Apologies to all authors whose work is not cited.
Techniques and Challenges

Main challenge is scalability. Usually the case in verification.

Two kinds of techniques:
- Sound and complete: limited scalability, always succeed.
- Sound and incomplete: better scalability, can return "don't know".

Orthogonal: abstraction techniques.
Related: testing techniques (e.g., coverage criteria, concolic testing). Not covered here.
Techniques and Challenges

- Main challenge is *scalability*
Main challenge is *scalability*

- Usually the case in verification
Main challenge is *scalability*

- Usually the case in verification

Two kinds of techniques:

- Sound and complete: limited scalability, always succeed
- Sound and incomplete: better scalability, can return "don't know"
Techniques and Challenges

- Main challenge is *scalability*
  - Usually the case in verification

- Two kinds of techniques:
  - *Sound* and *complete*:

Orthogonal: abstraction techniques
Related: testing techniques (e.g., *coverage criteria*, *concolic* testing). Not covered here.
Techniques and Challenges

- Main challenge is *scalability*
  - Usually the case in verification

- Two kinds of techniques:
  - *Sound* and *complete*:
    - limited scalability

Orthogonal: abstraction techniques
Related: testing techniques (e.g., coverage criteria, concolic testing). Not covered here.
Main challenge is *scalability*
- Usually the case in verification

Two kinds of techniques:
- *Sound* and *complete*:
  - limited scalability
  - always succeed

Orthogonal: abstraction techniques
Related: testing techniques (e.g., coverage criteria, concolic testing). Not covered here
Techniques and Challenges

- Main challenge is *scalability*
  - Usually the case in verification

- Two kinds of techniques:
  - *Sound* and *complete*:
    - limited scalability
    - always succeed
  - *Sound* and *incomplete*:
Main challenge is *scalability*

- Usually the case in verification

Two kinds of techniques:

- *Sound* and *complete*:
  - limited scalability
  - always succeed
- *Sound* and *incomplete*:
  - better scalability
Main challenge is *scalability*
- Usually the case in verification

Two kinds of techniques:
- *Sound and complete*:
  - limited scalability
  - always succeed
- *Sound and incomplete*:
  - better scalability
  - can return “don’t know”
Techniques and Challenges

- Main challenge is *scalability*
  - Usually the case in verification

- Two kinds of techniques:
  - *Sound* and *complete*:
    - limited scalability
    - always succeed
  - *Sound* and *incomplete*:
    - better scalability
    - can return “don’t know”

- Orthogonal: *abstraction* techniques
Techniques and Challenges

- Main challenge is *scalability*
  - Usually the case in verification

- Two kinds of techniques:
  - *Sound* and *complete*:
    - limited scalability
    - always succeed
  - *Sound* and *incomplete*:
    - better scalability
    - can return “don’t know”

- Orthogonal: *abstraction* techniques

- Related: testing techniques (e.g., *coverage criteria*, *concolic testing*). Not covered here
So, How Big a Network can you Verify?

Very difficult to compare!

Different properties make a huge difference.

Compare complete and incomplete techniques.

Different underlying engines.

Different benchmarks.

Comparative study: Bunel et al, 2017 [BTT+17]

Still, as a rule of thumb...

Complete techniques: hundreds to thousands.

Incomplete techniques: thousands to tens of thousands.
So, How Big a Network can you Verify?

- Very difficult to compare!
So, How Big a Network can you Verify?

- Very difficult to compare!
  - Different properties make a huge difference
So, How Big a Network can you Verify?

- Very difficult to compare!
  - Different properties make a huge difference
  - Compare complete and incomplete techniques
So, How Big a Network can you Verify?

- Very difficult to compare!
  - Different *properties* make a huge difference
  - Compare *complete* and *incomplete* techniques
  - Different underlying *engines*
So, How Big a Network can you Verify?

- Very difficult to compare!
  - Different properties make a huge difference
  - Compare complete and incomplete techniques
  - Different underlying engines
  - Different benchmarks

Comparative study: Bunel et al, 2017 [BTT+17]

Still, as a rule of thumb...
- Complete techniques: hundreds to thousands
- Incomplete techniques: thousands to tens of thousands
So, How Big a Network can you Verify?

- Very difficult to compare!
  - Different *properties* make a huge difference
  - Compare *complete* and *incomplete* techniques
  - Different underlying *engines*
  - Different *benchmarks*
    - Comparative study: Bunel et al, 2017 [BTT$^+$17]
Very difficult to compare!

- Different properties make a huge difference
- Compare complete and incomplete techniques
- Different underlying engines
- Different benchmarks
  - Comparative study: Bunel et al, 2017 [BTT+17]

Still, as a rule of thumb...
So, How Big a Network can you Verify?

- Very difficult to compare!
  - Different *properties* make a huge difference
  - Compare *complete* and *incomplete* techniques
  - Different underlying *engines*
  - Different *benchmarks*
    - Comparative study: Bunel et al, 2017 [BTT+17]

- Still, as a rule of thumb...
  - *Complete* techniques: hundreds to *thousands*
So, How Big a Network can you Verify?

- Very difficult to compare!
  - Different *properties* make a huge difference
  - Compare *complete* and *incomplete* techniques
  - Different underlying *engines*
  - Different *benchmarks*
    - Comparative study: Bunel et al, 2017 [BTT⁺17]

- Still, as a rule of thumb...
  - *Complete* techniques: hundreds to *thousands*
  - *Incomplete* techniques: thousands to *tens of thousands*
NeVeR (Pulina and Tacchella, 2010) [PT10]

Among first attempts to verify neural networks, focused on networks with Sigmoid activation functions. The main idea was to over-approximate Sigmoids using interval arithmetic, and then apply the interval arithmetic solver HySAT.
Among first attempts to verify neural networks
Among first attempts to verify neural networks
Focused on networks with Sigmoid activation functions
Among first attempts to verify neural networks
Focused on networks with Sigmoid activation functions
Main idea: *over-approximate* Sigmoids using *interval arithmetic*
Among first attempts to verify neural networks
Focused on networks with Sigmoid activation functions
Main idea: *over-approximate* Sigmoids using *interval arithmetic*
... and then apply the interval arithmetic solver HySAT
Over-Approximations

A common theme in verification

Core idea: replace a system $S$ with a simpler $\bar{S}$

All behaviors of $S$ appear in $\bar{S}$

But additional, spurious behaviors also exist in $\bar{S}$

Because $\bar{S}$ is simpler, it is easier to verify
Over-Approximations

- A common theme in verification
Over-Approximations

- A common theme in verification
- Core idea: replace a system $S$ with a *simpler* $\bar{S}$
Over-Approximations

- A common theme in verification
- Core idea: replace a system $S$ with a \textit{simpler} $\bar{S}$
- \textit{All behaviors} of $S$ appear in $\bar{S}$
Over-Approximations

- A common theme in verification
- Core idea: replace a system $S$ with a simpler $\bar{S}$
- *All behaviors* of $S$ appear in $\bar{S}$
  - But additional, *spurious* behaviors also exist in $\bar{S}$
Over-Approximations

- A common theme in verification
- Core idea: replace a system $S$ with a simpler $\bar{S}$
- *All behaviors* of $S$ appear in $\bar{S}$
  - But additional, *spurious* behaviors also exist in $\bar{S}$
  - Because $\bar{S}$ is simpler, it is *easier to verify*
Over-Approximations (cnt’d)

If $\overline{S}$ is correct, so is $S$. Because all behaviors of $S$ exist in $\overline{S}$.

If $\overline{S}$ is incorrect:

Either $S$ is also incorrect
Or the detected bad behavior is spurious

If needed, $\overline{S}$ is refined to remove the spurious behavior, and the process is repeated.
Over-Approximations (cnt’d)

If $\bar{S}$ is correct, so is $S$ because all behaviors of $S$ exist in $\bar{S}$.

If $\bar{S}$ is incorrect:
Either $S$ is also incorrect
Or the detected bad behavior is spurious

If needed, $\bar{S}$ is refined to remove the spurious behavior, and the process is repeated.
If $\bar{S}$ is correct, so is $S$. Because all behaviors of $S$ exist in $\bar{S}$.

If $\bar{S}$ is incorrect:
- Either $S$ is also incorrect
- Or the detected bad behavior is spurious

If needed, $\bar{S}$ is refined to remove the spurious behavior, and the process is repeated.
Over-Approximations (cnt’d)

If $\bar{S}$ is correct, so is $S$

- If $\bar{S}$ is correct, so is $S$

Because all behaviors of $S$ exist in $\bar{S}$.

If $\bar{S}$ is incorrect:

- Either $S$ is also incorrect
- Or the detected bad behavior is spurious

If needed, $\bar{S}$ is refined to remove the spurious behavior, and the process is repeated.
Over-Approximations (cnt’d)

- If $\overline{S}$ is correct, so is $S$
  - Because all behaviors of $S$ exist in $\overline{S}$

If $\overline{S}$ is incorrect:
- Either $S$ is also incorrect
- Or the detected bad behavior is spurious

If needed, $\overline{S}$ is refined to remove the spurious behavior, and the process is repeated.
Over-Approximations (cnt’d)

If $\bar{S}$ is correct, so is $S$

- Because all behaviors of $S$ exist in $\bar{S}$

If $\bar{S}$ is incorrect:

- Either $S$ is also incorrect
- Or the detected bad behavior is spurious

If needed, $\bar{S}$ is refined to remove the spurious behavior, and the process is repeated.
Over-Approximations (cnt’d)

- If $\overline{S}$ is correct, so is $S$
  - Because all behaviors of $S$ exist in $\overline{S}$
- If $\overline{S}$ is incorrect:
  - Either $S$ is also incorrect

Guy Katz (HUJI)
If $\bar{S}$ is correct, so is $S$
  - Because all behaviors of $S$ exist in $\bar{S}$

If $\bar{S}$ is incorrect:
  - Either $S$ is also incorrect
Over-Approximations (cnt’d)

- If $\overline{S}$ is correct, so is $S$
  - Because all behaviors of $S$ exist in $\overline{S}$

- If $\overline{S}$ is incorrect:
  - Either $S$ is also incorrect
  - Or the detected bad behavior is spurious
Over-Approximations (cnt’d)

- If $\overline{S}$ is correct, so is $S$
  - Because all behaviors of $S$ exist in $\overline{S}$

- If $\overline{S}$ is incorrect:
  - Either $S$ is also incorrect
  - Or the detected bad behavior is spurious
Over-Approximations (cnt’d)

- If $\bar{S}$ is correct, so is $S$
  - Because all behaviors of $S$ exist in $\bar{S}$

- If $\bar{S}$ is incorrect:
  - Either $S$ is also incorrect
  - Or the detected bad behavior is spurious

- If needed, $\bar{S}$ is *refined* to remove the spurious behavior, and the process is repeated
NeVeR (Pulina and Tacchella, 2010) [PT10]

Abstraction used by Pulina and Tacchella:

For $x \in [a, b]$, we just know that $f(x)$ is in some range $[y_a, y_b]$.

When a spurious example is found, the $x$ segments are made smaller, and bounds are made tighter.

First step, but could only tackle very small networks (10 neurons)
Abstraction used by Pulina and Tacchella:
Abstraction used by Pulina and Tacchella:

For $x \in [x_a, x_b]$ we just know that $f(x)$ is in some range $[y_a, y_b]$. 

![Diagram showing the abstraction process](image-url)
Abstraction used by Pulina and Tacchella:

For $x \in [x_a, x_b]$ we just know that $f(x)$ is in some range $[y_a, y_b]$

When a spurious example is found, the $x$ segments are made smaller, and bounds are made tighter
Abstraction used by Pulina and Tacchella:

For $x \in [x_a, x_b]$ we just know that $f(x)$ is in some range $[y_a, y_b]$

When a spurious example is found, the $x$ segments are made smaller, and bounds are made tighter.

First step, but could only tackle very small networks (10 neurons)
A technique for evaluating a network’s adversarial robustness

A reduction from a verification-like problem to linear programming

Did not directly study verification

But core idea very useful for verification
Bastani et al, 2016 [BIL⁺16]

- A technique for evaluating a network’s adversarial robustness
Bastani et al, 2016 [BIL+16]

- A technique for evaluating a network’s adversarial robustness
- A reduction from a verification-like problem to *linear programming*
A technique for evaluating a network’s adversarial robustness

A reduction from a verification-like problem to linear programming

Did not directly study verification
- A technique for evaluating a network’s adversarial robustness
- A reduction from a verification-like problem to *linear programming*
- Did not directly study verification
  - But core idea very useful for verification
Linear Programming (LP)

A linear program:

$$\min \bar{c} \cdot \bar{x}$$
subject to

$$A \cdot \bar{x} = \bar{b}$$
and

$$\bar{l} \leq \bar{x} \leq \bar{u}$$

Intuitively:

- Set of variables $\bar{x}$, each with lower ($\bar{l}$) and upper ($\bar{u}$) bounds
- Set of linear equations that need to hold ($A \cdot \bar{x} = \bar{b}$)
- Some objective function to optimize $\bar{c} \cdot \bar{x}$

Highly useful for many problems in CS, studied for many decades

Problem known to be in P, powerful solvers exist
A linear program:
A linear program:

minimize $\bar{c} \cdot \bar{x}$

subject to $A \cdot \bar{x} = \bar{b}$

and $\bar{l} \leq \bar{x} \leq \bar{u}$

Intuitively:
- Set of variables $\bar{x}$, each with lower ($\bar{l}$) and upper ($\bar{u}$) bounds
- Set of linear equations that need to hold ($A \cdot \bar{x} = \bar{b}$)
- Some objective function to optimize $\bar{c} \cdot \bar{x}$

Highly useful for many problems in CS, studied for many decades

Problem known to be in $P$, powerful solvers exist
A linear program:

\[
\begin{align*}
\text{minimize} & \quad \bar{c} \cdot \bar{x} \\
\text{subject to} & \quad A \cdot \bar{x} = \bar{b} \\
\text{and} & \quad \bar{l} \leq \bar{x} \leq \bar{u}
\end{align*}
\]

Intuitively:

- Set of variables $\bar{x}$, each with lower ($\bar{l}$) and upper ($\bar{u}$) bounds
- Set of linear equations that need to hold ($A \cdot \bar{x} = \bar{b}$)
- Some objective function to optimize $\bar{c} \cdot \bar{x}$

Highly useful for many problems in CS, studied for many decades. The problem known to be in $P$, powerful solvers exist.
Linear Programming (LP)

A linear program:

\[
\begin{align*}
\text{minimize} & \quad \bar{c} \cdot \bar{x} \\
\text{subject to} & \quad A \cdot \bar{x} = \bar{b} \\
\text{and} & \quad \bar{l} \leq \bar{x} \leq \bar{u}
\end{align*}
\]

Intuitively:
- Set of variables $\bar{x}$, each with lower ($\bar{l}$) and upper ($\bar{u}$) bounds
Linear Programming (LP)

- A linear program:

  
  
  minimize $\bar{c} \cdot \bar{x}$  
  subject to $A \cdot \bar{x} = \bar{b}$  
  and $\bar{l} \leq \bar{x} \leq \bar{u}$

- Intuitively:
  
  - Set of variables $\bar{x}$, each with lower ($\bar{l}$) and upper ($\bar{u}$) bounds
  - Set of linear equations that need to hold ($A \cdot \bar{x} = \bar{b}$)
A linear program:

\[
\begin{align*}
\text{minimize} & \quad \bar{c} \cdot \bar{x} \\
\text{subject to} & \quad A \cdot \bar{x} = \bar{b} \\
\text{and} & \quad \bar{l} \leq \bar{x} \leq \bar{u}
\end{align*}
\]

Intuitively:

- Set of variables \( \bar{x} \), each with lower (\( \bar{l} \)) and upper (\( \bar{u} \)) bounds
- Set of linear equations that need to hold (\( A \cdot \bar{x} = \bar{b} \))
- Some objective function to optimize \( \bar{c} \cdot \bar{x} \)
A linear program:

\[
\begin{align*}
\text{minimize} & \quad \bar{c} \cdot \bar{x} \\
\text{subject to} & \quad A \cdot \bar{x} = \bar{b} \\
\text{and} & \quad \bar{l} \leq \bar{x} \leq \bar{u}
\end{align*}
\]

Intuitively:

- Set of variables $\bar{x}$, each with lower ($\bar{l}$) and upper ($\bar{u}$) bounds
- Set of linear equations that need to hold ($A \cdot \bar{x} = \bar{b}$)
- Some objective function to optimize $\bar{c} \cdot \bar{x}$

\textbf{Highly} useful for many problems in CS, studied for many decades
Linear Programming (LP)

- A linear program:
  \[
  \text{minimize} \quad \bar{c} \cdot \bar{x} \\
  \text{subject to} \quad A \cdot \bar{x} = \bar{b} \\
  \text{and} \quad \bar{l} \leq \bar{x} \leq \bar{u}
  \]

- Intuitively:
  - Set of variables \( \bar{x} \), each with lower (\( \bar{l} \)) and upper (\( \bar{u} \)) bounds
  - Set of linear equations that need to hold (\( A \cdot \bar{x} = \bar{b} \))
  - Some objective function to optimize \( \bar{c} \cdot \bar{x} \)

- Highly useful for many problems in CS, studied for many decades

- Problem known to be in \( \mathbb{P} \), powerful solvers exist
Replacing ReLUs with Linear Constraints

Let $y = \text{ReLU}(x)$. Each ReLU has two phases:

**Active phase:** $(x \geq 0) \land (y = x)$

**Inactive phase:** $(x \leq 0) \land (y = 0)$

Each phase is a linear constraint.

True for all piece-wise linear functions, not just ReLUs.

If a ReLU is known to be in a specific phase, it can be discarded and replaced with a linear equation.
Let $y = \text{ReLU}(x)$. Each ReLU has two phases:
Let $y = \text{ReLU}(x)$. Each ReLU has two phases:

- **Active** phase: $(x \geq 0) \land (y = x)$
Let $y = \text{ReLU}(x)$. Each ReLU has two phases:

- **Active** phase: $(x \geq 0) \land (y = x)$
- **Inactive** phase: $(x \leq 0) \land (y = 0)$
Let \( y = \text{ReLU}(x) \). Each ReLU has two phases:

- **Active** phase: \((x \geq 0) \land (y = x)\)
- **Inactive** phase: \((x \leq 0) \land (y = 0)\)

Each phase is a *linear* constraint.
Replacing ReLUs with Linear Constraints

- Let $y = \text{ReLU}(x)$. Each ReLU has two phases:
  - **Active** phase: $(x \geq 0) \land (y = x)$
  - **Inactive** phase: $(x \leq 0) \land (y = 0)$

- Each phase is a *linear* constraint
  - True for all piece-wise linear functions, not just ReLUs
Replacing ReLUs with Linear Constraints

- Let $y = \text{ReLU}(x)$. Each ReLU has two phases:
  - *Active* phase: $(x \geq 0) \land (y = x)$
  - *Inactive* phase: $(x \leq 0) \land (y = 0)$

- Each phase is a *linear* constraint
  - True for all piece-wise linear functions, not just ReLUs

- If a ReLU is known to be in a specific phase, it can be discarded and *replaced* with a linear equation
To look for adversarial inputs around a point $\bar{x}_0$:

1. Encode the network's weighted sums as linear equations.
2. Evaluate the network for $\bar{x}_0$.
3. For every $y = \text{ReLU}(x)$:
   - If it is active for $\bar{x}_0$, replace it with $(x \geq 0) \land (y = x)$.
   - If it is inactive, replace it with $(x \leq 0) \land (y = 0)$.
4. Have an LP solver look for adversarial inputs.

Evaluated on image recognition networks:

- Efficient (LP solvers are fast), sound, but incomplete:
  - Discovered adversarial inputs are correct.
  - But may miss some adversarial inputs.
To look for adversarial inputs around a point $\bar{x}_0$:

1. Encode the network's weighted sums as linear equations.
2. Evaluate the network for $\bar{x}_0$.
3. For every $y = \text{ReLU}(x)$:
   - If it is active for $\bar{x}_0$, replace it with $x \geq 0 \land y = x$.
   - If it is inactive, replace it with $x \leq 0 \land y = 0$.
4. Have an LP solver look for adversarial inputs.

Evaluated on image recognition networks:
- Efficient (LP solvers are fast), sound, but incomplete.
- Discovered adversarial inputs are correct but may miss some adversarial inputs.
To look for adversarial inputs around a point $\bar{x}_0$:

- Encode the network’s weighted sums as linear equations.
To look for adversarial inputs around a point $\bar{x}_0$:

- Encode the network’s weighted sums as linear equations
- Evaluate the network for $\bar{x}_0$
To look for adversarial inputs around a point $\bar{x}_0$:
- Encode the network’s weighted sums as linear equations
- Evaluate the network for $\bar{x}_0$
- For every $y = \text{ReLU}(x)$:
  - If it is active for $\bar{x}_0$, replace it with $(x \geq 0) \land (y = x)$
  - If it is inactive, replace it with $(x \leq 0) \land (y = 0)$

Have an LP solver look for adversarial inputs

Evaluated on image recognition networks
Efficient (LP solvers are fast), sound, but incomplete:
Discovered adversarial inputs are correct
But may miss some adversarial inputs
To look for adversarial inputs around a point $\bar{x}_0$:
- Encode the network’s weighted sums as linear equations
- Evaluate the network for $\bar{x}_0$
- For every $y = \text{ReLU}(x)$:
  - If it is active for $\bar{x}_0$, replace it with $(x \geq 0) \land (y = x)$
  - If it is inactive, replace it with $(x \leq 0) \land (y = 0)$

Evaluated on image recognition networks
Efficient (LP solvers are fast), sound, but incomplete:
Discovered adversarial inputs are correct
But may miss some adversarial inputs
To look for adversarial inputs around a point $\bar{x}_0$:

- Encode the network’s weighted sums as linear equations
- Evaluate the network for $\bar{x}_0$
- For every $y = \text{ReLU}(x)$:
  - If it is \textit{active} for $\bar{x}_0$, replace it with $(x \geq 0) \land (y = x)$
  - If it is \textit{inactive}, replace it with $(x \leq 0) \land (y = 0)$

Have an LP solver look for adversarial inputs

Evaluated on image recognition networks

Efficient (LP solvers are fast), sound, but incomplete:

Discovered adversarial inputs are correct
But may miss some adversarial inputs
To look for adversarial inputs around a point $\bar{x}_0$:

- Encode the network’s weighted sums as linear equations.
- Evaluate the network for $\bar{x}_0$.
- For every $y = \text{ReLU}(x)$:
  - If it is *active* for $\bar{x}_0$, replace it with $(x \geq 0) \land (y = x)$.
  - If it is *inactive*, replace it with $(x \leq 0) \land (y = 0)$.

- Have an LP solver look for adversarial inputs.
To look for adversarial inputs around a point $\bar{x}_0$:

1. Encode the network’s weighted sums as linear equations
2. Evaluate the network for $\bar{x}_0$
3. For every $y = \text{ReLU}(x)$:
   - If it is *active* for $\bar{x}_0$, replace it with $(x \geq 0) \land (y = x)$
   - If it is *inactive*, replace it with $(x \leq 0) \land (y = 0)$
4. Have an LP solver look for adversarial inputs

Evaluated on image recognition networks
To look for adversarial inputs around a point $\bar{x}_0$:

- Encode the network’s weighted sums as linear equations
- Evaluate the network for $\bar{x}_0$
- For every $y = \text{ReLU}(x)$:
  - If it is active for $\bar{x}_0$, replace it with $(x \geq 0) \land (y = x)$
  - If it is inactive, replace it with $(x \leq 0) \land (y = 0)$
- Have an LP solver look for adversarial inputs

Evaluated on image recognition networks

- **Efficient** (LP solvers are fast), **sound**, but **incomplete**:
To look for adversarial inputs around a point $\bar{x}_0$:

- Encode the network’s weighted sums as linear equations.
- Evaluate the network for $\bar{x}_0$.
- For every $y = \text{ReLU}(x)$:
  - If it is active for $\bar{x}_0$, replace it with $(x \geq 0) \land (y = x)$
  - If it is inactive, replace it with $(x \leq 0) \land (y = 0)$

- Have an LP solver look for adversarial inputs.

Evaluated on image recognition networks.

*Efficient* (LP solvers are fast), *sound*, but *incomplete*:

- Discovered adversarial inputs are correct.
To look for adversarial inputs around a point $\overline{x}_0$:
- Encode the network’s weighted sums as linear equations
- Evaluate the network for $\overline{x}_0$
- For every $y = \text{ReLU}(x)$:
  - If it is active for $\overline{x}_0$, replace it with $(x \geq 0) \land (y = x)$
  - If it is inactive, replace it with $(x \leq 0) \land (y = 0)$
- Have an LP solver look for adversarial inputs

Evaluated on image recognition networks

**Efficient** (LP solvers are fast), **sound**, but **incomplete**:
- Discovered adversarial inputs are correct
- But may miss some adversarial inputs
Reducing Verification to Linear Programming

A complete extension of the technique from Bastani et al
Case splitting: an enumeration of all possibilities:
For each ReLU, guess whether it is active or inactive
Solve the resulting LP
If a solution is found, return SAT
Otherwise, go back and try another guess
If all guesses are exhausted, return UNSAT

Very similar to the naive algorithm for Boolean satisfiability
A *complete* extension of the technique from Bastani et al
A *complete* extension of the technique from Bastani et al

*Case splitting*: an enumeration of all possibilities:
A *complete* extension of the technique from Bastani et al

**Case splitting**: an enumeration of all possibilities:
- For each ReLU, *guess* whether it is active or inactive
A complete extension of the technique from Bastani et al

Case splitting: an enumeration of all possibilities:
- For each ReLU, guess whether it is active or inactive
- Solve the resulting LP
A complete extension of the technique from Bastani et al

Case splitting: an enumeration of all possibilities:
- For each ReLU, guess whether it is active or inactive
- Solve the resulting LP
- If a solution is found, return SAT
A complete extension of the technique from Bastani et al

Case splitting: an enumeration of all possibilities:
  - For each ReLU, guess whether it is active or inactive
  - Solve the resulting LP
  - If a solution is found, return SAT
  - Otherwise, go back and try another guess
A complete extension of the technique from Bastani et al

Case splitting: an enumeration of all possibilities:
- For each ReLU, guess whether it is active or inactive
- Solve the resulting LP
- If a solution is found, return SAT
- Otherwise, go back and try another guess
- If all guesses are exhausted, return UNSAT
Reducing Verification to Linear Programming

- A *complete* extension of the technique from Bastani et al

- **Case splitting**: an enumeration of all possibilities:
  - For each ReLU, *guess* whether it is active or inactive
  - Solve the resulting LP
  - If a solution is found, return SAT
  - Otherwise, go back and try another guess
  - If all guesses are exhausted, return UNSAT

- Very similar to the naive algorithm for Boolean satisfiability
Reducing Verification to Linear Programming (cnt’d)
Case splitting creates a *search tree*
Reducing Verification to Linear Programming (cnt’d)

- Case splitting creates a search tree
- Problem is SAT iff at least one leaf is SAT
Case splitting creates a \textit{search tree}

Problem is SAT iff at least one leaf is SAT

\[
y_1 = \text{ReLU}(x_1), \quad y_2 = \text{ReLU}(x_2)
\]

\[
y_1 = 0, x_1 \leq 0 \quad y_1 = x_1, x_1 \geq 0
\]

\[
y_2 = 0, x_2 \leq 0 \quad y_2 = x_2, x_2 \geq 0 \quad y_2 = 0, x_2 \leq 0 \quad y_2 = x_2, x_2 \geq 0
\]
Reducing Verification to Linear Programming (cnt’d)

Sound and complete case splitting approach proposed in [KBD+17a]

Approach very sensitive to heuristics and tricks for trimming the search space

Much like Boolean satisfiability

Several sound and complete variations, including:

- Ehlers, 2017 [Ehl17] (the Planet solver)
- Tjeng and Tedrake, 2017 [TT17]
- Bunel et al, 2017 [BTT+17] (the BaB solver)
- Lomuscio and Maganti, 2017 [LM17]
- Dutta et al, 2018 [DJST18] (the Sherlock solver)
Reducing Verification to Linear Programming (cnt’d)

- *Sound* and *complete* case splitting approach proposed in [KBD\textsuperscript{+}17a]

Approach very sensitive to heuristics and tricks for trimming the search space

Much like Boolean satisfiability

Several sound and complete variations, including:
- Ehlers, 2017 [Ehl17] (the Planet solver)
- Tjeng and Tedrake, 2017 [TT17]
- Bunel et al, 2017 [BTT\textsuperscript{+}17] (the BaB solver)
- Lomuscio and Maganti, 2017 [LM17]
- Dutta et al, 2018 [DJST18] (the Sherlock solver)
Reducing Verification to Linear Programming (cnt’d)

- *Sound* and *complete* case splitting approach proposed in [KBD+17a]
- Approach very sensitive to *heuristics* and tricks for trimming the search space
Reducing Verification to Linear Programming (cnt’d)

- **Sound** and **complete** case splitting approach proposed in [KBD+17a]

- Approach very sensitive to **heuristics** and tricks for trimming the search space
  - Much like Boolean satisfiability
Reducing Verification to Linear Programming (cnt’d)

- **Sound** and **complete** case splitting approach proposed in [KBD$^+$17a]

- Approach very sensitive to **heuristics** and tricks for trimming the search space
  - Much like Boolean satisfiability

- Several **sound** and **complete** variations, including:
  - Ehlers, 2017 [Ehl17] (the Planet solver)
  - Tjeng and Tedrake, 2017 [TT17]
  - Bunel et al, 2017 [BTT$^+$17] (the BaB solver)
  - Lomuscio and Maganti, 2017 [LM17]
  - Dutta et al, 2018 [DJST18] (the Sherlock solver)
Reducing Verification to Linear Programming (cnt’d)

- *Sound* and *complete* case splitting approach proposed in [KBD+17a]

- Approach very sensitive to *heuristics* and tricks for trimming the search space
  - Much like Boolean satisfiability

- Several *sound* and *complete* variations, including:
  - Ehlers, 2017 [Ehl17] (the *Planet* solver)
Reducing Verification to Linear Programming (cnt’d)

- **Sound** and **complete** case splitting approach proposed in [KBD⁺17a]

- Approach very sensitive to **heuristics** and tricks for trimming the search space
  - Much like Boolean satisfiability

- Several **sound** and **complete** variations, including:
  - Ehlers, 2017 [Ehl17] (the *Planet* solver)
  - Tjeng and Tedrake, 2017 [TT17]
Sound and complete case splitting approach proposed in [KBD+17a]

Approach very sensitive to heuristics and tricks for trimming the search space
  - Much like Boolean satisfiability

Several sound and complete variations, including:
  - Ehlers, 2017 [Ehl17] (the Planet solver)
  - Tjeng and Tedrake, 2017 [TT17]
  - Bunel et al, 2017 [BTT+17] (the BaB solver)
Sound and complete case splitting approach proposed in [KBD⁺17a]

Approach very sensitive to heuristics and tricks for trimming the search space
  - Much like Boolean satisfiability

Several sound and complete variations, including:
  - Ehlers, 2017 [Ehl17] (the Planet solver)
  - Tjeng and Tedrake, 2017 [TT17]
  - Bunel et al, 2017 [BTT⁺17] (the BaB solver)
  - Lomuscio and Maganti, 2017 [LM17]
Reducing Verification to Linear Programming (cnt’d)

- **Sound** and **complete** case splitting approach proposed in [KBD⁺17a]

- Approach very sensitive to **heuristics** and tricks for trimming the search space
  - Much like Boolean satisfiability

- Several **sound** and **complete** variations, including:
  - Ehlers, 2017 [Ehl17] (the **Planet** solver)
  - Tjeng and Tedrake, 2017 [TT17]
  - Bunel et al, 2017 [BTT⁺17] (the **BaB** solver)
  - Lomuscio and Maganti, 2017 [LM17]
  - Dutta et al, 2018 [DJST18] (the **Sherlock** solver)
DLV (Huang et al, 2017) [HKWW17]

Apply a discretization of the input space. Discretization via manipulations. These can represent camera scratches, rotations, etc. Sound but incomplete.

Then do an exhaustive search, layer-by-layer.

Tool: the DLV solver, evaluated on image recognition networks.
DLV (Huang et al, 2017) [HKWW17]

- Apply a *discretization* of the input space
DLV (Huang et al, 2017) [HKWW17]

- Apply a *discretization* of the input space
  - Discretization via *manipulations*
Apply a *discretization* of the input space

- Discretization via *manipulations*
- These can represent camera scratches, rotations, etc
Apply a \textit{discretization} of the input space

- Discretization via \textit{manipulations}
- These can represent camera scratches, rotations, etc
- \textit{Sound} but \textit{incomplete}
Apply a *discretization* of the input space

- Discretization via *manipulations*
- These can represent camera scratches, rotations, etc
- *Sound* but *incomplete*
Apply a *discretization* of the input space

- Discretization via *manipulations*
- These can represent camera scratches, rotations, etc
- *Sound* but *incomplete*

Then do an *exhaustive* search, layer-by-layer
Apply a discretization of the input space
- Discretization via manipulations
- These can represent camera scratches, rotations, etc
- Sound but incomplete

Then do an exhaustive search, layer-by-layer

Tool: the DLV solver, evaluated on image recognition networks
AI² (Gehr et al, 2018) [GMDC⁺18]
Over-approximation of the *input property*
Over-approximation of the *input property*

- Over-approximate with polyhedra
Over-approximation of the *input property*
- Over-approximate with polyhedra
- Propagate polyhedra layer-by-layer
Over-approximation of the *input property*

- Over-approximate with polyhedra
- Propagate polyhedra layer-by-layer
Over-approximation of the *input property*

- Over-approximate with polyhedra
- Propagate polyhedra layer-by-layer

*Sound but incomplete*
Over-approximation of the *input property*
- Over-approximate with polyhedra
- Propagate polyhedra layer-by-layer

*Sound but incomplete*
- Abstract property holds $\Rightarrow$ original property holds
Over-approximation of the *input property*

- Over-approximate with polyhedra
- Propagate polyhedra layer-by-layer

*Sound but incomplete*

- Abstract property holds \(\Rightarrow\) original property holds
- Converse not necessarily true
Networks as Continuous Functions

The network is a continuous function from input to output.

Verification: analyzing this function's properties.

Can reduce properties to single output.

Analyze a real-valued function.

Find lower and upper bounds on the output.
The network is a *continuous* function from input to output.
The network is a *continuous* function from input to output

Verification: analyzing this *function’s properties*
Networks as Continuous Functions

- The network is a *continuous* function from input to output
- Verification: analyzing this *function’s properties*
  - Can reduce properties to single output
Networks as Continuous Functions

The network is a *continuous* function from input to output

Verification: analyzing this *function’s properties*
  - Can reduce properties to single output
  - Analyze a real-valued function
The network is a *continuous* function from input to output

Verification: analyzing this *function’s properties*
- Can reduce properties to single output
- Analyze a real-valued function

Find lower and upper bounds on the output
DeepGO (Ruan et al, 2018) [RHK18]

Lipschitz Continuity:

\[ |f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2| \]

- \( K \) is the Lipschitz constant.
- The best \( K \) is the smallest one.

Partition input, bound output on each piece, refine if needed.
Lipschitz Continuity: \[ |f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2| \]
Lipschitz Continuity: \[|f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2|\]

- \(K\) is the Lipschitz constant

\[\text{DeepGO (Ruan et al, 2018) [RHK18]}\]
DeepGO (Ruan et al, 2018) [RHK18]

- **Lipschitz Continuity**: $|f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2|$
  - $K$ is the Lipschitz constant
  - The *best* $K$ is the smallest one
DeepGO (Ruan et al, 2018) [RHK18]

- **Lipschitz Continuity**: \(|f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2|
  - \(K\) is the Lipschitz constant
  - The *best* \(K\) is the smallest one

- Partition input, bound output on each piece, refine if needed
**Lipschitz Continuity:** \[ |f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2| \]

- \( K \) is the Lipschitz constant
- The best \( K \) is the smallest one

Partition input, bound output on each piece, refine if needed
DeepGO (Ruan et al, 2018) [RHK18] (cnt’d)

Terminate when the discovered bounds imply the property

Complexity also related to size of input domain
Tool: DeepGO [RHK18]
DeepGO (Ruan et al, 2018) [RHK18] (cnt’d)

- Tool: DeepGO [RHK18]
- Iteratively refine partition until bounds sufficiently accurate
Tool: DeepGO [RHK18]

Iteratively refine partition until bounds sufficiently accurate

- Guaranteed to converge (complete), assuming a small acceptable error
Tool: DeepGO [RHK18]

- Iteratively refine partition until bounds sufficiently accurate
  - Guaranteed to converge (complete), assuming a small acceptable error
  - Smaller values of $K$ lead to faster convergence
DeepGO (Ruan et al, 2018) [RHK18] (cnt’d)

- Tool: DeepGO [RHK18]

- Iteratively refine partition until bounds sufficiently accurate
  - Guaranteed to converge (complete), assuming a small acceptable error
  - Smaller values of $K$ lead to faster convergence

- Terminate when the discovered bounds imply the property
Tool: DeepGO [RHK18]

- Iteratively refine partition until bounds sufficiently accurate
  - Guaranteed to converge (complete), assuming a small acceptable error
  - Smaller values of $K$ lead to faster convergence

- Terminate when the discovered bounds imply the property

- Complexity also related to size of input domain
Additional Techniques at a Glance

Verification of Binarized Neural Networks
Cheng et al [CNR17b], Narodytska et al [NKR+18]
Verification using quadratic solvers
Cheng et al [CNR17a]
Network reachability analysis via over-approximations
around specific inputs
Xiang et al [XTJ18]
Additional Techniques at a Glance

- Verification of *Binarized* Neural Networks
Additional Techniques at a Glance

- Verification of *Binarized* Neural Networks
  - Cheng et al [CNR17b], Narodytska et al [NKR+18]
Additional Techniques at a Glance

- Verification of *Binarized* Neural Networks
  - Cheng et al [CNR17b], Narodytska et al [NKR\textsuperscript{+}18]
- Verification using *quadratic solvers*
Additional Techniques at a Glance

- Verification of *Binarized* Neural Networks
  - Cheng et al [CNR17b], Narodytska et al [NKR$^+$18]

- Verification using *quadratic solvers*
  - Cheng et al [CNR17a]
Additional Techniques at a Glance

- Verification of *Binarized* Neural Networks
  - Cheng et al [CNR17b], Narodytska et al [NKR\textsuperscript{+}18]

- Verification using *quadratic solvers*
  - Cheng et al [CNR17a]

- Network reachability analysis via *over-approximations* around specific inputs
Additional Techniques at a Glance

- Verification of *Binarized* Neural Networks
  - Cheng et al [CNR17b], Narodytska et al [NKR+18]

- Verification using *quadratic solvers*
  - Cheng et al [CNR17a]

- Network reachability analysis via *over-approximations* around specific inputs
  - Xiang et al [XTJ18]
Additional Techniques at a Glance (cnt’d)

- **Supporting the $\| \cdot \|_0$ norm**
  - Ruan et al [RWS+18]

- **Parallelization**
  - Katz et al [KBD+17b], Wang et al [WPW+18]

- **Additional Lipschitz-based approaches**
  - Hull et al [HWZ02], Hein and Andriushchenko [HA17], Weng et al [WZC+18]

- **Training safe networks**
  - Dvijotham et al [DGS+18], Raghunathan et al [RSL18]
Additional Techniques at a Glance (cnt’d)

- Supporting the $L_0$ norm
Additional Techniques at a Glance (cnt’d)

- Supporting the $L_0$ norm
  - Ruan et al [RWS$^+$18]
Supporting the $L_0$ norm
- Ruan et al [RWS$^+$18]

Parallelization by partitioning the input space

*Parallelization* by partitioning the input space
Additional Techniques at a Glance (cnt’d)

- Supporting the $L_0$ norm
  - Ruan et al [RWS$^+$18]

- **Parallelization** by partitioning the input space
  - Katz et al [KBD$^+17b$], Wang et al [WPW$^+18$]
Additional Techniques at a Glance (cnt’d)

- Supporting the $L_0$ norm
  - Ruan et al [RWS$^+$18]

- **Parallelization** by partitioning the input space
  - Katz et al [KBD$^+$17b], Wang et al [WPW$^+$18]

- Additional *Lipschitz-based* approaches
Additional Techniques at a Glance (cnt’d)

- Supporting the $L_0$ norm
  - Ruan et al [RWS+18]

- **Parallelization** by partitioning the input space
  - Katz et al [KBD+17b], Wang et al [WPW+18]

- Additional *Lipschitz-based* approaches
  - Hull et al [HWZ02], Hein and Andriushchenko [HA17], Weng et al [WZC+18]
Additional Techniques at a Glance (cnt’d)

- Supporting the $L_0$ norm
  - Ruan et al [RWS$^+$18]

- Parallelization by partitioning the input space
  - Katz et al [KBD$^+$17b], Wang et al [WPW$^+$18]

- Additional Lipschitz-based approaches
  - Hull et al [HWZ02], Hein and Andriushchenko [HA17], Weng et al [WZC$^+$18]

- Training safe networks
Additional Techniques at a Glance (cnt’d)

- Supporting the $L_0$ norm
  - Ruan et al [RWS$^+$18]

- **Parallelization** by partitioning the input space
  - Katz et al [KBD$^+$17b], Wang et al [WPW$^+$18]

- Additional **Lipschitz-based** approaches
  - Hull et al [HWZ02], Hein and Andriushchenko [HA17], Weng et al [WZC$^+$18]

- **Training** safe networks
  - Dvijotham et al [DGS$^+$18], Raghunathan et al [RSL18]
Roadmap

Neural network verification is hard NP-complete even for simple networks and properties. Reducible to an exponential sequence of easy problems. Sound and complete. Much work on finding efficient heuristics. Can trade completeness for better scalability. Can be combined with abstraction techniques.

Next, we will: 1. Focus on one sound and complete technique (Reluplex) in greater detail.
Neural network verification is hard
Neural network verification is hard

- NP-complete even for simple networks and properties
Roadmap

- Neural network verification is **hard**
  - NP-complete even for simple networks and properties
- Reducible to an *exponential sequence of easy problems*
Roadmap

- Neural network verification is *hard*
  - NP-complete even for simple networks and properties

- Reducible to an *exponential sequence of easy problems*
  - Sound and complete
Roadmap

- Neural network verification is *hard*
  - NP-complete even for simple networks and properties

- Reducible to an *exponential sequence of easy problems*
  - Sound and complete
  - Much work on finding efficient heuristics
Roadmap

- Neural network verification is *hard*
  - NP-complete even for simple networks and properties
- Reducible to an *exponential sequence of easy problems*
  - Sound and complete
  - Much work on finding efficient heuristics
- Can *trade completeness* for better *scalability*
Neural network verification is \textit{hard}.

- NP-complete even for simple networks and properties

Reducible to an \textit{exponential sequence of easy problems}.

- Sound and complete
- Much work on finding efficient heuristics

Can \textit{trade completeness} for better \textit{scalability}.

Can be combined with \textit{abstraction techniques}.
Neural network verification is hard
- NP-complete even for simple networks and properties

Reducible to an exponential sequence of easy problems
- Sound and complete
- Much work on finding efficient heuristics

Can trade completeness for better scalability

Can be combined with abstraction techniques

Next, we will:
Neural network verification is *hard*
- NP-complete even for simple networks and properties

Reducible to an *exponential sequence of easy problems*
- Sound and complete
- Much work on finding efficient heuristics

Can *trade completeness* for better *scalability*

Can be combined with *abstraction techniques*

Next, we will:
1. Focus on one sound and complete technique (Reluplex) in greater detail
1. Introduction
2. Neural Networks
3. The Neural Network Verification Problem
4. State-of-the-Art Verification Techniques
5. Reluplex
6. Summary
Reluplex

Joint work with Clark Barrett, David Dill, Kyle Julian and Mykel Kochenderfer (CAV 2017 [KBD+17a]), supported by the FAA and Intel.

A sound and complete verification procedure.

Applied to the ACAS Xu case study.

Networks an order of magnitude larger than previously possible.

Project still ongoing.
Joint work with Clark Barrett, David Dill, Kyle Julian and Mykel Kochenderfer (CAV 2017 [KBD+17a]), supported by the FAA and Intel.
Joint work with Clark Barrett, David Dill, Kyle Julian and Mykel Kochenderfer (CAV 2017 [KBD+17a]), supported by the FAA and Intel

A sound and complete verification procedure
Reluplex

- Joint work with Clark Barrett, David Dill, Kyle Julian and Mykel Kochenderfer (CAV 2017 [KBD+17a]), supported by the FAA and Intel

- A sound and complete verification procedure

- Applied to the ACAS Xu case study
Joint work with Clark Barrett, David Dill, Kyle Julian and Mykel Kochenderfer (CAV 2017 [KBD+17a]), supported by the FAA and Intel

- A sound and complete verification procedure
- Applied to the ACAS Xu case study
  - Networks an order of magnitude larger than previously possible
Reluplex

- Joint work with Clark Barrett, David Dill, Kyle Julian and Mykel Kochenderfer (CAV 2017 [KBD+17a]), supported by the FAA and Intel

- A sound and complete verification procedure

- Applied to the ACAS Xu case study
  - Networks an order of magnitude larger than previously possible

- Project still ongoing
Reluplex (cnt’d)

SMT-solver for quantifier-free linear real arithmetic + ReLUs

Based on the Simplex method for linear programming

Simplex + ReLUs = Reluplex

Applicable to other piece-wise linear functions

Key SMT idea: handle ReLUs lazily

As opposed to eager case splitting

Defer splitting for as long as possible

May not have to split at all!

But first, an introduction to Simplex
Reluplex (cnt’d)

- SMT-solver for quantifier-free linear real arithmetic + ReLUs
Reluplex (cnt’d)

- SMT-solver for quantifier-free linear real arithmetic + ReLUs
- Based on the *Simplex* method for linear programming
SMT-solver for quantifier-free linear real arithmetic + ReLUs
Based on the *Simplex* method for linear programming
  - Simplex + ReLUs = Reluplex
Reluplex (cnt’d)

- SMT-solver for quantifier-free linear real arithmetic + ReLUs
- Based on the *Simplex* method for linear programming
  - Simplex + ReLUs = Reluplex
  - Applicable to other piece-wise linear functions
Reluplex (cnt’d)

- SMT-solver for quantifier-free linear real arithmetic + ReLUs
- Based on the *Simplex* method for linear programming
  - Simplex + ReLUs = Reluplex
  - Applicable to other piece-wise linear functions
- Key SMT idea: handle ReLUs *lazily*
Reluplex (cnt’d)

- SMT-solver for quantifier-free linear real arithmetic + ReLUs
- Based on the *Simplex* method for linear programming
  - Simplex + ReLUs = Reluplex
  - Applicable to other piece-wise linear functions
- Key SMT idea: handle ReLUs *lazily*
  - As opposed to eager case splitting
Reluplex (cnt’d)

- SMT-solver for quantifier-free linear real arithmetic + ReLUs
- Based on the *Simplex* method for linear programming
  - Simplex + ReLUs = Reluplex
  - Applicable to other piece-wise linear functions
- Key SMT idea: handle ReLUs *lazily*
  - As opposed to eager case splitting
  - *Defer* splitting for as long as possible
Reluplex (cnt’d)

- SMT-solver for quantifier-free linear real arithmetic + ReLUs
- Based on the *Simplex* method for linear programming
  - Simplex + ReLUs = Reluplex
  - Applicable to other piece-wise linear functions
- Key SMT idea: handle ReLUs *lazily*
  - As opposed to eager case splitting
  - *Defer* splitting for as long as possible
  - May not have to split at all!
Reluplex (cnt’d)

- SMT-solver for quantifier-free linear real arithmetic + ReLUs
- Based on the *Simplex* method for linear programming
  - Simplex + ReLUs = Reluplex
  - Applicable to other piece-wise linear functions
- Key SMT idea: handle ReLUs *lazily*
  - As opposed to eager case splitting
  - *Defer* splitting for as long as possible
  - May not have to split at all!
- But first, an introduction to Simplex
Simplex

Developed shortly after WW2 by George Dantzig

An algorithm for solving linear programs

Linear equations
Variable bounds
Objective function

Very efficient, still in use today
Simplex

- Developed shortly after WW2 by George Dantzig
Simplex

- Developed shortly after WW2 by George Dantzig

- An algorithm for solving linear programs
Simplex

- Developed shortly after WW2 by George Dantzig
- An algorithm for solving linear programs
  - Linear equations
Simplex

- Developed shortly after WW2 by George Dantzig

- An algorithm for solving linear programs
  - Linear equations
  - Variable bounds

Very efficient, still in use today
Developed shortly after WW2 by George Dantzig

An algorithm for solving linear programs

- Linear equations
- Variable bounds
- Objective function
Simplex

- Developed shortly after WW2 by George Dantzig
- An algorithm for solving linear programs
  - Linear equations
  - Variable bounds
  - Objective function
- Very efficient, still in use today
Simplex (cnt’d)

Divided into two phases:

1. Find a feasible solution
2. Optimize with respect to objective function

We focus on phase 1, which is just a satisfiability check.
Simplex (cnt’d)

- Divided into two phases:
Simplex (cnt’d)

- Divided into two phases:
  1. Find a feasible solution
Simplex (cnt’d)

Divided into two phases:

1. Find a feasible solution
2. Optimize with respect to objective function
Simplex (cnt’d)

Divided into two phases:
1. Find a feasible solution
2. Optimize with respect to objective function

We focus on phase 1, which is just a *satisfiability check*
Simplex: Phase 1

Iterative algorithm
Always maintain a variable assignment
Assignment always satisfies equations
But may violate bounds
In every iteration, attempt to reduce the overall infeasibility
Simplex: Phase 1

- Iterative algorithm
Simplex: Phase 1

- Iterative algorithm
- Always maintain a *variable assignment*
Simplex: Phase 1

- Iterative algorithm
- Always maintain a *variable assignment*
- Assignment always *satisfies equations*
Simplex: Phase 1

- Iterative algorithm
- Always maintain a *variable assignment*
- Assignment always *satisfies equations*
  - But may *violate bounds*
Simplex: Phase 1

- Iterative algorithm
- Always maintain a *variable assignment*
- Assignment always *satisfies equations*
  - But may *violate bounds*
- In every iteration, attempt to reduce the overall *infeasibility*
Simplex: Basics and Non-Basics

Variables partitioned into basic and non-basic variables.

Non-basics are "free". Basics are "bounded".

Non-basic assignment dictates basic assignment. This is how the equations are maintained.

In every iteration, we can perform:
1. an update: change the assignment of a non-basic variable and any affected basics
2. a pivot: switch a basic and non-basic variable.

Guy Katz (HUJI)
Verification of ML
FoPSS 2018
Variables partitioned into *basic* and *non-basic* variables.
Simplex: Basics and Non-Basics

- Variables partitioned into *basic* and *non-basic* variables
  - Non-basics are “free”
Variables partitioned into *basic* and *non-basic* variables

- Non-basics are “free”
- Basics are “bounded”
Variables partitioned into **basic** and **non-basic** variables
- Non-basics are “free”
- Basics are “bounded”

Non-basic assignment dictates basic assignment
Simplex: Basics and Non-Basics

- Variables partitioned into *basic* and *non-basic* variables
  - Non-basics are “free”
  - Basics are “bounded”

- Non-basic assignment dictates basic assignment
  - This is how the equations are maintained
Variables partitioned into *basic* and *non-basic* variables

- Non-basics are “free”
- Basics are “bounded”

Non-basic assignment dictates basic assignment

- This is how the equations are maintained

In every iteration, we can perform
Simplex: Basics and Non-Basics

- Variables partitioned into `basic` and `non-basic` variables
  - Non-basics are “free”
  - Basics are “bounded”

- Non-basic assignment dictates basic assignment
  - This is how the equations are maintained

- In every iteration, we can perform
  1. an `update`: change the assignment of a non-basic variable
Simplex: Basics and Non-Basics

- Variables partitioned into \textit{basic} and \textit{non-basic} variables
  - Non-basics are “free”
  - Basics are “bounded”

- Non-basic assignment dictates basic assignment
  - This is how the equations are maintained

- In every iteration, we can perform
  1. an \textit{update}: change the assignment of a non-basic variable
     - and any affected basics
Simplex: Basics and Non-Basics

- Variables partitioned into *basic* and *non-basic* variables
  - Non-basics are “free”
  - Basics are “bounded”

- Non-basic assignment dictates basic assignment
  - This is how the equations are maintained

- In every iteration, we can perform
  1. an *update*: change the assignment of a non-basic variable
     - and any affected basics
  2. a *pivot*: switch a basic and non-basic variable
Simplex: Example

\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]

\[ 1 - 1 \]

Hidden layer
Input layer
Output layer

No activation functions

Property being checked: for \( x_1 \in [0, 1] \), always \( x_4 \in [0.5, 1] \)

Negated output property: \( x_1 \in [0, 1] \) and \( x_4 \in [0.5, 1] \)
Simplex: Example

Input layer → Hidden layer → Output layer

$x_1$ → $x_2$ → $x_4$

$x_1$ ∈ [0, 1], always $x_4$ ∈ [0.5, 1]

Negated output property: $x_1$ ∈ [0, 1] and $x_4$ ∈ [0.5, 1]
Simplex: Example

- No activation functions
Simplex: Example

- No activation functions
- Property being checked: for \( x_1 \in [0, 1] \), always \( x_4 \notin [0.5, 1] \)
Simplex: Example

- No activation functions
- Property being checked: for $x_1 \in [0, 1]$, always $x_4 \notin [0.5, 1]$
  - Negated output property: $x_1 \in [0, 1]$ and $x_4 \in [0.5, 1]$
Equations for weighted sums:

\[ x_2 - x_1 = 0 \]
\[ x_3 + x_1 = 0 \]
\[ x_4 - x_3 - x_2 = 0 \]

Bounds:

\[ x_1 \in [0, 1] \]
\[ x_4 \in [0, 0.5, 1] \]
\[ x_2, x_3 \text{ unbounded} \]

Technicality: replace constants by auxiliary variables
Simplex: Example (cnt’d)

Equations for weighted sums:

\[ x_2 - x_1 = 0 \]
\[ x_3 + x_1 = 0 \]
\[ x_4 - x_3 - x_2 = 0 \]

Bounds:

\[ x_1 \in [0, 1] \]
\[ x_4 \in [0.5, 1] \]
\[ x_2, x_3 \text{ unbounded} \]

Technicality: replace constants by auxiliary variables.

Guy Katz (HUJI)
Verification of ML
FoPSS 2018 67 / 115
Simplex: Example (cnt’d)

- Equations for weighted sums:

```
\begin{align*}
    x_2 - x_1 &= 0 \\
    x_3 + x_1 &= 0 \\
    x_4 - x_3 - x_2 &= 0
\end{align*}
```

Bounds:
- $x_1 \in [0, 1]$
- $x_4 \in [0.5, 1]$
- $x_2, x_3$ unbounded
Equations for weighted sums:

\[ x_2 - x_1 = 0 \]
\[ x_3 + x_1 = 0 \]
\[ x_4 - x_3 - x_2 = 0 \]
Equations for weighted sums:

\[ x_2 - x_1 = 0 \]
\[ x_3 + x_1 = 0 \]
\[ x_4 - x_3 - x_2 = 0 \]

Bounds:
Equations for weighted sums:

\[ x_2 - x_1 = 0 \]
\[ x_3 + x_1 = 0 \]
\[ x_4 - x_3 - x_2 = 0 \]

Bounds:

\[ x_1 \in [0, 1] \]
\[ x_4 \in [0.5, 1] \]
\[ x_2, x_3 \text{ unbounded} \]
Equations for weighted sums:

\[ x_2 - x_1 = 0 \]
\[ x_3 + x_1 = 0 \]
\[ x_4 - x_3 - x_2 = 0 \]

Bounds:

\[ x_1 \in [0, 1] \]
\[ x_4 \in [0.5, 1] \]
\[ x_2, x_3 \text{ unbounded} \]

Technicality: replace constants by *auxiliary* variables
Equations for weighted sums:

\[ x_2 - x_1 = 0 \]
\[ x_3 + x_1 = 0 \]
\[ x_4 - x_3 - x_2 = 0 \]

Bounds:

\[ x_1 \in [0, 1] \]
\[ x_4 \in [0.5, 1] \]
\[ x_2, x_3 \text{ unbounded} \]
\[ x_5, x_6, x_7 \in [0, 0] \]

Technicality: replace constants by *auxiliary* variables
Simplex: Example (cnt’d)

- Equations for weighted sums:
  \[ x_2 - x_1 = x_5 \]
  \[ x_3 + x_1 = x_6 \]
  \[ x_4 - x_3 - x_2 = x_7 \]

- Bounds:
  \[ x_1 \in [0, 1] \]
  \[ x_4 \in [0.5, 1] \]
  \[ x_2, x_3 \text{ unbounded} \]
  \[ x_5, x_6, x_7 \in [0, 0] \]

- Technicality: replace constants by *auxiliary* variables
Simplex: Example (cnt’d)

\[ x_5 = x_2 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_7 = x_4 - x_3 - x_2 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ x_5 = x_2 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_7 = x_4 - x_3 - x_2 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
\[
x_5 = x_2 - x_1 \\
x_6 = x_3 + x_1 \\
x_7 = x_4 - x_3 - x_2
\]

**Update:**  
\[
x_4 := x_4 + 0.5
\]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_2 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_7 = x_4 - x_3 - x_2 \]

Update:
\[ x_4 := x_4 + 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_2 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_7 = x_4 - x_3 - x_2 \]

Update:
\[ x_4 := x_4 + 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
\( x_5 = x_2 - x_1 \)
\( x_6 = x_3 + x_1 \)
\( x_7 = x_4 - x_3 - x_2 \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_2 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_7 = x_4 - x_3 - x_2 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ x_5 = x_2 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_7 = x_4 - x_3 - x_2 \]

**Pivot:** \( x_7, x_2 \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_2 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_7 = x_4 - x_3 - x_2 \]

← \[ x_2 = x_4 - x_3 - x_7 \]

Pivot: \(x_7, x_2\)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[
x_5 = x_2 - x_1 \quad \leftarrow \quad x_5 = x_4 - x_3 - x_7 - x_1
\]
\[
x_6 = x_3 + x_1
\]
\[
x_7 = x_4 - x_3 - x_2 \quad \leftarrow \quad x_2 = x_4 - x_3 - x_7
\]

Pivot: \(x_7, x_2\)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_4 - x_3 - x_7 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_2 = x_4 - x_3 - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_4 - x_3 - x_7 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_2 = x_4 - x_3 - x_7 \]

Update:
\[ x_7 := x_7 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_4 - x_3 - x_7 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_2 = x_4 - x_3 - x_7 \]

Update:
\[ x_7 := x_7 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_4 - x_3 - x_7 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_2 = x_4 - x_3 - x_7 \]

Update:
\[ x_7 := x_7 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_4 - x_3 - x_7 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_2 = x_4 - x_3 - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_4 - x_3 - x_7 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_2 = x_4 - x_3 - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_4 - x_3 - x_7 - x_1 \]
\[ x_6 = x_3 + x_1 \]
\[ x_2 = x_4 - x_3 - x_7 \]

Pivot: \( x_5, x_1 \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_3 )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_4 - x_3 - x_7 - x_1 \quad \leftarrow \quad x_1 = x_4 - x_3 - x_7 - x_5 \]

\[ x_6 = x_3 + x_1 \]

\[ x_2 = x_4 - x_3 - x_7 \]

Pivot: \( x_5, x_1 \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_5 = x_4 - x_3 - x_7 - x_1 \quad \leftarrow \quad x_1 = x_4 - x_3 - x_7 - x_5 \]
\[ x_6 = x_3 + x_1 \quad \leftarrow \quad x_6 = x_4 - x_7 - x_5 \]
\[ x_2 = x_4 - x_3 - x_7 \]

Pivot: \( x_5, x_1 \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_3 )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[x_1 = x_4 - x_3 - x_7 - x_5\]
\[x_6 = x_4 - x_7 - x_5\]
\[x_2 = x_4 - x_3 - x_7\]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2)</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_1 = x_4 - x_3 - x_7 - x_5 \]
\[ x_6 = x_4 - x_7 - x_5 \]
\[ x_2 = x_4 - x_3 - x_7 \]

Update:
\[ x_5 := x_5 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2)</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_1 = x_4 - x_3 - x_7 - x_5 \]
\[ x_6 = x_4 - x_7 - x_5 \]
\[ x_2 = x_4 - x_3 - x_7 \]

Update:
\[ x_5 := x_5 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_1 = x_4 - x_3 - x_7 - x_5 \]
\[ x_6 = x_4 - x_7 - x_5 \]
\[ x_2 = x_4 - x_3 - x_7 \]

Update:
\[ x_5 := x_5 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_3 )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
\begin{align*}
x_1 &= x_4 - x_3 - x_7 - x_5 \\
x_6 &= x_4 - x_7 - x_5 \\
x_2 &= x_4 - x_3 - x_7 \\
\end{align*}

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>(x_2)</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x_3)</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[
x_1 = x_4 - x_3 - x_7 - x_5 \\
x_6 = x_4 - x_7 - x_5 \\
x_2 = x_4 - x_3 - x_7
\]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>(x_2)</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x_3)</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Simplex: Example (cnt’d)

\[ x_1 = x_4 - x_3 - x_7 - x_5 \]
\[ x_6 = x_4 - x_7 - x_5 \]
\[ x_2 = x_4 - x_3 - x_7 \]

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
<th>Lower B.</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>0.5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(x_2)</td>
<td>0.5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(x_3)</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(x_4)</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(x_6)</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The Simplex Calculus

A simplex configuration:

- **Distinguished symbols**
  - SAT or UNSAT
- Or a tuple $⟨B, T, l, u, α⟩$
  - $B$: set of basic variables
  - $T$: a set of equations
  - $l, u$: lower and upper bounds
  - $α$: an assignment function from variables to reals

For notation:

- Slack $+$ on $x_i$:
  - $\{x_j / \in B | T_{i,j} > 0 \land α(x_j) < u(x_j) \lor T_{i,j} < 0 \land α(x_j) > l(x_j)\}$

- Slack $-$ on $x_i$:
  - $\{x_j / \in B | T_{i,j} < 0 \land α(x_j) < u(x_j) \lor T_{i,j} > 0 \land α(x_j) > l(x_j)\}$
The Simplex Calculus

- A simplex configuration:
A simplex configuration:

- Distinguished symbols SAT or UNSAT
The Simplex Calculus

- A simplex configuration:
  - Distinguished symbols SAT or UNSAT
  - Or a tuple $\langle B, T, l, u, \alpha \rangle$, where:
    - $B$: set of basic variables
    - $T$: a set of equations
    - $l, u$: lower and upper bounds
    - $\alpha$: an assignment function from variables to reals

For notation:
- $\text{slack} + (x_i) =$ \{ $x_j / \in B | (T_{i,j} > 0 \land \alpha(x_j) < u(x_j)) \lor (T_{i,j} < 0 \land \alpha(x_j) > l(x_j))$\}
- $\text{slack} - (x_i) =$ \{ $x_j / \in B | (T_{i,j} < 0 \land \alpha(x_j) < u(x_j)) \lor (T_{i,j} > 0 \land \alpha(x_j) > l(x_j))$\}
A simplex configuration:

- Distinguished symbols SAT or UNSAT
- Or a tuple $\langle B, T, l, u, \alpha \rangle$, where:
  - $B$: set of basic variables
A simplex configuration:

- Distinguished symbols SAT or UNSAT
- Or a tuple $\langle B, T, l, u, \alpha \rangle$, where:
  - $B$: set of basic variables
  - $T$: a set of equations
A simplex configuration:

- Distinguished symbols SAT or UNSAT
- Or a tuple \( \langle B, T, l, u, \alpha \rangle \), where:
  - \( B \): set of basic variables
  - \( T \): a set of equations
  - \( l, u \): lower and upper bounds
A simplex configuration:

- Distinguished symbols SAT or UNSAT
- Or a tuple $\langle B, T, l, u, \alpha \rangle$, where:
  - $B$: set of basic variables
  - $T$: a set of equations
  - $l, u$: lower and upper bounds
  - $\alpha$: an assignment function from variables to reals
A simplex configuration:

- Distinguished symbols SAT or UNSAT
- Or a tuple $\langle B, T, l, u, \alpha \rangle$, where:
  - $B$: set of basic variables
  - $T$: a set of equations
  - $l, u$: lower and upper bounds
  - $\alpha$: an assignment function from variables to reals

For notation:
The Simplex Calculus

- A simplex configuration:
  - Distinguished symbols SAT or UNSAT
  - Or a tuple \( \langle B, T, l, u, \alpha \rangle \), where:
    - \( B \): set of basic variables
    - \( T \): a set of equations
    - \( l, u \): lower and upper bounds
    - \( \alpha \): an assignment function from variables to reals

- For notation:

\[
\text{slack}^+(x_i) = \{ x_j \notin B \mid (T_{i,j} > 0 \land \alpha(x_j) < u(x_j)) \lor (T_{i,j} < 0 \land \alpha(x_j) > l(x_j)) \}
\]

\[
\text{slack}^-(x_i) = \{ x_j \notin B \mid (T_{i,j} < 0 \land \alpha(x_j) < u(x_j)) \lor (T_{i,j} > 0 \land \alpha(x_j) > l(x_j)) \}
\]
The Simplex Calculus (cnt’d)

Pivot

1 \ x_i \in B, \ \alpha(x_i) < \ x_i, \ x_j \in \text{slack} + \ x_i \ T := \text{pivot}(T, i, j), \ B := B \cup \{x_j\} \setminus \{x_i\}

Pivot

2 \ x_i \in B, \ \alpha(x_i) > u(x_i), \ x_j \in \text{slack} - \ x_i \ T := \text{pivot}(T, i, j), \ B := B \cup \{x_j\} \setminus \{x_i\}

Update

\ x_j/ \in B, \ \alpha(x_j) < l(x_j) \lor \alpha(x_j) > u(x_j), \ l(x_j) \leq \alpha(x_j) + \delta \leq u(x_j)

\alpha := \text{update}(\alpha, x_j, \delta)

Failure

\ x_i \in B, \ \alpha(x_i) < l(x_i) \land \text{slack} + \ x_i = \emptyset \lor \alpha(x_i) > u(x_i) \land \text{slack} - \ x_i = \emptyset

UNSAT

Success

\forall x_i \in X. \ l(x_i) \leq \alpha(x_i) \leq u(x_i)

SAT
The Simplex Calculus (cnt’d)

\[
Pivot_1: \quad \frac{x_i \in B, \ \alpha(x_i) < l(x_i), \ x_j \in \text{slack}^+(x_i)}{T := pivot(T, i, j), \ B := B \cup \{x_j\} \setminus \{x_i\}}
\]
The Simplex Calculus (cnt’d)

\[\begin{align*}
\text{Pivot}_1 & \quad x_i \in \mathcal{B}, \quad \alpha(x_i) < l(x_i), \quad x_j \in \text{slack}^+(x_i) \\
T & := \text{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}
\end{align*}\]

\[\begin{align*}
\text{Pivot}_2 & \quad x_i \in \mathcal{B}, \quad \alpha(x_i) > u(x_i), \quad x_j \in \text{slack}^-(x_i) \\
T & := \text{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}
\end{align*}\]
### The Simplex Calculus (cnt’d)

<table>
<thead>
<tr>
<th>Case</th>
<th>Expression</th>
</tr>
</thead>
</table>
| **Pivot 1** | $x_i \in B$, $\alpha(x_i) < l(x_i)$, $x_j \in \text{slack}^+(x_i)$  
$T := \text{pivot}(T, i, j)$, $B := B \cup \{x_j\} \setminus \{x_i\}$ |
| **Pivot 2** | $x_i \in B$, $\alpha(x_i) > u(x_i)$, $x_j \in \text{slack}^-(x_i)$  
$T := \text{pivot}(T, i, j)$, $B := B \cup \{x_j\} \setminus \{x_i\}$ |
| **Update** | $x_j \notin B$, $\alpha(x_j) < l(x_j) \lor \alpha(x_j) > u(x_j)$, $l(x_j) \leq \alpha(x_j) + \delta \leq u(x_j)$  
$\alpha := \text{update}(\alpha, x_j, \delta)$ |

**Failure**

$x_i \in B$, $(\alpha(x_i) < l(x_i) \land \text{slack}^+(x_i) = \emptyset) \lor (\alpha(x_i) > u(x_i) \land \text{slack}^-(x_i) = \emptyset)$
The Simplex Calculus (cnt’d)

Pivot_1  \[ x_i \in \mathcal{B}, \ \alpha(x_i) < l(x_i), \ x_j \in \text{slack}^+(x_i) \]
\[ T := \text{pivot}(T, i, j), \ \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\} \]

Pivot_2  \[ x_i \in \mathcal{B}, \ \alpha(x_i) > u(x_i), \ x_j \in \text{slack}^-(x_i) \]
\[ T := \text{pivot}(T, i, j), \ \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\} \]

Update  \[ x_j \notin \mathcal{B}, \ \alpha(x_j) < l(x_j) \lor \alpha(x_j) > u(x_j), \ l(x_j) \leq \alpha(x_j) + \delta \leq u(x_j) \]
\[ \alpha := \text{update}(\alpha, x_j, \delta) \]

Failure  \[ x_i \in \mathcal{B}, \ (\alpha(x_i) < l(x_i) \land \text{slack}^+(x_i) = \emptyset) \lor (\alpha(x_i) > u(x_i) \land \text{slack}^-(x_i) = \emptyset) \]
\[ \text{UNSAT} \]
The Simplex Calculus (cnt’d)

**Pivot 1**
\[ x_i \in B, \quad \alpha(x_i) \leq l(x_i), \quad x_j \in \text{slack}^+(x_i) \]
\[ T := \text{pivot}(T, i, j), \quad B := B \cup \{x_j\} \setminus \{x_i\} \]

**Pivot 2**
\[ x_i \in B, \quad \alpha(x_i) > u(x_i), \quad x_j \in \text{slack}^-(x_i) \]
\[ T := \text{pivot}(T, i, j), \quad B := B \cup \{x_j\} \setminus \{x_i\} \]

**Update**
\[ x_j \notin B, \quad \alpha(x_j) < l(x_j) \lor \alpha(x_j) > u(x_j), \quad l(x_j) \leq \alpha(x_j) + \delta \leq u(x_j) \]
\[ \alpha := \text{update}(\alpha, x_j, \delta) \]

**Failure**
\[ x_i \in B, \quad \alpha(x_i) < l(x_i) \land \text{slack}^+(x_i) = \emptyset \lor \alpha(x_i) > u(x_i) \land \text{slack}^-(x_i) = \emptyset \]
\[ \text{UNSAT} \]

**Success**
\[ \forall x_i \in X. \quad l(x_i) \leq \alpha(x_i) \leq u(x_i) \]
\[ \text{SAT} \]
Properties of Simplex

Theorem (Soundness and Completeness of Simplex)
The simplex algorithm is sound and complete*

Soundness:
- SAT $\implies$ assignment is correct
- UNSAT $\implies$ no assignment exists

Completeness: depends on variable selection strategy
- Bland's rule: guarantees termination
  - Always pick variables with smallest index
  - Prevents cycling
  - But unfortunately quite slow
- Better selection strategies exist (e.g., steepest edge)

Problem is in $\mathcal{P}$, unknown whether simplex is in $\mathcal{P}$
Properties of Simplex

Theorem (Soundness and Completeness of Simplex)

*The simplex algorithm is sound and complete*

- **Soundness:**
  - SAT \( \Rightarrow \) assignment is correct
  - UNSAT \( \Rightarrow \) no assignment exists

- **Completeness:** depends on variable selection strategy
  - Bland’s rule: guarantees termination
    - Always pick variables with smallest index
    - Prevents cycling
  - But unfortunately quite slow
  - Better selection strategies exist (e.g., steepest edge)

Problem is in \( P \), unknown whether simplex is in \( P \)
Properties of Simplex

Theorem (Soundness and Completeness of Simplex)

The simplex algorithm is sound and complete*

- Soundness:
Properties of Simplex

Theorem (Soundness and Completeness of Simplex)

*The simplex algorithm is sound and complete*

- **Soundness:**
  - $\text{SAT} \Rightarrow \text{assignment is correct}$

- **Completeness:** depends on variable selection strategy
  - **Bland's rule:** guarantees termination
    - Always pick variables with smallest index
    - Prevents cycling
    - But unfortunately quite slow
  - Better selection strategies exist (e.g., steepest edge)
  - Problem is in $\text{P}$, unknown whether simplex is in $\text{P}$
Theorem (Soundness and Completeness of Simplex)

The simplex algorithm is sound and complete.*

- **Soundness:**
  - SAT $\Rightarrow$ assignment is correct
  - UNSAT $\Rightarrow$ no assignment exists

- **Completeness:** depends on variable selection strategy
  - Bland's rule: guarantees termination
    - Always pick variables with smallest index
    - Prevents cycling
  - But unfortunately quite slow
  - Better selection strategies exist (e.g., steepest edge)

Problem is in $P$, unknown whether simplex is in $P$
Theorem (Soundness and Completeness of Simplex)

The simplex algorithm is sound and complete*

- **Soundness:**
  - \( \text{SAT} \Rightarrow \text{assignment is correct} \)
  - \( \text{UNSAT} \Rightarrow \text{no assignment exists} \)

- **Completeness:** depends on variable selection strategy
  - Bland's rule: guarantees termination
    - Always pick variables with smallest index
    - Prevents cycling
  - But unfortunately quite slow
  - Better selection strategies exist (e.g., steepest edge)
  - Problem is in \( \mathcal{P} \), unknown whether simplex is in \( \mathcal{P} \)
Properties of Simplex

**Theorem (Soundness and Completeness of Simplex)**

*The simplex algorithm is sound and complete*\

- **Soundness:**
  - SAT $\Rightarrow$ assignment is correct
  - UNSAT $\Rightarrow$ no assignment exists

- **Completeness:** depends on variable selection strategy
  - *Bland’s rule:* guarantees termination
Properties of Simplex

Theorem (Soundness and Completeness of Simplex)

The simplex algorithm is sound and complete*

- Soundness:
  - SAT ⇒ assignment is correct
  - UNSAT ⇒ no assignment exists

- Completeness: depends on variable selection strategy
  - Bland’s rule: guarantees termination
    - Always pick variables with smallest index
Theorem (Soundness and Completeness of Simplex)

*The simplex algorithm is sound and complete*

- **Soundness:**
  - SAT $\Rightarrow$ assignment is correct
  - UNSAT $\Rightarrow$ no assignment exists

- **Completeness:** depends on variable selection strategy

- **Bland’s rule:** guarantees termination
  - Always pick variables with smallest index
  - Prevents cycling

Guy Katz (HUJI)
Verification of ML
FoS 2018
Properties of Simplex

**Theorem (Soundness and Completeness of Simplex)**

*The simplex algorithm is sound and complete*

- **Soundness:**
  - SAT $\Rightarrow$ assignment is correct
  - UNSAT $\Rightarrow$ no assignment exists

- **Completeness:** depends on variable selection strategy
  - **Bland’s rule:** guarantees termination
  - Always pick variables with smallest index
  - Prevents cycling
  - But unfortunately quite slow
Properties of Simplex

**Theorem (Soundness and Completeness of Simplex)**

*The simplex algorithm is sound and complete*

- **Soundness:**
  - SAT ⇒ assignment is correct
  - UNSAT ⇒ no assignment exists

- **Completeness:** depends on variable selection strategy

- **Bland’s rule:** guarantees termination
  - Always pick variables with smallest index
  - Prevents cycling
  - But unfortunately quite slow

- Better selection strategies exist (e.g., *steepest edge*)
Properties of Simplex

**Theorem (Soundness and Completeness of Simplex)**

The simplex algorithm is sound and complete*

- **Soundness:**
  - SAT ⇒ assignment is correct
  - UNSAT ⇒ no assignment exists

- **Completeness:** depends on variable selection strategy

- **Bland’s rule:** guarantees termination
  - Always pick variables with smallest index
  - Prevents cycling
  - But unfortunately quite slow

- Better selection strategies exist (e.g., *steepest edge*)

- Problem is in P, unknown whether simplex is in P
From Simplex to Reluplex

Each ReLU node $x$ represented as two variables: $x$ to represent the (input) weighted sum $xw$ to represent the (output) activation result $xw$ and $xa$ change independently

May violate ReLU constraint

Similar to bound constraints

Fix incrementally

Use pivots and updates, same as before
Each ReLU node $x$ represented as two variables:
Each ReLU node $x$ represented as two variables:
- $x^w$ to represent the (input) *weighted sum*
Each ReLU node $x$ represented as two variables:
- $x^w$ to represent the (input) \textit{weighted sum}
- $x^a$ to represent the (output) \textit{activation result}
Each ReLU node $x$ represented as two variables:

- $x^w$ to represent the (input) weighted sum
- $x^a$ to represent the (output) activation result

$x^w$ and $x^a$ change independently
From Simplex to Reluplex

- Each ReLU node \( x \) represented as two variables:
  - \( x^w \) to represent the (input) *weighted sum*
  - \( x^a \) to represent the (output) *activation result*

- \( x^w \) and \( x^a \) change independently
  - May violate ReLU constraint
Each ReLU node $x$ represented as two variables:
- $x^w$ to represent the (input) *weighted sum*
- $x^a$ to represent the (output) *activation result*

$x^w$ and $x^a$ change independently
- May violate ReLU constraint
- Similar to bound constraints
From Simplex to Reluplex

- Each ReLU node $x$ represented as two variables:
  - $x^w$ to represent the (input) \textit{weighted sum}
  - $x^a$ to represent the (output) \textit{activation result}

- $x^w$ and $x^a$ change independently
  - May violate ReLU constraint
  - Similar to bound constraints
  - Fix \textit{incrementally}
From Simplex to Reluplex

- Each ReLU node $x$ represented as two variables:
  - $x^w$ to represent the (input) weighted sum
  - $x^a$ to represent the (output) activation result

- $x^w$ and $x^a$ change independently
  - May violate ReLU constraint
  - Similar to bound constraints
  - Fix *incrementally*

- Use pivots and updates, same as before
Reluplex: Example

\[ x_1 \rightarrow x_2 \rightarrow x_4 \]
\[ x_1 \rightarrow x_3 \rightarrow x_4 \]
\[ x_2 \rightarrow x_4 \]
\[ x_3 \rightarrow x_4 \]

ReLU

Guy Katz (HUJI)
Reluplex: Example
Reluplex: Example (cnt’d)
Equations for weighted sums:

\[ x_5 = x_w^2 - x_1 \]
\[ x_6 = x_w^3 + x_1 \]
\[ x_7 = x_4 - x_a^3 - x_a^2 \]

Bounds:

\[ x_1 \in [0, 1] \]
\[ x_4 \in [0, 0.5, 1] \]
\[ x_w^2, x_w^3 \text{ unbounded} \]
\[ x_a^2, x_a^3 \in [0, \infty) \]
\[ x_5, x_6, x_7 \in [0, 0] \]
Equations for weighted sums:
Equations for weighted sums:

\[
\begin{align*}
    x_5 &= x_2^w - x_1 \\
    x_6 &= x_3^w + x_1 \\
    x_7 &= x_4 - x_3^a - x_2^a
\end{align*}
\]
Reluplex: Example (cnt’d)

- Equations for weighted sums:
  - $x_5 = x_2^w - x_1$
  - $x_6 = x_3^w + x_1$
  - $x_7 = x_4 - x_3^a - x_2^a$

- Bounds:
Reluplex: Example (cnt’d)

Equations for weighted sums:

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_7 = x_4 - x_3^a - x_2^a \]

Bounds:

\[ x_1 \in [0, 1] \]
\[ x_4 \in [0.5, 1] \]
\[ x_2^w, x_3^w \text{ unbounded} \]
\[ x_2^a, x_3^a \in [0, \infty) \]
\[ x_5, x_6, x_7 \in [0, 0] \]
Reluplex: Example (cnt’d)

$x_5 = x_w^2 - x_1$

$x_6 = x_w^3 + x_1$

$x_7 = x_4 - x_{a_3} - x_{a_2}$

Update: $x_4 := x_4 + 0.5$.

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$x_w^2$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$x_a^2$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$x_w^3$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$x_4$</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>$x_5$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$x_6$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$x_7$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Reluplex: Example (cnt’d)**

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_7 = x_4 - x_3^a - x_2^a \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_7 = x_4 - x_3^a - x_2^a \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
$x_5 = x_2^w - x_1$

$x_6 = x_3^w + x_1$

$x_7 = x_4 - x_3^a - x_2^a$

**Update:**

$x_4 := x_4 + 0.5$

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x_1$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$x_2^w$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$x_2^a$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$x_3^w$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$x_3^a$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>$x_4$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$x_5$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>$x_6$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$x_7$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_7 = x_4^a - x_3^a - x_2^a \]

Update:
\[ x_4 := x_4 + 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_7 = x_4 - x_3^a - x_2^a \]

Update:
\[ x_4 := x_4 + 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x^w_2 - x_1 \]
\[ x_6 = x^w_3 + x_1 \]
\[ x_7 = x_4 - x^a_3 - x^a_2 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x^w_2 )</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x^a_2 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( x^w_3 )</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x^a_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[x_5 = x_2^w - x_1\]
\[x_6 = x_3^w + x_1\]
\[x_7 = x_4 - x_3^a - x_2^a\]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2^w)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>(x_2^a)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3^w)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>(x_3^a)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_7 = x_4 - x_3^a - x_2^a \]

Pivot: \( x_7, x_2^a \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>( x_5 )</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>( x_6 )</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_7 = x_4 - x_3^a - x_2^a \]

Pivot: \( x_7, x_2^a \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>

Guy Katz (HUJI)
Verification of ML
FoPSS 2018 78 / 115
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

**Pivot:** \( x_7, x_2^a \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

**Update:**
\[ x_7 := x_7 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_2^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

**Update:**
\[ x_7 := x_7 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Update:
\[ x_7 := x_7 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Update:
\[ x_2^w := x_2^w + 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Update:
\[ x_2^w := x_2^w + 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_2^w )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( x_4 )</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Update:
\[ x_2^w := x_2^w + 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2^w)</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>(x_2^a)</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3^w)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>(x_3^a)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x_2^w)</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>(x_2^a)</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3^w)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>(x_3^a)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Pivot: \( x_5, x_1 \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_5 = x_2^w - x_1 \]
\[ x_6 = x_3^w + x_1 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Pivot: \( x_5, x_1 \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x_2^w - x_5 \]
\[ x_6 = x_3^w + x_2^w - x_5 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Pivot: \( x_5, x_1 \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ x_1 = x_2^w - x_5 \]
\[ x_6 = x_3^w + x_2^w - x_5 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_2^w )</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x^w_2 - x_5 \]
\[ x_6 = x^w_3 + x^w_2 - x_5 \]
\[ x^a_2 = x_4 - x^a_3 - x_7 \]

Update:
\[ x_5 := x_5 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x^w_2 )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x^a_2 )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x^w_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x^a_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x_2^w - x_5 \]
\[ x_6 = x_3^w + x_2^w - x_5 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Update:
\[ x_5 := x_5 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(x_1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_2^w)</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>(x_2^a)</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3^w)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x_3^a)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ x_1 = x_2^w - x_5 \]
\[ x_6 = x_3^w + x_2^w - x_5 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

**Update:**
\[ x_5 := x_5 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>x_2^w</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>x_3^w</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x^w_2 - x_5 \]
\[ x_6 = x^w_3 + x^w_2 - x_5 \]
\[ x^a_2 = x_4 - x^a_3 - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x^w_2 )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x^a_2 )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x^w_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x^a_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[
\begin{align*}
    x_1 &= x^w_2 - x_5 \\
    x_6 &= x^w_3 + x^w_2 - x_5 \\
    x^a_2 &= x_4 - x^a_3 - x_7
\end{align*}
\]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(x^w_2)</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>(x^a_2)</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x^w_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x^a_3)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>(x_4)</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_6)</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>(x_7)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x_2^w - x_5 \]
\[ x_6 = x_3^w + x_2^w - x_5 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Pivot: \( x_6, x_3^w \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>( x_2^w )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>( x_3^w )</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x_2^w - x_5 \]
\[ x_6 = x_3^w + x_2^w - x_5 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Pivot: \( x_6, x_3^w \)

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>( x_2^w )</td>
<td></td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>( x_3^w )</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x_2^w - x_5 \]
\[ x_3^w = x_6 - x_2^w + x_5 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x_2^w - x_5 \]
\[ x_3^w = x_6 - x_2^w + x_5 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Update:
\[ x_6 := x_6 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
$x_1 = x^w_2 - x_5$

$x^w_3 = x_6 - x^w_2 + x_5$

$x^a_2 = x_4 - x^a_3 - x_7$

**Update:**

$x_6 := x_6 - 0.5$

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x_1$</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>$x^w_2$</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$x^a_2$</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>$x^w_3$</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$x^a_3$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>$x_4$</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>$x_5$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>$x_6$</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>$x_7$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x_2^w - x_5 \]
\[ x_3^w = x_6 - x_2^w + x_5 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Update:
\[ x_6 := x_6 - 0.5 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>( x_2^w )</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>( x_3^w )</td>
<td>-0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x^w_2 - x_5 \]
\[ x^w_3 = x_6 - x^w_2 + x_5 \]
\[ x^a_2 = x_4 - x^a_3 - x_7 \]

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>( x^w_2 )</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x^a_2 )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>( x^w_3 )</td>
<td>-0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x^a_3 )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Reluplex: Example (cnt’d)

\[ x_1 = x_2^w - x_5 \]
\[ x_3^w = x_6 - x_2^w + x_5 \]
\[ x_2^a = x_4 - x_3^a - x_7 \]

Success

<table>
<thead>
<tr>
<th>Lower B.</th>
<th>Var</th>
<th>Value</th>
<th>Upper B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( x_2^w )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_2^a )</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_3^w )</td>
<td>-0.5</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>( x_3^a )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>( x_4 )</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>( x_5 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( x_7 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
ReLU Property:

\[ x_1 \in [0, 1) \text{ and } x_4 \in \left[0.5, 1\right) \]
Property: $x_1 \in [0, 1]$ and $x_4 \in [0.5, 1]$
Property: \( x_1 \in [0, 1] \) and \( x_4 \in [0.5, 1] \)
The Reluplex Calculus

A Reluplex configuration:

- **Distinguished symbols**
- SAT or UNSAT

- Or a tuple $\langle B, T, l, u, \alpha, R \rangle$, where:
  - $B$: set of basic variables
  - $T$: a set of equations
  - $l, u$: lower and upper bounds
  - $\alpha$: an assignment function from variables to reals
  - $R \subset X \times X$ is a set of ReLU connections

Guy Katz (HUJI)
A Reluplex configuration:
A Reluplex configuration:
  - Distinguished symbols SAT or UNSAT
A Reluplex configuration:

- Distinguished symbols SAT or UNSAT
- Or a tuple \( \langle B, T, l, u, \alpha, R \rangle \), where:
  - \( B \): set of basic variables
  - \( T \): a set of equations
  - \( l, u \): lower and upper bounds
  - \( \alpha \): an assignment function from variables to reals
  - \( R \subset X \times X \) is a set of ReLU connections
A Reluplex configuration:
- Distinguished symbols SAT or UNSAT
- Or a tuple $\langle B, T, l, u, \alpha, R \rangle$, where:
  - $B$: set of basic variables
  - $T$: a set of equations
  - $l, u$: lower and upper bounds
  - $\alpha$: an assignment function from variables to reals
  - $R \subseteq X \times X$ is a set of ReLU connections
A Reluplex configuration:

- Distinguished symbols SAT or UNSAT
- Or a tuple $\langle B, T, l, u, \alpha, R \rangle$, where:
  - $B$: set of basic variables
  - $T$: a set of equations
A Reluplex configuration:

- Distinguished symbols SAT or UNSAT
- Or a tuple $\langle \mathcal{B}, T, l, u, \alpha, R \rangle$, where:
  - $\mathcal{B}$: set of basic variables
  - $T$: a set of equations
  - $l, u$: lower and upper bounds
A Reluplex configuration:

- Distinguished symbols SAT or UNSAT
- Or a tuple \( \langle B, T, l, u, \alpha, R \rangle \), where:
  - \( B \): set of basic variables
  - \( T \): a set of equations
  - \( l, u \): lower and upper bounds
  - \( \alpha \): an assignment function from variables to reals
  - \( R \subset X \times X \): a set of ReLU connections
A Reluplex configuration:

- Distinguished symbols SAT or UNSAT
- Or a tuple \( \langle B, T, l, u, \alpha, R \rangle \), where:
  - \( B \): set of basic variables
  - \( T \): a set of equations
  - \( l, u \): lower and upper bounds
  - \( \alpha \): an assignment function from variables to reals
  - \( R \subset \mathcal{X} \times \mathcal{X} \) is a set of ReLU connections
The Reluplex Calculus (cnt’d)

Pivot

1, Pivot 2, Update and Failure are as before

SAT iff at least one leaf of the derivation tree is SAT

Update \( w \langle x_i, x_j \rangle \in R, \alpha(x_j) \neq \max(0, \alpha(x_i)) \), \( \alpha(x_j) \geq 0 \)

\( \alpha := \text{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i)) \)

Update \( a \langle x_i, x_j \rangle \in B, \langle x_i, x_j \rangle \in R, \alpha(x_j) \neq \max(0, \alpha(x_i)) \)

\( \alpha := \text{update}(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j)) \)

Pivot

For Relu

\( x_i \in B, \exists x_l. \langle x_i, x_l \rangle \in R \lor \langle x_l, x_i \rangle \in R, x_j \not\in B, T_{i,j} \neq 0 \)

\( T := \text{pivot}(T, i, j) \), \( B := B \cup \{ x_j \} \setminus \{ x_i \} \)

Relu Split

\( \langle x_i, x_j \rangle \in R, l(x_i) < 0, u(x_i) > 0 \)

\( u(x_i) := 0 \), \( l(x_i) := 0 \)

Relu Success

\( \forall x \in X. l(x) \leq \alpha(x) \leq u(x), \forall \langle x_w, x_a \rangle \in R. \alpha(x_a) = \max(0, \alpha(x_w)) \)
Pivot\textsubscript{1}, Pivot\textsubscript{2}, Update and Failure are as before
The Reluplex Calculus (cnt’d)

- Pivot<sub>1</sub>, Pivot<sub>2</sub>, Update and Failure are as before
- SAT iff at least one leaf of the derivation tree is SAT
The Reluplex Calculus (cnt’d)

- Pivot\(_1\), Pivot\(_2\), Update and Failure are as before
- SAT iff at least one leaf of the derivation tree is SAT

\[
\text{Update}_w \quad \begin{array}{c}
x_i \notin B, \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i)), \quad \alpha(x_j) \geq 0 \\
\end{array}
\alpha := \text{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))
\]
Pivot₁, Pivot₂, Update and Failure are as before

SAT iff at least one leaf of the derivation tree is SAT

\[
\text{Update}_w \quad x_i \notin B, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i)), \quad \alpha(x_j) \geq 0
\]
\[\alpha := \text{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))\]

\[
\text{Update}_a \quad x_j \notin B, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i))
\]
\[\alpha := \text{update}(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))\]
Pivot₁, Pivot₂, Update and Failure are as before

SAT iff at least one leaf of the derivation tree is SAT

\[ \text{Update}_w \quad \frac{x_i \notin B, \langle x_i, x_j \rangle \in R, \alpha(x_j) \neq \max(0, \alpha(x_i)), \alpha(x_j) \geq 0}{\alpha := \text{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))} \]

\[ \text{Update}_a \quad \frac{x_j \notin B, \langle x_i, x_j \rangle \in R, \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := \text{update}(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))} \]

\[ \text{PivotForRelu} \quad \frac{x_i \in B, \exists x_l. \langle x_i, x_l \rangle \in R \lor \langle x_l, x_i \rangle \in R, x_j \notin B, T_{i,j} \neq 0}{T := \text{pivot}(T, i, j), B := B \cup \{x_j\} \setminus \{x_i\}} \]
Pivot₁, Pivot₂, Update and Failure are as before

SAT iff at least one leaf of the derivation tree is SAT

\[
\text{Update}_w \quad \frac{x_i \notin B, \langle x_i, x_j \rangle \in R, \alpha(x_j) \neq \max(0, \alpha(x_i)), \alpha(x_j) \geq 0}{\alpha := \text{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}
\]

\[
\text{Update}_a \quad \frac{x_j \notin B, \langle x_i, x_j \rangle \in R, \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := \text{update}(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}
\]

\[
\text{PivotForRelu} \quad \frac{x_i \in B, \exists x_l. \langle x_i, x_l \rangle \in R \lor \langle x_l, x_i \rangle \in R, x_j \notin B, T_{i,j} \neq 0}{T := \text{pivot}(T, i, j), B := B \cup \{x_j\} \setminus \{x_i\}}
\]

\[
\text{ReluSplit} \quad \frac{\langle x_i, x_j \rangle \in R, l(x_i) < 0, u(x_i) > 0}{u(x_i) := 0 \quad l(x_i) := 0}
\]
Pivot₁, Pivot₂, Update and Failure are as before

SAT iff at least one leaf of the derivation tree is SAT

\[
\text{Update}_w \quad \frac{x_i \notin B, \langle x_i, x_j \rangle \in R, \alpha(x_j) \neq \max(0, \alpha(x_i)), \alpha(x_j) \geq 0}{\alpha := \text{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}
\]

\[
\text{Update}_a \quad \frac{x_j \notin B, \langle x_i, x_j \rangle \in R, \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := \text{update}(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}
\]

\[
\text{PivotForRelu} \quad \frac{x_i \in B, \exists x_l. \langle x_i, x_l \rangle \in R \lor \langle x_l, x_i \rangle \in R, x_j \notin B, T_{i,j} \neq 0}{T := \text{pivot}(T, i, j), B := B \cup \{x_j\} \setminus \{x_i\}}
\]

\[
\text{ReluSplit} \quad \frac{\langle x_i, x_j \rangle \in R, l(x_i) < 0, u(x_i) > 0}{u(x_i) := 0 \quad l(x_i) := 0}
\]

\[
\text{ReluSuccess} \quad \forall x \in X. \ l(x) \leq \alpha(x) \leq u(x), \forall \langle x^w, x^a \rangle \in R. \alpha(x^a) = \max(0, \alpha(x^w)) \quad \text{SAT}
\]
Theorem (Soundness and Completeness of Reluplex)

The Reluplex algorithm is sound and complete*.

Soundness:
- SAT $\Rightarrow$ assignment is correct
- UNSAT $\Rightarrow$ no assignment exists

Completeness: depends on variable selection strategy and splitting strategy.

Naive approach: split on all variables immediately, apply Bland’s rule.

This is the case-splitting approach from before.

Ensures termination.
Properties of Reluplex

Theorem (Soundness and Completeness of Reluplex)

The Reluplex algorithm is sound and complete*

Soundness:
SAT ⇒ assignment is correct
UNSAT ⇒ no assignment exists

Completeness: depends on variable selection strategy and splitting strategy

Naive approach: split on all variables immediately, apply Bland's rule
This is the case-splitting approach from before
Ensures termination
Properties of Reluplex

Theorem (Soundness and Completeness of Reluplex)

*The Reluplex algorithm is sound and complete*

- **Soundness:**

  - SAT $\Rightarrow$ assignment is correct
  - UNSAT $\Rightarrow$ no assignment exists

- Completeness: depends on variable selection strategy and splitting strategy

  - Naive approach: split on all variables immediately, apply Bland's rule
  - This is the case-splitting approach from before

  Ensures termination
Properties of Reluplex

**Theorem (Soundness and Completeness of Reluplex)**

*The Reluplex algorithm is sound and complete*

- **Soundness:**
  - $\text{SAT} \implies \text{assignment is correct}$
- **Completeness:** depends on variable selection strategy and splitting strategy.

Naive approach: split on all variables immediately, apply Bland's rule.

This is the case-splitting approach from before.

Ensures termination.

---

Guy Katz (HUJI)

Verification of ML

FoPSS 2018 86 / 115
The Reluplex algorithm is sound and complete*

Soundness:

- SAT ⇒ assignment is correct
- UNSAT ⇒ no assignment exists
Properties of Reluplex

Theorem (Soundness and Completeness of Reluplex)

*The Reluplex algorithm is sound and complete*

- **Soundness:**
  - SAT $\Rightarrow$ assignment is correct
  - UNSAT $\Rightarrow$ no assignment exists

- **Completeness:** depends on *variable selection strategy* and *splitting strategy*
Theorem (Soundness and Completeness of Reluplex)

The Reluplex algorithm is sound and complete*

- **Soundness:**
  - SAT $\Rightarrow$ assignment is correct
  - UNSAT $\Rightarrow$ no assignment exists

- **Completeness:** depends on *variable selection strategy* and *splitting strategy*

- Naive approach: split on all variables immediately, apply Bland’s rule
Properties of Reluplex

Theorem (Soundness and Completeness of Reluplex)

The Reluplex algorithm is sound and complete*

- **Soundness:**
  - SAT $\Rightarrow$ assignment is correct
  - UNSAT $\Rightarrow$ no assignment exists

- **Completeness:** depends on *variable selection strategy* and *splitting strategy*

- Naive approach: split on all variables immediately, apply Bland’s rule
  - This is the case-splitting approach from before
Properties of Reluplex

Theorem (Soundness and Completeness of Reluplex)

The Reluplex algorithm is sound and complete*

- Soundness:
  - SAT $\Rightarrow$ assignment is correct
  - UNSAT $\Rightarrow$ no assignment exists

- Completeness: depends on variable selection strategy and splitting strategy

- Naive approach: split on all variables immediately, apply Bland's rule
  - This is the case-splitting approach from before
  - Ensures termination
More Efficient Reluplex

Better approach:
lazy splitting

Start fixing bound violations

Once all variables within bounds, address broken ReLUs

If a ReLU is repeatedly broken, split on it

Otherwise, fix it without splitting

And repeat as needed

Usually end up splitting on a fraction of the ReLUs (20%)

Can reduce splitting further with some additional work
More Efficient Reluplex

- Better approach: lazy splitting
More Efficient Reluplex

- Better approach: *lazy splitting*
  - Start fixing bound violations
Better approach: *lazy splitting*

- Start fixing bound violations
- Once all variables within bounds, address broken ReLUs

If a ReLU is repeatedly broken, split on it. Otherwise, fix it without splitting. And repeat as needed. Usually end up splitting on a fraction of the ReLUs (20\%). Can reduce splitting further with some additional work.
Better approach: *lazy splitting*

- Start fixing bound violations
- Once all variables within bounds, address broken ReLUs
  - If a ReLU is repeatedly broken, split on it
Better approach: *lazy splitting*

- Start fixing bound violations
- Once all variables within bounds, address broken ReLUs
  - If a ReLU is repeatedly broken, split on it
  - Otherwise, fix it without splitting
Better approach: *lazy splitting*

- Start fixing bound violations
- Once all variables within bounds, address broken ReLUs
  - If a ReLU is repeatedly broken, split on it
  - Otherwise, fix it without splitting
- And repeat as needed
Better approach: *lazy splitting*

- Start fixing bound violations
- Once all variables within bounds, address broken ReLUs
  - If a ReLU is repeatedly broken, split on it
  - Otherwise, fix it without splitting
- And repeat as needed

Usually end up splitting on a fraction of the ReLUs (20%)
Better approach: *lazy splitting*

- Start fixing bound violations
- Once all variables within bounds, address broken ReLUs
  - If a ReLU is repeatedly broken, split on it
  - Otherwise, fix it without splitting
- And repeat as needed

Usually end up splitting on a fraction of the ReLUs (20%)

Can reduce splitting further with some additional work
More Efficient Reluplex: Bound Tightening

During execution we encounter many equations. Can use them for bound tightening.

Example:
\[ x = y + z \]
\[ x \geq -2, \quad y \geq 1, \quad z \geq 1 \]

Can derive tighter bound:
\[ x \geq 2 \]

If \( x \) is part of a ReLU pair, we say the ReLUs phase is fixed. And we replace it by a linear equation. Same as in case splitting, only no back-tracking required.
During execution we encounter many equations

\begin{align*}
x &= y + z \\
x &\geq -2 \\
y &\geq 1 \\
z &\geq 1
\end{align*}

Can derive tighter bound:

\begin{align*}
x &\geq 2
\end{align*}

If \(x\) is part of a ReLU pair, we say the ReLUs phase is fixed and we replace it by a linear equation. Same as in case splitting, only no back-tracking required.
During execution we encounter many equations
Can use them for *bound tightening*
During execution we encounter many equations

Can use them for *bound tightening*

Example:

\[ x = y + z \quad x \geq -2, \quad y \geq 1, \quad z \geq 1 \]
More Efficient Reluplex: Bound Tightening

- During execution we encounter many equations
- Can use them for *bound tightening*
- Example:

  \[ x = y + z \quad x \geq -2, \quad y \geq 1, \quad z \geq 1 \]

- Can derive *tighter* bound: \( x \geq 2 \)
During execution we encounter many equations

Can use them for *bound tightening*

Example:

\[ x = y + z \]

\[ x \geq -2, \quad y \geq 1, \quad z \geq 1 \]

Can derive *tighter* bound: \[ x \geq 2 \]

If \( x \) is part of a ReLU pair, we say the ReLUs phase is *fixed*
More Efficient Reluplex: Bound Tightening

- During execution we encounter many equations
- Can use them for *bound tightening*
- Example:
  \[ x = y + z \quad \quad x \geq -2, \quad y \geq 1, \quad z \geq 1 \]
- Can derive *tighter* bound: \( x \geq 2 \)
- If \( x \) is part of a ReLU pair, we say the ReLUs phase is *fixed*
  - And we replace it by a linear equation
More Efficient Reluplex: Bound Tightening

- During execution we encounter many equations.
- Can use them for *bound tightening*.
- Example:
  \[
  x = y + z \quad x \geq -2, \quad y \geq 1, \quad z \geq 1
  \]
- Can derive *tighter* bound: \( x \geq 2 \).
- If \( x \) is part of a ReLU pair, we say the ReLUs phase is *fixed*.
  - And we replace it by a linear equation.
  - Same as in case splitting, only no back-tracking required.
In every pivot step we examine an equation
Use that equation for bound tightening
For the basic variable
For other variables, too?
Complexity: linear in the size of the equation
Particularly useful after splitting
Because new bounds have been introduced
Can be combined with
backjumping
In every pivot step we examine an equation.
In every pivot step we examine an equation

Use that equation for bound tightening
In every pivot step we examine an equation

Use that equation for bound tightening
  For the basic variable
More Efficient Reluplex: Bound Tightening (cnt’d)

- In every pivot step we examine an equation.
- Use that equation for bound tightening:
  - For the basic variable.
  - For other variables, too?
In every pivot step we examine an equation

Use that equation for bound tightening
  - For the basic variable
  - For other variables, too?
  - Complexity: linear in the size of the equation
In every pivot step we examine an equation

Use that equation for bound tightening
  - For the basic variable
  - For other variables, too?
  - Complexity: linear in the size of the equation

Particularly useful after splitting
In every pivot step we examine an equation

Use that equation for bound tightening

- For the basic variable
- For other variables, too?
- Complexity: linear in the size of the equation

Particularly useful after splitting

- Because new bounds have been introduced
In every pivot step we examine an equation

Use that equation for bound tightening
- For the basic variable
- For other variables, too?
- Complexity: linear in the size of the equation

Particularly useful after splitting
- Because new bounds have been introduced

Can be combined with backjumping
Non-Chronological Backtracking (Backjumping)

A useful technique in SAT and SMT solving

Backtracking: change last guess
Backjumping: change an earlier guess

Need to keep track of the discovery of new bounds
Non-Chronological Backtracking (Backjumping)

- A useful technique in SAT and SMT solving
Non-Chronological Backtracking (Backjumping)

- A useful technique in SAT and SMT solving
- Backtracking: change *last* guess
Non-Chronological Backtracking (Backjumping)

- A useful technique in SAT and SMT solving
- Backtracking: change *last* guess
- Backjumping: change an *earlier* guess
Non-Chronological Backtracking (Backjumping)

- A useful technique in SAT and SMT solving
- Backtracking: change last guess
- Backjumping: change an earlier guess
- Need to keep track of the discovery of new bounds
Non-Chronological Backtracking (Backjumping) (cnt’d)

$y_1 = \text{ReLU}(x_1), \quad y_2 = \text{ReLU}(x_2)$

0

$y_1 = 0, \quad x_1 \leq 0$
$y_2 = 0, \quad x_2 \leq 0$

1

$y_2 = x_2, \quad x_2 \geq 0$

2

UNSAT

1

$y_1 = x_1, \quad x_1 \geq 0$

2

$y_2 = x_2, \quad x_2 \geq 0$

UNSAT

2

UNSAT
Non-Chronological Backtracking (Backjumping) (cnt’d)

\[ y_1 = \text{ReLU}(x_1), \quad y_2 = \text{ReLU}(x_2) \]

1. \( y_1 = 0, x_1 \leq 0 \)
2. \( y_2 = 0, x_2 \leq 0 \)
   - UNSAT
3. \( y_2 = x_2, x_2 \geq 0 \)
   - UNSAT
4. \( y_1 = x_1, x_1 \geq 0 \)
   - UNSAT

Guy Katz (HUJI)
Precision and Numerical Stability

SMT solvers typically use precise arithmetic. This ensures soundness, but it is quite slow.

LP solvers typically use floating point arithmetic. Rounding errors can harm soundness, but it is much faster.

LP solvers attempt to avoid division by tiny fractions. One should do the same when implementing Reluplex.
SMT solvers typically use *precise arithmetic*
SMT solvers typically use *precise arithmetic*. This ensures soundness.
SMT solvers typically use *precise arithmetic*

- This ensures soundness
- But is quite slow
SMT solvers typically use *precise arithmetic*
- This ensures soundness
- But is quite slow

LP solvers typically use *floating point arithmetic*
SMT solvers typically use \textit{precise arithmetic}\par  This ensures soundness\par  But is quite slow\par

LP solvers typically use \textit{floating point arithmetic}\par  Rounding errors can harm soundness
SMT solvers typically use *precise arithmetic*
- This ensures soundness
- But is quite slow

LP solvers typically use *floating point arithmetic*
- Rounding errors can harm soundness
- But is much faster
SMT solvers typically use *precise arithmetic*
- This ensures soundness
- But is quite slow

LP solvers typically use *floating point arithmetic*
- Rounding errors can harm soundness
- But is much faster

LP solvers attempt to avoid division by tiny fractions
SMT solvers typically use \textit{precise arithmetic}
- This ensures soundness
- But is quite slow

LP solvers typically use \textit{floating point arithmetic}
- Rounding errors can harm soundness
- But is much faster

LP solvers attempt to avoid division by tiny fractions

Should do the same when implementing Reluplex
Precision and Numerical Stability (cnt’d)

- Can monitor numerical instability
- Plug current assignment into input formulas
- Measure the error
- If the degradation exceeds a certain threshold, restore the equations from the original
- Fewer pivot operations, and hence more accuracy
- Still does not guarantee soundness
- Open question for most techniques
Can monitor numerical instability
Can monitor numerical instability
  Plug current assignment into input formulas
Can monitor numerical instability
- Plug current assignment into input formulas
- Measure the error
Can monitor numerical instability
  Plug current assignment into input formulas
  Measure the error

If the degradation exceeds a certain threshold, restore the equations from the original
Can monitor numerical instability
- Plug current assignment into input formulas
- Measure the error

If the degradation exceeds a certain threshold, restore the equations from the original
- Fewer pivot operations, and hence more accuracy
Can monitor numerical instability
  - Plug current assignment into input formulas
  - Measure the error

If the degradation exceeds a certain threshold, restore the equations from the original
  - Fewer pivot operations, and hence more accuracy

Still *does not guarantee* soundness
Can monitor numerical instability
  - Plug current assignment into input formulas
  - Measure the error

If the degradation exceeds a certain threshold, restore the equations from the original
  - Fewer pivot operations, and hence more accuracy

Still *does not guarantee* soundness
  - Open question for most techniques
Roadmap

The simplex algorithm, for solving linear programs
Extension into Reluplex, for solving linear programs + ReLUs

Some highlights for an efficient implementation

Up next:
We will talk about use-cases where Reluplex was applied

1. ACAS Xu Verification
2. Adversarial Robustness
3. Reluplex + Clustering

Guy Katz (HUJI)
Roadmap

- The *simplex* algorithm, for solving linear programs
Roadmap

- The *simplex* algorithm, for solving linear programs
- Extension into *Reluplex*, for solving linear programs + ReLUs

Up next:
- ACAS Xu Verification
- Adversarial Robustness
- Reluplex + Clustering
Roadmap

- The *simplex* algorithm, for solving linear programs
- Extension into *Reluplex*, for solving linear programs + ReLUs
- Some highlights for an efficient implementation
Roadmap

- The *simplex* algorithm, for solving linear programs
- Extension into *Reluplex*, for solving linear programs + ReLUs
- Some highlights for an efficient implementation

Up next:
Roadmap

- The *simplex* algorithm, for solving linear programs
- Extension into *Reluplex*, for solving linear programs + ReLUs
- Some highlights for an efficient implementation

Up next:

- We will talk about use-cases where Reluplex was applied
Roadmap

- The *simplex* algorithm, for solving linear programs
- Extension into *Reluplex*, for solving linear programs + ReLUs
- Some highlights for an efficient implementation

Up next:

- We will talk about use-cases where Reluplex was applied
  - ACAS Xu Verification
Roadmap

- The *simplex* algorithm, for solving linear programs
- Extension into *Reluplex*, for solving linear programs + ReLUs
- Some highlights for an efficient implementation

Up next:

We will talk about use-cases where Reluplex was applied

1. ACAS Xu Verification
2. Adversarial Robustness
Roadmap

- The *simplex* algorithm, for solving linear programs
- Extension into *Reluplex*, for solving linear programs + ReLUs
- Some highlights for an efficient implementation

Up next:

- We will talk about use-cases where Reluplex was applied
  1. ACAS Xu Verification
  2. Adversarial Robustness
  3. Reluplex + Clustering
The ACAS Xu System

An Airborne Collision-Avoidance System, for drones
Being developed by the US Federal Aviation Administration (FAA)
Produce an advisory:
Clear-of-conflict (COC)
Strong left
Weak left
Strong right
Weak right
Ownship $v_{own}$
Intruder $v_{int}$
$\rho$
$\psi$
$\theta$
Implemented using neural networks
Guy Katz (HUJI)
Verification of ML
FoPSS 2018 95 / 115
The ACAS Xu System

- An *Airborne Collision-Avoidance System*, for drones
The ACAS Xu System

- An *Airborne Collision-Avoidance System*, for drones
- Being developed by the US Federal Aviation Administration (FAA)
The ACAS Xu System

- An *Airborne Collision-Avoidance System*, for drones
- Being developed by the US Federal Aviation Administration (FAA)
- Produce an advisory:
  - **Clear-of-conflict (COC)**
  - **Strong left**
  - **Weak left**
  - **Strong right**
  - **Weak right**
The ACAS Xu System

- An *Airborne Collision-Avoidance System*, for drones
- Being developed by the US Federal Aviation Administration (FAA)
- Produce an advisory:
  - *Clear-of-conflict (COC)*
  - *Strong left*
  - *Weak left*
  - *Strong right*
  - *Weak right*
- Implemented using neural networks
Certifying ACAS Xu

There are properties that the FAA cares about:
- Consistent alerting regions
- No unnecessary turning advisories
- Strong alerts do not occur when intruder vertically distant

Properties defined formally:
- Constraints on inputs and outputs
There are properties that the FAA cares about
There are properties that the FAA cares about

- Consistent alerting regions
There are properties that the FAA cares about
  - Consistent alerting regions
  - No unnecessary turning advisories
There are properties that the FAA cares about:
- Consistent alerting regions
- No unnecessary turning advisories
- Strong alerts do not occur when intruder vertically distant
Certifying ACAS Xu

- There are properties that the FAA cares about
  - Consistent alerting regions
  - No unnecessary turning advisories
  - Strong alerts do not occur when intruder vertically distant

- Properties defined formally
There are properties that the FAA cares about
- Consistent alerting regions
- No unnecessary turning advisories
- Strong alerts do not occur when intruder vertically distant

Properties defined formally
- Constraints on inputs and outputs
Certifying ACAS Xu (cnt’d)

Example 1:
If the intruder is near and approaching from the left, the network advises strong right.

Distance: $12000 \leq \rho \leq 62000$

Angle to intruder: $0.2 \leq \theta \leq 0.4$

Etc.

Proved in less than 1.5 hours, using 4 machines.
We worked on a list of 10 properties
We worked on a list of 10 properties

Example 1:

- If the intruder is near and approaching from the left, the network advises strong right
- Distance: $12000 \leq \rho \leq 62000$
- Angle to intruder: $0.2 \leq \theta \leq 0.4$

Etc.

Proved in less than 1.5 hours, using 4 machines
Certifying ACAS Xu (cnt’d)

- We worked on a list of 10 properties
- Example 1:
  - If the intruder is near and approaching from the left, the network advises strong right
Certifying ACAS Xu (cnt’d)

- We worked on a list of 10 properties
- Example 1:
  - If the intruder is near and approaching from the left, the network advises strong right
    - Distance: $12000 \leq \rho \leq 62000$
Certifying ACAS Xu (cnt’d)

- We worked on a list of 10 properties
- Example 1:
  - If the intruder is near and approaching from the left, the network advises strong right
    - Distance: $12000 \leq \rho \leq 62000$
    - Angle to intruder: $0.2 \leq \theta \leq 0.4$
We worked on a list of 10 properties

Example 1:

If the intruder is near and approaching from the left, the network advises strong right

- Distance: \(12000 \leq \rho \leq 62000\)
- Angle to intruder: \(0.2 \leq \theta \leq 0.4\)
- Etc.
We worked on a list of 10 properties

Example 1:
- If the intruder is near and approaching from the left, the network advises strong right
  - Distance: \(12000 \leq \rho \leq 62000\)
  - Angle to intruder: \(0.2 \leq \theta \leq 0.4\)
  - Etc.

- Proved in less than 1.5 hours, using 4 machines
Example 2:
If vertical separation is large and the previous advisory is weak left, the network advises clear-of-conflict or weak left.

Distance:
\[ 0 \leq \rho \leq 60 \]

Time to loss of vertical separation:
\[ \tau = 100 \]

Etc.

Found a counter-example in 11 hours.
Example 2:

If vertical separation is large and the previous advisory is weak left, the network advises clear-of-conflict or weak left.

Distance: $0 \leq \rho \leq 60$

Time to loss of vertical separation: $\tau = 100$

Etc.
Example 2:

- If vertical separation is large and the previous advisory is weak left, the network advises clear-of-conflict or weak left.
Example 2:

- If vertical separation is large and the previous advisory is weak left, the network advises clear-of-conflict or weak left
  - Distance: $0 \leq \rho \leq 60760$
Example 2:

If vertical separation is large and the previous advisory is weak left, the network advises clear-of-conflict or weak left

- Distance: $0 \leq \rho \leq 60760$
- Time to loss of vertical separation: $\tau = 100$
Example 2:
If vertical separation is large and the previous advisory is weak left, the network advises clear-of-conflict or weak left
- Distance: $0 \leq \rho \leq 60760$
- Time to loss of vertical separation: $\tau = 100$
- Etc.
Example 2:
- If vertical separation is large and the previous advisory is weak left, the network advises clear-of-conflict or weak left
  - Distance: \(0 \leq \rho \leq 60760\)
  - Time to loss of vertical separation: \(\tau = 100\)
  - Etc.
- Found a counter-example in 11 hours
<table>
<thead>
<tr>
<th>Networks</th>
<th>Result</th>
<th>Time</th>
<th>Stack</th>
<th>Splits</th>
</tr>
</thead>
<tbody>
<tr>
<td>φ1</td>
<td>UNSAT</td>
<td>1522384</td>
<td>47</td>
<td>394517</td>
</tr>
<tr>
<td>φ2</td>
<td>UNSAT</td>
<td>88388</td>
<td>55</td>
<td>463</td>
</tr>
<tr>
<td>φ3</td>
<td>UNSAT</td>
<td>52080</td>
<td>22</td>
<td>28156</td>
</tr>
<tr>
<td>φ4</td>
<td>UNSAT</td>
<td>23940</td>
<td>21</td>
<td>12475</td>
</tr>
<tr>
<td>φ5</td>
<td>UNSAT</td>
<td>58914</td>
<td>46</td>
<td>19355</td>
</tr>
<tr>
<td>φ6</td>
<td>UNSAT</td>
<td>548496</td>
<td>50</td>
<td>180288</td>
</tr>
<tr>
<td>φ7</td>
<td>TIMEOUT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>φ8</td>
<td>SAT</td>
<td>116697</td>
<td>69</td>
<td>40102</td>
</tr>
<tr>
<td>φ9</td>
<td>UNSAT</td>
<td>227002</td>
<td>48</td>
<td>99634</td>
</tr>
<tr>
<td>φ10</td>
<td>UNSAT</td>
<td>88520</td>
<td>49</td>
<td>19944</td>
</tr>
</tbody>
</table>
## Certifying ACAS Xu (cnt’d)

<table>
<thead>
<tr>
<th>Networks</th>
<th>Result</th>
<th>Time</th>
<th>Stack</th>
<th>Splits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_1$</td>
<td>41</td>
<td>UNSAT</td>
<td>394517</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>TIMEOUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\phi_2$</td>
<td>1</td>
<td>UNSAT</td>
<td>463</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>SAT</td>
<td>82419</td>
<td>44</td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>42</td>
<td>UNSAT</td>
<td>28156</td>
<td>22</td>
</tr>
<tr>
<td>$\phi_4$</td>
<td>42</td>
<td>UNSAT</td>
<td>12475</td>
<td>21</td>
</tr>
<tr>
<td>$\phi_5$</td>
<td>1</td>
<td>UNSAT</td>
<td>19355</td>
<td>46</td>
</tr>
<tr>
<td>$\phi_6$</td>
<td>1</td>
<td>UNSAT</td>
<td>180288</td>
<td>50</td>
</tr>
<tr>
<td>$\phi_7$</td>
<td>1</td>
<td>TIMEOUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\phi_8$</td>
<td>1</td>
<td>SAT</td>
<td>40102</td>
<td>69</td>
</tr>
<tr>
<td>$\phi_9$</td>
<td>1</td>
<td>UNSAT</td>
<td>99634</td>
<td>48</td>
</tr>
<tr>
<td>$\phi_{10}$</td>
<td>1</td>
<td>UNSAT</td>
<td>19944</td>
<td>49</td>
</tr>
</tbody>
</table>
Adversarial Robustness

Slight perturbations of inputs lead to misclassification.

Verification can prove that this cannot occur.

Allows us to assess attacks defenses.

Guy Katz (HUJI)
Adversarial Robustness

Slight perturbations of inputs lead to misclassification. Verification can prove that this cannot occur, allowing us to assess attacks and defenses.

Goodfellow et al., 2015

"panda" 57.7% confidence

\[ \text{image} + \epsilon \times \text{perturbation} = \text{image} \]

"gibbon" 99.3% confidence
Adversarial Robustness

Slight perturbations of inputs lead to misclassification

Goodfellow et al., 2015

- “panda” 57.7% confidence
- \( \epsilon \)
- “gibbon” 99.3% confidence
Adversarial Robustness

Slight perturbations of inputs lead to misclassification

Verification can prove that this cannot occur
Adversarial Robustness

- Slight perturbations of inputs lead to misclassification
- Verification can prove that this cannot occur
- Allows us to assess attacks defenses

Goodfellow et al., 2015
Local Adversarial Robustness

Verification question: for a given panda $\bar{x}$ and a given amount of noise $\delta$, does classification remain the same?

If $\|\bar{x} - \bar{x}_0\|_L \leq \delta$ then $\bigwedge_i (\bar{y}[i]_0 \geq \bar{y}[i])$, where $\bar{y}[i]_0$ is the desired label.

Easiest norm to handle: $L_\infty$, the infinity norm

$\|\bar{x} - \bar{x}_0\|_L \leq \delta \iff \forall i. -\delta \leq \bar{x}[i] - \bar{x}_0[i] \leq \delta$.

Can also handle $L_1$:

$\|\bar{x} - \bar{x}_0\|_L \leq \delta \iff \sum_{i=1}^n |\bar{x}[i] - \bar{x}_0[i]| \leq \delta$.

And we know that $\max(a,b) = \text{ReLU}(a-b) + b$. 

Guy Katz (HUJI)
Local Adversarial Robustness

- Verification question: for a given panda $\bar{x}_0$ and a given amount of noise $\delta$, does classification remain the same?
Verification question: for a given panda $\bar{x}_0$ and a given amount of noise $\delta$, does classification remain the same?

If $\|\bar{x} - \bar{x}_0\|_L \leq \delta$ then $\bigwedge_i (\bar{y}[i_0] \geq \bar{y}[i])$, where $\bar{y}[i_0]$ is the desired label.
Verification question: for a given panda $\bar{x}_0$ and a given amount of noise $\delta$, does classification remain the same?

- If $\|\bar{x} - \bar{x}_0\|_L \leq \delta$ then $\bigwedge_i (\bar{y}[i_0] \geq \bar{y}[i])$, where $\bar{y}[i_0]$ is the desired label.

- Easiest norm to handle: $L_\infty$, the infinity norm.
Local Adversarial Robustness

- Verification question: for a given panda \( \bar{x}_0 \) and a given amount of noise \( \delta \), does classification remain the same?
  - If \( \| \bar{x} - \bar{x}_0 \|_L \leq \delta \) then \( \bigwedge_i (\bar{y}[i_0] \geq \bar{y}[i]) \), where \( \bar{y}[i_0] \) is the desired label

- Easiest norm to handle: \( L_{\infty} \), the infinity norm
  - \( \| \bar{x} - \bar{x}_0 \|_{L_{\infty}} \leq \delta \iff \forall i. -\delta \leq \bar{x}[i] - \bar{x}_0[i] \leq \delta \)
Local Adversarial Robustness

- Verification question: for a given panda $\tilde{x}_0$ and a given amount of noise $\delta$, does classification remain the same?
  - If $\|\tilde{x} - \tilde{x}_0\|_L \leq \delta$ then $\bigwedge_i(\tilde{y}[i_0] \geq \tilde{y}[i])$, where $\tilde{y}[i_0]$ is the desired label.

- Easiest norm to handle: $L_\infty$, the infinity norm
  - $\|\tilde{x} - \tilde{x}_0\|_{L_\infty} \leq \delta \iff \forall i. -\delta \leq \tilde{x}[i] - \tilde{x}_0[i] \leq \delta$

- Can also handle $L_1$: 
Verification question: for a given panda $\tilde{x}_0$ and a given amount of noise $\delta$, does classification remain the same?

- If $\|\tilde{x} - \tilde{x}_0\|_L \leq \delta$ then $\bigwedge_i (\bar{y}[i_0] \geq \bar{y}[i])$, where $\bar{y}[i_0]$ is the desired label.

Easiest norm to handle: $L_\infty$, the infinity norm

- $\|\tilde{x} - \tilde{x}_0\|_{L_\infty} \leq \delta \iff \forall i. -\delta \leq \tilde{x}[i] - \tilde{x}_0[i] \leq \delta$

Can also handle $L_1$:

- $\|\tilde{x} - \tilde{x}_0\|_{L_1} \leq \delta \iff \sum_{i=1}^n |\tilde{x}[i] - \tilde{x}_0[i]| \leq \delta$
Local Adversarial Robustness

- Verification question: for a given panda $\bar{x}_0$ and a given amount of noise $\delta$, does classification remain the same?
  - If $\|\bar{x} - \bar{x}_0\|_L \leq \delta$ then $\bigwedge_i (\bar{y}[i_0] \geq \bar{y}[i])$, where $\bar{y}[i_0]$ is the desired label.

- Easiest norm to handle: $L_\infty$, the infinity norm
  - $\|\bar{x} - \bar{x}_0\|_{L_\infty} \leq \delta \iff \forall i. -\delta \leq \bar{x}[i] - \bar{x}_0[i] \leq \delta$

- Can also handle $L_1$:
  - $\|\bar{x} - \bar{x}_0\|_{L_1} \leq \delta \iff \sum_{i=1}^{n} |\bar{x}[i] - \bar{x}_0[i]| \leq \delta$
  - $|\bar{x}[i] - \bar{x}_0[i]| = \max(\bar{x}[i] - \bar{x}_0[i], \bar{x}[i] - \bar{x}_0[i])$
Local Adversarial Robustness

Verification question: for a given panda $\bar{x}_0$ and a given amount of noise $\delta$, does classification remain the same?

- If $\|\bar{x} - \bar{x}_0\|_L \leq \delta$ then $\bigwedge_i (\bar{y}[i_0] \geq \bar{y}[i])$, where $\bar{y}[i_0]$ is the desired label

Easiest norm to handle: $L_\infty$, the infinity norm

- $\|\bar{x} - \bar{x}_0\|_{L_\infty} \leq \delta \iff \forall i. -\delta \leq \bar{x}[i] - \bar{x}_0[i] \leq \delta$

Can also handle $L_1$:

- $\|\bar{x} - \bar{x}_0\|_{L_1} \leq \delta \iff \sum_{i=1}^{n} |\bar{x}[i] - \bar{x}_0[i]| \leq \delta$
- $|\bar{x}[i] - \bar{x}_0[i]| = \max(\bar{x}[i] - \bar{x}_0[i], \bar{x}_0[i] - \bar{x}[i])$
- And we know that $\max(a, b) = \text{ReLU}(a - b) + b$
Local Adversarial Robustness (cnt'd)

Can find the optimal $\delta$ for which robustness holds

Using binary search

Example: an ACAS Xu network

$\delta = 0.1$

$\delta = 0.075$

$\delta = 0.05$

$\delta = 0.025$

$\delta = 0.01$

<table>
<thead>
<tr>
<th>Point</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>135</td>
<td>239</td>
<td>24</td>
<td>285</td>
<td>57</td>
</tr>
<tr>
<td>SAT</td>
<td>2</td>
<td>977</td>
<td>1168</td>
<td>656</td>
<td>7</td>
</tr>
<tr>
<td>UNSAT</td>
<td>5880</td>
<td>1167</td>
<td>285</td>
<td>57</td>
<td>5</td>
</tr>
<tr>
<td>UNSAT</td>
<td>863</td>
<td>436</td>
<td>99</td>
<td>53</td>
<td>1</td>
</tr>
<tr>
<td>UNSAT</td>
<td>14560</td>
<td>4344</td>
<td>1331</td>
<td>221</td>
<td>6</td>
</tr>
</tbody>
</table>

Result Time Result Time Result Time Result Time Result Time
Can find the \textit{optimal} $\delta$ for which robustness holds
Can find the *optimal* $\delta$ for which robustness holds
- Using binary search
Can find the *optimal* $\delta$ for which robustness holds
- Using binary search

Example: an ACAS Xu network
Can find the **optimal** $\delta$ for which robustness holds

- Using binary search

Example: an ACAS Xu network

<table>
<thead>
<tr>
<th>Point</th>
<th>$\delta = 0.1$</th>
<th>$\delta = 0.075$</th>
<th>$\delta = 0.05$</th>
<th>$\delta = 0.025$</th>
<th>$\delta = 0.01$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Result</td>
<td>Time</td>
<td>Result</td>
<td>Time</td>
<td>Result</td>
</tr>
<tr>
<td>Point 1</td>
<td>SAT</td>
<td>135</td>
<td>SAT</td>
<td>239</td>
<td>SAT</td>
</tr>
<tr>
<td>Point 2</td>
<td>UNSAT</td>
<td>5880</td>
<td>UNSAT</td>
<td>1167</td>
<td>UNSAT</td>
</tr>
<tr>
<td>Point 3</td>
<td>UNSAT</td>
<td>863</td>
<td>UNSAT</td>
<td>436</td>
<td>UNSAT</td>
</tr>
<tr>
<td>Point 4</td>
<td>SAT</td>
<td>2</td>
<td>SAT</td>
<td>977</td>
<td>SAT</td>
</tr>
<tr>
<td>Point 5</td>
<td>UNSAT</td>
<td>14560</td>
<td>UNSAT</td>
<td>4344</td>
<td>UNSAT</td>
</tr>
</tbody>
</table>
Assessing Attacks and Defenses [CKBD18]

Assessing attacks:

- Pick point $\bar{x}$
- Use verification to find optimal $\delta$
- Use attack to find $\delta'$
- See how close $\delta'$ is to $\delta$.

On average, $\delta$ about 6% smaller than $\delta'$. 

Guy Katz (HUJI)
Assessing attacks:

- Pick point $\bar{x}$
- Use verification to find optimal $\delta$
- Use attack to find $\delta'$
- See how close $\delta'$ is to $\delta$
Assessing Attacks and Defenses [CKBD18]

- Assessing attacks:
  - Pick point $\bar{x}$
Assessing attacks:
- Pick point $\bar{x}$
- Use verification to find optimal $\delta$
Assessing attacks:

- Pick point $\bar{x}$
- Use *verification* to find optimal $\delta$
- Use *attack* to find $\delta'$

Example: Carlini-Wagner attack [CW17] on a small MNIST network

On average, $\delta$ about $6\%$ smaller than $\delta'$. 
Assessing attacks:

- Pick point $\bar{x}$
- Use *verification* to find optimal $\delta$
- Use *attack* to find $\delta'$
- See how close $\delta'$ is to $\delta$
Assessing attacks:
- Pick point $\bar{x}$
- Use verification to find optimal $\delta$
- Use attack to find $\delta'$
- See how close $\delta'$ is to $\delta$

Example: Carlini-Wagner attack [CW17] on a small MNIST network
Assessing attacks:

- Pick point $\bar{x}$
- Use *verification* to find optimal $\delta$
- Use *attack* to find $\delta'$
- See how close $\delta'$ is to $\delta$

Example: Carlini-Wagner attack [CW17] on a small MNIST network

On average, $\delta$ about 6% smaller than $\delta'$
Assessing defenses:

Start with network $N$

Train hardened network $\overline{N}$

Pick point $\overline{x}$

Compare optimal $\delta$ before and after hardening

Example: Madry defense [MMS+18] on a small MNIST network

On average, hardened $\delta$ about 423% larger

However, smaller in some cases
Assessing defenses:

- Assessing defenses:

  - Start with network $N$
  - Train hardened network $\bar{N}$
  - Pick point $\bar{x}$
  - Compare optimal $\delta$ before and after hardening

  Example: Madry defense [MMS+18] on a small MNIST network

  On average, hardened $\delta$ about 423% larger

  However, smaller in some cases
Assessing defenses:
- Start with network $N$

Example: Madry defense [MMS+18] on a small MNIST network

On average, hardened $\delta$ about 423% larger
However, smaller in some cases
Assessing defenses:

- Start with network $N$
- Train *hardened* network $\bar{N}$
Assessing defenses:
- Start with network $N$
- Train *hardened* network $\tilde{N}$
- Pick point $\bar{x}$
Assessing defenses:
- Start with network $N$
- Train \textit{hardened} network $\tilde{N}$
- Pick point $\bar{x}$
- Compare optimal $\delta$ \textit{before} and \textit{after} hardening
Assessing defenses:

- Start with network $N$
- Train *hardened* network $\tilde{N}$
- Pick point $\bar{x}$
- Compare optimal $\delta$ *before* and *after* hardening

Example: Madry defense [MMS$^+$18] on a small MNIST network

On average, hardened $\delta$ about 423% larger
However, smaller in some cases
Assessing defenses:

- Start with network $N$
- Train hardened network $\tilde{N}$
- Pick point $\tilde{x}$
- Compare optimal $\delta$ before and after hardening

Example: Madry defense [MMS+18] on a small MNIST network

On average, hardened $\delta$ about $423\%$ larger
Assessing defenses:
- Start with network $N$
- Train *hardened* network $\overline{N}$
- Pick point $\bar{x}$
- Compare optimal $\delta$ *before* and *after* hardening

Example: Madry defense [MMS$^+$18] on a small MNIST network
- On average, hardened $\delta$ about 423% larger
- However, smaller in some cases
Global Robustness?

Previous definition: for a particular input $\bar{x}_0$

What's an acceptable $\delta$?

How do you pick $\bar{x}_0$?

Can you evaluate the overall robustness?
Global Robustness?

- Previous definition: for a particular input $\bar{x}_0$
Global Robustness?

- Previous definition: for a particular input $\bar{x}_0$
  - What’s an acceptable $\delta$?
Global Robustness?

- Previous definition: for a particular input $\bar{x}_0$
  - What’s an acceptable $\delta$?
  - How do you pick $\bar{x}_0$?
Global Robustness?

- Previous definition: for a particular input $\bar{x}_0$
  - What’s an acceptable $\delta$?
  - How do you pick $\bar{x}_0$?
  - Can you evaluate the overall robustness?
Global Robustness?

- Previous definition: for a particular input $\bar{x}_0$
  - What’s an acceptable $\delta$?
  - How do you pick $\bar{x}_0$?
  - Can you evaluate the overall robustness?
Global Robustness Queries

Region boundaries: look at confidence instead of label. Let $p_1, p_2$ be confidence levels for certain label: $\forall \bar{x}_1, \bar{x}_2. \|\bar{x}_1 - \bar{x}_2\| \leq \delta \Rightarrow |p_1 - p_2| \leq \epsilon$.

Small changes to input do not change output by much. Significantly slower to compute.

Double the network size. Large input regions. And also still need to choose $\delta, \epsilon$.

A compromise: a clustering based approach.
Global Robustness Queries

- Region boundaries: look at confidence instead of label

Let \( p_1, p_2 \) be confidence levels for certain label:

\[
\forall \bar{x}_1, \bar{x}_2. \quad \|\bar{x}_1 - \bar{x}_2\| \leq \delta \implies |p_1 - p_2| \leq \epsilon
\]

Small changes to input do not change output by much

Significantly slower to compute

Double the network size

Large input regions

And also still need to choose \( \delta, \epsilon \)

A compromise: a clustering based approach
Global Robustness Queries

- Region boundaries: look at *confidence* instead of label
- Let $p_1, p_2$ be confidence levels for certain label:

\[ \forall \overline{x}_1, \overline{x}_2. \quad \|\overline{x}_1 - \overline{x}_2\| \leq \delta \Rightarrow |p_1 - p_2| \leq \epsilon \]
Global Robustness Queries

- Region boundaries: look at confidence instead of label
- Let $p_1, p_2$ be confidence levels for certain label:

$$\forall \bar{x}_1, \bar{x}_2. \quad \|\bar{x}_1 - \bar{x}_2\| \leq \delta \Rightarrow |p_1 - p_2| \leq \epsilon$$

- Small changes to input do not change output by much
Regional boundaries: look at confidence instead of label.

Let $p_1, p_2$ be confidence levels for certain label:

$$\forall \bar{x}_1, \bar{x}_2. \ ||\bar{x}_1 - \bar{x}_2|| \leq \delta \Rightarrow |p_1 - p_2| \leq \epsilon$$

- Small changes to input do not change output by much.

- **Significantly** slower to compute.


Global Robustness Queries

- Region boundaries: look at *confidence* instead of label
- Let $p_1, p_2$ be confidence levels for certain label:

  \[ \forall \bar{x}_1, \bar{x}_2. \quad \|\bar{x}_1 - \bar{x}_2\| \leq \delta \implies |p_1 - p_2| \leq \epsilon \]

- Small changes to input do not change output by much
- *Significantly* slower to compute
  - Double the network size
Global Robustness Queries

- Region boundaries: look at *confidence* instead of label

- Let \( p_1, p_2 \) be confidence levels for certain label:

  \[
  \forall \bar{x}_1, \bar{x}_2. \quad \|\bar{x}_1 - \bar{x}_2\| \leq \delta \Rightarrow |p_1 - p_2| \leq \epsilon
  \]

- Small changes to input do not change output by much

- *Significantly* slower to compute
  - Double the network size
  - Large input regions
Global Robustness Queries

- Region boundaries: look at *confidence* instead of label

- Let $p_1, p_2$ be confidence levels for certain label:
  \[ \forall \bar{x}_1, \bar{x}_2. \quad \|\bar{x}_1 - \bar{x}_2\| \leq \delta \Rightarrow |p_1 - p_2| \leq \epsilon \]

- Small changes to input do not change output by much

- *Significantly* slower to compute
  - Double the network size
  - Large input regions

- And also still need to choose $\delta, \epsilon$
Global Robustness Queries

- Region boundaries: look at confidence instead of label
- Let $p_1, p_2$ be confidence levels for certain label:
  \[
  \forall \bar{x}_1, \bar{x}_2. \quad \| \bar{x}_1 - \bar{x}_2 \| \leq \delta \Rightarrow |p_1 - p_2| \leq \epsilon
  \]
- Small changes to input do not change output by much
- **Significantly** slower to compute
  - Double the network size
  - Large input regions
- And also still need to choose $\delta, \epsilon$
- A compromise: a clustering based approach
Use clustering to identify regions on which the network should be consistent.

Clustering applied to known points (e.g., training set)

Identify centroid \( \bar{x}_0 \) and radius \( \delta \) for each cluster

Higher degree of automation

Discovered an adversarial example in ACAS Xu
Use *clustering* to identify regions on which the network should be consistent.
Use *clustering* to identify regions on which the network should be consistent

- Clustering applied to known points (e.g., training set)
Use *clustering* to identify regions on which the network should be consistent

- Clustering applied to known points (e.g., training set)
- Identify centroid $\bar{x}_0$ and radius $\delta$ for each cluster
Use *clustering* to identify regions on which the network should be consistent

- Clustering applied to known points (e.g., training set)
- Identify centroid $\bar{x}_0$ and radius $\delta$ for each cluster
Use *clustering* to identify regions on which the network should be consistent

- Clustering applied to known points (e.g., training set)
- Identify centroid $\overline{x}_0$ and radius $\delta$ for each cluster

Higher degree of automation
Use clustering to identify regions on which the network should be consistent

- Clustering applied to known points (e.g., training set)
- Identify centroid $\bar{x}_0$ and radius $\delta$ for each cluster

- Higher degree of automation
- Discovered an adversarial example in ACAS Xu
Summary

Software generated by machine learning is becoming widespread. Certifying this software is a new and exciting challenge. Verification can play a key role.

The main questions:

- How do we verify?
- What do we verify?
Software generated by machine learning is becoming *widespread*
Software generated by machine learning is becoming widespread.

Certifying this software is a new and exciting challenge.
Summary

- Software generated by machine learning is becoming widespread.
- Certifying this software is a new and exciting challenge.
- Verification can play a key role.
Software generated by machine learning is becoming \textit{widespread}

\textit{Certifying} this software is a new and exciting challenge

\textit{Verification} can play a key role

The main questions:
Summary

- Software generated by machine learning is becoming widespread
- **Certifying** this software is a new and exciting challenge
- **Verification** can play a key role
- The main questions:
  - *How* do we verify?
Software generated by machine learning is becoming widespread.

Certifying this software is a new and exciting challenge.

Verification can play a key role.

The main questions:
- How do we verify?
- What do we verify?
Summary - Approaches to Verification

The sound and complete approaches

An NP-complete problem

Usually based on the case splitting approach

Can be improved with:

- Tighter bound derivation
- Splitting heuristics
- Local optimization steps

Guy Katz (HUJI)
The *sound and complete* approaches
Summary - Approaches to Verification

- The *sound and complete* approaches
  - An NP-complete problem
The sound and complete approaches

- An NP-complete problem
- Usually based on the case splitting approach
The *sound and complete* approaches

- An NP-complete problem
- Usually based on the *case splitting* approach
- Can be improved with:
The *sound and complete* approaches

- An NP-complete problem
- Usually based on the *case splitting* approach
- Can be improved with:
  - Tighter bound derivation
Summary - Approaches to Verification

The *sound and complete* approaches

- An NP-complete problem
- Usually based on the *case splitting* approach
- Can be improved with:
  - Tighter bound derivation
  - Splitting heuristics
The *sound and complete* approaches

- An NP-complete problem
- Usually based on the *case splitting* approach
- Can be improved with:
  - Tighter bound derivation
  - Splitting heuristics
  - Local optimization steps
Summary - Approaches to Verification (cnt’d)

Trading completeness for scalability

Discretization and exhaustive search techniques

Correct-by-construction networks

Abstraction techniques

Approximating the network

Approximating the input property
Trading *completeness* for *scalability*
Trading *completeness* for *scalability*

- Discretization and exhaustive search techniques
• Trading *completeness* for *scalability*
  - Discretization and exhaustive search techniques
  - Correct-by-construction networks
Summary - Approaches to Verification (cnt’d)

• Trading *completeness* for *scalability*
  - Discretization and exhaustive search techniques
  - Correct-by-construction networks

• Abstraction techniques
Trading *completeness* for *scalability*
- Discretization and exhaustive search techniques
- Correct-by-construction networks

Abstraction techniques
- Approximating the *network*
Summary - Approaches to Verification (cnt’d)

- Trading *completeness* for *scalability*
  - Discretization and exhaustive search techniques
  - Correct-by-construction networks

- Abstraction techniques
  - Approximating the *network*
  - Approximating the *input property*
Properties to Verify

- Domain-specific properties
  - Example: ACAS Xu
  - Human input required — a known issue in verification

- General properties
  - Adversarial robustness
  - Always desirable, regardless of networks
  - Can we find other such properties?
Properties to Verify

- *Domain-specific* properties

Example: ACAS Xu

Human input required — a known issue in verification

General properties

Adversarial robustness

Always desirable, regardless of networks

Can we find other such properties?
Properties to Verify

- **Domain-specific** properties
  - Example: ACAS Xu
Properties to Verify

- **Domain-specific** properties
  - Example: ACAS Xu
  - Human input required — a known issue in verification
Properties to Verify

- **Domain-specific** properties
  - Example: ACAS Xu
  - Human input required — a known issue in verification

- **General properties**
Properties to Verify

- **Domain-specific** properties
  - Example: ACAS Xu
  - Human input required — a known issue in verification

- **General properties**
  - Adversarial robustness
Properties to Verify

- **Domain-specific** properties
  - Example: ACAS Xu
  - Human input required — a known issue in verification

- **General properties**
  - Adversarial robustness
  - Always desirable, regardless of networks
Properties to Verify

- **Domain-specific** properties
  - Example: ACAS Xu
  - Human input required — a known issue in verification

- **General properties**
  - Adversarial robustness
  - Always desirable, regardless of networks
  - Can we find other such properties?
Ongoing Work in the Reluplex Project

Improving scalability

Currently: linear and non-linear steps roughly independent

Can we solve both kinds of constraints together?

Better SMT techniques?

Proof certificates

Numerical stability is an issue

SAT answers can be checked, but what about UNSAT?

Replay the solution, using precise arithmetic

Generate an externally-checkable proof certificate
Ongoing Work in the Reluplex Project

- Improving *scalability*
Ongoing Work in the Reluplex Project

- Improving *scalability*
  - Currently: linear and non-linear steps roughly independent
Improving *scalability*

- Currently: linear and non-linear steps roughly independent
- Can we solve both kinds of constraints *together*?
Ongoing Work in the Reluplex Project

- Improving *scalability*
  - Currently: linear and non-linear steps roughly independent
  - Can we solve both kinds of constraints *together*?
  - Better *SMT techniques*?

- Numerical stability is an issue
  - SAT answers can be checked, but what about UNSAT?
  - Replay the solution, using precise arithmetic
  - Generate an externally-checkable proof certificate
Ongoing Work in the Reluplex Project

- Improving *scalability*
  - Currently: linear and non-linear steps roughly independent
  - Can we solve both kinds of constraints *together*?
  - Better *SMT techniques*?

- *Proof certificates*
Ongoing Work in the Reluplex Project

- Improving *scalability*
  - Currently: linear and non-linear steps roughly independent
  - Can we solve both kinds of constraints *together*?
  - Better *SMT techniques*?

- **Proof certificates**
  - Numerical stability is an issue
Improving *scalability*
- Currently: linear and non-linear steps roughly independent
- Can we solve both kinds of constraints *together*?
- Better *SMT techniques*?

*Proof certificates*
- Numerical stability is an issue
- SAT answers can be checked, but what about UNSAT?
Ongoing Work in the Reluplex Project

- Improving *scalability*
  - Currently: linear and non-linear steps roughly independent
  - Can we solve both kinds of constraints *together*?
  - Better *SMT techniques*?

- **Proof certificates**
  - Numerical stability is an issue
  - SAT answers can be checked, but what about UNSAT?
  - *Replay* the solution, using *precise arithmetic*
Ongoing Work in the Reluplex Project

- Improving *scalability*
  - Currently: linear and non-linear steps roughly independent
  - Can we solve both kinds of constraints *together*?
  - Better *SMT techniques*?

- **Proof certificates**
  - Numerical stability is an issue
  - SAT answers can be checked, but what about UNSAT?
  - *Replay* the solution, using *precise arithmetic*
  - Generate an *externally-checkable* proof certificate
Ongoing Work in the Reluplex Project (cnt’d)

More expressiveness to handle non-piece-wise linear activation functions?

Case studies: More extensive verification of ACAS Xu systems in which the network is just a component?

Collaboration with various industrial partners
More expressiveness
Ongoing Work in the Reluplex Project (cnt’d)

- More expressiveness
  - Handle non piece-wise linear activation functions?
Ongoing Work in the Reluplex Project (cnt’d)

- More *expressiveness*
  - Handle *non piece-wise linear* activation functions?
- *Case studies*
Ongoing Work in the Reluplex Project (cnt’d)

- More *expressiveness*
  - Handle *non piece-wise linear* activation functions?

- *Case studies*
  - More extensive verification of *ACAS Xu*
Ongoing Work in the Reluplex Project (cnt’d)

- More *expressiveness*
  - Handle *non piece-wise linear* activation functions?

- *Case studies*
  - More extensive verification of *ACAS Xu*
  - Systems in which the network is just a component?
More *expressiveness*

- Handle *non piece-wise linear* activation functions?

*Case studies*

- More extensive verification of *ACAS Xu*
- Systems in which the network is just a component?
- Collaboration with various industrial partners
Ongoing Work in the Reluplex Project (cnt’d)

- More expressiveness
  - Handle non piece-wise linear activation functions?
- Case studies
  - More extensive verification of ACAS Xu
  - Systems in which the network is just a component?
  - Collaboration with various industrial partners.
Thank You!

Questions
Measuring Neural Net Robustness with Constraints.


N. Carlini, G. Katz, C. Barrett, and D. Dill.

C. Cheng, G. Nührenberg, and H. Ruess.
Maximum Resilience of Artificial Neural Networks.

C. Cheng, G. Nührenberg, and H. Ruess.
Verification of Binarized Neural Networks, 2017.

N. Carlini and D. Wagner.
Towards Evaluating the Robustness of Neural Networks.

Training Verified Learners with Learned Verifiers, 2018.

S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari.
Output Range Analysis for Deep Feedforward Neural.

R. Ehlers.
Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks.
DeepSafe: A Data-Driven Approach for Assessing Robustness of Neural Networks.
In Proc. 16th Int. Symp. on Automated Technology for Verification and Analysis (ATVA), 2018.
To appear.

T. Gehr, M. Mirman, D. Drachsler-Cohen, E. Tsankov, S. Chaudhuri, and M. Vechev.
AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation.

M. Hein and M. Andriushchenko.
Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation.

X. Huang, M. Kwiatkowska, S. Wang, and M. Wu.
Safety Verification of Deep Neural Networks.

Verification and Validation of Neural Networks for Safety-Critical Applications.

Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks.

Towards Proving the Adversarial Robustness of Deep Neural Networks.

A. Lomuscio and L. Maganti.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu.
Towards Deep Learning Models Resistant to Adversarial Attacks.
Verifying Properties of Binarized Deep Neural Networks.

L. Pulina and A. Tacchella.
An Abstraction-Refinement Approach to Verification of Artificial Neural Networks.

W. Ruan, X. Huang, and M. Kwiatkowska.
Reachability Analysis of Deep Neural Networks with Provable Guarantees.
In Proc. 27th Int. Joint Conf. on Artificial Intelligence (IJCAI), 2018.

A. Raghunathan, J. Steinhardt, and P. Liang.
Certified Defenses against Adversarial Examples.
In Proc. 6th Int. Conf. on Learning Representations (ICLR), 2018.

W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska.

V. Tjeng and R. Tedrake.
Evaluating Robustness of Neural Networks with Mixed Integer Programming, 2017.


Towards Fast Computation of Certified Robustness for ReLU Networks.
In Proc. 35th Int. Conf. on Machine Learning (ICML), 2018.

W. Xiang, H. Tran, and T. Johnson.