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 $N : \bar{x} \to \bar{y}$, 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

Neural Network Verification

Definition (The Neural Network Verification Problem)

For a neural network $N : \bar{x} \to \bar{y}$, 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

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*

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

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*

$P(\bar{x})$:

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*

$P(\bar{x})$:

- $\bar{x}[0] \geq 40000$

$Q(\bar{y})$:

- $(\bar{y}[0] \leq \bar{y}[1]) \vee (\bar{y}[0] \leq \bar{y}[2]) \vee (\bar{y}[0] \leq \bar{y}[3]) \vee (\bar{y}[0] \leq \bar{y}[4])$

Example: ACAS Xu

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

$P(\bar{x})$:

- $\bar{x}[0] \geq 40000$

$Q(\bar{y})$:

- $(\bar{y}[0] \leq \bar{y}[1]) \vee (\bar{y}[0] \leq \bar{y}[2]) \vee (\bar{y}[0] \leq \bar{y}[3]) \vee (\bar{y}[0] \leq \bar{y}[4])$

`UNSAT` means the system behaves as expected

Example: Adversarial Robustness

Example: Adversarial Robustness

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

Example: Adversarial Robustness

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

$P(\bar{x})$:

Example: Adversarial Robustness

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

$P(\bar{x})$:

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

Example: Adversarial Robustness

Want to ensure: for a given input $\bar{x}_0$ and a given amount of noise δ, 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 δ, 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 δ, 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

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

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 $Q(N(x))$

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 $Q(N(x))$

NP-Hardness: by reduction from 3-SAT

Verification Complexity (cnt'd)

Verification Complexity (cnt'd)

Boolean variables: $x_1, \ldots, x_n$

Verification Complexity (cnt'd)

Boolean variables: $x_1, \ldots, x_n$

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

Verification Complexity (cnt'd)

Boolean variables: $x_1, \ldots, x_n$

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

Each clause C_i is $q_i^1 \vee q_i^2 \vee q_i^3$

Verification Complexity (cnt'd)

Boolean variables: $x_1, \ldots, x_n$

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

Each clause C_i is $q_i^1 \vee q_i^2 \vee q_i^3$

- q's are variables or their negations

Verification Complexity (cnt'd)

Boolean variables: $x_1, \ldots, x_n$

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

Each clause C_i is $q_i^1 \vee q_i^2 \vee q_i^3$

- 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: $C_1 \wedge C_2 \wedge \ldots \wedge C_k$

Each clause C_i is $q_i^1 \vee q_i^2 \vee q_i^3$

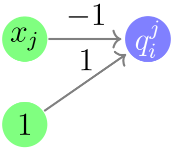
- q's are variables or their negations

Goal: find a variable assignment that satisfies the formula

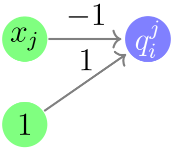
We will construct an input to the verification problem that is satisfiable iff the formula is satisfiable

Reduction: Handling Negations

Reduction: Handling Negations



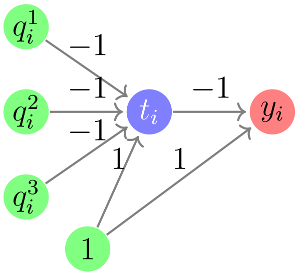
Reduction: Handling Negations



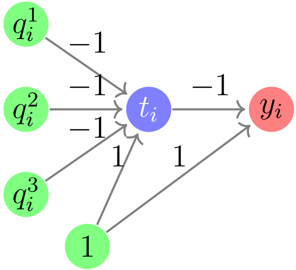
q_i^j gets $1 - x_j$, i.e. $q_i^j = \neg x_j$

Reduction: Handling Disjunctions

Reduction: Handling Disjunctions

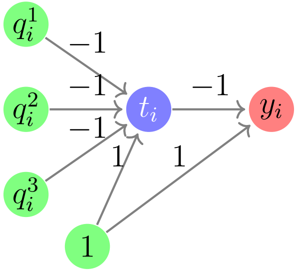


Reduction: Handling Disjunctions



At least one input is 1: t_i is 0, y_i is 1

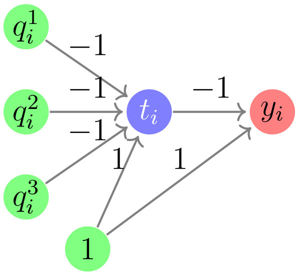
Reduction: Handling Disjunctions



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

Reduction: Handling Disjunctions



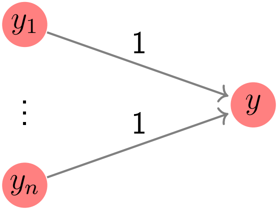
At least one input is 1: t_i is 0, y_i is 1

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

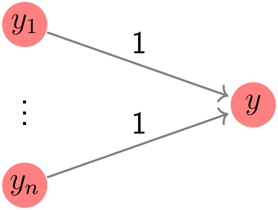
In other words: $y_i = q_i^1 \lor q_i^2 \lor q_i^3$

Reduction: Handling Conjunctions

Reduction: Handling Conjunctions

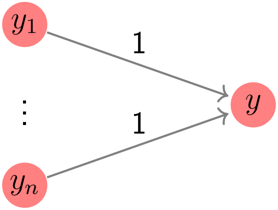


Reduction: Handling Conjunctions



y is the final output of the network

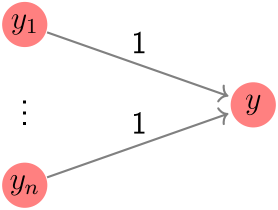
Reduction: Handling Conjunctions



y is the final output of the network

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

Reduction: Handling Conjunctions



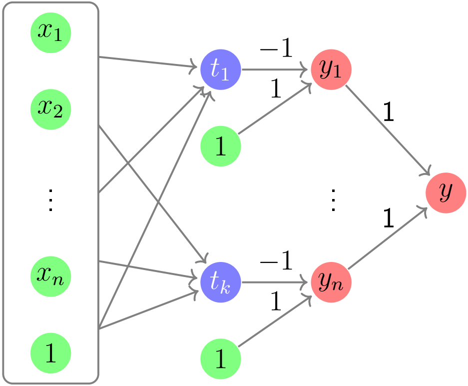
y is the final output of the network

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

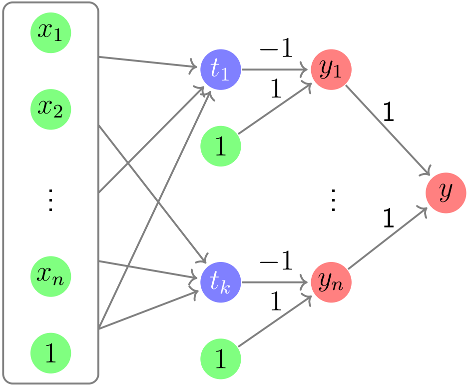
This is satisfied only if all conjuncts are 1

Reduction: Putting it all Together

Reduction: Putting it all Together

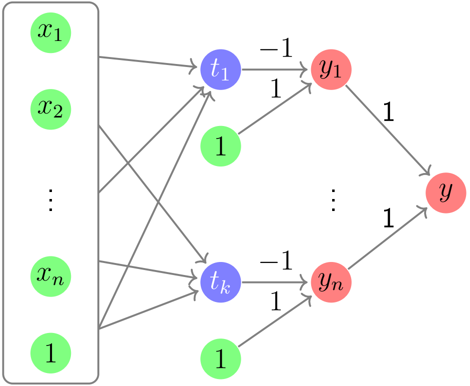


Reduction: Putting it all Together



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

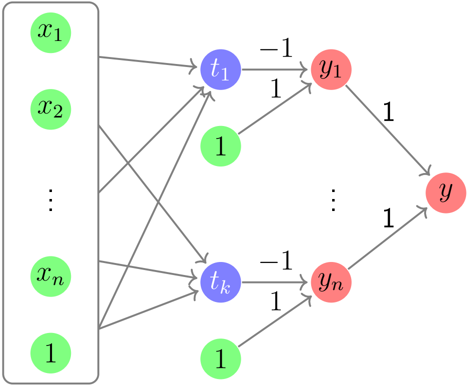
Reduction: Putting it all Together



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

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

Reduction: Putting it all Together



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

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

Verification property `SAT` iff original formula is `SAT`

Another Extension: Max-Pooling

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:

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:

- $\mathsf{ReLU}(x) = \max(x, 0)$
- $\max(x, y) = \mathsf{ReLU}(x - y) + y$

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:

- $\text{ReLU}(x) = \max(x, 0)$
- $\max(x, y) = \text{ReLU}(x - y) + y$

It is enough to solve just for ReLUs

Other piece-wise linear functions?

Non piece-wise linear functions?

Roadmap

Roadmap

Neural network verification is *hard*

Roadmap

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
- Real networks can be quite large

Techniques and Challenges

Techniques and Challenges

Main challenge is *scalability*

Techniques and Challenges

Main challenge is *scalability*

- Usually the case in verification

Techniques and Challenges

Main challenge is *scalability*

- Usually the case in verification

Two kinds of techniques:

Techniques and Challenges

Main challenge is *scalability*

- Usually the case in verification

Two kinds of techniques:

- *Sound* and *complete*:

Techniques and Challenges

Main challenge is *scalability*

- Usually the case in verification

Two kinds of techniques:

- *Sound* and *complete*:
  - limited scalability

Techniques and Challenges

Main challenge is *scalability*

- Usually the case in verification

Two kinds of techniques:

- *Sound* and *complete*:
  - limited scalability
  - always succeed

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*:

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

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”

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

NeVeR (Pulina and Tacchella, 2010) [PT10]

NeVeR (Pulina and Tacchella, 2010) [PT10]

Among first attempts to verify neural networks

NeVeR (Pulina and Tacchella, 2010) [PT10]

Among first attempts to verify neural networks

Focused on networks with Sigmoid activation functions

NeVeR (Pulina and Tacchella, 2010) [PT10]

Among first attempts to verify neural networks

Focused on networks with Sigmoid activation functions

Main idea: *over-approximate* Sigmoids using *interval arithmetic*

NeVeR (Pulina and Tacchella, 2010) [PT10]

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

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 *simpler* $\bar{S}$

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

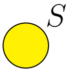
Core idea: replace a system S with a *simpler* $\bar{S}$

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

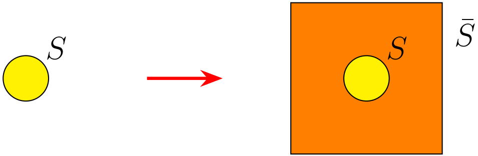
- But additional, *spurious* behaviors also exist in $\bar{S}$
- Because $\bar{S}$ is simpler, it is *easier to verify*

Over-Approximations (cnt'd)

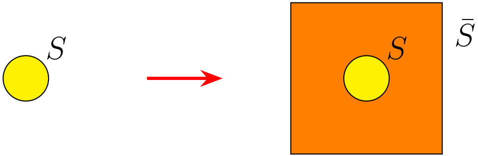
Over-Approximations (cnt'd)



Over-Approximations (cnt'd)

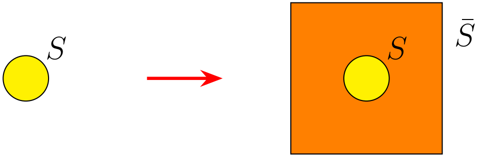


Over-Approximations (cnt'd)



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

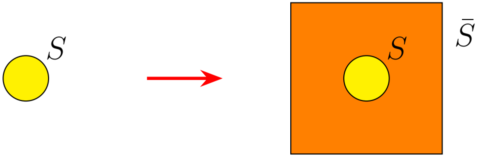
Over-Approximations (cnt'd)



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

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

Over-Approximations (cnt'd)

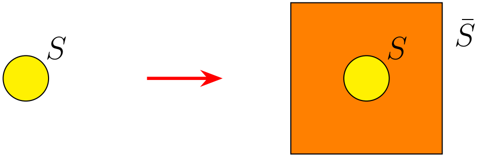


If $\bar{S}$ is correct, so is S

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

If $\bar{S}$ is incorrect:

Over-Approximations (cnt'd)



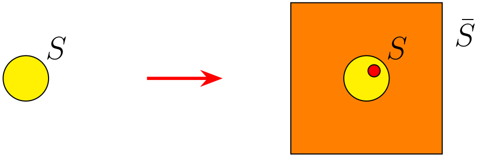
If $\bar{S}$ is correct, so is S

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

If $\bar{S}$ is incorrect:

- Either S is also incorrect

Over-Approximations (cnt'd)



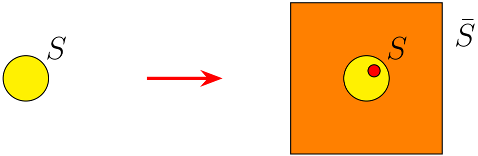
If $\bar{S}$ is correct, so is S

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

If $\bar{S}$ is incorrect:

- Either S is also incorrect

Over-Approximations (cnt'd)



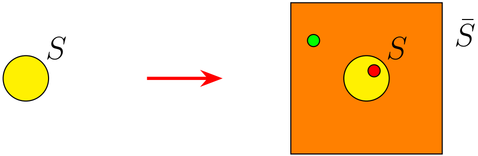
If $\bar{S}$ is correct, so is S

- 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

Over-Approximations (cnt'd)



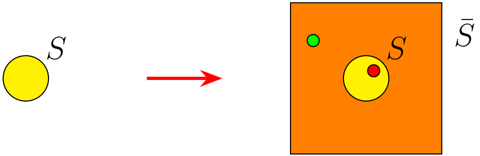
If $\bar{S}$ is correct, so is S

- 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

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:

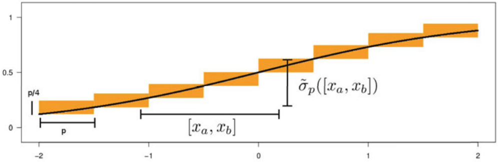
- Either S is also incorrect
- Or the detected bad behavior is spurious

If needed, $\bar{S}$ is *refined* to remove the spurious behavior, and the process is repeated

NeVeR (Pulina and Tacchella, 2010) [PT10]

NeVeR (Pulina and Tacchella, 2010) [PT10]

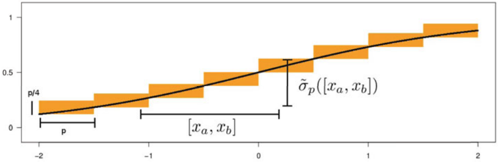
Abstraction used by Pulina and Tacchella:



$$f(s) = 1 / (1 + e^{-s})$$

NeVeR (Pulina and Tacchella, 2010) [PT10]

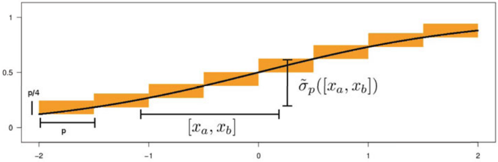
Abstraction used by Pulina and Tacchella:



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

NeVeR (Pulina and Tacchella, 2010) [PT10]

Abstraction used by Pulina and Tacchella:

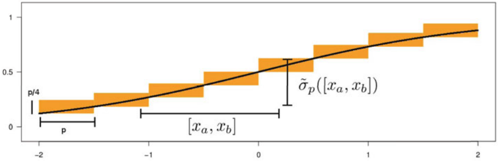


For $x \in [x_a, x_b]$ we just know that $f(x)$ is in some range $[y_a, y_b]$

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

NeVeR (Pulina and Tacchella, 2010) [PT10]

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)

Bastani et al, 2016 [BIL+16]

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*

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*

Did not directly study verification

Bastani et al, 2016 [BIL$^+$16] NIPS 2016

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)

Linear Programming (LP)

A linear program:

Linear Programming (LP)

A linear program:

$$\begin{array}{lr} \text{minimize} & \bar{c} \cdot \bar{x} \\ \text{subject to} & A \cdot \bar{x} = \bar{b} \\ \text{and} & \bar{l} \leq \bar{x} \leq \bar{u} \end{array}$$

Linear Programming (LP)

A linear program:

$$
\begin{array}{ll}
\text{minimize} & \bar{c} \cdot \bar{x} \\
\text{subject to} & A \cdot \bar{x} = \bar{b} \\
\text{and} & \bar{l} \leq \bar{x} \leq \bar{u}
\end{array}
$$

Intuitively:

Linear Programming (LP)

A linear program:

$$\begin{array}{lr} \text{minimize} & \bar{c} \cdot \bar{x} \\ \text{subject to} & A \cdot \bar{x} = \bar{b} \\ \text{and} & \bar{l} \leq \bar{x} \leq \bar{u} \end{array}$$

Intuitively:

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

Linear Programming (LP)

A linear program:

$$\begin{array}{ll} \text{minimize} & \bar{c} \cdot \bar{x} \\ \text{subject to} & A \cdot \bar{x} = \bar{b} \\ \text{and} & \bar{l} \leq \bar{x} \leq \bar{u} \end{array}$$

Intuitively:

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

Linear Programming (LP)

A linear program:

$$\begin{array}{lr} \text{minimize} & \bar{c} \cdot \bar{x} \\ \text{subject to} & A \cdot \bar{x} = \bar{b} \\ \text{and} & \bar{l} \leq \bar{x} \leq \bar{u} \end{array}$$

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

Linear Programming (LP)

A linear program:

$$\begin{array}{lr} \text{minimize} & \bar{c} \cdot \bar{x} \\ \text{subject to} & A \cdot \bar{x} = \bar{b} \\ \text{and} & \bar{l} \leq \bar{x} \leq \bar{u} \end{array}$$

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

Linear Programming (LP)

A linear program:

$$\begin{array}{ll} \text{minimize} & \bar{c} \cdot \bar{x} \\ \text{subject to} & A \cdot \bar{x} = \bar{b} \\ \text{and} & \bar{l} \leq \bar{x} \leq \bar{u} \end{array}$$

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 $\mathbf{P}$, powerful solvers exist

Replacing ReLUs with Linear Constraints

Replacing ReLUs with Linear Constraints

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

Replacing ReLUs with Linear Constraints

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

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

Replacing ReLUs with Linear Constraints

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

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

Replacing ReLUs with Linear Constraints

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

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

Each phase is a *linear* constraint

Replacing ReLUs with Linear Constraints

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

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

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: $(x \geq 0) \wedge (y = x)$
- *Inactive* phase: $(x \leq 0) \wedge (y = 0)$

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

Bastani et al, 2016 [BIL+16] (cnt'd)

Bastani et al, 2016 [BIL+16] (cnt'd)

To look for adversarial inputs around a point $\bar{x}_0$:

Bastani et al, 2016 [BIL+16] (cnt'd)

To look for adversarial inputs around a point $\bar{x}_0$:

- Encode the network's weighted sums as linear equations

Bastani et al, 2016 [BIL+16] (cnt'd)

To look for adversarial inputs around a point $\bar{x}_0$:

- Encode the network's weighted sums as linear equations
- Evaluate the network for $\bar{x}_0$

Bastani et al, 2016 [BIL+16] (cnt'd)

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 $y = \text{ReLU}(x)$:

Bastani et al, 2016 [BIL$^+$16] (cnt'd)

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 $(x \geq 0) \wedge (y = x)$

Bastani et al, 2016 [BIL$^+$16] (cnt'd)

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 $(x \geq 0) \wedge (y = x)$
  - If it is *inactive*, replace it with $(x \leq 0) \wedge (y = 0)$

Bastani et al, 2016 [BIL+16] (cnt'd)

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 $(x \geq 0) \wedge (y = x)$
  - If it is *inactive*, replace it with $(x \leq 0) \wedge (y = 0)$
- Have an LP solver look for adversarial inputs

Bastani et al, 2016 [BIL+16] (cnt'd)

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 $(x \geq 0) \wedge (y = x)$
  - If it is *inactive*, replace it with $(x \leq 0) \wedge (y = 0)$
- Have an LP solver look for adversarial inputs

Evaluated on image recognition networks

Bastani et al, 2016 [BIL$^+$16] (cnt'd)

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 $(x \geq 0) \wedge (y = x)$
    - If it is *inactive*, replace it with $(x \leq 0) \wedge (y = 0)$
- Have an LP solver look for adversarial inputs

Evaluated on image recognition networks

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

Bastani et al, 2016 [BIL+16] (cnt'd)

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 $(x \geq 0) \wedge (y = x)$
  - If it is *inactive*, replace it with $(x \leq 0) \wedge (y = 0)$
- 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

Bastani et al, 2016 [BIL+16] (cnt'd)

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 $(x \geq 0) \wedge (y = x)$
    - If it is *inactive*, replace it with $(x \leq 0) \wedge (y = 0)$
- 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

Reducing Verification to Linear Programming

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

Reducing Verification to Linear Programming

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

Case splitting: an enumeration of all possibilities:

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

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

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`

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

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`

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)

Reducing Verification to Linear Programming (cnt'd)

Case splitting creates a *search tree*

Reducing Verification to Linear Programming (cnt'd)

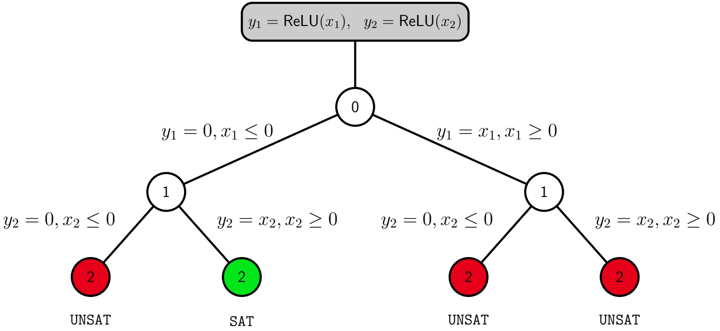
Case splitting creates a *search tree*

Problem is `SAT` iff at least one leaf is `SAT`

Reducing Verification to Linear Programming (cnt'd)

Case splitting creates a *search tree*

Problem is `SAT` iff at least one leaf is `SAT`



Reducing Verification to Linear Programming (cnt'd)

Reducing Verification to Linear Programming (cnt'd)

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

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:

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]

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)

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]

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)

Table of Contents

Reluplex

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

Reluplex (cnt'd)

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

An algorithm for solving linear programs

Simplex

An algorithm for solving linear programs

- Linear equations

Simplex

An algorithm for solving linear programs

- Linear equations
- Variable bounds

Simplex

An algorithm for solving linear programs

- Linear equations
- Variable bounds
- Objective function

Simplex

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:

Simplex (cnt'd)

Divided into two phases:

- Find a feasible solution

Simplex (cnt'd)

Divided into two phases:

- Find a feasible solution
- Optimize with respect to objective function

Simplex (cnt'd)

Divided into two phases:

- Find a feasible solution
- Optimize with respect to objective function

Focus on phase 1, which is just a *satisfiability check*

Simplex: Phase 1

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*

-- bring variables closer to their bound

Simplex: Basics and Non-Basics

Simplex: Basics and Non-Basics

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"

Simplex: Basics and Non-Basics

Variables partitioned into *basic* and *non-basic* variables

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

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

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

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

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

an *update*: change the assignment of a non-basic variable

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

an *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

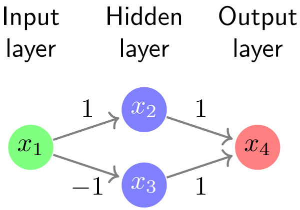
an *update*: change the assignment of a non-basic variable

- and any affected basics

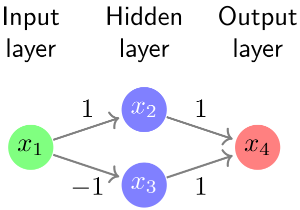
a *pivot*: switch a basic and non-basic variable

Simplex: Example

Simplex: Example

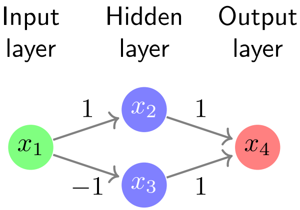


Simplex: Example



No activation functions

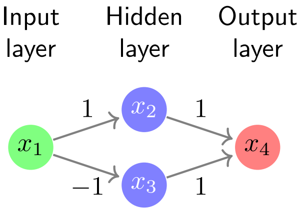
Simplex: Example



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 $x_4 \notin [0.5, 1]$

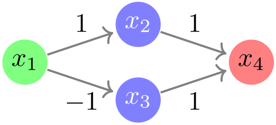
- Negated output property: $x_1 \in [0, 1]$ and $x_4 \in [0.5, 1]$

x4 = x2 + x3 = x1 - x1 = 0 ==> The original property holds

(Negated property UNSAT)

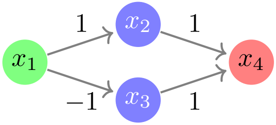
Simplex: Example (cnt'd)

Simplex: Example (cnt'd)

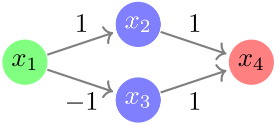


Simplex: Example (cnt'd)

Equations for weighted sums:



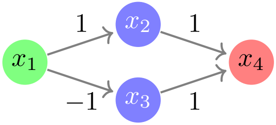
Simplex: Example (cnt'd)



Equations for weighted sums:

$$x_2 - x_1 = 0$$
$$x_3 + x_1 = 0$$
$$x_4 - x_3 - x_2 = 0$$

Simplex: Example (cnt'd)

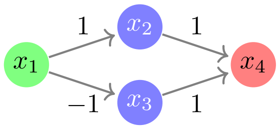


Equations for weighted sums:

$$x_2 - x_1 = 0$$
$$x_3 + x_1 = 0$$
$$x_4 - x_3 - x_2 = 0$$

Bounds:

Simplex: Example (cnt'd)



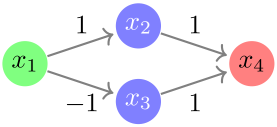
Equations for weighted sums:

$$x_2 - x_1 = 0$$
$$x_3 + x_1 = 0$$
$$x_4 - x_3 - x_2 = 0$$

Bounds:

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

Simplex: Example (cnt'd)



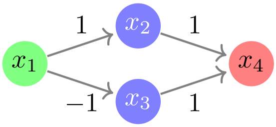
Equations for weighted sums:

$$x_2 - x_1 = 0$$
$$x_3 + x_1 = 0$$
$$x_4 - x_3 - x_2 = 0$$

Bounds:

$$x_1 \in [0, 1]$$
$$x_4 \in [0.5, 1]$$
x_2, x_3 unbounded

Technicality: replace constants by *auxiliary* variables



Equations for weighted sums:

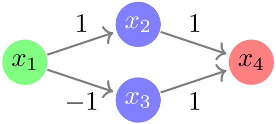
$$x_2 - x_1 = 0$$
$$x_3 + x_1 = 0$$
$$x_4 - x_3 - x_2 = 0$$

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)

Simplex: Example (cnt'd)

x5, x6 and x7 are basic
Non-basic can change

$$x_5 = x_2 - x_1$$
$$x_6 = x_3 + x_1$$
$$x_7 = x_4 - x_3 - x_2$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

Simplex: Example (cnt'd)

$$x_5 = x_2 - x_1$$
$$x_6 = x_3 + x_1$$
$$x_7 = x_4 - x_3 - x_2$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

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:
$x_4 := x_4 + 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

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:

$x_4 := x_4 + 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

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:
$x_4 := x_4 + 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

Simplex: Example (cnt'd)

$$x_5 = x_2 - x_1$$
$$x_6 = x_3 + x_1$$
$$x_7 = x_4 - x_3 - x_2$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

Simplex: Example (cnt'd)

$$x_5 = x_2 - x_1$$
$$x_6 = x_3 + x_1$$
$$x_7 = x_4 - x_3 - x_2$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

Simplex: Example (cnt'd)

$$x_5 = x_2 - x_1$$
$$x_6 = x_3 + x_1$$
$$x_7 = x_4 - x_3 - x_2$$

Pivot: x_7, x_2

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

Simplex: Example (cnt'd)

$$x_5 = x_2 - x_1$$
$$x_6 = x_3 + x_1$$
$$x_7 = x_4 - x_3 - x_2 \qquad \leftarrow \qquad x_2 = x_4 - x_3 - x_7$$

Pivot: x_7, x_2

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

Simplex: Example (cnt'd)

$$x_5 = x_2 - x_1 \qquad \leftarrow \qquad x_5 = x_4 - x_3 - x_7 - x_1$$

$$x_6 = x_3 + x_1$$

$$x_7 = x_4 - x_3 - x_2 \qquad \leftarrow \qquad x_2 = x_4 - x_3 - x_7$$

Pivot: x_7, x_2

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

Simplex: Example (cnt'd)

$$x_5 = x_4 - x_3 - x_7 - x_1$$

$$x_6 = x_3 + x_1$$

$$x_2 = x_4 - x_3 - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

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:

$x_7 := x_7 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

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:
$x_7 := x_7 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

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:

$x_7 := x_7 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

Simplex: Example (cnt'd)

$$x_5 = x_4 - x_3 - x_7 - x_1$$

$$x_6 = x_3 + x_1$$

$$x_2 = x_4 - x_3 - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

Simplex: Example (cnt'd)

$$x_5 = x_4 - x_3 - x_7 - x_1$$
$$x_6 = x_3 + x_1$$
$$x_2 = x_4 - x_3 - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

Simplex: Example (cnt'd)

$$x_5 = x_4 - x_3 - x_7 - x_1 \qquad \leftarrow \qquad x_1 = x_4 - x_3 - x_7 - x_5$$

$$x_6 = x_3 + x_1$$

$$x_2 = x_4 - x_3 - x_7$$

Pivot: x_5, x_1

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

Simplex: Example (cnt'd)

$$x_1 = x_4 - x_3 - x_7 - x_5$$
$$x_6 = x_4 - x_7 - x_5$$
$$x_2 = x_4 - x_3 - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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:
$x_5 := x_5 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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:

$x_5 := x_5 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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:

$x_5 := x_5 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

Simplex: Example (cnt'd)

$$x_1 = x_4 - x_3 - x_7 - x_5$$

$$x_6 = x_4 - x_7 - x_5$$

$$x_2 = x_4 - x_3 - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

Simplex: Example (cnt'd)

$$x_1 = x_4 - x_3 - x_7 - x_5$$
$$x_6 = x_4 - x_7 - x_5$$
$$x_2 = x_4 - x_3 - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

Simplex: Example (cnt'd)

$$x_1 = x_4 - x_3 - x_7 - x_5$$
$$x_6 = x_4 - x_7 - x_5$$
$$x_2 = x_4 - x_3 - x_7$$

Failure

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2	0.5	
	x_3	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

The Simplex Calculus

The Simplex Calculus

A simplex configuration:

The Simplex Calculus

A simplex configuration:

- Distinguished symbols `SAT` or `UNSAT`

The Simplex Calculus

A simplex configuration:

- Distinguished symbols `SAT` or `UNSAT`
- Or a tuple $\langle \mathcal{B}, T, l, u, \alpha \rangle$, where:

The Simplex Calculus

A simplex configuration:

- Distinguished symbols `SAT` or `UNSAT`
- Or a tuple $\langle \mathcal{B}, T, l, u, \alpha \rangle$, where:
  - $\mathcal{B}$: set of basic variables

The Simplex Calculus

A simplex configuration:

- Distinguished symbols `SAT` or `UNSAT`
- Or a tuple $\langle \mathcal{B}, T, l, u, \alpha \rangle$, where:
  - $\mathcal{B}$: set of basic variables
  - T: a set of equations

The Simplex Calculus

A simplex configuration:

- Distinguished symbols `SAT` or `UNSAT`
- Or a tuple $\langle \mathcal{B}, T, l, u, \alpha \rangle$, where:
    - $\mathcal{B}$: set of basic variables
    - T: a set of equations
    - l, u: lower and upper bounds

The Simplex Calculus

A simplex configuration:

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

The Simplex Calculus

A simplex configuration:

- Distinguished symbols `SAT` or `UNSAT`
- Or a tuple $\langle \mathcal{B}, T, l, u, \alpha \rangle$, where:
    - $\mathcal{B}$: 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:

The Simplex Calculus

A simplex configuration:

- Distinguished symbols `SAT` or `UNSAT`
- Or a tuple $\langle \mathcal{B}, T, l, u, \alpha \rangle$, where:
  - $\mathcal{B}$: 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:

$$\text{slack}^+(x_i) = \{x_j \notin \mathcal{B} \mid (T_{i,j} > 0 \wedge \alpha(x_j) < u(x_j)) \vee (T_{i,j} < 0 \wedge \alpha(x_j) > l(x_j))$$
$$\text{slack}^-(x_i) = \{x_j \notin \mathcal{B} \mid (T_{i,j} < 0 \wedge \alpha(x_j) < u(x_j)) \vee (T_{i,j} > 0 \wedge \alpha(x_j) > l(x_j))$$

The Simplex Calculus (cnt'd)

The Simplex Calculus (cnt'd)

$$\text{Pivot}_1 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) < l(x_i), \quad x_j \in \text{slack}^+(x_i)}{T := \textit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

The Simplex Calculus (cnt'd)

$$\text{Pivot}_1 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) < l(x_i), \quad x_j \in \mathsf{slack}^+(x_i)}{T := \textit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Pivot}_2 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) > u(x_i), \quad x_j \in \mathsf{slack}^-(x_i)}{T := \textit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

The Simplex Calculus (cnt'd)

$$\text{Pivot}_1 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) < l(x_i), \quad x_j \in \mathsf{slack}^+(x_i)}{T := \textit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Pivot}_2 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) > u(x_i), \quad x_j \in \mathsf{slack}^-(x_i)}{T := \textit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Update} \quad \frac{x_j \notin \mathcal{B}, \quad \alpha(x_j) < l(x_j) \vee \alpha(x_j) > u(x_j), \quad l(x_j) \leq \alpha(x_j) + \delta \leq u(x_j)}{\alpha := \textit{update}(\alpha, x_j, \delta)}$$

The Simplex Calculus (cnt'd)

$$\text{Pivot}_1 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) < l(x_i), \quad x_j \in \textsf{slack}^+(x_i)}{T := \textsf{\textit{pivot}}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Pivot}_2 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) > u(x_i), \quad x_j \in \textsf{slack}^-(x_i)}{T := \textsf{\textit{pivot}}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Update} \quad \frac{x_j \notin \mathcal{B}, \quad \alpha(x_j) < l(x_j) \vee \alpha(x_j) > u(x_j), \quad l(x_j) \leq \alpha(x_j) + \delta \leq u(x_j)}{\alpha := \textsf{\textit{update}}(\alpha, x_j, \delta)}$$

$$\text{Failure} \quad \frac{x_i \in \mathcal{B}, \quad (\alpha(x_i) < l(x_i) \ \wedge \ \textsf{slack}^+(x_i) = \emptyset) \vee (\alpha(x_i) > u(x_i) \ \wedge \ \textsf{slack}^-(x_i) = \emptyset)}{\texttt{UNSAT}}$$

The Simplex Calculus (cnt'd)

$$\text{Pivot}_1 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) < l(x_i), \quad x_j \in \mathsf{slack}^+(x_i)}{T := \textit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Pivot}_2 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) > u(x_i), \quad x_j \in \mathsf{slack}^-(x_i)}{T := \textit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Update} \quad \frac{x_j \notin \mathcal{B}, \quad \alpha(x_j) < l(x_j) \vee \alpha(x_j) > u(x_j), \quad l(x_j) \le \alpha(x_j) + \delta \le u(x_j)}{\alpha := \textit{update}(\alpha, x_j, \delta)}$$

$$\text{Failure} \quad \frac{x_i \in \mathcal{B}, \quad (\alpha(x_i) < l(x_i) \ \wedge \ \mathsf{slack}^+(x_i) = \emptyset) \vee (\alpha(x_i) > u(x_i) \ \wedge \ \mathsf{slack}^-(x_i) = \emptyset)}{\texttt{UNSAT}}$$

$$\text{Success} \quad \frac{\forall x_i \in \mathcal{X}.\ l(x_i) \le \alpha(x_i) \le u(x_i)}{\texttt{SAT}}$$

Properties of Simplex

Properties of Simplex

Theorem (Soundness and Completeness of Simplex)

*The simplex algorithm is sound and complete**

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:

- `SAT` $\Rightarrow$ assignment is correct

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

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

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` $\Rightarrow$ 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

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

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` $\Rightarrow$ 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*)

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 $\mathbf{P}$, unknown whether simplex is in $\mathbf{P}$

From Simplex to Reluplex

From Simplex to Reluplex

Each ReLU node x represented as two variables:

From Simplex to Reluplex

Each ReLU node x represented as two variables:

- x^w to represent the (input) *weighted sum*

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*

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

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

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

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*

From Simplex to Reluplex

Each ReLU node x represented as two variables:

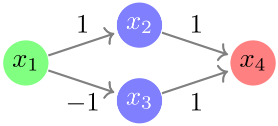
- x^w to represent the (input) *weighted sum*
- x^a to represent the (output) *activation result*

x^w and x^a change independently

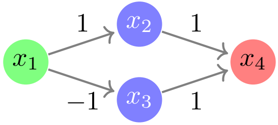
- May violate ReLU constraint
- Similar to bound constraints
- Fix *incrementally*

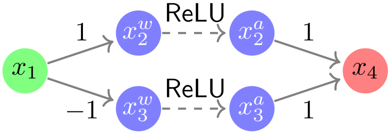
Use pivots and updates, same as before

Reluplex: Example



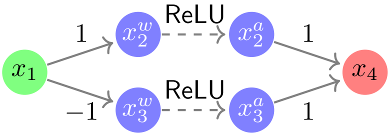
Reluplex: Example





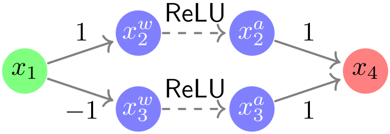
Reluplex: Example (cnt'd)

Reluplex: Example (cnt'd)

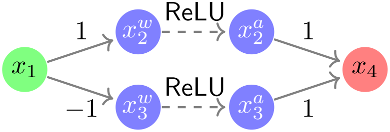


Reluplex: Example (cnt'd)

Equations for weighted sums:



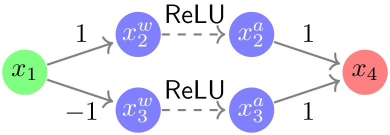
Reluplex: Example (cnt'd)



Equations for weighted sums:

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_7 = x_4 - x_3^a - x_2^a$$

Reluplex: Example (cnt'd)

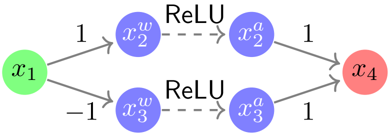


Equations for weighted sums:

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_7 = x_4 - x_3^a - x_2^a$$

Bounds:

Reluplex: Example (cnt'd)



Assignment that satisfies RELU along with bounds <==> NN satisfies property

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 unbounded
$$x_2^a, x_3^a \in [0, \infty)$$
$$x_5, x_6, x_7 \in [0, 0]$$

Reluplex: Example (cnt'd)

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_7 = x_4 - x_3^a - x_2^a$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_7 = x_4 - x_3^a - x_2^a$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

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:

$x_4 := x_4 + 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

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:

$x_4 := x_4 + 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

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:
$x_4 := x_4 + 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_7 = x_4 - x_3^a - x_2^a$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_7 = x_4 - x_3^a - x_2^a$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

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

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

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

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

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

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

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:

$x_7 := x_7 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

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:
$x_7 := x_7 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0.5	0

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:

$x_7 := x_7 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

Update:
$x_2^w := x_2^w + 0.5$

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

Update:
$x_2^w := x_2^w + 0.5$

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

Update:
$x_2^w := x_2^w + 0.5$

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_5 = x_2^w - x_1$$
$$x_6 = x_3^w + x_1$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_1 = x_2^w - x_5$$
$$x_6 = x_3^w + x_2^w - x_5$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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:

$x_5 := x_5 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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:

$x_5 := x_5 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0	0
0	x_7	0	0

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:

$x_5 := x_5 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_1 = x_2^w - x_5$$
$$x_6 = x_3^w + x_2^w - x_5$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_1 = x_2^w - x_5$$
$$x_6 = x_3^w + x_2^w - x_5$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_1 = x_2^w - x_5$$
$$x_6 = x_3^w + x_2^w - x_5$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

Pivot: x_6, x_3^w

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

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_1 = x_2^w - x_5$$
$$x_3^w = x_6 - x_2^w + x_5$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

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:

$x_6 := x_6 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

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:

$x_6 := x_6 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	0	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0.5	0
0	x_7	0	0

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:

$x_6 := x_6 - 0.5$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	-0.5	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

Reluplex: Example (cnt'd)

$$x_1 = x_2^w - x_5$$
$$x_3^w = x_6 - x_2^w + x_5$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	-0.5	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

Reluplex: Example (cnt'd)

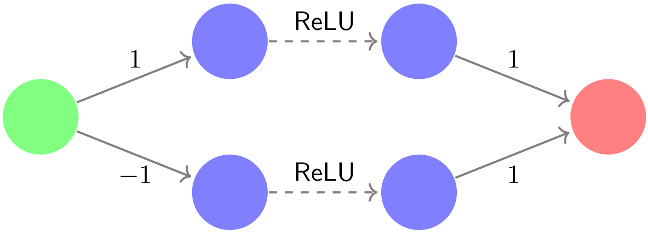
$$x_1 = x_2^w - x_5$$
$$x_3^w = x_6 - x_2^w + x_5$$
$$x_2^a = x_4 - x_3^a - x_7$$

Lower B.	Var	Value	Upper B.
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	-0.5	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0	0
0	x_6	0	0
0	x_7	0	0

Success

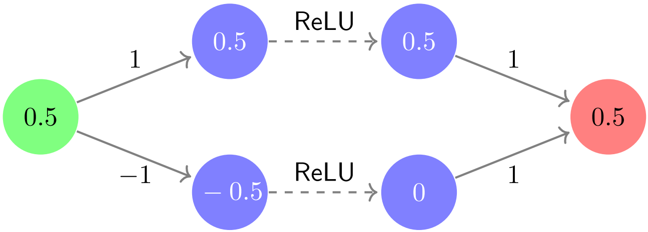
Reluplex: Example (cnt'd)

Reluplex: Example (cnt'd)



Property: $x_1 \in [0, 1]$ and $x_4 \in [0.5, 1]$

Reluplex: Example (cnt'd)



Property: $x_1 \in [0, 1]$ and $x_4 \in [0.5, 1]$

The Reluplex Calculus

The Reluplex Calculus

A Reluplex configuration:

The Reluplex Calculus

A Reluplex configuration:

- Distinguished symbols `SAT` or `UNSAT`

The Reluplex Calculus

A Reluplex configuration:

- Distinguished symbols `SAT` or `UNSAT`
- Or a tuple $\langle \mathcal{B}, T, l, u, \alpha, R \rangle$, where:

The Reluplex Calculus

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

The Reluplex Calculus

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

The Reluplex Calculus

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

The Reluplex Calculus

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
  - α: an assignment function from variables to reals

The Reluplex Calculus

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
  - α: 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)

The Reluplex Calculus (cnt'd)

Pivot_1, Pivot_2, Update and Failure are as before

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`

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`

$$\text{Update}_w \quad \frac{x_i \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max\left(0, \alpha(x_i)\right), \quad \alpha(x_j) \geq 0}{\alpha := \textit{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}$$

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`

$$\text{Update}_w \quad \frac{x_i \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i)), \quad \alpha(x_j) \geq 0}{\alpha := \textit{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}$$

$$\text{Update}_a \quad \frac{x_j \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := \textit{update}(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}$$

The Reluplex Calculus (cnt'd)

Pivot$_1$, Pivot$_2$, Update and Failure are as before

$\texttt{SAT}$ iff at least one leaf of the derivation tree is $\texttt{SAT}$

$$\text{Update}_w \quad \frac{x_i \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i)), \quad \alpha(x_j) \geq 0}{\alpha := \textit{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}$$

$$\text{Update}_a \quad \frac{x_j \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := \textit{update}(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}$$

$$\text{PivotForRelu} \quad \frac{x_i \in \mathcal{B}, \quad \exists x_l.\ \langle x_i, x_l \rangle \in R \vee \langle x_l, x_i \rangle \in R, \quad x_j \notin \mathcal{B}, \quad T_{i,j} \neq 0}{T := \textit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

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`

$$\text{Update}_w \quad \frac{x_i \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i)), \quad \alpha(x_j) \geq 0}{\alpha := \textit{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}$$

$$\text{Update}_a \quad \frac{x_j \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := \textit{update}(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}$$

$$\text{PivotForRelu} \quad \frac{x_i \in \mathcal{B}, \quad \exists x_l.\ \langle x_i, x_l \rangle \in R \vee \langle x_l, x_i \rangle \in R, \quad x_j \notin \mathcal{B}, \quad T_{i,j} \neq 0}{T := \textit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{ReluSplit} \quad \frac{\langle x_i, x_j \rangle \in R, \quad l(x_i) < 0, \quad u(x_i) > 0}{u(x_i) := 0 \qquad l(x_i) := 0}$$

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`

$$\text{Update}_w \quad \frac{x_i \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i)), \quad \alpha(x_j) \geq 0}{\alpha := \textit{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}$$

$$\text{Update}_a \quad \frac{x_j \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := \textit{update}(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}$$

$$\text{PivotForRelu} \quad \frac{x_i \in \mathcal{B}, \quad \exists x_l.\ \langle x_i, x_l \rangle \in R \vee \langle x_l, x_i \rangle \in R, \quad x_j \notin \mathcal{B}, \quad T_{i,j} \neq 0}{T := \textit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{ReluSplit} \quad \frac{\langle x_i, x_j \rangle \in R, \quad l(x_i) < 0, \quad u(x_i) > 0}{u(x_i) := 0 \qquad l(x_i) := 0}$$

$$\text{ReluSuccess} \quad \frac{\forall x \in \mathcal{X}.\ l(x) \leq \alpha(x) \leq u(x), \quad \forall \langle x^w, x^a \rangle \in R.\ \alpha(x^a) = \max(0, \alpha(x^w))}{\texttt{SAT}}$$

Properties of Reluplex

Properties of Reluplex

Theorem (Soundness and Completeness of Reluplex)

*The Reluplex algorithm is sound and complete**

Properties of Reluplex

Theorem (Soundness and Completeness of Reluplex)

*The Reluplex algorithm is sound and complete**

Soundness:

Properties of Reluplex

Theorem (Soundness and Completeness of Reluplex)

*The Reluplex algorithm is sound and complete**

Soundness:

- $\texttt{SAT} \Rightarrow$ assignment is correct

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

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*

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

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

More Efficient Reluplex

Better approach: *lazy splitting*

More Efficient Reluplex

Better approach: *lazy splitting*

- Start fixing bound violations

More Efficient Reluplex

Better approach: *lazy splitting*

- Start fixing bound violations
- Once all variables within bounds, address broken ReLUs

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

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

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

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%)

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: Bound Tightening

More Efficient Reluplex: Bound Tightening

During execution we encounter many equations

More Efficient Reluplex: Bound Tightening

During execution we encounter many equations

Can use them for *bound tightening*

More Efficient Reluplex: Bound Tightening

During execution we encounter many equations

Can use them for *bound tightening*

Example:

$$x = y + z \qquad\qquad x \geq -2, \quad y \geq 1, \quad z \geq 1$$

More Efficient Reluplex: Bound Tightening

During execution we encounter many equations

Can use them for *bound tightening*

Example:

$$x = y + z \qquad\qquad x \geq -2, \quad y \geq 1, \quad z \geq 1$$

Can derive *tighter* bound: $x \geq 2$

More Efficient Reluplex: Bound Tightening

During execution we encounter many equations

Can use them for *bound tightening*

Example:

$$x = y + z \qquad\qquad x \geq -2, \quad y \geq 1, \quad z \geq 1$$

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:

$$x = y + z \qquad\qquad x \geq -2, \quad y \geq 1, \quad z \geq 1$$

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:

$$x = y + z \qquad\qquad x \geq -2, \quad y \geq 1, \quad z \geq 1$$

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

More Efficient Reluplex: Bound Tightening (cnt'd)

More Efficient Reluplex: Bound Tightening (cnt'd)

In every pivot step we examine an equation

More Efficient Reluplex: Bound Tightening (cnt'd)

In every pivot step we examine an equation

Use that equation for bound tightening

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

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?

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?
- Complexity: linear in the size of the equation

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?
- Complexity: linear in the size of the equation

Particularly useful after splitting

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?
- Complexity: linear in the size of the equation

Particularly useful after splitting

- Because new bounds have been introduced

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?
- 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)

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)

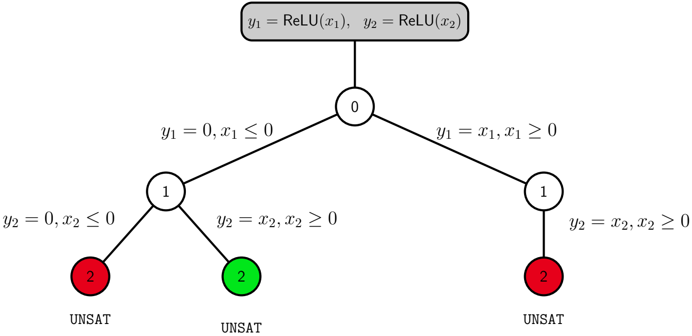
A useful technique in SAT and SMT solving

Backtracking: change *last* guess

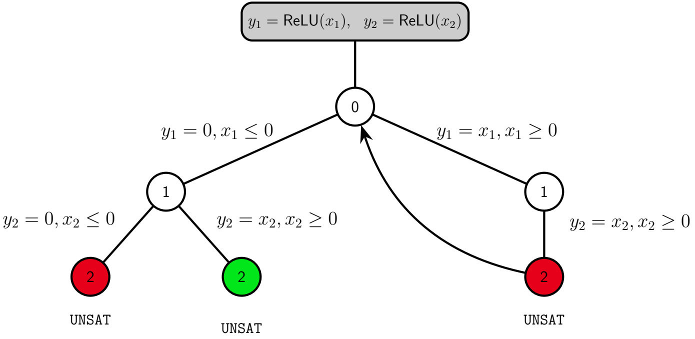
Backjumping: change an *earlier* guess

Need to keep track of the discovery of new bounds

Non-Chronological Backtracking (Backjumping) (cnt'd)



Non-Chronological Backtracking (Backjumping) (cnt'd)



Precision and Numerical Stability

Precision and Numerical Stability

SMT solvers typically use *precise arithmetic*

Precision and Numerical Stability

SMT solvers typically use *precise arithmetic*

- This ensures soundness

Precision and Numerical Stability

SMT solvers typically use *precise arithmetic*

- This ensures soundness
- But is quite slow

Precision and Numerical Stability

SMT solvers typically use *precise arithmetic*

- This ensures soundness
- But is quite slow

LP solvers typically use *floating point arithmetic*

Precision and Numerical Stability

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

Precision and Numerical Stability

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

Precision and Numerical Stability

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

Precision and Numerical Stability

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

Should do the same when implementing Reluplex

Precision and Numerical Stability (cnt'd)

Precision and Numerical Stability (cnt'd)

Can monitor numerical instability

Precision and Numerical Stability (cnt'd)

Can monitor numerical instability

- Plug current assignment into input formulas

Precision and Numerical Stability (cnt'd)

Can monitor numerical instability

- Plug current assignment into input formulas
- Measure the error

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

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

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

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

Roadmap

Roadmap

The *simplex* algorithm, for solving linear programs

Roadmap

The *simplex* algorithm, for solving linear programs

Extension into *Reluplex*, for solving linear programs + ReLUs

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

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

Roadmap

The *simplex* algorithm, for solving linear programs

Extension into *Reluplex*, for solving linear programs + ReLUs

Some highlights for an efficient implementation

The ACAS Xu System

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)

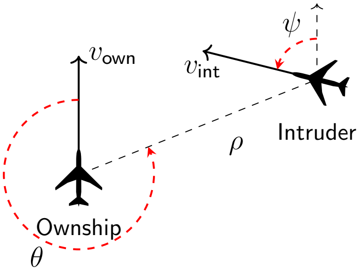
The ACAS Xu System

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*



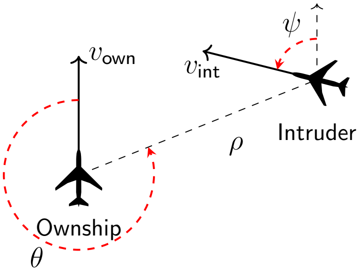
The ACAS Xu System

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

Certifying ACAS Xu

There are properties that the FAA cares about

Certifying ACAS Xu

There are properties that the FAA cares about

- Consistent alerting regions

Certifying ACAS Xu

There are properties that the FAA cares about

- Consistent alerting regions
- No unnecessary turning advisories

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

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

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

Certifying ACAS Xu (cnt'd)

Certifying ACAS Xu (cnt'd)

We worked on a list of 10 properties

Certifying ACAS Xu (cnt'd)

We worked on a list of 10 properties

Example 1:

Certifying ACAS Xu (cnt'd)

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)

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)

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: $0.2 \leq \theta \leq 0.4$

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.

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

Certifying ACAS Xu (cnt'd)

Certifying ACAS Xu (cnt'd)

Example 2:

Certifying ACAS Xu (cnt'd)

Example 2:

- If vertical separation is large and the previous advisory is weak left, the network advises clear-of-conflict or weak left

Certifying ACAS Xu (cnt'd)

Example 2:

- If vertical separation is large and the previous advisory is weak left, the network advises clear-of-conflict or weak left
  - Distance: $0 \leq \rho \leq 60760$

Certifying ACAS Xu (cnt'd)

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: $\tau = 100$

Certifying ACAS Xu (cnt'd)

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.

Certifying ACAS Xu (cnt'd)

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

Certifying ACAS Xu (cnt'd)

Certifying ACAS Xu (cnt'd)

	Networks	Result	Time	Stack	Splits
ϕ_1	41	UNSAT	394517	47	1522384
	4	TIMEOUT			
ϕ_2	1	UNSAT	463	55	88388
	35	SAT	82419	44	284515
ϕ_3	42	UNSAT	28156	22	52080
ϕ_4	42	UNSAT	12475	21	23940
ϕ_5	1	UNSAT	19355	46	58914
ϕ_6	1	UNSAT	180288	50	548496
ϕ_7	1	TIMEOUT			
ϕ_8	1	SAT	40102	69	116697
ϕ_9	1	UNSAT	99634	48	227002
ϕ_{10}	1	UNSAT	19944	49	88520

Ongoing Work in the Reluplex Project

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*?

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

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`?

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)

Ongoing Work in the Reluplex Project (cnt'd)

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?

O Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi.
Measuring Neural Net Robustness with Constraints.
In *Proc. 30th Conf. on Neural Information Processing Systems (NIPS)*, 2016.

R. Bunel, I. Turksaslan, P. Torr, P. Kohli, and P. Kumar.
Piecewise Linear Neural Network Verification: A Comparative Study, 2017.
Technical Report. `http://arxiv.org/abs/1711.00455`.

N. Carlini, G. Katz, C. Barrett, and D. Dill.
Provably Minimally-Distorted Adversarial Examples, 2018.
Technical Report. `http://arxiv.org/abs/1709.10207`.

C. Cheng, G. Nührenberg, and H. Ruess.
Maximum Resilience of Artificial Neural Networks.
In *Proc. 15th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 251–268, 2017.

C. Cheng, G. Nührenberg, and H. Ruess.
Verification of Binarized Neural Networks, 2017.
Technical Report. `http://arxiv.org/abs/1710.03107`.

N. Carlini and D. Wagner.
Towards Evaluating the Robustness of Neural Networks.
IEEE Symposium on Security and Privacy, 2017.

K. Dvijotham, S. Gowal, R. Stanforth, R. Arandjelovic, B. O'Donoghue, J. Uesato, and P. Kohli.
Training Verified Learners with Learned Verifiers, 2018.
Technical Report. `http://arxiv.org/abs/1805.10265`.

S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari.
Output Range Analysis for Deep Feedforward Neural.
In *Proc. 10th NASA Formal Methods Symposium (NFM)*, pages 121–138, 2018.

R. Ehlers.
Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks.
In *Proc. 15th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 269–286, 2017.

D. Gopinath, G. Katz, C. Păsăreanu, and C. Barrett.
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.
In *Proc. 39th IEEE Symposium on Security and Privacy (S&P)*, 2018.

M. Hein and M. Andriushchenko.
Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation.
In *Proc. 31st Conf. on Neural Information Processing Systems (NIPS)*, 2017.

X. Huang, M. Kwiatkowska, S. Wang, and M. Wu.
Safety Verification of Deep Neural Networks.
In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 3–29, 2017.

J. Hull, D. Ward, and R. Zakrzewski.
Verification and Validation of Neural Networks for Safety-Critical Applications.
In *Proc. 21st American Control Conf. (ACC)*, 2002.

G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer.
Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks.
In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 97–117, 2017.

G. Katz, C. Barrett, D. Dill, K. Julian, and M Kochenderfer.
Towards Proving the Adversarial Robustness of Deep Neural Networks.
In *Proc. 1st Workshop on Formal Verification of Autonomous Vehicles (FVAV)*, pages 19–26, 2017.

A. Lomuscio and L. Maganti.
An Approach to Reachability Analysis for Feed-Forward ReLU Neural Networks, 2017.
Technical Report. `http://arxiv.org/abs/1706.07351`.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu.
Towards Deep Learning Models Resistant to Adversarial Attacks.

Proc. 6th Int. Conf. on Learning Representations (ICLR), 2018.

N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh.
Verifying Properties of Binarized Deep Neural Networks.
In *Proc. 32nd AAAI Conf. on Artificial Intelligence (AAAI)*, pages 6615–6624, 2018.

L. Pulina and A. Tacchella.
An Abstraction-Refinement Approach to Verification of Artificial Neural Networks.
In *Proc. 22nd Int. Conf. on Computer Aided Verification (CAV)*, pages 243–257, 2010.

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.
Global Robustness Evaluation of Deep Neural Networks with Provable Guarantees for L0 Norm, 2018.
Technical Report. `http://arxiv.org/abs/1804.05805`.

V. Tjeng and R. Tedrake.
Evaluating Robustness of Neural Networks with Mixed Integer Programming, 2017.
Technical Report. `http://arxiv.org/abs/1711.07356`.

S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana.
Formal Security Analysis of Neural Networks using Symbolic Intervals, 2018.
Technical Report. `http://arxiv.org/abs/1804.10829`.

T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, D. Boning, I. Dhillon, and L. Daniel.
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.

Output Reachable Set Estimation and Verification for Multi-Layer Neural Networks.
IEEE Transactions on Neural Networks and Learning Systems (TNNLS), 2018.