

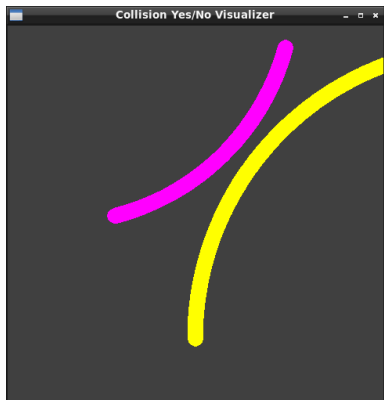
Safety and Reliability for Learning Systems

Part III - Verifying Learned Models

Rüdiger Ehlers, Clausthal University of Technology

Marktoberdorf Summer School, August 2019

Running example verification question for this part



Verification question

No collision predicted if compared to the situation depicted:

- Starting point obstacle vehicle is \leftarrow , \uparrow , or \swarrow
- Turning radius of the obstacle vehicle is \curvearrowright
- Turning radius of the controlled (yellow) vehicle is \curvearrowright
- Starting direction of controlled vehicle is exactly as shown
- Obstacle is at most as fast, but moves a bit.

Formalization

Data point

$$Rel_X = 0.2871866775226$$

$$Rel_Y = 0.50724176367$$

$$v_{2nd} = 0.121540046988$$

$$obDir = 0.58023977559$$

$$vRot_{1st} = 0.2$$

$$vRot_{2st} = -0.268581840689$$

Formalization

Setting encoding

$0 \leq Rel_X \leq$	0.2871866775226
$0 \leq Rel_Y \leq$	0.50724176367
$0.05 \leq v_{2nd} \leq$	0.121540046988
$\leq obDir =$	0.58023977559
$0.2 \leq vRot_{1st} \leq$	1
$-1 \leq vRot_{2st} \leq$	-0.268581840689

Formalization

Setting encoding

$0 \leq Rel_X \leq$	0.2871866775226
$0 \leq Rel_Y \leq$	0.50724176367
$0.05 \leq v_{2nd} \leq$	0.121540046988
$\leq obDir =$	0.58023977559
$0.2 \leq vRot_{1st} \leq$	1
$-1 \leq vRot_{2st} \leq$	-0.268581840689

Specification

- **Verification:** For all possible values for $\vec{x} = (Rel_X, Rel_Y, v_{2nd}, obDir, vRot_{1st}, vRot_{2st})$, we have that $f(\vec{x}) = \text{no collision}$.
- **Falsification:** There exists some $\vec{x} = (Rel_X, Rel_Y, v_{2nd}, obDir, vRot_{1st}, vRot_{2st})$ such that we have $f(\vec{x}) = \text{collision}$.

Verification methods for different orders

Model types considered

- Verifying linear models
- Verifying models using additional basis functions
- Verifying artificial neural networks

Verifying a linear model

Idea

- Connection between \vec{x} and $y = f(\vec{x})$ is linear.
- Constraint on \vec{x} is linear
- Requirements on y for *falsification* are linear

Example to follow

We use the linear regression model and find the minimal predicted distance between the vehicles.

Verifying a linear model

Idea

- Connection between \vec{x} and $y = f(\vec{x})$ is linear.
- Constraint on \vec{x} is linear
- Requirements on y for *falsification* are linear

→ we can use a linear programming tool to check if the learned model satisfies the requirements!

Example to follow

We use the linear regression model and find the minimal predicted distance between the vehicles.



Verifying a linear model

Idea

- Connection between \vec{x} and $y = f(\vec{x})$ is linear.
- Constraint on \vec{x} is linear
- Requirements on y for *falsification* are linear

→ we can use a linear programming tool to check if the learned model satisfies the requirements!

Example to follow

We use the linear regression model and find the minimal predicted distance between the vehicles.

Verifying a linear model with added basis functions

Difficulty

The added inputs to the model depend *non-linearly* on the core inputs to the model.

Verifying a linear model with added basis functions

Difficulty

The added inputs to the model depend *non-linearly* on the core inputs to the model.

Solution

Use an *SMT Solver* specializing on *non-linear arithmetic*
Here, this will be **dReach**.

What is an SMT Solver?

Core definition

SMT solvers check *Boolean combinations of terms* in various *theories* for *satisfiability*.

What is an SMT Solver?

Core definition

SMT solvers check *Boolean combinations of terms* in various *theories* for *satisfiability*.

Example

$$(x + y > 3) \wedge (x \leq 2) \wedge (y \leq 2) \wedge (x < 2 \vee y < 2)$$

What is an SMT Solver?

Core definition

SMT solvers check *Boolean combinations of terms* in various *theories* for *satisfiability*.

Example

$$(x + y > 3) \wedge (x \leq 2) \wedge (y \leq 2) \wedge (x < 2 \vee y < 2)$$

The formula is:

- Satisfiable if x and y are real-values variables
- Unsatisfiable if x and y are integer variables

SMT Solvers

Properties

- Modern SMT solvers frequently support multiple theories that also permit combining them in the terms
- The SMTLIB competition tracks progress in the area
- They often even support undecidable theories (e.g., non-linear arithmetic over integers)

The SMT solver dReal

SMT solver used here

- The solver **dReal** supports non-linear real arithmetic (with many operations).
- Since that theory is undecidable, the solver tackles a decidable *approximate version* of the problem instead.

Approximate satisfaction

Given a fixed value δ , every comparison of the form $t \leq t'$ is replaced by $t \leq t' + \delta$ (similarly for the other types of comparisons)

- satisfying assignments are close to real assignments
- UNSAT answers are correct
- computation time of **dReal** depends on choice of δ .

The SMT solver dReal



SMT solver used here

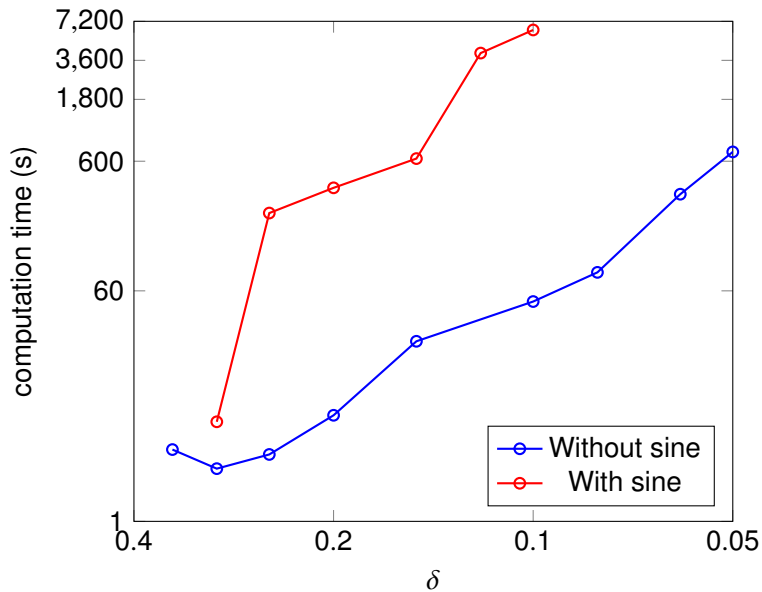
- The solver **dReal** supports non-linear real arithmetic (with many operations).
- Since that theory is undecidable, the solver tackles a decidable *approximate version* of the problem instead.

Approximate satisfaction

Given a fixed value δ , every comparison of the form $t \leq t'$ is replaced by $t \leq t' + \delta$ (similarly for the other types of comparisons)

- satisfying assignments are close to real assignments
- UNSAT answers are correct
- computation time of **dReal** depends on choice of δ .

Performance using dReal



Summary (dReal and additional basis functions)



Main problem

We have a massive scalability problem due to the non-linearities!

...and that is not dReal's fault – the verification problem **is** difficult!

How to solve this problem?

Types on the learned model

- Linear models are not powerful enough

How to solve this problem?

Types on the learned model

- Linear models are not powerful enough
- Non-linear models are too difficult to handle (in general) when verifying

How to solve this problem?

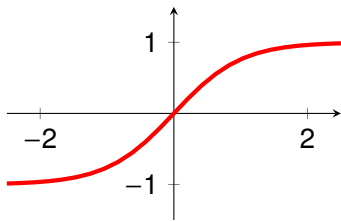
Types on the learned model

- Linear models are not powerful enough
- Non-linear models are too difficult to handle (in general) when verifying
- **Would piecewise linear models work (both in learning and verification)?**



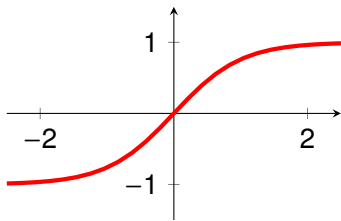
Artificial Neural Network Verification

Some common NN activation functions

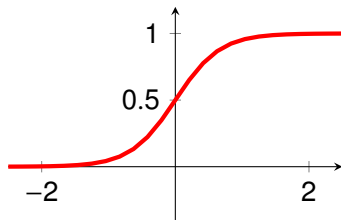


Tangens Hyperbolicus

Some common NN activation functions

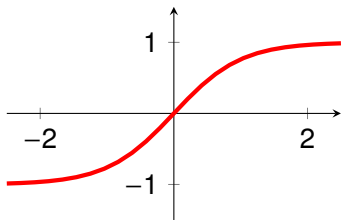


Tangens Hyperbolicus

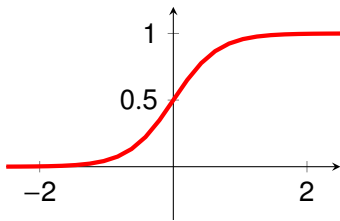


Sigmoid

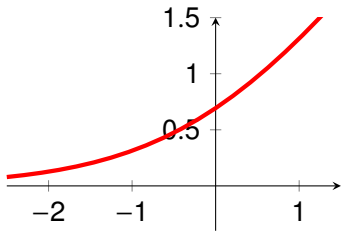
Some common NN activation functions



Tangens Hyperbolicus

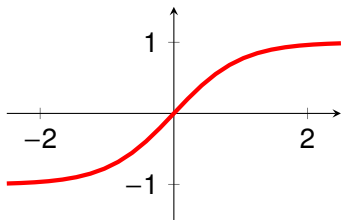


Sigmoid

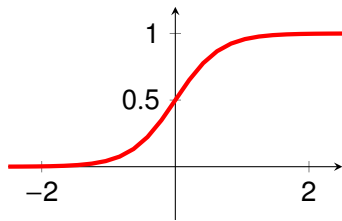


SoftPlus

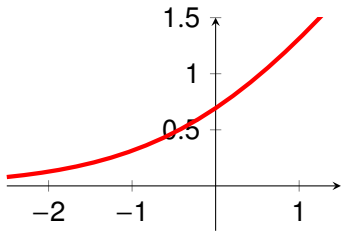
Some common NN activation functions



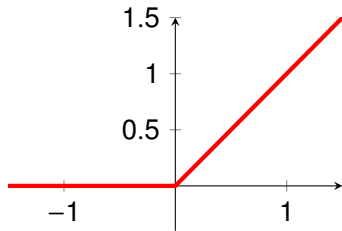
Tangens Hyperbolicus



Sigmoid

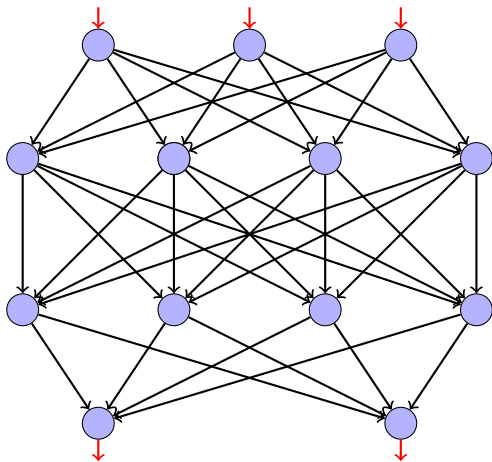


SoftPlus

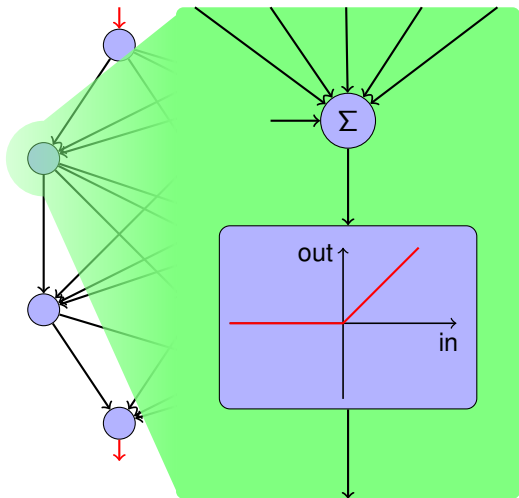


ReLU

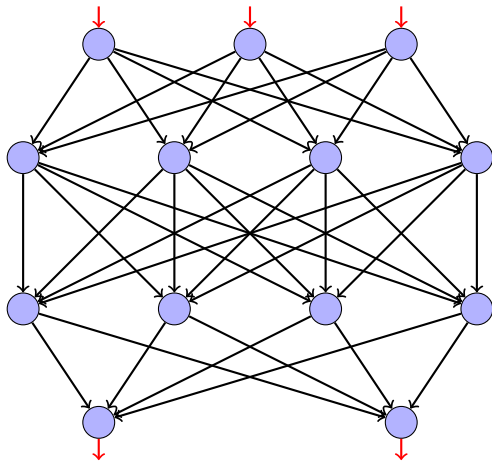
ReLU – example



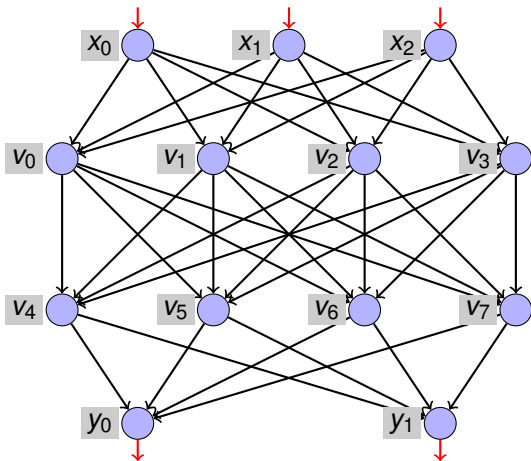
ReLU – example



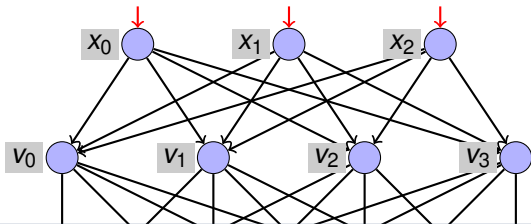
Starting point for verification: a constraint system



Starting point for verification: a constraint system



Starting point for verification: a constraint system

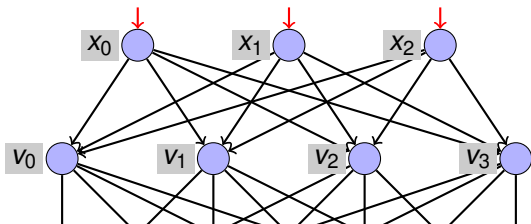


Example constraint

$$v_1 = \max(0, c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3)$$



Starting point for verification: a constraint system

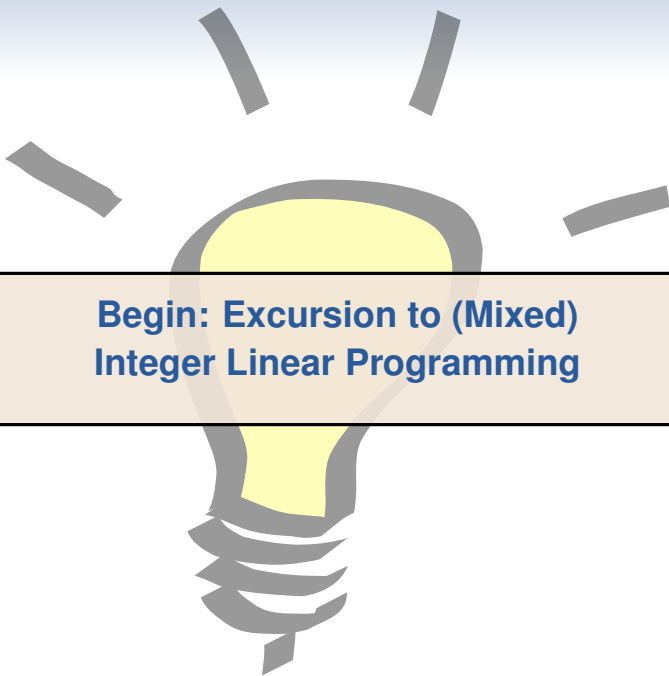


Example constraint

$$v_1 = \max(0, c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3)$$

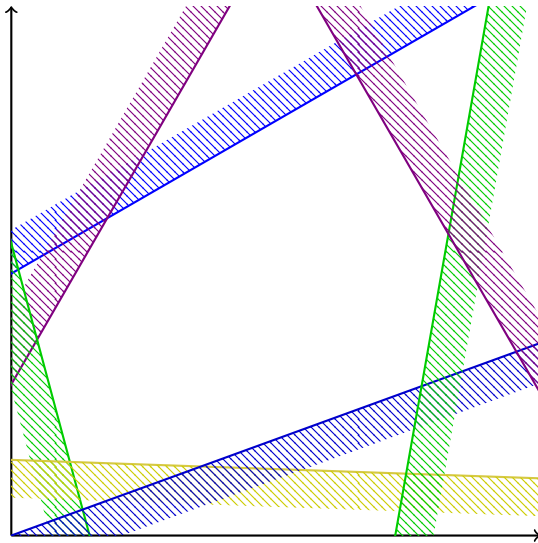
How do we encode that in LP solving?



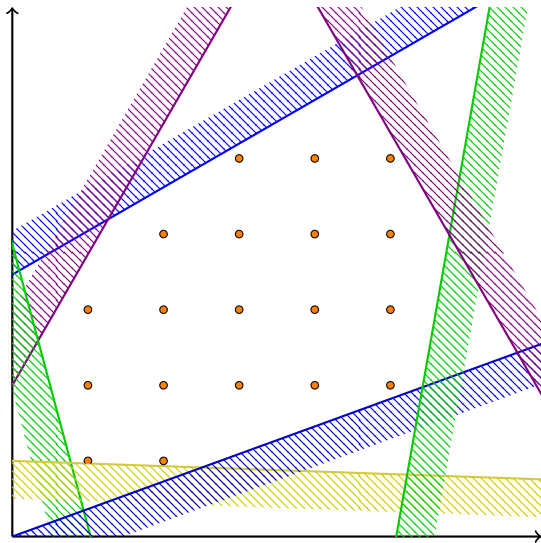


**Begin: Excursion to (Mixed)
Integer Linear Programming**

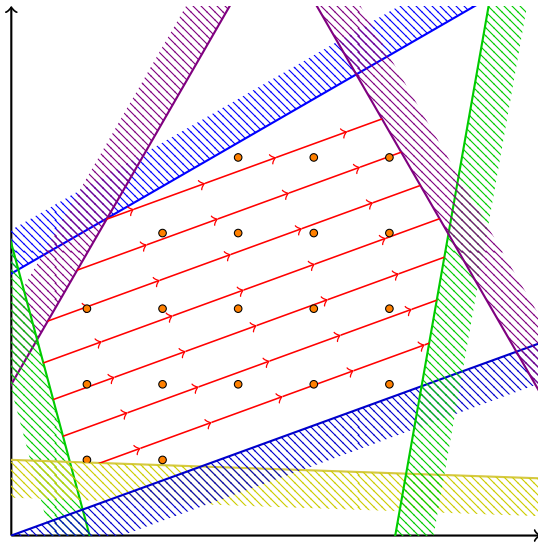
Integer Linear programming



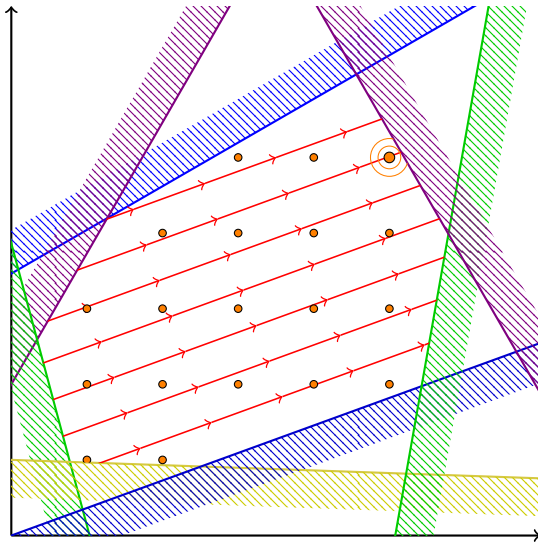
Integer Linear programming



Integer Linear programming



Integer Linear programming



Tool demonstration

Example problem that is suitable for lp_solve

```
max: x1+x2;  
x1 >= 3;  
x2 >= 1;  
x1 + 3*x2 >= 2;  
5*x1 + 7*x2 <= 109;  
int x1;  
int x2;
```

Integer linear programming (ILP) – Properties

Complexity

NP-hard, as SAT can be reduced to ILP

Integer linear programming (ILP) – Properties

Complexity

NP-hard, as SAT can be reduced to ILP

Algorithmic question

However, ILP solving is typically done with *branch-and-bound* algorithms (instead of SAT-like reasoning), as it is more well-suited to practical problems for which ILP solving is a “natural” computation engine.

Integer linear programming (ILP) – Properties

Complexity

NP-hard, as SAT can be reduced to ILP

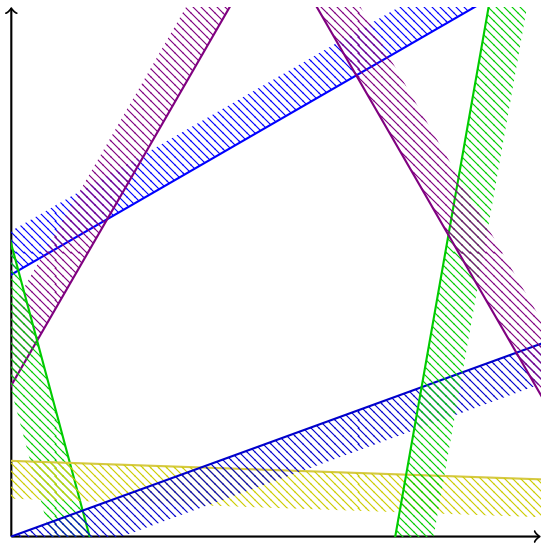
Algorithmic question

However, ILP solving is typically done with *branch-and-bound* algorithms (instead of SAT-like reasoning), as it is more well-suited to practical problems for which ILP solving is a “natural” computation engine.

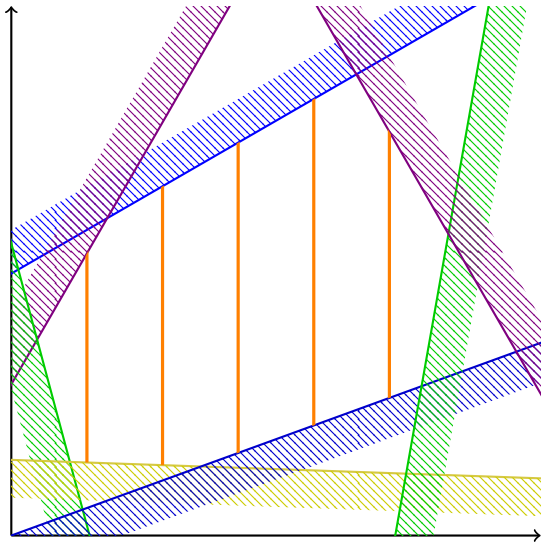
Example for the special properties of ILP problems

If
 $\vec{x} = (x_1, x_2, \dots, x_{k-1}, 1, x_{k+1}, \dots, x_n)$ and
 $\vec{x} = (x_1, x_2, \dots, x_{k-1}, 3, x_{k+1}, \dots, x_n)$ are valid solutions, then so is
 $\vec{x} = (x_1, x_2, \dots, x_{k-1}, 2, x_{k+1}, \dots, x_n)$.

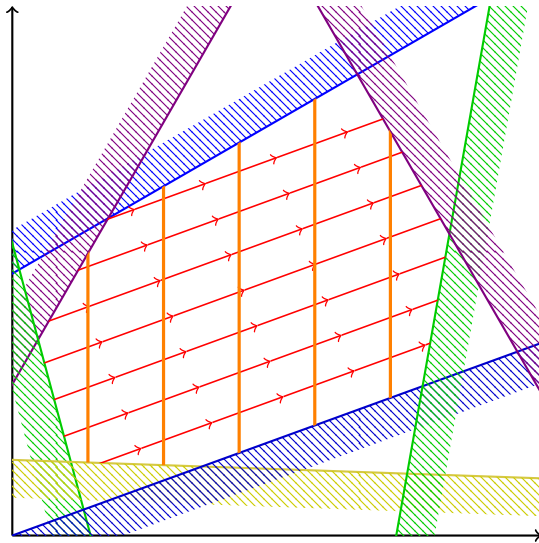
Mixed integer linear programming



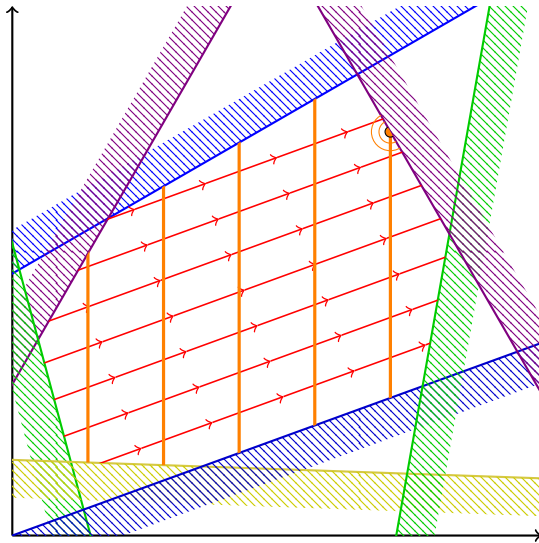
Mixed integer linear programming




Mixed integer linear programming



Mixed integer linear programming





**Begin: Excursion to (Mixed)
Integer Linear Programming**

Encoding a ReLU node

Starting point

$$v_1 = \max(0, c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3)$$

Constraints

Let L and U be *known* upper and lower bounds to $c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3$.

Inequalities (over an additional variable $b \in \{0, 1\}$):

- 1 $c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3 < 0 \rightarrow b = 0$
- 2 $c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3 > 0 \rightarrow b = 1$
- 3 $v_1 \geq 0$
- 4 $v_1 \geq c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3$
- 5 $b = 0 \rightarrow v_1 \leq 0$
- 6 $b = 1 \rightarrow v_1 \leq c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3$

Encoding a ReLU node

Starting point

$$v_1 = \max(0, c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3)$$

Constraints

Let L and U be *known* upper and lower bounds to $c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3$.

Inequalities (over additional variable $b \in \{0, 1\}$):

- 1 $c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3 \geq L \cdot (1 - b)$
- 2 $c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3 \leq U \cdot b$
- 3 $v_1 \geq 0$
- 4 $v_1 \geq c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3$
- 5 $v_1 \leq U \cdot b$
- 6 $v_1 \leq c_0 \cdot x_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + c_3 + \max(U, 0) \cdot (1 - b)$

Let's try it out!



Example for classification with successive approximation and artificial neural networks and MILP solving

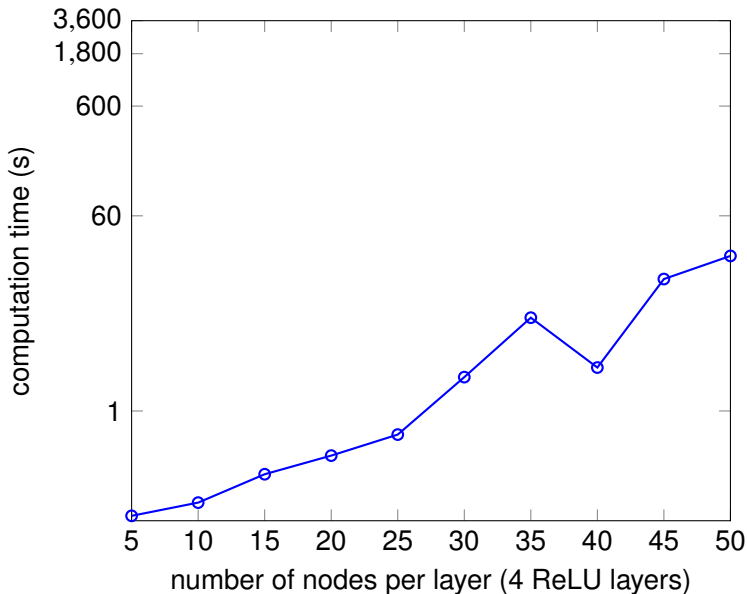
```
In [1]: 1 # Import libraries
        2 import numpy
        3 import matplotlib
        4 %matplotlib inline
```

1. Learn

1.1 Load input data

```
In [2]: 1 # Read CSV file
        2 with open("../collisions_regression.csv", "r") as inFile:
        3     allCSVLines = [a.strip().split(",") for a in inFile.readlines()]
        4 allCSVLines = [[float(a) for a in b] for b in allCSVLines]
        5
        6 # Visualize a few lines of the output
        7 from IPython.display import HTML, display
        8 import tabulate
        9 table = allCSVLines[0:9]
```

Computation time (CPLEX 12.8.0.0) for NN verification



So is neural network verification still hard?

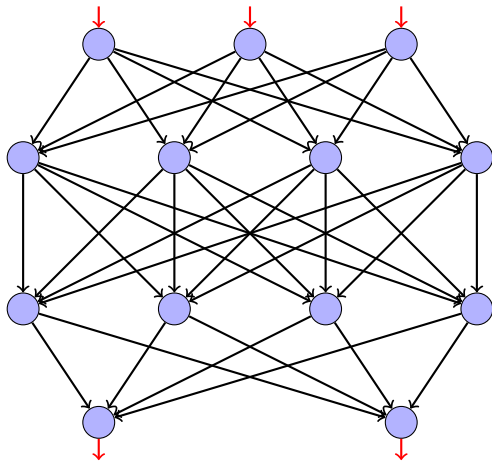
Answer

Yes! ReLU-networks are NP-complete to verify (Katz et al., 2017).

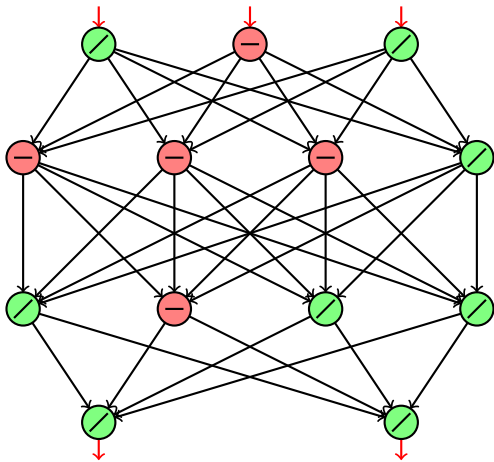
Containment in NP

For every *node phase assignment*, the problem can be solved in polynomial time.

Node phase assignment



Node phase assignment



Proof of NP-hardness (Katz et al., 2017)

Starting point

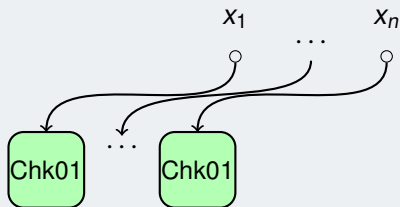
We reduce n -variable SAT to *NN falsification* for a network with n inputs that represent a satisfying assignment.

Proof of NP-hardness (Katz et al., 2017)

Starting point

We reduce n -variable SAT to *NN falsification* for a network with n inputs that represent a satisfying assignment.

General shape



Constraints:

$$0 \leq x_1 \leq 1$$

\dots

$$0 \leq x_n \leq 1$$

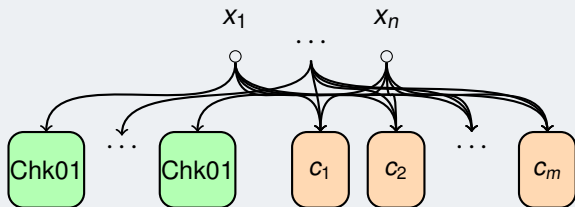
$$y \geq 1$$

Proof of NP-hardness (Katz et al., 2017)

Starting point

We reduce n -variable SAT to *NN falsification* for a network with n inputs that represent a satisfying assignment.

General shape



Constraints:

$$0 \leq x_1 \leq 1$$

\dots

$$0 \leq x_n \leq 1$$

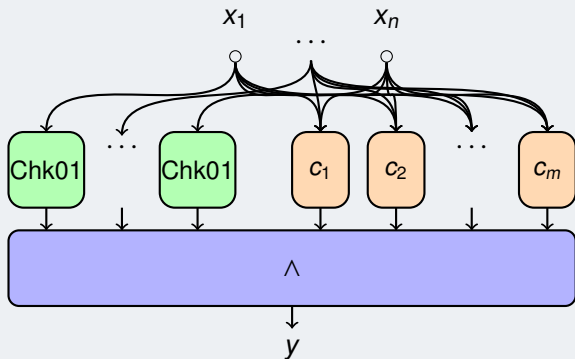
$$y \geq 1$$

Proof of NP-hardness (Katz et al., 2017)

Starting point

We reduce n -variable SAT to *NN falsification* for a network with n inputs that represent a satisfying assignment.

General shape



Constraints:

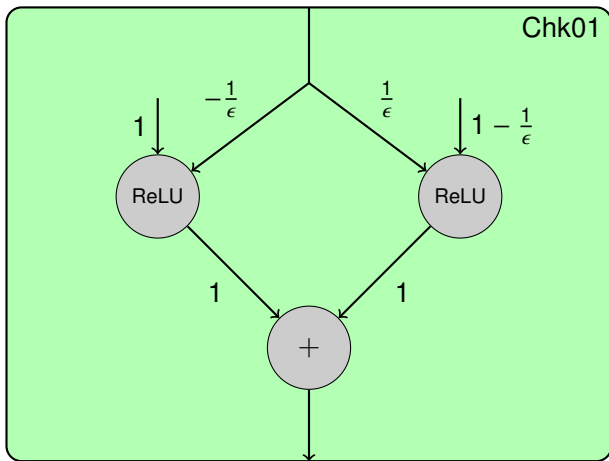
$$0 \leq x_1 \leq 1$$

...

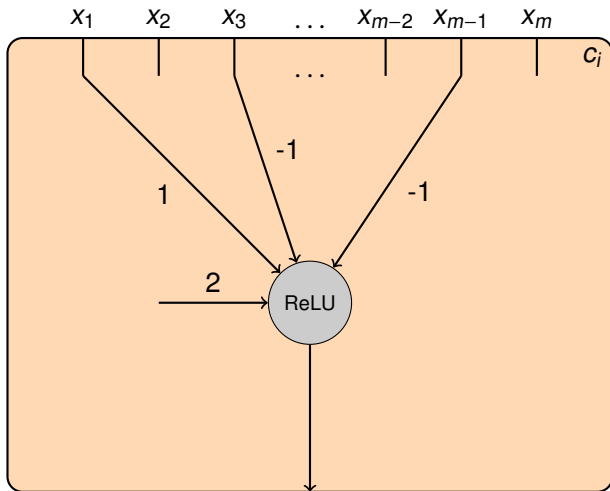
$$0 \leq x_n \leq 1$$

$$y \geq 1$$

The Chk01 component

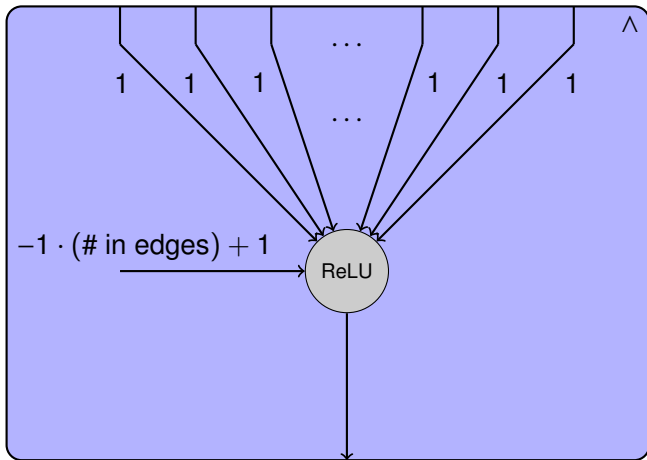


The c_i component



Example clause: $x_1 \vee \neg x_3 \vee \neg x_{m-1}$

The \wedge component



Ok, so it's hard in theory...

But what about practice?

In the domain of SAT solving, the NP-hardness of the problem does not tell us something about the *practical* complexity of solving the problems.

Ok, so it's hard in theory...

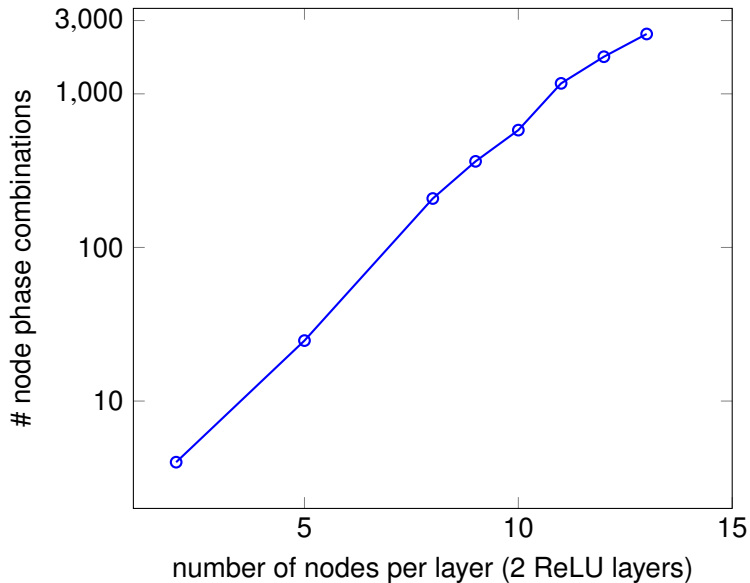
But what about practice?

In the domain of SAT solving, the NP-hardness of the problem does not tell us something about the *practical* complexity of solving the problems.

Idea

If we *had* a low number of *node phase combinations*, the verification problem may be simple.

Number of node phase combinations



So what now?

Comparison to SAT solving

As in SAT solving, we need to make use of the *structure* of the problem.

Here: we have a *specification* that can be made use of and the network response is *continuous*. We can use them to help the solver?



Begin: Excursion to how LP/ILP/MILP solving works

The simplex algorithm

Basic observation

For finding an optimal point, it suffices to check all corners of the space of feasible solutions.

Simplex algorithm - basic idea

- Start in a corner of the solution space
- Follow edges along the *polytope* of solutions that improve the valuation
- When no improvement is possible, we found the optimum.

Starting in a point

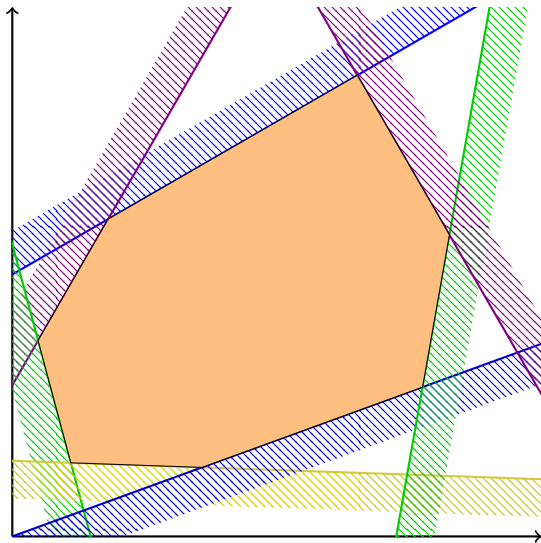
One possibility

- Start from an arbitrary point satisfying the bounds
- Choose a violated linear constraint
- Fix the constraint by changing the candidate solution
- Repeat the last two steps until a point satisfying all constraints has been found.

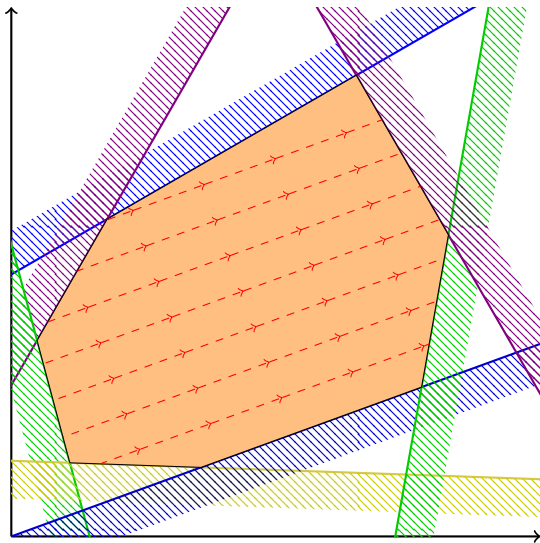
Another possibility

- Build an LP instance that is trivially satisfiable by introducing non-negative *slack* variables for every constraint
- Then, use the optimization step to minimize the sum of slack variables (to 0, if possible).

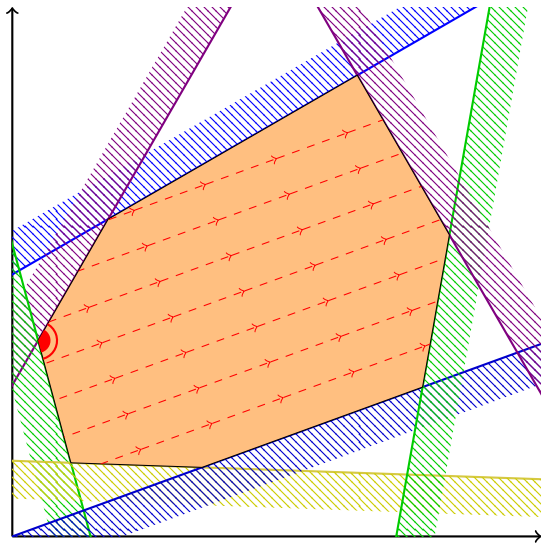
Geometric interpretation of the improvement step of the simplex algorithm



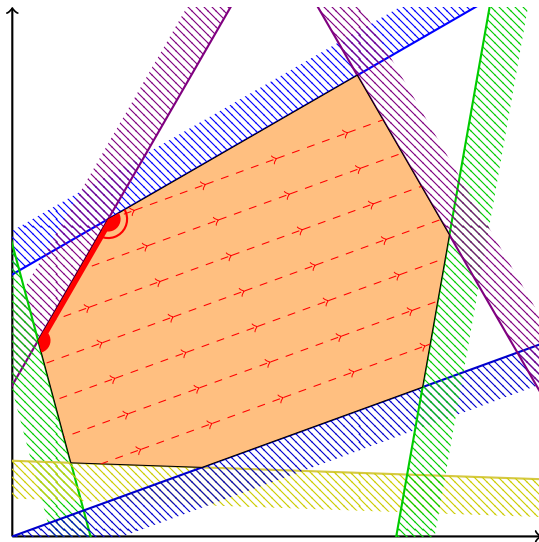
Geometric interpretation of the improvement step of the simplex algorithm



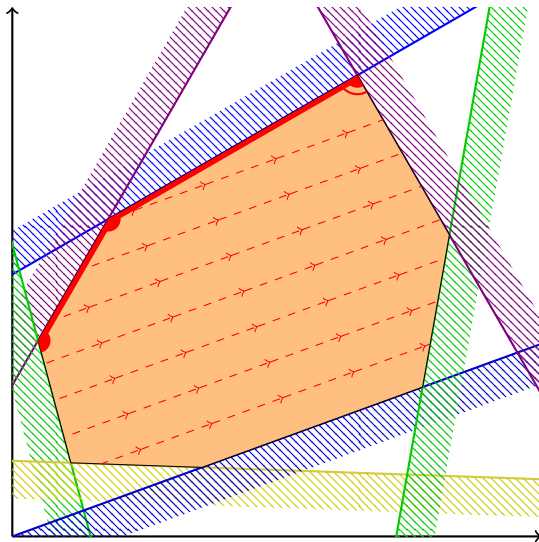
Geometric interpretation of the improvement step of the simplex algorithm



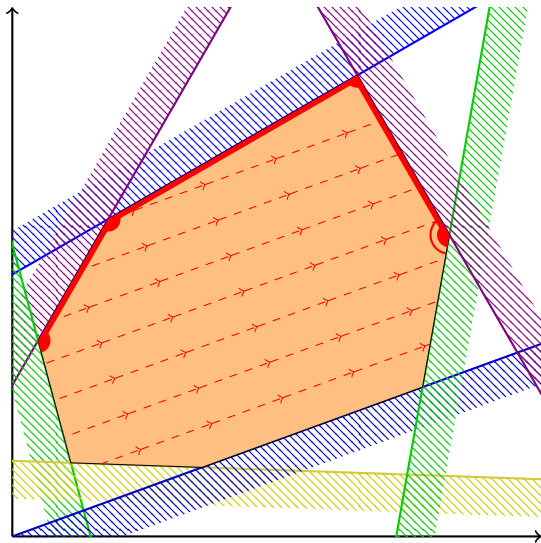
Geometric interpretation of the improvement step of the simplex algorithm



Geometric interpretation of the improvement step of the simplex algorithm



Geometric interpretation of the improvement step of the simplex algorithm



The simplex algorithm – problem

Complexity

- The maximum number of corners of a polytope increases polynomially with the number of constraints, but exponentially with the number of dimensions

The simplex algorithm – problem

Complexity

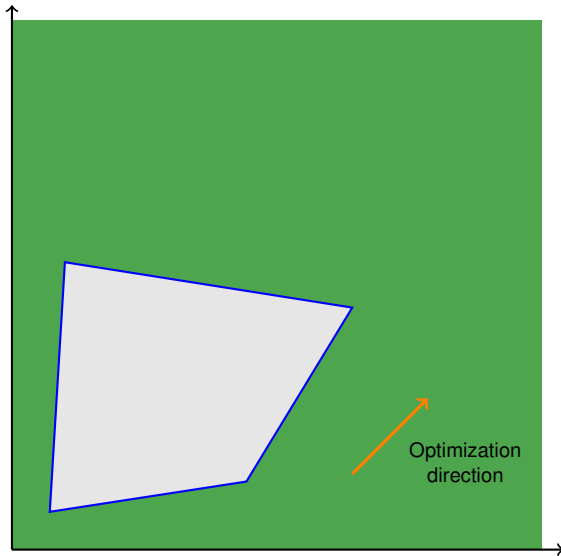
- The maximum number of corners of a polytope increases polynomially with the number of constraints, but exponentially with the number of dimensions
- We can build instances for which an exponentially long sequence of “improving edges” exists.

The simplex algorithm – problem

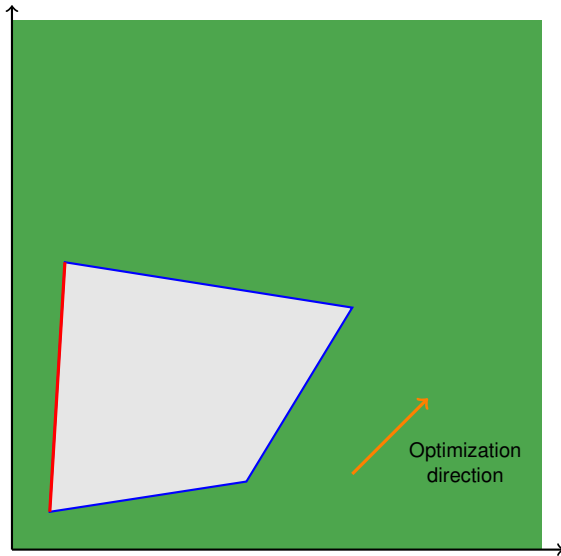
Complexity

- The maximum number of corners of a polytope increases polynomially with the number of constraints, but exponentially with the number of dimensions
- We can build instances for which an exponentially long sequence of “improving edges” exists.
- For all commonly used variants of the simplex algorithm, there exists a family of instances on which the algorithm needs exponential time.

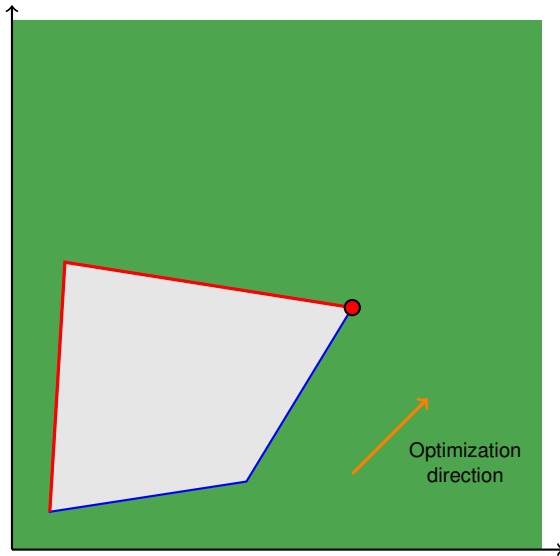
2D Case



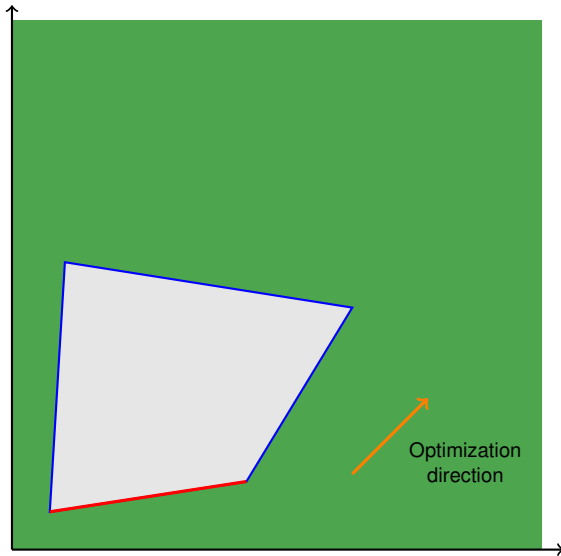
2D Case



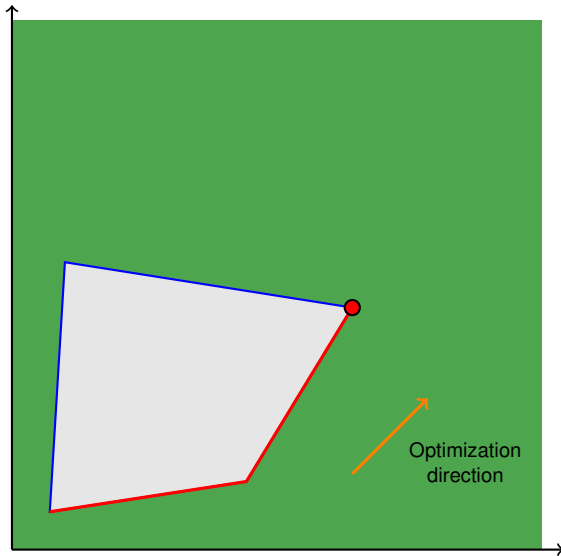
2D Case



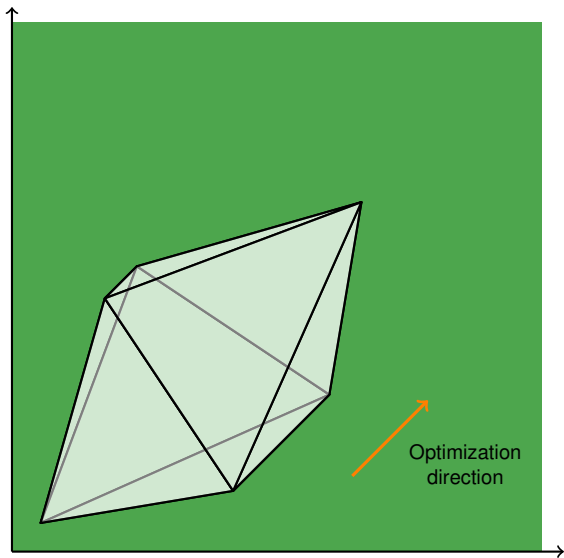
2D Case



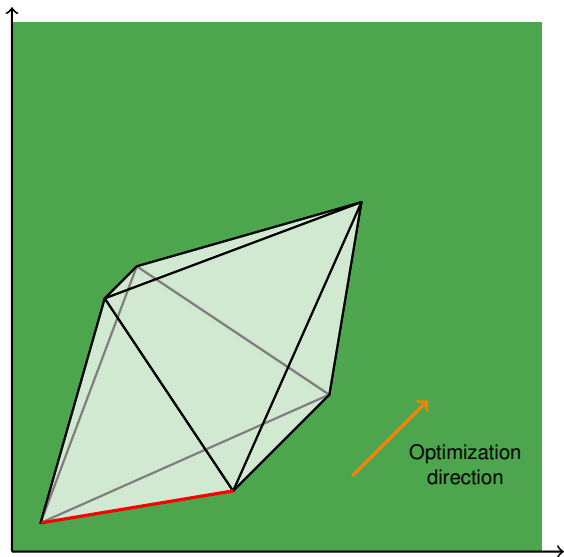
2D Case



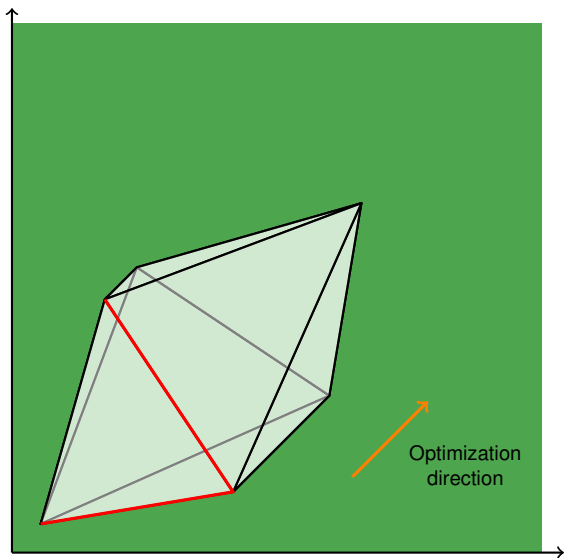
3D Case (Orthographic projection)



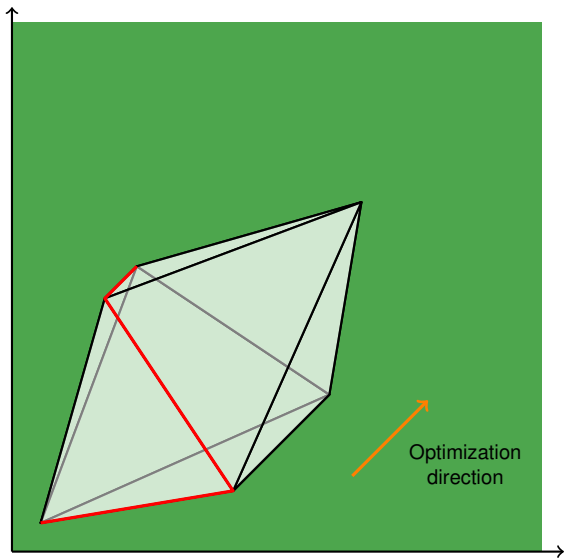
3D Case (Orthographic projection)



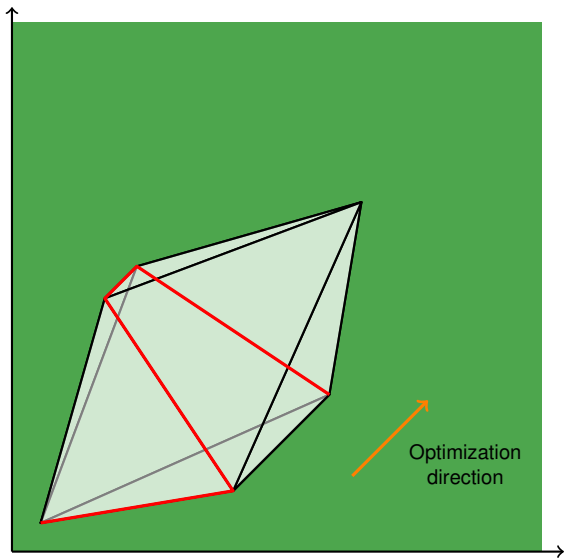
3D Case (Orthographic projection)



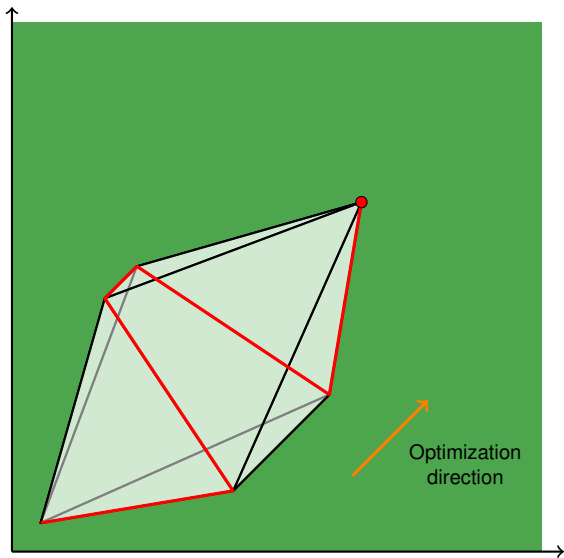
3D Case (Orthographic projection)



3D Case (Orthographic projection)



3D Case (Orthographic projection)



Reminder

Reminder

Despite the exponential worst-case running time, the simplex algorithm is still used in practice heavily because hard instances are *rare*.

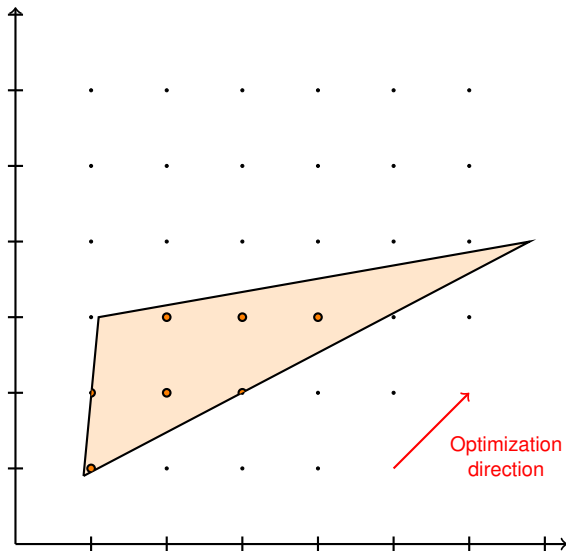
Expected computation time

For some reasonable probability distribution over LP problems, the simplex algorithm runs in expected polynomial time (see “Probabilistic Analysis in Linear Programming” by Ron Shamir, *Statistical Science*, Vol. 8, No. 1, 1993)

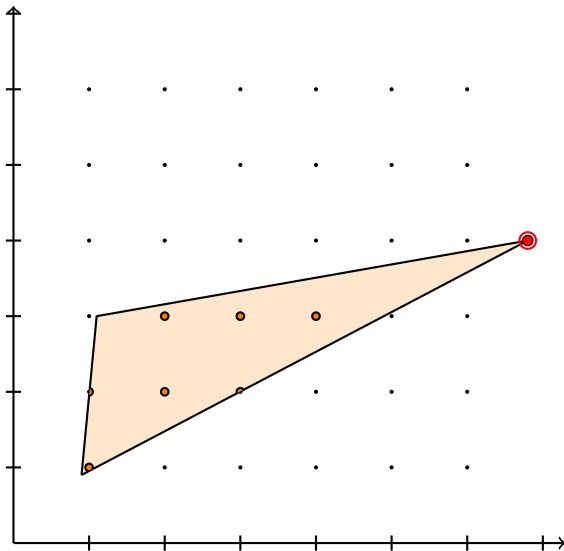
ILP – Branch-and-bound algorithm

```
1: function B-AND-B-ILP( $\mathcal{V}, P, o$ )
2:    $(\vec{s}, q) = \text{SolveLP}(\mathcal{V}, P)$ 
3:   if  $q \leq o$  then
4:     return  $(\emptyset, -\infty)$ 
5:   else if  $\vec{s}$  is integral then
6:     return  $(\vec{s}, q)$ 
7:   else
8:     select  $v \in \mathcal{V}$  such that  $\vec{s}(v)$  is non-integral
9:      $(\vec{s}', q') := \text{B-AND-B-ILP}(\mathcal{V}, P \cup \{v \geq \lceil \vec{s}(v) \rceil\}, o)$ 
10:     $(\vec{s}'', q'') := \text{B-AND-B-ILP}(\mathcal{V}, P \cup \{v \leq \lfloor \vec{s}(v) \rfloor\}, \max(o, q'))$ 
11:    if  $q' > q''$  then
12:      return  $(s', q')$ 
13:    else
14:      return  $(s'', q'')$ 
15:    end if
16:  end if
17: end function
```

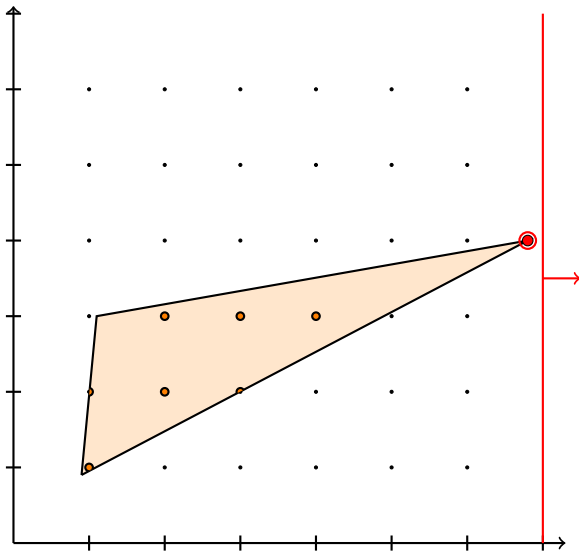
ILP – Branch and bound example 1



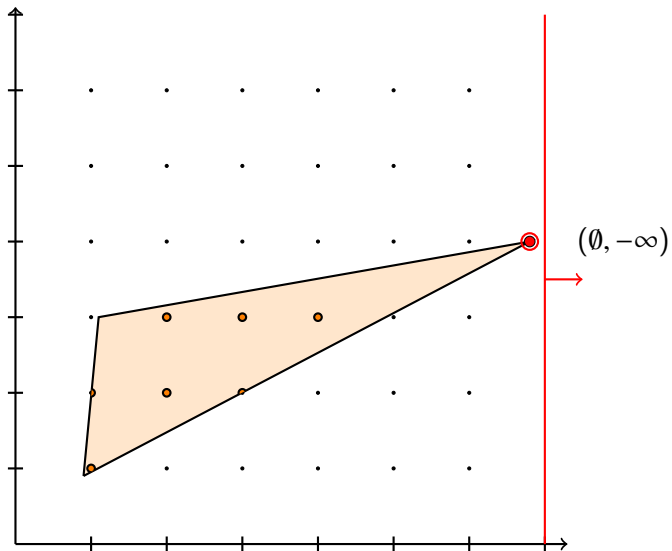
ILP – Branch and bound example 1



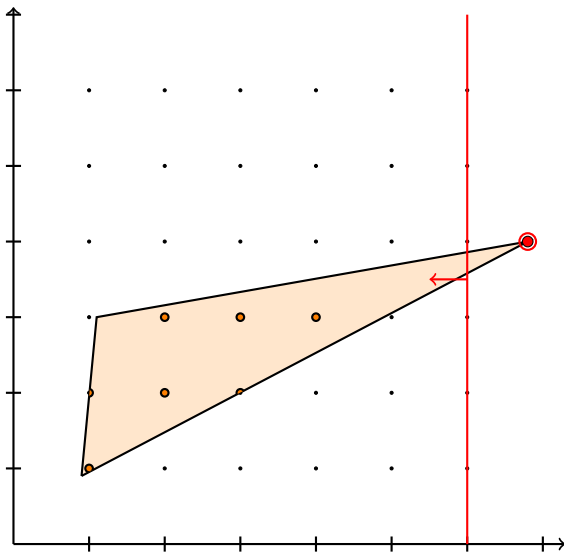
ILP – Branch and bound example 1



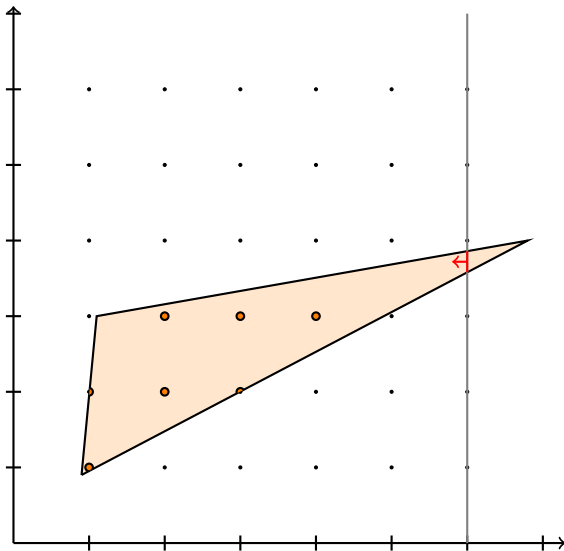
ILP – Branch and bound example 1



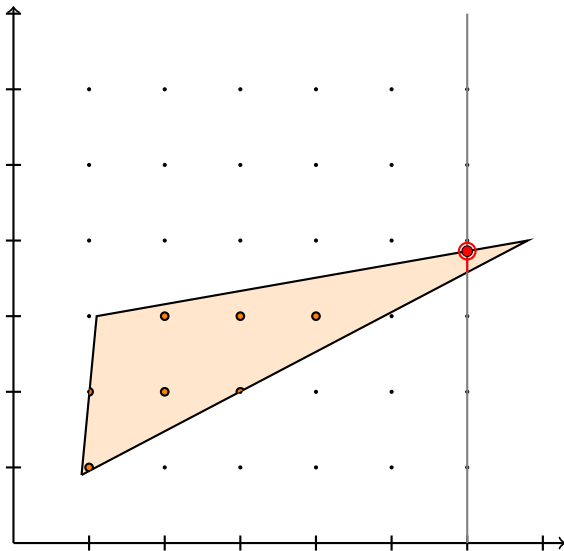
ILP – Branch and bound example 1



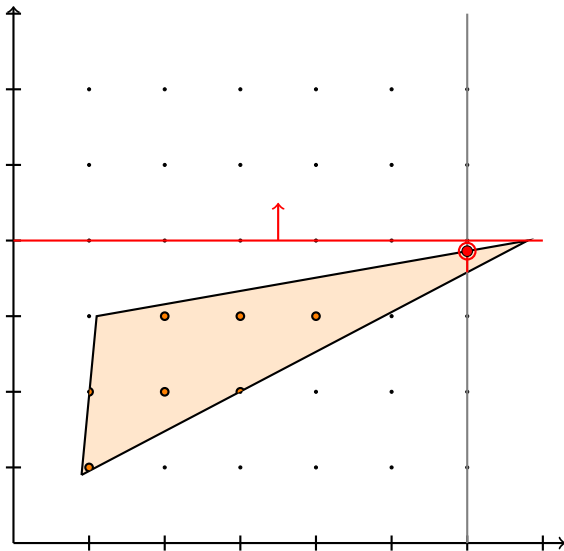
ILP – Branch and bound example 1



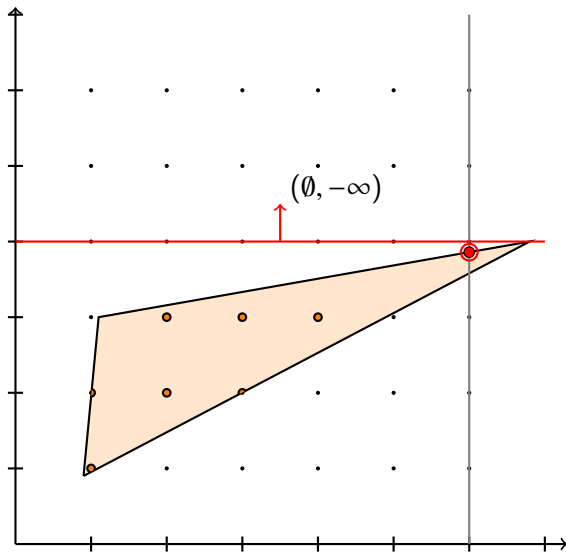
ILP – Branch and bound example 1



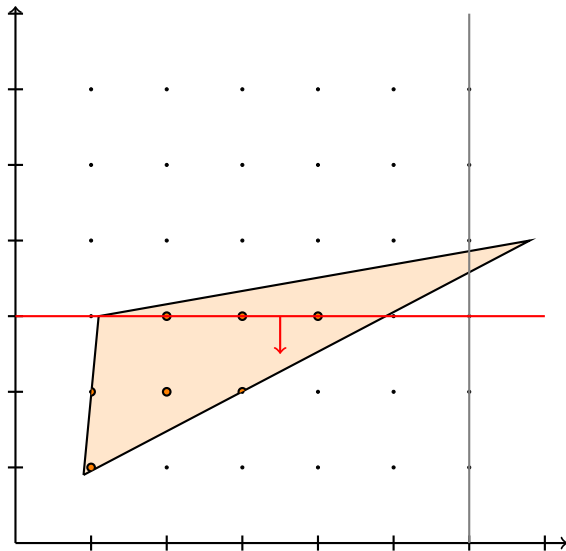
ILP – Branch and bound example 1



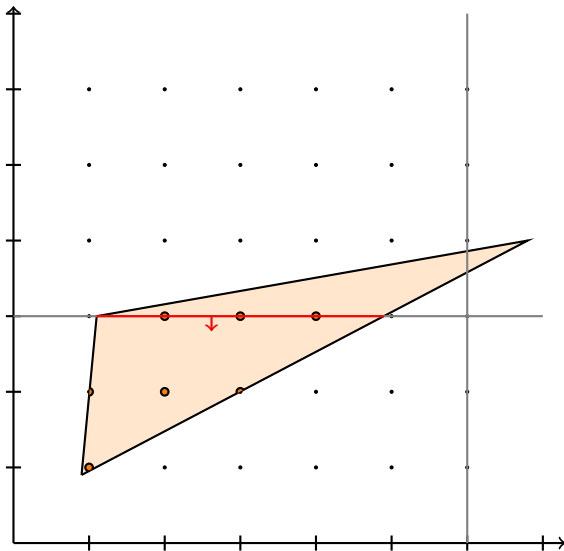
ILP – Branch and bound example 1



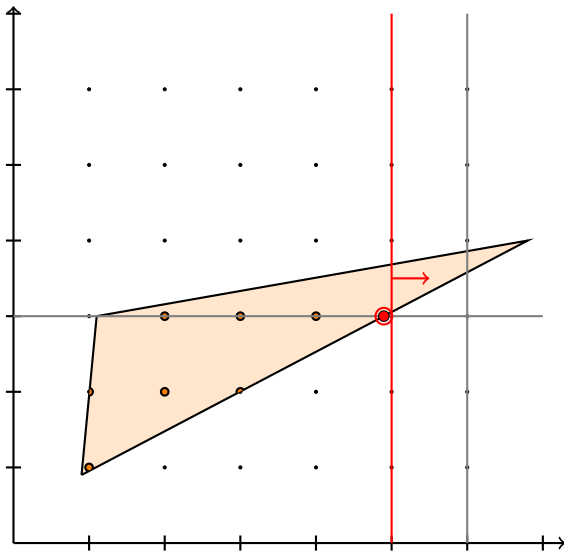
ILP – Branch and bound example 1



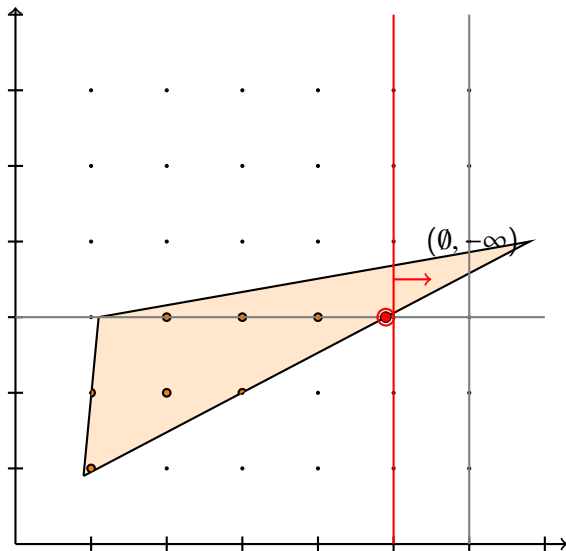
ILP – Branch and bound example 1



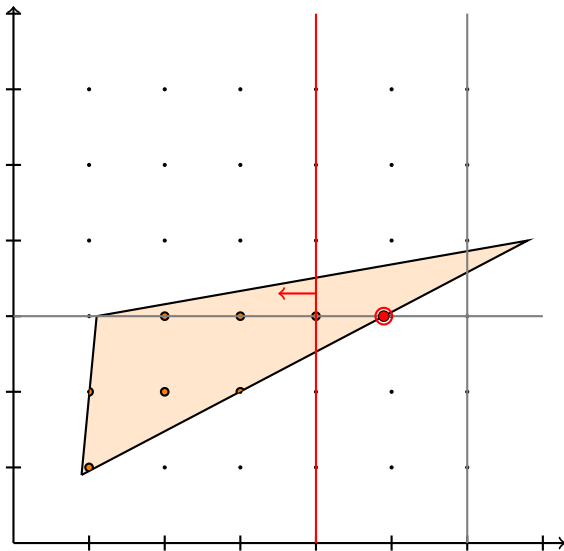
ILP – Branch and bound example 1



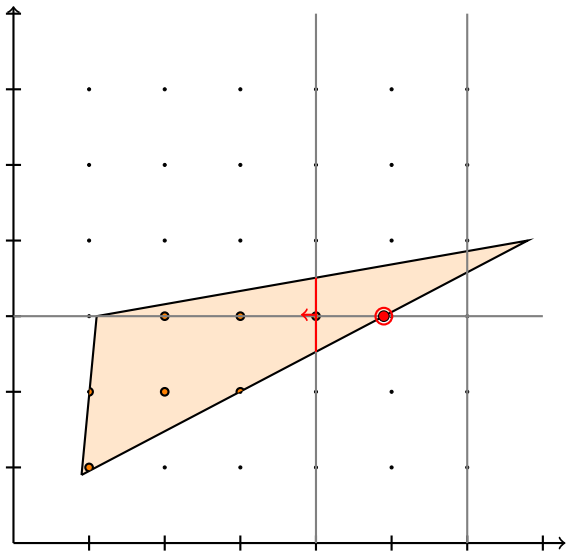
ILP – Branch and bound example 1



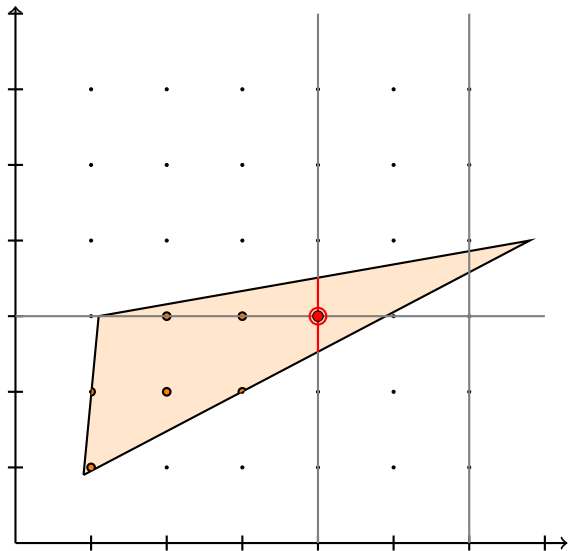
ILP – Branch and bound example 1



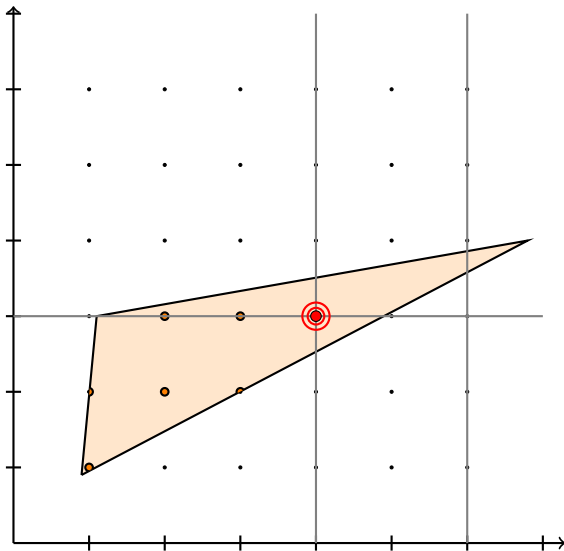
ILP – Branch and bound example 1



ILP – Branch and bound example 1



ILP – Branch and bound example 1



Mixed integer programming – Notes

Extension: “branch and cut”

As an extension to branch and bound, “branch and cut” can be applied. Here, constraints that only exclude non-integer solutions are added during the solution process.

Variable selection

There are many pivoting rules for ILP that select variables to constrain next. Also, in some cases, we may want to start with the smaller value for a variable.



End: Excursion to how LP/ILP/MILP solving works

Observations on integer linear programming

What makes it hard?

In practice, it's the *integer* part that makes it difficult to solve.

Observations on integer linear programming

What makes it hard?

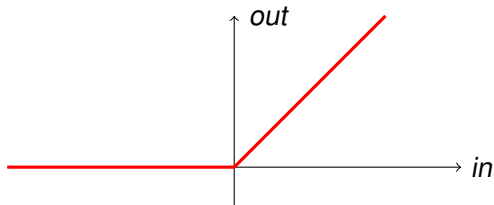
In practice, it's the *integer* part that makes it difficult to solve.

So how do we improve verification speed?

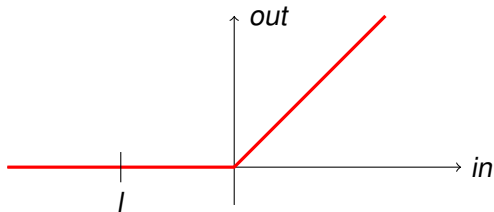
By putting as much power as possible into the *linear* solver part.

Adding *implied* linear constraints can be beneficial if that reduces the number of branchings that need to be performed.

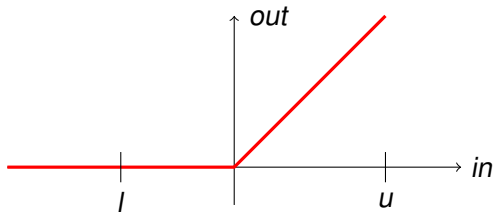
Approach: Approximating ReLU non-convexity



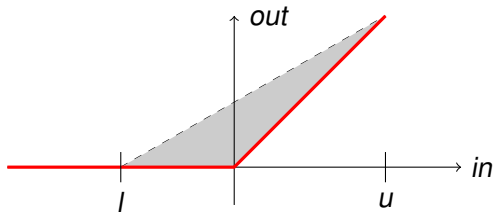
Approach: Approximating ReLU non-convexity



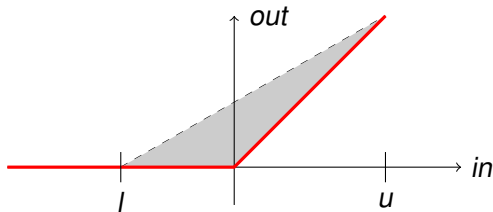
Approach: Approximating ReLU non-convexity



Approach: Approximating ReLU non-convexity



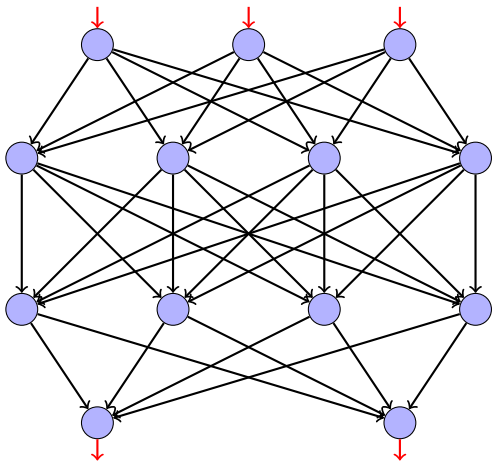
Approach: Approximating ReLU non-convexity



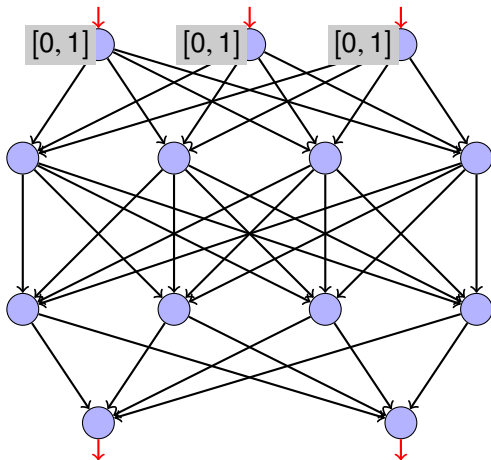
What we need...

...are upper and lower bounds on in .

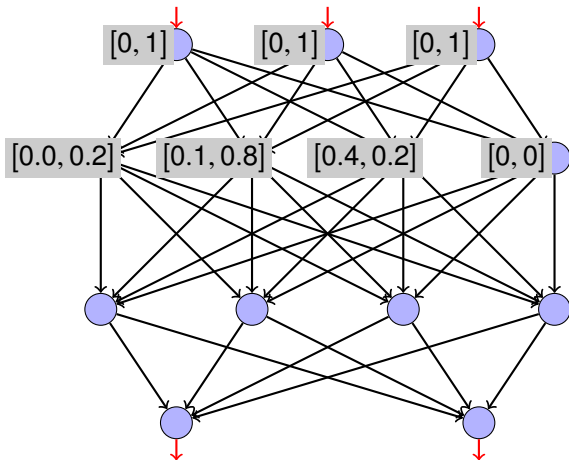
Obtaining upper and lower bounds by basic interval arithmetic



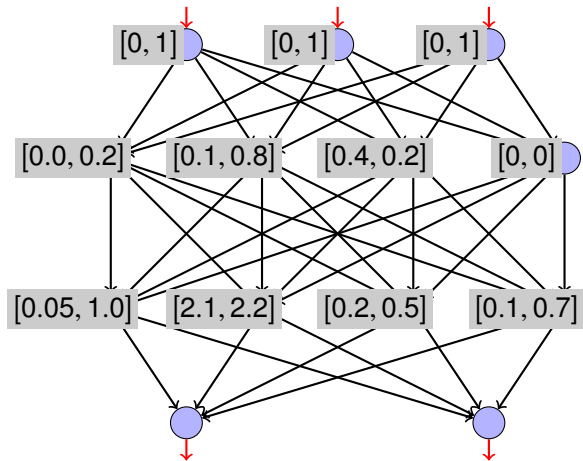
Obtaining upper and lower bounds by basic interval arithmetic



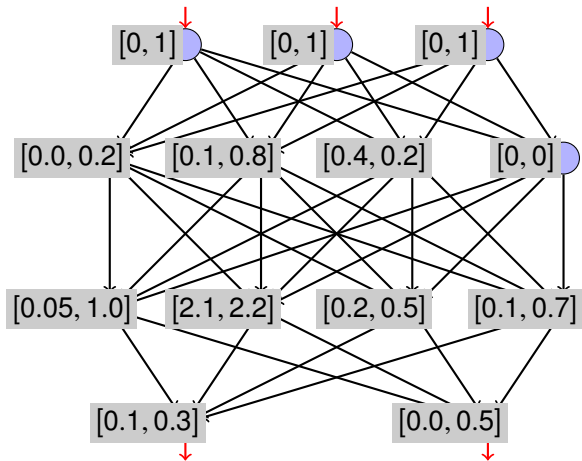
Obtaining upper and lower bounds by basic interval arithmetic



Obtaining upper and lower bounds by basic interval arithmetic



Obtaining upper and lower bounds by basic interval arithmetic



Step 2: Improving the bounds

Observation

The bounds obtained with this process are very coarse.

Step 2: Improving the bounds

Observation

The bounds obtained with this process are very coarse.

Idea

Use the approximation of the network behavior to improve the bounds!

- Let V the nodes in the network, φ be a linear approximation of the network behavior on V , and ψ be a constraint on what network behavior we search for.

Step 2: Improving the bounds

Observation

The bounds obtained with this process are very coarse.

Idea

Use the approximation of the network behavior to improve the bounds!

- Let V the nodes in the network, φ be a linear approximation of the network behavior on V , and ψ be a constraint on what network behavior we search for.
- Use $\varphi \cup \psi$ as linear programming (LP) problem instance and obtain for each $v \in V$ better lower and upper bounds.

Step 2: Improving the bounds

Observation

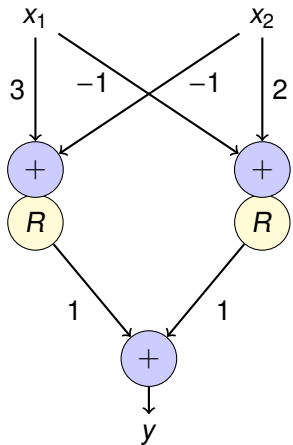
The bounds obtained with this process are very coarse.

Idea

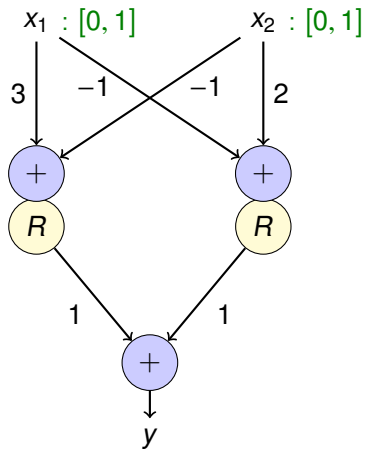
Use the approximation of the network behavior to improve the bounds!

- Let V the nodes in the network, φ be a linear approximation of the network behavior on V , and ψ be a constraint on what network behavior we search for.
- Use $\varphi \cup \psi$ as linear programming (LP) problem instance and obtain for each $v \in V$ better lower and upper bounds.
- Repeat the process a couple of times

Example

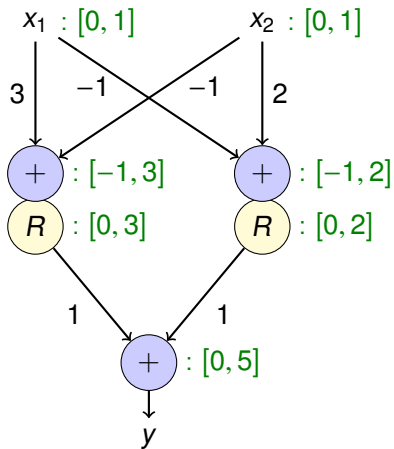


Example



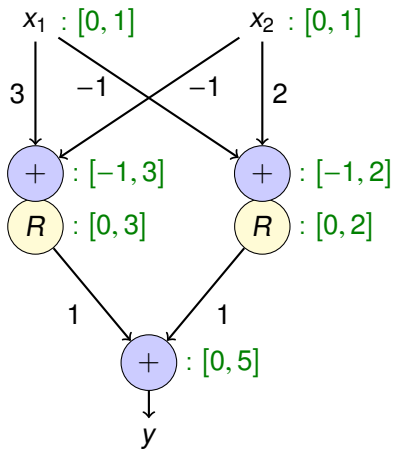
Spec: $y \geq 2.5$

Example



Spec: $y \geq 2.5$

Example



Spec: $y \geq 2.5$

```
max: ...;
```

```
/* Bounds */
```

```
0 <= x1 <= 1;
```

```
0 <= x2 <= 1;
```

```
/* Spec: */
```

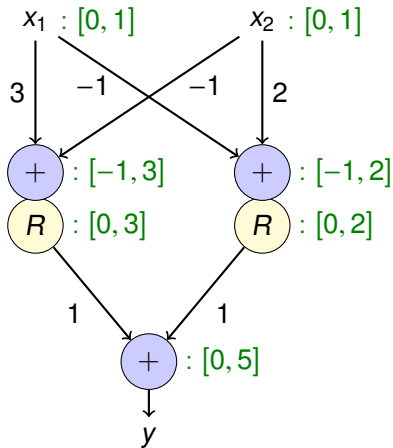
```
y >= 2.5;
```

```
/* Unbounded (at end) */
```

```
free n1;
```

```
free n2;
```

Example



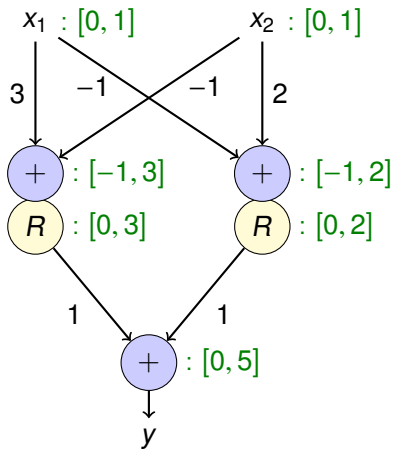
Spec: $y \geq 2.5$

```
max: ...;
```

```
/* Linear constraints */  
n1 = 3*x1 - 1*x2;  
n2 = -1*x1 + 2*x2;  
y = n1r + n2r;
```

```
/* ReLU approximation */  
n1r >= 0;  
n2r >= 0;  
n1r >= n1;  
n2r >= n2;
```

Example



Spec: $y \geq 2.5$

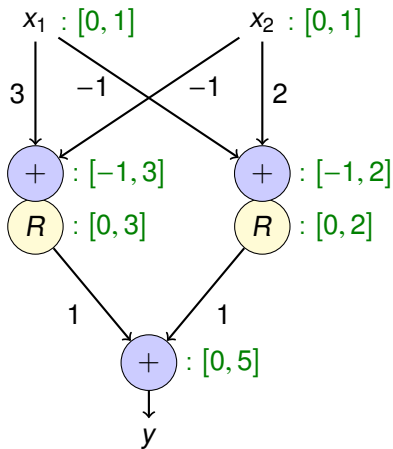
max: ...;

```
/* ReLU approx. (2) */  
n1r <= 0.75*n1 + 0.75;  
n2r <= 0.667*n2 + 0.667;
```

Formula for third ReLU
inequality:

$$out \leq \frac{u}{u-l} in - \frac{u \cdot l}{u-l}$$

Example

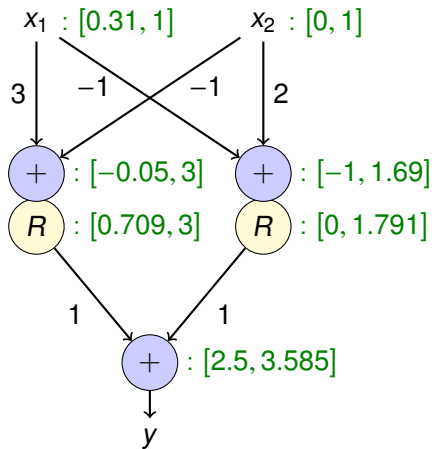


Spec: $y \geq 2.5$

Next step:

Solve the LP instance multiple times using all of x_1 , x_2 , n_1 , n_2 , and y as objective function for both minimization and maximization.

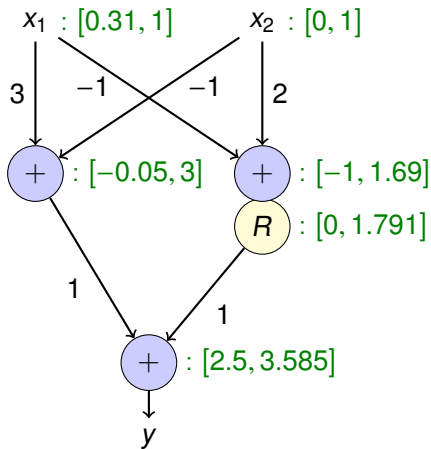
Example



We can see from the lower bounds on the left ReLU node that the ReLU needs to be in the linear phase.

Spec: $y \geq 2.5$

Example



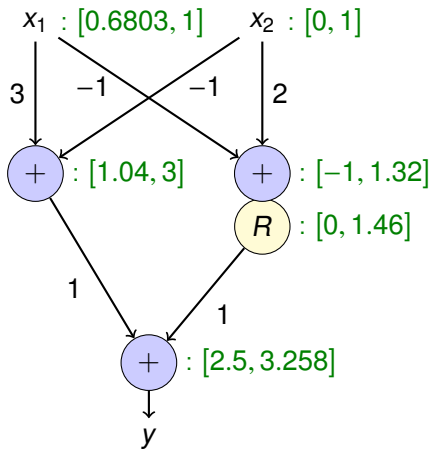
After replacing the ReLU node, we can obtain updated bounds again.

New inequation:

$$n2r \leq 0.629 * n2 + 0.629;$$

Spec: $y \geq 2.5$

Example



The whole process can then be repeated some additional times.

Spec: $y \geq 2.5$

Observation

Most important aspect

The inclusion of the specification led to a reduction in the number of ReLU nodes.

Observation

Most important aspect

The inclusion of the specification led to a reduction in the number of ReLU nodes.

Also important

The process can be repeated several times to increase precision. Note that the *rounding* direction is quite important.

Observation

Most important aspect

The inclusion of the specification led to a reduction in the number of ReLU nodes.

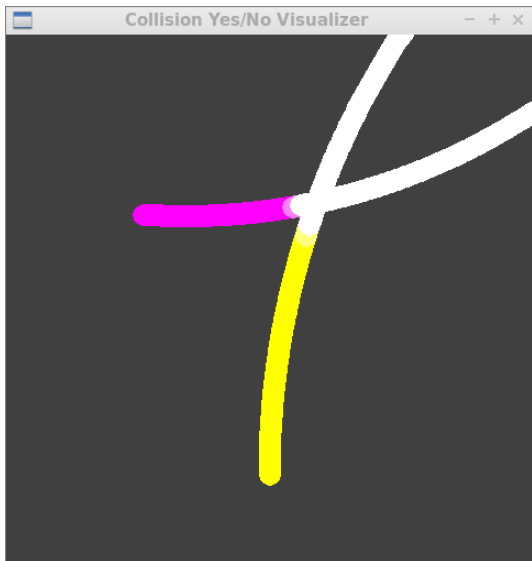
Also important

The process can be repeated several times to increase precision. Note that the *rounding* direction is quite important.

Drawback

This is only a *pre-processing* step. Normally, not all ReLU nodes can be eliminated (as in our example).

How much does this help – Collision detection



How much does this help – Collision detection

Network size

59 ReLU nodes, 19 MaxPool nodes, 2 linear nodes

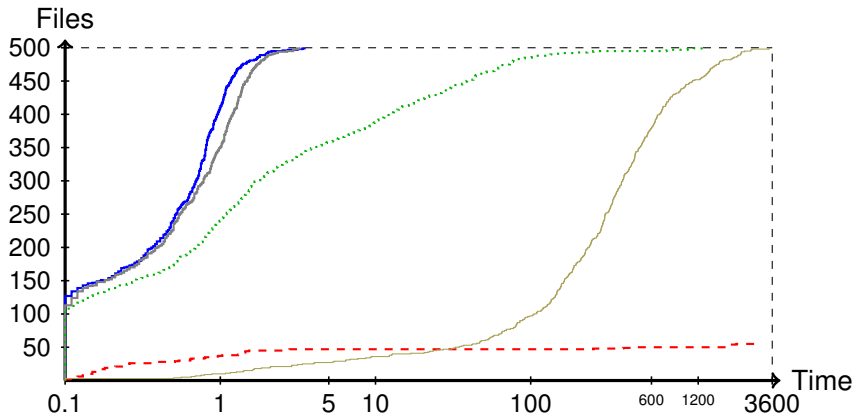
Verification condition pattern

Is the output value *robust* against disturbances of magnitude ϵ in the input vector?

Note

The learned network has a classification accuracy of 100% (on 3000 data points).

Performance comparison



Time is given in seconds (on a log-scale), and the lines, from bottom right to top left, represent Gurobi without linear approximation (dashed), Yices without linear approximation (solid), Yices with linear approximation (dotted), Planet (solid), and Gurobi with linear approximation (solid).

Results are from Ehlers (2017)

References I

- Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In Deepak D'Souza and K. Narayan Kumar, editors, *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, volume 10482 of *Lecture Notes in Computer Science*, pages 269–286. Springer, 2017. doi: 10.1007/978-3-319-68167-2_19. URL https://doi.org/10.1007/978-3-319-68167-2_19.
- Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 97–117, 2017. doi: 10.1007/978-3-319-63387-9_5. URL https://doi.org/10.1007/978-3-319-63387-9_5.