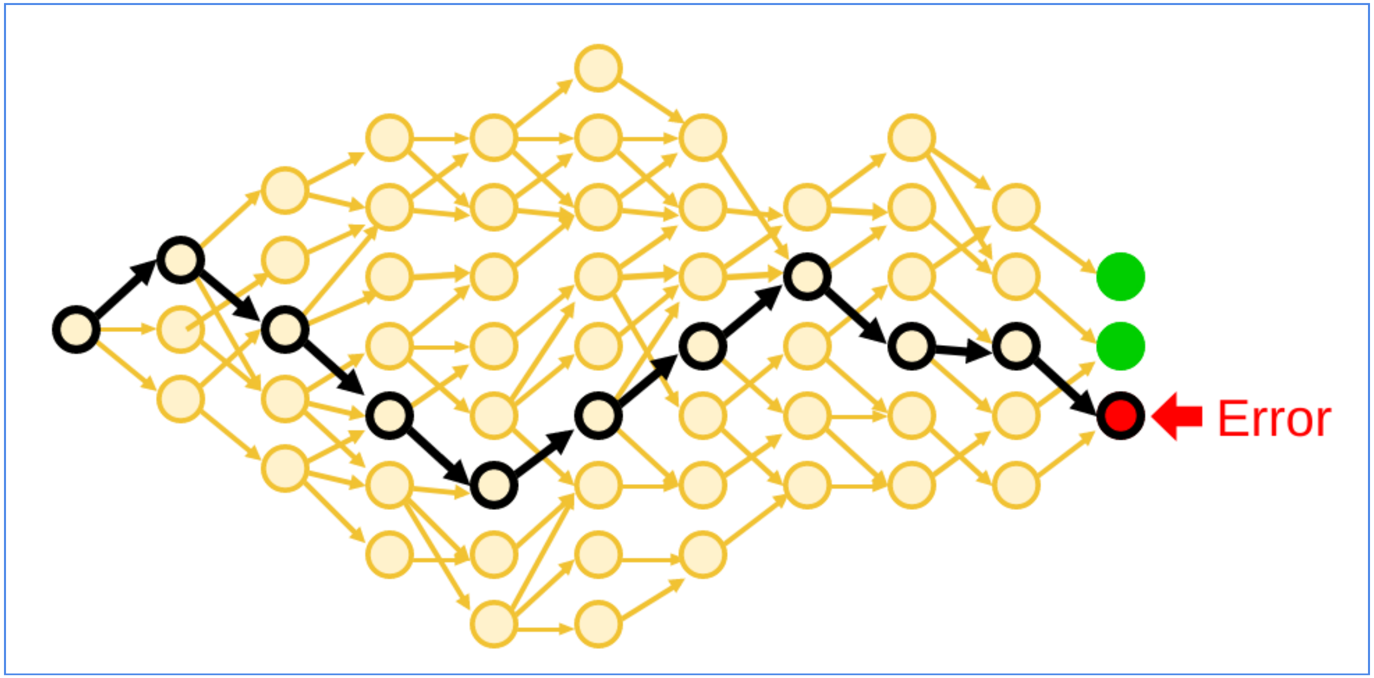
What Can we Prove About Neural Networks?

AI Institute Seminar, Nov 2021

Stony Brook University

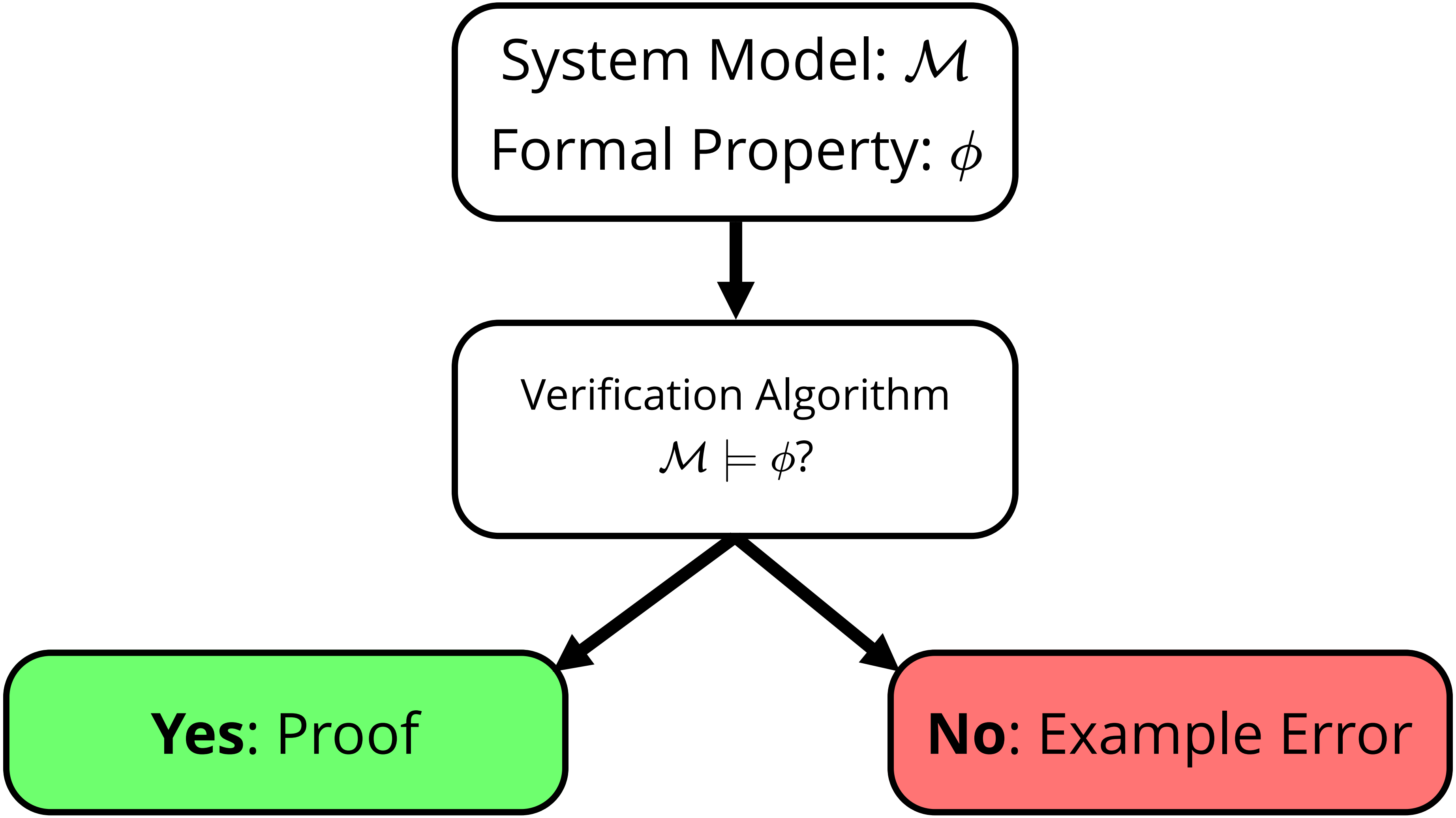
Formal Verification

In hardware circuits and software, formal verification methods can prove correctness in all cases



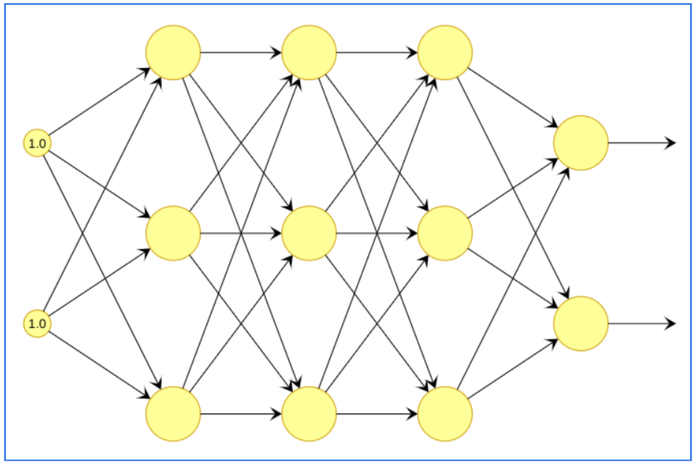
Model checking analyzes **all** possible behaviors

Formal Verification in the Abstract



What is Meant by Neural Network Verification?

Input Set



Output Set

$$i_1 \in [0, 1]$$
$$i_2 \in [0, 1]$$
$$\ldots$$
$$i_n \in [0, 1]$$

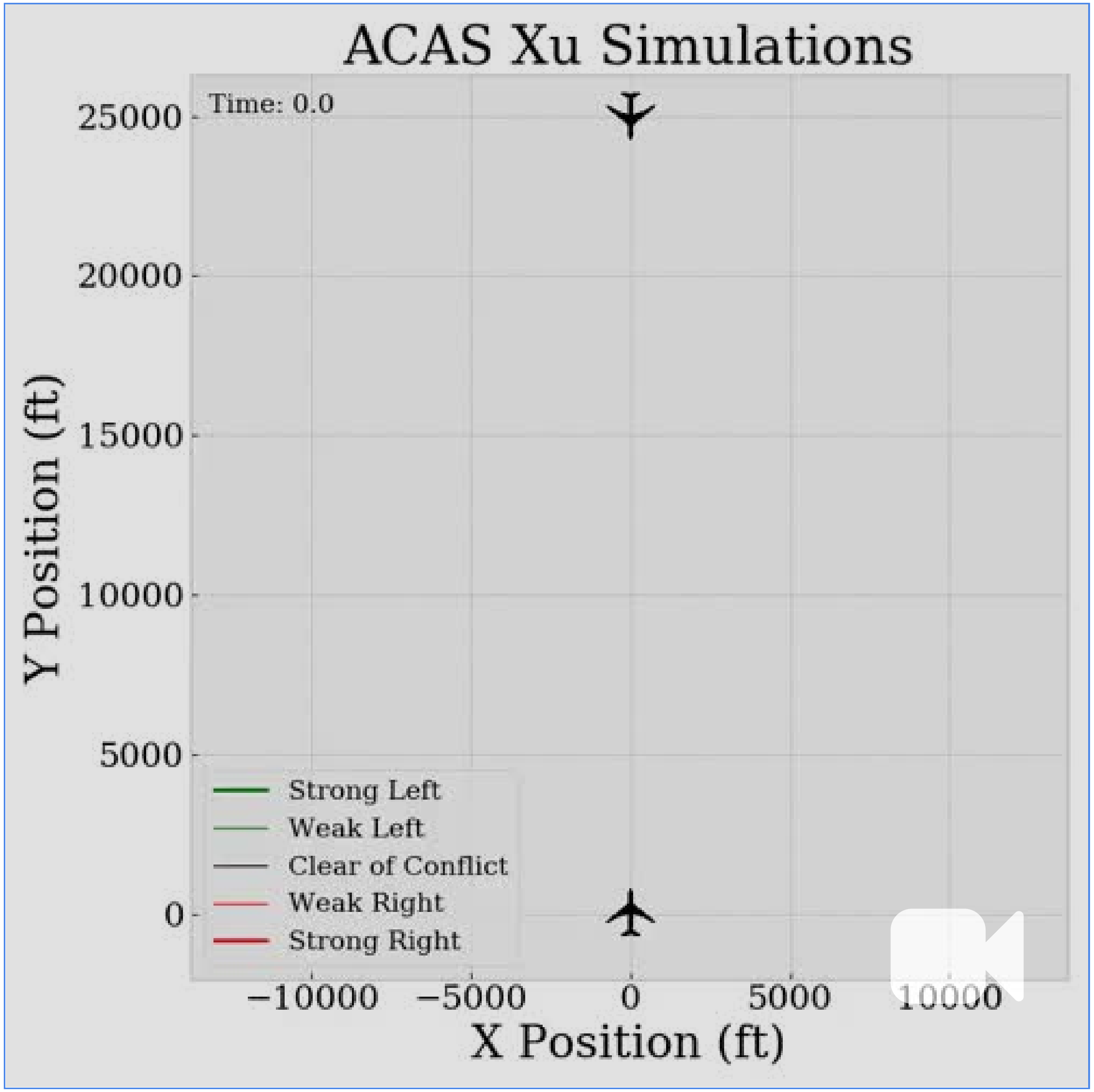
$$o_1 \geq o_2$$
$$o_1 \geq o_3$$
$$\ldots$$
$$o_1 \geq o_m$$

What Can we Prove About Neural Networks?

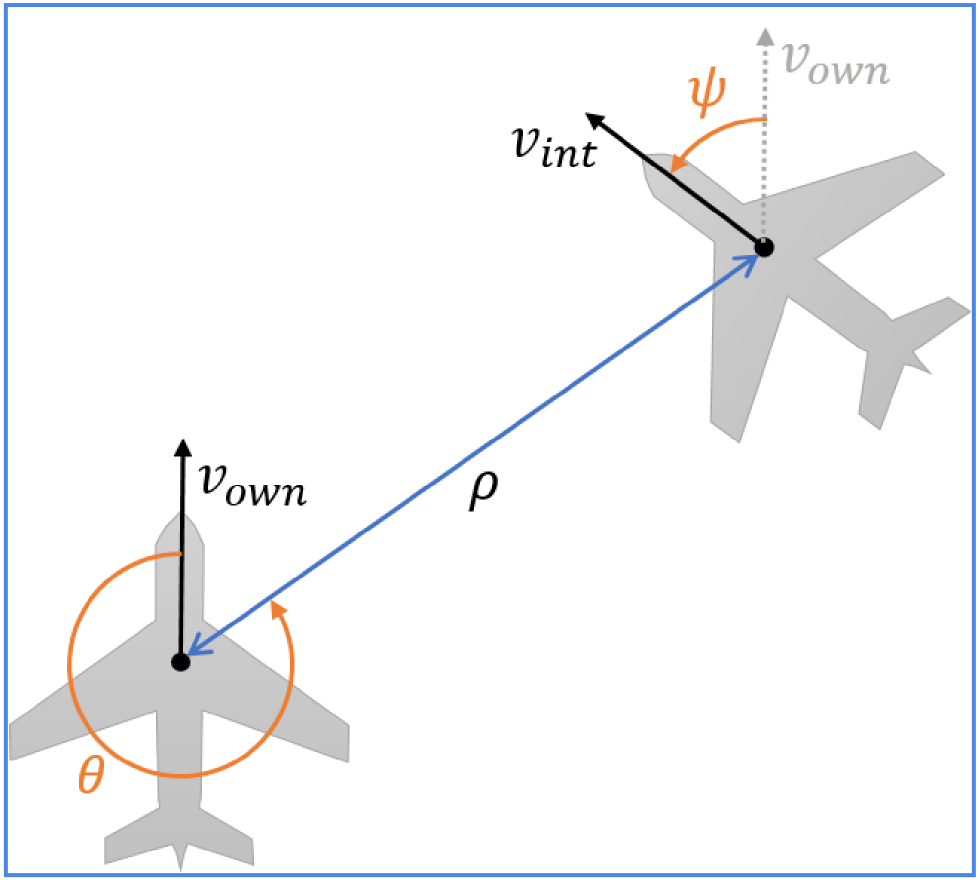
Theoretically?

Practically?

Verification Example 1: ACAS Xu Air-to-Air Collision Avoidance System



ACAS Xu Collision Avoidance System [Katz '17]

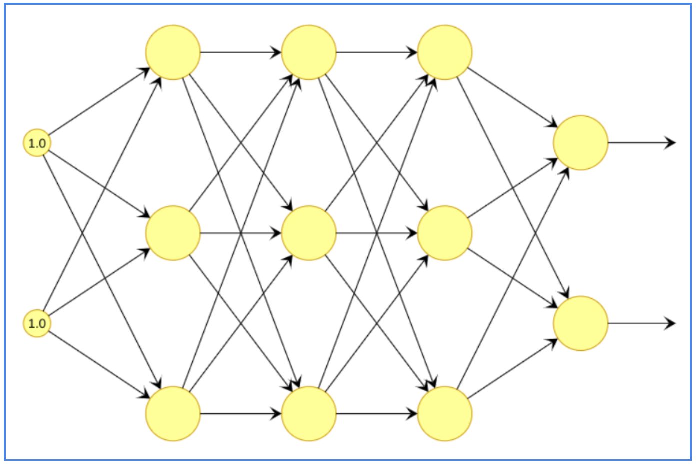


Why NN?: Replace a several GB lookup table with 45 neural networks (compression)

ACAS Xu Collision Avoidance System [Katz '17]

Inputs:

1. v_{int}
2. v_{own}
3. ρ
4. ψ
5. θ



300 neurons in 6 layers

Outputs:

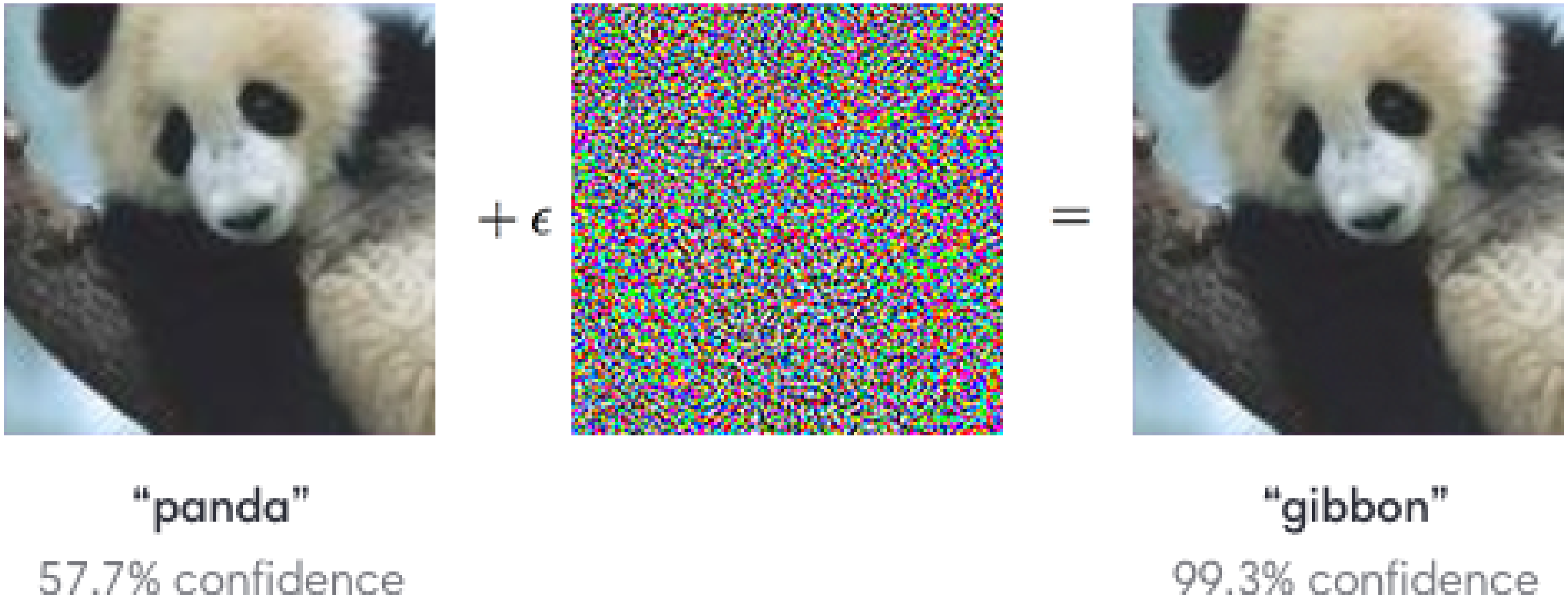
1. Clear
2. Weak-Left
3. Weak-Right
4. Strong-Left
5. Strong-Right

Property φ_3: If the intruder is directly ahead and is moving towards the ownship, a turn will be commanded.

Input: $1500 \leq \rho \leq 1800, |\theta| \leq 0.06, \psi \geq 3.1, v_{own} \geq 980, v_{int} \geq 960$

Unsafe Output: Clear $\leq$ Weak-Left $\wedge$ Clear $\leq$ Weak-Right $\wedge$ Clear $\leq$ Strong-Left $\wedge$ Clear $\leq$ Strong-Right

Verification Example 2: Proving the Absence of Adversarial Examples



"Explaining and harnessing adversarial examples.", Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. 2014

Verification Competition

Reachability for Verification

Other Recent Results

Verification Competition

Reachability for Verification

Other Recent Results

Comparison of Tools

VNN-COMP 2021

Organization and History of the Competition

- 1st VNN-COMP was in 2020
  - Friendly Competition
- 2nd VNN-COMP
  - Standardized Competition
  - Sponsorship



Goals: Standardized Competition

- Unified format for specifications: Vnnlib
- Unified format for NNs: onnx
- Common hardware: CPU and GPU

ONNX

VNN-LIB
Verification of Neural Networks

aws

Overview of Benchmarks

https://github.com/stanleybak/vnncomp2021/tree/main/benchmarks

Benchmark Name	Application	Network Types	Size of Each NN	Provider
Acasxu	Control	Feedforward + ReLU Only	**54.6k**	From last year
Cifar10_resnet	Image Classification	ResNet	440k, **487k**	CMU [US]
Cifar2020 (unscored)	Image Classification	Conv + ReLU	8.3M, **9.41M**	From last year
Eran	Image Classification	Feedforward + non-ReLU	1.37M, **1.68M**	ETH [Switzerland]
Marabou-cifar10	Image Classification	Conv + ReLU	336k, 649k, **1.29M**	Stanford [US]
Mnistfc	Image Classification	Feedforward + ReLU Only	1.03M, 1.53M, **2.03M**	Imperial College London [UK]
nn4sys	Database Indexing	Feedforward + ReLU Only	Zipped 1.79M, 790k Original 194.2M, **336.5M**	CMU, Northeastern [US]
Oval21	Image Classification	Conv + ReLU	216k, 415k, **840k**	Oxford [UK]
Verivital	Image Classification	Conv + maxpool / avgpool	46.3k, **46.3k**	Vanderbilt [US]

Overview of Tools (12 Tools)

GPU: p3.2xlarge, 8vCPUs, 61 GB memory, 1x V100 GPU, **$3.06/hour**
CPU: r5.12xlarge, 48vCPUs, 384 GB memory, no GPU, **$3.02/hour**

Tool Name	Institution of Participants	Link	CPU(r5.12xlarge) /GPU(p3.2xlarge)	Gurobi?
Marabou	Stanford [US]	https://github.com/anwu1219/Marabou_private	CPU	Yes
VeriNet	Imperial College London [UK]	https://vas.doc.ic.ac.uk/software/	CPU	
ERAN	ETH [Switzerland]	https://github.com/mnmueller/eran_vnncomp2021	GPU	Yes
Alpha-Beta-CROWN	CMU, Northeastern, Columbia, UCLA [US]	https://github.com/huanzhang12/alpha-beta-CROWN	GPU	
DNNF	U Virginia [US]	https://github.com/dlshriver/DNNF	CPU	
NNV	Vanderbilt [US]	https://github.com/verivital/nnv	CPU	
OVAL	Oxford [UK]	https://github.com/oval-group/oval-bab	GPU	Yes
NN-Reach	Stanford [US]	https://github.com/StanfordMSL/Neural-Network-Reach	CPU	
NeuralVerification.jl	CMU [US]	https://github.com/intelligent-control-lab/NeuralVerification.jl	CPU	
Venus	Imperial College London [UK]	https://github.com/pkouvaros/venus2_vnncomp21	CPU	Yes
Debona	RWTH Aachen [Germany]	https://github.com/ChristopherBrix/Debona	CPU	Yes
nnenum	Stony Brook [US]	https://github.com/stanleybak/nnenum	CPU	

Competition Challenges

Incorrect Results: Tools would lose points when results are wrong. How to judge what's wrong?

Scoring: Different tools support different architectures or layer types. What's the best way to perform scoring?

Overhead Measurement: Importing tensorflow / pytorch or initializing a GPU can take a few seconds. Some easier benchmarks could be checked in less than one second. How to judge fairly?

Common Hardware: We wanted to run things on identical hardware this year, what hardware to use?

and the winner is …

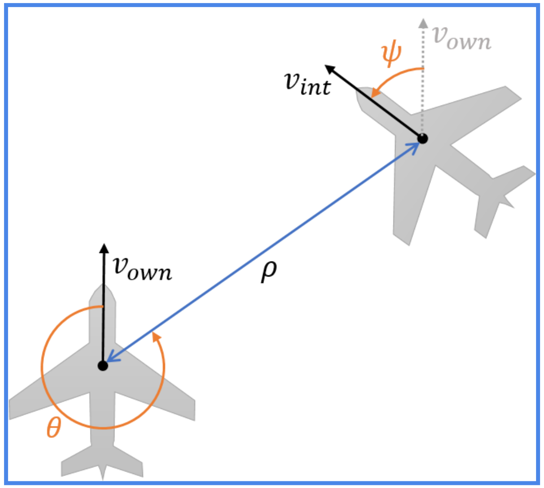
Voting:

1. **alpha-beta-CROWN: 776.67**
2. **VeriNet: 709.21**
3. **ERAN: 588.71**
4. oval: 588.38
5. Marabou: 302.14
6. Debona: 208.7
7. venus2: 194.56
8. nnenum: 194.21
9. nnv: 59.05
10. NeuralVerification.jl: 48.06
11. DNNF: 24.93
12. Neural-Network-Reach: 20.08
13. randgen: 1.84

But Reachability Methods Did Well Some of the Categories...

ACASXu

nnenum	1910	100.00%
VeriNet	1852	96.96%
Marabou	1809	94.71%
oval	1794	93.93%
venus2	1778	93.09%
a-b-CROWN	1732	90.68%
ERAN	1506	78.85%
Debona	1086	56.86%
NN-R	486	25.45%
nnv	348	18.22%
DNNF	182	9.53%
randgen	28	1.47%
NV.jl	-23	0%



Full VNN-COMP Results & Presentation:

https://docs.google.com/presentation/d/1oM3NqqU03EUqgQVc3bGK2ENgHa57u-W6Q63Vflkv000/edit?usp=sharing

Verification Competition

Reachability for Verification

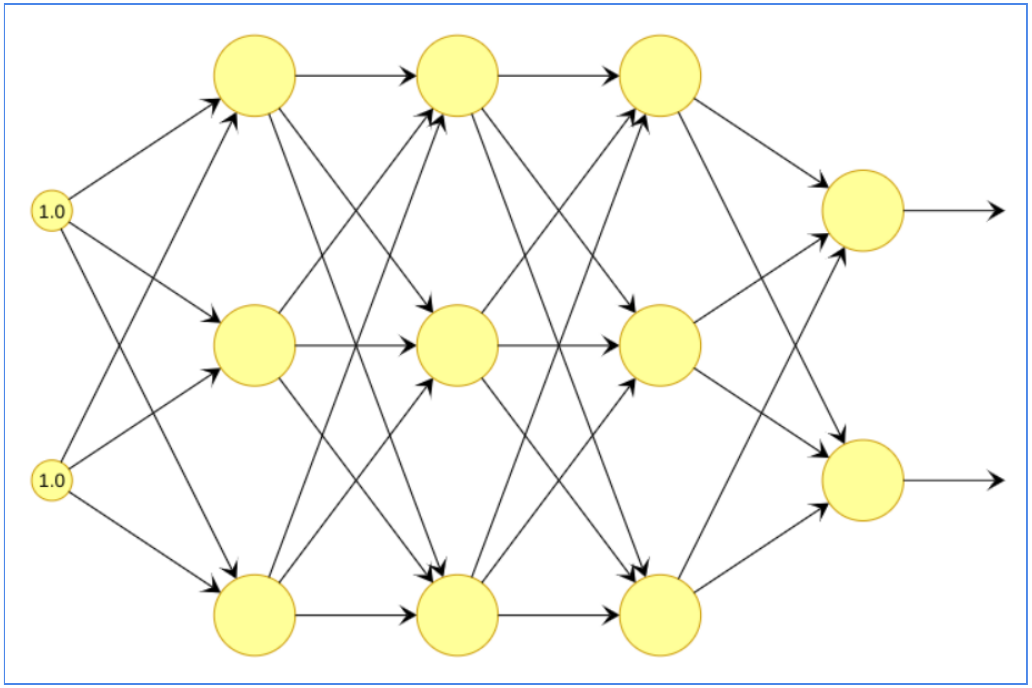
Other Recent Results

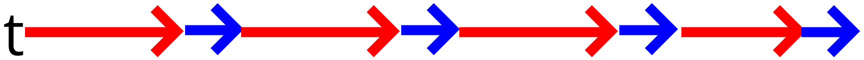
Technical Methods

So how do you verify a neural network anyway?

Neural Network Execution

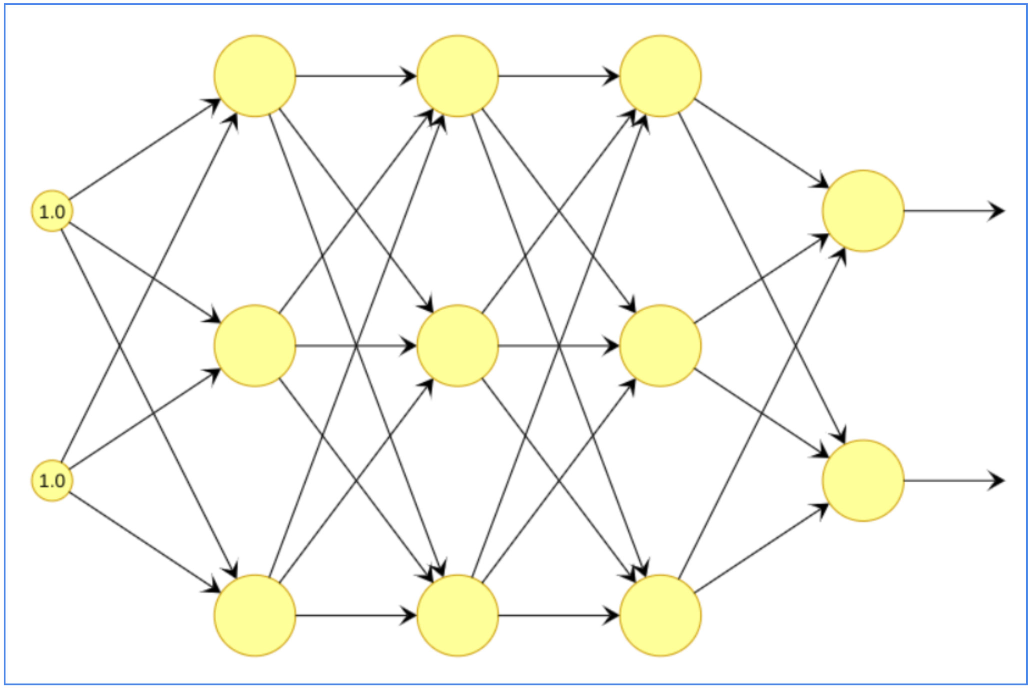
Executing fully-connected neural networks uses two operations:
(1) affine transformations, and (2) activation functions.

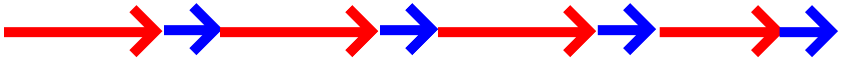


Input Point  Output Point

Two Set Operations Needed

Verification needs two types of **set** operations:
(1) affine transformations, and (2) activation functions.



Input **Set**  Output **Set**

Affine Transform

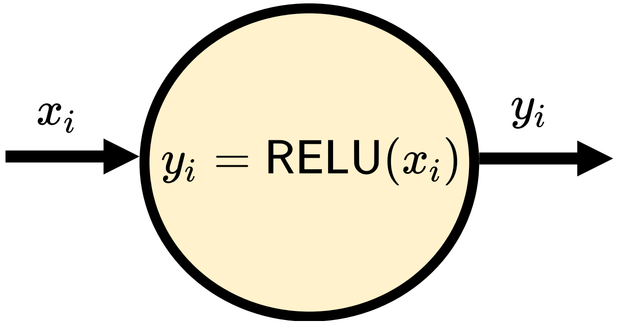
An **affine transformation** f is a function that transforms an n-dimensional point x to a q-dimensional point defined using a matrix A and vector b.

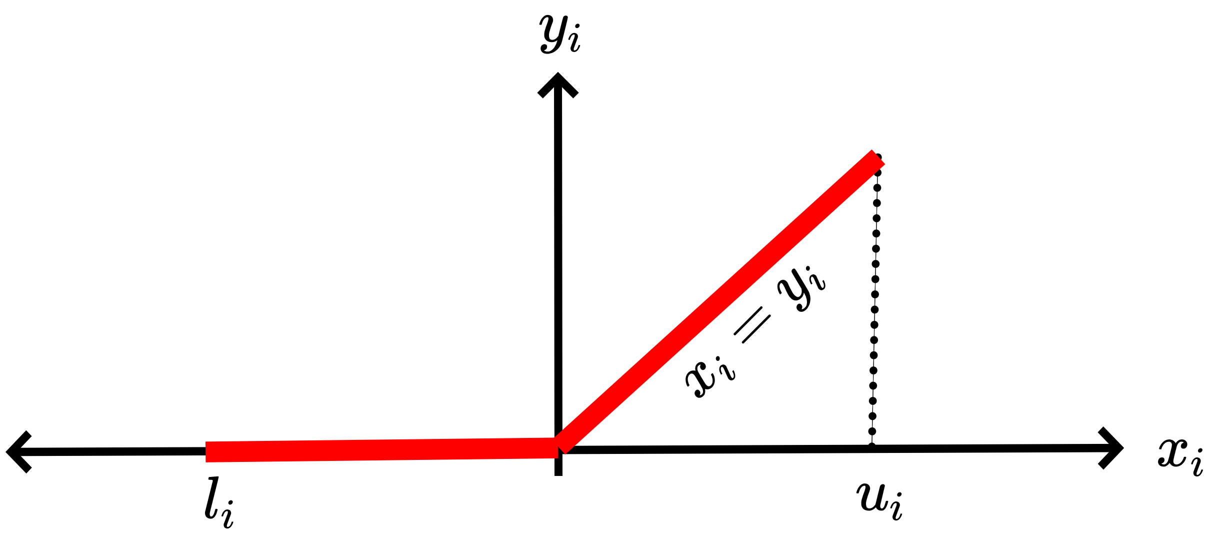
$$f(x) : \mathbb{R}^n \to \mathbb{R}^q$$
$$x \mapsto Ax + b$$

If x is a vector of n outputs of some layer, then the q inputs to the next layer are $Ax + b$, where A is the weights matrix and b is the bias vector.

ReLU Activation Functions

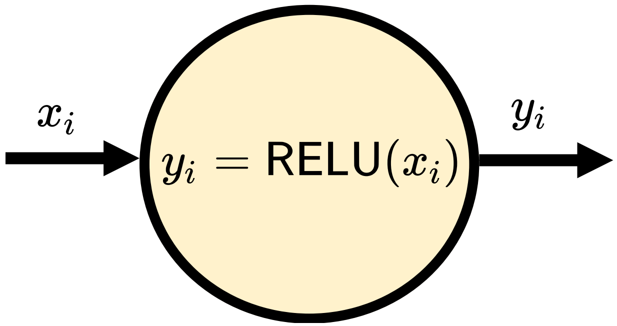
$$\text{RELU}(x) = \max(x, 0)$$

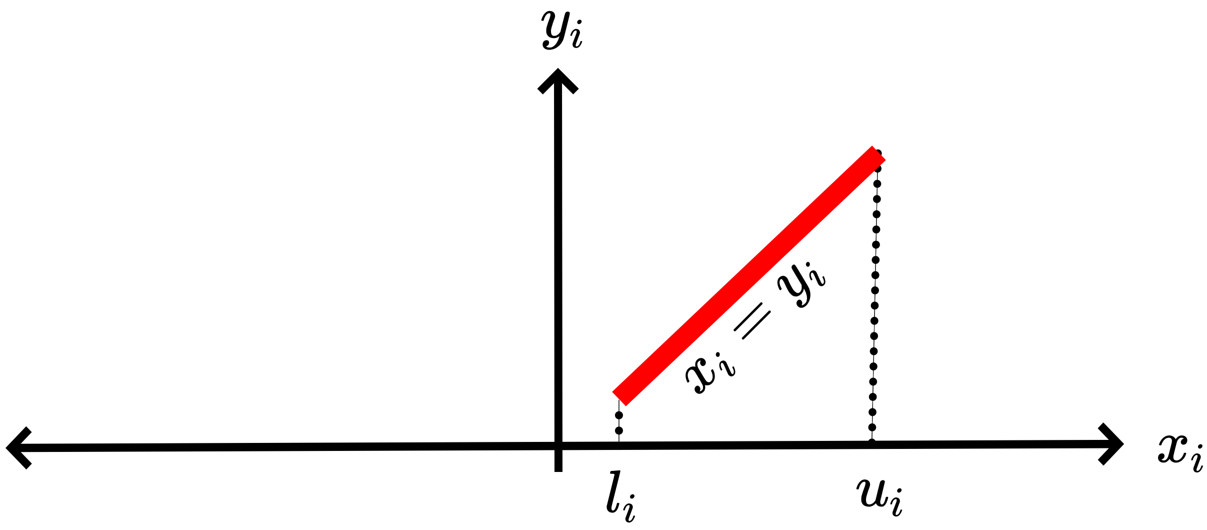




ReLU Activation Functions

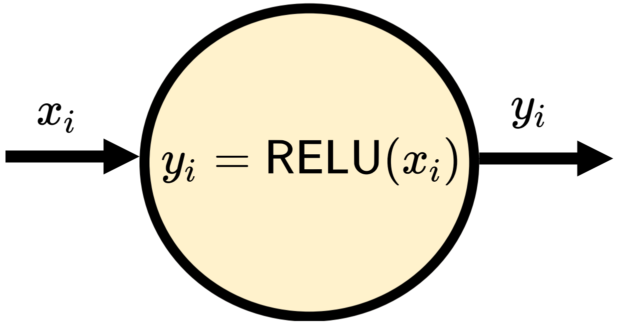
$$\text{RELU}(x) = \max(x, 0)$$

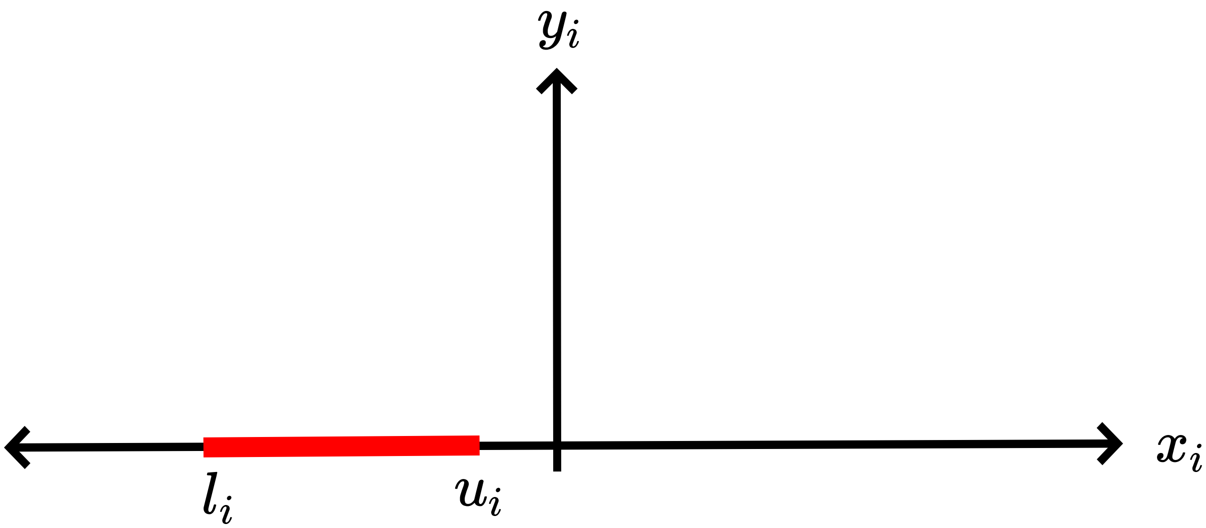




ReLU Activation Functions

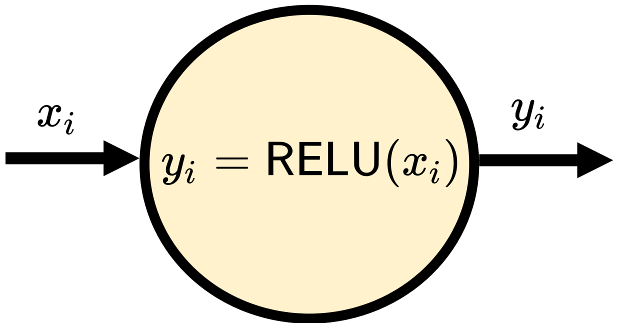
$$\mathsf{RELU}(x) = \max(x, 0)$$

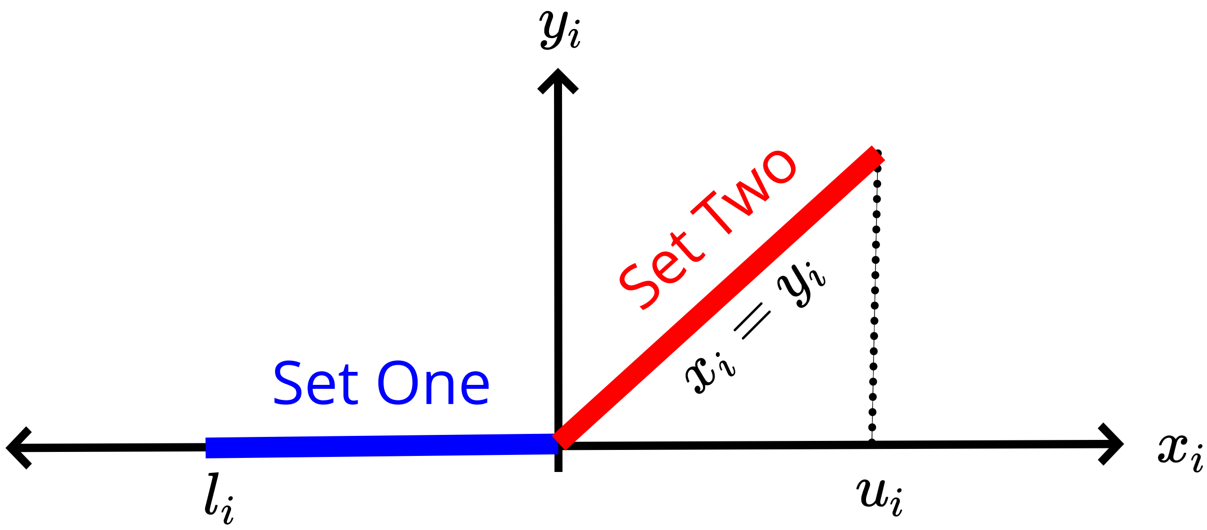




ReLU Activation Functions

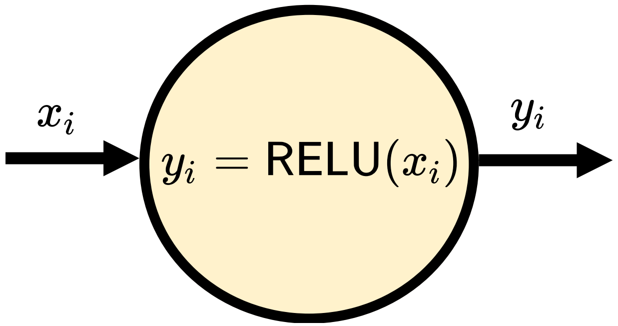
$$\text{RELU}(x) = \max(x, 0)$$

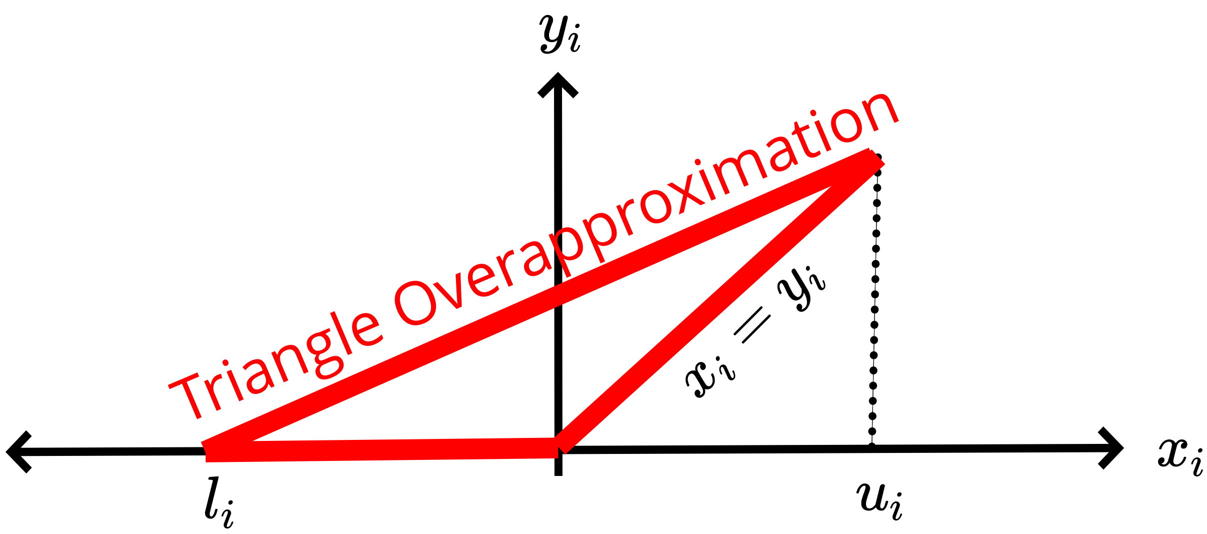




ReLU Activation Functions

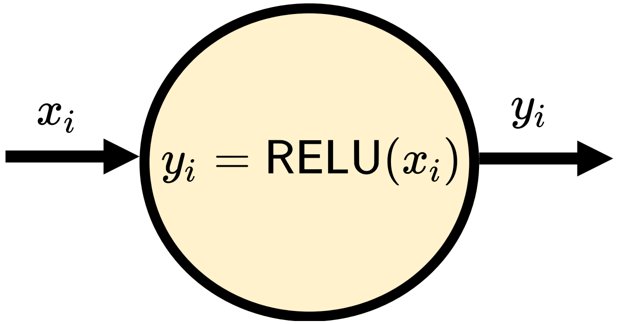
$$\text{RELU}(x) = \max(x, 0)$$

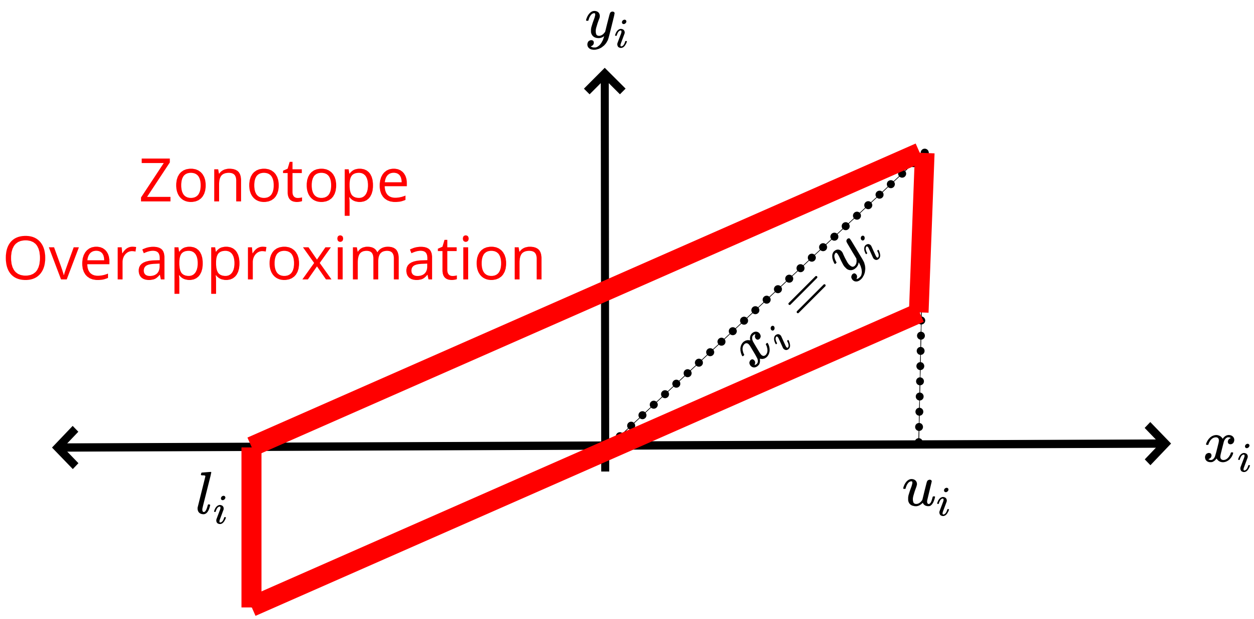




ReLU Activation Functions

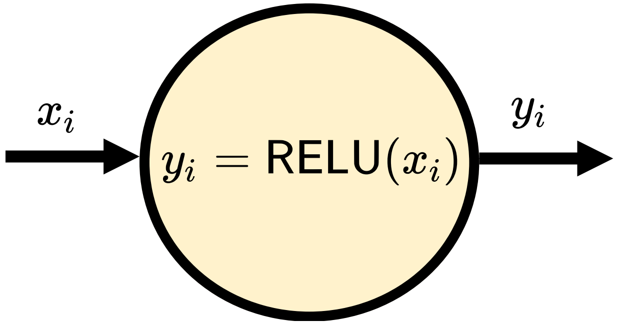
$$\text{RELU}(x) = \max(x, 0)$$



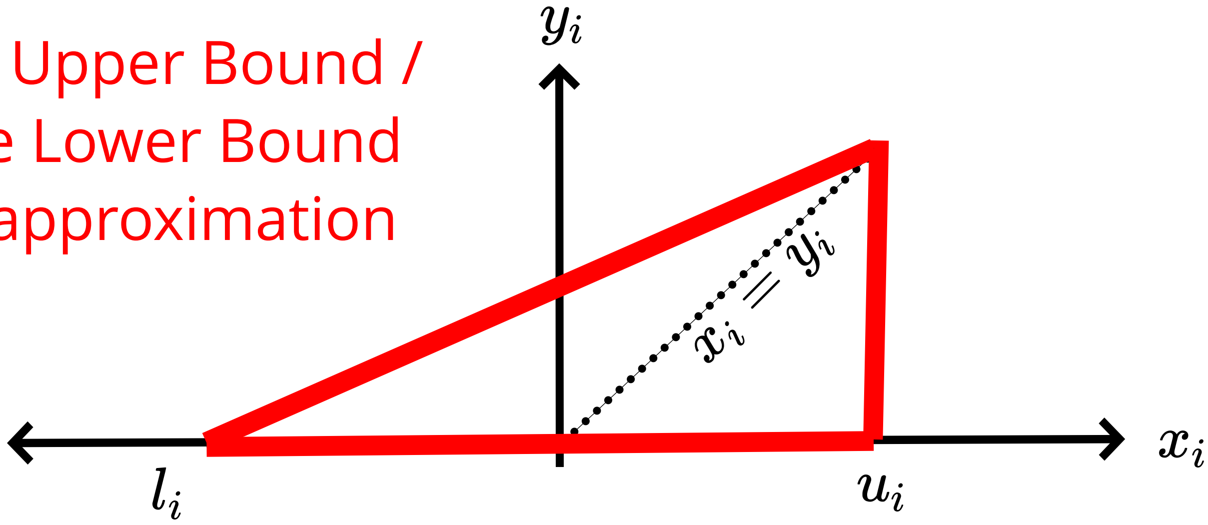


ReLU Activation Functions

$$\text{RELU}(x) = \max(x, 0)$$

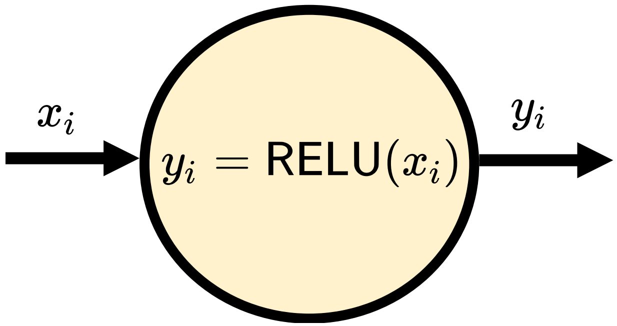


Single Upper Bound /
Single Lower Bound
Overapproximation

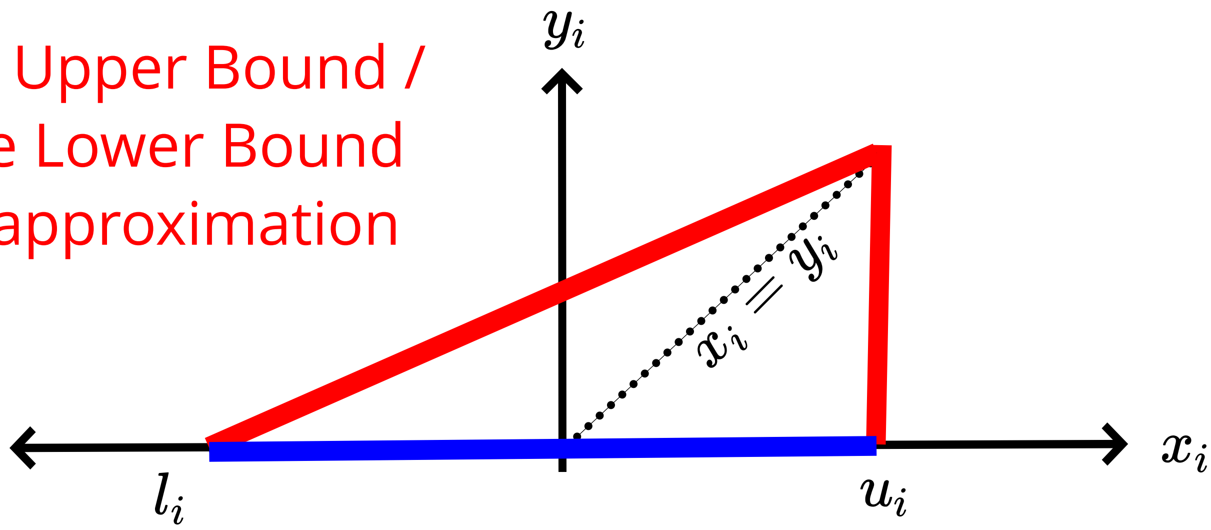


ReLU Activation Functions

$$\text{RELU}(x) = \max(x, 0)$$

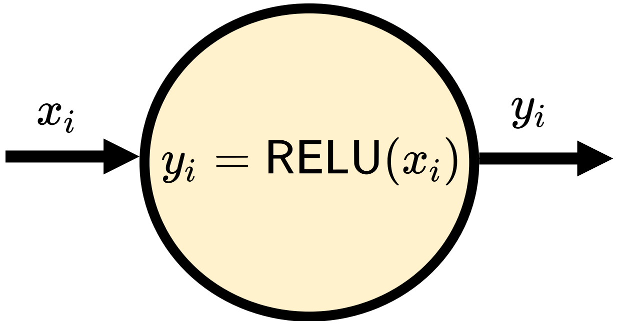


Single Upper Bound / Single Lower Bound Overapproximation

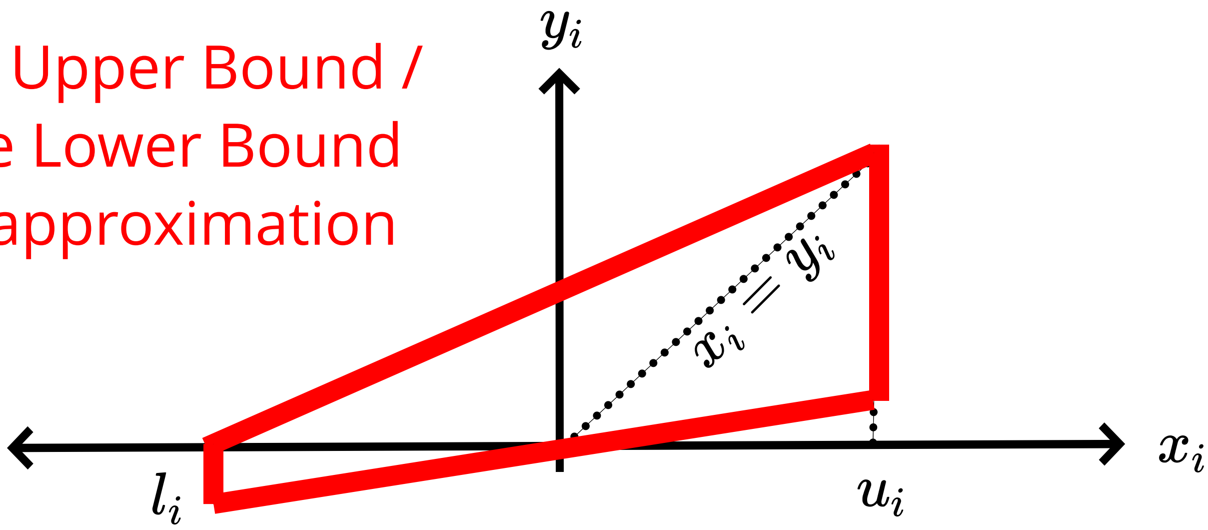


ReLU Activation Functions

$$\mathsf{RELU}(x) = \max(x, 0)$$



Single Upper Bound /
Single Lower Bound
Overapproximation



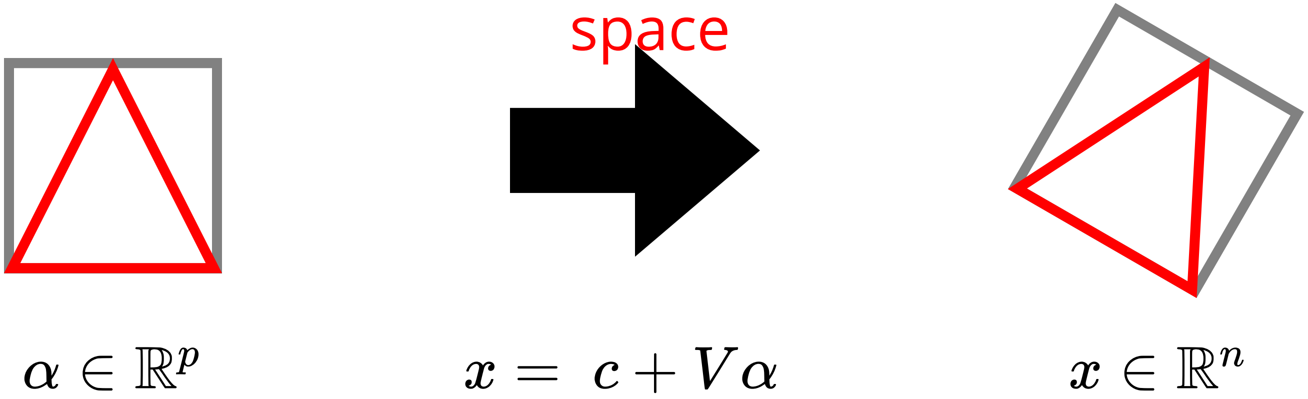
Zonotope and Star Set Intuition

$$Z = \{x \in \mathbb{R}^n \mid x = c + V\alpha, \alpha \in [-1, 1]^p\}$$

$$S = \{x \in \mathbb{R}^n \mid x = c + V\alpha, \alpha \in Cx \leq d\}$$

A zonotope is a set of points defined with an affine transformation from a p-dim unit box to an n-dim space

A linear star set is a set of points defined with an affine transformation from a p-dim **polytope** to an n-dim space



Operations on Linear Star Set $\langle c, V, P \rangle$

Affine Transformation: matrix-matrix multiplication to compute c' and V'. Result is $\langle c', V', P \rangle$.

Optimization: put star set definition into a linear program (LP) and minimize.

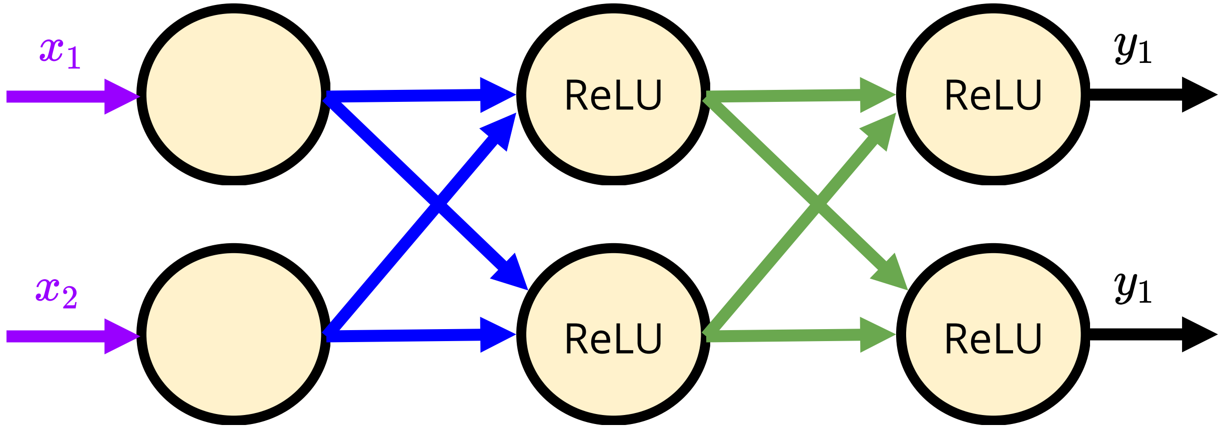
Intersection: given a halfspace $H = \{x \mid Gx \leq g\}$, let $P_H(\alpha) = GV\alpha \leq g - Gc$. Result is $\langle c, V, P \wedge P_H \rangle$.

Numerical Example

Initial Set:

$x_1 \in [0.5, 1]$

$x_2 \in [0, 2]$



Rotate 45 degrees:

$$W = \begin{pmatrix} \cos(\frac{\pi}{4}) & -\sin(\frac{\pi}{4}) \\ \sin(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \end{pmatrix}$$

$$b = (0, 0)^T$$

Translate down:

$$W = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$b = (0, -\frac{\sqrt{2}}{2})$$

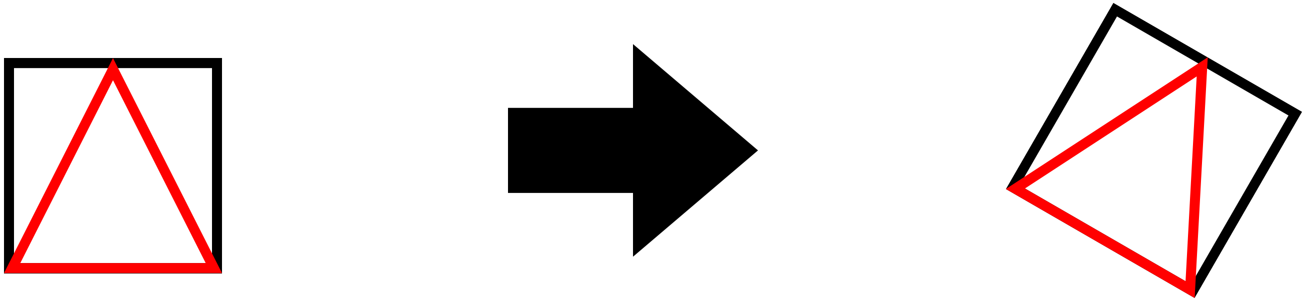
Zonotope and Star Set

$$Z = \{x \in \mathbb{R}^n \mid x = c + V\alpha, \alpha \in [-1,1]^p\}$$

$$S = \{x \in \mathbb{R}^n \mid x = c + V\alpha, \alpha \in Cx \leq d\}$$

A zonotope is a set of points defined with an affine transformation from a p-dim unit box to an n-dim space

A linear star set is a set of points defined with an affine transformation from a p-dim polytope to an n-dim space

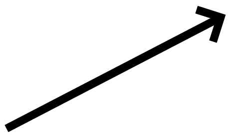


$\alpha \in \mathbb{R}^p$ $\qquad x = c + V\alpha$ $\qquad x \in \mathbb{R}^n$

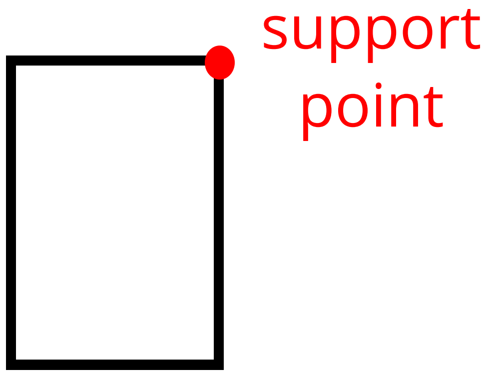
In our example, $p = n = 2$. Using star sets, all operations (intersection, optimization), are performed in the input space.

Notes on Zonotopes

Optimization on zonotopes is quick. Why? It becomes an optimization problem over rectangles in the input space, which can be done with a simple loop.



Optimization problem: Maximize in this direction



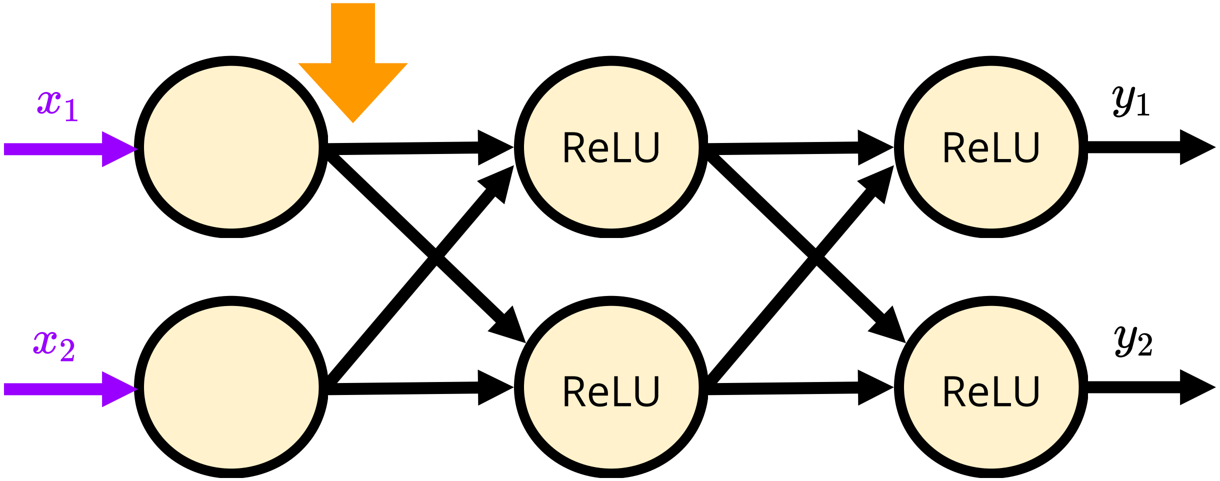
Subject to being inside this rectangle

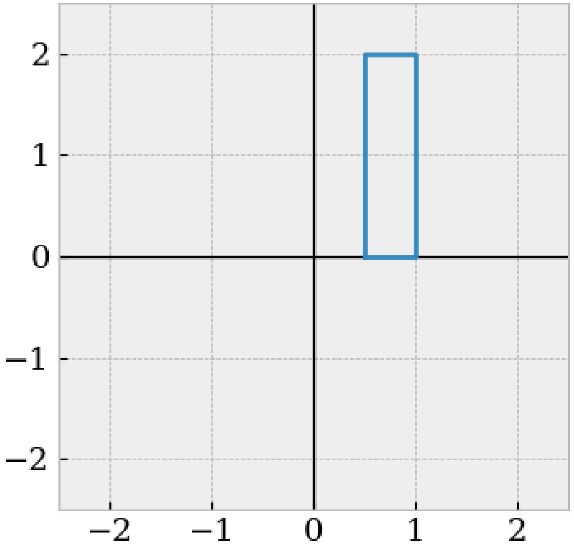
```
support_pt = []

for d in range(dims):
    if optimization_vec[d] > 0:
        dim_value = box[d].upper_bound
    else:
        dim_value = box[d].lower_bound

    support_pt.append(dim_value)
```

Initial Set:

$x_1 \in [0.5, 1]$

$x_2 \in [0, 2]$



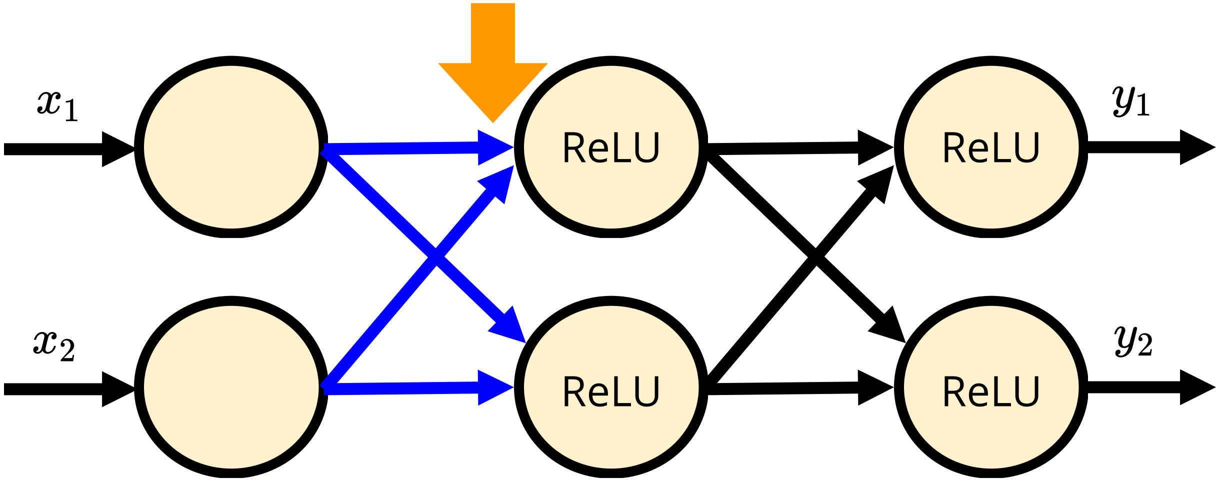


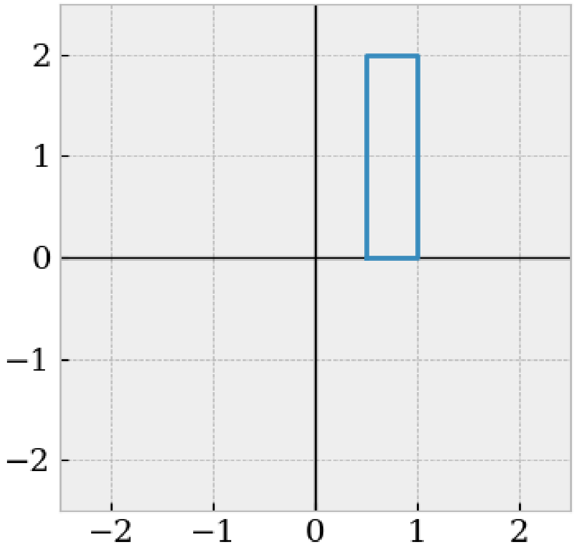
Input Space

Rotate 45 degrees:

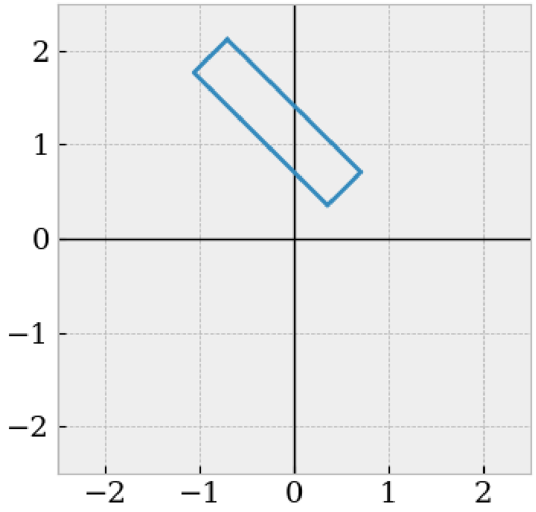
$$W = \begin{pmatrix} \cos(\frac{\pi}{4}) & -\sin(\frac{\pi}{4}) \\ \sin(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \end{pmatrix}$$

$$b = (0, 0)^T$$

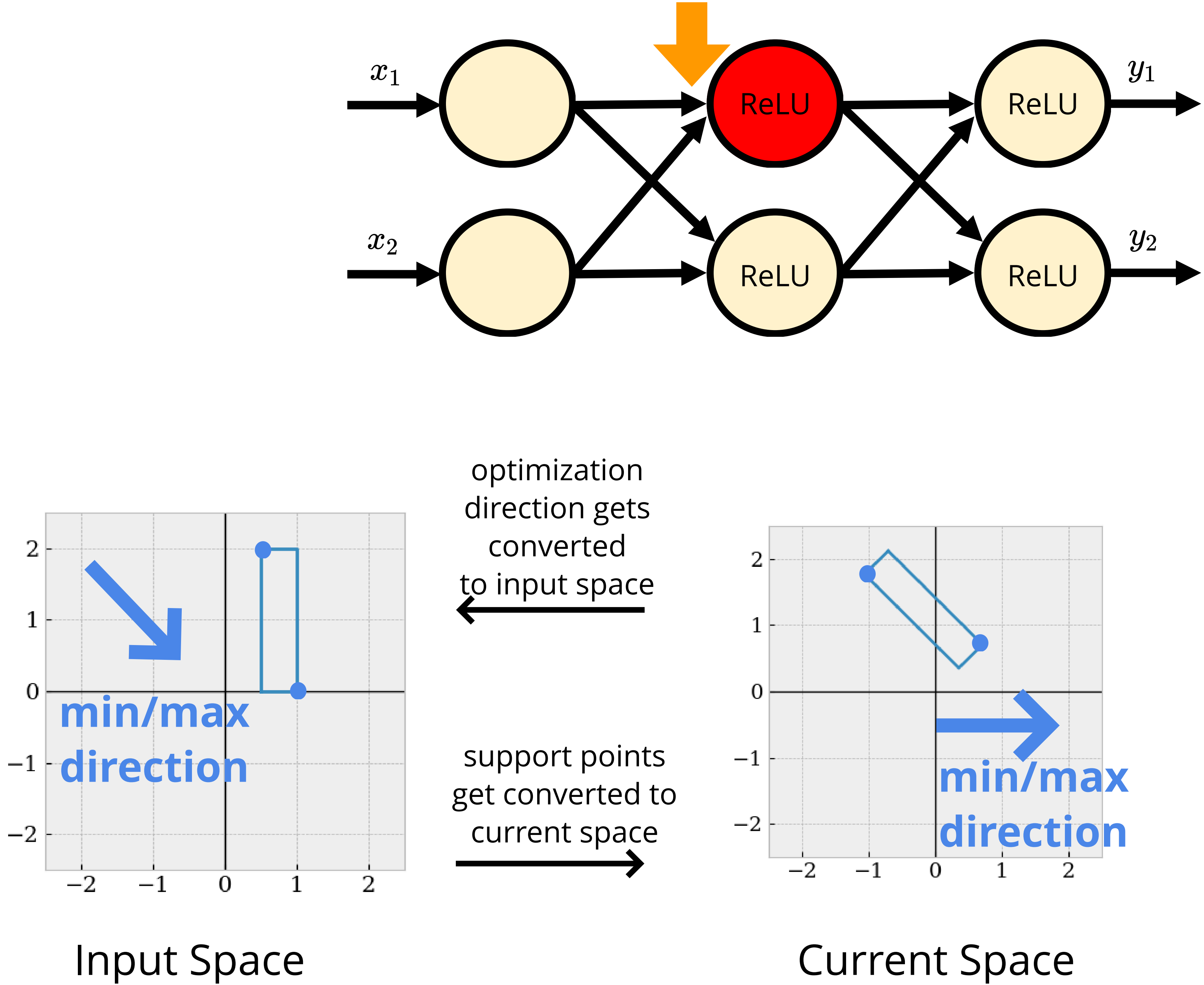




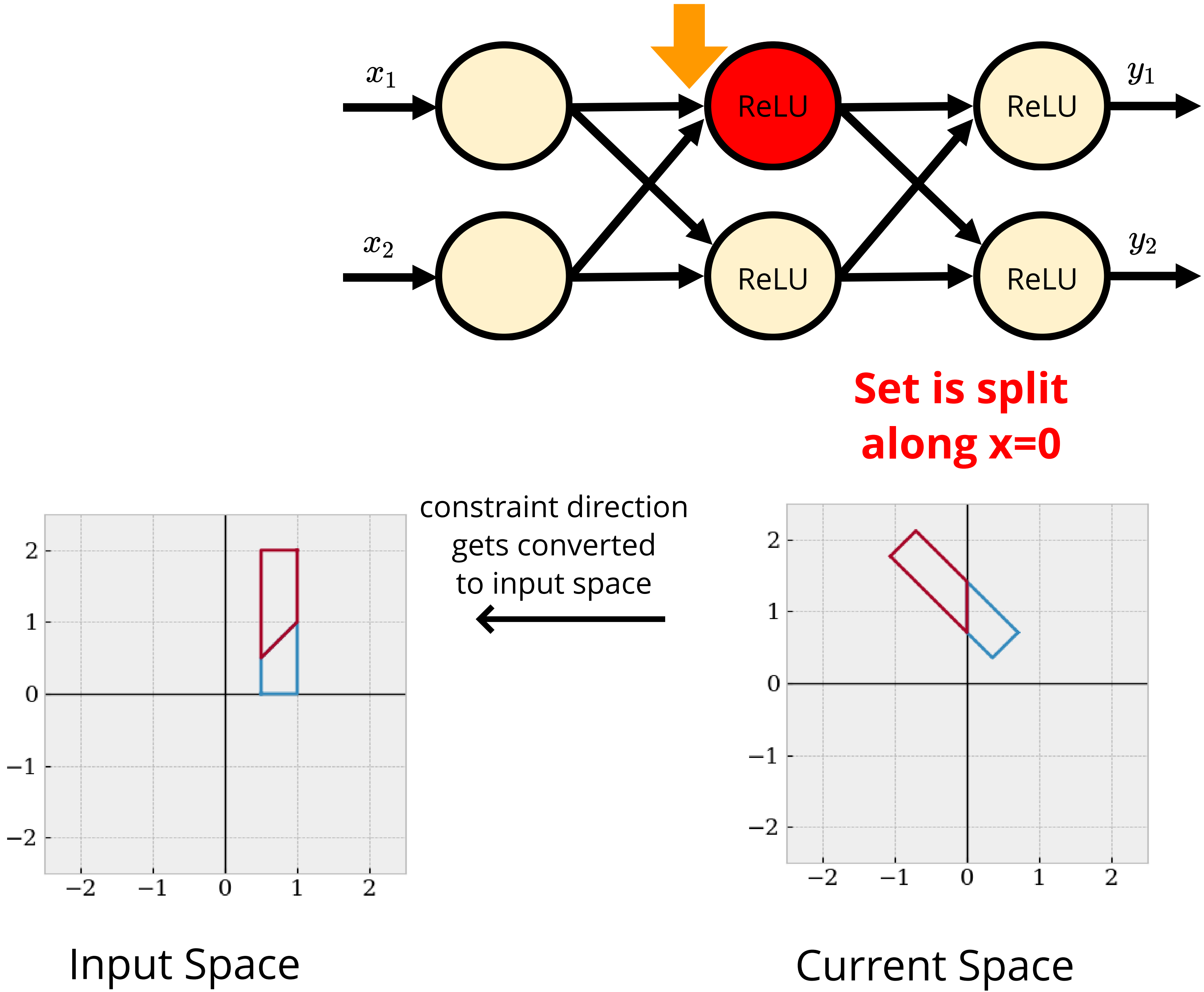
Input Space



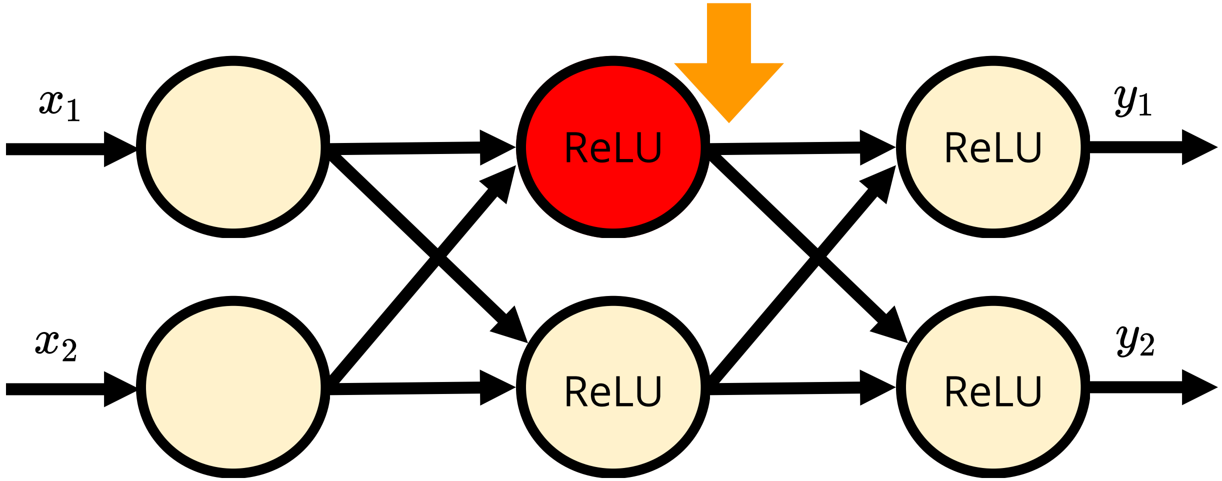
Current Space



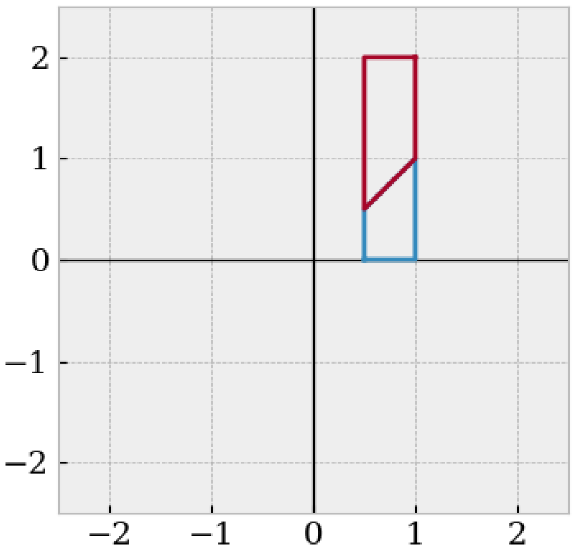
x_1
x_2
ReLU
ReLU
ReLU
ReLU
y_1
y_2
optimization direction gets converted to input space
support points get converted to current space
min/max direction
min/max direction
Input Space
Current Space



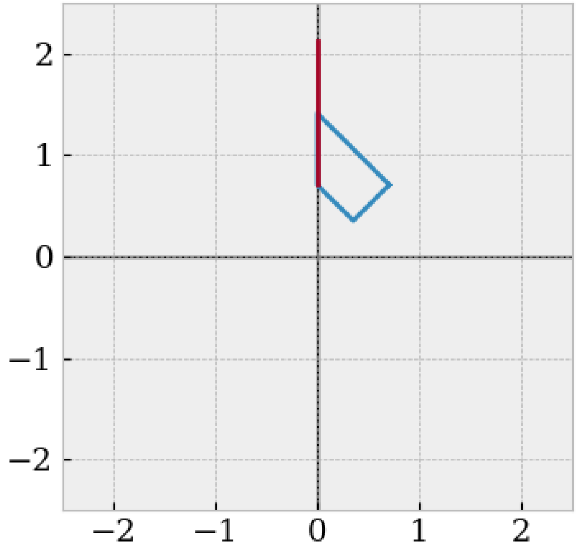
x_1
x_2
ReLU
ReLU
ReLU
ReLU
y_1
y_2
Set is split
along x=0
constraint direction
gets converted
to input space
Input Space
Current Space



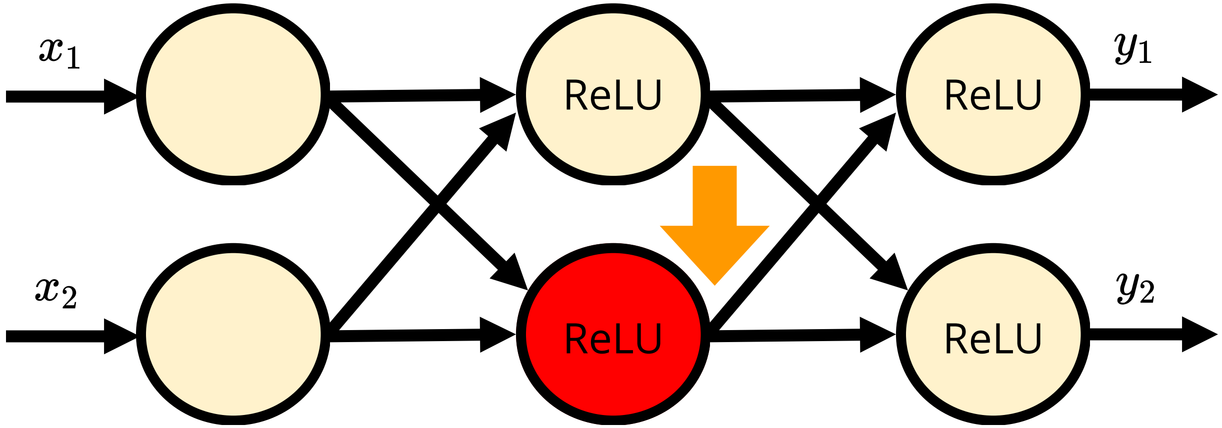
Negative set projected to x=0



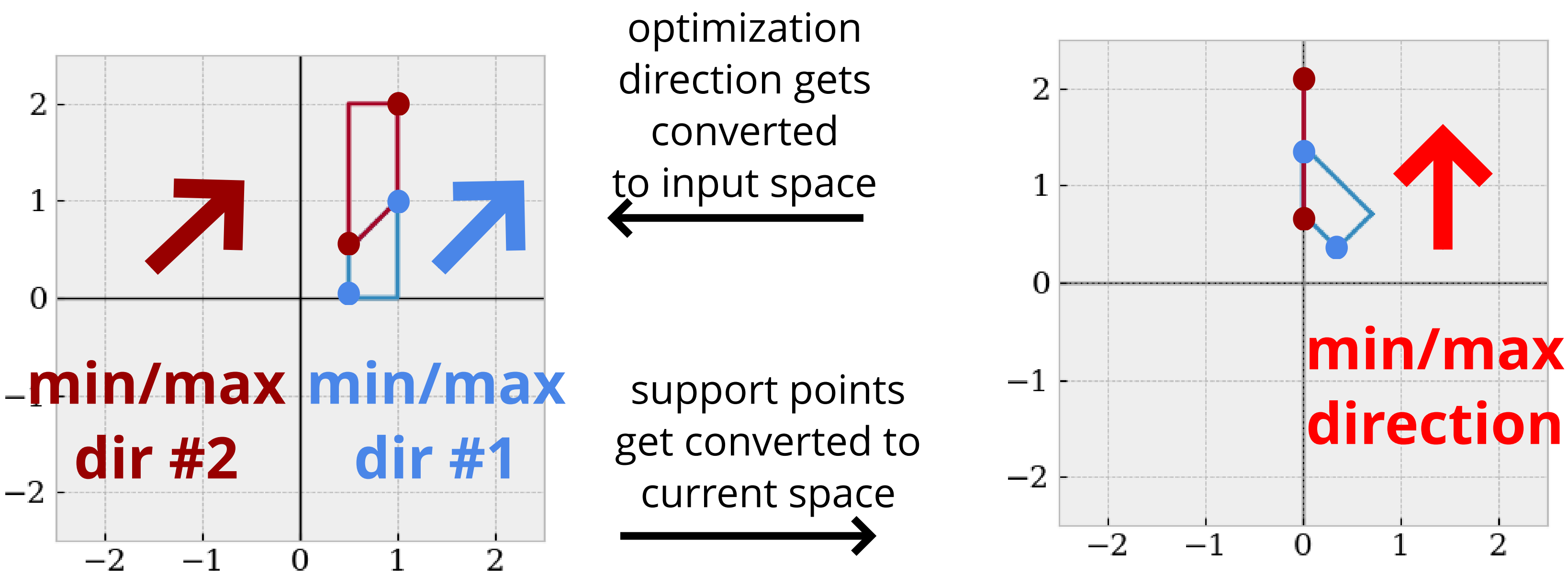
Input Space



Current Space



No splitting is possible for second ReLU (along y=0)



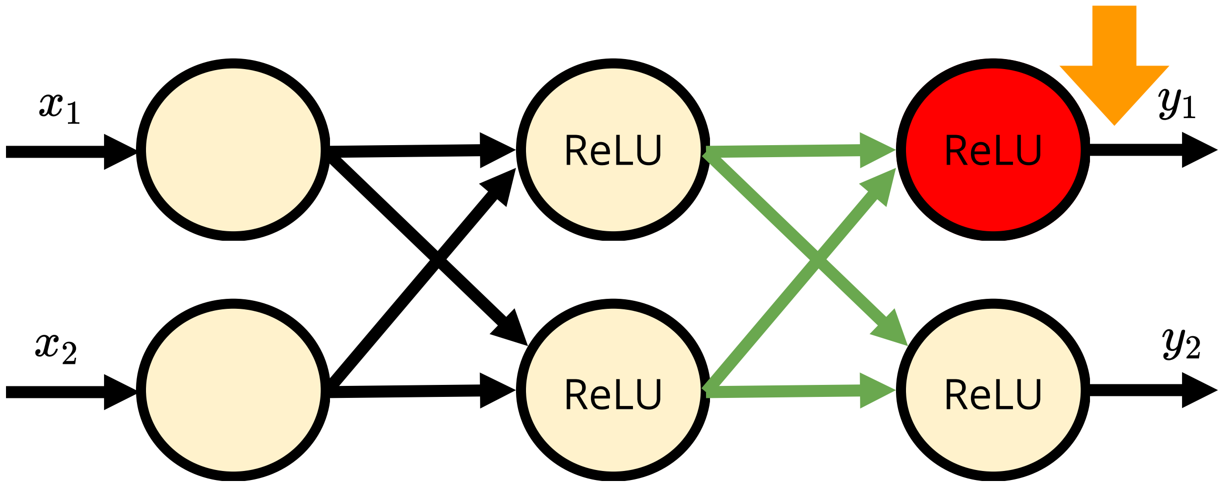
Input Space

Current Space

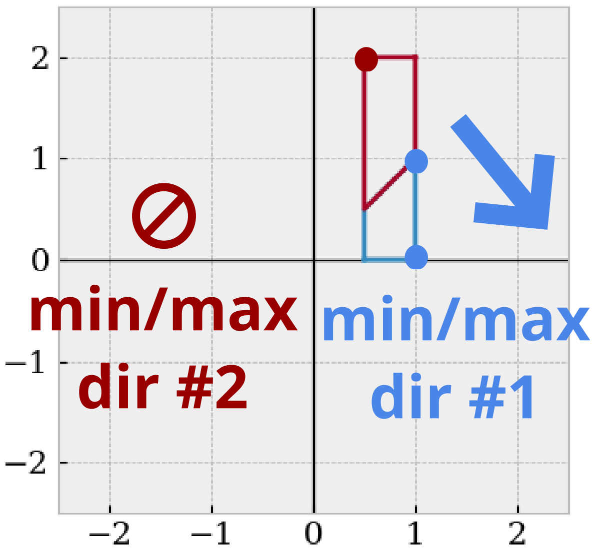
Translate down:

$$W = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$b = (0, -\frac{\sqrt{2}}{2})$$



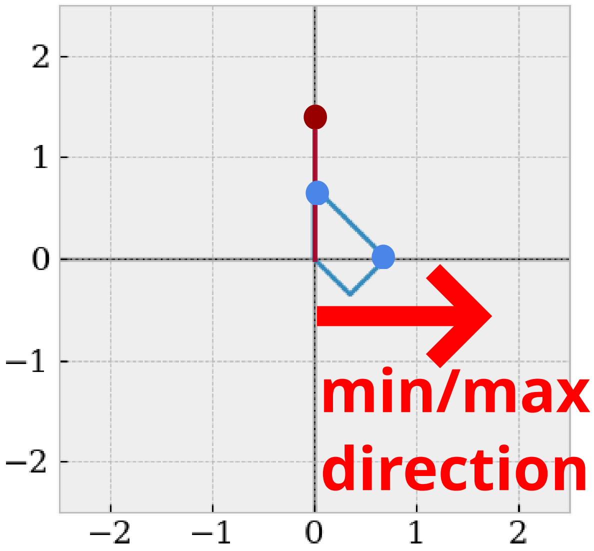
No splitting is possible for first ReLU (along x=0)



Input Space

optimization direction gets converted to input space

support points get converted to current space

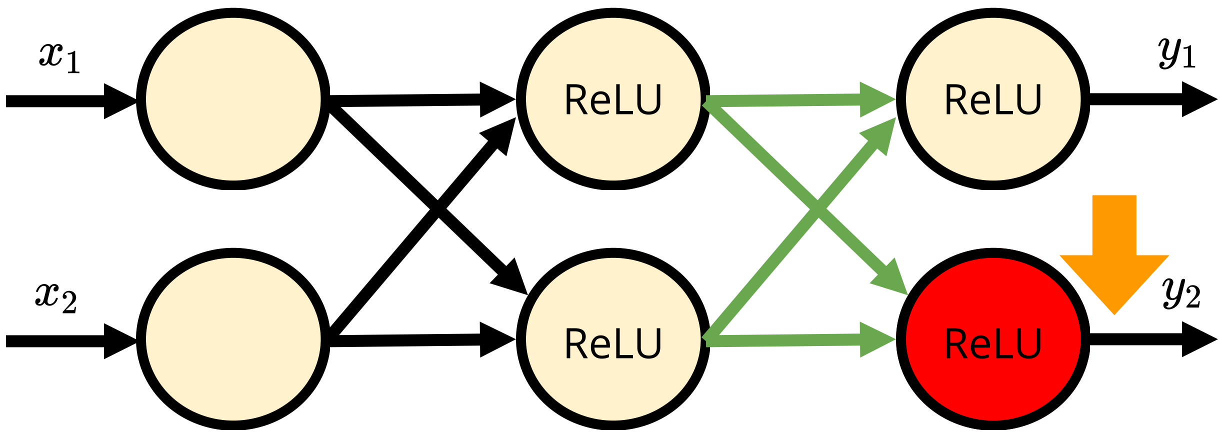


Current Space

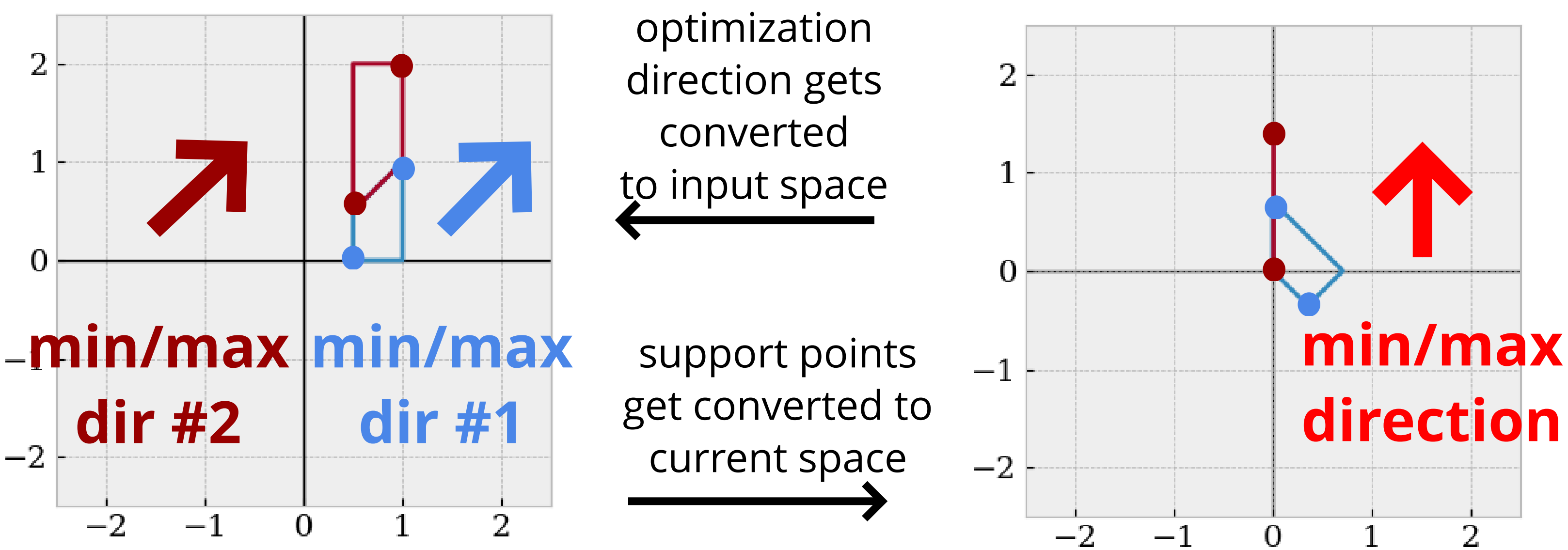
Translate down:

$$W = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$b = (0, -\frac{\sqrt{2}}{2})$$

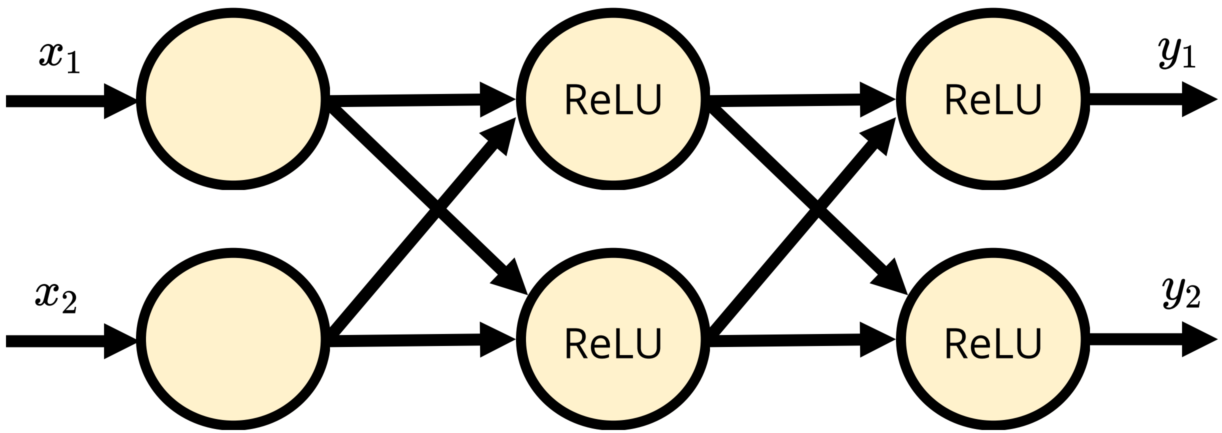


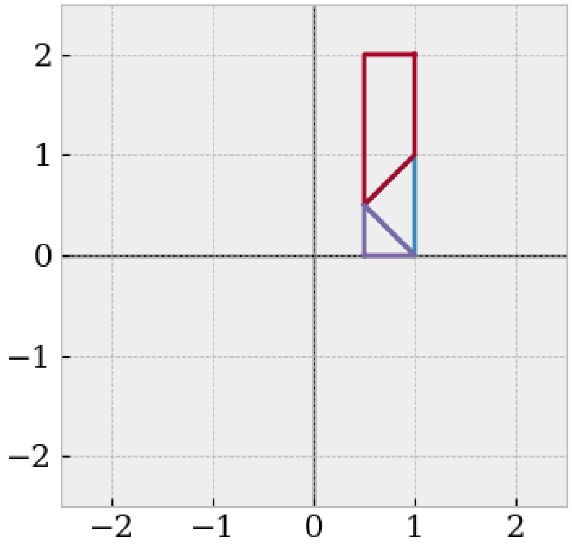
Splitting is needed for second ReLU for blue set (along y=0)



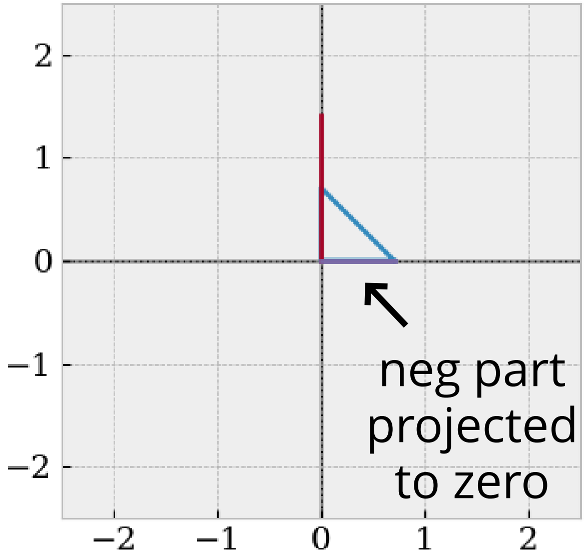
Input Space

Current Space





Final Input Space



Final Output Space

Efficiency

How do you speed things up?

Ideas from two papers:

"Improved Geometric Path Enumeration for Verifying ReLU Neural Networks", S. Bak, H.D Tran, K. Hobbs and T. T. Johnson, 32nd International *Conference on Computer-Aided Verification* (CAV 2020)

"nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement.", Bak, Stanley. *NASA Formal Methods Symposium* (NFM 2021)

Two Paths to Improvement

Create
New
Algorithms

Optimize
Existing
Algorithms

Formal Methods

"Engineering matters: you can't properly evaluate a technique without an efficient implementation."

-Ken McMillan

Optimization: Measure Don't Guess

To improve performance, you must first find the bottleneck of the algorithm.

The majority of the runtime is spent making unnecessary copies.

Optimization: Measure Don't Guess

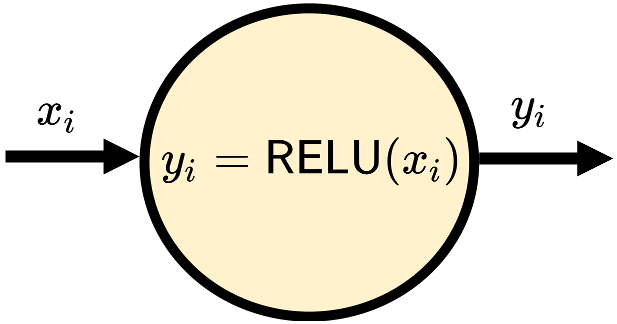
To improve performance, you must first find the bottleneck of the algorithm.

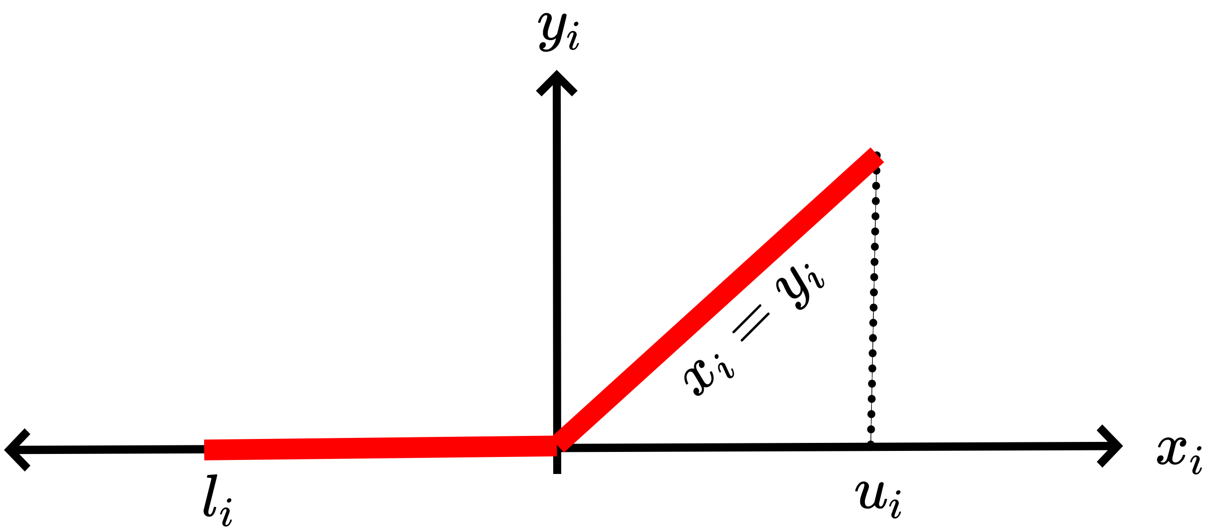
~~The majority of the runtime is spent making unnecessary copies.~~

The majority of the runtime is spent optimizing (solving LPs), to find the input bounds for each neuron.

ReLU Activation Functions

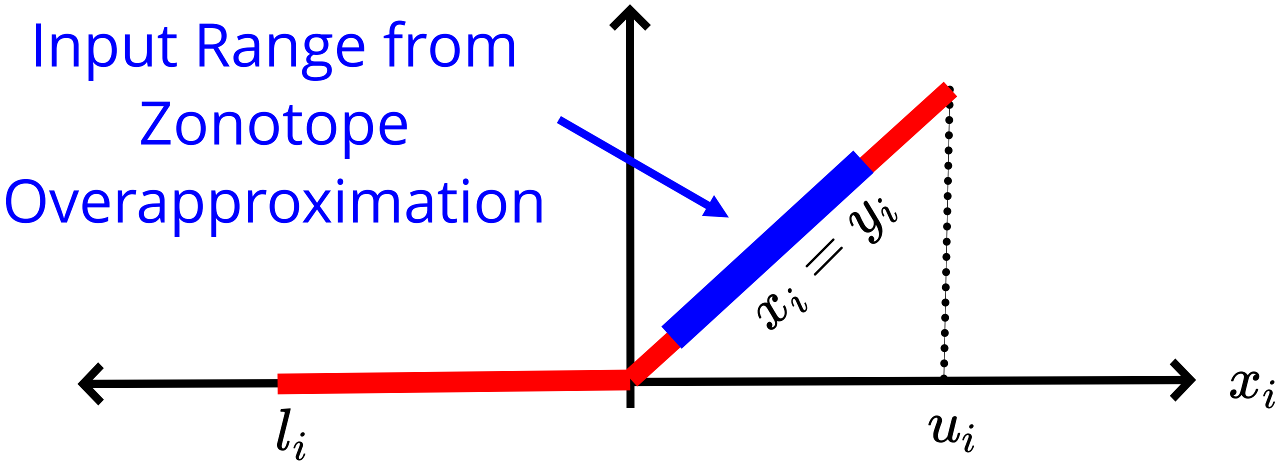
$$\text{RELU}(x) = \max(x, 0)$$





Two LPs are solved to find l_i and u_i for each neuron.

LP Reductions

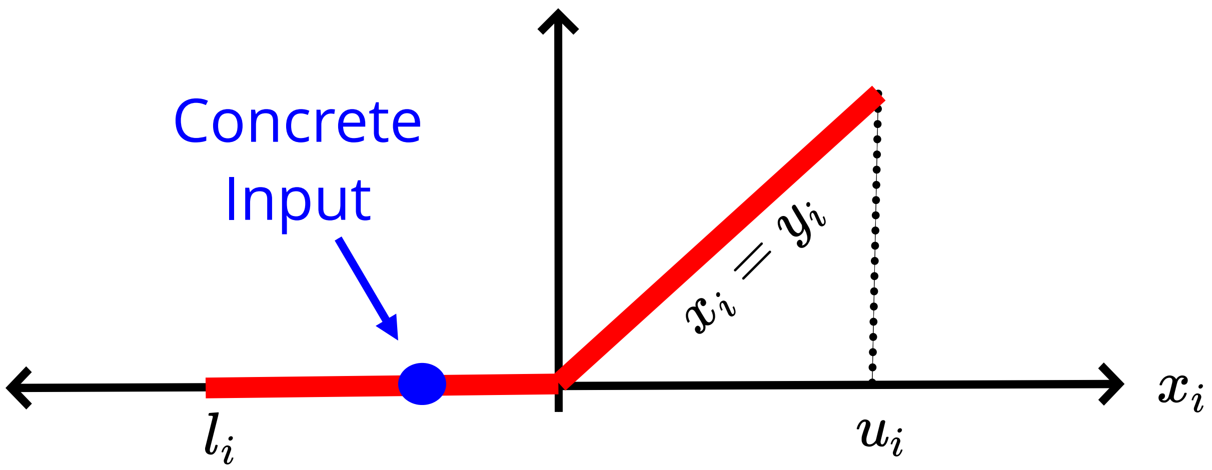


How can we avoid LP solving?

In formal verification, achieving high performance means using the appropriate level of abstraction

Idea: Use Zonotope overapproximations to prove branching is possible without LP solving

What if Zonotope doesn't help?



Actually, we don't usually need to compute l_i and u_i, just to check if $l_i < 0 < u_i$.

If l_i > 0, we're done (single LP)!

Also, if u_i < 0, we're done... how to choose direction?

Idea: use a concrete execution of the NN

Zonotope Accuracy

LP solving is still the bottleneck, how can we do better?

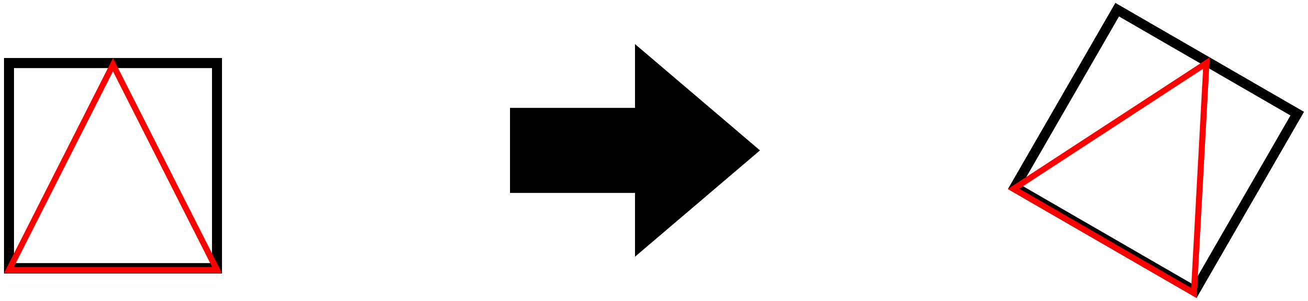
The zonotope prefilter works better if it's more accurate. How can we increase it's accuracy?

Idea: Contract the domain of the zonotope overapproximation when splitting.

Zonotope Domain Contraction

Black: Zonotope

Red: Star Set

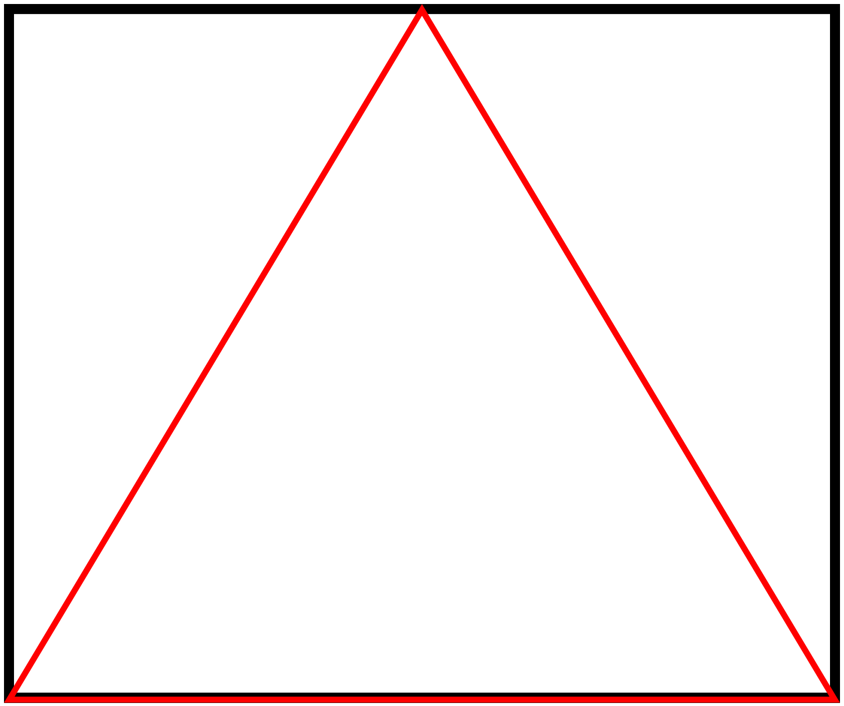


$\alpha \in \mathbb{R}^p$ $\qquad$ $x = c + V\alpha$ $\qquad$ $x \in \mathbb{R}^n$

Zonotope Domain Contraction

Black: Zonotope

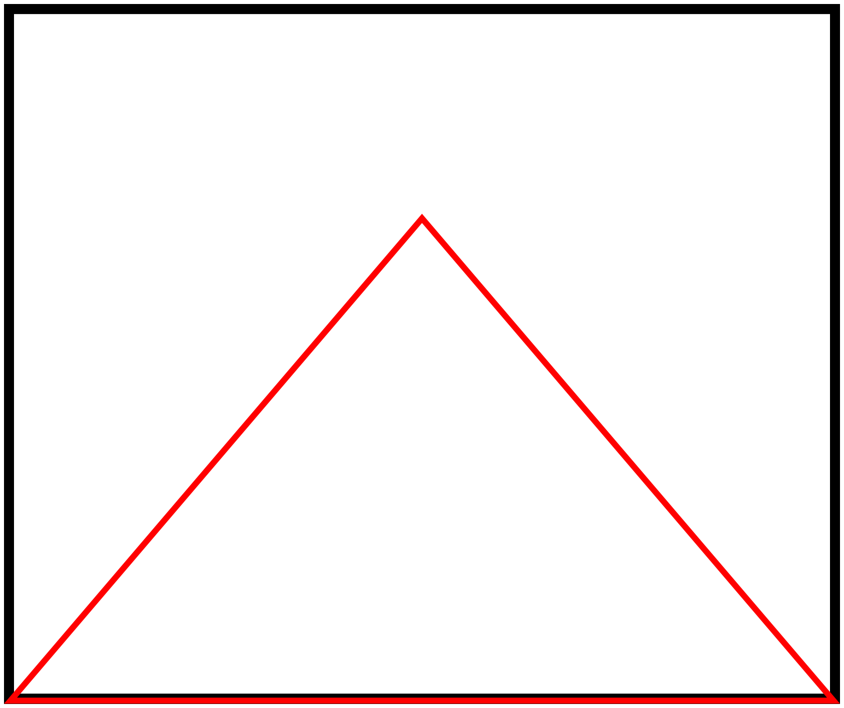
Red: Star Set



Zonotope Domain Contraction

Black: Zonotope

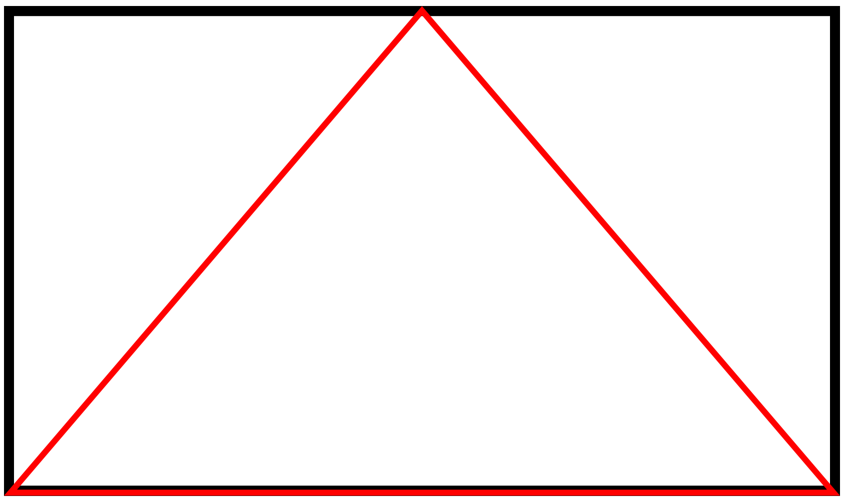
Red: Star Set



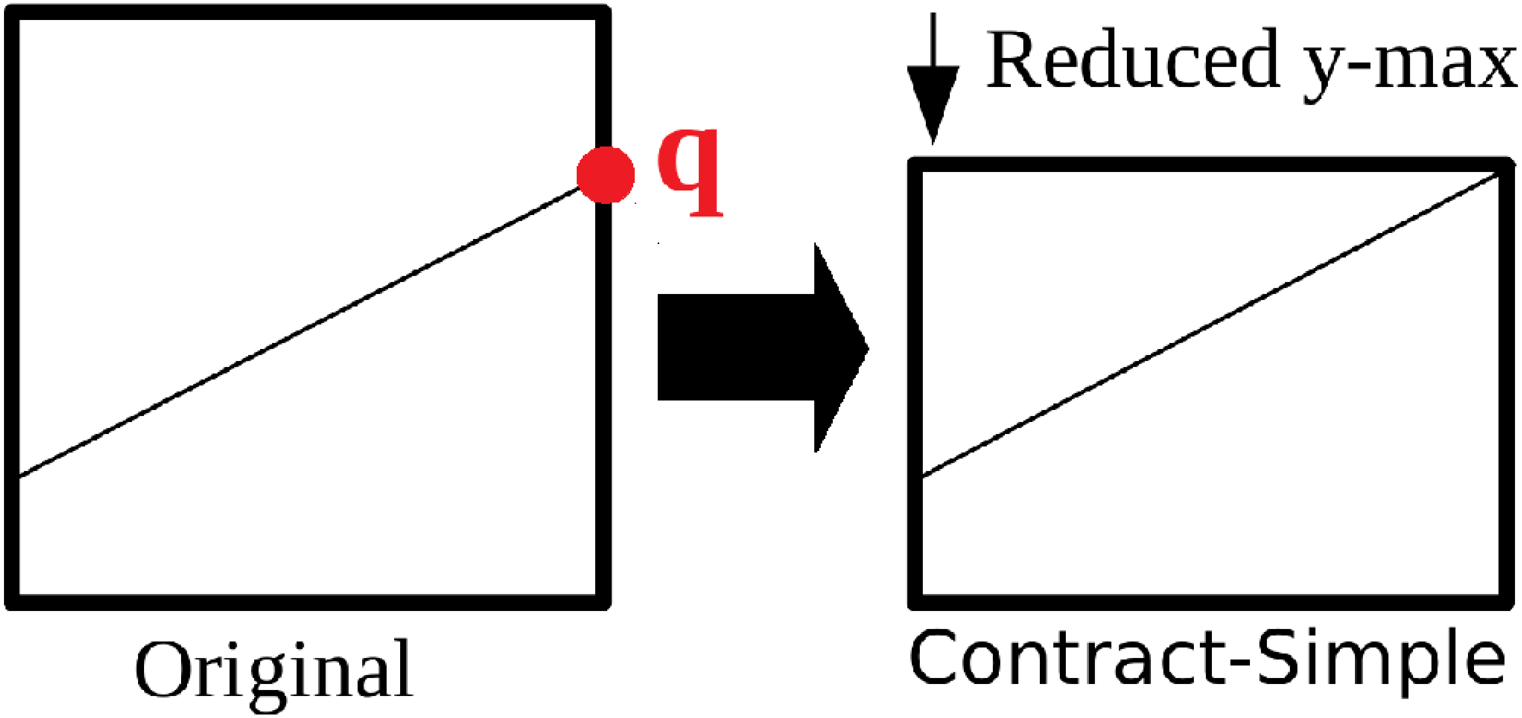
Zonotope Domain Contraction

Black: Zonotope

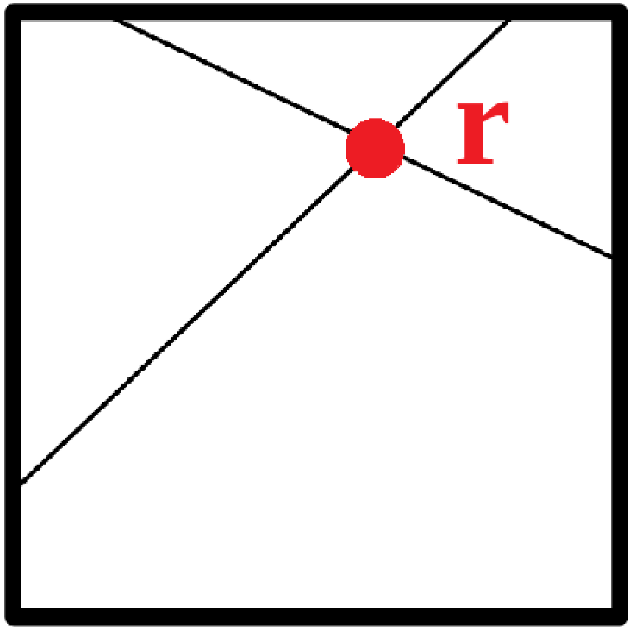
Red: Star Set



Zonotope Domain Contraction

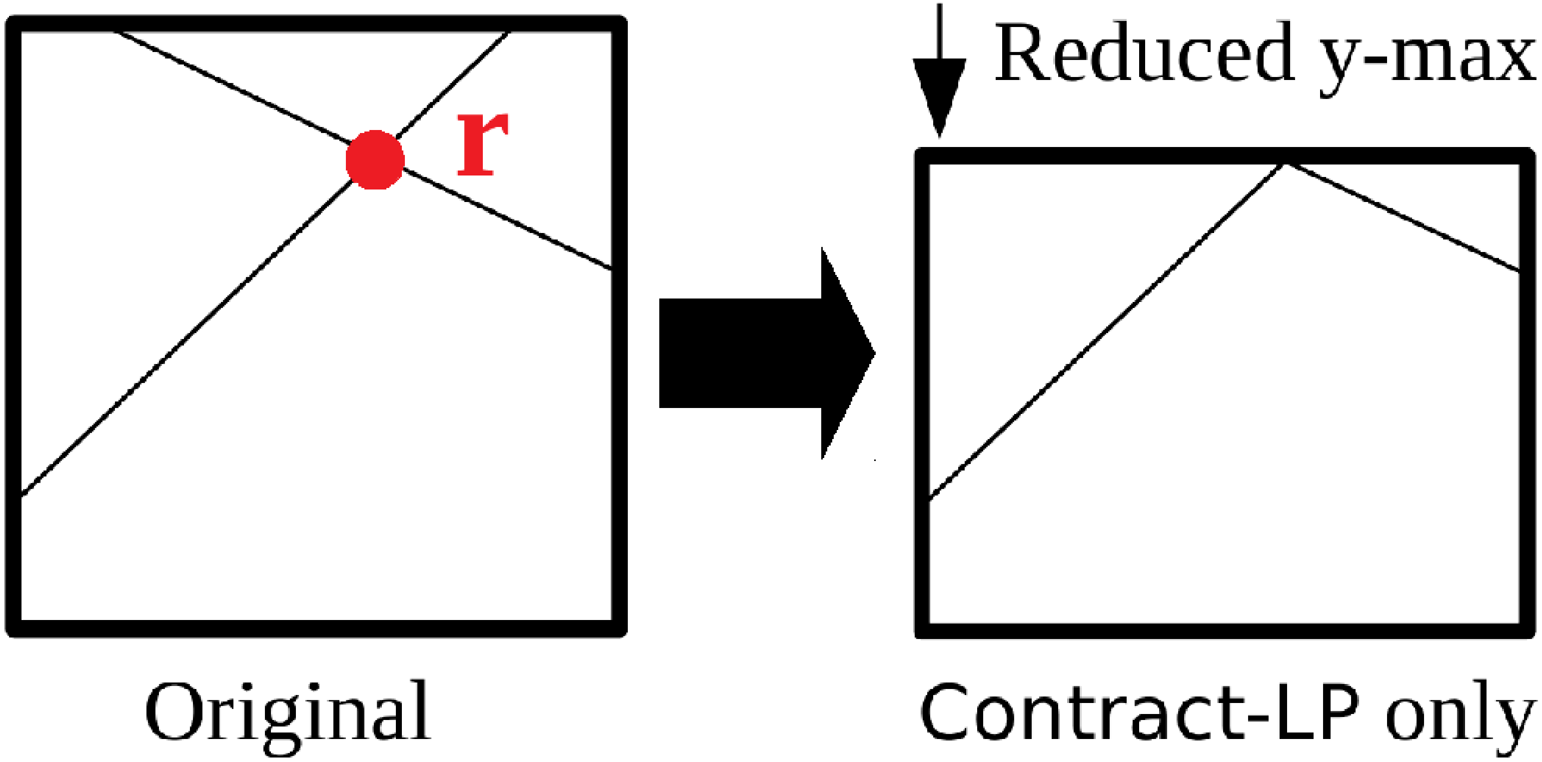


Zonotope Domain Contraction 2



Original

Zonotope Domain Contraction 2



Zonotope Domain Contraction Approaches

Q: How often do we contract?

A: Every time we take an intersection.

Algorithms:

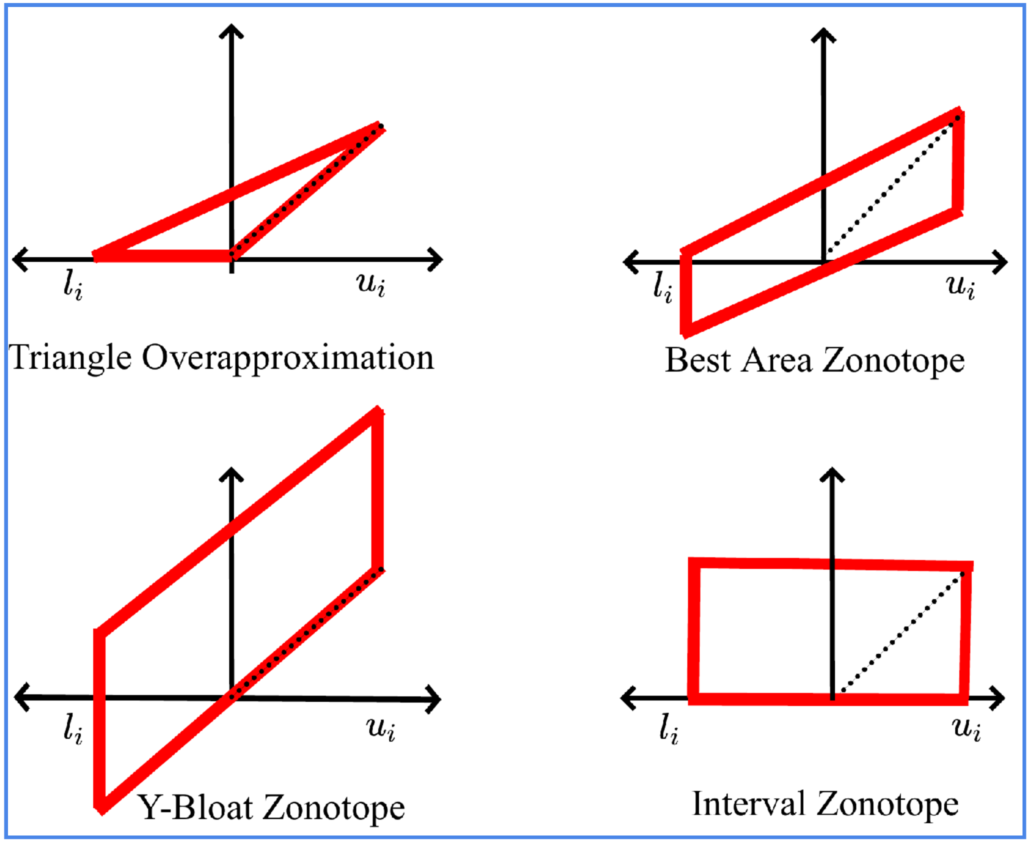
1. Single Loop Algorithm: does old box + single new constraint reduce box bounds?

2. "Old LP" - Solve one LP for each lower and upper bound

3. "Witnesses" - Store min/max points, and then when adding constraint check to see if they are removed

4. "New LP" - Optimize in multiple directions concurrently first, to check if bounds have changed.

What about Overapproximation?



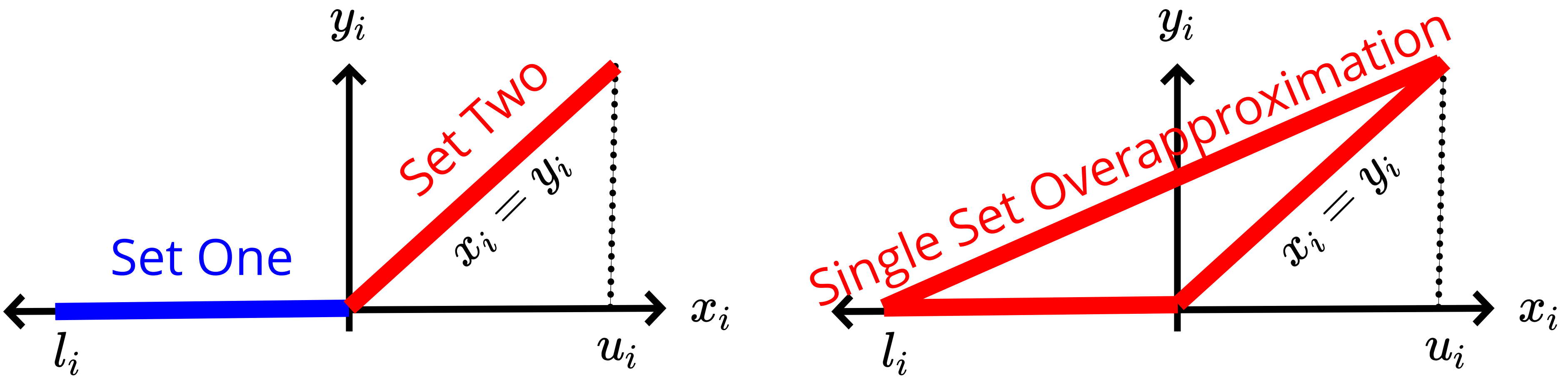
Since zonotopes are so much faster, why not consider all three zonotopes at the same time (multi-abstraction analysis)?

How about try zonotopes first, and if that fails use star sets (multi-round analysis)?

Exact vs Overapproximation

For each ReLU with $l_i < 0$ and $u_i > 0$, you can choose between splitting (exact) or single-set triangle overapproximation.

Neither is always best.



In formal verification, achieving high performance means using the appropriate level of abstraction.

Idea: Combine splitting and overapproximation. Challenge: how to choose?

CEGAR - Counter-example guided abstraction refinement

The **CEGAR** approach is to overapproximate everywhere, and if verification fails, go back and refine.

Where to refine? Simple approach: at the *first* neuron.

Subproblems are generally analyzed from **more abstract to more concrete**.

Potential downside: overapproximation analysis, which often fails, can take a long time.

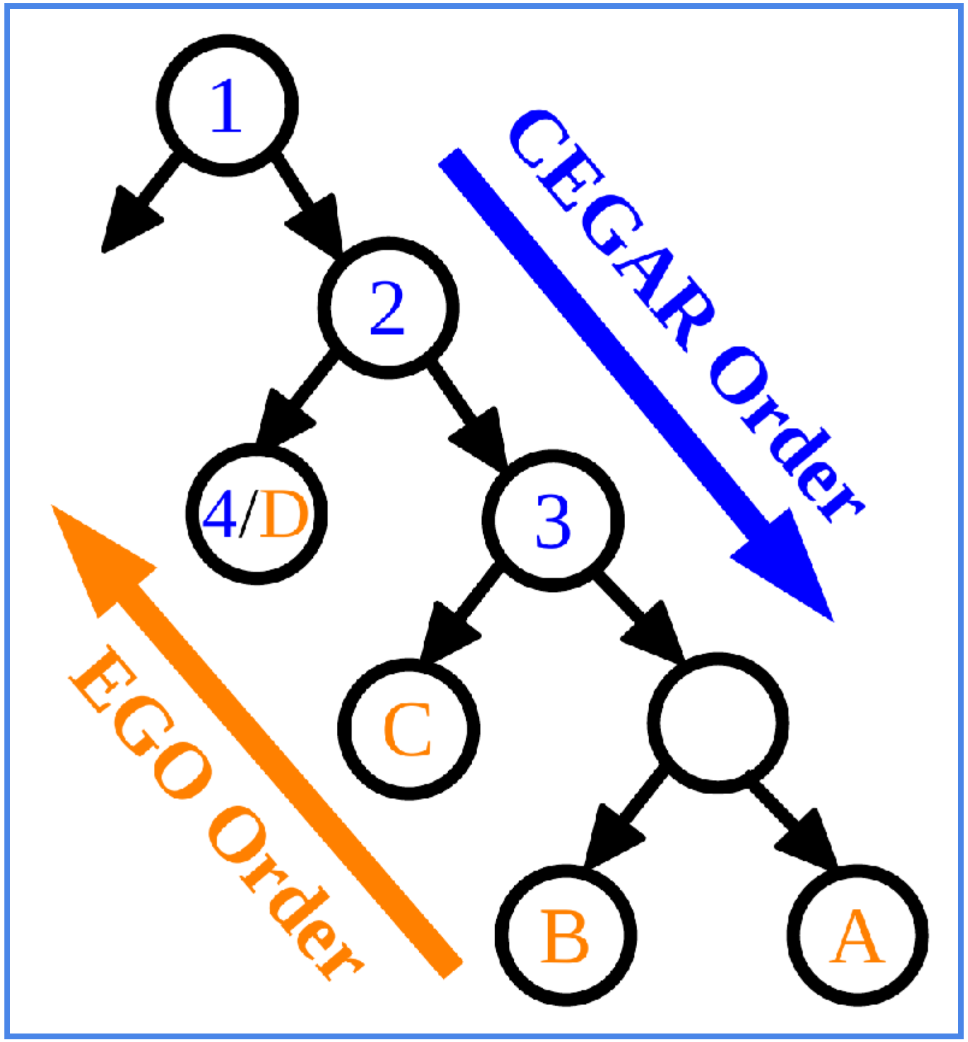
EGO - Execution-Guided Overapproximation

The **EGO** approach keep splitting until one branch of the search tree is verified.

Then, overapproximations are done from the tips of the tree, rather than the root.

Subproblems are generally analyzed from **more concrete to more abstract**.

CEGAR vs EGO Exploration Order

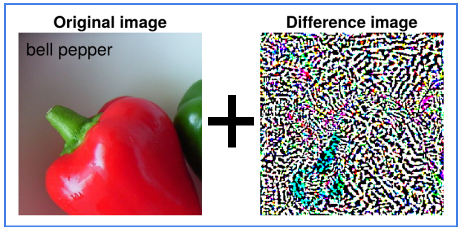


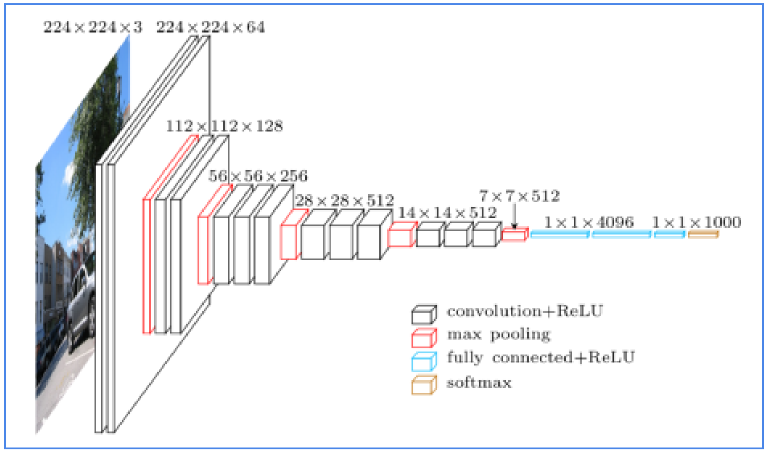
Verification Competition

Reachability for Verification

Other Recent Results

Larger Perception NNs





Bell Pepper?

VGG-16
(>10 million neurons)

See the CAV 2020 paper:
"Verification of Deep Convolutional Neural Networks Using **ImageStars**"
H.D Tran, S. Bak, W. Xiang and T. T. Johnson

Image Star

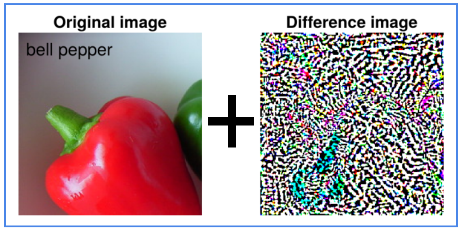
$$\Theta = c + \alpha v = \begin{bmatrix} 0 & 4 & 1 & 2 \\ 2 & 3 & 2 & 3 \\ 1 & 3 & 1 & 2 \\ 2 & 1 & 3 & 2 \end{bmatrix} + \alpha \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, P \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

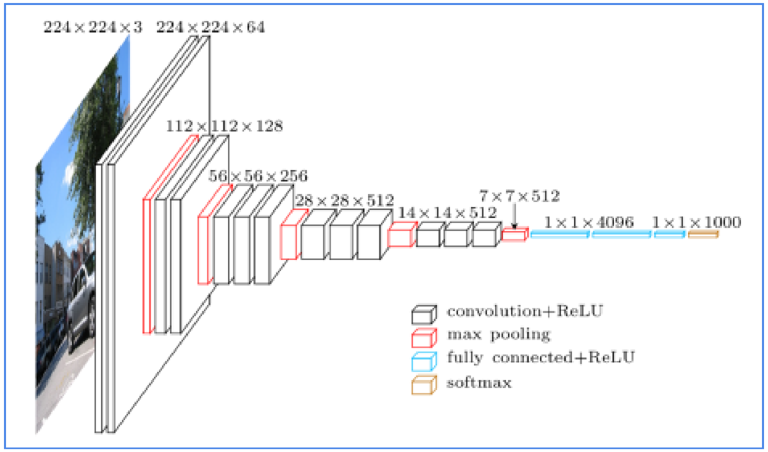
$$c \in R^{4 x 4 x 1} \qquad v \in R^{4 x 4 x 1}$$

Linear Layer examples (from onnx): 'Add', 'AveragePool', 'Constant', 'Concat', 'Conv' (diluted convolution, transpose convolution), 'Flatten', 'Gather', 'Gemm', 'MatMul', 'Mul', 'Reshape', 'Shape', 'Sub', 'Unsqueeze'

Nonlinear layers: max-pooling, atan, tanh, sigmoid, softmax (but can usually ignore)

Another Problem: Larger Perception NNs





Bell Pepper?

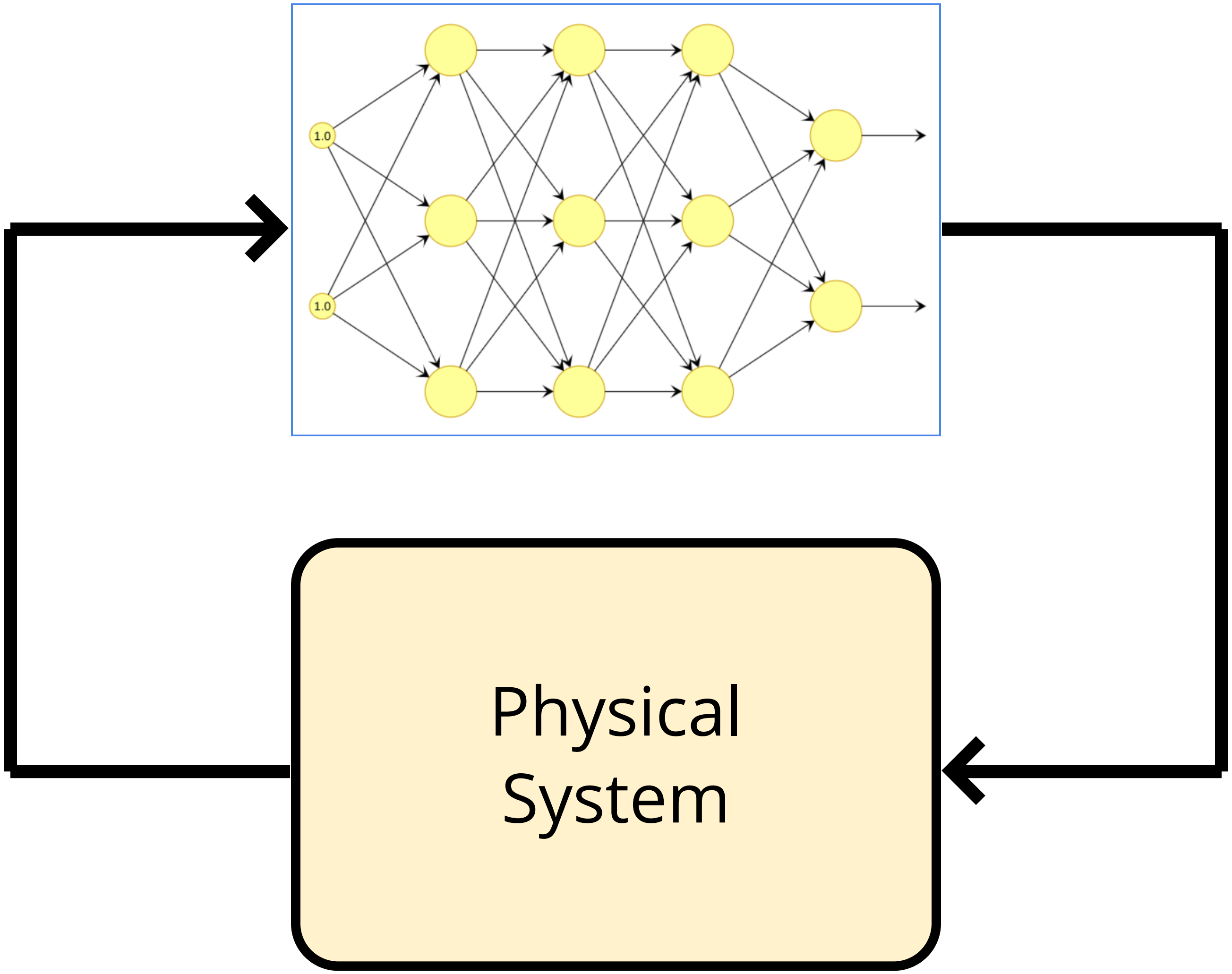
VGG-16
(>10 million neurons)

Numeric Issue: L-inf spec will require 224*224*3=~150k generators, first layer of VGG16 is 224*224*64=~3m, single-precision floats need 4 bytes per number

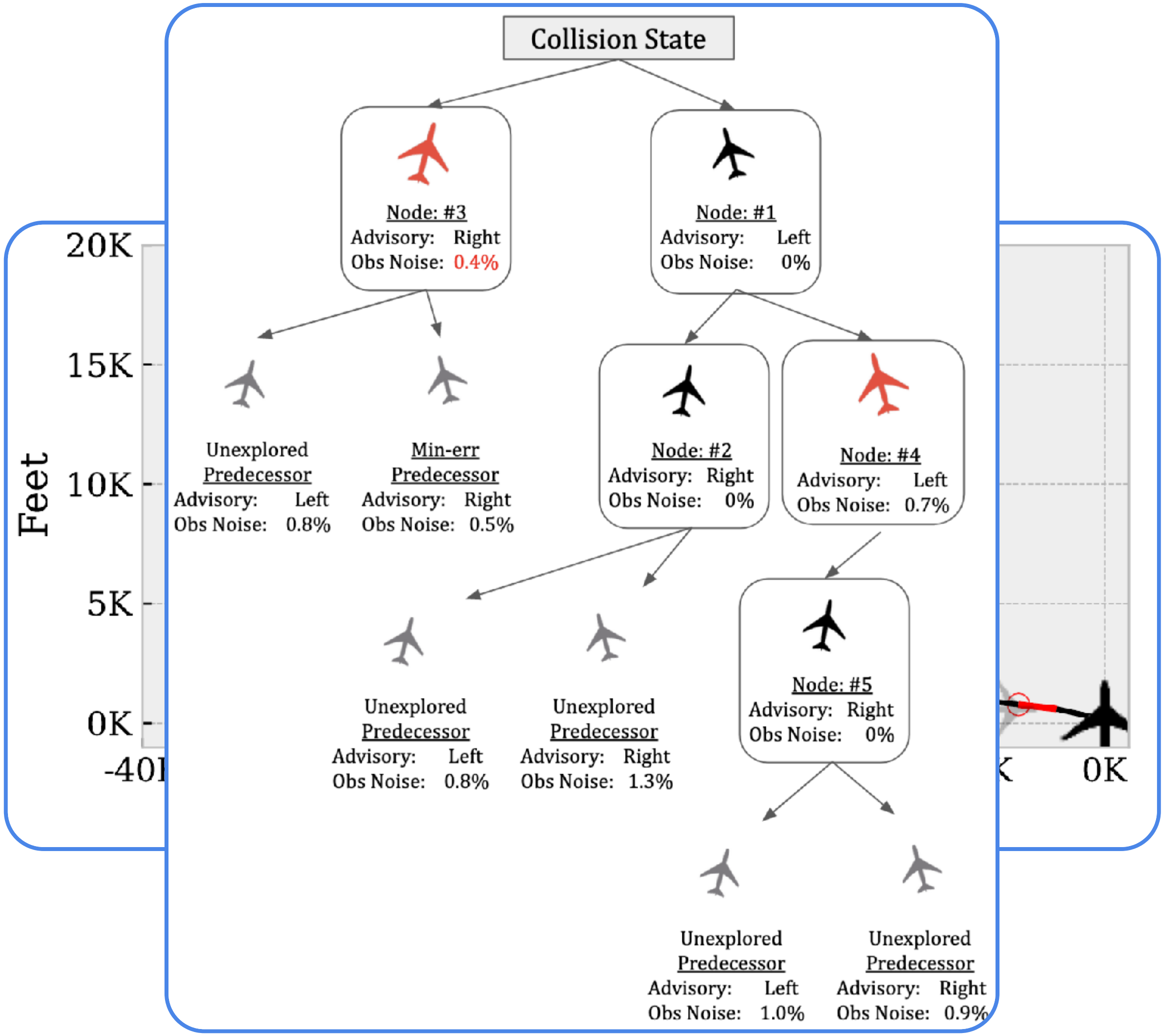
Storage space needed at first layer is ~1.9TB
and then you can start to analyze if splitting is possible

See our CAV 2020 paper:
"Verification of Deep Convolutional Neural Networks Using ImageStars"
H.D Tran, S. Bak, W. Xiang and T. T. Johnson

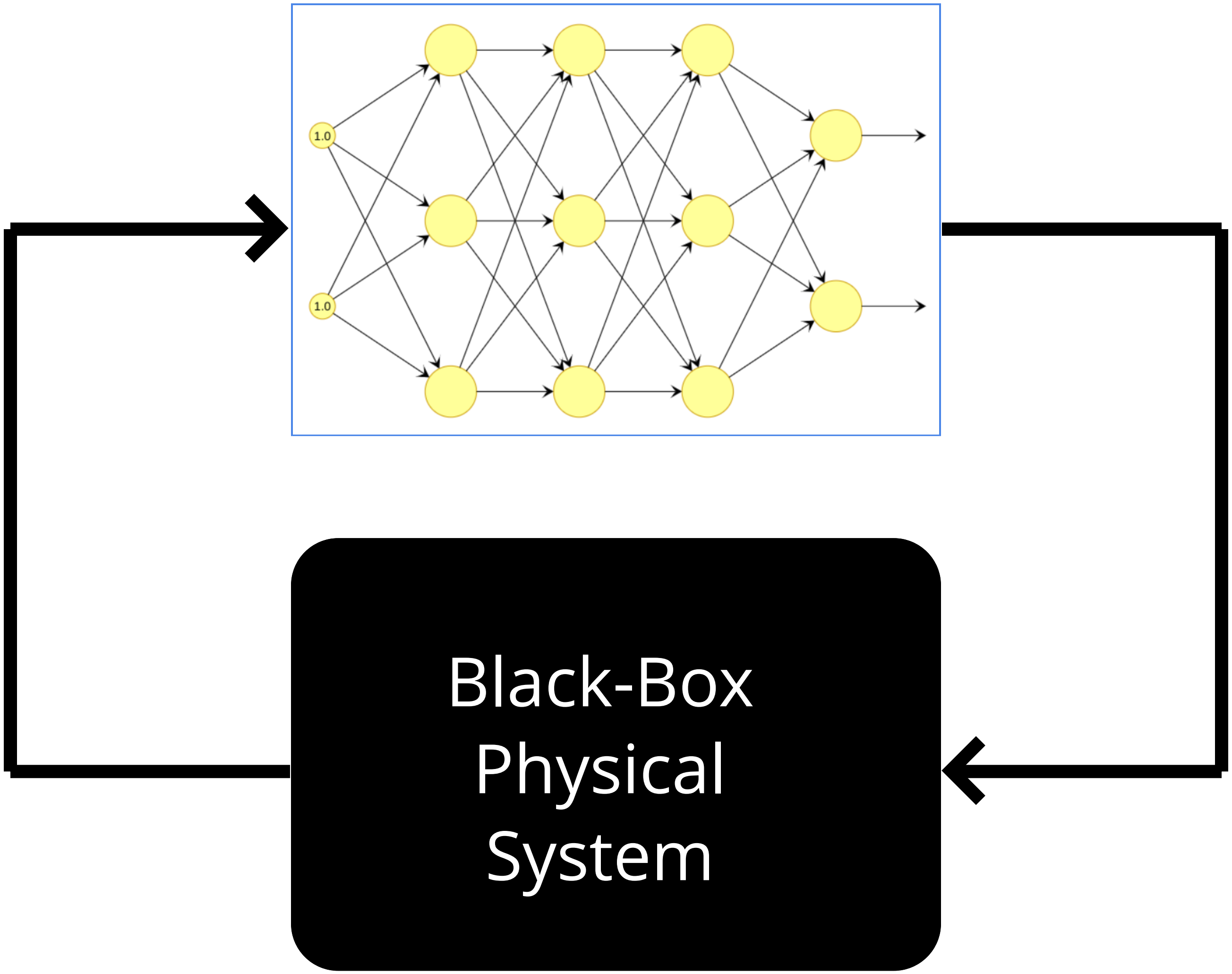
Closed-Loop Analysis



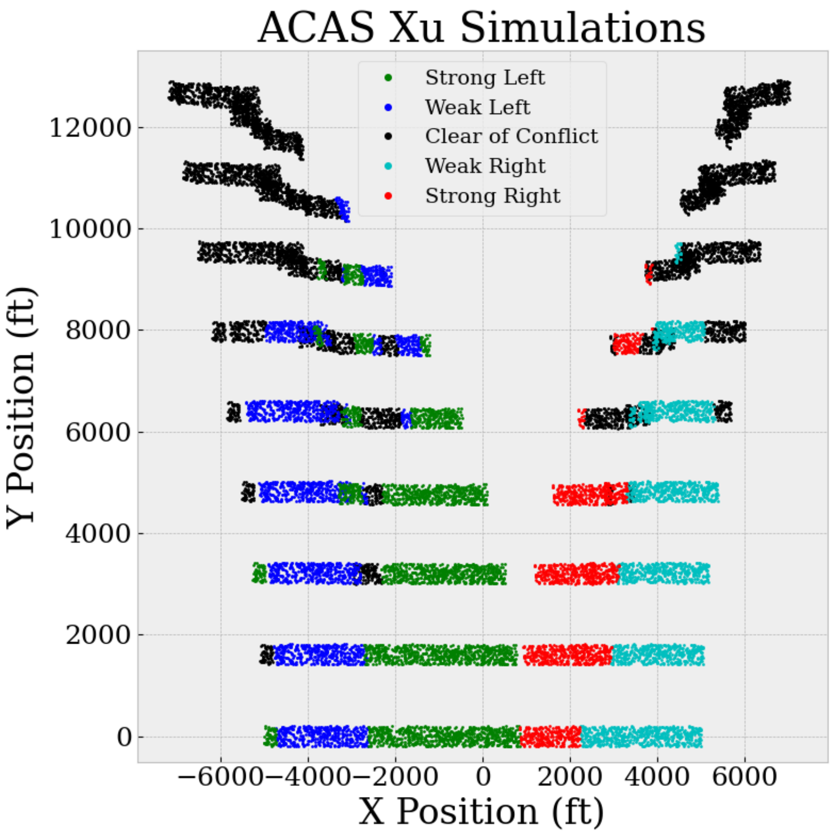
Closed-Loop Analysis with Noise



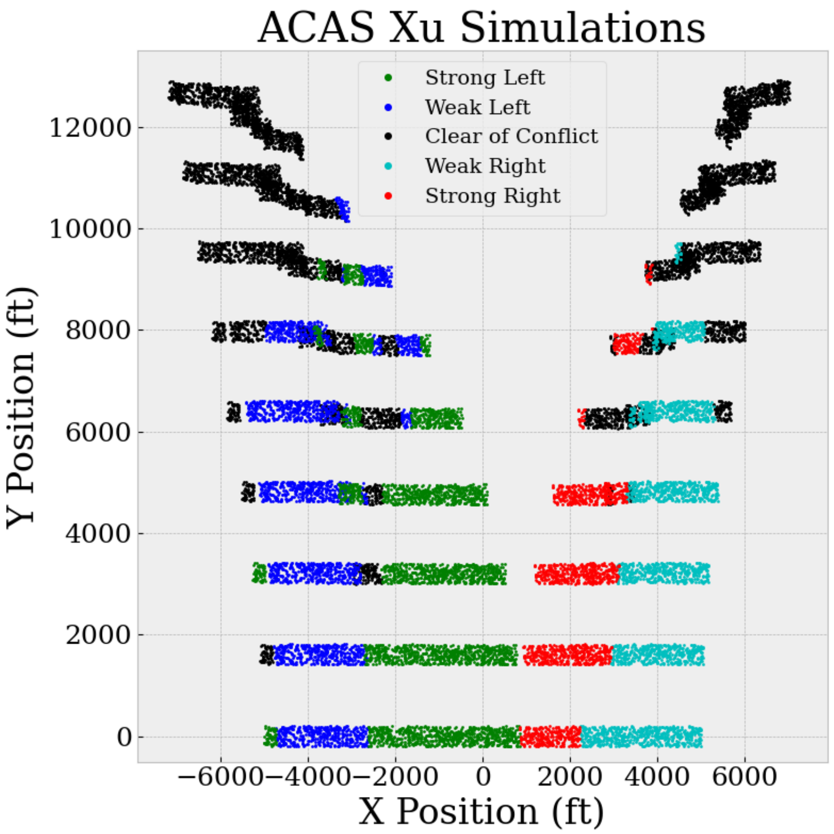
Closed-Loop Analysis

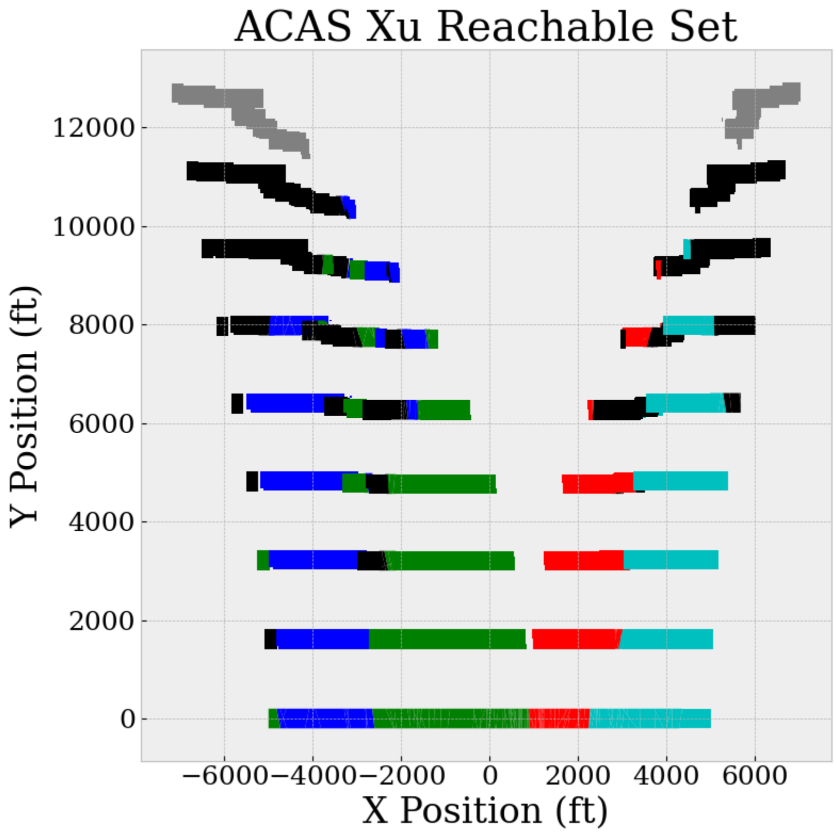


Decision Points



Decision Points

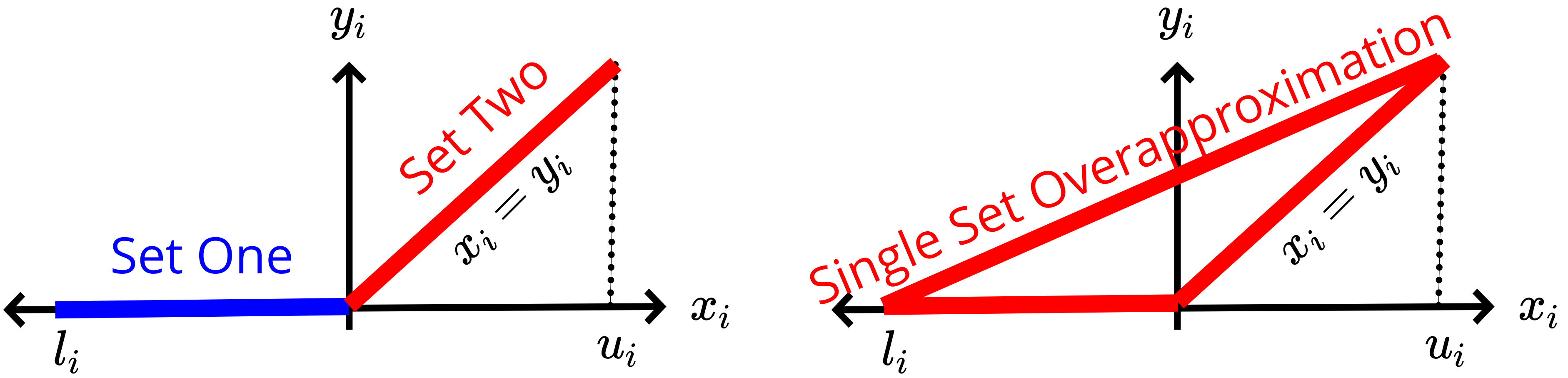




From black-box analysis with local numerical linearization

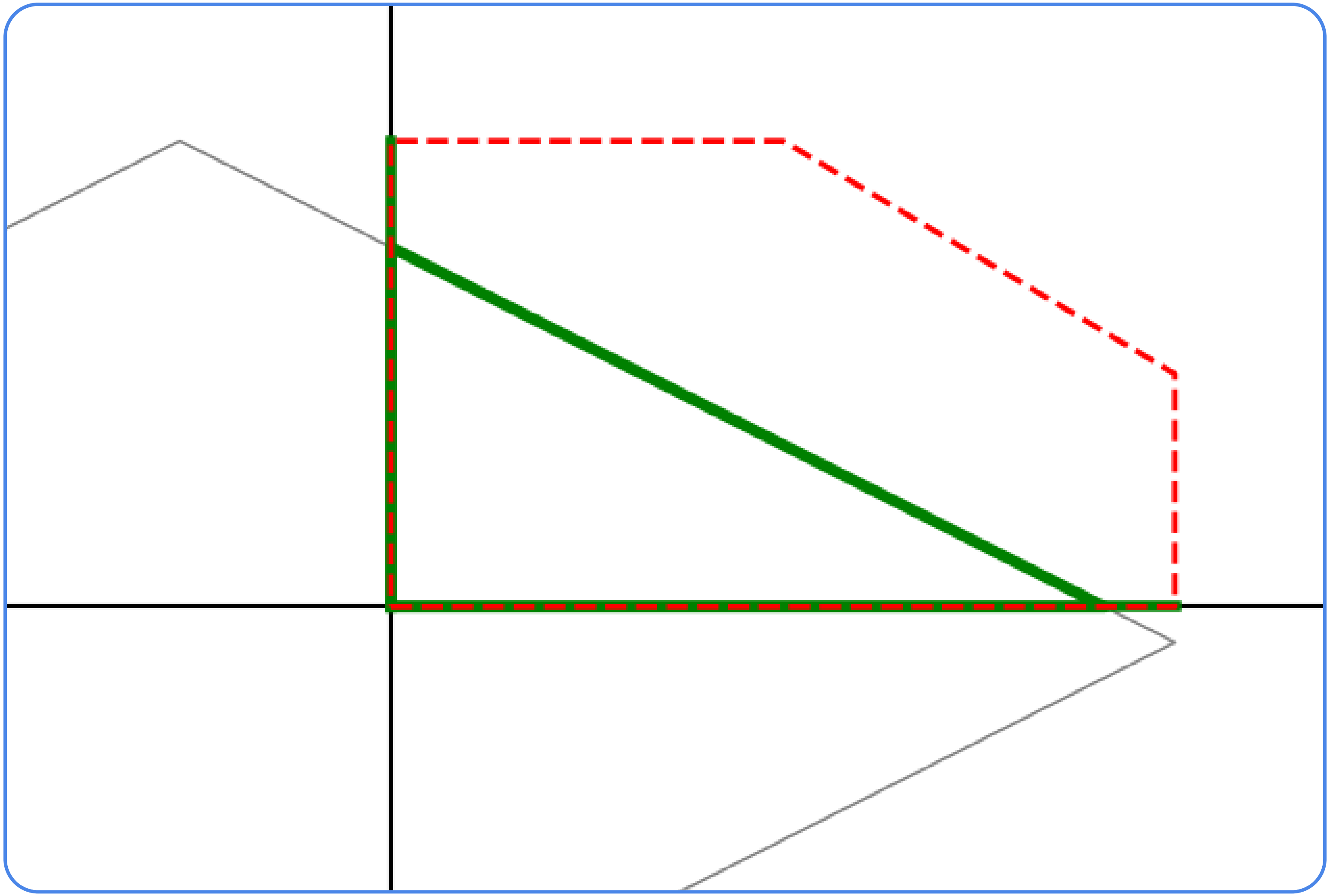
Best Convex Overapproximation?

For each ReLU with $l_i < 0$ and $u_i > 0$, you can choose between splitting (exact) or single-set triangle overapproximation.



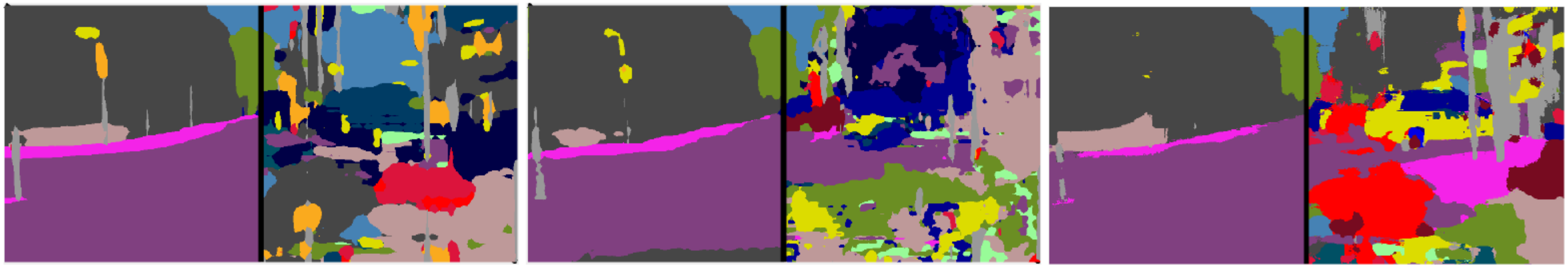
Triangle overapproximation is only tight with respect to a single neuron. With multiple neurons it can be conservative.

Best Convex Overapproximation?



Semantic Segmentation Networks

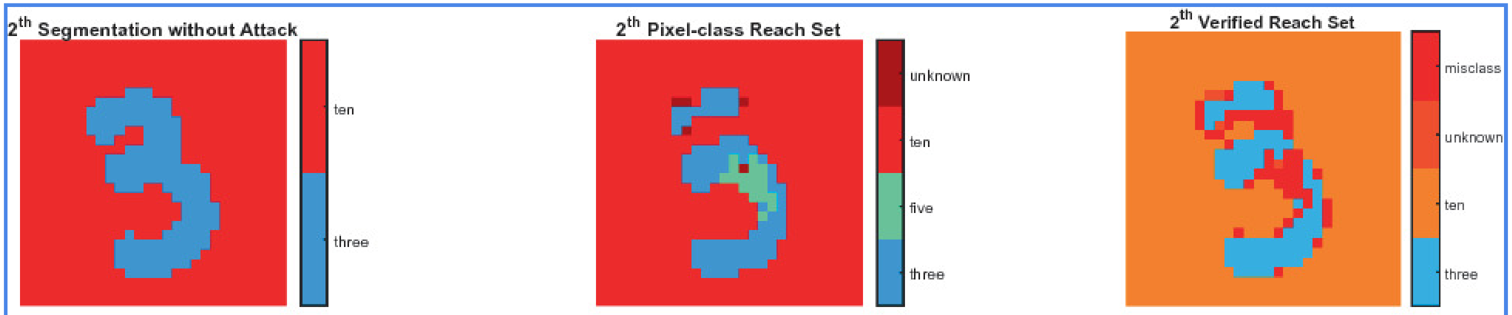
(a) Input image (perturbed half on right) (b) Ground Truth (c) PSPNet [71]



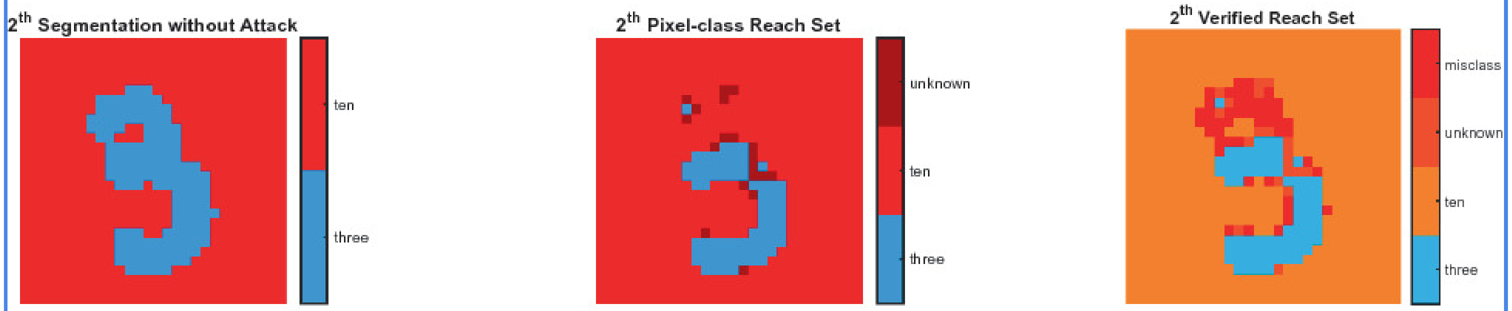
(d) DilatedNet [69] (e) ICNet [70] (f) CRF-RNN [72]

https://www.robots.ox.ac.uk/~aarnab/adversarial_robustness.html

Verification of Semantic Segmentation Networks



(a) $R_f(N_1), N_{unknown} = 6$ ($N_{max} = 50, \Delta_\epsilon = 0.003$).



(b) $R_f(N_2), N_{unknown} = 19$ ($N_{max} = 50, \Delta_\epsilon = 0.003$).

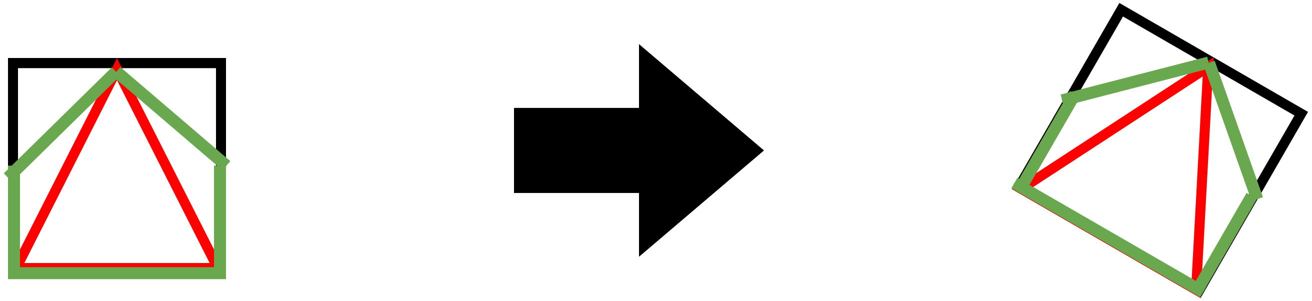
Tran, Hoang-Dung, et al. "Robustness verification of semantic segmentation neural networks using relaxed reachability." International Conference on Computer Aided Verification, 2021.

Octatopes

$$Z = \{x \in \mathbb{R}^n \mid x = c + V\alpha, \alpha \in [-1, 1]^p\}$$
$$S = \{x \in \mathbb{R}^n \mid x = c + V\alpha, \alpha \in Cx \leq d\}$$
$$O = \{x \in \mathbb{R}^n \mid x = c + V\alpha, \alpha \text{ is UTVPI}\}$$

A unit two variable per inequality (UTVPI) constraint is of the form $a\alpha_i + b\alpha_j \leq d$ where the coefficients $a, b \in \{-1, 0, 1\}$.

An octatope is a set of points defined with an affine transformation from a p-dim **octagon** to an n-dim space



$\alpha \in \mathbb{R}^p$ $\qquad x = c + V\alpha$ $\qquad x \in \mathbb{R}^n$

Summary

- There are still lots of research problems in NN verification.
- I've only focused on reachability approaches.
- See the other VNNCOMP tools in the report for lots of other great ideas and up-to-date related work.