ECE/CS 584: Verification of Embedded and Cyber-physical Systems

Lecture 7: Introduction to Machine Learning and Its Verification Problems

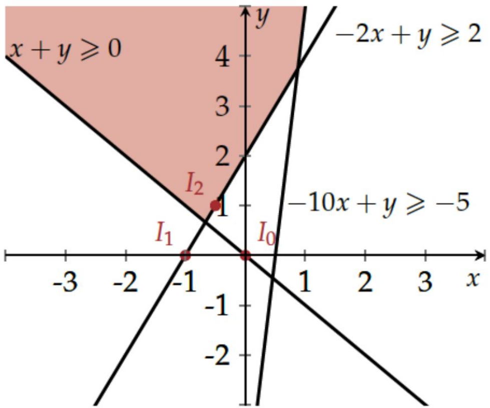
Prof. Huan Zhang

huan@huan-zhang.com

Review: Linear Real Arithmetic (LRA) Theory

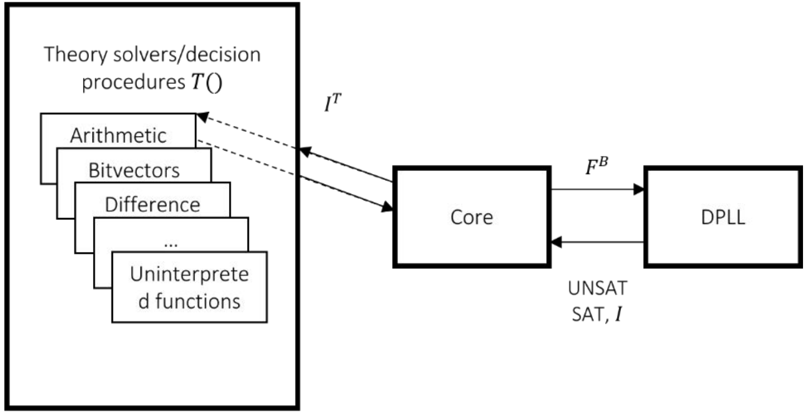
$(x+y \geq 0) \wedge (-2x+y \geq 2) \wedge (-10x+y \geq -5)$

Decision problem can be solved using Simplex algorithm.



Review: DPLL(T) to solve SMT problems

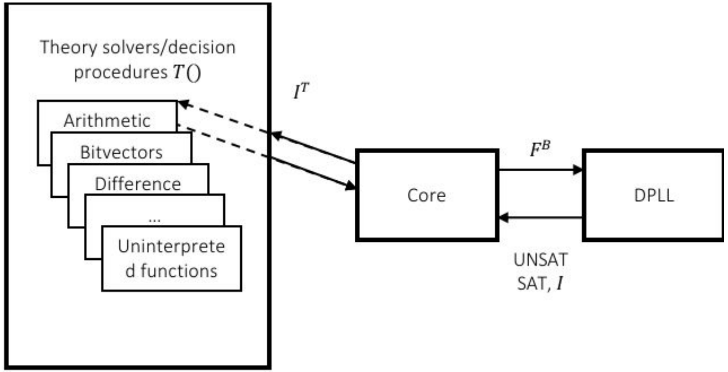
Input: A formula F in CNF form over theory T
Output: $I \vDash F$ or UNSAT
Let F^B be the abstraction of F
while true **do**
  if $\mathrm{DPLL}(F^B)$ is unsat then **return** UNSAT
else
  Let I be the model returned by $DPLL$
  Assume I is represented as a formula
  if $T(I^T)$ is sat **then return** SAT and the model returned by $T()$
  else $F^B := F^B \wedge \neg I$



- $\phi \equiv g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$

  $1 \quad \bar{2} \quad 3 \quad \bar{4}$

- abstract $\phi \equiv x_1 \wedge (\neg x_2 \vee x_3) \wedge \neg x_4$
- send $\phi^B \equiv \{1, \bar{2} \vee 3, \bar{4}\}$ to DPLL
- DPLL returns SAT with model I:$\{1, \bar{2}, \bar{4}\}$
- UF solver concretizes $I^{UF} \equiv g(a) = c, f(g(a)) \neq f(c), c \neq d$
- UF checks I^{UF} as UNSAT
- send $\phi^B \wedge \neg I$: $\{1, \bar{2} \vee 3, \bar{4}, \bar{1} \vee 2 \vee 4\}$ to DPLL; this is a new fact learned by DPLL
- DPLL returns model I': $\{1, 2, 3, \bar{4}\}$
- UF solver concretizes I'^{UF} and finds this to be UNSAT
- send $\phi^B \wedge \neg I \wedge \neg I'$: $\{1, \bar{2} \vee 3, \bar{4}, \bar{1} \vee 2 \vee 4, \bar{1} \vee \bar{2} \vee \bar{3} \vee 4\}$ to DPLL; another fact
- returns UNSAT



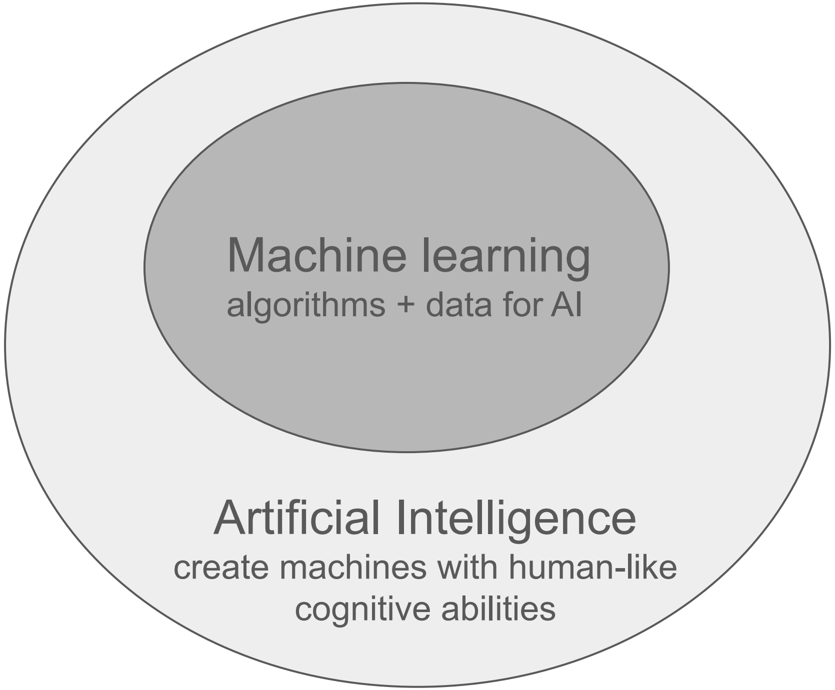
What is machine learning?

“the capacity of computers to **learn** and adapt **without following explicit instructions**, by using **algorithms** and **statistical** models to analyse and infer from patterns in **data**”

-- Oxford English Dictionary

“a field of study in artificial intelligence concerned with the development and study of **statistical algorithms** that can **learn** from **data** and generalize to unseen data, and thus perform tasks **without explicit instructions**.”

--Wikipedia



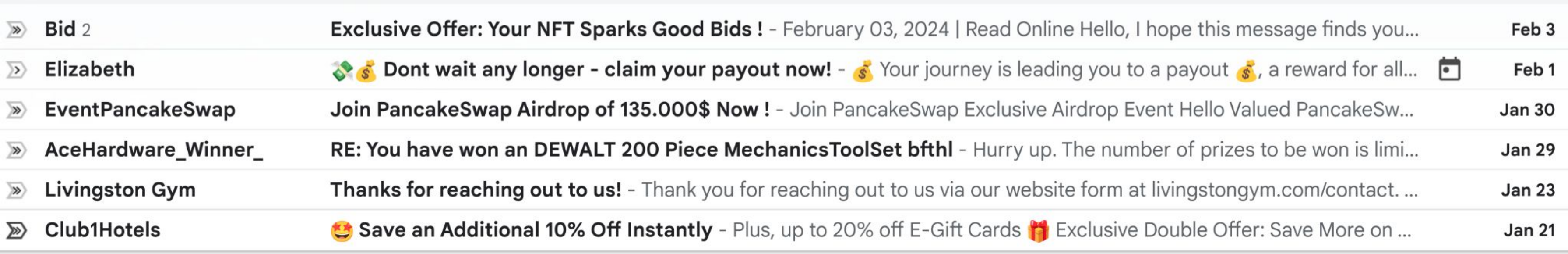
Example: spam email classification

These are what show up in my Gmail “spam” folder:

Bid 2	**Exclusive Offer: Your NFT Sparks Good Bids !** - February 03, 2024 | Read Online Hello, I hope this message finds you...	Feb 3
Elizabeth	💸💰 **Dont wait any longer - claim your payout now!** - 💰 Your journey is leading you to a payout 💰, a reward for all...	Feb 1
EventPancakeSwap	**Join PancakeSwap Airdrop of 135.000$ Now !** - Join PancakeSwap Exclusive Airdrop Event Hello Valued PancakeSw...	Jan 30
AceHardware_Winner_	**RE: You have won an DEWALT 200 Piece MechanicsToolSet bfthl** - Hurry up. The number of prizes to be won is limi...	Jan 29
Livingston Gym	**Thanks for reaching out to us!** - Thank you for reaching out to us via our website form at livingstongym.com/contact. ...	Jan 23
Club1Hotels	🤩 **Save an Additional 10% Off Instantly** - Plus, up to 20% off E-Gift Cards 🎁 Exclusive Double Offer: Save More on ...	Jan 21

TODO: write a program to classify whether an email is a spam email?

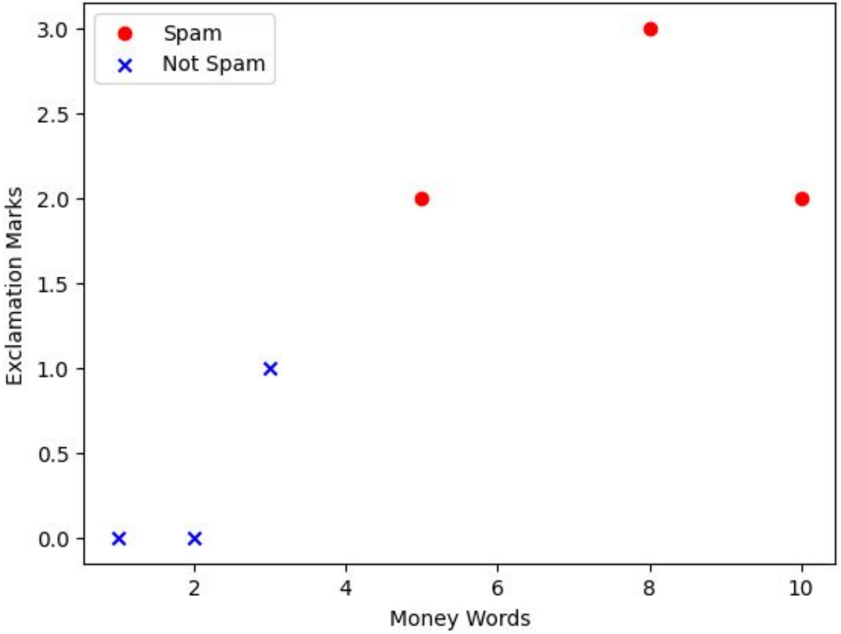
Step 1: collect data



Define some “Features”:
Count of “money words” (“payout”, “$”, “dollar”, “prizes”, “NFT”, …)
Count of exclamation marks

Step 1: collect data

Money Words (x_1)	Exclamation Marks (x_2)	Spam (y)
10	2	1
2	0	0
5	2	1
3	1	0
8	3	1
1	0	0



Non-machine-learning approach: write a program explicitly

Define some “Features”:
Count of “money words” (“payout”, “$”, “dollar”, “prizes”, “NFT”, …)
Count of exclamation marks

```
def is_spam(count_money_word, count_exclamation_mark):
    if a * count_money_word + b *count_exclamation_mark >= threshold:
        return True
    else:
        return False
```

A human programmer chooses `a`, `b`, `threshold`

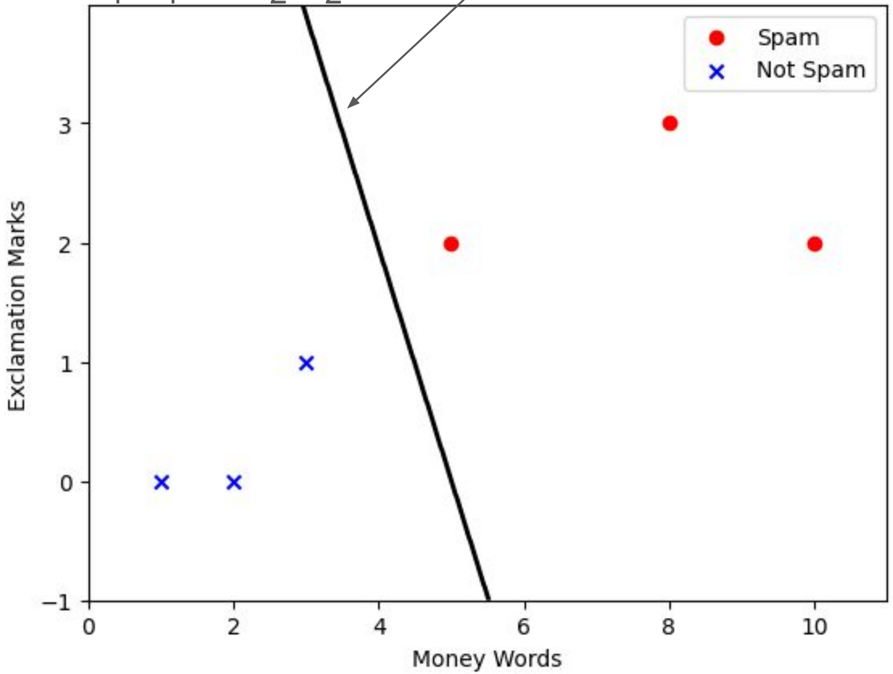
Step 2: learning from data (model training)

Instead of choosing these parameters, we learn these from data

Algorithms to find this classifier (not discussed in this class):

- Logistic regression
- Support vector machines

$w_1 x_1 + w_2 x_2 = c$ (decision boundary)



Step 3: prediction

Given **model weights** $w \in R^N$

Given **features** of an input $x \in R^N$

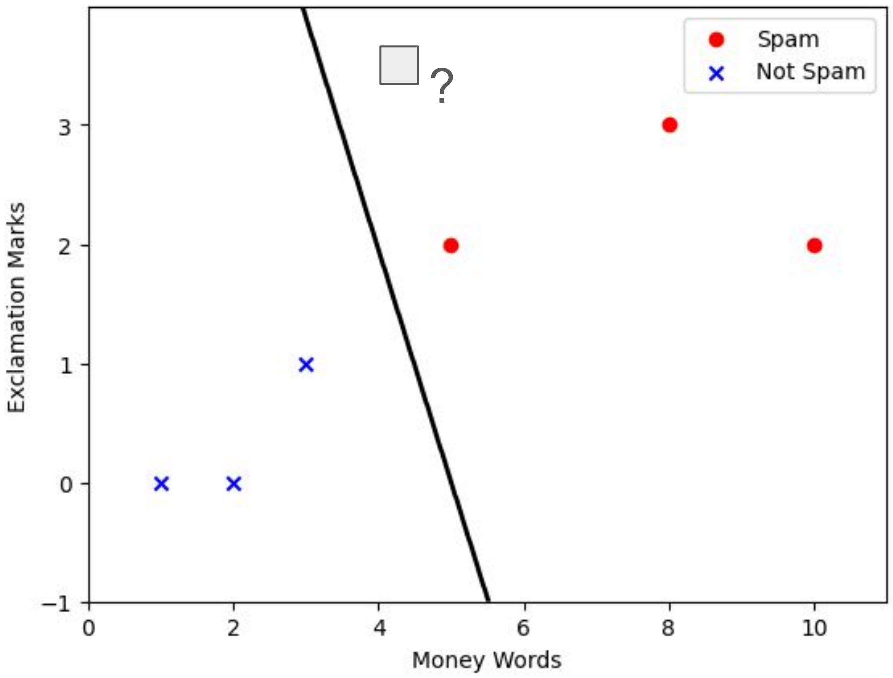
If $w^T x > c$: predict positive class (e.g., spam)

If $w^T x < c$: negative class (e.g., not spam)

Note that by simple transformations on w and x, we just need to check $w'^T x' > 0$ or $w'^T x' < 0$:

$w' = [w_1, \ldots, w_N, -c]$

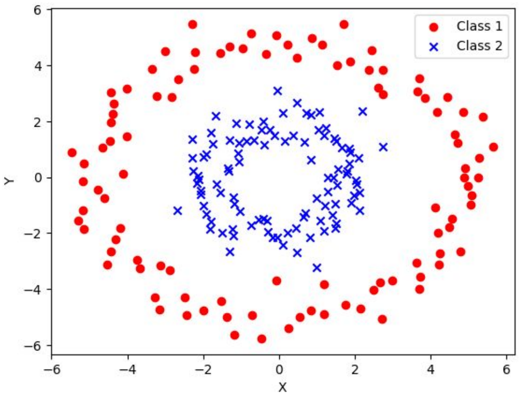
$x' = [x_1, \ldots, x_N, 1]$



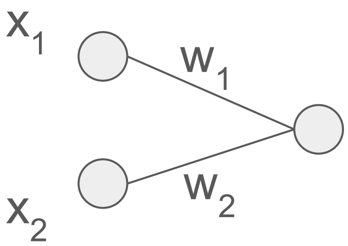
When linear function does not work well

To solve most practical classification problems, non-linear classifiers are needed. Many different approaches:

- Kernel method
- **Neural networks**
- Tree ensembles
- …

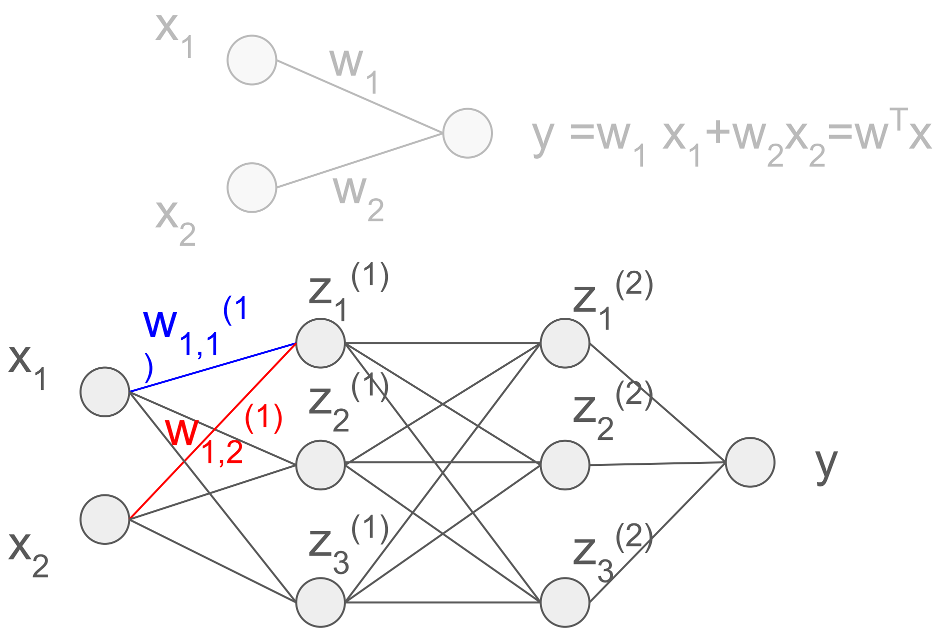


Neural Networks: let's just stack linear functions multiple times?



$y = w_1 x_1 + w_2 x_2 = w^T x$

Neural Networks: let’s just stack linear functions multiple times?

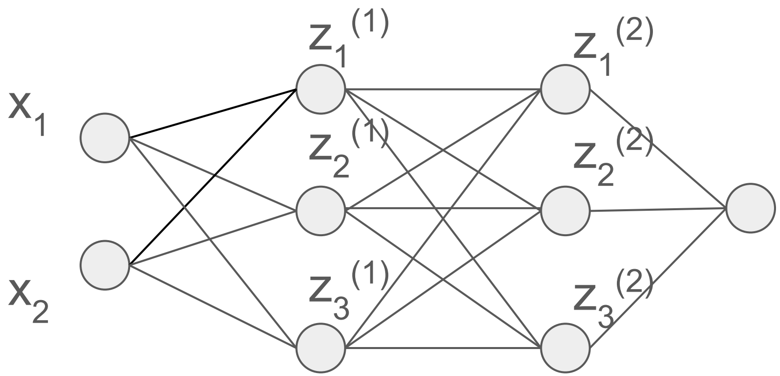


$$z_1^{(1)} = w_{1,1}^{(1)} x_1 + w_{1,2}^{(1)} x_2$$

In general we write in matrix form: $z^{(1)} = W^{(1)} x$, $W^{(1)}$ is a 3x2 matrix above

Neural Networks: let's just stack linear functions multiple times?

$z^{(1)}=W^{(1)}x+b^{(1)}$ $z^{(2)}=W^{(2)}z^{(1)}+b^{(2)}$ A bias term can be added

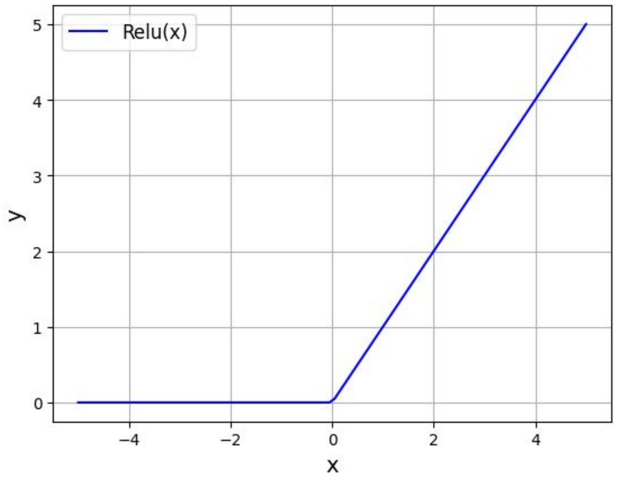


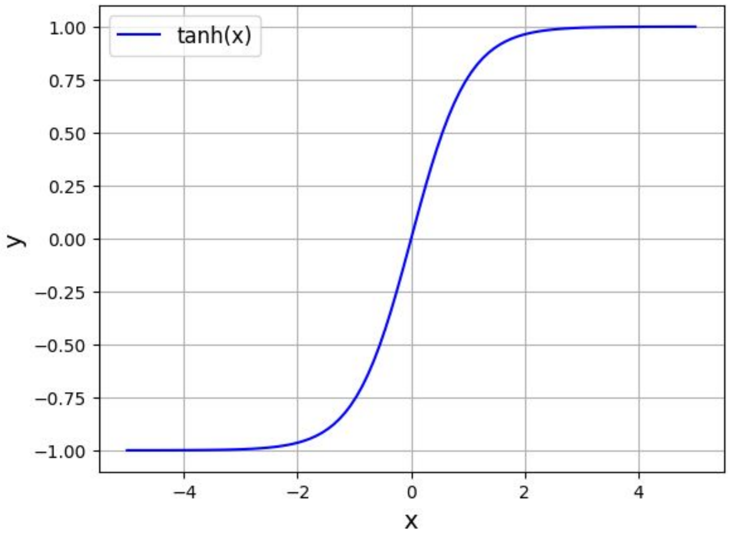
$y=w^{(3)T}z^{(2)}+b^{(3)}$

$y=w^{(3)T}z^{(2)}=w^{(3)T}W^{(2)}z^{(1)}=w^{(3)T}W^{(2)}W^{(1)}x$ still a linear function of x!

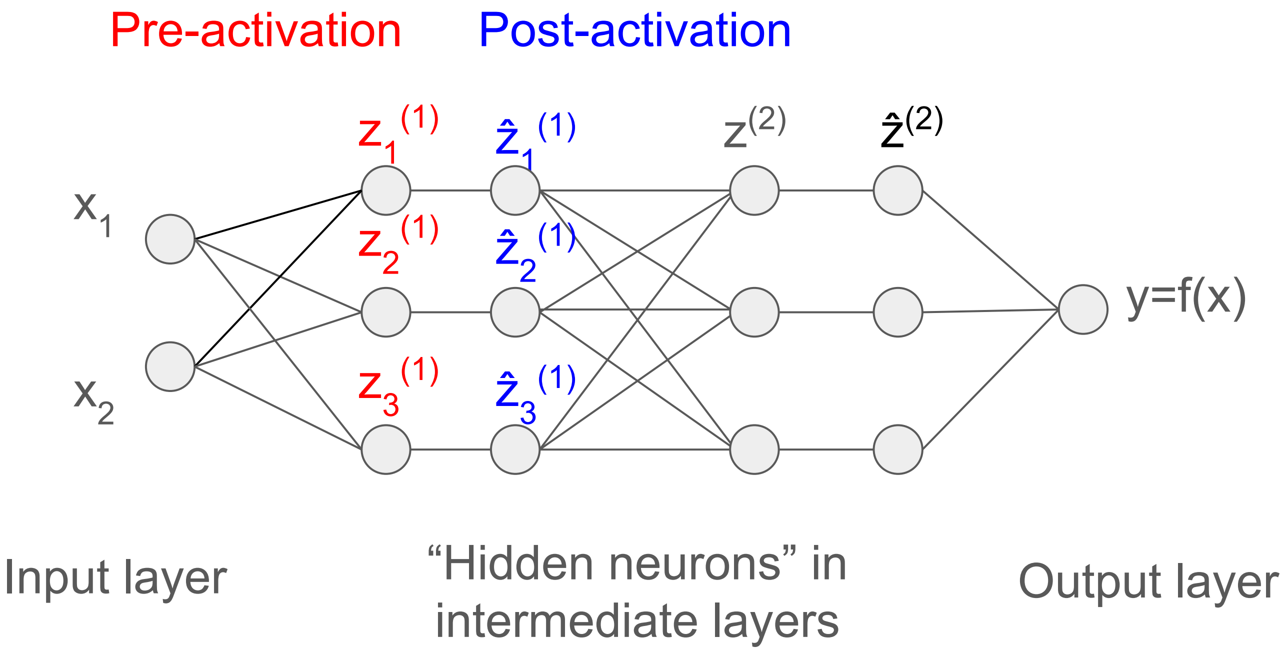
Must introduce nonlinear functions (“activation” functions)

ReLU: rectified linear unit
ReLU(x) := max(0, x)





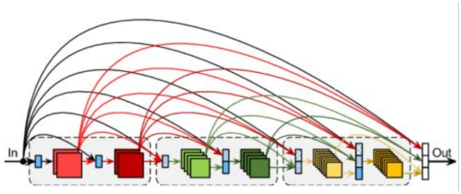
Neural Networks: linear + non-linear layers (multi-layer perceptron)

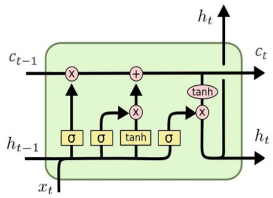


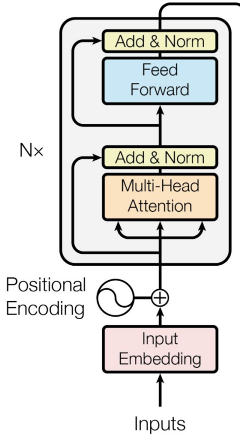
$\hat{z}_j^{(i)}=\sigma(z_j^{(i)})$

Neural networks are "**Universal approximators**"

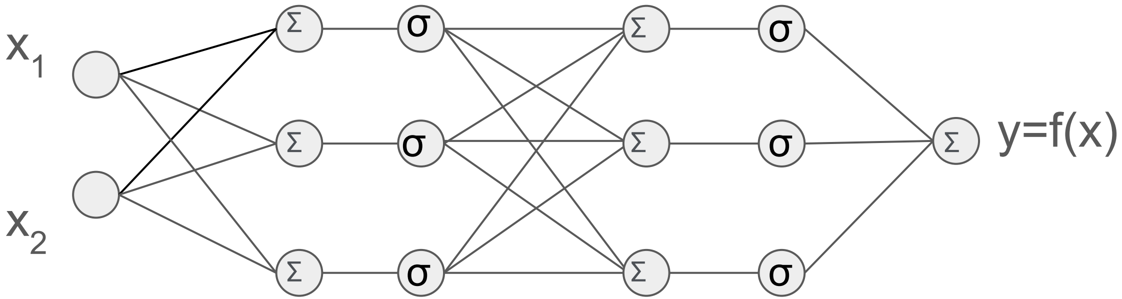
Many other neural network architectures available



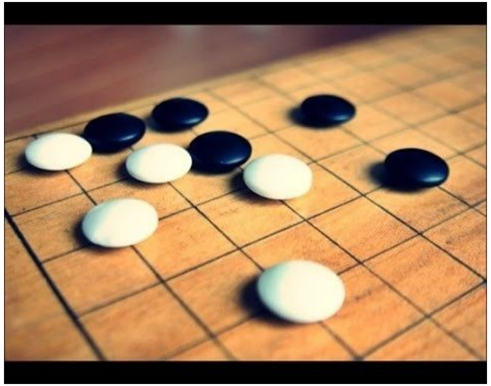




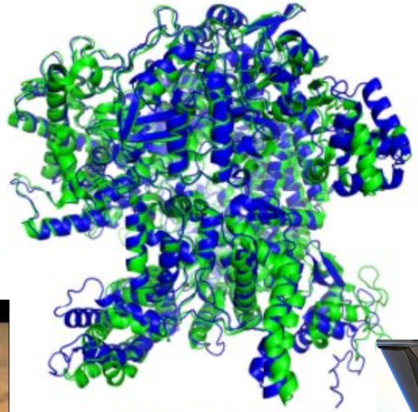
In general, neural networks can be presented as a "computation graph"



Many other neural network architectures available



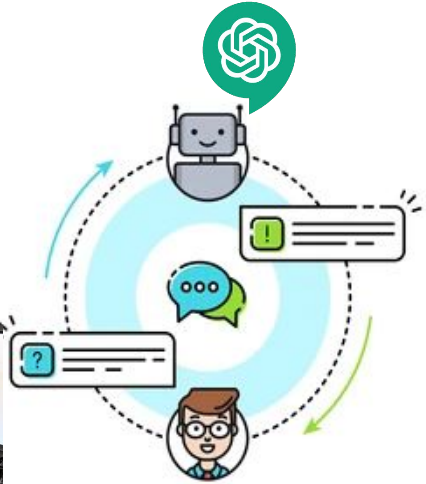
AlphaGo (2020)



AlphaFold (2021)

"A robot manipulating an aircraft"

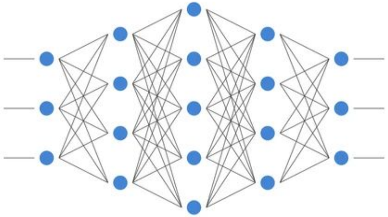
Stable Diffusion (2022)



ChatGPT (2023)

Neural networks are not safe enough for mission-critical tasks

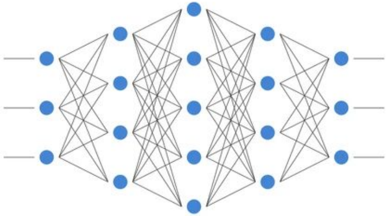
Eykholt et al. 2018



"Speed limit 45"

Deep Neural Networks

Sharif et al. 2018



"Brad Pitt"

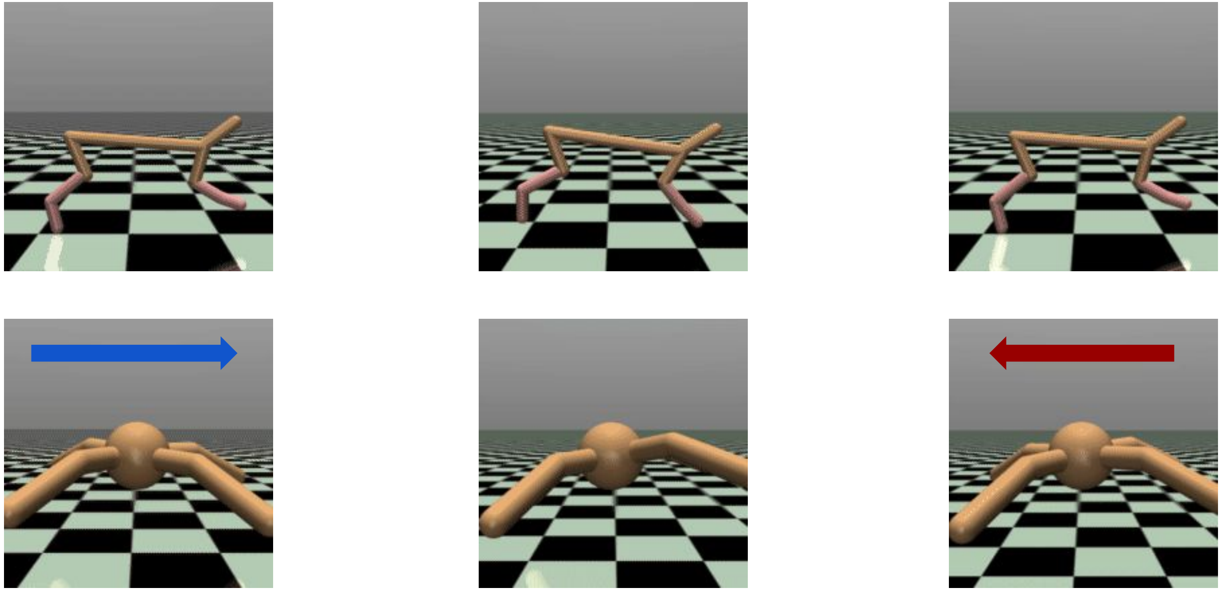
"Adversarial examples"

Neural networks are not safe enough for mission-critical tasks

Neural network controlled robots (simulated) + adversarial sensor noise

HalfCheetah

Ant



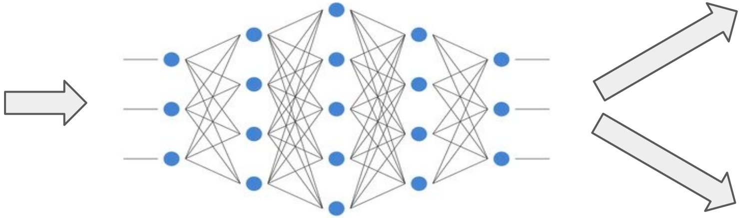
No attack

MAD attack
[Z*C*XLLBH **NeurIPS 2020**]

Optimal attack
[Z*CBH **ICLR 2021**]

Formal verification of neural networks: robustness verification

Eykholt et al. 2018

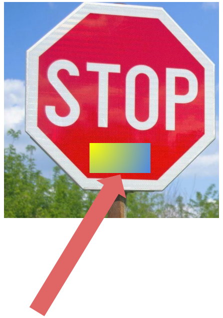


STOP 😀

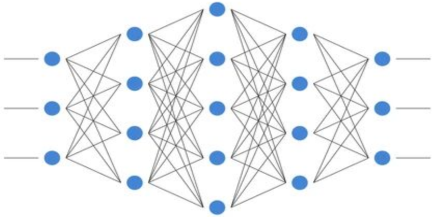
~~Speed limit~~ 😈

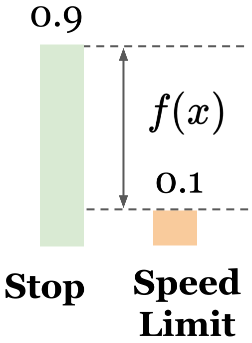
Goal: **prove** adversarial examples do *not* exist!

Formal verification of neural networks: robustness verification



Attacker may put *anything* here



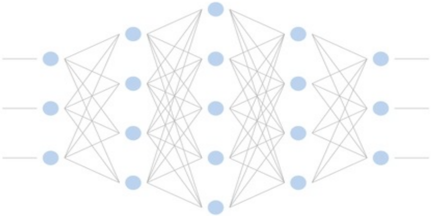


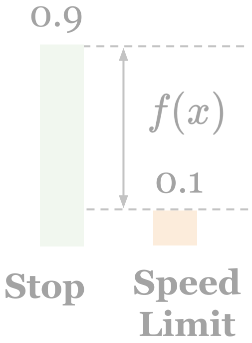
Score for each label

$$f(x) > 0 \Rightarrow \text{ No adversarial examples}$$

Formal verification of neural networks: robustness verification

Attacker may put *anything* here





Score for each label

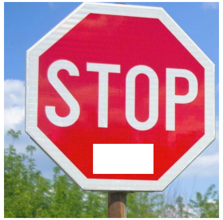
Just an **example** of how $f(x)$ can be defined

Just an **example** of verification problem

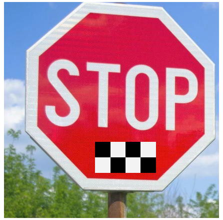
Prove: $\forall x \in \mathcal{S},\ f(x) > 0$

For multi-class cases, we can define multiple $f_i(x)$, one for each class

$\mathcal{S}$ = all possible pixel perturbations



$x_1 \in \mathcal{S}$



$x_2 \in \mathcal{S}$

$x_3 \in \mathcal{S}$

…

Verification example: ACAS Xu system

3MB DNN represents a large (2GB) lookup table for collision avoidance of unmanned aircraft

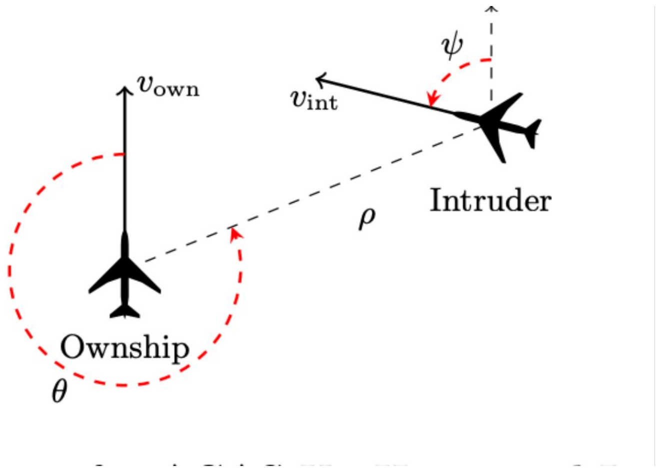
Input: $x \in \mathbb{R}^5$, x = (d, θ, ψ, v_own, v_in)
d: Distance; θ: relative angle; ψ: relative heading;
v_{own}, v_{in}: speeds

Output $y \in \mathbb{R}^5$: Clear of Conflict (COC), or advisory weak/strong left/right. Five scores for these actions:

y_0: COC, y_1: weak left at 1.5 deg/s

y_2: strong left at 3.0 deg/s

y_3: weak right y_4: strong right



“Neural Network Verification Methods for Closed-Loop ACAS Xu Properties”, Bak et. al.

Verification example: ACAS Xu system

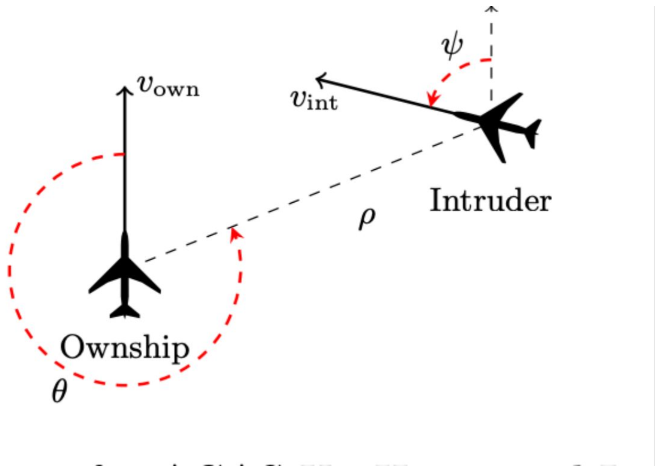
3MB DNN represents a large (2GB) lookup table for collision avoidance of unmanned aircraft

Input: $x \in \mathbb{R}^5$, $x = (d, \theta, \psi, v_{own}, v_{in})$
d: Distance; θ: relative angle; ψ: relative heading; v_{own}, v_{in}: speeds

Output $y \in \mathbb{R}^5$: Clear of Conflict (COC), or advisory weak/strong left/right.

Requirement: E.g. If the intruder is far then the score for COC should be above some threshold

$\forall x \in \mathbb{R}^5$, $d \geq 55947$, $v_{own} \geq 1145$, $v_{in} \leq 60$
Prove: $y_0 > 1500$



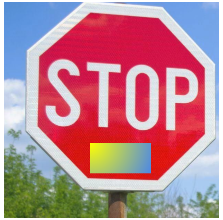
“Neural Network Verification Methods for Closed-Loop ACAS Xu Properties”, Bak et. al.

Verification of neural networks

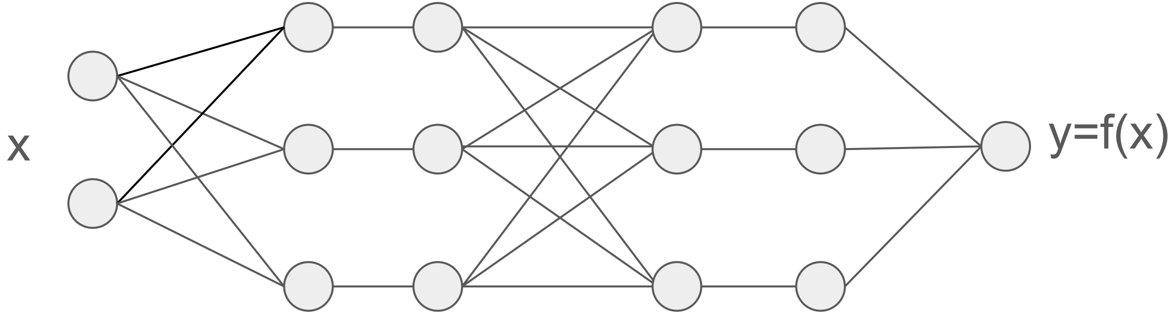
For all desired input x (image, text, sensor readings, etc), f(x) meets some conditions

Satisfiability problem: does there *exist* x, such that f(x) does not meet these conditions?

$$\exists\ x \in S \ \wedge\ y \leq 0 \ \wedge\ y = f(x)$$



Can also be multiple conditions, like in some ACAS Xu requirements and robustness verification of multi-class classification



Verification example: ACAS Xu system (from VNN-COMP)

Input: $x \in \mathbb{R}^5$, x = (d, θ, ψ, v_own, v_in)
d: Distance; θ: relative angle; ψ: relative heading; v_{own}, v_{in}: speeds

Output $y \in \mathbb{R}^5$: y_0: COC, y_1: weak left, y_2: strong left, y_3: weak right, y_4: strong right

```
; Unscaled Input 0: (55947.691, 60760)
(assert (<= X_0 0.679857769))
(assert (>= X_0 0.6))

; Unscaled Input 1: (-3.141592653589793, 3.141592653589793)
(assert (<= X_1 0.5))
(assert (>= X_1 -0.5))

; Unscaled Input 2: (-3.141592653589793, 3.141592653589793)
(assert (<= X_2 0.5))
(assert (>= X_2 -0.5))

; Unscaled Input 3: (1145, 1200)
(assert (<= X_3 0.5))
(assert (>= X_3 0.45))

; Unscaled Input 4: (0, 60)
(assert (<= X_4 -0.45))
(assert (>= X_4 -0.5))

; Unsafe if COC is maximal
(assert (<= Y_1 Y_0))
(assert (<= Y_2 Y_0))
(assert (<= Y_3 Y_0))
(assert (<= Y_4 Y_0))
```

Requirements written in VNNLIB format

multiple conditions on y

$(y_1 - y_0 <= 0) \land (y_2 - y_0 <= 0) \land (y_3 - y_0 <= 0) \land (y_4 - y_0 <= 0)$

Verification of neural networks

$$\exists\ x \in S \ \wedge \ y \leq 0 \ \wedge \ y = f(x)$$

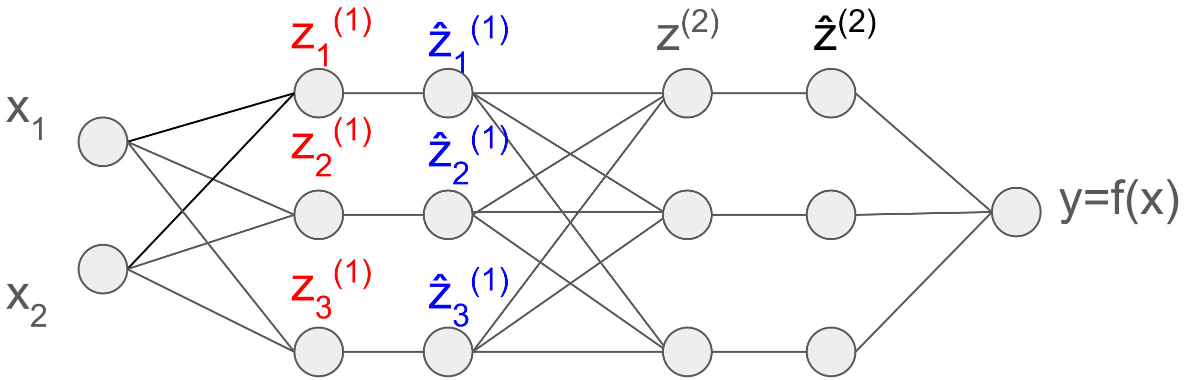
$x \in S$ condition is easy to handle for box constraints:

$$x_i \leq u_i \ \wedge \ x_i \geq l_i$$

How to handle $y = f(x)$?

Verification of neural networks

How to handle the constraint $y = f(x)$?

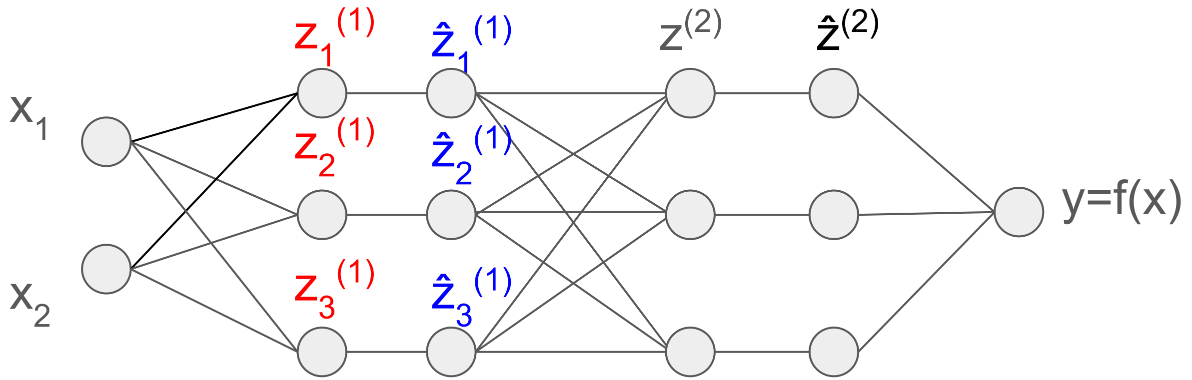


$$\hat{z}_j^{(i)} = \sigma(z_j^{(i)})$$

Linear layers: $z^{(1)} = W^{(1)} x$ $\quad z^{(2)} = W^{(2)} \hat{z}^{(1)}$ $\quad y = w^{(3)T} \hat{z}^{(2)}$

Verification of neural networks

How to handle the constraint y = f(x)?

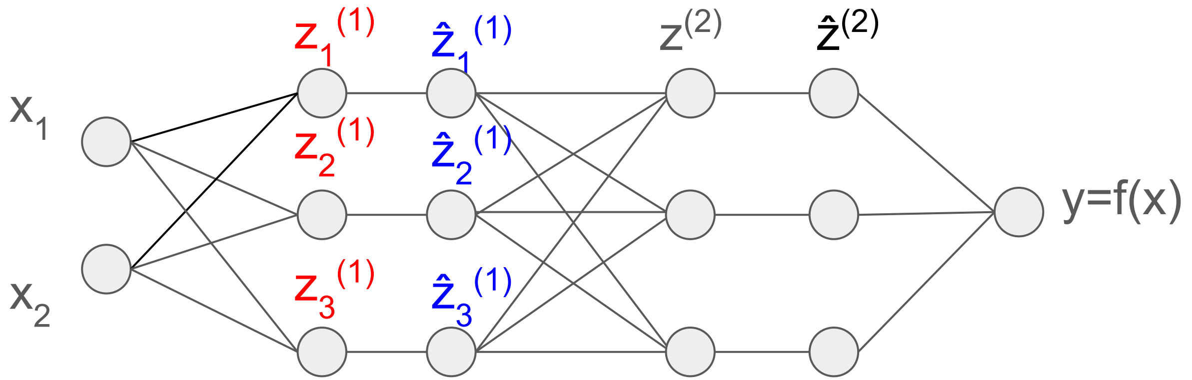


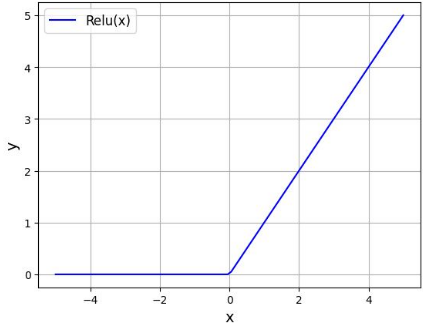
Linear layers: $z_1 = W^{(1)} x$ $\quad z^{(2)} = W^{(2)} \hat{z}^{(1)}$ $\quad y = w^{(3)T} \hat{z}^{(2)}$

Directly copy all the linear equality constraints to the SMT formulation.

Verification of neural networks

How to handle the constraint y = f(x)?





$$\hat{z}_j^{(i)} = \mathrm{ReLU}(z_j^{(i)}) \Rightarrow (z_j^{(i)} \geq 0 \land \hat{z}_j^{(i)} = z_j^{(i)}) \lor (z_j^{(i)} < 0 \land \hat{z}_j^{(i)} = 0)$$

Verification of neural networks

Satisfiability problem: $\exists\ x \in S\ \wedge\ y \leq 0\ \wedge\ y = f(x)$

$x_i \leq u_i\ \wedge\ x_i \geq l_i$ for each dimension of x

$((z_j^{(i)} \geq 0\ \wedge\ \hat{z}_j^{(i)} = z_j^{(i)})\ \vee\ (z_j^{(i)} < 0\ \wedge\ \hat{z}_j^{(i)} = 0))$ for each ReLU neuron

$z_1 = W^{(1)} x\ \wedge\ z^{(2)} = W^{(2)} \hat{z}^{(1)}\ \wedge\ y = w^{(3)T} \hat{z}^{(2)}\ \wedge\ y \leq 0$

Add all clauses to the formula and solve using DPLL(T) with **Linear Real Arithmetic**.

In general this is very slow! Faster methods in the next a few lectures.

Summary

- Machine learning
- Neural networks
- Verification problems on neural networks
- Neural network verification as a SMT problem
- Please checkout **verification of neural networks competitions** (VNN-COMP) for more examples of verification problems
  - https://sites.google.com/view/vnn2023
  - https://sites.google.com/view/vnn2022
  - https://sites.google.com/view/vnn2021
- Next lecture: integer programming and linear programming formulations for neural network verification
- **Reading**:
  - https://arxiv.org/pdf/1711.07356.pdf
  - https://arxiv.org/pdf/1711.00851.pdf