Neural network verification as piecewise linear optimization

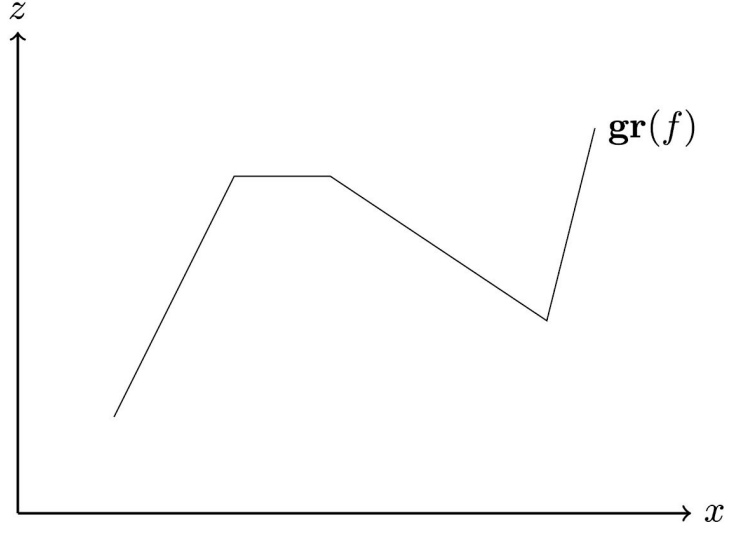
Joey Huchette
Rice University

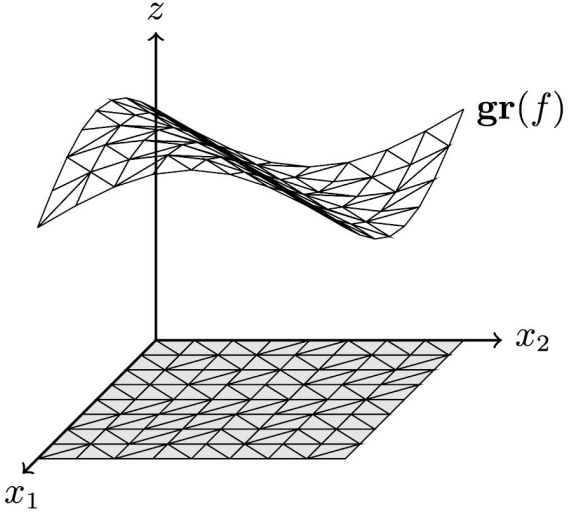
Joint work with:
Ross Anderson[1], Bochuan Lyu[2], Will Ma[3], Krunal Patel[1], Christian Tjandraatmadja[1], and Juan Pablo Vielma[1]

[1]Google Research
[2]Rice University
[3]Columbia University

Piecewise linear functions

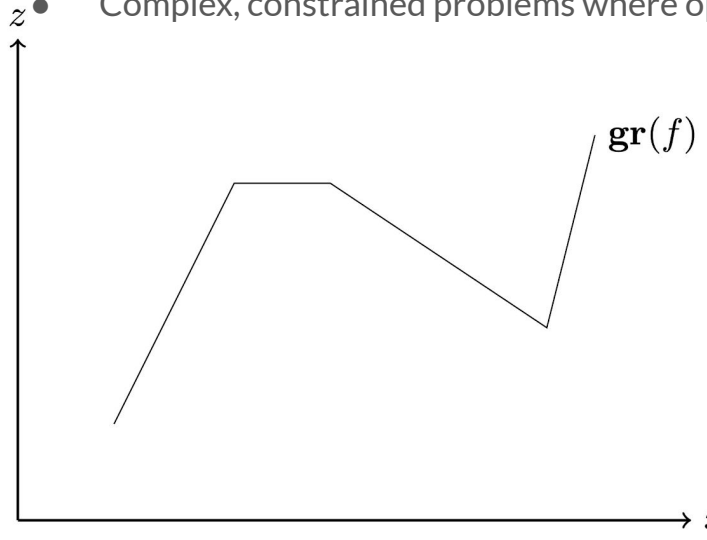
- f is piecewise linear = domain partition + f is linear on each partition piece
- The "simplest" nonlinear functions?
- Capable of articulating complex nonlinear, nonconvex behaviors
- "Universal approximators"

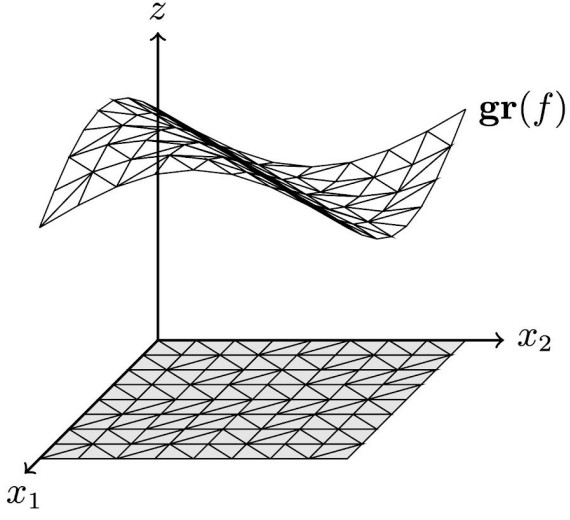




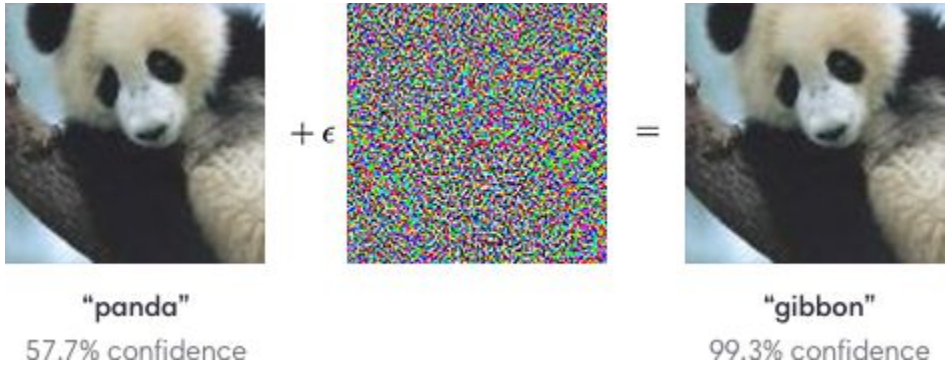
Piecewise linear functions in OR

- Used to model:
  - diminishing returns or economies of scale
  - Physical systems (e.g. process engineering)
- Low-dimensional or separable
- Sophisticated exact methods
  - Special ordered set constraints (Beale and Tomlin 1970, 1976)
  - MIP formulations (e.g. Vielma et al. 2010)
- Complex, constrained problems where optimality matters





Neural network verification



$$\begin{aligned} \max \quad & \mathbb{P}[x \text{ is a gibbon}] - \mathbb{P}[x \text{ is a panda}] \\ \text{s.t.} \quad & ||x - (\text{reference panda})||_\infty \leq \epsilon \end{aligned}$$

- **Adversarial Examples**: Small change in input → big change in output
- Neural networks are extremely susceptible to adversarial attacks (Szegedy et al. 2014, Papernot et al. 2015, Carlini et al. 2016)
- Hugely popular area of ML research (~10,000 citations for three papers above)
- Optimization: Find most pathological perturbation around labeled image
- No big changes in small neighborhood → model is (locally) robust

Image source: https://openai.com/blog/adversarial-example-research/

Key insights and approach

- Trained neural network = piecewise linear function, built through composition
- Neural network verification = piecewise linear optimization

(Neither is novel to this work)

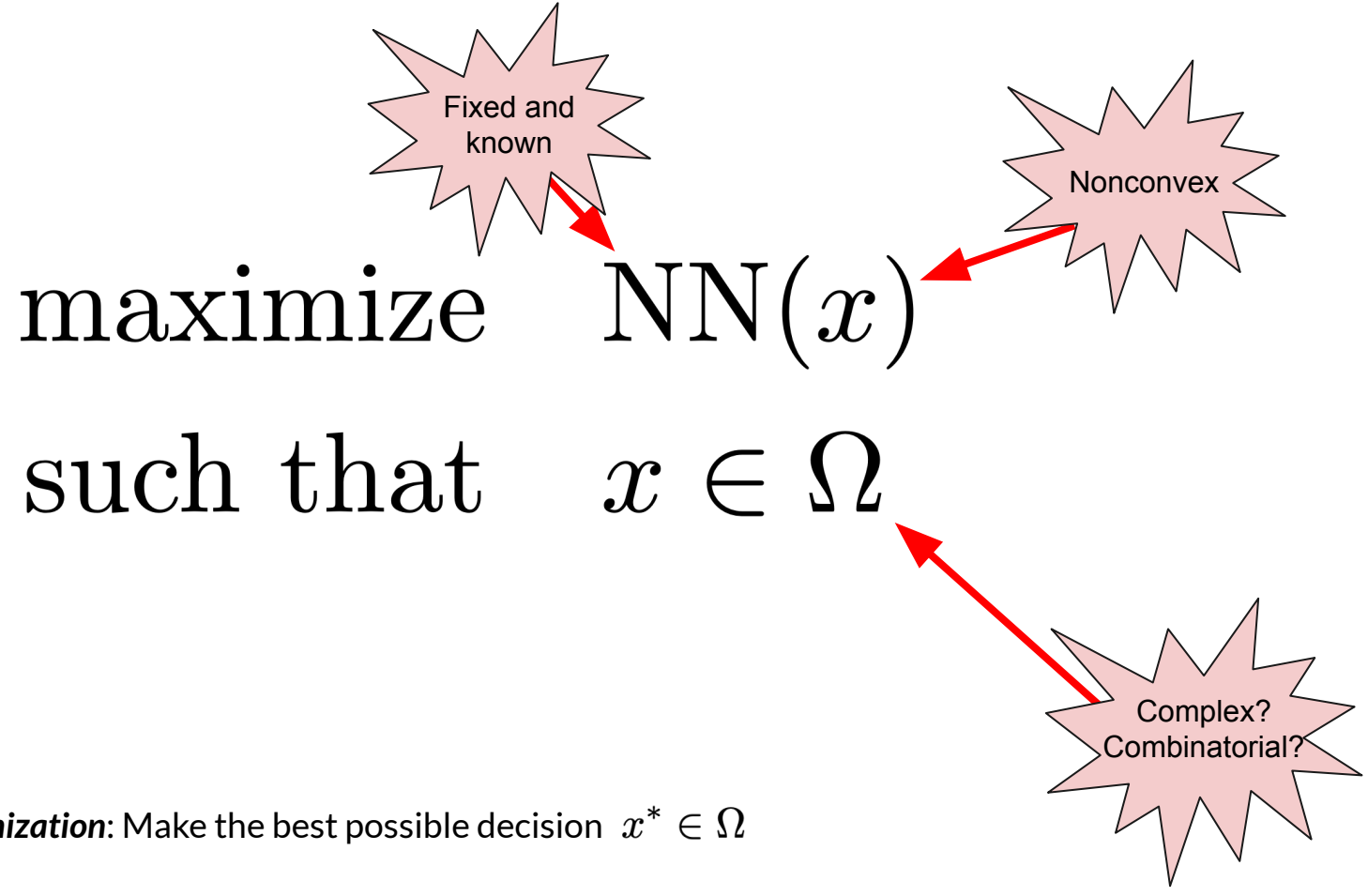
Goal: Use OR techniques to produce good algorithms for neural network verification

Optimization over trained neural networks

- *Strong mixed-integer programming formulations for trained neural networks.*
  - Anderson, H., Tjandraatmadja, and Vielma. Proceedings of IPCO 2019.
  - Anderson, H., Ma, Tjandraatmadja, and Vielma. Mathematical Programming, 2020.
- *The convex barrier, revisited: Tightened single-neuron relaxations for neural network verification.*
  - Tjandraatmadja, Anderson, H., Ma, Patel, and Vielma. Proceedings of NeurIPS 2020.

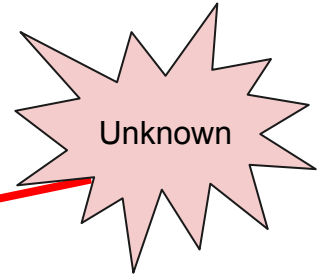
An aside: Optimization over trained neural networks

Optimization over a trained neural network



End goal is ***optimization***: Make the best possible decision $x^* \in \Omega$

Optimization over an unknown function

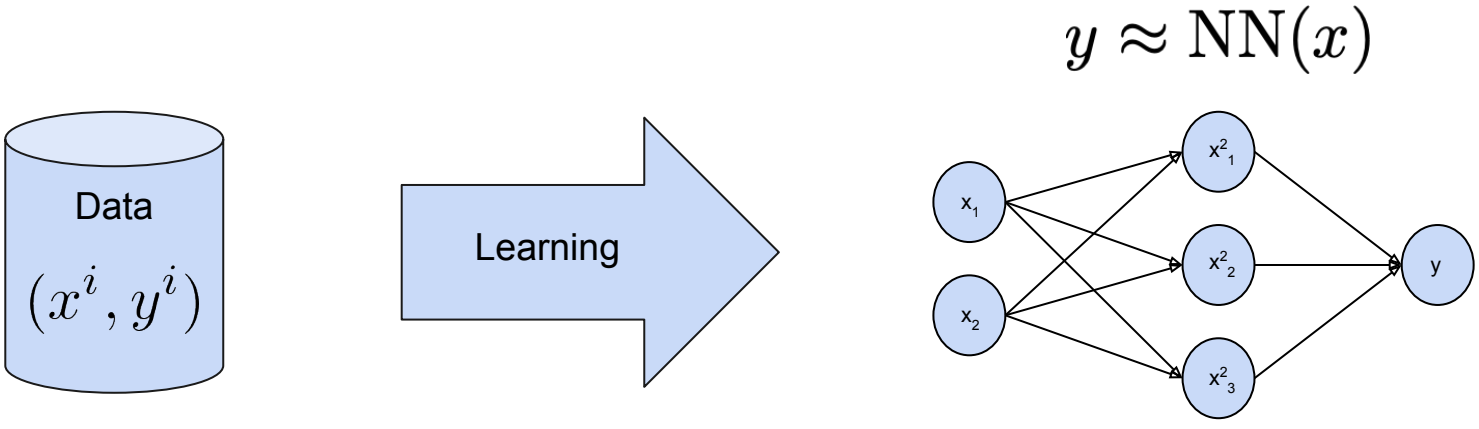


$$\begin{aligned} \text{maximize} \quad & f(x) \\ \text{such that} \quad & x \in \Omega \end{aligned}$$

End goal is ***optimization***: Make the best possible decision $x^* \in \Omega$

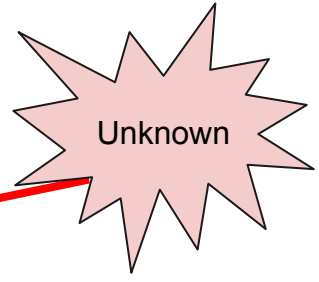
Fitting unknown functions to make predictions

Supervised learning: Learn a function using historical input/output data



End goal is ***generalization***: Given "reasonable" unseen point (x^*, y^*), want $y^* \approx \mathrm{NN}(x^*)$

Fitting unknown functions to make predictions



$$\begin{aligned} \text{maximize} \quad & f(x) \\ \text{such that} \quad & x \in \Omega \end{aligned}$$

End goal is ***optimization***: Make the best possible decision $x^* \in \Omega$

Fitting unknown functions to make predictions

$$\begin{aligned} \text{maximize} \quad & \text{NN}(x) \\ \text{such that} \quad & x \in \Omega \end{aligned}$$

End goal is ***optimization***: Make the best possible decision $x^* \in \Omega$

Application: Deep reinforcement learning

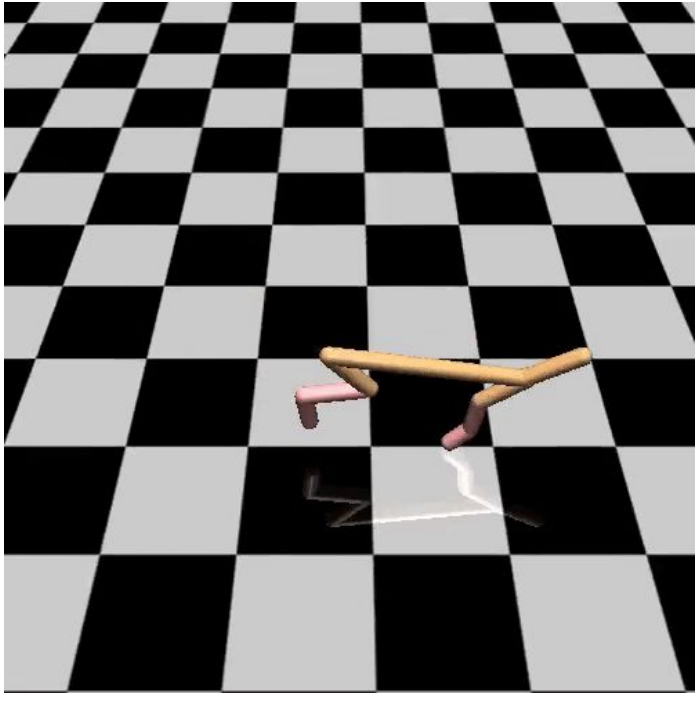
Prediction: future cost of action *a* in state *x*:

$$Q(x, a) \approx \mathrm{NN}(x, a)$$

Optimization: pick the lowest cost action:

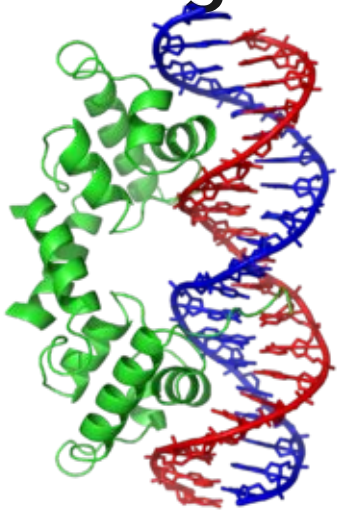
$$\min_a \; c(x, a) + \alpha \cdot \mathrm{NN}(x, a)$$

In a combinatorial or continuous action space, optimization can be hard! (e.g. Ryu 2019)



Teach a cheetah to run (without falling over)

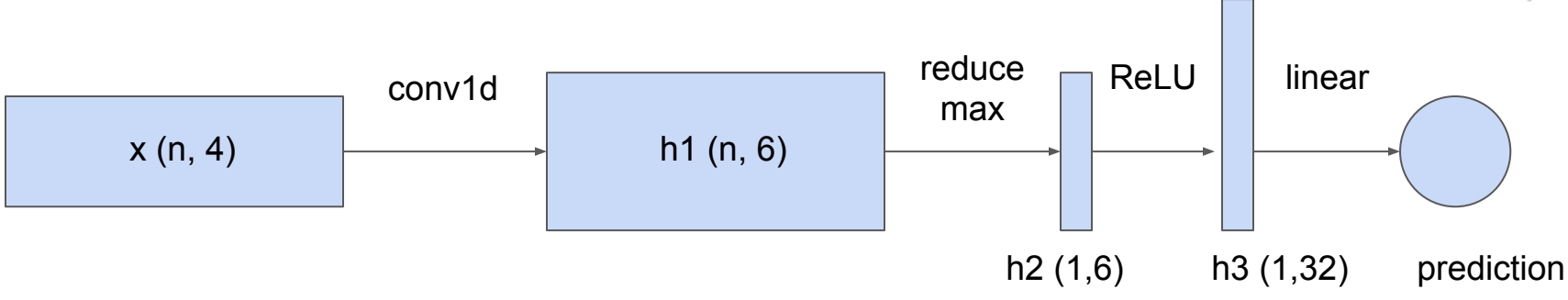
Application: Designing DNA for protein binding



Prediction: probability/strength of DNA sequence binding to a given protein

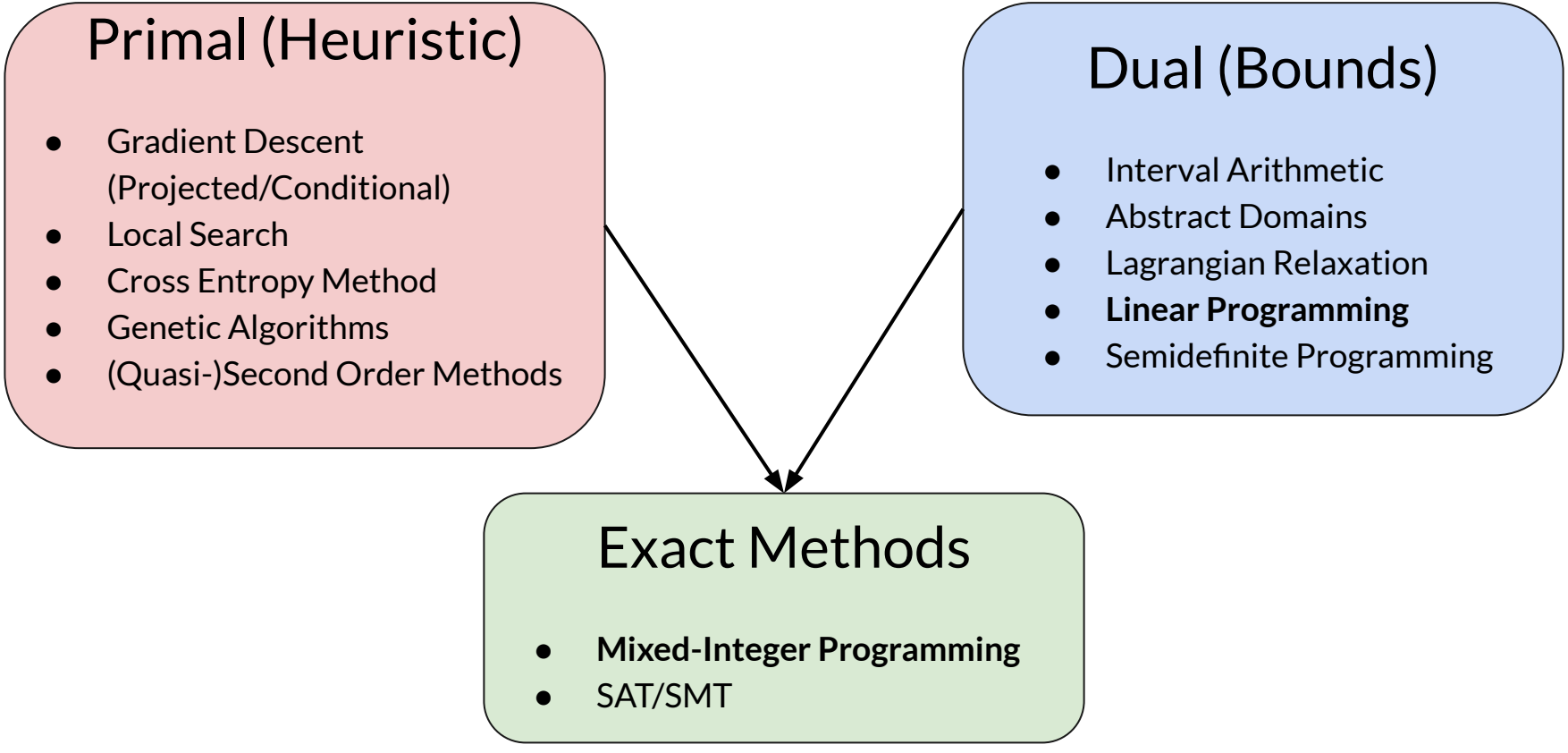
Optimization: find the best/many binding sequences (potentially with side constraints)

Good neural network architectures for prediction problem (Alipanahi et al. 2015, Zeng et al. 2016)

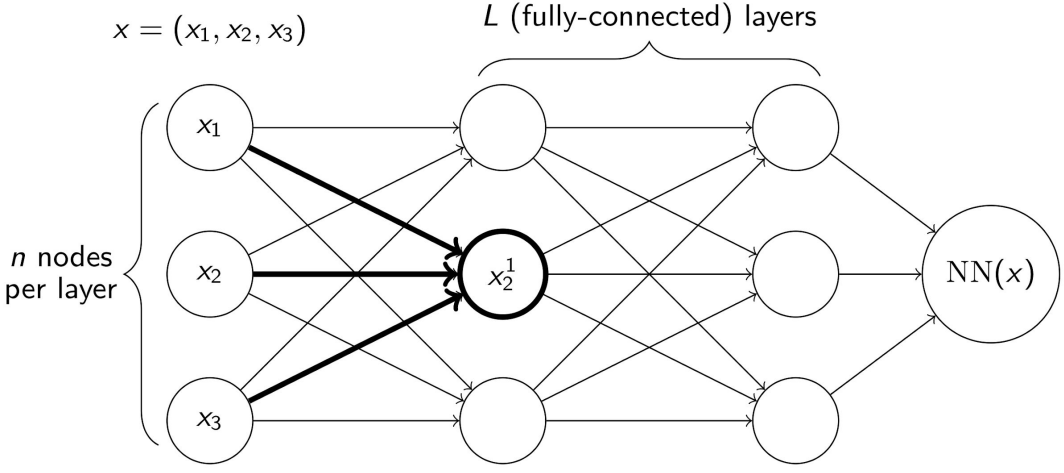


How to solve these problems?

An algorithmic zoo

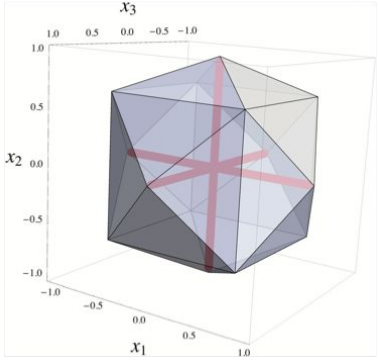


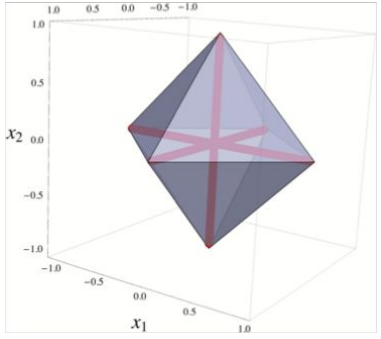
Neural networks (in one slide)



- At i'th neuron in k'th layer:
  - $x^k_i = \sigma(w^{k,i} \cdot x^{k-1} + b^{k,i})$
  - x^{k-1} is vector of variables from previous layer
  - σ is a nonlinear activation function
  - Typically use ReLU: $\sigma(v) = \max\{0, v\}$
- If σ is piecewise linear, then NN is piecewise linear (built through composition)

MIP formulations (in one slide)





- A MIP formulation for some set $S \subseteq R^n$ is:
  - A polyhedra $Q \subseteq R^{n+r}$, where

    $$\text{Proj}_x(\{ (x,z) \in Q \mid z \in Z^r \}) = S$$

- What makes a MIP formulation good?
  - **Size**: r is small, Q is "simple"
  - **Sharp**: $\text{Proj}_x(Q) = \text{Conv}(S)$
  - ***Hereditarily sharp***: Sharp after any fixings of binary variables z
  - ***Ideal (perfect)***: $\text{ext}(Q) \subseteq R^n \times Z^r$

- Ideal ⇒ sharp, so…

Ideal formulations = Best possible = Our goal

Most important theoretical result

- **Idea:** Formulate each neuron in network separately, using MIP
- **Result:** MIP formulations for convex piecewise linear function with d pieces

	Big-M	Our Formulation	Disjunctive ("Balas")
Sharpness of LP-relaxation for Single Neuron	not sharp	sharp, **ideal when** $d = 2$	**ideal**
# of Continuous Variables	$\mathbf{\Theta(Ln)}$	$\mathbf{\Theta(Ln)}$	$\Theta(Ln^2 d)$

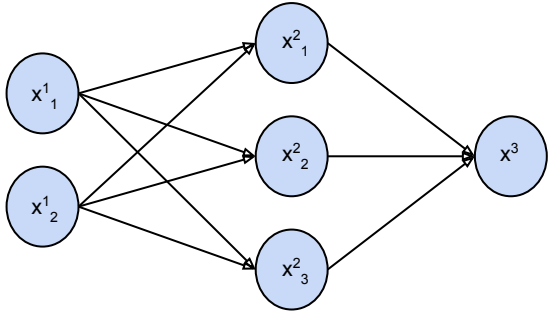
n = Number of neurons per layer, L = number of layers

MIP formulations for a single ReLU neuron

- Single neuron: $y = \sigma(w \cdot x + b)$, where $\sigma(v) = \max\{0,v\}$
- Input x, output y, learned parameters w and b
- Bounds on inputs $L \le x \le U$ from, e.g., interval arithmetic
- Set containing all feasible input/output pairs:

  $\{(x,y) : y = \sigma(w \cdot x + b), L \le x \le U\}$
- Big-M formulation for single ReLU neuron (e.g. Fischetti and Jo 2018, Serra et al. 2018, Tjeng et al. 2019, etc.)

$$
\begin{aligned}
y &\geq 0 \\
y &\geq w \cdot x + b \\
y &\leq M^{+} z \\
y &\leq w \cdot x + b - M^{-}(1 - z) \\
(x, y, z) &\in [L, U] \times \mathbb{R} \times \{0, 1\}
\end{aligned}
$$

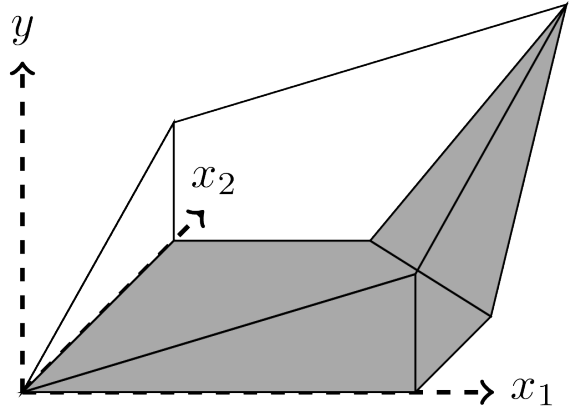


- How strong is it?

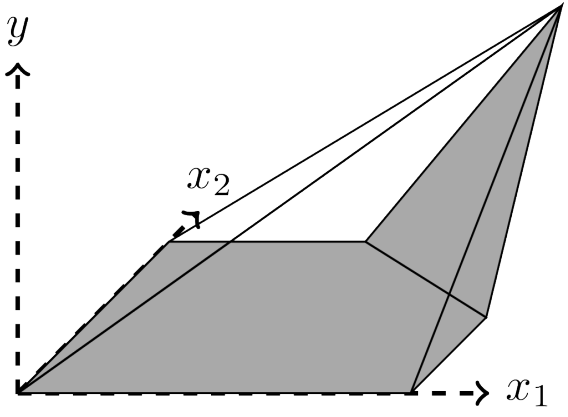
MIP formulation strength

- How strong is it? *Not very!*

  Can be arbitrarily bad, even in fixed input dimension



big-M formulation



tightest possible formulation

- How to close the gap?

Ideal formulation for a single ReLU neuron

Theorem (Anderson, H., Tjandraatmadja, Vielma 2019)

- An ideal formulation for {(x,y) : y = max{0, w · x + b}, L ≤ x ≤ U} is

$$y \geq w \cdot x + b \tag{1a}$$

$$y \leq \sum_{i \in I} w_i \left(x_i - L_i(1 - z)\right) + \left(b + \sum_{i \notin I} w_i U_i\right) z \quad \forall I \subseteq [n] \tag{1b}$$

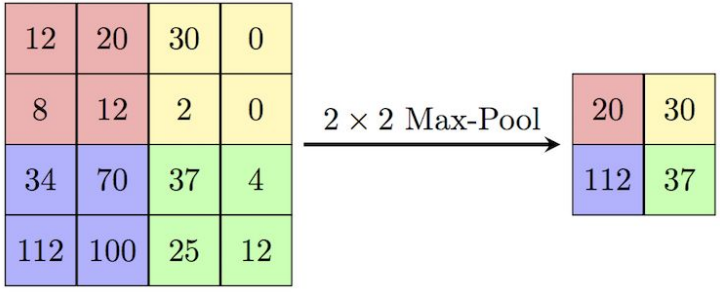
$$(x, y, z) \in [L, U] \times \mathbb{R}_{\geq 0} \times \{0, 1\}. \tag{1c}$$

- Each inequality in (1b) is facet-defining (under very mild conditions).
- Moreover, we can identify the most violated constraint in (1b) in O(n) time.

- Big-M formulation = (1a), (1c), and two constraints from (1b)
- **Idea:** Start with big-M formulation, use cut callbacks to separate (1b) as-needed

Formulations for convex PWL functions

- Max-of-d affine functions ≡
  - Max pooling (small d)
  - Reduce max (large d)



Proposition (Anderson, H., Ma, Tjandraatmadja, Vielma 2020)

- A hereditarily sharp MIP formulation for $\{(x,y) : y = \max\{w^k \cdot x + b^k\}_{k=1}^d, L \le x \le U\}$ is

$$y \le \sum_{i=1}^{\eta} \left(w_i^{I(i)} x_i + \sum_{k=1}^{d} \max\{(w_i^k - w_i^{I(i)})L_i, (w_i^k - w^{I(i)})U_i\} z_k \right) + \sum_{k=1}^{d} b^k z_k \quad \forall I : [\eta] \to [d]$$

$$y \ge w^k \cdot x + b^k \quad \forall k \in [d]$$

$$(x, z) \in D \times \Delta^d$$

$$z \in \{0, 1\}^d.$$

- The most violated inequality can be identified in O(dn) time.

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

$$\left\{(x, y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k \right\}$$

Step 1: Write down ideal "multiple choice" formulation (i.e. the "Balas" formulation):

$$(x, y) = \sum_{k=1}^{d} (\tilde{x}^k, w^k \cdot \tilde{x}^k + b^k z_k)$$

$$w^k \cdot \tilde{x}^k + b^k z_k \geq w^\ell \cdot \tilde{x}^k + b^\ell z_k \qquad \forall k, \ell \in [d] : \; k \neq \ell$$

$$\tilde{x}^k \in z_k \cdot D \qquad \forall k \in [d]$$

$$z \in \Delta^d.$$

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

$$\left\{(x, y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k\right\}$$

Step 2: Re-write constraints in "set" form:

$$\begin{aligned}(x, y) &= \sum_{k=1}^{d} (\tilde{x}^k, w^k \cdot \tilde{x}^k + b^k z_k) \\ \tilde{x}^k &\in z_k \cdot D_{|k} && \forall k \in [d] \\ z &\in \Delta^d,\end{aligned}$$

where

$$\begin{aligned}D_{|k} &\equiv \left\{x \in D : k \in \arg\max_{\ell=1}^{d} w^\ell \cdot x + b^\ell\right\} \\ &= \left\{x \in D : w^k \cdot x + b^k \geq w^\ell \cdot x + b^\ell \quad \forall \ell \neq k\right\}\end{aligned}$$

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

$$\left\{(x,y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k\right\}$$

Step 3: Rewrite all logic as bounds on output y (a *primal* characterization):

$$y \leq \overline{g}(x,z) \equiv \max_{\tilde{x}^1,\ldots,\tilde{x}^d} \left\{ \sum_{k=1}^{d} w^k \cdot \tilde{x}^k + b^k z_k : \begin{array}{ll} x = \sum_k \tilde{x}^k & \\ \tilde{x}^k \in z_k \cdot D_{|k} & \forall k \end{array} \right\}$$

$$y \geq \underline{g}(x,z) \equiv \min_{\tilde{x}^1,\ldots,\tilde{x}^d} \left\{ \sum_{k=1}^{d} w^k \cdot \tilde{x}^k + b^k z_k : \begin{array}{ll} x = \sum_k \tilde{x}^k & \\ \tilde{x}^k \in z_k \cdot D_{|k} & \forall k \end{array} \right\}$$

$$(x,z) \in D \times \Delta^d,$$

where

$$\begin{aligned} D_{|k} &\equiv \left\{x \in D : k \in \arg\max_{\ell=1}^{d} w^\ell \cdot x + b^\ell\right\} \\ &= \left\{x \in D : w^k \cdot x + b^k \geq w^\ell \cdot x + b^\ell \quad \forall \ell \neq k\right\} \end{aligned}$$

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

$$\left\{(x,y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k \right\}$$

Step 4: Apply Lagrangian relaxation to aggregation constraints (a *dual* characterization)

$$y \leq \overline{\alpha} \cdot x + \sum_{k=1}^{d} \left(\max_{x^k \in D_{|k}} \{(w^k - \overline{\alpha}) \cdot x^k\} + b^k \right) z_k \quad \forall \overline{\alpha} \in \mathbb{R}^\eta$$

$$y \geq \underline{\alpha} \cdot x + \sum_{k=1}^{d} \left(\min_{x^k \in D_{|k}} \{(w^k - \underline{\alpha}) \cdot x^k\} + b^k \right) z_k \quad \forall \underline{\alpha} \in \mathbb{R}^\eta$$

$$(x,z) \in D \times \Delta^d.$$

End of analysis: in general, $D_{|k}$ is complicated.
Can separate over via subgradient method, or...

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

$$\left\{(x,y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k \right\}$$

Step 1: Write down ideal "multiple choice" formulation (i.e. the "Balas" formulation):

$$(x,y) = \sum_{k=1}^{d} (\tilde{x}^k, w^k \cdot \tilde{x}^k + b^k z_k)$$

$$w^k \cdot \tilde{x}^k + b^k z_k \geq w^\ell \cdot \tilde{x}^k + b^\ell z_k \qquad \forall k, \ell \in [d] : \; k \neq \ell$$

$$\tilde{x}^k \in z_k \cdot D \qquad \forall k \in [d]$$

$$z \in \Delta^d.$$

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

$$\left\{(x, y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k\right\}$$

Step 2: Re-write constraints in "set" form:

$$\begin{aligned}(x, y) &= \sum_{k=1}^{d} (\tilde{x}^k, w^k \cdot \tilde{x}^k + b^k z_k) \\ \tilde{x}^k &\in z_k \cdot D_{|k} && \forall k \in [d] \\ z &\in \Delta^d,\end{aligned}$$

where

$$\begin{aligned}D_{|k} &\equiv \left\{x \in D : k \in \arg\max_{\ell=1}^{d} w^\ell \cdot x + b^\ell\right\} \\ &= \left\{x \in D : w^k \cdot x + b^k \geq w^\ell \cdot x + b^\ell \quad \forall \ell \neq k\right\}\end{aligned}$$

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

$$\left\{(x,y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k\right\}$$

Step 3: Rewrite all logic as bounds on output y (a *primal* characterization):

$$y \leq \overline{g}(x,z) \equiv \max_{\tilde{x}^1,\ldots,\tilde{x}^d} \left\{ \sum_{k=1}^{d} w^k \cdot \tilde{x}^k + b^k z_k : \begin{array}{ll} x = \sum_k \tilde{x}^k & \\ \tilde{x}^k \in z_k \cdot D_{|k} & \forall k \end{array} \right\}$$

$$y \geq \underline{g}(x,z) \equiv \min_{\tilde{x}^1,\ldots,\tilde{x}^d} \left\{ \sum_{k=1}^{d} w^k \cdot \tilde{x}^k + b^k z_k : \begin{array}{ll} x = \sum_k \tilde{x}^k & \\ \tilde{x}^k \in z_k \cdot D_{|k} & \forall k \end{array} \right\}$$

$$(x,z) \in D \times \Delta^d,$$

where

$$\begin{aligned} D_{|k} &\equiv \left\{x \in D : k \in \arg\max_{\ell=1}^{d} w^\ell \cdot x + b^\ell\right\} \\ &= \left\{x \in D : w^k \cdot x + b^k \geq w^\ell \cdot x + b^\ell \quad \forall \ell \neq k\right\} \end{aligned}$$

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

$$\left\{(x,y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k\right\}$$

Step 4: Relax domain constraints inside bounding functions

$$y \leq \overline{h}(x,z) \equiv \max_{\tilde{x}^1,\ldots,\tilde{x}^d} \left\{ \sum_{k=1}^{d} w^k \cdot \tilde{x}^k + b^k z_k : \begin{array}{ll} x = \sum_k \tilde{x}^k & \\ \tilde{x}^k \in z_k \cdot D & \forall k \end{array} \right\}$$

$$y \geq \underline{h}(x,z) \equiv \min_{\tilde{x}^1,\ldots,\tilde{x}^d} \left\{ \sum_{k=1}^{d} w^k \cdot \tilde{x}^k + b^k z_k : \begin{array}{ll} x = \sum_k \tilde{x}^k & \\ \tilde{x}^k \in z_k \cdot D & \forall k \end{array} \right\}$$

$$(x,z) \in D \times \Delta^d.$$

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

$$\left\{(x,y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k \right\}$$

Step 5: Replace lower bounds for a *hereditarily sharp* formulation:

$$\begin{aligned} y \leq \overline{h}(x,z) \equiv \max_{\tilde{x}^1,\ldots,\tilde{x}^d} \left\{ \sum_{k=1}^{d} w^k \cdot \tilde{x}^k + b^k z_k : \begin{array}{l} x = \sum_k \tilde{x}^k \\ \tilde{x}^k \in z_k \cdot D \end{array} \quad \forall k \right\} \\ y \geq w^k \cdot x + b^k \quad \forall k \in [d] \\ (x,z) \in D \times \Delta^d. \end{aligned}$$

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

$$\left\{(x, y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k \right\}$$

Step 6: Apply Lagrangian relaxation to aggregation constraints:

$$y \leq \overline{\alpha} \cdot x + \sum_{k=1}^{d} \left(\max_{x^k \in D} \{(w^k - \overline{\alpha}) \cdot x^k\} + b^k \right) z_k \qquad \forall \overline{\alpha} \in \mathbb{R}^\eta$$

$$y \geq w^k \cdot x + b^k \quad \forall k \in [d]$$

$$(x, z) \in D \times \Delta^d.$$

How do we get here?

Modeling the maximum of d affine functions over shared input domain D:

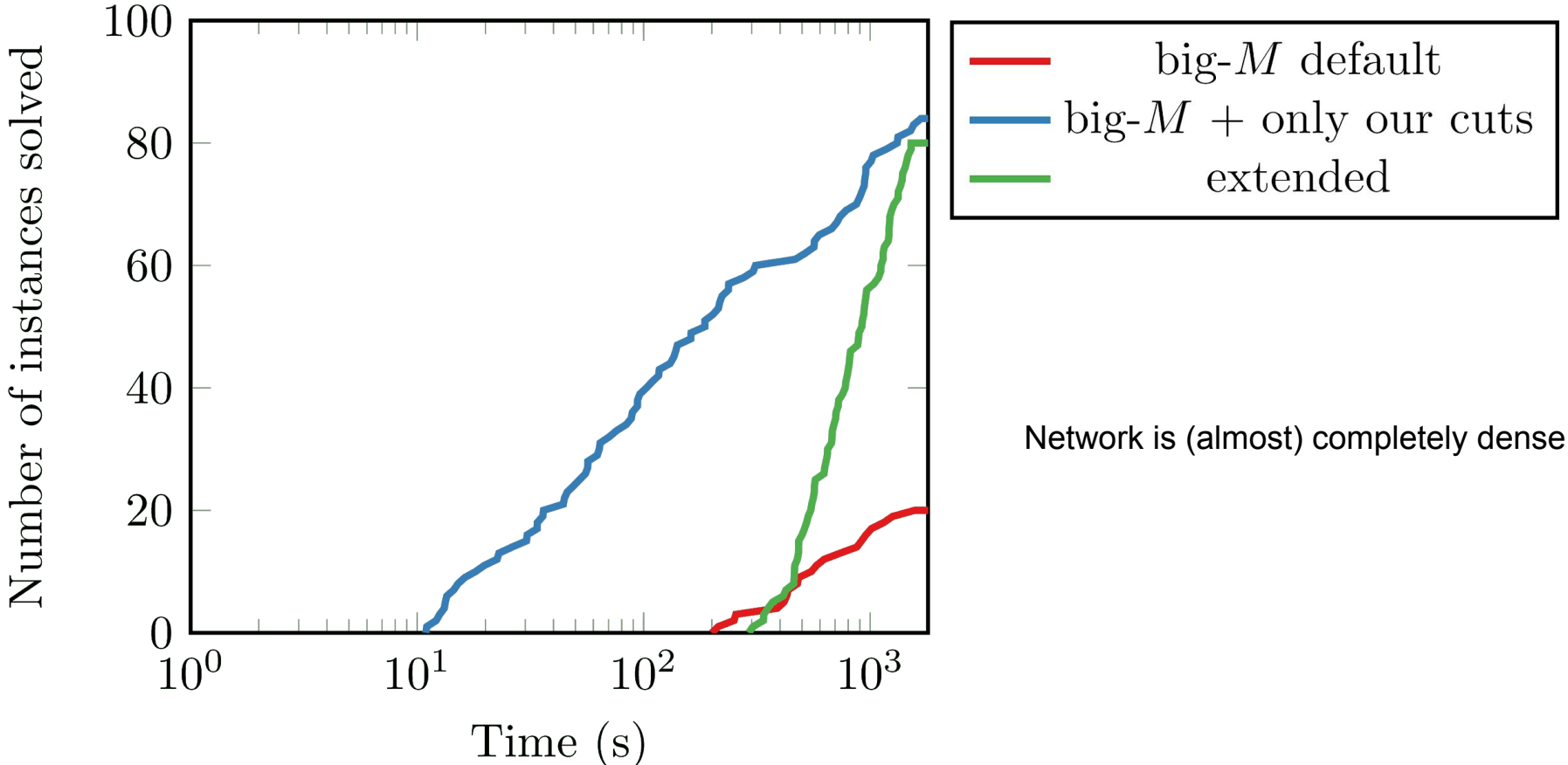
$$\left\{(x, y) \in D \times \mathbb{R} : y = \max_{k=1}^{d} w^k \cdot x + b^k\right\}$$

Step 7: Analyze further:

- **Proposition** If d=2, then the hereditarily sharp formulation is ideal.
- **Proposition** If the domain is a product of simplices, separation over hereditarily sharp formulation reduces to a transportation problem.
- **Proposition** If d=2 and the domain is a product of simplices, the transportation problem has a closed form solution and efficient separation.

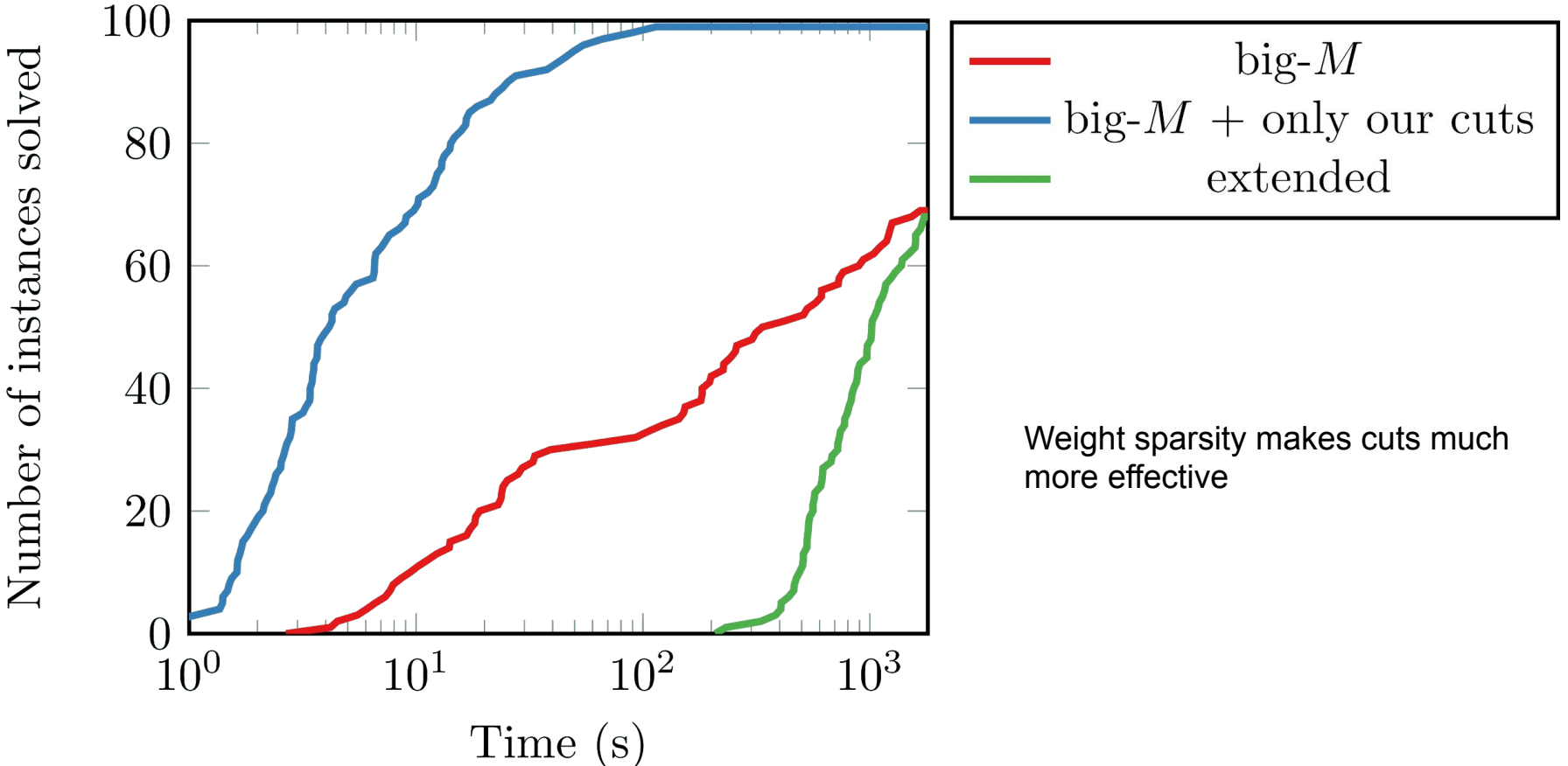
Computational Results

Network 1: Small network, standard training



Network is (almost) completely dense

Network 2: Small network + L1 regularization (Xiao et al. 2019)



Weight sparsity makes cuts much more effective

What if we want to go bigger?

Tightened convex relaxations

- **Problem:** MIP does not scale to large networks
- **Approach:** Linear programming (LP) is
  - Much more scalable than MIP
  - Still offers an incomplete verifier (no false positives, potentially false negatives)
- **`OptC2V`:** Project out binary variable to get LP relaxation, then use LP solver

Theorem (Tjandraatmadja, Anderson, H., Ma, Patel, Vielma 2020)

- An inequality description for Conv({(x,y) : y = max{0, w · x + b}, L ≤ x ≤ U}) is

$$y \geq w \cdot x + b$$

$$y \leq \sum_{i \in I} w_i (x_i - L_i) + \frac{\ell(I)}{U_h - L_h}(x_h - L_h) \quad \forall I, h \text{ s.t } \ell(I \cup \{h\}) < 0 \leq \ell(I)$$

$(x, y) \in [L, U] \times \mathbb{R}_{\geq 0}$.

where $\ell(I) = \sum_{i \in I} w_i L_i + \sum_{i \notin I} w_i U_i + b$.

- Moreover, we can identify the most violated constraint in in O(n) time.

What if we want to go *even* bigger?

Propagation algorithms

- **Problem:** Even LP won't scale to very large networks!
- **Approach:**
  - Relax LP even more so that it can be solved via *propagation* (e.g. Weng et al. 2018, Wong and Kolter 2018, Zang et al. 2018, Singh et al. 2019)
  - Use *only* two inequalities to bound output based on values of inputs:

$$L^k(x_1, \ldots, x_{k-1}) \leq x_k \leq U^k(x_1, \ldots, x_{k-1}) \quad \text{for each neuron } k$$

  - Propagation ≡ Fourier-Motzkin elimination (efficient in this case!)
- **Result:** Incomplete verifier that runs in O((# layers) x (# neurons per layer)^2) time
- But wait--How do we choose each L^k and U^k?
- **`FastC2V`:**
  1. Run once with "typical" choices
  2. Propagate forward, computing values x_k at each neuron k
  3. Compute most violated inequality for each neuron k, swap in for U^k
  4. Repeat as desired

Computational results

Method		MNIST 6x100	MNIST 9x100	MNIST 6x200	MNIST 9x200	MNIST ConvS	MNIST ConvB	CIFAR-10 ConvS
DeepPoly	#verified	160	182	292	259	162	652	359
	Time (s)	0.7	1.4	2.4	5.6	0.9	7.4	2.8
FastC2V	#verified	279	269	477	392	274	691	390
	Time (s)	8.7	19.3	25.2	57.2	5.3	16.3	15.3
LP	#verified	201	223	344	307	242	743	373
	Time (s)	50.5	385.6	218.2	2824.7	23.1	24.9	38.1
OptC2V	#verified	429	384	601	528	436	771	398
	Time (s)	136.7	759.4	402.8	3450.7	55.4	102.0	104.8
RefineZono	#verified	312	304	341	316	179	648	347
kPoly	#verified	441	369	574	506	347	736	399

Take-aways:

- OptC2V can verify the most instances
- FastC2V is nearly as good on larger networks, and an order of magnitude faster.

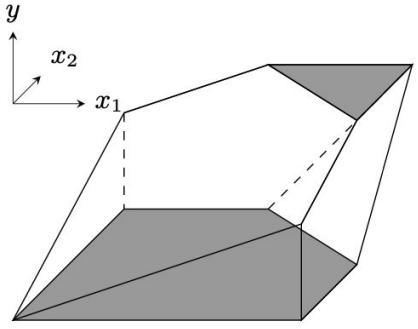
Extensions: Binarized and quantized networks

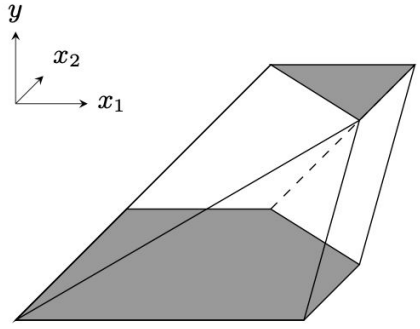
- Modern neural networks are *big*...and might not fit on small devices
- Using floating point numbers for each weight/activation value is potentially "wasteful"
- Binarized neural networks = replace floats with binary values! (e.g. Courbariaux 2016, Hubara 2016, Rastegari 2016)
- (Quantized neural networks = replace floats with "a few bits") (e.g. Hubara 2017, Jacob 2018, Zhou 2016)

New wrinkles for verification:

- How to handle discontinuity?
- Are convex relaxations good?
- Are cuts useful at the root, or in the tree?

(Ongoing work with Bochuan Lyu)





Thank you for your attention!

Questions?