Reverse-Engineering Deep ReLU Networks

David Rolnick and Konrad Körding

University of Pennsylvania

International Conference on Machine Learning (ICML) 2020

Reverse-engineering a neural network

Problem:

Recover network architecture and weights from black-box access.

Implications for:

- Proprietary networks
- Confidential training data
- Adversarial attacks

Is perfect reverse-engineering possible?

What if two networks define exactly the same function?

ReLU networks unaffected by:

- **Permutation:** re-labeling neurons/weights in any layer
- **Scaling:** at any neuron, multiplying incoming weights & bias by $c > 0$, multiplying outgoing weights by $1/c$

Our goal:

Reverse engineering deep ReLU networks up to permutation & scaling.

Related work

- Recovering networks with one hidden layer (e.g. Goel & Klivans 2017, Milli et al. 2019, Jagielski et al. 2019, Ge et al. 2019)
- Neuroscience, simple circuits in brain (Heggelund 1981)
- *No algorithm to recover even the first layer of a deep network*

Linear regions in a ReLU network

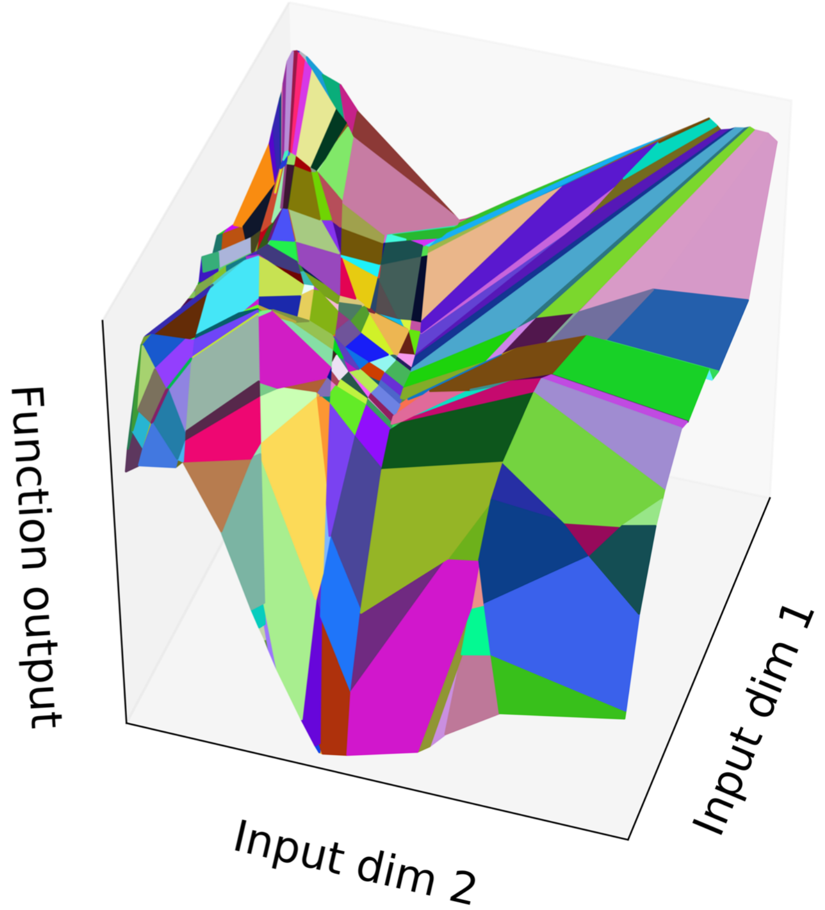
- Activation function:

$$\mathrm{ReLU}(z) = \max(0, z)$$

- Deep ReLU networks are piecewise linear functions:

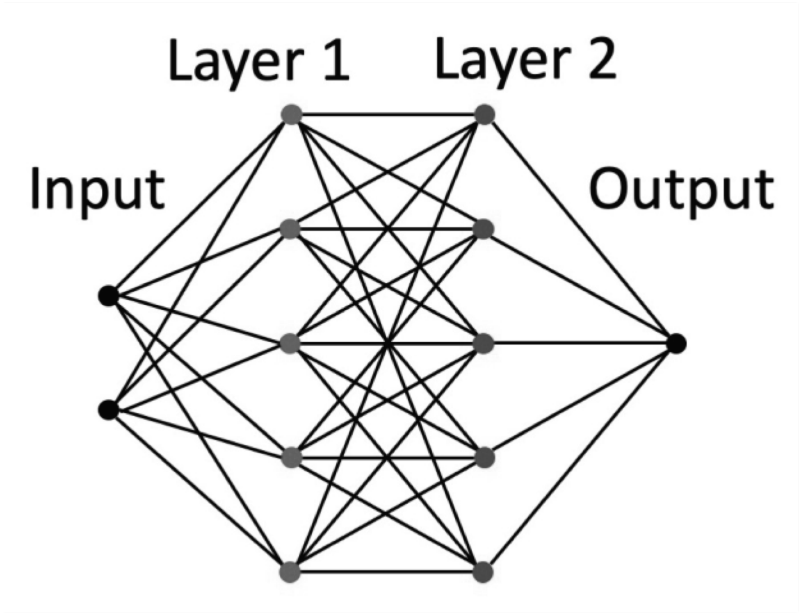
$$\mathcal{N} : \mathbb{R}^{n_{\mathrm{in}}} \to \mathbb{R}^{n_{\mathrm{out}}}$$

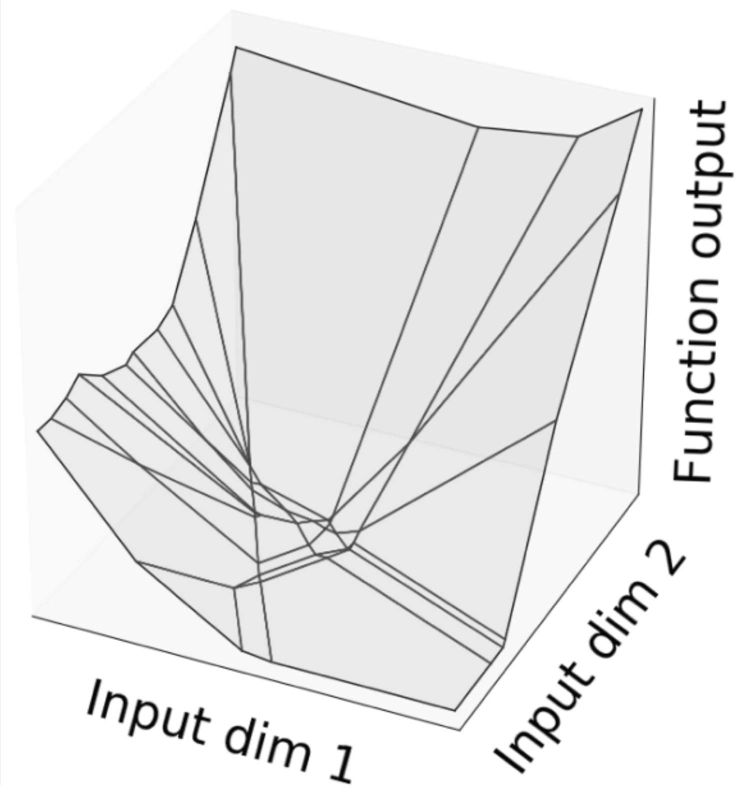
- Linear regions = pieces of $\mathbb{R}^{n_{\mathrm{in}}}$ on which $\nabla\mathcal{N}$ is constant

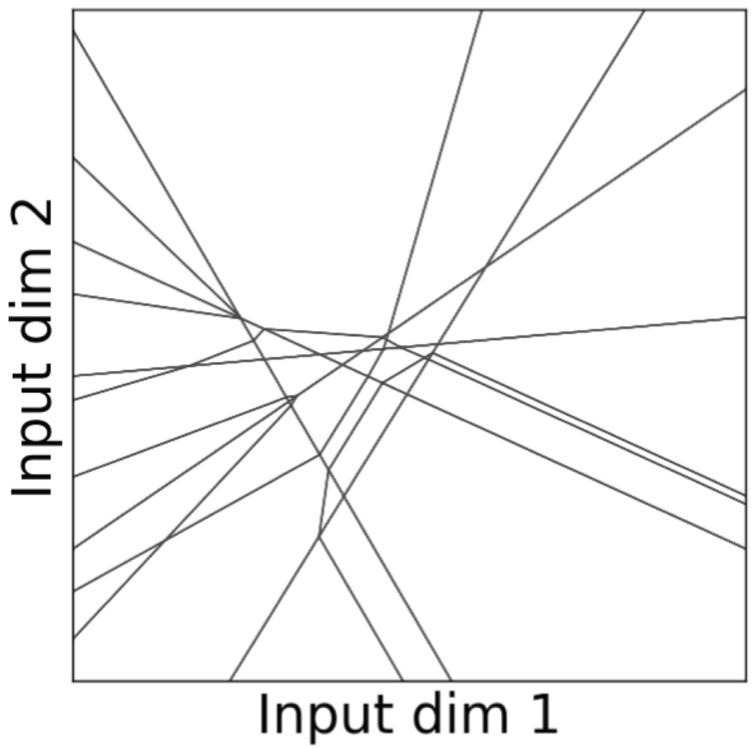


(Hanin & Rolnick 2019)

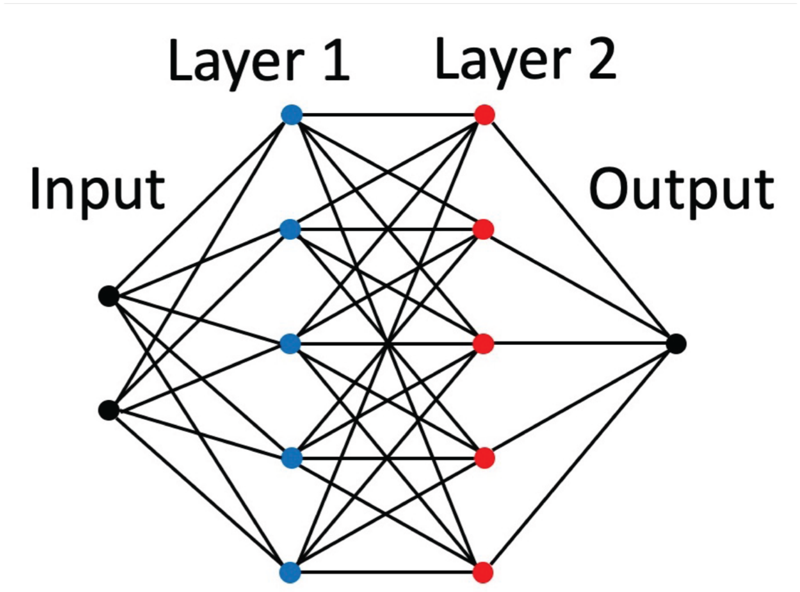
Boundaries of linear regions

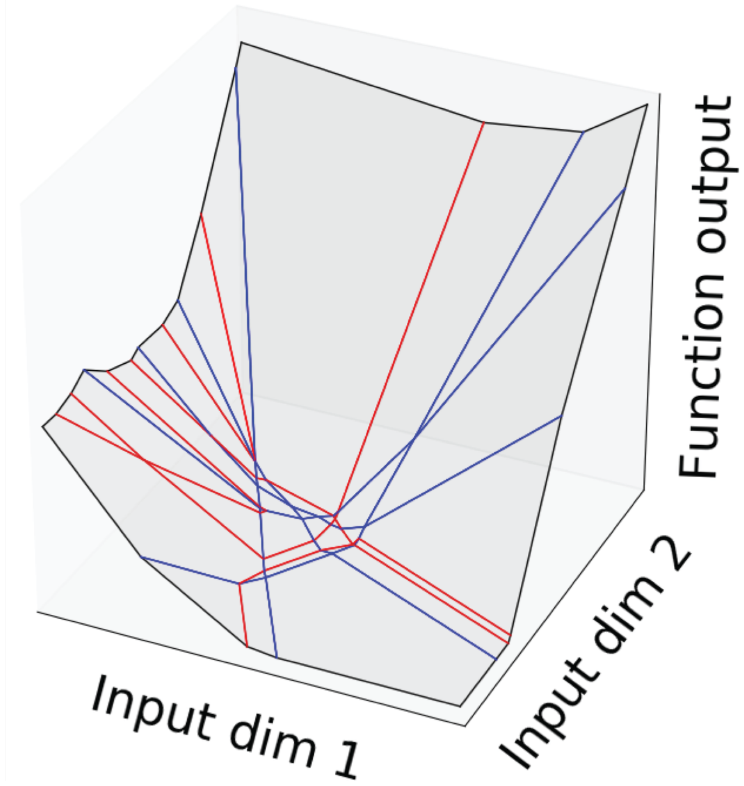


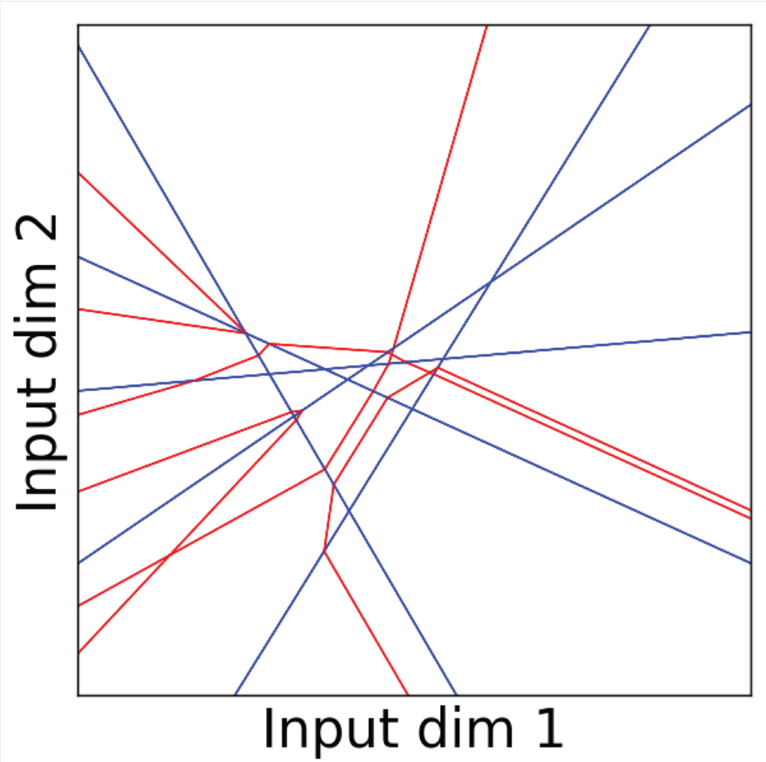




Boundaries of linear regions







Piecewise linear boundary component $\mathcal{B}_z$ for each neuron z
(Hanin & Rolnick 2019)

Main theorem (informal)

For a fully connected ReLU network of any depth, suppose that each boundary component $\mathcal{B}_z$ is connected and that $\mathcal{B}_z$ and $\mathcal{B}_{z'}$ intersect for each pair of adjacent neurons z and z'.

a) Given the set of linear region boundaries, it is possible to recover the complete structure and weights of the network, up to permutation and scaling, except for a measure-zero set of networks.

b) It is possible to approximate the set of linear region boundaries and thus the architecture/weights by querying the network.

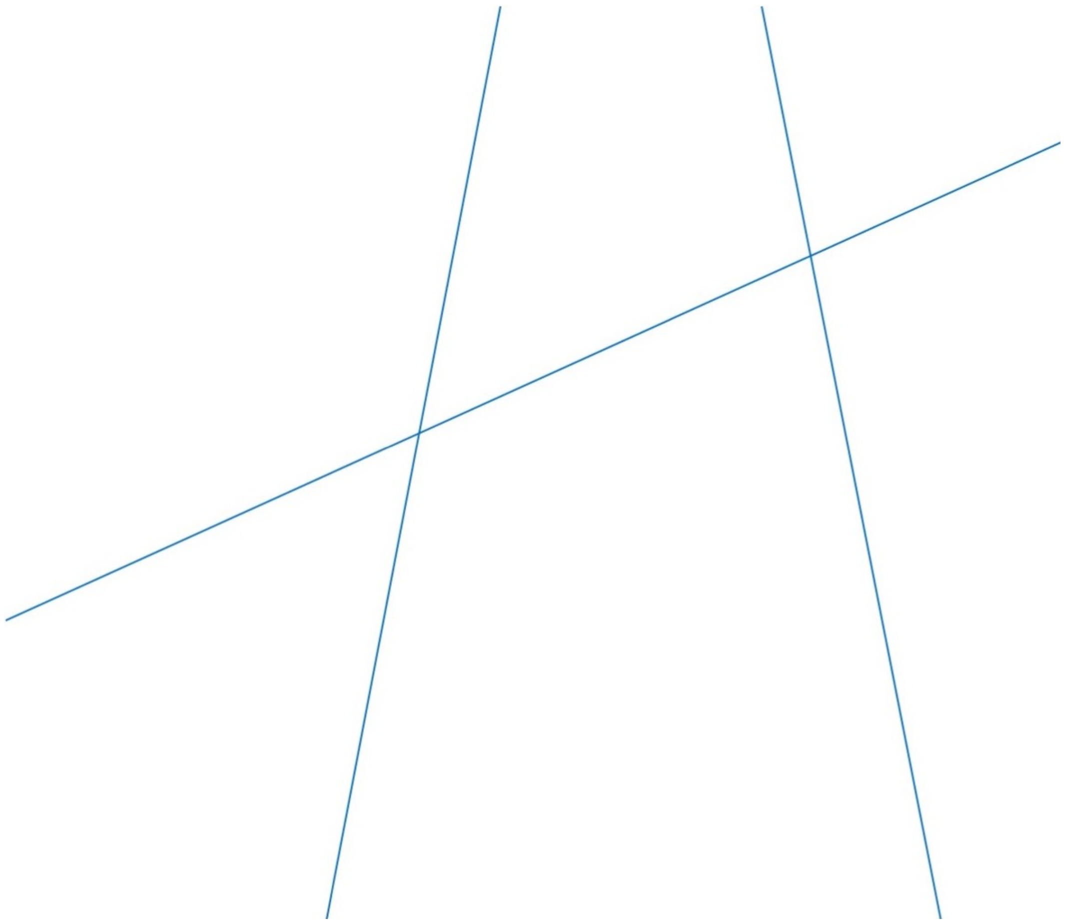
Main theorem (informal)

For a fully connected ReLU network of any depth, suppose that each boundary component $\mathcal{B}_z$ is connected and that $\mathcal{B}_z$ and $\mathcal{B}_{z'}$ intersect for each pair of adjacent neurons z and z'.

a) **Given the set of linear region boundaries, it is possible to recover the complete structure and weights of the network, up to permutation and scaling, except for a measure-zero set of networks.**

b) It is possible to approximate the set of linear region boundaries and thus the architecture/weights by querying the network.

Part (a), proof intuition

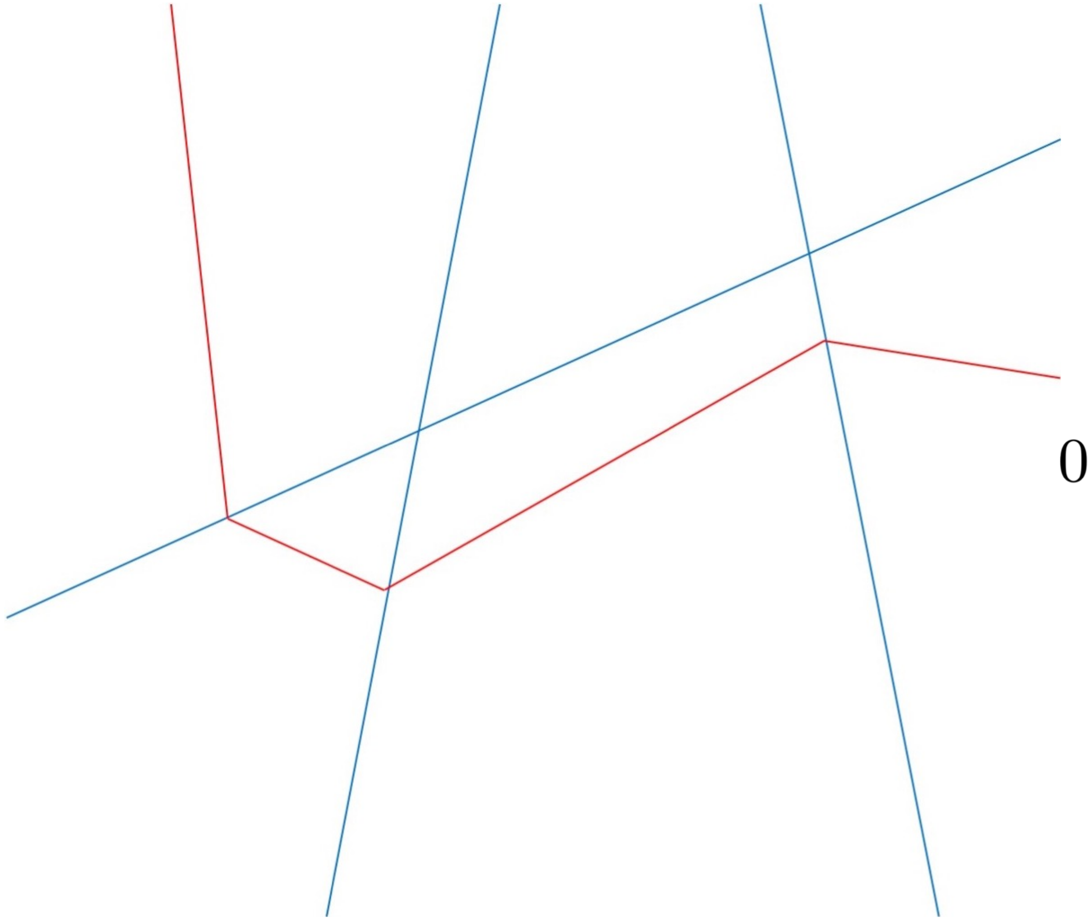


Neuron z in Layer 1

$$0 = z(x) = \sum_i w_i x_i + b$$

Part (a), proof intuition

<u>Neuron z' in Layer 2</u>



$$0 = z'(x) = \sum_z w_z \mathrm{ReLU}(z(x)) + b$$

Main theorem (informal)

For a fully connected ReLU network of any depth, suppose that each boundary component $\mathcal{B}_z$ is connected and that $\mathcal{B}_z$ and $\mathcal{B}_{z'}$ intersect for each pair of adjacent neurons z and z'.

a) Given the set of linear region boundaries, it is possible to recover the complete structure and weights of the network, up to permutation and scaling, except for a measure-zero set of networks.

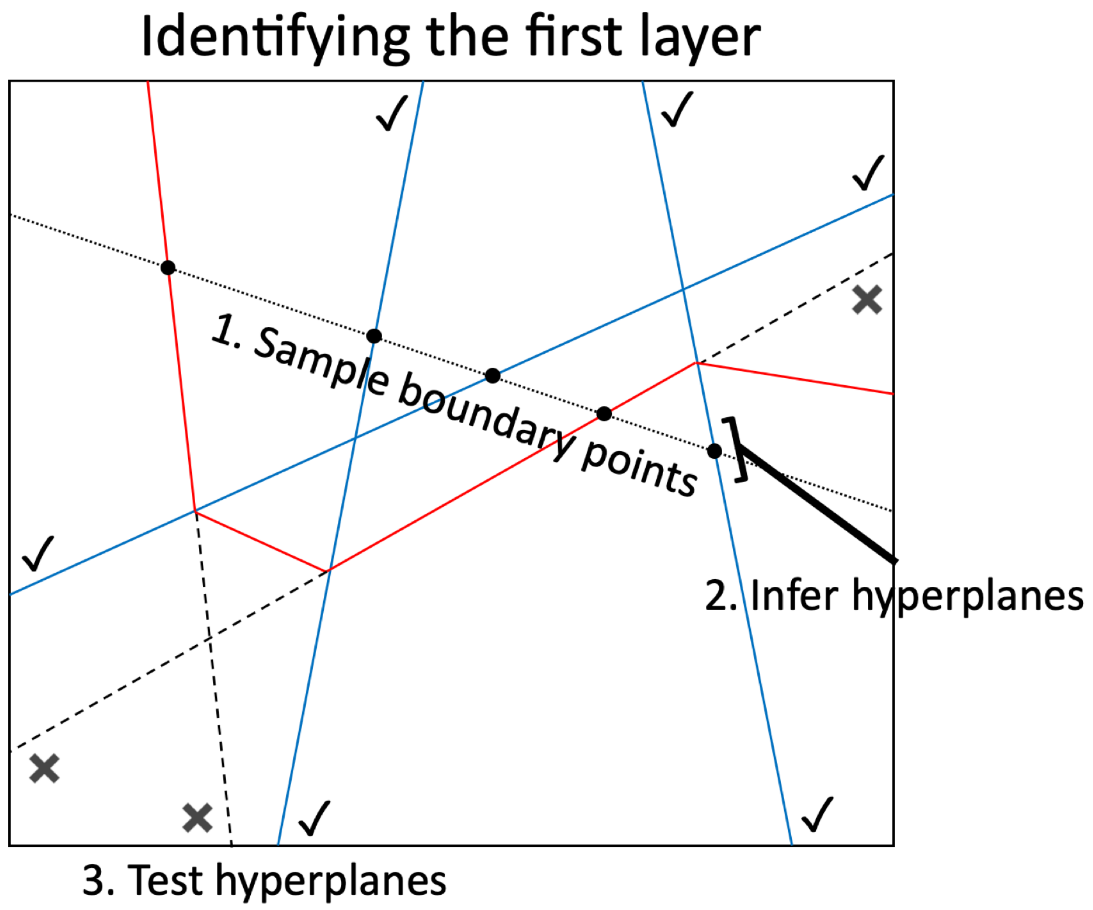
b) It is possible to approximate the set of linear region boundaries and thus the architecture/weights by querying the network.

Part (b): reconstructing Layer 1

Goal: Approximate boundaries by querying network adaptively

Approach: Identify points on the boundary by binary search using $\nabla\mathcal{N}$

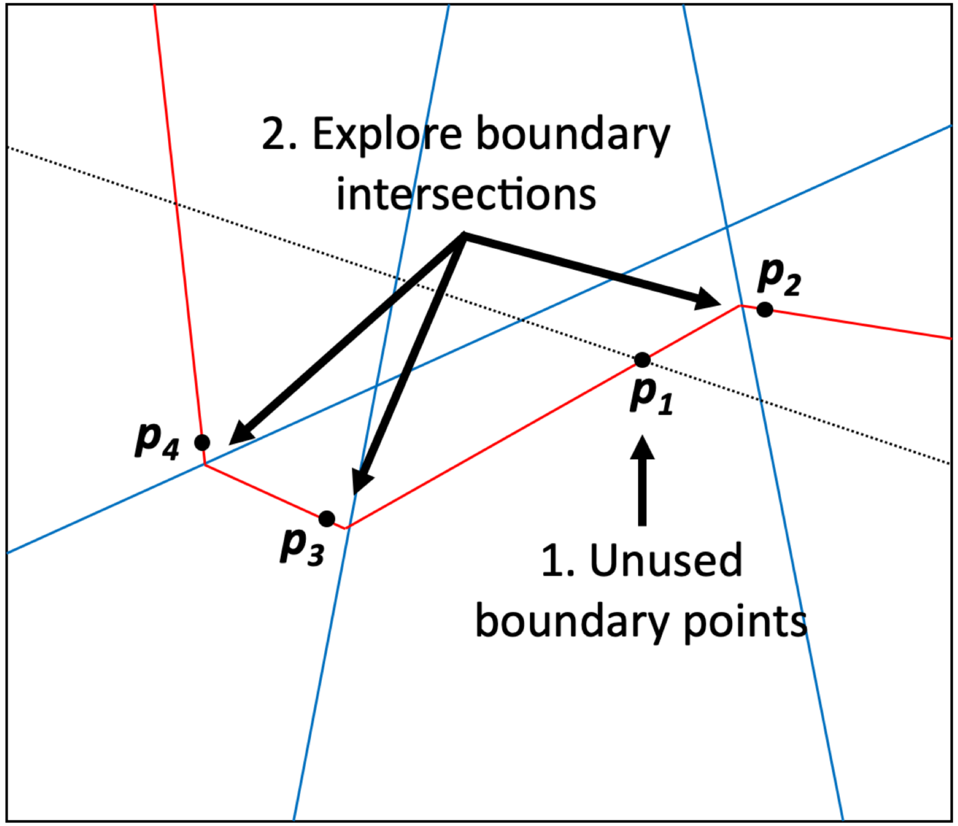
1) Find boundary points along a line
2) Each belongs to some $\mathcal{B}_z$, identify the local hyperplane by regression
3) Test whether $\mathcal{B}_z$ is a hyperplane



Part (b): reconstructing Layers ≥ 2

1) Start with unused boundary points identified in previous algorithm
2) Explore how $\mathcal{B}_z$ bends as it intersects $\mathcal{B}_{z'}$ already identified

Additional layers



Why don’t we just…

…train on the output of the black-box network to recover it?

It doesn’t work.

…repeat our algorithm for Layer 1 to learn Layer 2?

Requires arbitrary inputs to Layer 2, but cannot invert Layer 1.

Assumptions of the algorithm

Boundary components are connected

$\Rightarrow$ generally holds unless input dimension small

Adjacent neurons have intersecting boundary components

$\Rightarrow$ failure can result from unavoidable ambiguities in network (beyond permutation and scaling)

Note: Algorithm “degrades gracefully”

- When assumptions don’t hold exactly, still recovers *most of the network*

More complex networks

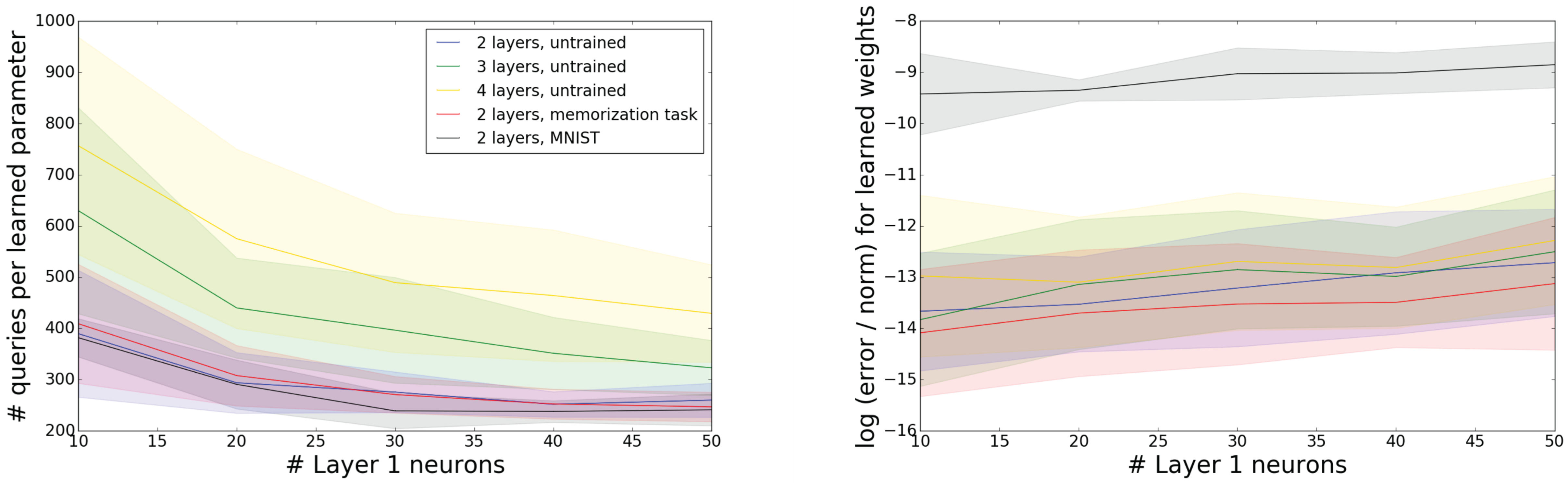
Convolutional layers

- Algorithm still works
- Doesn’t account for weight-sharing, so less efficient

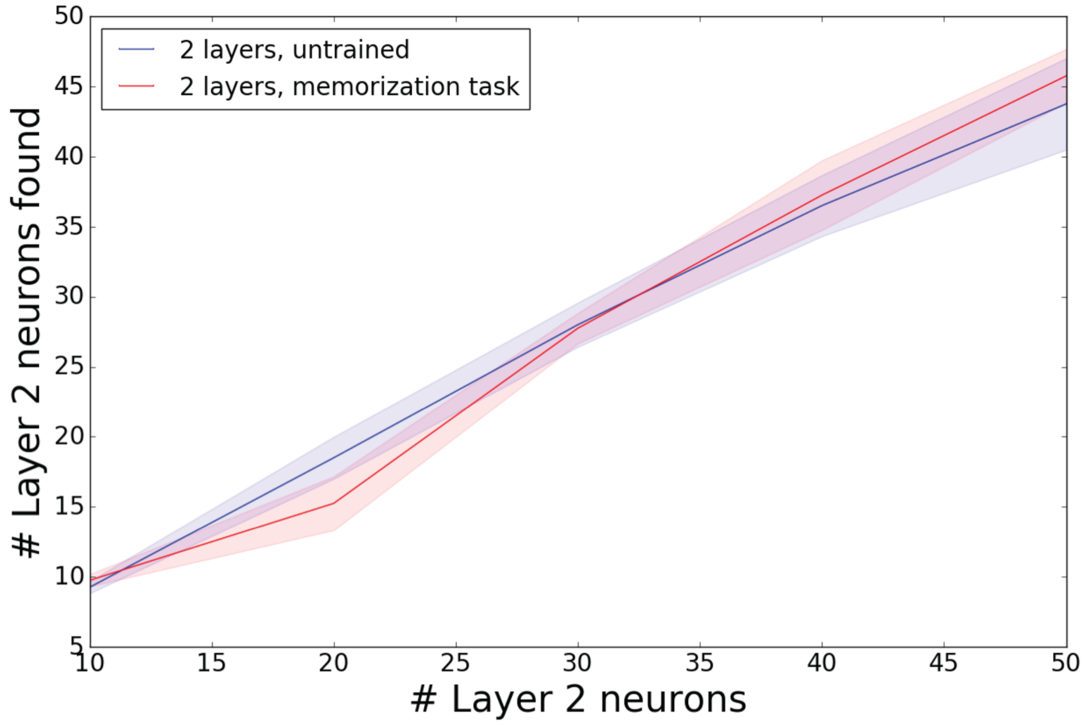
Skip connections

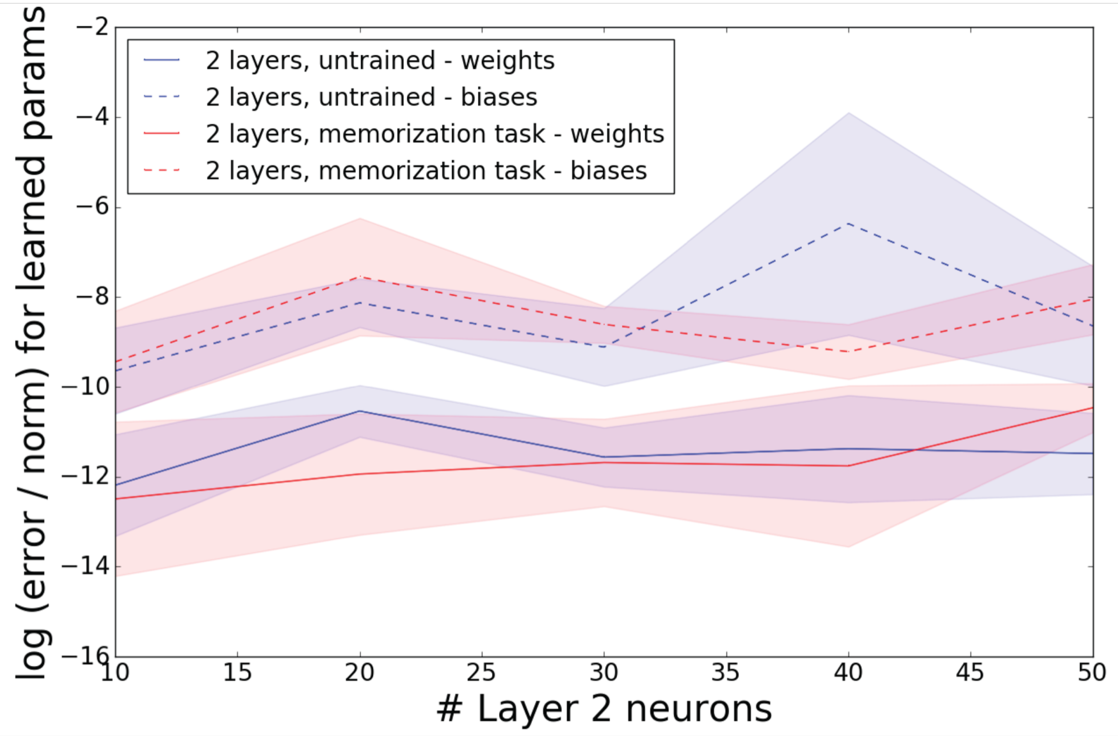
- Algorithm works with modification
- Need to consider intersections between more pairs of boundary components

Experimental results – Layer 1 algorithm



Experimental results – Layer ≥ 2 algorithm





Summary

- **Prove:** Can recover architecture, weights, & biases of deep ReLU networks from linear region boundaries (under natural assumptions).
- **Implement:** Algorithm for recovering full network from black-box access by approximating these boundaries.
- **Demonstrate:** Success of our algorithm at reverse-engineering networks in practice.