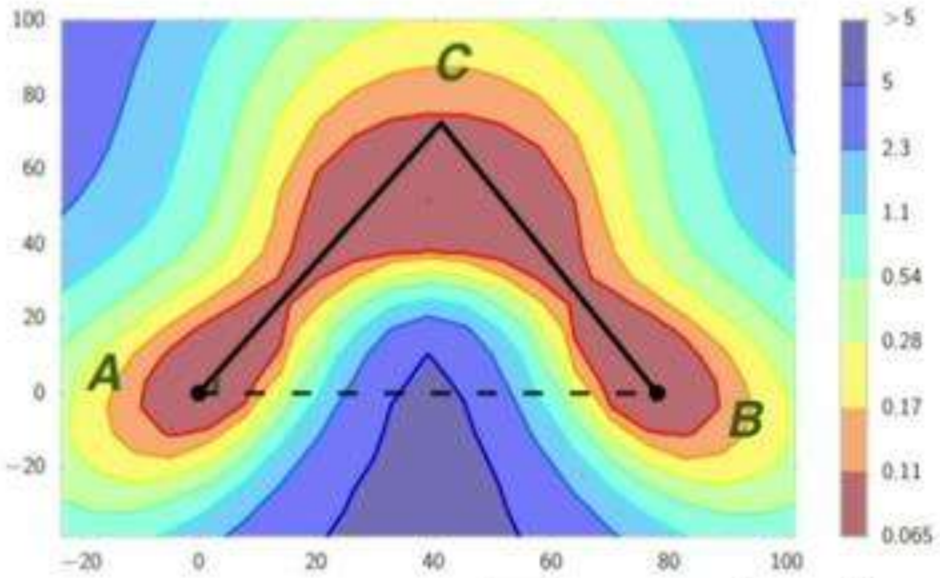
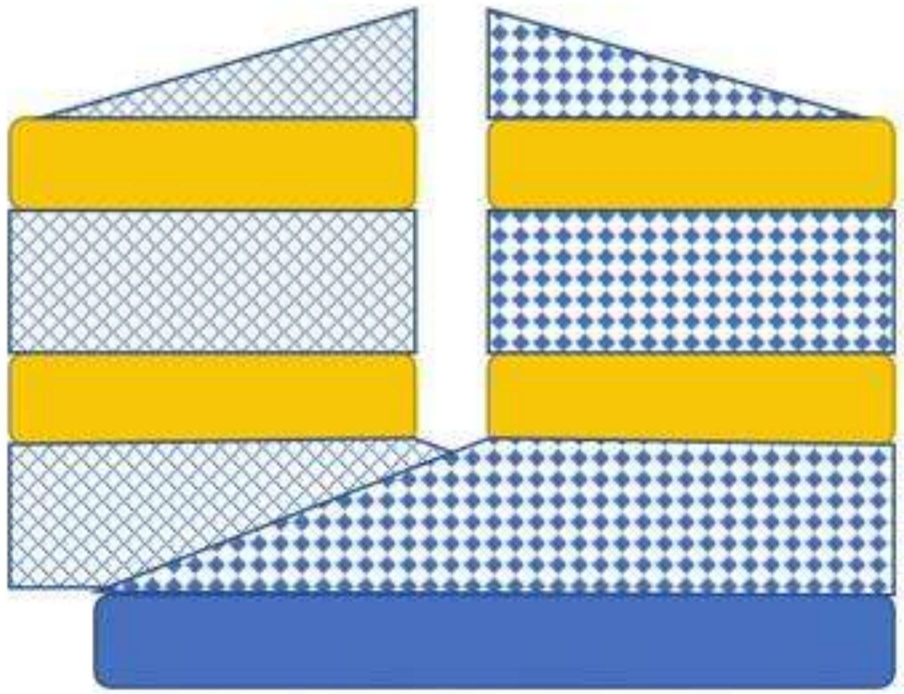


Explaining Landscape Connectivity of Low-cost Solutions for Multilayer Nets

Geometry of Deep Learning Workshop

Rong Ge, Duke University

Joint work with Rohith Kuditipudi, Xiang Wang (Duke)
Holden Lee, Yi Zhang, Zhiyuan Li, Wei Hu, Sanjeev Arora (Princeton)



Mode Connectivity [Freeman and Bruna 16, Garipov et al. 18, Draxler et al. 18]

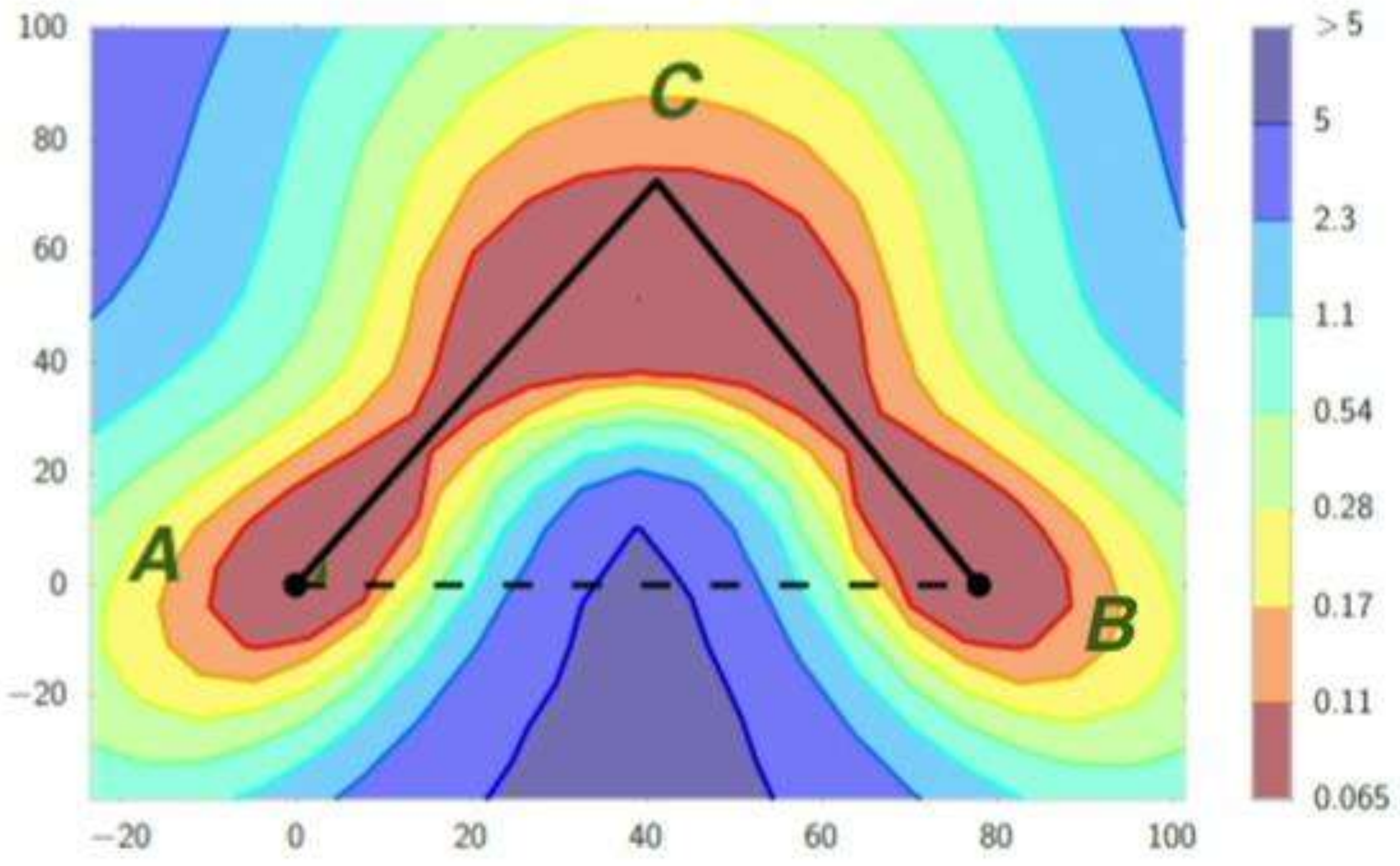


Image from [Garipov et al. 18]

Mode Connectivity[Freeman and Bruna 16, Garipov et al. 18, Draxler et al. 18]

- For neural networks, local minima found via gradient descent are connected by simple paths in the parameter space

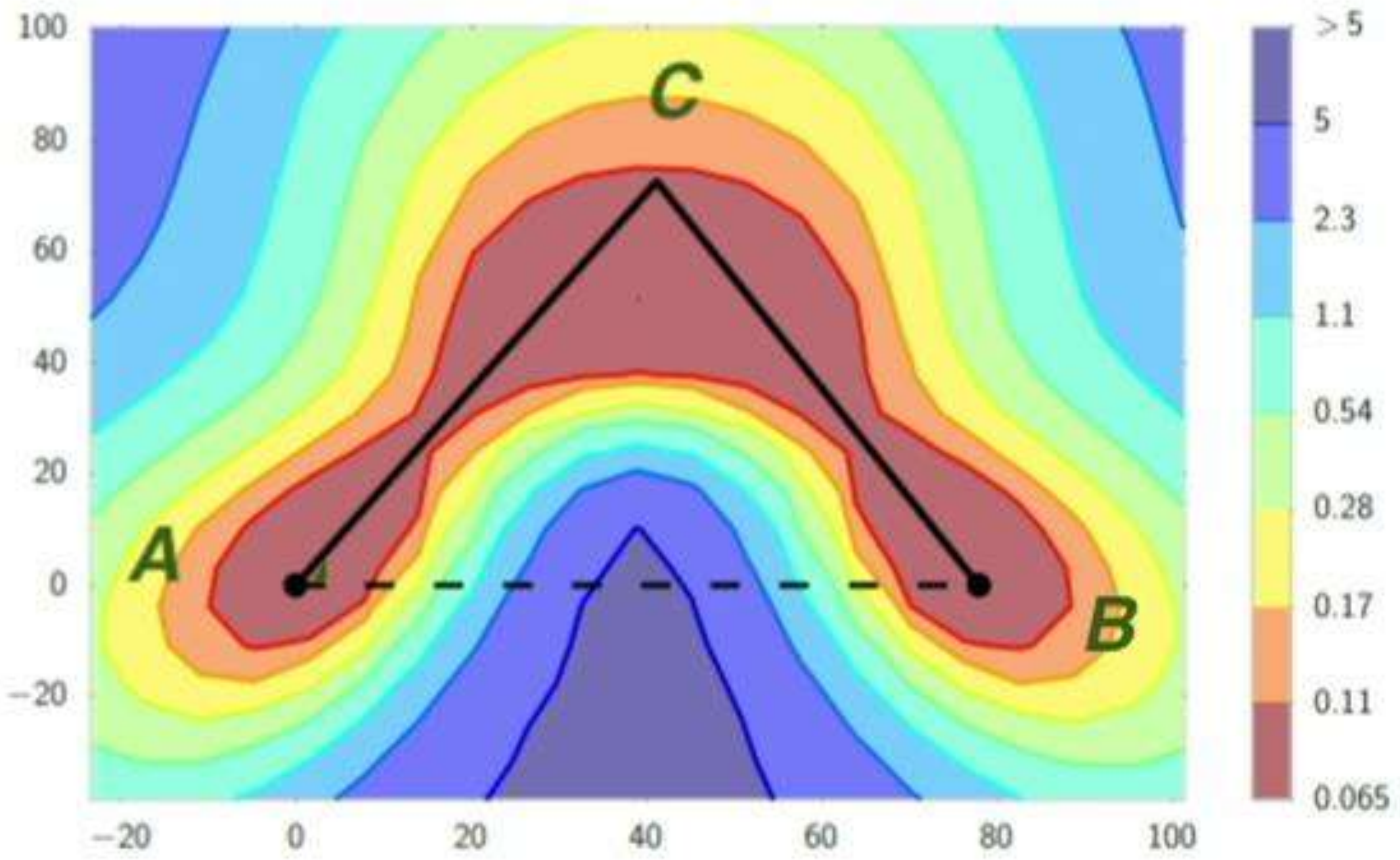


Image from [Garipov et al. 18]

Mode Connectivity[Freeman and Bruna 16, Garipov et al. 18, Draxler et al. 18]

- For neural networks, local minima found via gradient descent are connected by simple paths in the parameter space
- Every point on the path is another solution of almost the same cost.

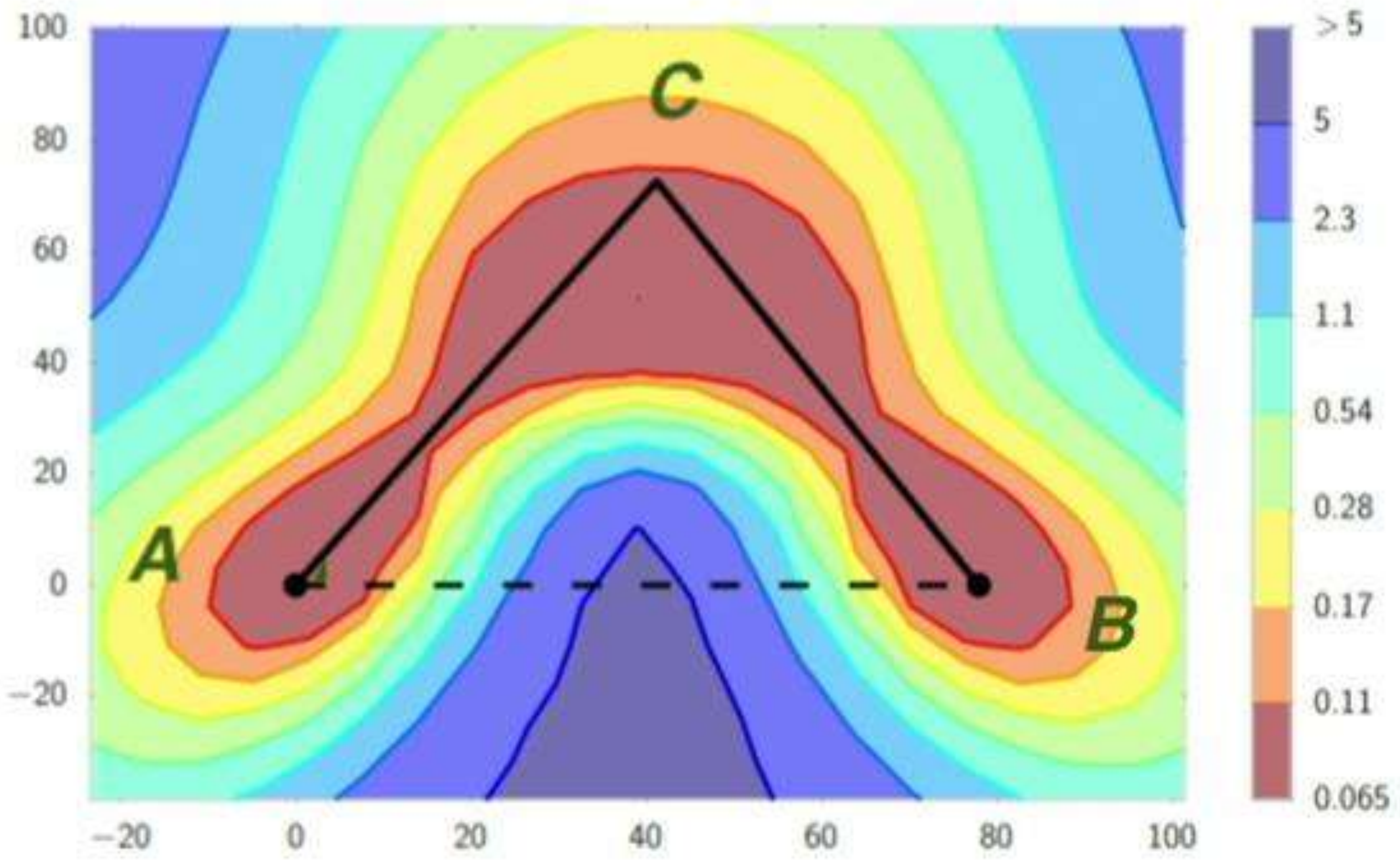
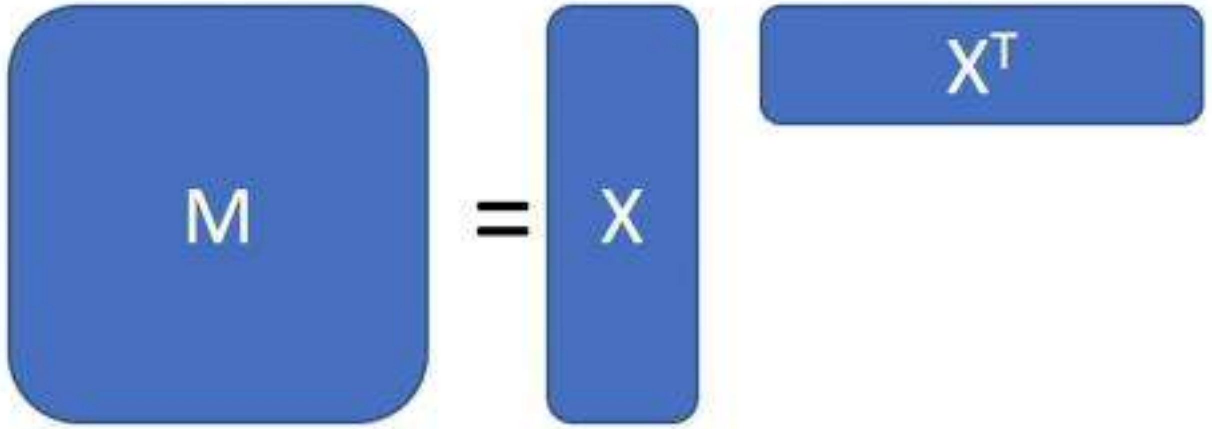


Image from [Garipov et al. 18]

Equivalent local minima and symmetry

Equivalent local minima and symmetry

- Matrix Problems
- Goal: find low rank matrix M

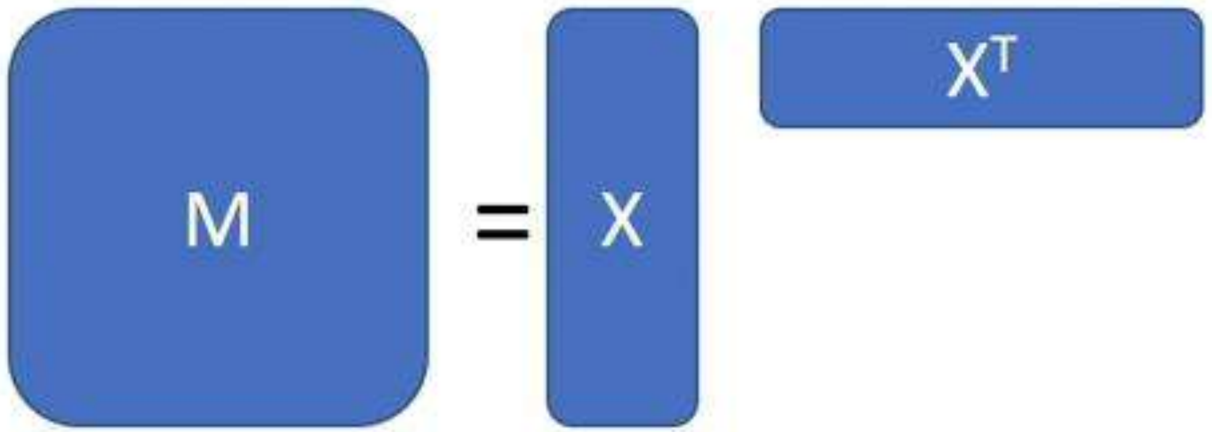


- Equivalent solutions:

$$X = X^* R, RR^\top = I$$

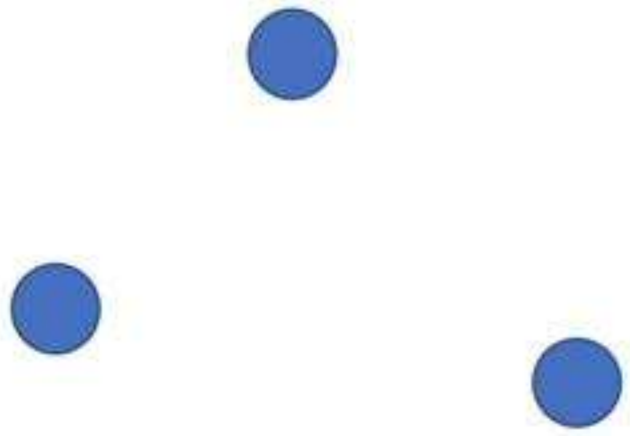
Equivalent local minima and symmetry

- Matrix Problems
- Goal: find low rank matrix M



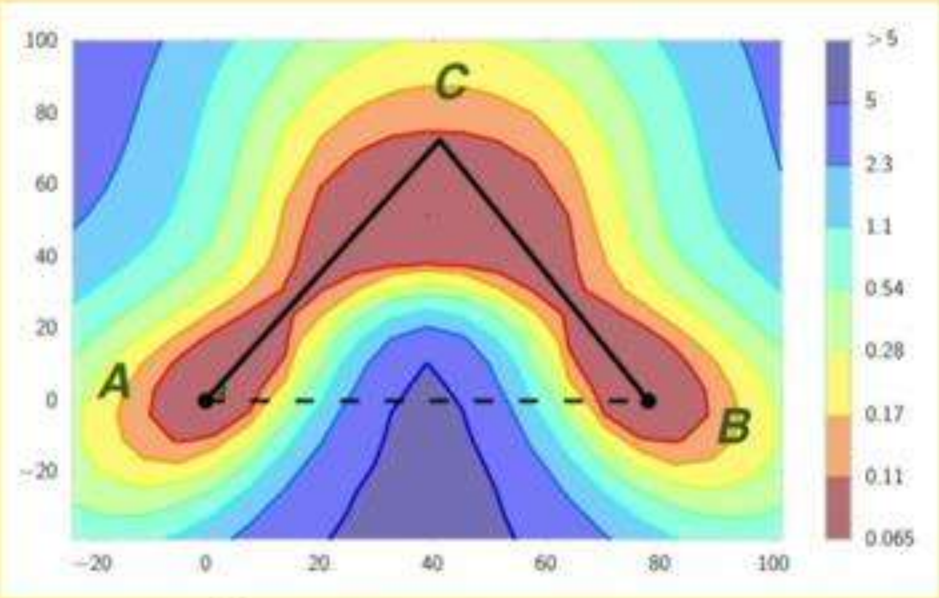
- Equivalent solutions:
  $$X = X^*R, RR^\top = I$$

- "Tensor" Problems
- Goal: find k components



- Equivalent solutions:
  $$X = X^*P, P\ permutation$$

Equivalent local minima and symmetry

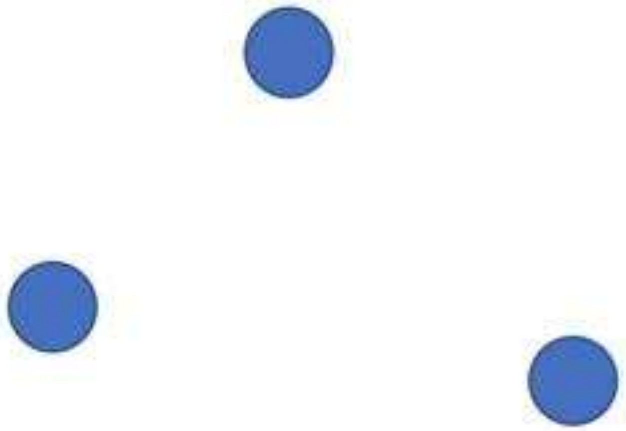


(mostly) connected local min

- Equivalent solutions:

$$X = X^* R, RR^\top = I$$

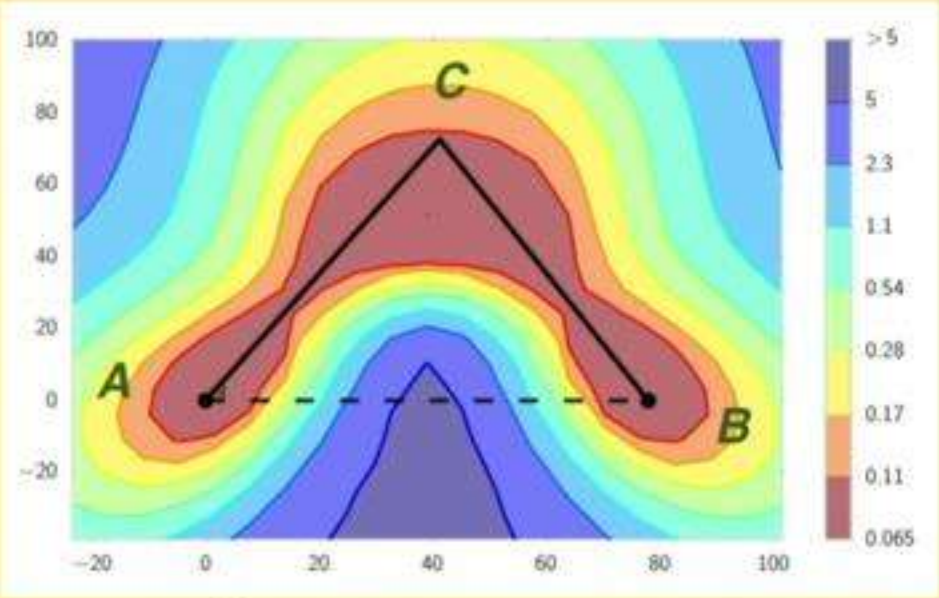
- "Tensor" Problems
- Goal: find k components



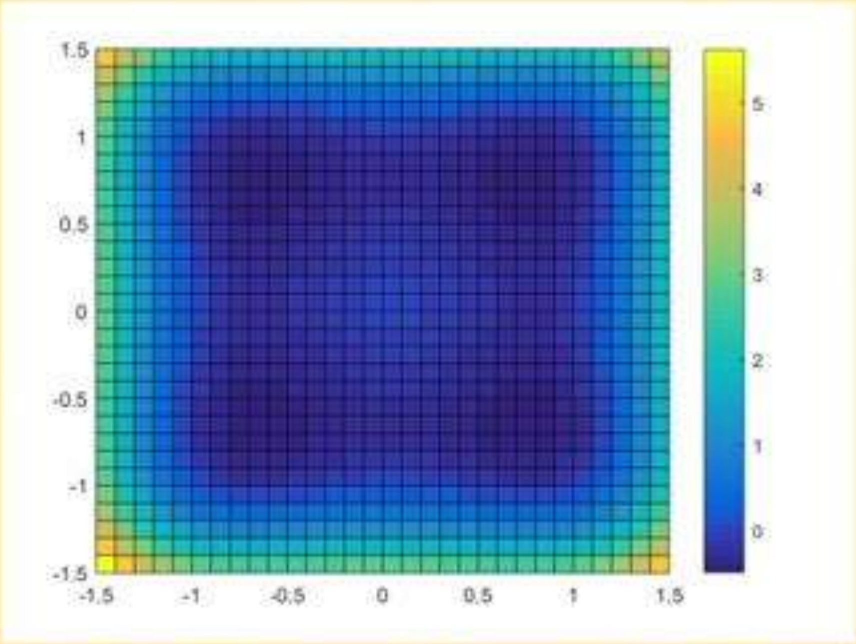
- Equivalent solutions:

$$X = X^* P, P\ permutation$$

Equivalent local minima and symmetry



(mostly) connected local min

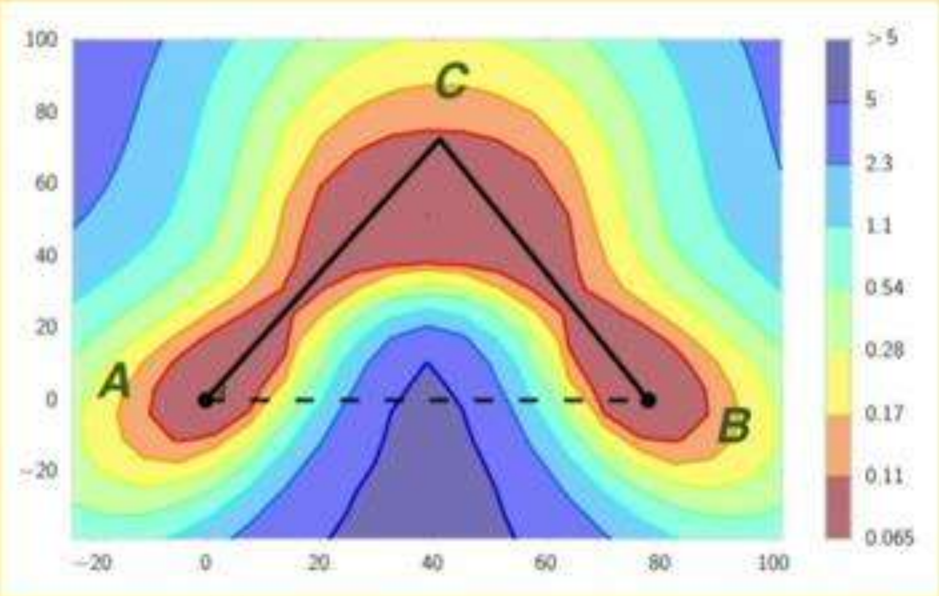


Isolated local min

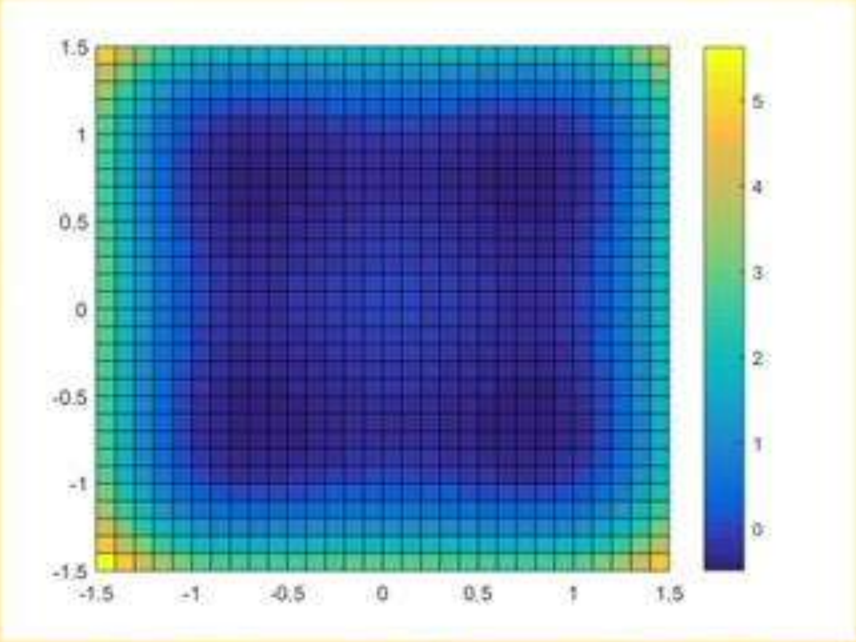
- Equivalent solutions:
  $$X = X^* R, RR^\top = I$$

- Equivalent solutions:
  $$X = X^* P, P\ permutation$$

Equivalent local minima and symmetry



(mostly) connected local min



Isolated local min

- Equivalent solutions:
  $$X = X^* R, RR^\top = I$$

- Equivalent solutions:
  $$X = X^* P, P\ permutation$$

Neural networks only have permutation symmetry,
why do they have connected local min?

(Partial) short answer: overparametrization

(Partial) short answer: overparametrization

- Existing explanations of mode connectivity: [Freeman and Bruna, 2016, Venturi et al. 2018, Liang et al. 2018, Nguyen et al. 2018, Nguyen et al. 2019]

(Partial) short answer: overparametrization

- Existing explanations of mode connectivity: [Freeman and Bruna, 2016, Venturi et al. 2018, Liang et al. 2018, Nguyen et al. 2018, Nguyen et al. 2019]
- If the network has special structure, and is highly overparametrized (#neurons > #training samples), then local mins are connected.

(Partial) short answer: overparametrization

- Existing explanations of mode connectivity: [Freeman and Bruna, 2016, Venturi et al. 2018, Liang et al. 2018, Nguyen et al. 2018, Nguyen et al. 2019]
- If the network has special structure, and is highly overparametrized (#neurons > #training samples), then local mins are connected.
- Problem: Networks that are not as overparametrized were also found to have connected local min.

(Partial) short answer: overparametrization

- Existing explanations of mode connectivity:
  [Freeman and Bruna, 2016, Venturi et al. 2018, Liang et al. 2018, Nguyen et al. 2018, Nguyen et al. 2019]
- If the network has special structure, and is highly overparametrized (#neurons > #training samples), then local mins are connected.

- Problem: Networks that are not as overparametrized were also found to have connected local min.
- Can we prove similar results in mildly overparametrized regime?

Our Results

Our Results

- For neural networks, not all local/global min are connected, even in the overparametrized setting.

Our Results

- For neural networks, not all local/global min are connected, even in the overparametrized setting.

- Solutions that satisfy dropout stability are connected.

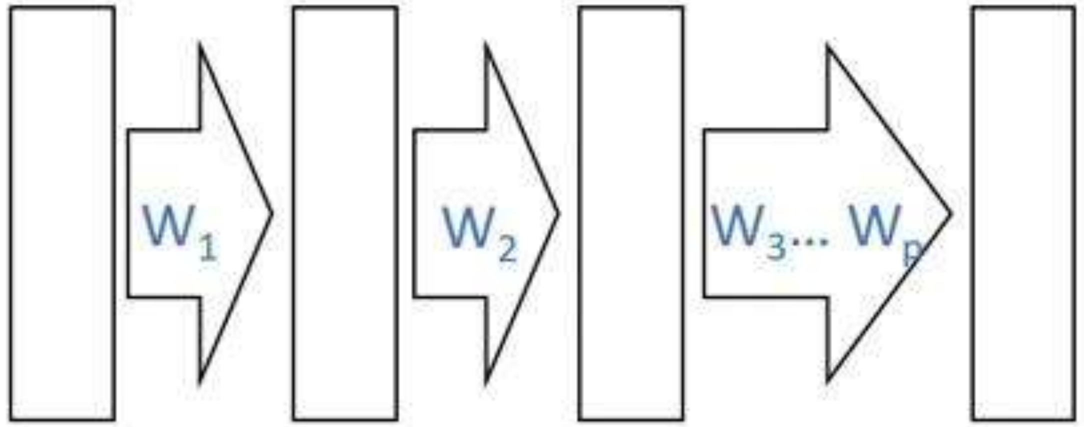
Our Results

- For neural networks, not all local/global min are connected, even in the overparametrized setting.

- Solutions that satisfy dropout stability are connected.

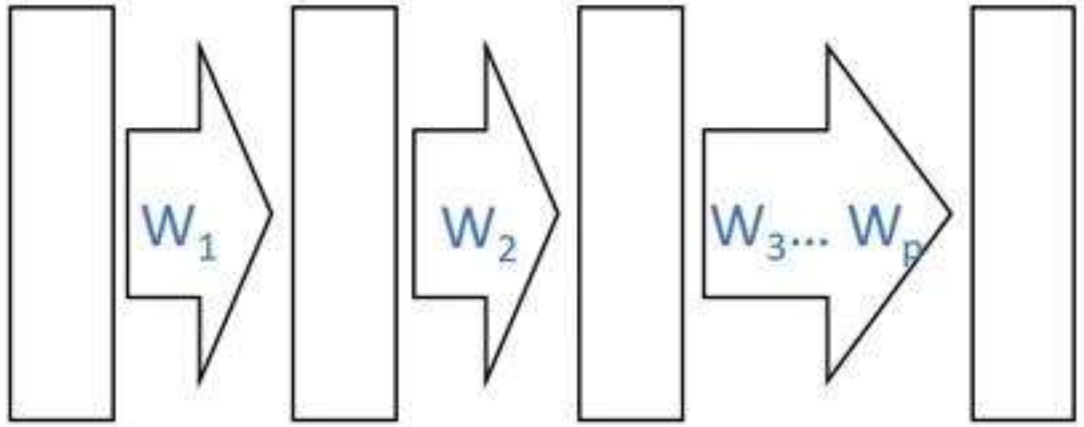
- Possible to switch dropout stability with noise stability (used for proving generalization bounds for neural nets)

Deep Neural Networks



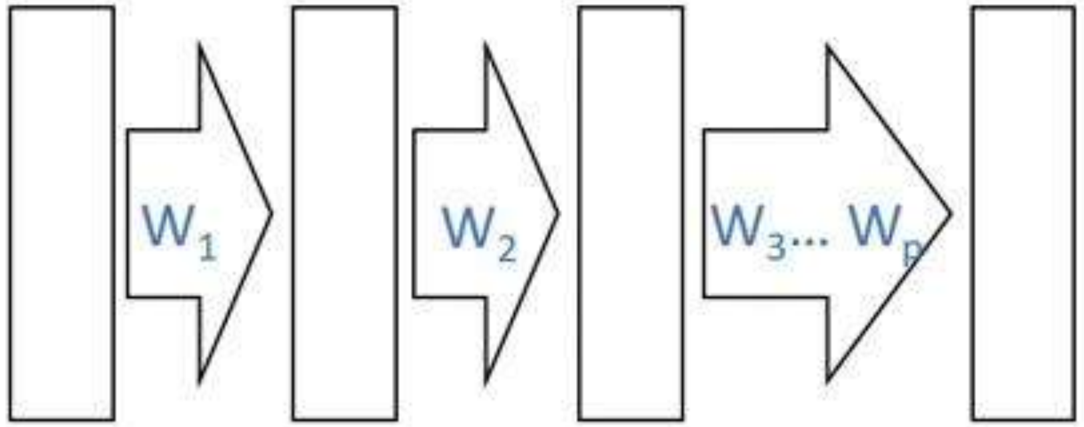
Deep Neural Networks

- For simplicity: Fully Connected Networks



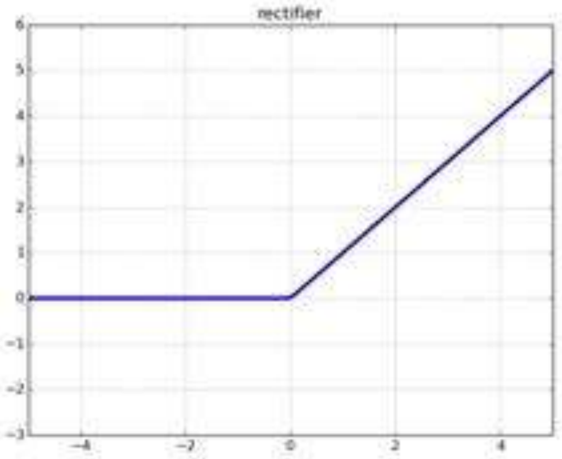
Deep Neural Networks

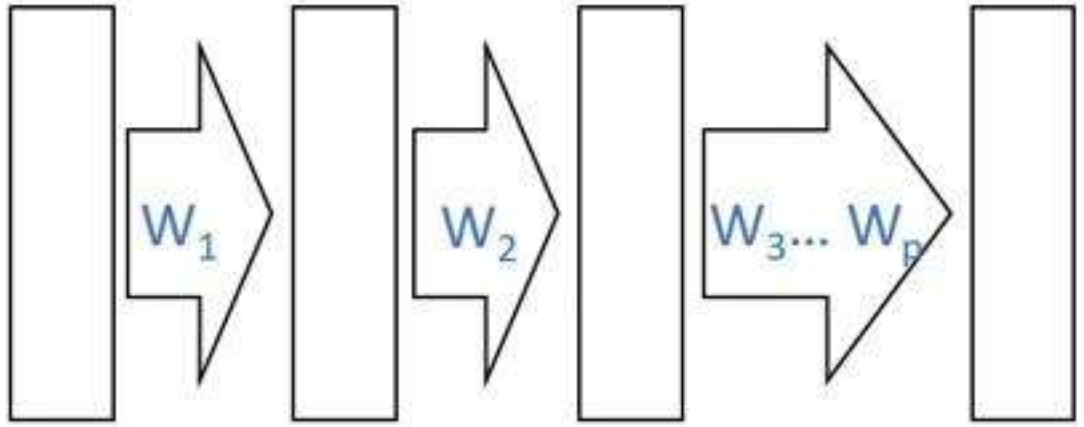
- For simplicity: Fully Connected Networks
- Weights $\theta = (W_1, W_2, \ldots, W_p)$, nonlinearity σ



Deep Neural Networks

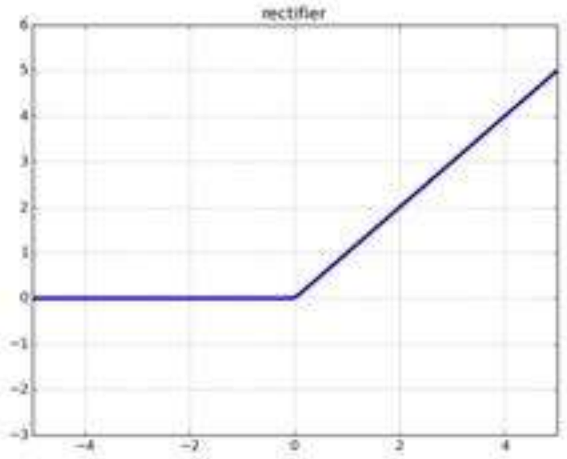
- For simplicity: Fully Connected Networks
- Weights $\theta = (W_1, W_2, \ldots, W_p)$, nonlinearity σ

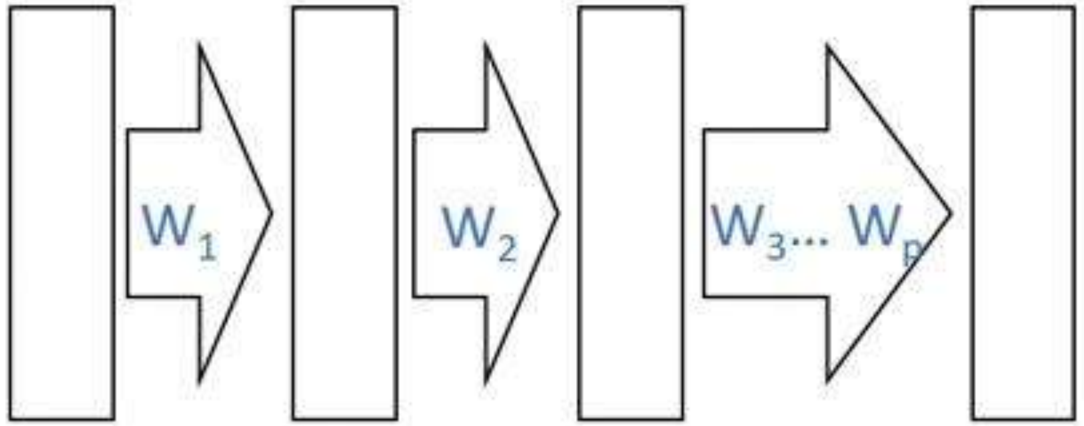




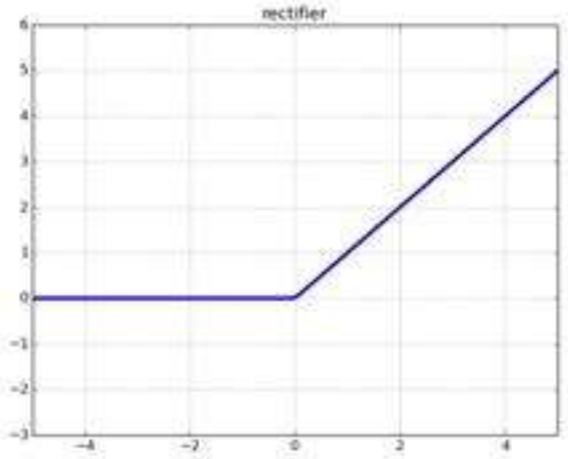
Deep Neural Networks

- For simplicity: Fully Connected Networks
- Weights $\theta = (W_1, W_2, \ldots, W_p)$, nonlinearity σ
- Samples (x,y), hope to learn a network that maps x to y

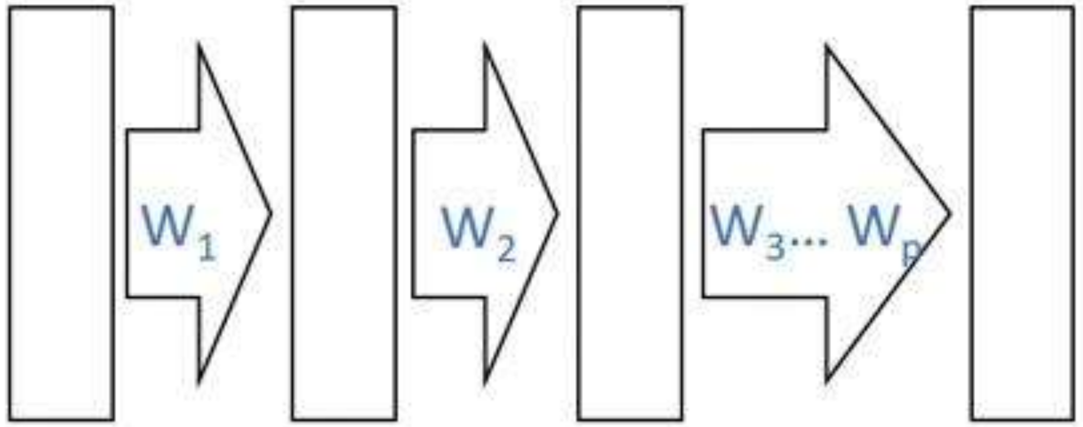




Deep Neural Networks



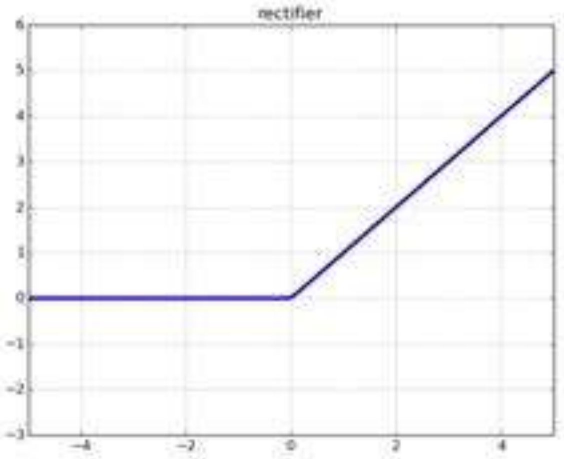
- For simplicity: Fully Connected Networks
- Weights $\theta = (W_1, W_2, \ldots, W_p)$, nonlinearity σ
- Samples (x,y), hope to learn a network that maps x to y



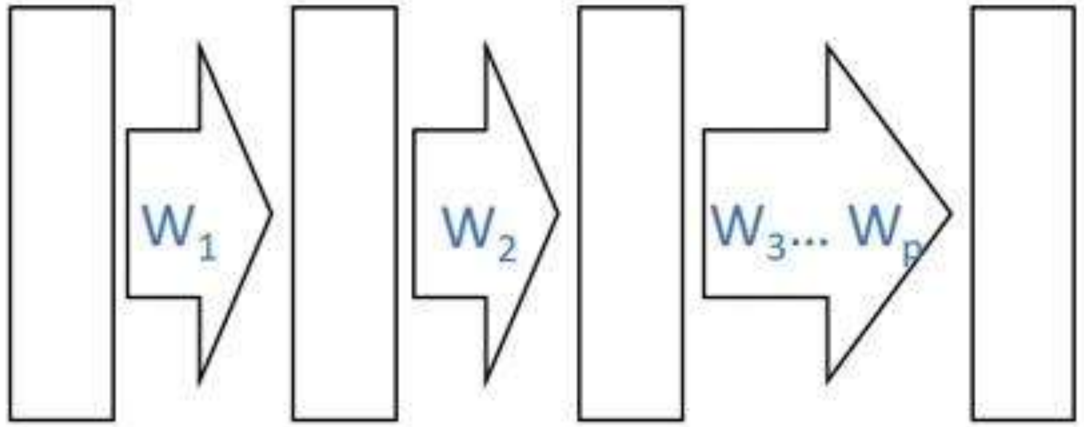
$x^{(0)} = x$ $\quad x^{(1)} = \sigma(W_1 x)$ $\qquad x^{(p)} = W_p x^{(p-1)}$

- Function $f_\theta(x) = W_p \sigma(W_{p-1}\sigma(\cdots\sigma(W_1 x)\cdots))$

Deep Neural Networks



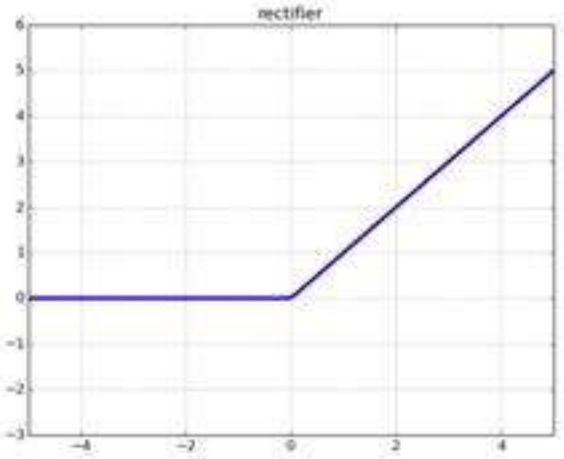
- For simplicity: Fully Connected Networks
- Weights $\theta = (W_1, W_2, \ldots, W_p)$, nonlinearity σ
- Samples (x,y), hope to learn a network that maps x to y



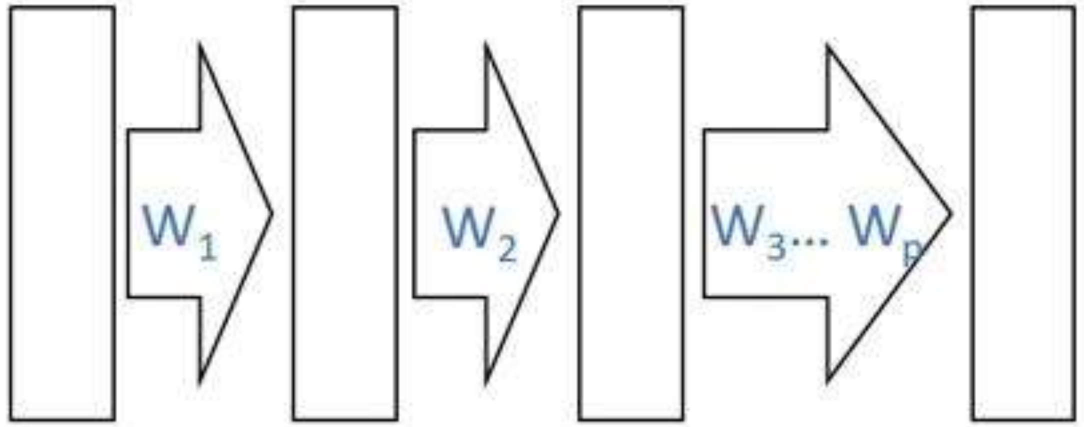
$x^{(0)} = x$ $\quad x^{(1)} = \sigma(W_1 x)$ $\qquad\qquad x^{(p)} = W_p x^{(p-1)}$

- Function $f_\theta(x) = W_p \sigma(W_{p-1} \sigma(\cdots \sigma(W_1 x) \cdots))$
- Objective: $L(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, f_\theta(x_i))$

Deep Neural Networks



- For simplicity: Fully Connected Networks
- Weights $\theta = (W_1, W_2, \ldots, W_p)$, nonlinearity σ
- Samples (x,y), hope to learn a network that maps x to y



$x^{(0)} = x$ $\quad x^{(1)} = \sigma(W_1 x)$ $\qquad x^{(p)} = W_p x^{(p-1)}$

- Function $f_\theta(x) = W_p \sigma(W_{p-1} \sigma(\cdots \sigma(W_1 x) \cdots))$
- Objective: $L(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, f_\theta(x_i))$

Convex loss function

Not all local min are connected

Not all local min are connected

- Simple setting: 2-layer net, data (x_i, y_i) generated by ground truth neural network with 2 hidden neurons.

Not all local min are connected

- Simple setting: 2-layer net, data (x_i, y_i) generated by ground truth neural network with 2 hidden neurons.
- Overparametrization: consider optimization of a 2 layer neural network with h (h >> 2) hidden neurons.

Not all local min are connected

- Simple setting: 2-layer net, data (x_i, y_i) generated by ground truth neural network with 2 hidden neurons.
- Overparametrization: consider optimization of a 2 layer neural network with h ($h >> 2$) hidden neurons.

- Theorem: For any $h > 2$, there exists a data-set with $h+2$ samples, such that the set of global minimizers are not connected.

What kind of local min are connected?

What kind of local min are connected?

- Only local min found by standard optimization algorithms are known to be connected.

What kind of local min are connected?

- Only local min found by standard optimization algorithms are known to be connected.
- Properties of such local min?
  - Closely connected to the question of generalization/implicit regularization.
  - Many conjectures: “flat” local min, margin, etc.

What kind of local min are connected?

- Only local min found by standard optimization algorithms are known to be connected.

- Properties of such local min?
  - Closely connected to the question of generalization/implicit regularization.
  - Many conjectures: “flat” local min, margin, etc.

- This talk: Dropout stability

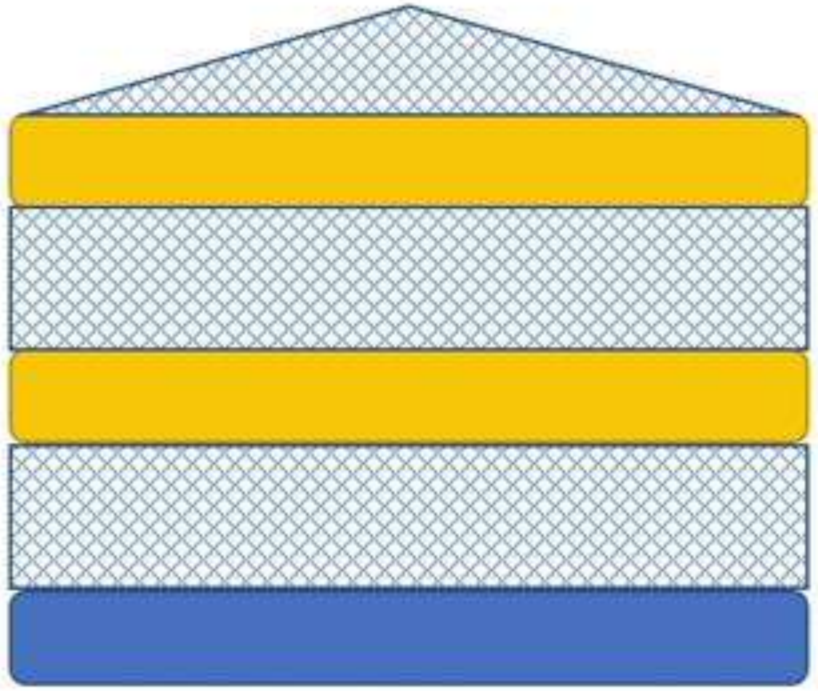
Dropout stability

Dropout stability

- A network is ε-dropout stable, if zeroing out 50% nodes at every layer (and rescale others appropriately) increases its loss by at most ε.

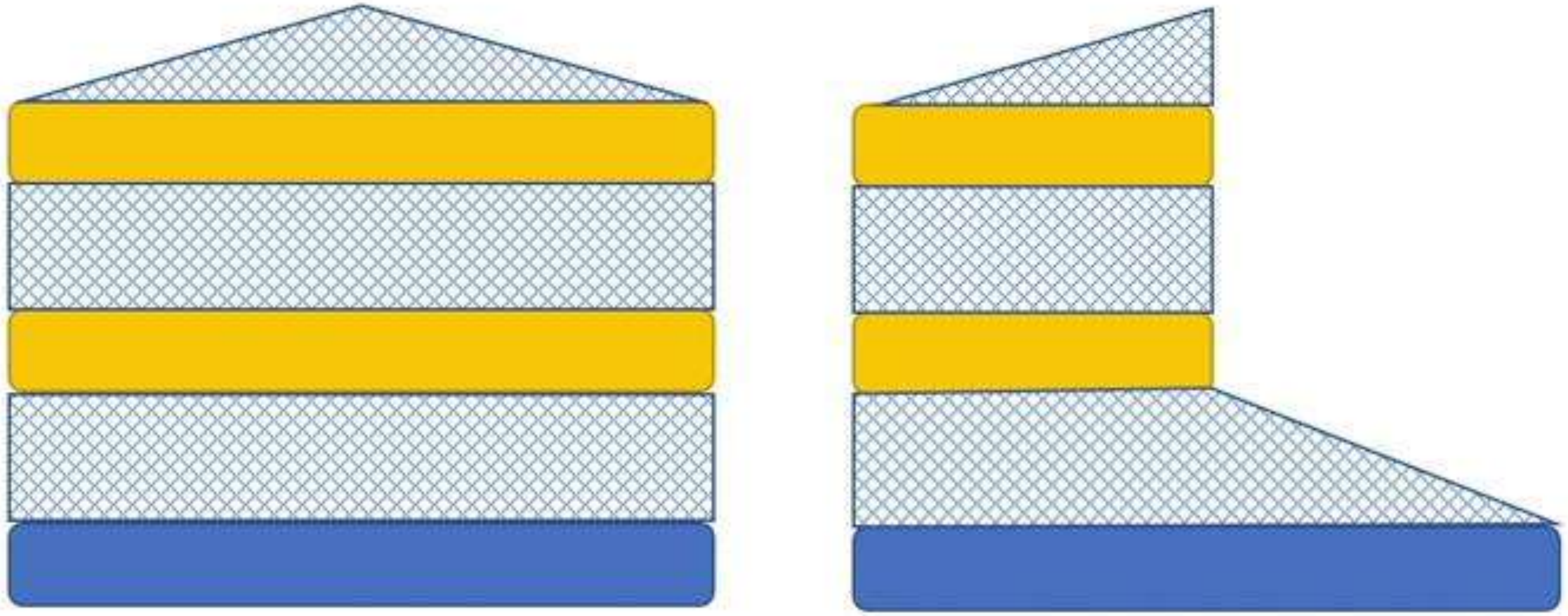
Dropout stability

- A network is ε-dropout stable, if zeroing out 50% nodes at every layer (and rescale others appropriately) increases its loss by at most ε.



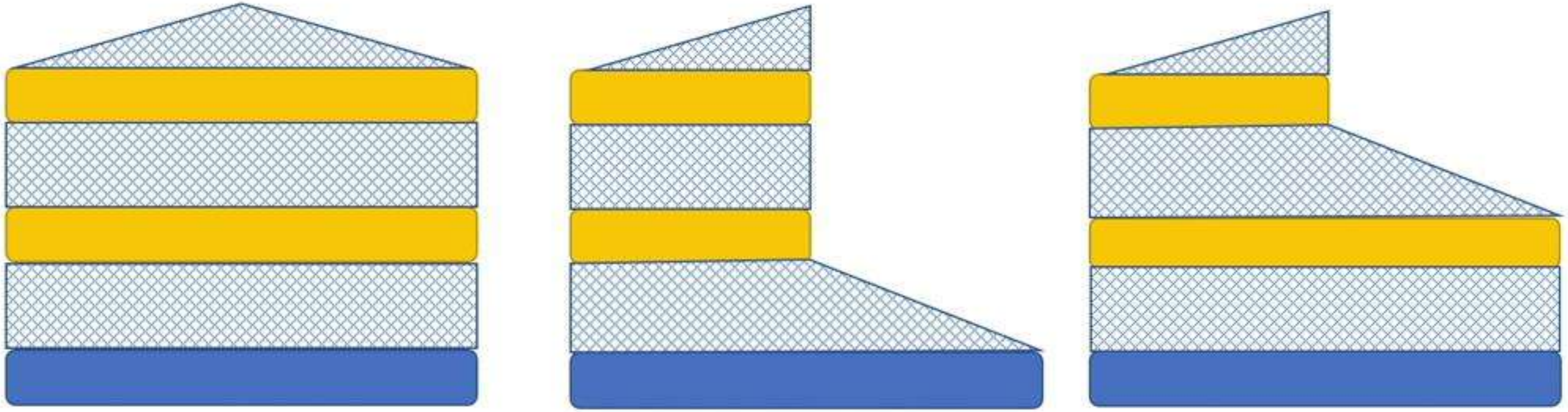
Dropout stability

- A network is ε-dropout stable, if zeroing out 50% nodes at every layer (and rescale others appropriately) increases its loss by at most ε.



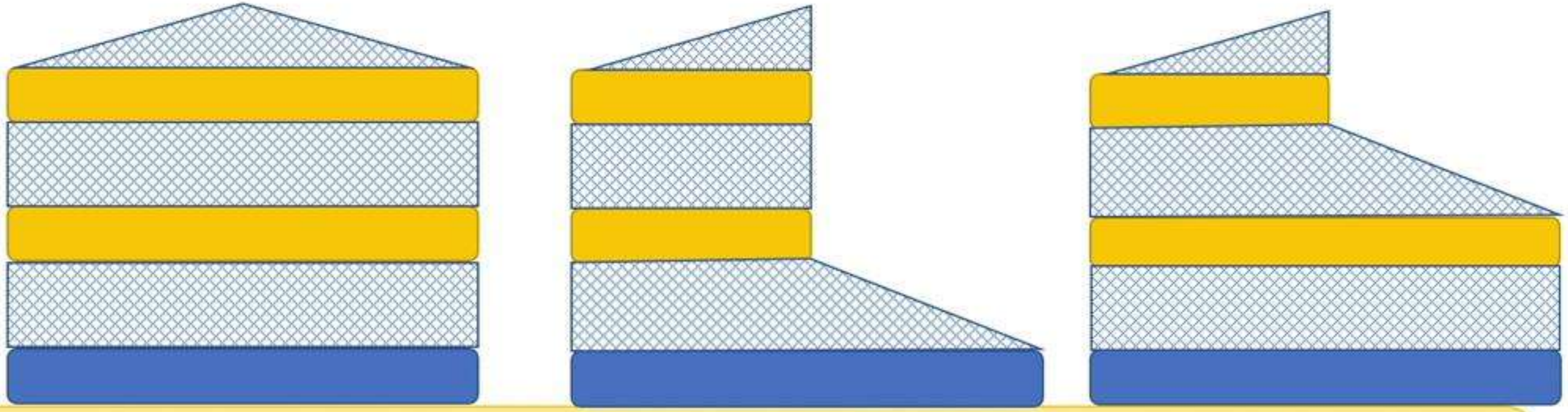
Dropout stability

- A network is ε-dropout stable, if zeroing out 50% nodes at every layer (and rescale others appropriately) increases its loss by at most ε.



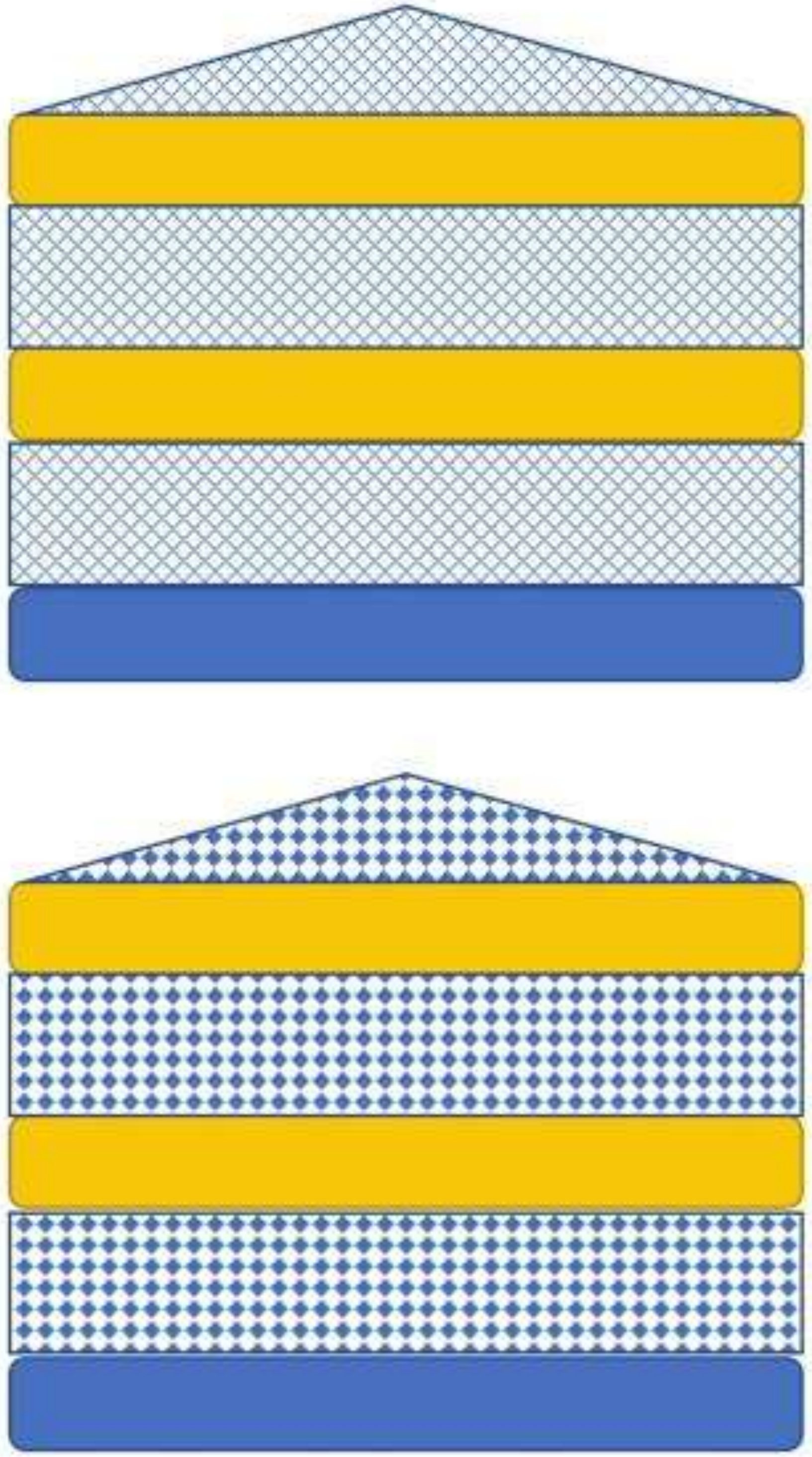
Dropout stability

- A network is ε-dropout stable, if zeroing out 50% nodes at every layer (and rescale others appropriately) increases its loss by at most ε.

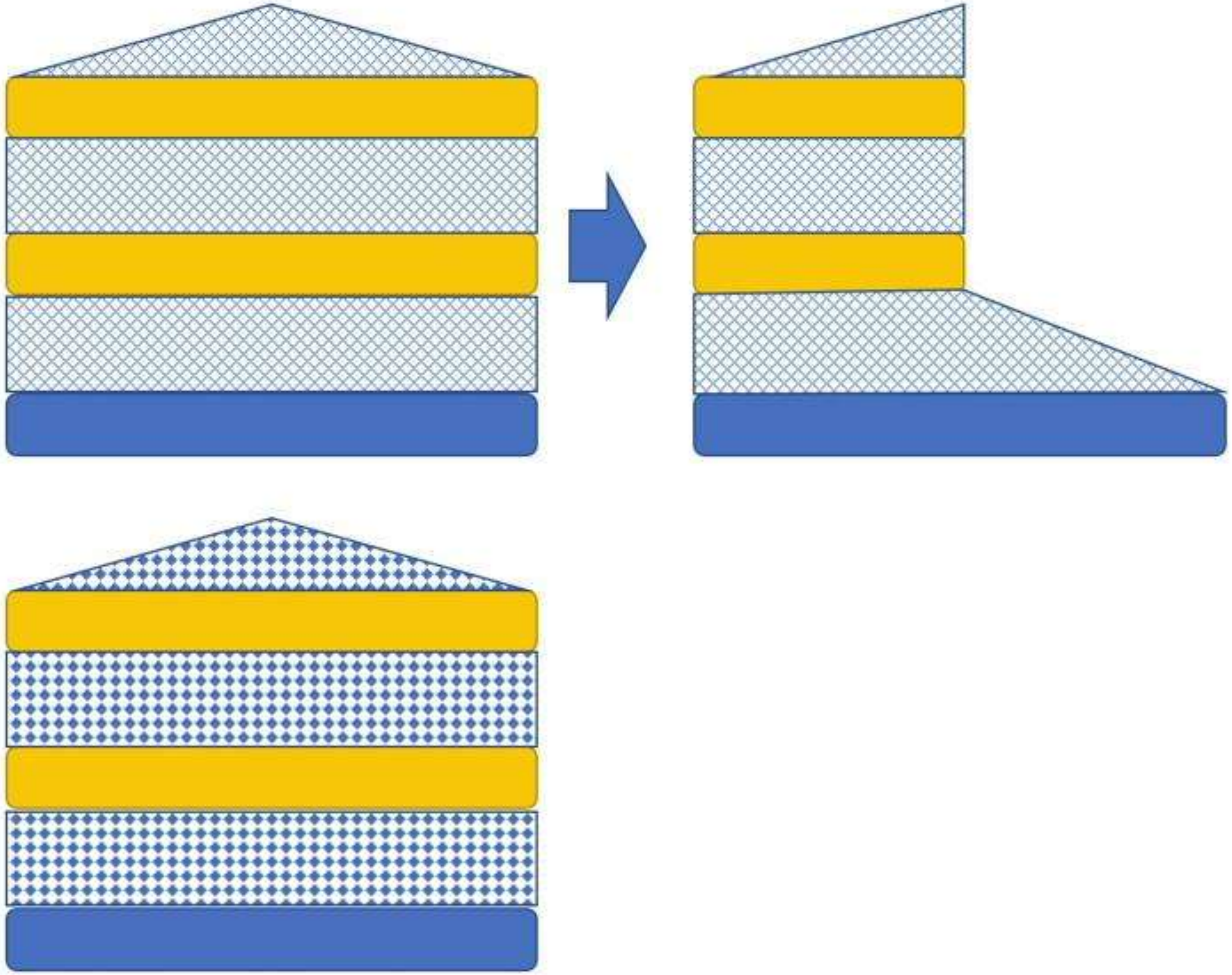


- Theorem: If both θ_A and θ_B are ε-dropout stable, then there exists a path between them with maximum loss $\leq \max\{L(\theta_A), L(\theta_B)\} + \varepsilon$

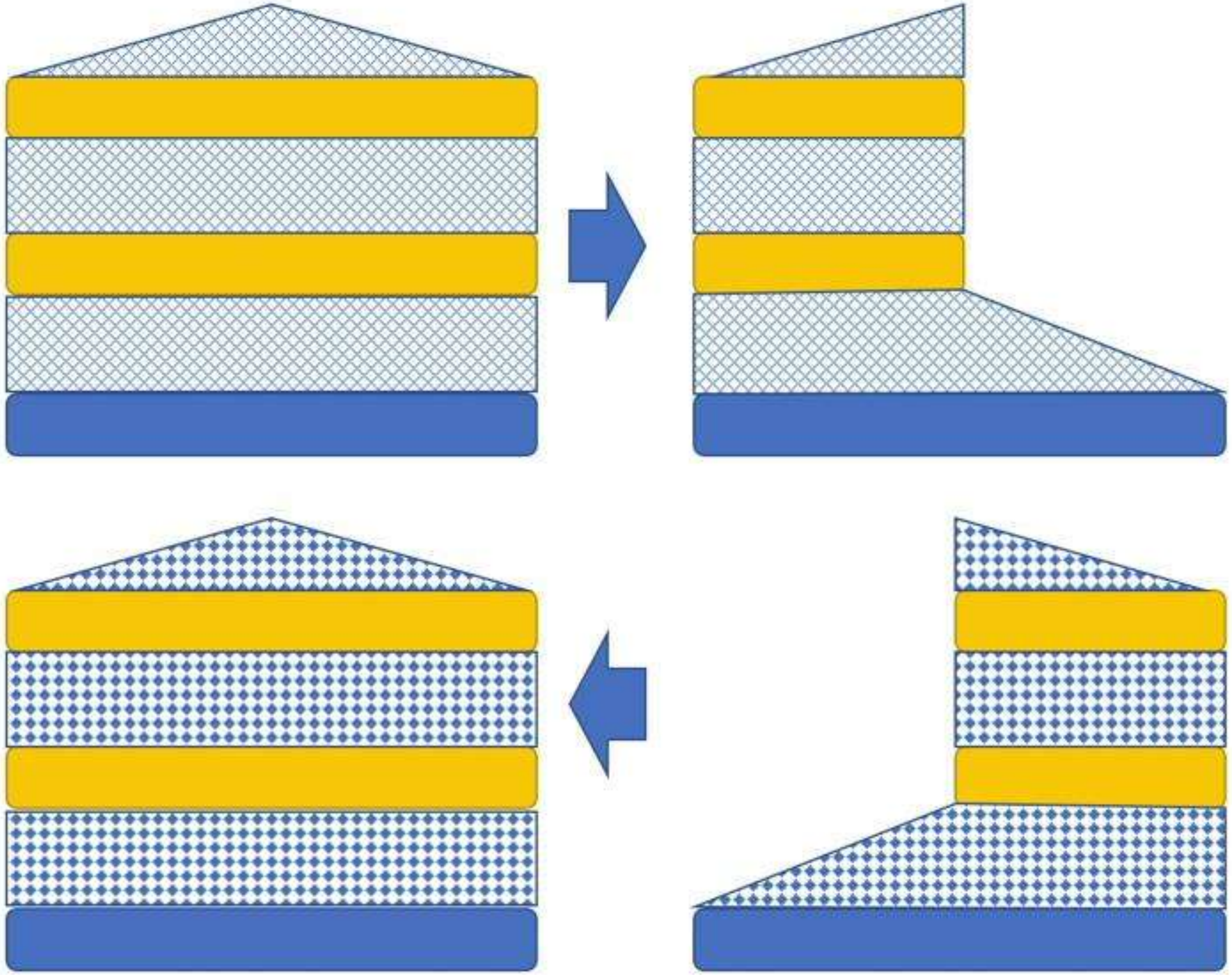
High level steps



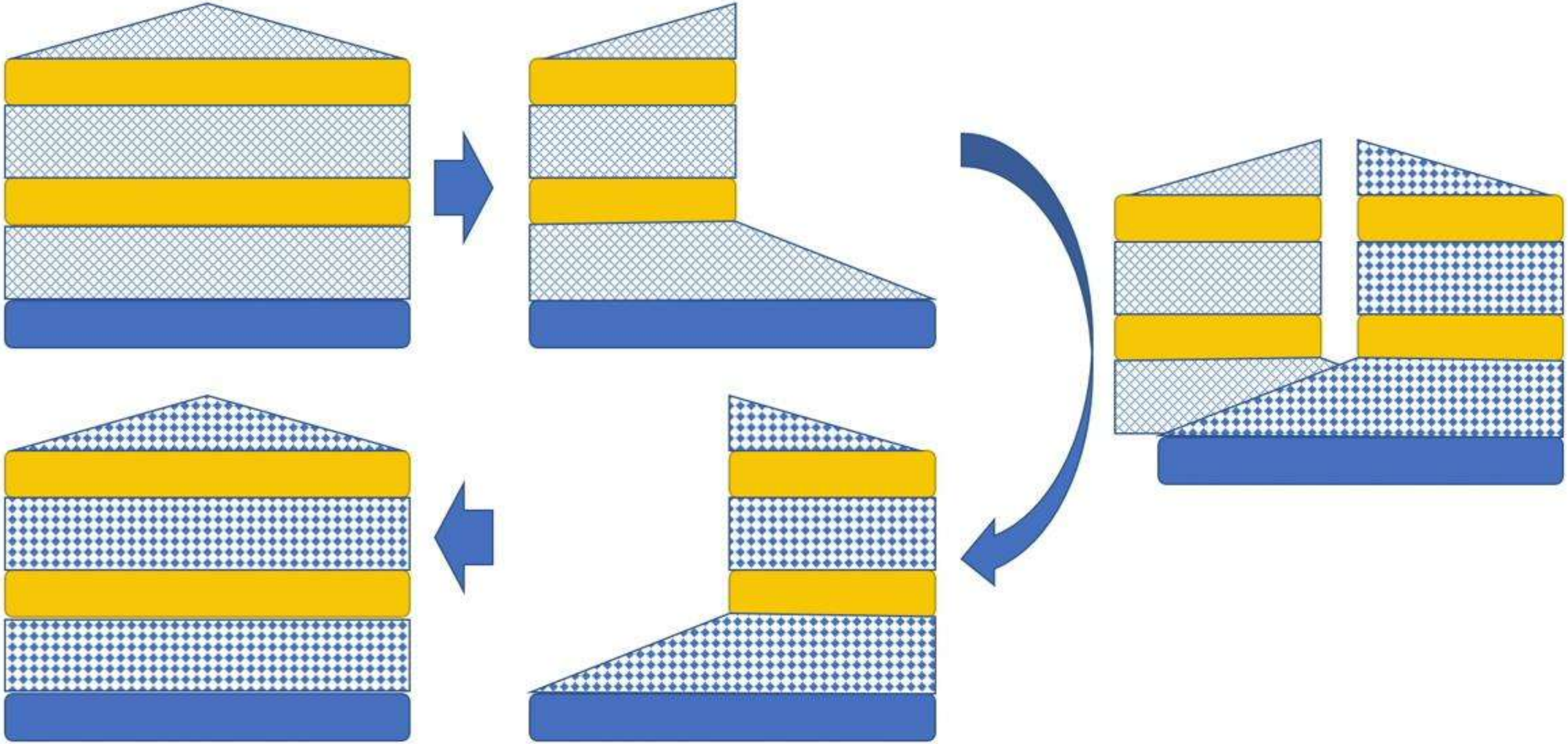
High level steps



High level steps

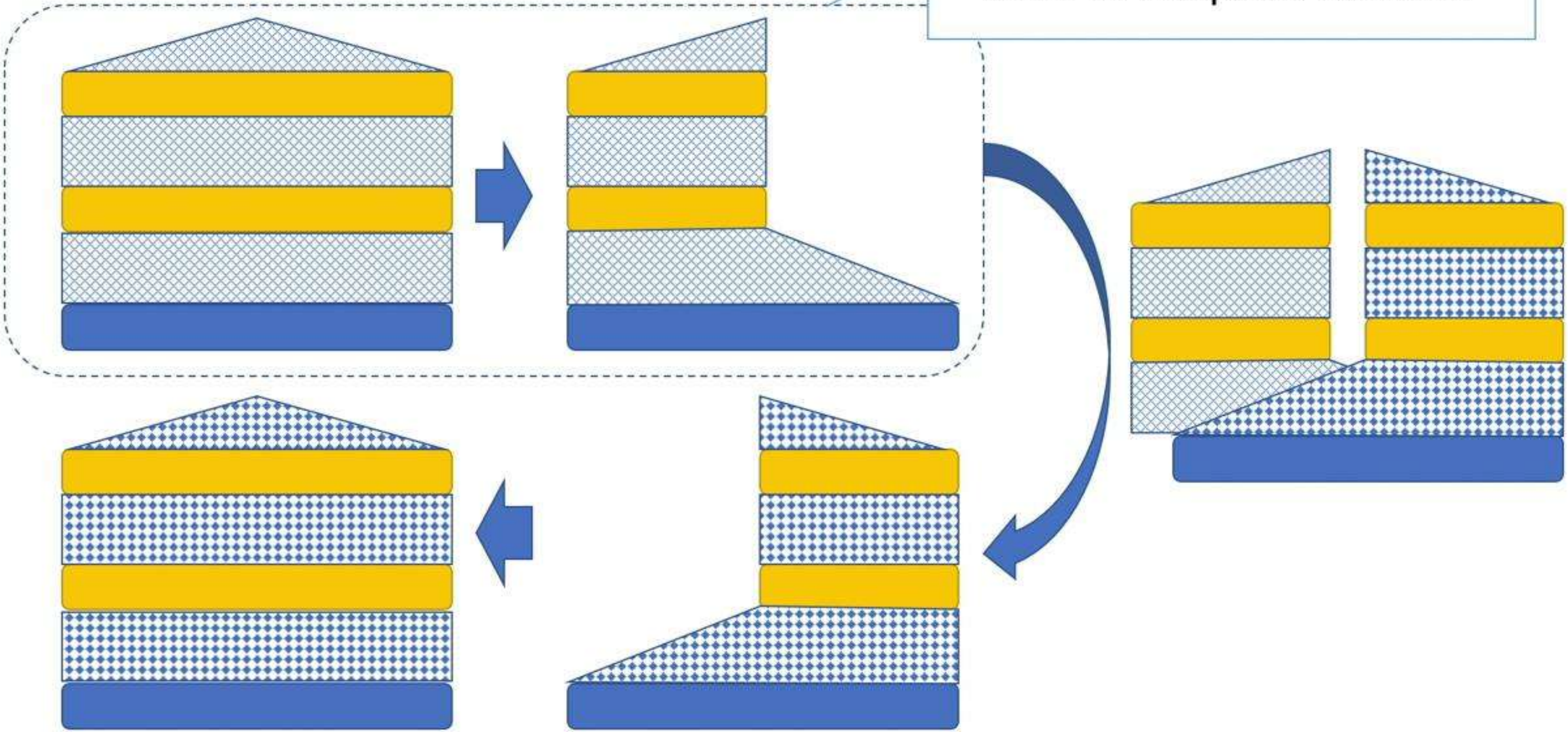


High level steps



High level steps

How to connect a network with its dropout version?



Connecting a network with its dropout

Connecting a network with its dropout

- Main observation: can use two types of line segments.

Connecting a network with its dropout

- Main observation: can use two types of line segments.

- Type (a): if θ_A and θ_B both have low loss, and they only differ in top layer weight, can linearly interpolate between them.

Connecting a network with its dropout

- Main observation: can use two types of line segments.
- Type (a): if θ_A and θ_B both have low loss, and they only differ in top layer weight, can linearly interpolate between them.
- Type (b): If a group of neurons do not have any outgoing edges, can change their incoming edges arbitrarily.

Connecting a network with its dropout

- Main observation: can use two types of line segments.
- Type (a): if θ_A and θ_B both have low loss, and they only differ in top layer weight, can linearly interpolate between them.
- Type (b): If a group of neurons do not have any outgoing edges, can change their incoming edges arbitrarily.
- Idea: Recurse from the top layer, use Type (b) moves to prepare for the next Type (a) move

An example path for 3 layer network

$$(1)\quad \left[\begin{array}{c|c} L_3 & R_3 \end{array}\right] \left(\begin{array}{c|c} L_2 & C_2 \\ \hline D_2 & R_2 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right)$$

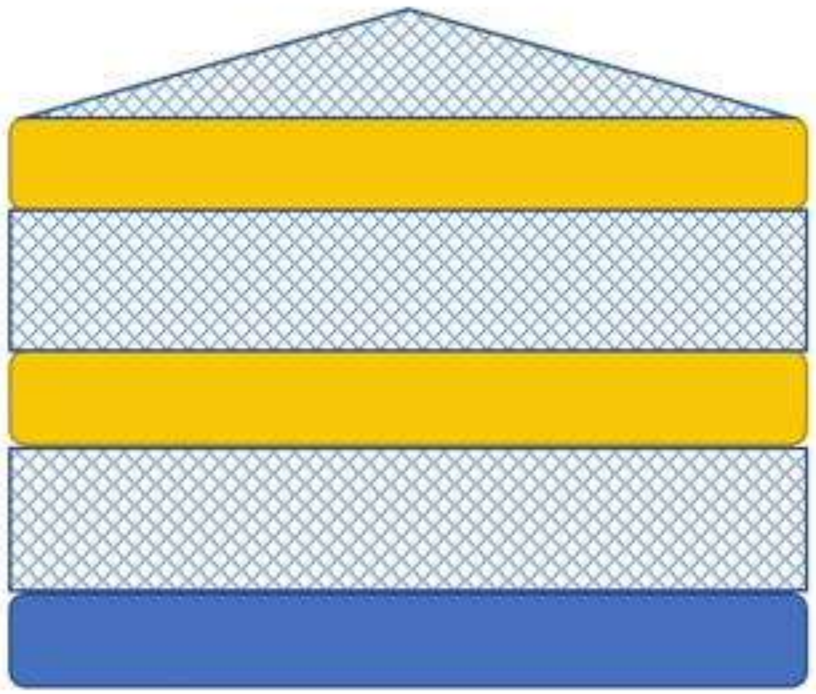
$$(2)\quad \left[\begin{array}{c|c} 2L_3 & 0 \end{array}\right] \left(\begin{array}{c|c} L_2 & C_2 \\ \hline D_2 & R_2 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right) \quad (a)$$

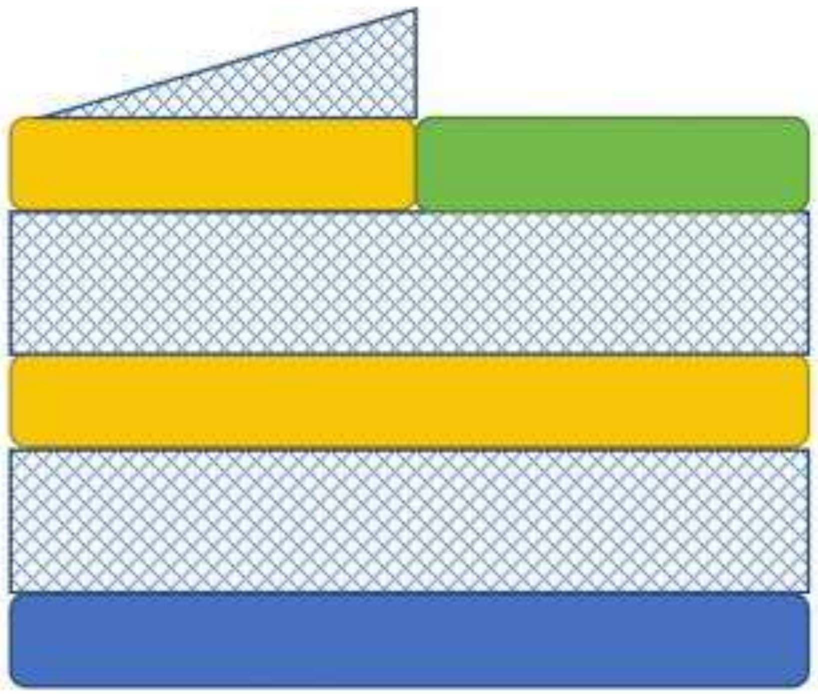
$$(3)\quad \left[\begin{array}{c|c} 2L_3 & 0 \end{array}\right] \left(\begin{array}{c|c} L_2 & C_2 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right) \quad (b)$$

$$(4)\quad \left[\begin{array}{c|c} 0 & 2L_3 \end{array}\right] \left(\begin{array}{c|c} L_2 & C_2 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right) \quad (a)$$

$$(5)\quad \left[\begin{array}{c|c} 0 & 2L_3 \end{array}\right] \left(\begin{array}{c|c} 0 & 0 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right) \quad (b)$$

Example path explained (1) -> (2)

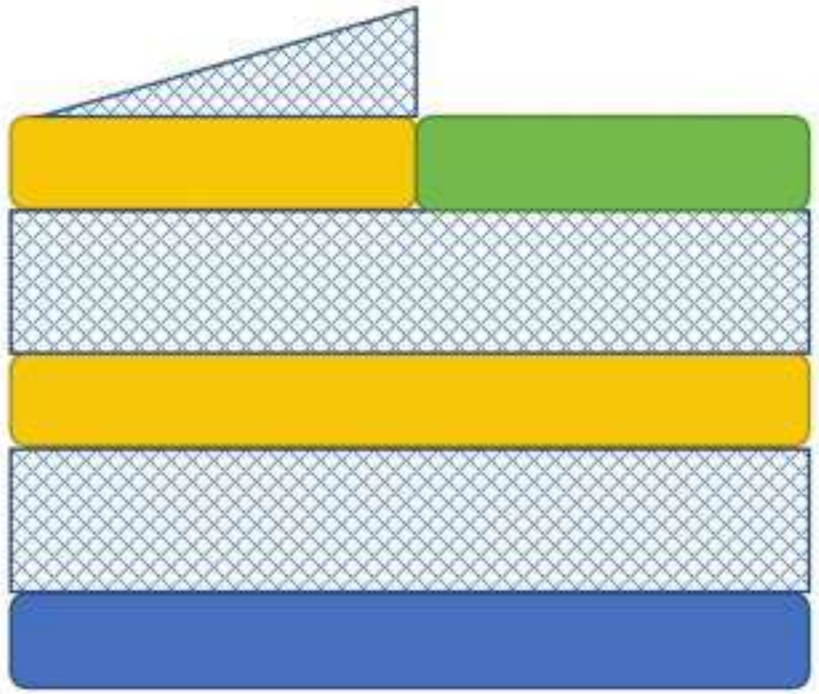


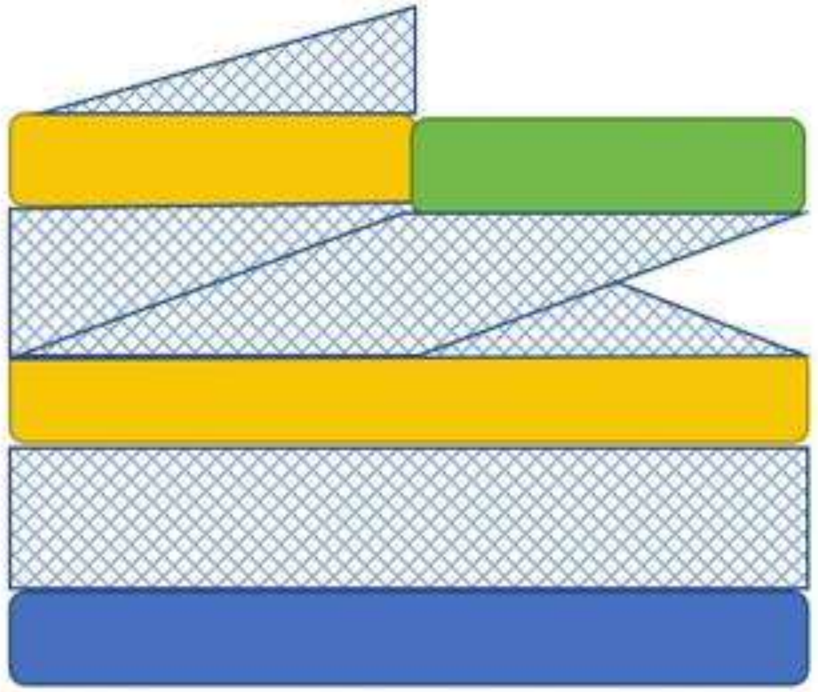


$$(1)\ \begin{pmatrix} L_3 & \big| & R_3 \end{pmatrix} \left(\begin{array}{c|c} L_2 & C_2 \\ \hline D_2 & R_2 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right)$$

$$(2)\ \begin{pmatrix} \color{red}{2L_3} & \big| & 0 \end{pmatrix} \left(\begin{array}{c|c} L_2 & C_2 \\ \hline D_2 & R_2 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right)\ (a)$$

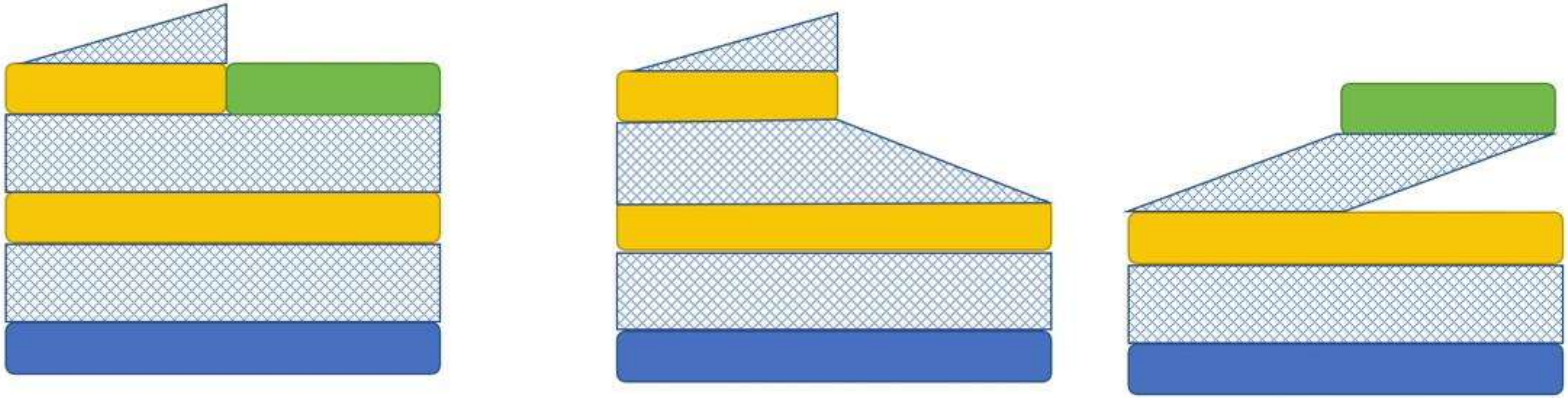
Example path explained (2) -> (3)





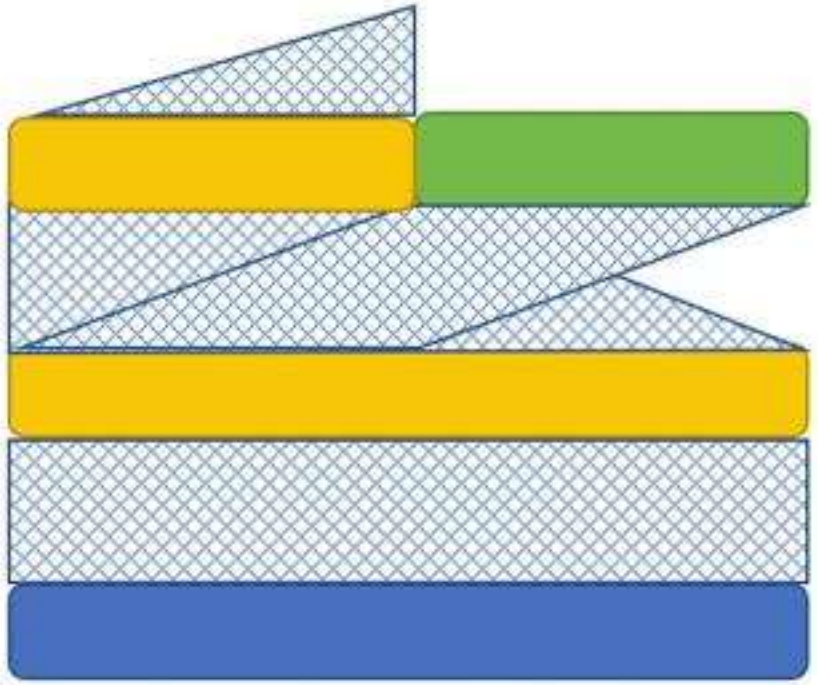
$$(2)\ \left(\begin{array}{c|c} 2L_3 & 0 \end{array}\right) \left(\begin{array}{c|c} L_2 & C_2 \\ \hline D_2 & R_2 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right) \text{(a)} \quad (3)\ \left(\begin{array}{c|c} 2L_3 & 0 \end{array}\right) \left(\begin{array}{c|c} L_2 & C_2 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right) \text{(b)}$$

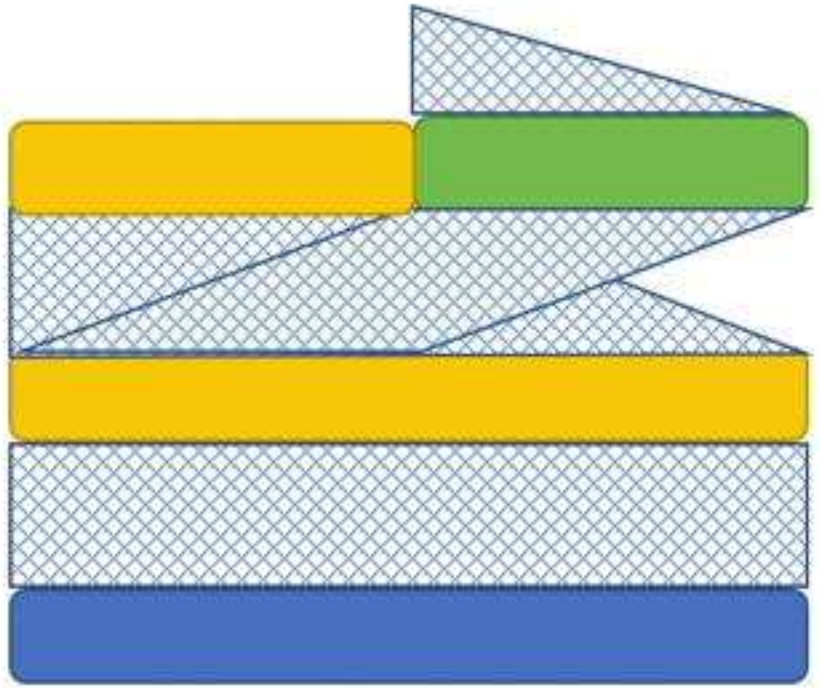
Example path explained (2) -> (3)



$$(2)\ \left(\begin{array}{c|c} 2L_3 & 0 \end{array}\right) \left(\begin{array}{c|c} L_2 & C_2 \\ \hline D_2 & R_2 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right)\ \text{(a)} \quad (3)\ \left(\begin{array}{c|c} 2L_3 & 0 \end{array}\right) \left(\begin{array}{c|c} L_2 & C_2 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right)\ \text{(b)}$$

Example path explained (3) -> (4)

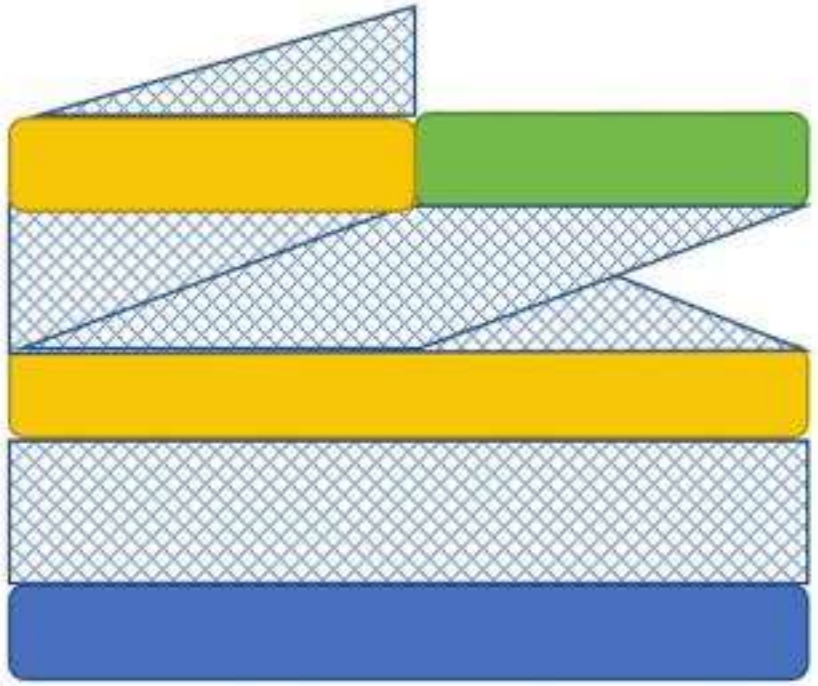


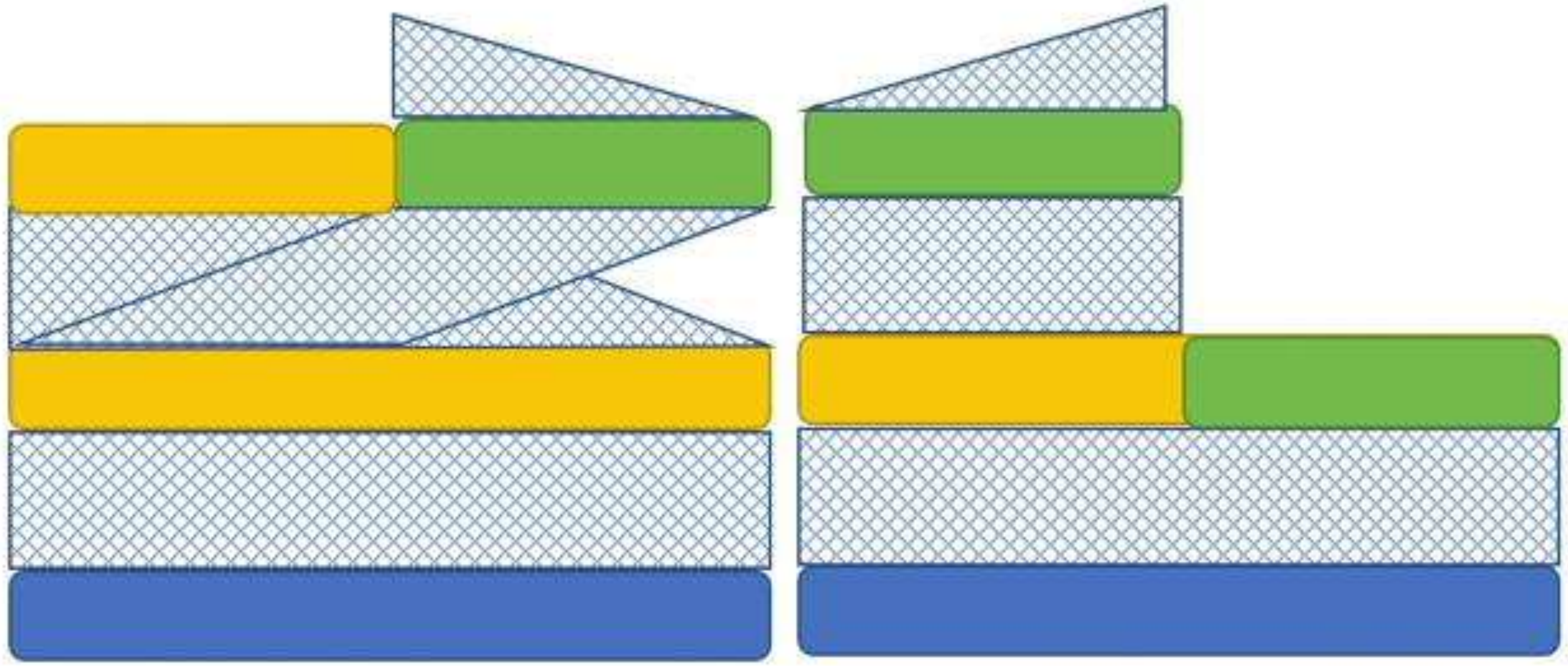


(3) $\left(\begin{array}{c|c} 2L_3 & 0 \end{array}\right) \left(\begin{array}{c|c} L_2 & C_2 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right)$ (b)

(4) $\left(\begin{array}{c|c} 0 & 2L_3 \end{array}\right) \left(\begin{array}{c|c} L_2 & C_2 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right)$ (a)

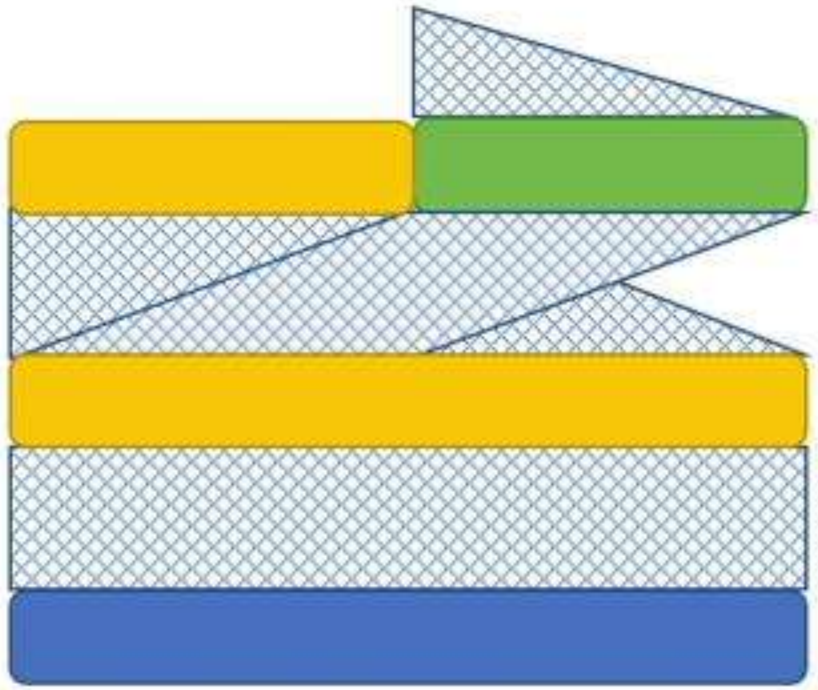
Example path explained (3) -> (4)

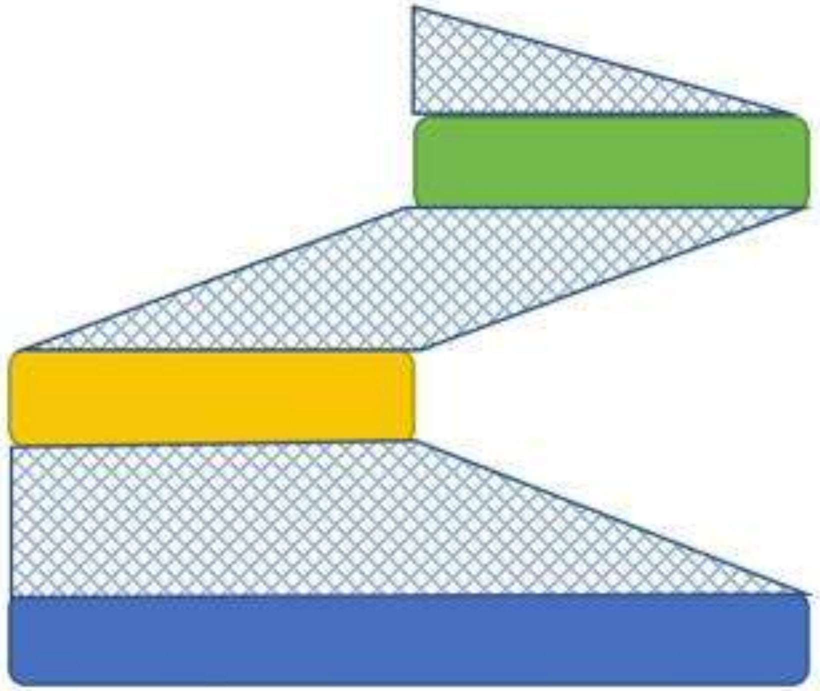




$$(3)\ \left(\begin{array}{c|c} 2L_3 & 0 \end{array}\right) \left(\begin{array}{c|c} L_2 & C_2 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right) \text{(b)} \qquad (4)\ \left(\begin{array}{c|c} 0 & 2L_3 \end{array}\right) \left(\begin{array}{c|c} L_2 & C_2 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right) \text{(a)}$$

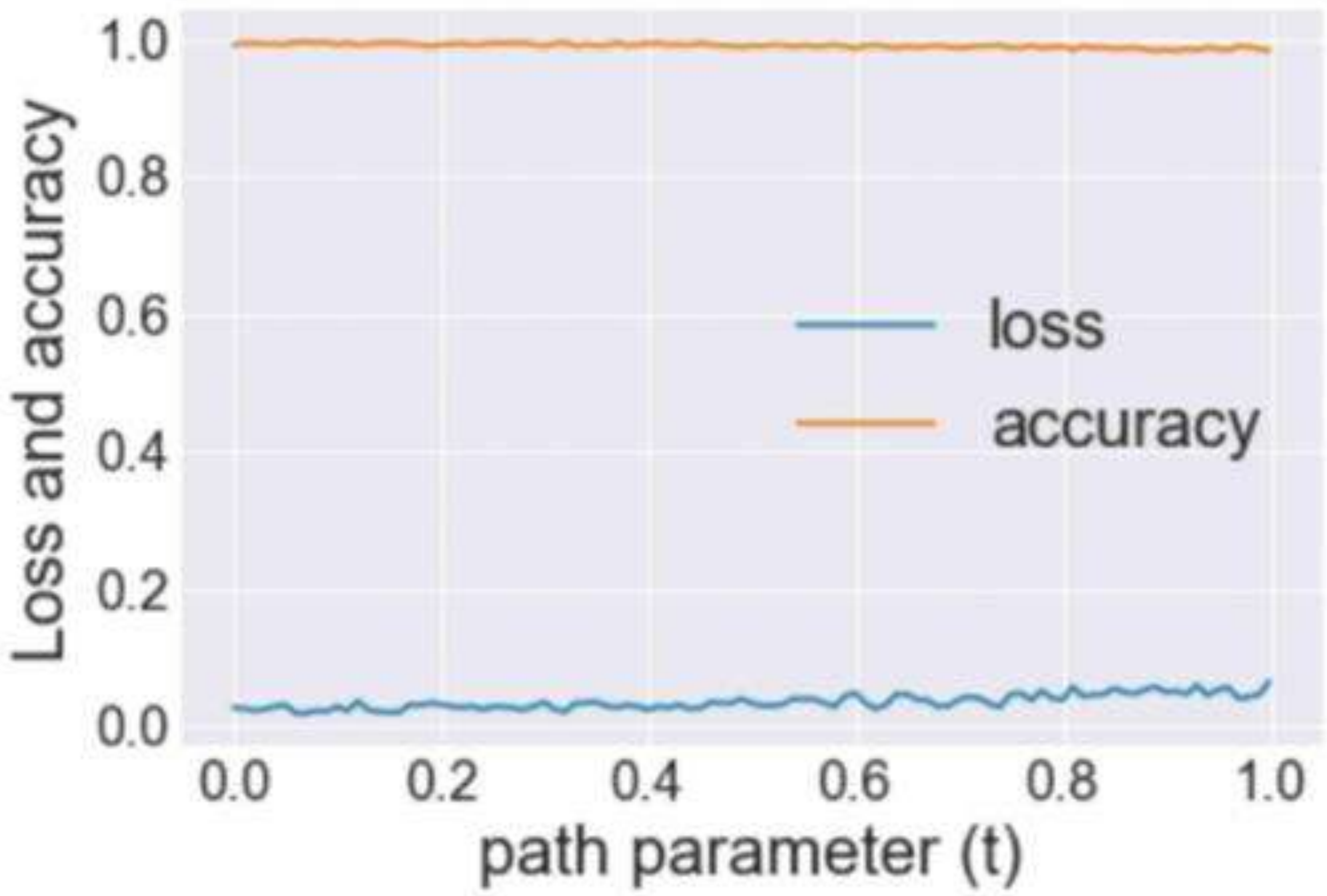
Example path explained (4) -> (5)



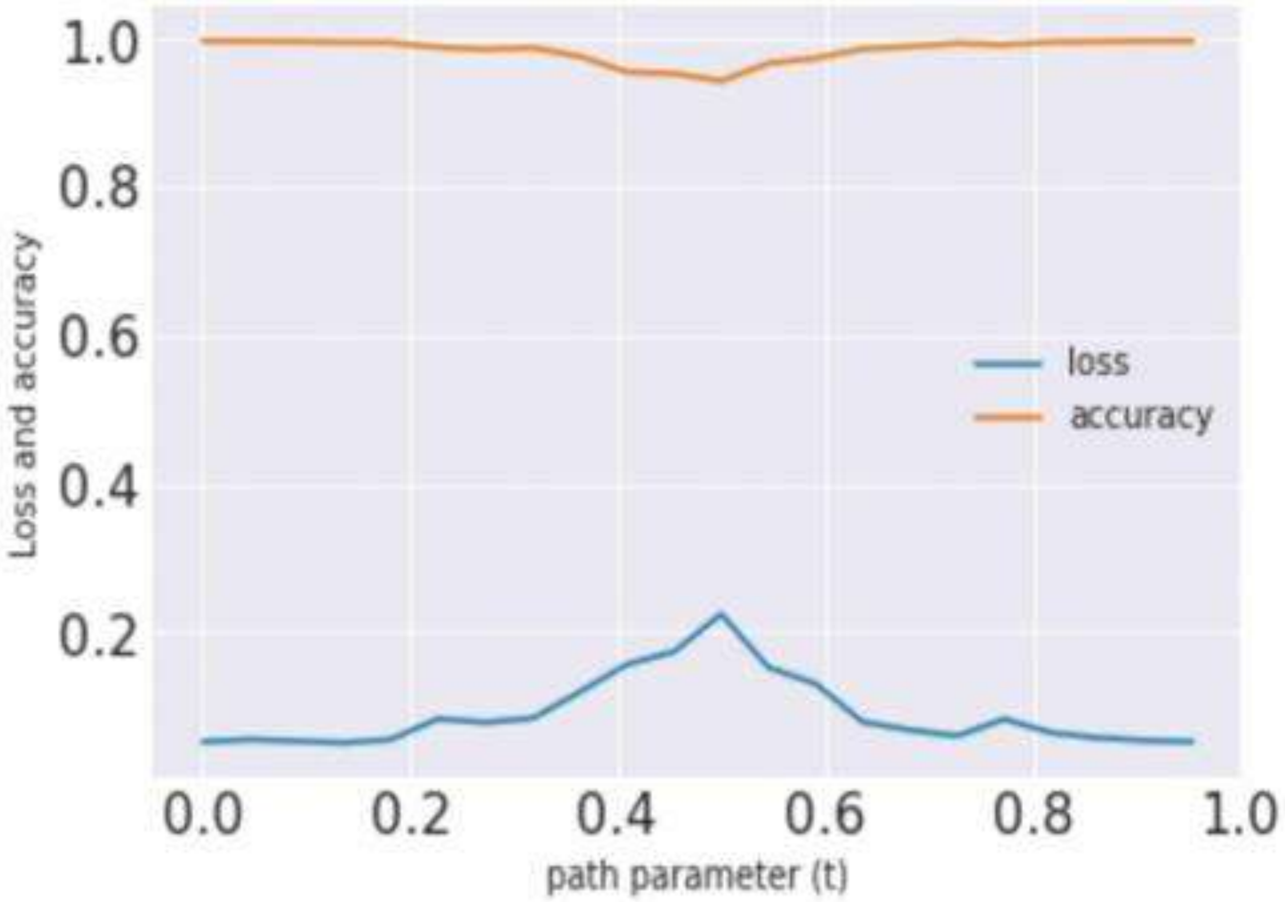


$$(4)\ \left(\begin{array}{c|c} 0 & 2L_3 \end{array}\right) \left(\begin{array}{c|c} L_2 & C_2 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right)\ \text{(a)}\qquad (5)\ \left(\begin{array}{c|c} 0 & 2L_3 \end{array}\right) \left(\begin{array}{c|c} 0 & 0 \\ \hline 2L_2 & 0 \end{array}\right) \left(\begin{array}{c} L_1 \\ \hline B_1 \end{array}\right)\ \text{(b)}$$

Experiments



MNIST, 3-layer CNN



CIFAR-10, VGG-11

Conclusions

For neural networks, not all local/global min are connected, even in the overparametrized setting.

Solutions that satisfy dropout/noise stability are connected.

Open Problems

- Path found by dropout/noise stability are still more complicated than the path found in practice.
- Path are known to exist in practice, even if the solutions are not as dropout stable as we hoped.

- Can we leverage mode connectivity to design better optimization algorithms? Maybe by proving the stronger convexity requirements as Leon talked about?

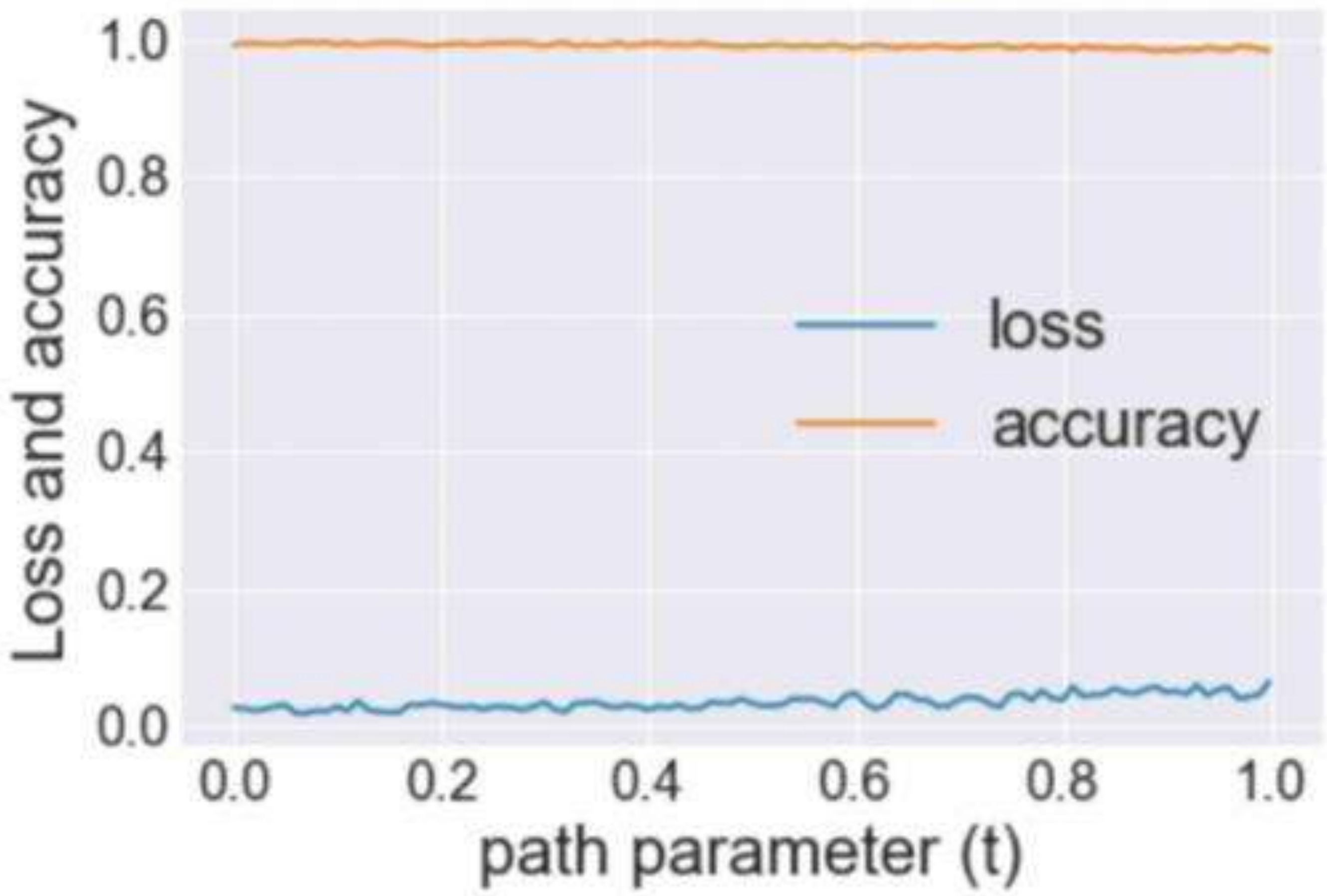
Open Problems

- Path found by dropout/noise stability are still more complicated than the path found in practice.
- Path are known to exist in practice, even if the solutions are not as dropout stable as we hoped.

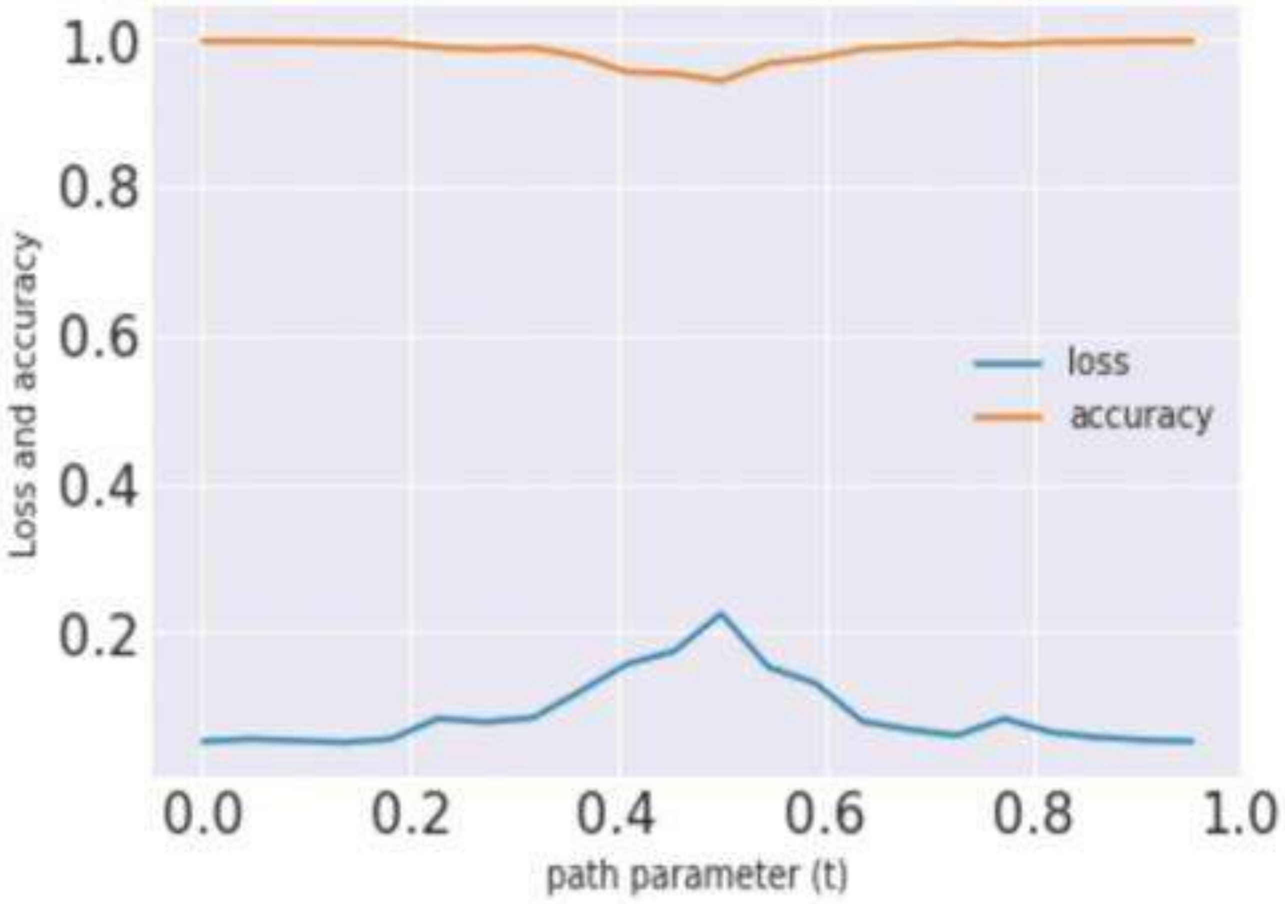
- Can we leverage mode connectivity to design better optimization algorithms? Maybe by proving the stronger convexity requirements as Leon talked about?

Thank you!

Experiments



MNIST, 3-layer CNN



CIFAR-10, VGG-11

Deep learning theory: The first proof for the most simple example of over-parameterization

Yuanzhi Li

Stanford University

date: Today

Over-parameterization (over-capacity)

- By having (much) more parameters in a neural network than necessary.

Over-parameterization (over-capacity)

- By having (much) more parameters in a neural network than necessary.
  - More parameters than the number of training data.

Over-parameterization (over-capacity)

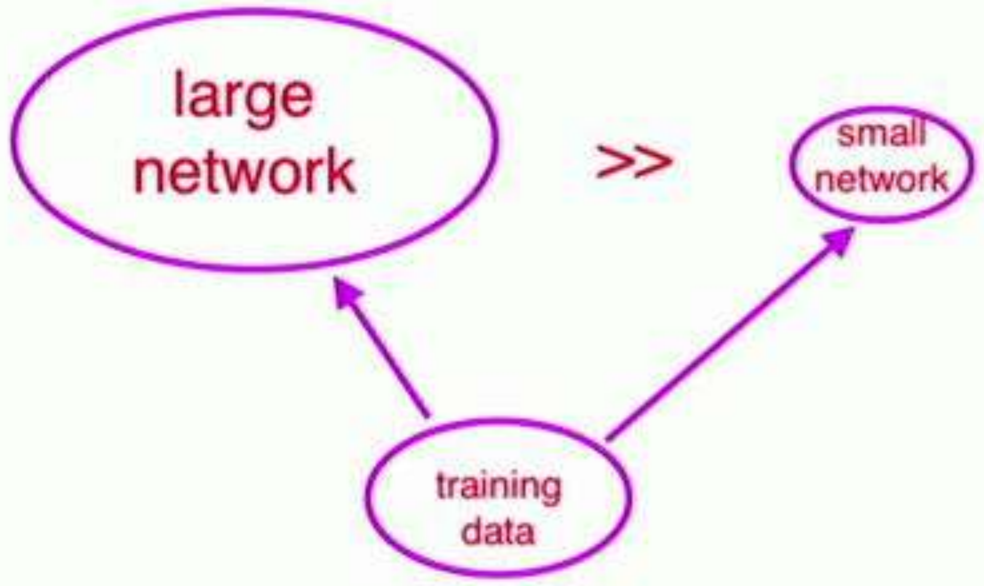
- By having (much) more parameters in a neural network than necessary.
  - More parameters than the number of training data.
  - More parameters than the minimal number of parameters to represent the target function.

Over-parameterization (over-capacity)

- By having (much) more parameters in a neural network than necessary.
  - More parameters than the number of training data.
  - More parameters than the minimal number of parameters to represent the target function.
- It improves the performance of the neural network (both training and testing).

Over-parameterization (over-capacity)

- By having (much) more parameters in a neural network than necessary.
  - More parameters than the number of training data.
  - More parameters than the minimal number of parameters to represent the target function.
- It improves the performance of the neural network (both training and testing).



The most simple example of over-parameterization

- Target network: $f^*(x) = \sum_{i=1}^{d} a_i^* \mathrm{ReLU}(\langle w_i^*, x \rangle)$.

The most simple example of over-parameterization

- Target network: $f^*(x) = \sum_{i=1}^{d} a_i^* \mathrm{ReLU}(\langle w_i^*, x \rangle)$.
  - Where $x \in \mathbb{R}^d$, w_i are orthonormal vectors, $a_i \in \{-1, 1\}$ (all sampled randomly).

The most simple example of over-parameterization

- Target network: $f^*(x) = \sum_{i=1}^{d} a_i^* \mathrm{ReLU}(\langle w_i^*, x\rangle)$.
  - Where $x \in \mathbb{R}^d$, w_i are orthonormal vectors, $a_i \in \{-1, 1\}$ (all sampled randomly).
- Given $N = \mathrm{poly}(d)$ training data $\{x_i, f^*(x_i)\}_{i=1}^{N}$, where $x_i \sim \mathcal{N}(0, I)$:

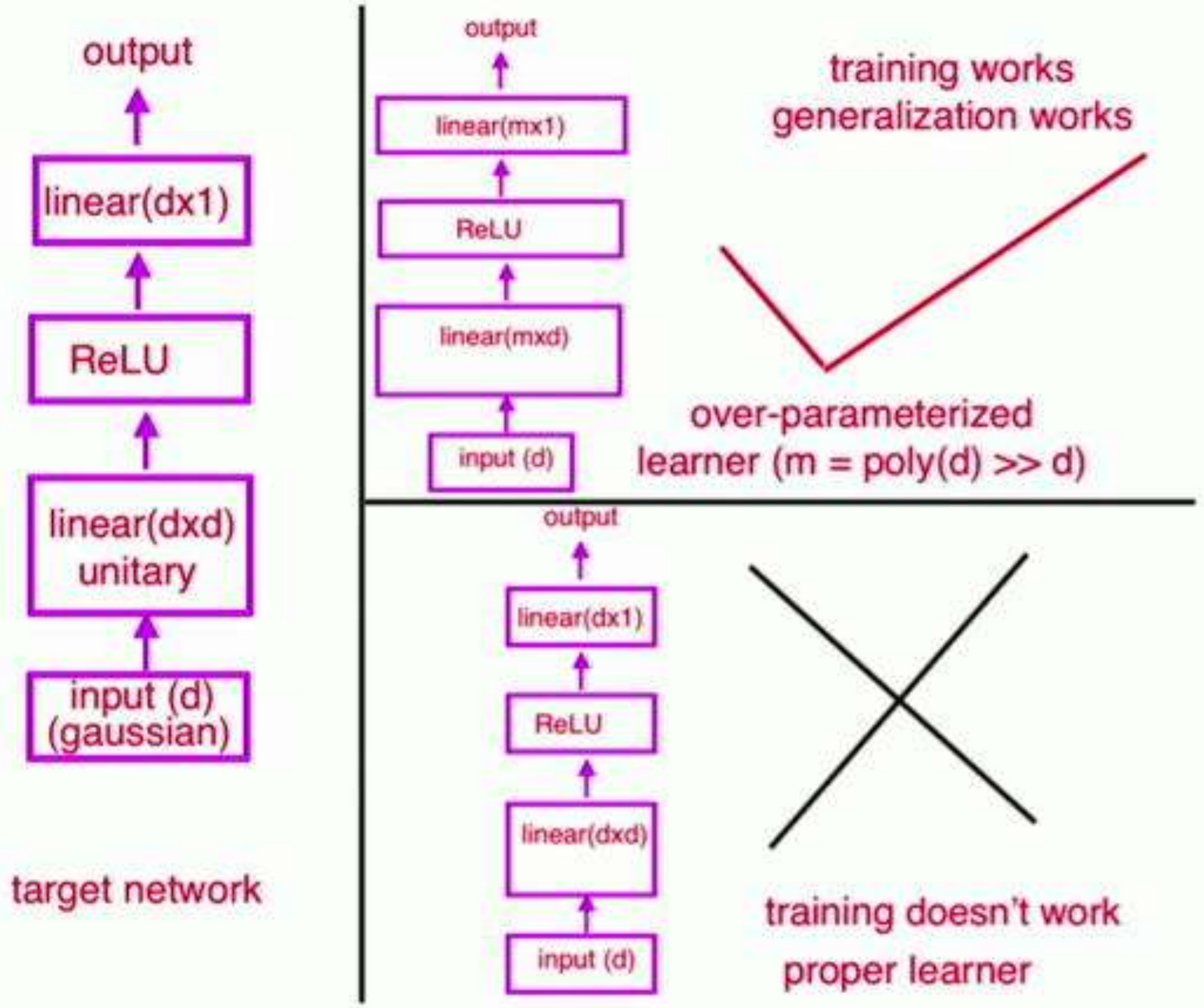
The most simple example of over-parameterization

- Target network: $f^*(x) = \sum_{i=1}^{d} a_i^* \mathrm{ReLU}(\langle w_i^*, x\rangle)$.
  - Where $x \in \mathbb{R}^d$, w_i are orthonormal vectors, $a_i \in \{-1, 1\}$ (all sampled randomly).
- Given $N = \mathrm{poly}(d)$ training data $\{x_i, f^*(x_i)\}_{i=1}^{N}$, where $x_i \sim \mathcal{N}(0, I)$:
  - Using a target network of form $f(x) = \sum_{i=1}^{d} a_i \mathrm{ReLU}(\langle w_i, x\rangle)$, with ℓ_2 loss, running SGD over $\{a_i, w_i\}_{i=1}^{d}$ starting from random initialization, the training process doesn't work (stuck at a bad local minimal).

The most simple example of over-parameterization

- Target network: $f^*(x) = \sum_{i=1}^{d} a_i^* \mathrm{ReLU}(\langle w_i^*, x\rangle)$.
  - Where $x \in \mathbb{R}^d$, w_i are orthonormal vectors, $a_i \in \{-1, 1\}$ (all sampled randomly).
- Given $N = \mathrm{poly}(d)$ training data $\{x_i, f^*(x_i)\}_{i=1}^{N}$, where $x_i \sim \mathcal{N}(0, I)$:
  - Using a target network of form $f(x) = \sum_{i=1}^{d} a_i \mathrm{ReLU}(\langle w_i, x\rangle)$, with ℓ_2 loss, running SGD over $\{a_i, w_i\}_{i=1}^{d}$ starting from random initialization, the training process doesn't work (stuck at a bad local minimal).
  - Using a target network of form $f(x) = \sum_{i=1}^{m} a_i \mathrm{ReLU}(\langle w_i, x\rangle)$ where $m \gg d$ ($m = \mathrm{poly}(d)$), with ℓ_2 loss, running SGD over $\{a_i, w_i\}_{i=1}^{m}$ starting from random initialization, the training process works and generalization works.

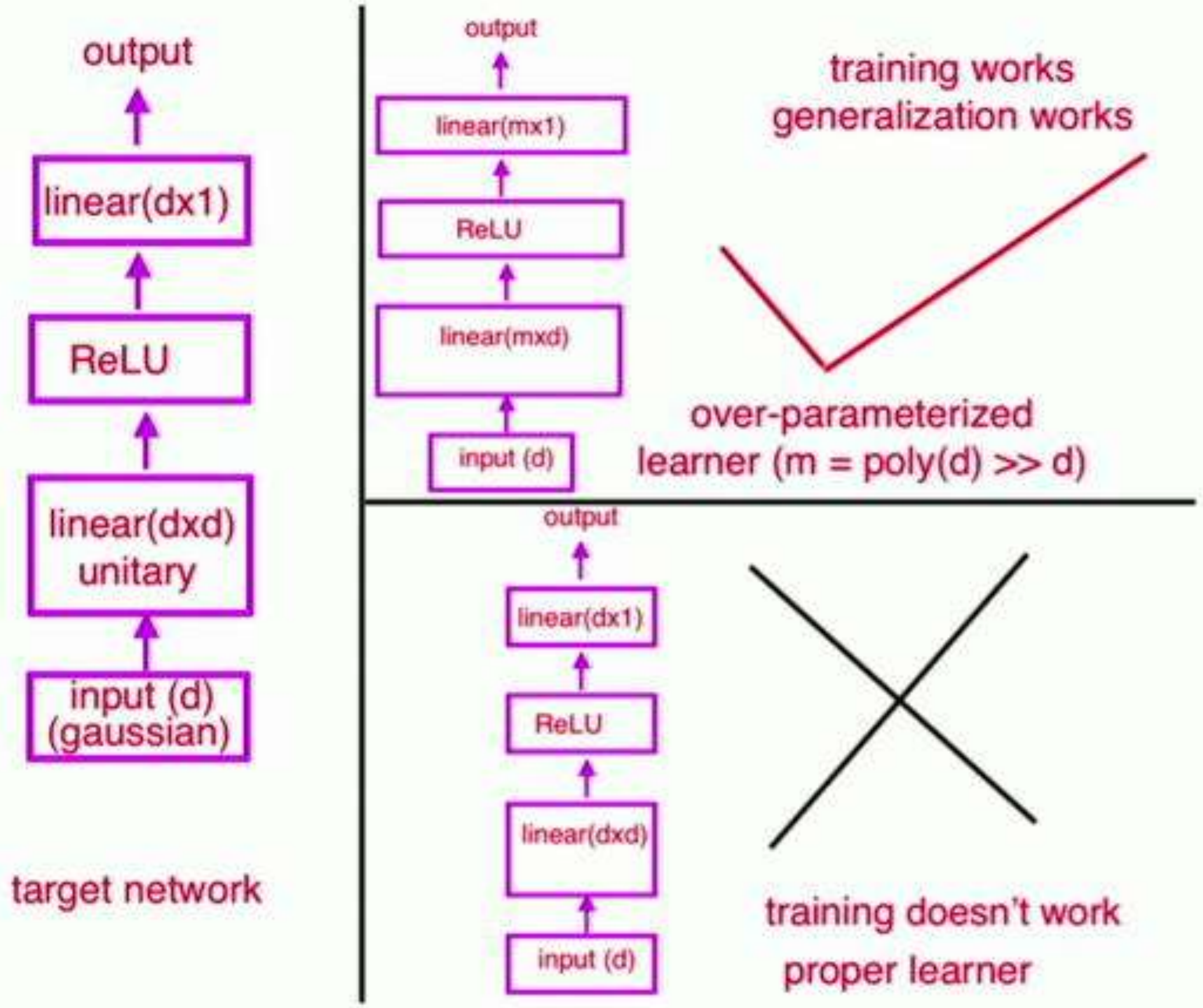
The most simple example of over-parameterization



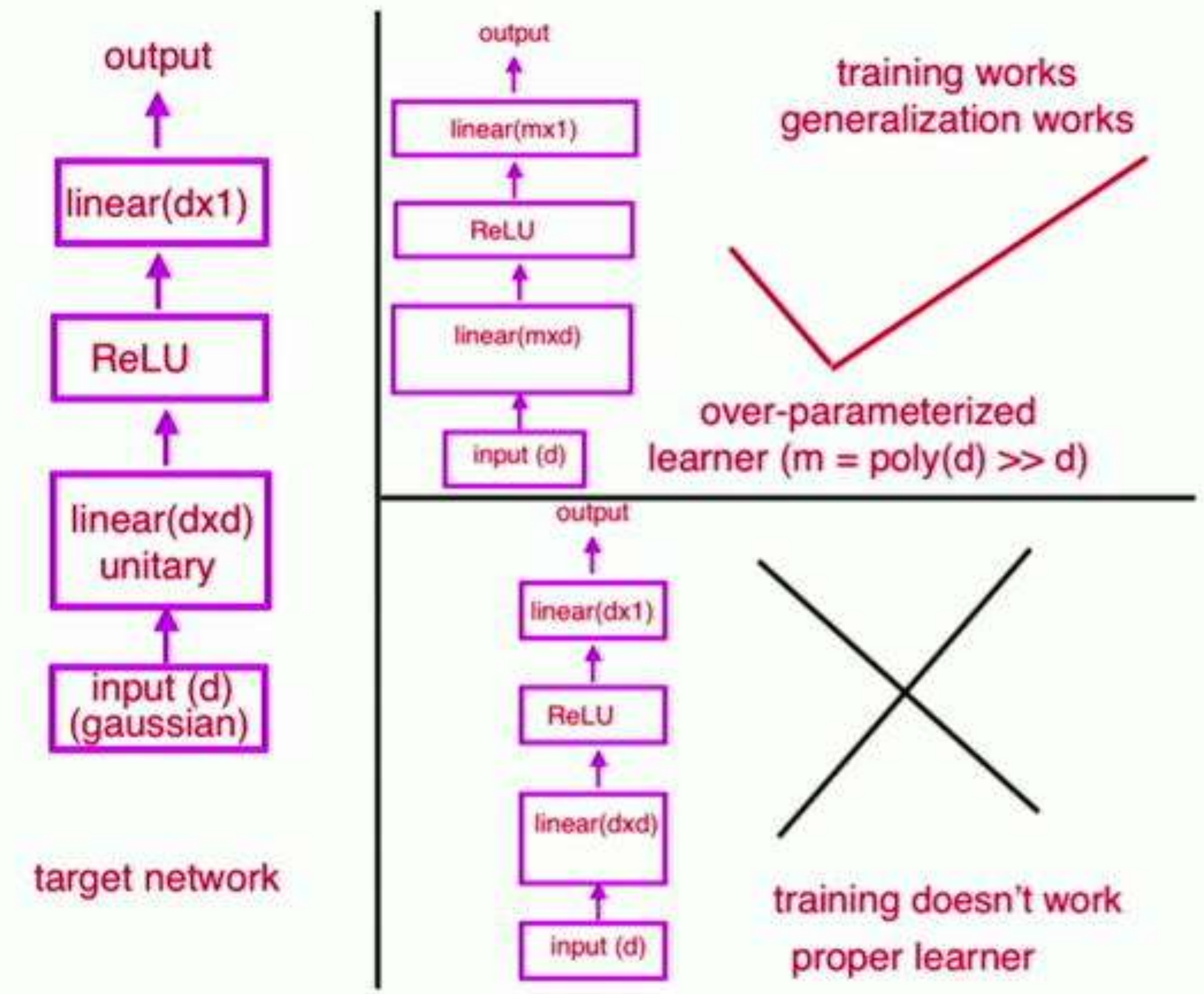
The most simple example of over-parameterization

- Target network: $f^*(x) = \sum_{i=1}^{d} a_i^* \mathrm{ReLU}(\langle w_i^*, x\rangle)$.
  - Where $x \in \mathbb{R}^d$, w_i are orthonormal vectors, $a_i \in \{-1, 1\}$ (all sampled randomly).
- Given $N = \mathrm{poly}(d)$ training data $\{x_i, f^*(x_i)\}_{i=1}^{N}$, where $x_i \sim \mathcal{N}(0, I)$:
  - Using a target network of form $f(x) = \sum_{i=1}^{d} a_i \mathrm{ReLU}(\langle w_i, x\rangle)$, with ℓ_2 loss, running SGD over $\{a_i, w_i\}_{i=1}^{d}$ starting from random initialization, the training process doesn't work (stuck at a bad local minimal).
  - Using a target network of form $f(x) = \sum_{i=1}^{m} a_i \mathrm{ReLU}(\langle w_i, x\rangle)$ where $m \gg d$ ($m = \mathrm{poly}(d)$), with ℓ_2 loss, running SGD over $\{a_i, w_i\}_{i=1}^{m}$ starting from random initialization, the training process works and generalization works.

The most simple example of over-parameterization



The most simple example of over-parameterization



-
- Folklore example ([Lecun et al'14]), formally reported in for example [GLM'17], empirically.

The most simple example of over-parameterization

- Question: Can we prove it?

The most simple example of over-parameterization

- Question: Can we prove it?
- Can we formally prove that over-parameterization does help in this simple simple simple simple simple simple example?

The most simple example of over-parameterization

- Question: Can we prove it?
- Can we formally prove that over-parameterization does help in this simple simple simple simple simple simple example?
- What is the fundamental reason behind over-parameterization?

The most simple example of over-parameterization

- Question: Can we prove it?
- Can we formally prove that over-parameterization does help in this simple simple simple simple simple simple example?
- What is the fundamental reason behind over-parameterization?
- Let us start with prior works on over-parameterization (which are fundamentally different from this work).

Prior works

- Over-parameterization and the neural tangent kernel (NTK). (More info: See my talk at the simons institute $https://simons.berkeley.edu/talks/tbd-70$ $https://arxiv.org/abs/1811.03962$).

Prior works

- Over-parameterization and the neural tangent kernel (NTK). (More info: See my talk at the simons institute $https://simons.berkeley.edu/talks/tbd-70$ $https://arxiv.org/abs/1811.03962$).
  - When the following Oops condition holds:

Prior works

- Over-parameterization and the neural tangent kernel (NTK). (More info: See my talk at the simons institute $https://simons.berkeley.edu/talks/tbd-70$ $https://arxiv.org/abs/1811.03962$).
  - When the following Oops condition holds:
    - Over-parameterization ($m = \text{poly}(N)$, N : number of training data, m : number of parameters).

Prior works

- Over-parameterization and the neural tangent kernel (NTK). (More info: See my talk at the simons institute
  $https://simons.berkeley.edu/talks/tbd-70$
  $https://arxiv.org/abs/1811.03962$).
  - When the following Oops condition holds:
    - Over-parameterization ($m = \text{poly}(N)$, N : number of training data, m : number of parameters).
    - Proper learning rate.

Prior works

- Over-parameterization and the neural tangent kernel (NTK). (More info: See my talk at the simons institute $https://simons.berkeley.edu/talks/tbd-70$ $https://arxiv.org/abs/1811.03962$).
  - When the following Oops condition holds:
    - Over-parameterization ($m = \text{poly}(N)$, N : number of training data, m : number of parameters).
    - Proper learning rate.
    - Small learning rate.

Prior works

- Over-parameterization and the neural tangent kernel (NTK). (More info: See my talk at the simons institute $https://simons.berkeley.edu/talks/tbd-70$ $https://arxiv.org/abs/1811.03962$).
    - When the following Oops condition holds:
        - Over-parameterization ($m = \text{poly}(N)$, N : number of training data, m : number of parameters).
        - Proper learning rate.
        - Small learning rate.
    - Then learning such a neural network via SGD essentially reduces to learning over its NTK (convex problem).

Prior works

- Over-parameterization and the neural tangent kernel (NTK). (More info: See my talk at the simons institute $https://simons.berkeley.edu/talks/tbd-70$ $https://arxiv.org/abs/1811.03962$).
  - When the following Oops condition holds:
    - Over-parameterization ($m = \text{poly}(N)$, N : number of training data, m : number of parameters).
    - Proper learning rate.
    - Small learning rate.
  - Then learning such a neural network via SGD essentially reduces to learning over its NTK (convex problem).
- As a corollary of $[AL'19]$($https://arxiv.org/abs/1905.10337$) NTK cannot learn a target network with ReLU activation with accuracy 1/poly(d), when the input is standard Gaussian (even under the case where the target network is a single ReLU) and $N = e^{o(d)}$.

Prior works

- Over-parameterization and the neural tangent kernel (NTK). (More info: See my talk at the simons institute $https://simons.berkeley.edu/talks/tbd-70$ $https://arxiv.org/abs/1811.03962$).
  - When the following Oops condition holds:
    - Over-parameterization ($m = \text{poly}(N)$, N : number of training data, m : number of parameters).
    - Proper learning rate.
    - Small learning rate.
  - Then learning such a neural network via SGD essentially reduces to learning over its NTK (convex problem).
- As a corollary of $[AL'19](https://arxiv.org/abs/1905.10337)$ NTK cannot learn a target network with ReLU activation with accuracy 1/poly(d), when the input is standard Gaussian (even under the case where the target network is a single ReLU) and $N = e^{o(d)}$.
- So this simple example can not be explained by theorem of NTK.

Prior works differ from NTK

- $[LY'17]$ Given a properly parameterized network and a good initialization ($\|W - W^*\|_2 \leq 0.1$ and $a_i = a_i^*$), then running SGD over W works. (Note: W^* is unitary so $\|W^*\|_2 = 1$.)

Prior works differ from NTK

- $[LY'17]$ Given a properly parameterized network and a good initialization ($\|W - W^*\|_2 \leq 0.1$ and $a_i = a_i^*$), then running SGD over W works. (Note: W^* is unitary so $\|W^*\|_2 = 1$.)
- This is such a dumb result – Michael Cohen.

Prior works differ from NTK

- $[LY'17]$ Given a properly parameterized network and a good initialization ($\|W - W^*\|_2 \leq 0.1$ and $a_i = a_i^*$), then running SGD over W works. (Note: W^* is unitary so $\|W^*\|_2 = 1$.)
- This is such a dumb result – Michael Cohen.
- $[GLM'17]$ Given proper parameterization, using a different loss function, a different learner network (degree ≤ 4 activation functions) and then switch back to ReLU when doing prediction, it works.

Prior works differ from NTK

- $[LY'17]$ Given a properly parameterized network and a good initialization ($\|W - W^*\|_2 \leq 0.1$ and $a_i = a_i^*$), then running SGD over W works. (Note: W^* is unitary so $\|W^*\|_2 = 1$.)
- This is such a dumb result – Michael Cohen.
- $[GLM'17]$ Given proper parameterization, using a different loss function, a different learner network (degree ≤ 4 activation functions) and then switch back to ReLU when doing prediction, it works.
- We believe that analyzing the original problem is technically challenging and its resolution will be potentially enlightening for the understanding landscape of loss function with permutation invariance – GLM.

Prior works differ from NTK

- $[LY'17]$ Given a properly parameterized network and a good initialization ($\|W - W^*\|_2 \leq 0.1$ and $a_i = a_i^*$), then running SGD over W works. (Note: W^* is unitary so $\|W^*\|_2 = 1$.)
- This is such a dumb result – Michael Cohen.
- $[GLM'17]$ Given proper parameterization, using a different loss function, a different learner network (degree ≤ 4 activation functions) and then switch back to ReLU when doing prediction, it works.
- We believe that analyzing the original problem is technically challenging and its resolution will be potentially enlightening for the understanding landscape of loss function with permutation invariance – GLM.
- $[LMW'19]$ Given an exponential amount of over-parameterization, noisy SGD works.

Prior works differ from NTK

- $[LY'17]$ Given a properly parameterized network and a good initialization ($\|W - W^*\|_2 \leq 0.1$ and $a_i = a_i^*$), then running SGD over W works. (Note: W^* is unitary so $\|W^*\|_2 = 1$.)
- This is such a dumb result – Michael Cohen.
- $[GLM'17]$ Given proper parameterization, using a different loss function, a different learner network (degree ≤ 4 activation functions) and then switch back to ReLU when doing prediction, it works.
- We believe that analyzing the original problem is technically challenging and its resolution will be potentially enlightening for the understanding landscape of loss function with permutation invariance – GLM.
- $[LMW'19]$ Given an exponential amount of over-parameterization, noisy SGD works.
- However, (ℓ_1 constrained) linear regression over an n exponential size set of feature mappings also works – Moving from exponential to polynomial is the key advantage of using neural networks.

This work

Theorem (LMZ'19, sketched)

Given target network of form $f^*(x) = \langle v^*, x\rangle + \sum_{i=1}^{r} a_i^* ReLU(\langle w_i^*, x\rangle)$*, where* $w_i^* \in \mathbb{R}^d$ *are orthonormal (so* $r \le d$*), each* $a_i^* \in [1, \kappa] \cup [-\kappa, -1]$ *for constant* κ*.*
Given learner network of form $f(x) = \langle v, x\rangle + \sum_{i=1}^{m} a_i ReLU(\langle w_i, x\rangle)$ *where* $m = poly(d)$*, given* $N = poly(d)$ *many training data, then (mini-batch) SGD starting from random initialization with* $1/poly(d)$ *learning rate reaches generalization error* $1/poly(d)$ *after* $poly(d)$ *iterations.*

This work

Theorem (LMZ'19, sketched)

Given target network of form $f^*(x) = \langle v^*, x\rangle + \sum_{i=1}^{r} a_i^* ReLU(\langle w_i^*, x\rangle)$, *where* $w_i^* \in \mathbb{R}^d$ *are orthonormal (so* $r \le d$*), each* $a_i^* \in [1, \kappa] \cup [-\kappa, -1]$ *for constant* κ.
Given learner network of form $f(x) = \langle v, x\rangle + \sum_{i=1}^{m} a_i ReLU(\langle w_i, x\rangle)$ *where* $m = poly(d)$, *given* $N = poly(d)$ *many training data, then (mini-batch) SGD starting from random initialization with* $1/poly(d)$ *learning rate reaches generalization error* $1/poly(d)$ *after* $poly(d)$ *iterations.*

- m can be larger than N.

This work

Theorem (LMZ'19, sketched)

Given target network of form $f^*(x) = \langle v^*, x\rangle + \sum_{i=1}^{r} a_i^* ReLU(\langle w_i^*, x\rangle)$, *where* $w_i^* \in \mathbb{R}^d$ *are orthonormal (so* $r \le d$*), each* $a_i^* \in [1, \kappa] \cup [-\kappa, -1]$ *for constant* κ.
Given learner network of form $f(x) = \langle v, x\rangle + \sum_{i=1}^{m} a_i ReLU(\langle w_i, x\rangle)$ *where* $m = poly(d)$, *given* $N = poly(d)$ *many training data, then (mini-batch) SGD starting from random initialization with* $1/poly(d)$ *learning rate reaches generalization error* $1/poly(d)$ *after* $poly(d)$ *iterations.*

- m can be larger than N.
- To simplify the analysis, note that a_i, w_i are (positive) scaling invariant, so in this work we re-parameterize a_i by $a_i = \|w_i\|_2$ for a half of the neuron and $a_i = -\|w_i\|_2$ for the other half.

This work

Theorem (LMZ'19, sketched)

Given target network of form $f^*(x) = \langle v^*, x\rangle + \sum_{i=1}^{r} a_i^* \, ReLU(\langle w_i^*, x\rangle)$, *where* $w_i^* \in \mathbb{R}^d$ *are orthonormal (so* $r \le d$*), each* $a_i^* \in [1, \kappa] \cup [-\kappa, -1]$ *for constant* κ.
Given learner network of form $f(x) = \langle v, x\rangle + \sum_{i=1}^{m} a_i \, ReLU(\langle w_i, x\rangle)$ *where* $m = poly(d)$, *given* $N = poly(d)$ *many training data, then (mini-batch) SGD starting from random initialization with* $1/poly(d)$ *learning rate reaches generalization error* $1/poly(d)$ *after* $poly(d)$ *iterations.*

- m can be larger than N.
- To simplify the analysis, note that a_i, w_i are (positive) scaling invariant, so in this work we re-parameterize a_i by $a_i = \|w_i\|_2$ for a half of the neuron and $a_i = -\|w_i\|_2$ for the other half.
- This is not the NTK regime due to the size of the random initialization (we use $w_i \sim \mathcal{N}(0, 1/m)$ so $|a_i| \approx \frac{1}{\sqrt{m}}$), NTK requires $|a_i| \approx 1$.

Why does over-parameterization help?

- The new tool: Poly-size coupling with a non-convex infinite neuron process.

Why does over-parameterization help?

- The new tool: Poly-size coupling with a non-convex infinite neuron process.
- With sufficient over-parameterization, the optimization process of the network is actually simulating an infinite neuron optimization process, which is not convex (not NTK) but still have a benign landscape.

Why does over-parameterization help?

- The new tool: Poly-size coupling with a non-convex infinite neuron process.
- With sufficient over-parameterization, the optimization process of the network is actually simulating an infinite neuron optimization process, which is not convex (not NTK) but still have a benign landscape.
- Difficulty: why the infinite neuron process has a better training performance? How does the simulation work with only poly-size over-parameterization?

The infinite neuron optimization process

- For simplicity, let us focus on the case when $r = d$, all a_i^* are positive and all a_i are positive (so $a_i = \|w_i\|_2$).

The infinite neuron optimization process

- For simplicity, let us focus on the case when $r = d$, all a_i^* are positive and all a_i are positive (so $a_i = \|w_i\|_2$).
- Without the linear term, the population loss $L = \mathbb{E}_{x \sim \mathcal{N}(0,I)}(f(x) - f^*(x))^2$ is given by

$$L = \sigma_0 \left\| \sum_{i=1}^{m} \|w_i\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \sum_{i=1}^{m} w_i^{\otimes 2} \bar{w}_i^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

The infinite neuron optimization process

- For simplicity, let us focus on the case when $r = d$, all a_i^* are positive and all a_i are positive (so $a_i = \|w_i\|_2$).
- Without the linear term, the population loss $L = \mathbb{E}_{x \sim \mathcal{N}(0,I)}(f(x) - f^*(x))^2$ is given by

$$L = \sigma_0 \left\| \sum_{i=1}^{m} \|w_i\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \sum_{i=1}^{m} w_i^{\otimes 2} \bar{w}_i^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Where $\bar{w}_i = \frac{w_i}{\|w_i\|_2}$ and $\sigma_i \approx \frac{1}{\text{poly}(i)}$ (the Hermits coefficients of ReLU).

The infinite neuron optimization process

- For simplicity, let us focus on the case when $r = d$, all a_i^* are positive and all a_i are positive (so $a_i = \|w_i\|_2$).
- Without the linear term, the population loss $L = \mathbb{E}_{x \sim \mathcal{N}(0,I)}(f(x) - f^*(x))^2$ is given by

$$L = \sigma_0 \left\| \sum_{i=1}^{m} \|w_i\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \sum_{i=1}^{m} w_i^{\otimes 2} \bar{w}_i^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Where $\bar{w}_i = \frac{w_i}{\|w_i\|_2}$ and $\sigma_i \approx \frac{1}{\mathrm{poly}(i)}$ (the Hermits coefficients of ReLU).
- At initialization, $w_i \sim \mathcal{N}(0, I/m)$.

The infinite neuron optimization process

- For simplicity, let us focus on the case when $r = d$, all a_i^* are positive and all a_i are positive (so $a_i = \|w_i\|_2$).
- Without the linear term, the population loss $L = \mathbb{E}_{x \sim \mathcal{N}(0,I)}(f(x) - f^*(x))^2$ is given by

$$L = \sigma_0 \left\| \sum_{i=1}^{m} \|w_i\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \sum_{i=1}^{m} w_i^{\otimes 2} \bar{w}_i^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Where $\bar{w}_i = \frac{w_i}{\|w_i\|_2}$ and $\sigma_i \approx \frac{1}{\text{poly}(i)}$ (the Hermits coefficients of ReLU).
- At initialization, $w_i \sim \mathcal{N}(0, I/m)$.
- Infinite neuron process: for a distribution P over $\mathcal{R}^d$, the (infinite neuron) population loss L_{inf} is given by

$$\sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

The infinite neuron optimization process

- For simplicity, let us focus on the case when $r = d$, all a_i^* are positive and all a_i are positive (so $a_i = \|w_i\|_2$).
- Without the linear term, the population loss $L = \mathbb{E}_{x \sim \mathcal{N}(0,I)}(f(x) - f^*(x))^2$ is given by

$$L = \sigma_0 \left\| \sum_{i=1}^{m} \|w_i\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \sum_{i=1}^{m} w_i^{\otimes 2} \bar{w}_i^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Where $\bar{w}_i = \frac{w_i}{\|w_i\|_2}$ and $\sigma_i \approx \frac{1}{\text{poly}(i)}$ (the Hermits coefficients of ReLU).
- At initialization, $w_i \sim \mathcal{N}(0, I/m)$.
- Infinite neuron process: for a distribution P over $\mathcal{R}^d$, the (infinite neuron) population loss L_{inf} is given by

$$\sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- At initialization, $P = \mathcal{N}(0, I)$.

The infinite neuron optimization process

- Infinite neuron optimization: Each step: update P via gradient descent (with learning rate η) over w on the infinite neuron loss:

$$w = w - \eta \nabla_w L_{\inf}$$

$$\nabla_w L_{\inf} = \sigma_0 \frac{\partial}{\partial v}\bigg|_{v=w} \left\| \left(\|v\|_2^2 - \|w\|_2^2 \right) + \mathbb{E}_{w' \sim P} \|w'\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2$$

$$+ \sum_{r=1}^{\infty} \sigma_{2r} \frac{\partial}{\partial v}\bigg|_{v=w} \left\| \left(v^{\otimes 2} \bar{v}^{\otimes 2r-2} - w^{\otimes 2} \bar{w}^{\otimes 2r-2} \right) + \mathbb{E}_{w' \sim P} (w')^{\otimes 2} \bar{w}'^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

The infinite neuron optimization process

- Infinite neuron optimization: Each step: update P via gradient descent (with learning rate η) over w on the infinite neuron loss:

$$w = w - \eta \nabla_w L_{\text{inf}}$$

$$\nabla_w L_{\text{inf}} = \sigma_0 \frac{\partial}{\partial v}\Big|_{v=w} \left\| \left(\|v\|_2^2 - \|w\|_2^2 \right) + \mathbb{E}_{w' \sim P} \|w'\|_2^2 - \textstyle\sum_{i=1}^d a_i^* \right\|_F^2$$

$$+ \textstyle\sum_{r=1}^{\infty} \sigma_{2r} \frac{\partial}{\partial v}\Big|_{v=w} \left\| \left(v^{\otimes 2} \bar{v}^{\otimes 2r-2} - w^{\otimes 2} \bar{w}^{\otimes 2r-2} \right) + \mathbb{E}_{w' \sim P} (w')^{\otimes 2} \bar{w}'^{\otimes 2r-2} - \textstyle\sum_{i=1}^d a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- The loss contains: Zero and second order term:

$$G = \sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^d a_i^* \right\|_F^2 + \sigma_2 \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} - \sum_{i=1}^d a_i^* (w_i^*)^{\otimes 2} \right\|_F^2$$

The infinite neuron optimization process

- Infinite neuron optimization: Each step: update P via gradient descent (with learning rate η) over w on the infinite neuron loss:

$$w = w - \eta \nabla_w L_{\text{inf}}$$

$$\nabla_w L_{\text{inf}} = \sigma_0 \frac{\partial}{\partial v}\bigg|_{v=w} \left\| \left(\|v\|_2^2 - \|w\|_2^2 \right) + \mathbb{E}_{w' \sim P} \|w'\|_2^2 - \sum_{i=1}^d a_i^* \right\|_F^2$$

$$+ \sum_{r=1}^\infty \sigma_{2r} \frac{\partial}{\partial v}\bigg|_{v=w} \left\| \left(v^{\otimes 2} \bar{v}^{\otimes 2r-2} - w^{\otimes 2} \bar{w}^{\otimes 2r-2} \right) + \mathbb{E}_{w' \sim P} (w')^{\otimes 2} \bar{w}'^{\otimes 2r-2} - \sum_{i=1}^d a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- The loss contains: Zero and second order term:

$$G = \sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^d a_i^* \right\|_F^2 + \sigma_2 \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} - \sum_{i=1}^d a_i^* (w_i^*)^{\otimes 2} \right\|_F^2$$

- These are PCA-type terms, can not identify w_i^* uniquely.

The infinite neuron optimization process

- Infinite neuron optimization: Each step: update P via gradient descent (with learning rate η) over w on the infinite neuron loss:

$$w = w - \eta \nabla_w L_{\text{inf}}$$

$$\nabla_w L_{\text{inf}} = \sigma_0 \frac{\partial}{\partial v}\Big|_{v=w} \left\| \left(\|v\|_2^2 - \|w\|_2^2 \right) + \mathbb{E}_{w' \sim P} \|w'\|_2^2 - \textstyle\sum_{i=1}^d a_i^* \right\|_F^2$$

$$+ \textstyle\sum_{r=1}^{\infty} \sigma_{2r} \frac{\partial}{\partial v}\Big|_{v=w} \left\| \left(v^{\otimes 2} \bar{v}^{\otimes 2r-2} - w^{\otimes 2} \bar{w}^{\otimes 2r-2} \right) + \mathbb{E}_{w' \sim P} (w')^{\otimes 2} \bar{w}'^{\otimes 2r-2} - \textstyle\sum_{i=1}^d a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- The loss contains: Zero and second order term:

$$G = \sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^d a_i^* \right\|_F^2 + \sigma_2 \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} - \sum_{i=1}^d a_i^* (w_i^*)^{\otimes 2} \right\|_F^2$$

- These are PCA-type terms, can not identify w_i^* uniquely.
- Forth order term:

$$H = \sigma_4 \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} \bar{w}^{\otimes 2} - \sum_{i=1}^d a_i^* (w_i^*)^{\otimes 4} \right\|_F^2$$

The infinite neuron optimization process

- Infinite neuron optimization: Each step: update P via gradient descent (with learning rate η) over w on the infinite neuron loss:

$$w = w - \eta \nabla_w L_{\text{inf}}$$

$$\nabla_w L_{\text{inf}} = \sigma_0 \frac{\partial}{\partial v}\Big|_{v=w} \left\| \left(\|v\|_2^2 - \|w\|_2^2 \right) + \mathbb{E}_{w' \sim P} \|w'\|_2^2 - \sum_{i=1}^d a_i^* \right\|_F^2$$
$$+ \sum_{r=1}^{\infty} \sigma_{2r} \frac{\partial}{\partial v}\Big|_{v=w} \left\| \left(v^{\otimes 2} \bar{v}^{\otimes 2r-2} - w^{\otimes 2} \bar{w}^{\otimes 2r-2} \right) + \mathbb{E}_{w' \sim P} (w')^{\otimes 2} \bar{w}'^{\otimes 2r-2} - \sum_{i=1}^d a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- The loss contains: Zero and second order term:

$$G = \sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^d a_i^* \right\|_F^2 + \sigma_2 \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} - \sum_{i=1}^d a_i^* (w_i^*)^{\otimes 2} \right\|_F^2$$

- These are PCA-type terms, can not identify w_i^* uniquely.
- Forth order term:

$$H = \sigma_4 \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} \bar{w}^{\otimes 2} - \sum_{i=1}^d a_i^* (w_i^*)^{\otimes 4} \right\|_F^2$$

- Tensor-type term, can identify w_i^* uniquely.

The infinite neuron optimization process

- Loss function:

$$\sigma_0 \left\| \mathbb{E}_{w\sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w\sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

The infinite neuron optimization process

- Loss function:

$$\sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Key observation:

The infinite neuron optimization process

- Loss function:

$$\sigma_0 \left\| \mathbb{E}_{w\sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w\sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Key observation:
  - Through out the optimization process, P has symmetric, in the sense that for every $i \in [d]$, conditional on $\mathcal{E}$: the value of $\langle w_j^*, w\rangle$ for every $j \neq i$ and the absolute value $|\langle w_i^*, w\rangle|$, then we still have

$$\Pr[\langle w_i^*, w\rangle \geq 0 \mid \mathcal{E}] = \Pr[\langle w_i^*, w\rangle \leq 0 \mid \mathcal{E}] = \frac{1}{2}$$

The infinite neuron optimization process

- Loss function:

$$\sigma_0 \left\| \mathbb{E}_{w\sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w\sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Key observation:
  - Through out the optimization process, P has symmetric, in the sense that for every $i \in [d]$, conditional on $\mathcal{E}$: the value of $\langle w_j^*, w\rangle$ for every $j \neq i$ and the absolute value $|\langle w_i^*, w\rangle|$, then we still have

$$\Pr[\langle w_i^*, w\rangle \geq 0 \mid \mathcal{E}] = \Pr[\langle w_i^*, w\rangle \leq 0 \mid \mathcal{E}] = \frac{1}{2}$$

  - This observation allows us to simplify many cross terms when calculating the gradient update.

Simplification with sign-symmetricity

- With the symmetric property of P, for example, we have:

Simplification with sign-symmetricity

- With the symmetric property of P, for example, we have:
- $$\mathbb{E}_{w \sim P} w^{\otimes 2} = \sum_{i=1}^{d} \mathbb{E}_{w \sim P} \left[\langle w_i^*, w \rangle^2 \right] (w_i^*)^{\otimes 2}$$

Simplification with sign-symmetricity

- With the symmetric property of P, for example, we have:
-
$$\mathbb{E}_{w\sim P} w^{\otimes 2} = \sum_{i=1}^{d} \mathbb{E}_{w\sim P}\left[\langle w_i^*, w\rangle^2\right](w_i^*)^{\otimes 2}$$
-
$$\mathbb{E}_{w\sim P} w^{\otimes 2}\bar{w}^{\otimes 2} = \sum_{i,j=1}^{d} \mathbb{E}_{w\sim P}\left[\langle w_i^*, w\rangle^2 \langle w_j^*, \bar{w}\rangle^2\right](w_i^*)^{\otimes 2}(w_j^*)^{\otimes 2}$$

The infinite neuron optimization process

- Loss function:

$$\sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

The infinite neuron optimization process

- Loss function:

$$\sigma_0 \left\| \mathbb{E}_{w\sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w\sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Optimization process:

The infinite neuron optimization process

- Loss function:

$$\sigma_0 \left\| \mathbb{E}_{w\sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w\sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Optimization process:
  - Using the key observation, it is not hard to show:

The infinite neuron optimization process

- Loss function:

$$\sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Optimization process:
  - Using the key observation, it is not hard to show:
  - After a small number of iterations, G (0, 2 order term, e.g. PCA term) will become small (smaller than 4-order term).

The infinite neuron optimization process

- Loss function:

$$\sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Optimization process:
  - Using the key observation, it is not hard to show:
  - After a small number of iterations, G (0, 2 order term, e.g. PCA term) will become small (smaller than 4-order term).
  - The optimization process then enters the tensor phase and the tensor decomposition process over H (the 4-order term e.g. tensor term) thus can identify w_i^* correctly.

The infinite neuron optimization process

- Loss function:

$$\sigma_0 \left\| \mathbb{E}_{w\sim P}\|w\|_2^2 - \sum_{i=1}^{d} a_i^* \right\|_F^2 + \sum_{r=1}^{\infty} \sigma_{2r} \left\| \mathbb{E}_{w\sim P} w^{\otimes 2} \bar{w}^{\otimes 2r-2} - \sum_{i=1}^{d} a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- Optimization process:
  - Using the key observation, it is not hard to show:
  - After a small number of iterations, G (0, 2 order term, e.g. PCA term) will become small (smaller than 4-order term).
  - The optimization process then enters the tensor phase and the tensor decomposition process over H (the 4-order term e.g. tensor term) thus can identify w_i^* correctly.
  - So the entire infinite neuron process is a mixture of PCA and tensor decomposition, which is highly non-convex (not NTK).

From infinite to finite (the naive way)

- How do we move from infinite process to poly-size finite neuron process?

From infinite to finite (the naive way)

- How do we move from infinite process to poly-size finite neuron process?
- Approach 1: $\{w_i\}_{i=1}^{m}$ is close to P in e.g. Wasserstein distance?

From infinite to finite (the naive way)

- How do we move from infinite process to poly-size finite neuron process?
- Approach 1: $\{w_i\}_{i=1}^m$ is close to P in e.g. Wasserstein distance?
- Impossible! Need at least $2^{\Omega(d)}$ neurons, but we focus on $m = \mathrm{poly}(d)$.

From infinite to finite (the naive way)

- How do we move from infinite process to poly-size finite neuron process?
- Approach 1: $\{w_i\}_{i=1}^m$ is close to P in e.g. Wasserstein distance?
- Impossible! Need at least $2^{\Omega(d)}$ neurons, but we focus on $m = \text{poly}(d)$.
- Approach 2: Finite neuron can preserve the key observation?

From infinite to finite (the naive way)

- How do we move from infinite process to poly-size finite neuron process?
- Approach 1: $\{w_i\}_{i=1}^m$ is close to P in e.g. Wasserstein distance?
- Impossible! Need at least $2^{\Omega(d)}$ neurons, but we focus on $m = \text{poly}(d)$.
- Approach 2: Finite neuron can preserve the key observation?
- Through out the optimization process, $\{w_i\}_{i=1}^m$ is (approximately) symmetric? in the sense that for every $r \in [d]$, conditional on $\mathcal{E}$: the value of $\langle w_j^*, w_i \rangle$ for every $j \neq r$ and the absolute value $|\langle w_r^*, w_i \rangle|$, then

$$\Pr_{i \sim Uniform[m]}[\langle w_r^*, w_i \rangle \geq 0 \mid \mathcal{E}] \approx \Pr_{i \sim Uniform[m]}[\langle w_r^*, w_i \rangle \leq 0 \mid \mathcal{E}] \approx \frac{1}{2}$$

From infinite to finite (the naive way)

- How do we move from infinite process to poly-size finite neuron process?
- Approach 1: $\{w_i\}_{i=1}^m$ is close to P in e.g. Wasserstein distance?
- Impossible! Need at least $2^{\Omega(d)}$ neurons, but we focus on $m = \text{poly}(d)$.
- Approach 2: Finite neuron can preserve the key observation?
- Through out the optimization process, $\{w_i\}_{i=1}^m$ is (approximately) symmetric? in the sense that for every $r \in [d]$, conditional on $\mathcal{E}$: the value of $\langle w_j^*, w_i \rangle$ for every $j \neq r$ and the absolute value $|\langle w_r^*, w_i \rangle|$, then

$$\Pr_{i \sim Uniform[m]}[\langle w_r^*, w_i \rangle \geq 0 \mid \mathcal{E}] \approx \Pr_{i \sim Uniform[m]}[\langle w_r^*, w_i \rangle \leq 0 \mid \mathcal{E}] \approx \frac{1}{2}$$

- Impossible! Need at least $2^{\Omega(d)}$ neurons, but we focus on $m = \text{poly}(d)$.

From infinite to finite (the actual way)

- Infinite neuron dynamic for each w:

$$w \to T_1(w) \to T_2(w) \to \cdots \to T_t(w), \quad \text{via } w = w - \eta \nabla_w L_{\text{inf}}$$

From infinite to finite (the actual way)

- Infinite neuron dynamic for each w:

$$w \to T_1(w) \to T_2(w) \to \cdots \to T_t(w), \quad \text{via } w = w - \eta \nabla_w L_{\text{inf}}$$

- Finite neuron dynamic for each w_i:

$$w_i \to \tilde{T}_1(w_i) \to \tilde{T}_2(w_i) \to \cdots \to \tilde{T}_t(w_i), \quad \text{via } w_i = w_i - \eta \nabla_{w_i} \tilde{L}$$

$$\tilde{L} = \frac{1}{N} \sum_{j=1}^{N} (f(x_j) - f^*(x_j))^2$$

From infinite to finite (the actual way)

- Infinite neuron dynamic for each w:

$$w \to T_1(w) \to T_2(w) \to \cdots \to T_t(w), \quad \text{via } w = w - \eta \nabla_w L_{\text{inf}}$$

- Finite neuron dynamic for each w_i:

$$w_i \to \tilde{T}_1(w_i) \to \tilde{T}_2(w_i) \to \cdots \to \tilde{T}_t(w_i), \quad \text{via } w_i = w_i - \eta \nabla_{w_i} \tilde{L}$$

$$\tilde{L} = \frac{1}{N} \sum_{j=1}^{N} (f(x_j) - f^*(x_j))^2$$

- Question: (ignoring the $\sqrt{m}$ multiplicative scaling between the norm of w_i and w) if $w = w_i$, then is $T_t(w)$ close to $\tilde{T}_t(w_i)$?

From infinite to finite, one step

- The error between $w \to T_1(w)$ and $w_i \to \tilde{T}_1(w_i)$:

From infinite to finite, one step

- The error between $w \to T_1(w)$ and $w_i \to \tilde{T}_1(w_i)$:
- Three steps: $w_i \to T_1(w_i) \to \hat{T}_1(w_i) \to \tilde{T}_1(w_i)$.

From infinite to finite, one step

- The error between $w \to T_1(w)$ and $w_i \to \tilde{T}_1(w_i)$:
- Three steps: $w_i \to T_1(w_i) \to \hat{T}_1(w_i) \to \tilde{T}_1(w_i)$.
- $T_1(w_i) \to \hat{T}_1(w_i)$: over-parameterization error: the error caused by moving from P to the "empirical version" $\{w_i\}_{i=1}^m$: e.g. the error between $\mathbb{E}_{w\sim P} w^{\otimes 2}$ and $\frac{1}{m}\sum_{i=1}^m w_i^{\otimes 2}$ where each w_i i.i.d. $w_i \sim P$.

From infinite to finite, one step

- The error between $w \to T_1(w)$ and $w_i \to \tilde{T}_1(w_i)$:
- Three steps: $w_i \to T_1(w_i) \to \hat{T}_1(w_i) \to \tilde{T}_1(w_i)$.
- $T_1(w_i) \to \hat{T}_1(w_i)$: over-parameterization error: the error caused by moving from P to the "empirical version" $\{w_i\}_{i=1}^m$: e.g. the error between $\mathbb{E}_{w\sim P} w^{\otimes 2}$ and $\frac{1}{m}\sum_{i=1}^m w_i^{\otimes 2}$ where each w_i i.i.d. $w_i \sim P$.
- $\hat{T}_1(w_i) \to \tilde{T}_1(w_i)$: sample error, the error caused by moving from population loss to empirical loss, e.g. the error between L and $\tilde{L}$.

From infinite to finite, one step

- The error between $w \to T_1(w)$ and $w_i \to \tilde{T}_1(w_i)$:
- Three steps: $w_i \to T_1(w_i) \to \hat{T}_1(w_i) \to \tilde{T}_1(w_i)$.
- $T_1(w_i) \to \hat{T}_1(w_i)$: over-parameterization error: the error caused by moving from P to the "empirical version" $\{w_i\}_{i=1}^m$: e.g. the error between $\mathbb{E}_{w \sim P} w^{\otimes 2}$ and $\frac{1}{m}\sum_{i=1}^m w_i^{\otimes 2}$ where each w_i i.i.d. $w_i \sim P$.
- $\hat{T}_1(w_i) \to \tilde{T}_1(w_i)$: sample error, the error caused by moving from population loss to empirical loss, e.g. the error between L and $\tilde{L}$.
- The key observation:

$$|T_1(w_i) - \hat{T}_1(w_i)| \approx \frac{1}{\sqrt{m}}, \quad |\hat{T}_1(w_i) - \tilde{T}_1(w_i)| \approx \frac{1}{\sqrt{N}}$$

From infinite to finite, one step

- The error between $w \to T_1(w)$ and $w_i \to \tilde{T}_1(w_i)$:
- Three steps: $w_i \to T_1(w_i) \to \hat{T}_1(w_i) \to \tilde{T}_1(w_i)$.
- $T_1(w_i) \to \hat{T}_1(w_i)$: over-parameterization error: the error caused by moving from P to the "empirical version" $\{w_i\}_{i=1}^m$: e.g. the error between $\mathbb{E}_{w\sim P} w^{\otimes 2}$ and $\frac{1}{m}\sum_{i=1}^m w_i^{\otimes 2}$ where each w_i i.i.d. $w_i \sim P$.
- $\hat{T}_1(w_i) \to \tilde{T}_1(w_i)$: sample error, the error caused by moving from population loss to empirical loss, e.g. the error between L and $\tilde{L}$.
- The key observation:

$$|T_1(w_i) - \hat{T}_1(w_i)| \approx \frac{1}{\sqrt{m}}, \quad |\hat{T}_1(w_i) - \tilde{T}_1(w_i)| \approx \frac{1}{\sqrt{N}}$$

- $\frac{1}{\sqrt{m}}$: The role of (poly-size) over-parameterization.

From infinite to finite, the entire process

- The error between $w \to T_t(w)$ and $w_i \to \tilde{T}_t(w_i)$:

From infinite to finite, the entire process

- The error between $w \to T_t(w)$ and $w_i \to \tilde{T}_t(w_i)$:
- Fundamental question: if $\| \tilde{T}_{t-1}(w_i) - T_{t-1}(w_i) \|_2 \leq \varepsilon$ for every i, then whether:

From infinite to finite, the entire process

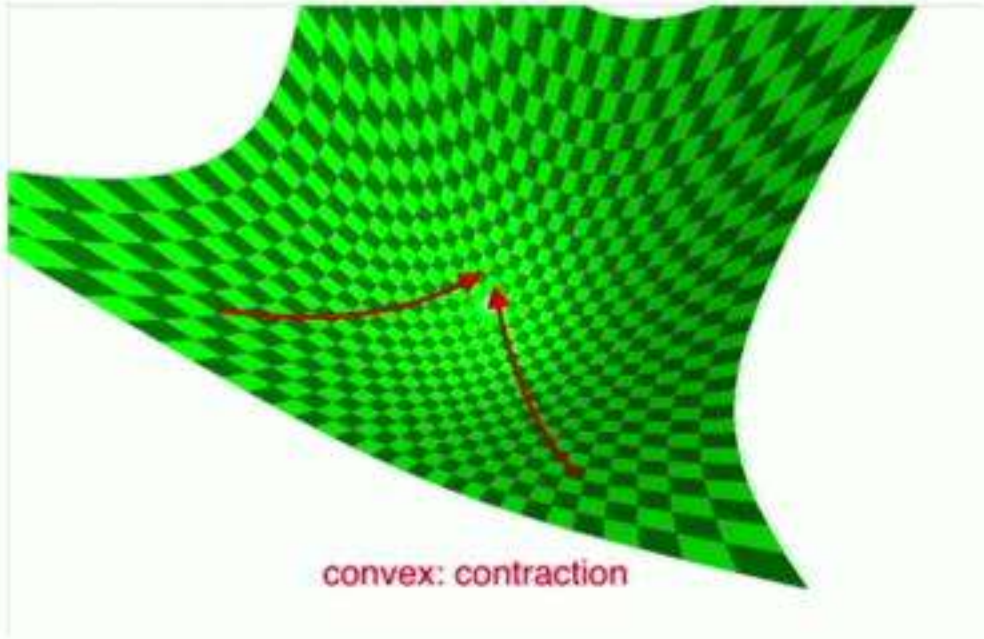
- The error between $w \to T_t(w)$ and $w_i \to \tilde{T}_t(w_i)$:
- Fundamental question: if $\|\tilde{T}_{t-1}(w_i) - T_{t-1}(w_i)\|_2 \le \varepsilon$ for every i, then whether:
  - Contraction? $\|\tilde{T}_t(w_i) - T_t(w_i)\|_2 \le \varepsilon$ (typical for convex process).

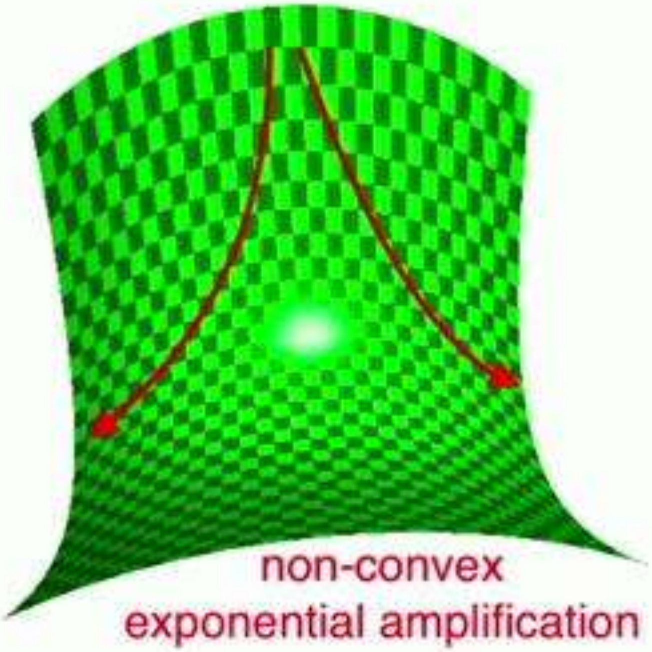
From infinite to finite, the entire process

- The error between $w \to T_t(w)$ and $w_i \to \tilde{T}_t(w_i)$:
- Fundamental question: if $\| \tilde{T}_{t-1}(w_i) - T_{t-1}(w_i) \|_2 \le \varepsilon$ for every i, then whether:
  - Contraction? $\| \tilde{T}_t(w_i) - T_t(w_i) \|_2 \le \varepsilon$ (typical for convex process).
  - Mild amplification? $\| T_t(w_i) - T_t(w_i) \|_2 \approx (1 + \eta/\text{poly}(d))\varepsilon$, where η is the learning rate of the SGD.

From infinite to finite, the entire process

- The error between $w \to T_t(w)$ and $w_i \to \tilde{T}_t(w_i)$:
- Fundamental question: if $\|\tilde{T}_{t-1}(w_i) - T_{t-1}(w_i)\|_2 \leq \varepsilon$ for every i, then whether:
  - Contraction? $\|\tilde{T}_t(w_i) - T_t(w_i)\|_2 \leq \varepsilon$ (typical for convex process).
  - Mild amplification? $\|T_t(w_i) - T_t(w_i)\|_2 \approx (1 + \eta/\mathrm{poly}(d))\varepsilon$, where η is the learning rate of the SGD.
  - Exponential amplification? $\|T_t(w_i) - T_t(w_i)\|_2 \approx (1 + \eta)\varepsilon$ (can happen for non-convex process). (If so, m, N has to be exponential in d)





From infinite to finite, the entire process

- The infinite neuron process is a mixture of PCA and tensor decomposition, which is highly non-convex.

From infinite to finite, the entire process

- The infinite neuron process is a mixture of PCA and tensor decomposition, which is highly non-convex.

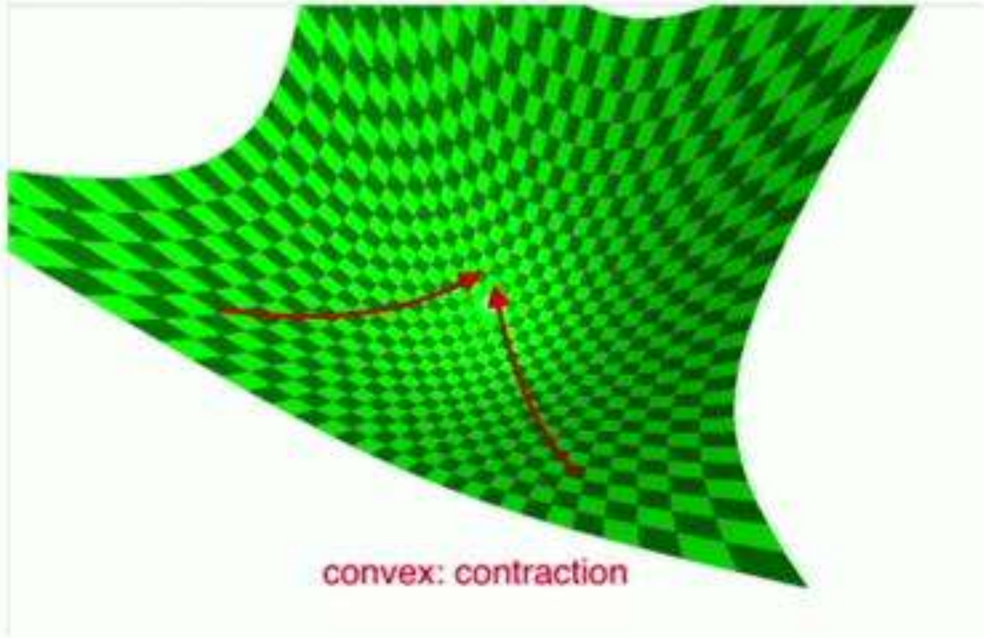
Lemma (Critical, sketched)

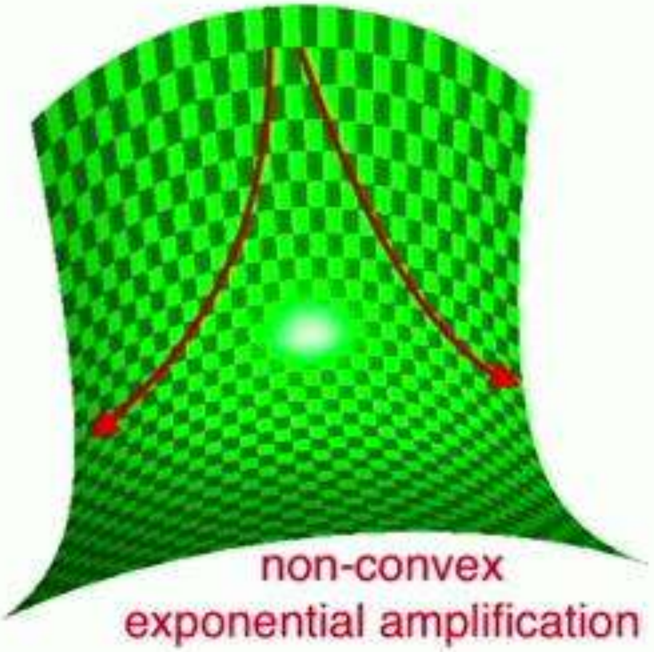
$T_t(w) - \tilde{T}_t(w)$ is an exponential amplification for an arbitrary w (due to non-convexity), but with high probability for w_i sampled from spherical Gaussian, $T_t(w_i) - \tilde{T}_t(w_i)$ is a mild amplification as long as $\varepsilon \le \frac{1}{poly(d)}$.

-

From infinite to finite, the entire process

- The error between $w \to T_t(w)$ and $w_i \to \tilde{T}_t(w_i)$:
- Fundamental question: if $\|\tilde{T}_{t-1}(w_i) - T_{t-1}(w_i)\|_2 \leq \varepsilon$ for every i, then whether:
    - Contraction? $\|\tilde{T}_t(w_i) - T_t(w_i)\|_2 \leq \varepsilon$ (typical for convex process).
    - Mild amplification? $\|T_t(w_i) - T_t(w_i)\|_2 \approx (1 + \eta/\text{poly}(d))\varepsilon$, where η is the learning rate of the SGD.
    - Exponential amplification? $\|T_t(w_i) - T_t(w_i)\|_2 \approx (1 + \eta)\varepsilon$ (can happen for non-convex process). (If so, m, N has to be exponential in d)



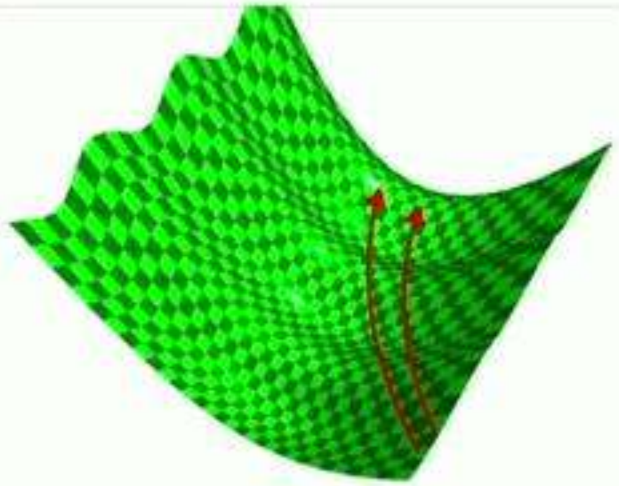


From infinite to finite, the entire process

- The infinite neuron process is a mixture of PCA and tensor decomposition, which is highly non-convex.

Lemma (Critical, sketched)

$T_t(w) - \tilde{T}_t(w)$ is an exponential amplification for an arbitrary w (due to non-convexity), but with high probability for w_i sampled from spherical Gaussian, $T_t(w_i) - \tilde{T}_t(w_i)$ is a mild amplification as long as $\varepsilon \leq \frac{1}{poly(d)}$.

- 
-

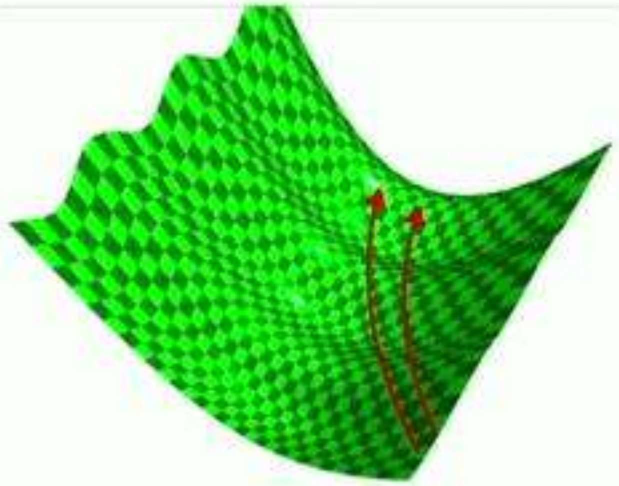
From infinite to finite, the entire process

- The infinite neuron process is a mixture of PCA and tensor decomposition, which is highly non-convex.

Lemma (Critical, sketched)

$T_t(w) - \tilde{T}_t(w)$ is an exponential amplification for an arbitrary w (due to non-convexity), but with high probability for w_i sampled from spherical Gaussian, $T_t(w_i) - \tilde{T}_t(w_i)$ is a mild amplification as long as $\varepsilon \leq \frac{1}{poly(d)}$.

-



-
- Together with the infinite neuron process result, this gives the final theorem.

Summary: the most simple example of over-parameterization

- We show that running SGD on a over-parameterized two-layer neural network with ReLU activation can learn a smaller two-layer neural network with ReLU activation, where the hidden weights are orthonormal, and the data distribution is spherical Gaussian.

Summary: the most simple example of over-parameterization

- We show that running SGD on a over-parameterized two-layer neural network with ReLU activation can learn a smaller two-layer neural network with ReLU activation, where the hidden weights are orthonormal, and the data distribution is spherical Gaussian.
  - Poly-size over-parameterization is actually simulating the infinite neuron optimization process.

Summary: the most simple example of over-parameterization

- We show that running SGD on a over-parameterized two-layer neural network with ReLU activation can learn a smaller two-layer neural network with ReLU activation, where the hidden weights are orthonormal, and the data distribution is spherical Gaussian.
    - Poly-size over-parameterization is actually simulating the infinite neuron optimization process.
    - Infinite neuron optimization process has a non-convex but benign landscape (a mixture of PCA and tensor decomposition, with symmetric property and mild-amplification property).

Summary: the most simple example of over-parameterization

- We show that running SGD on a over-parameterized two-layer neural network with ReLU activation can learn a smaller two-layer neural network with ReLU activation, where the hidden weights are orthonormal, and the data distribution is spherical Gaussian.
  - Poly-size over-parameterization is actually simulating the infinite neuron optimization process.
  - Infinite neuron optimization process has a non-convex but benign landscape (a mixture of PCA and tensor decomposition, with symmetric property and mild-amplification property).
- When the hidden weights are not orthonormal, it can require $2^{\Omega(d)}$ queries in the worst case $[VW'19]$ for any SQ model.

The infinite neuron optimization process

- Infinite neuron optimization: Each step: update P via gradient descent (with learning rate η) over w on the infinite neuron loss:

$$w = w - \eta \nabla_w L_{\inf}$$

$$\nabla_w L_{\inf} = \sigma_0 \frac{\partial}{\partial v}\Big|_{v=w} \left\| \left(\|v\|_2^2 - \|w\|_2^2 \right) + \mathbb{E}_{w' \sim P} \|w'\|_2^2 - \textstyle\sum_{i=1}^d a_i^* \right\|_F^2$$
$$+ \textstyle\sum_{r=1}^{\infty} \sigma_{2r} \frac{\partial}{\partial v}\Big|_{v=w} \left\| \left(v^{\otimes 2} \bar{v}^{\otimes 2r-2} - w^{\otimes 2} \bar{w}^{\otimes 2r-2} \right) + \mathbb{E}_{w' \sim P} (w')^{\otimes 2} \bar{w}'^{\otimes 2r-2} - \textstyle\sum_{i=1}^d a_i^* (w_i^*)^{\otimes 2r} \right\|_F^2$$

- The loss contains: Zero and second order term:

$$G = \sigma_0 \left\| \mathbb{E}_{w \sim P} \|w\|_2^2 - \sum_{i=1}^d a_i^* \right\|_F^2 + \sigma_2 \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} - \sum_{i=1}^d a_i^* (w_i^*)^{\otimes 2} \right\|_F^2$$

- These are PCA-type terms, can not identify w_i^* uniquely.
- Forth order term:

$$H = \sigma_4 \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} \bar{w}^{\otimes 2} - \sum_{i=1}^d a_i^* (w_i^*)^{\otimes 4} \right\|_F^2$$

The infinite neuron optimization process

- Infinite neuron optimization: Each step: update P via gradient descent (with learning rate η) over w on the infinite neuron loss:

$$w = w - \eta \nabla_w L_{\text{inf}}$$

$$\nabla_w L_{\text{inf}} = \sigma_0 \frac{\partial}{\partial v}\Big|_{v=w} \left\| \left(\|v\|_2^2 - \|w\|_2^2\right) + \mathbb{E}_{w' \sim P}\|w'\|_2^2 - \sum_{i=1}^d a_i^* \right\|_F^2$$
$$+ \sum_{r=1}^{\infty} \sigma_{2r} \frac{\partial}{\partial v}\Big|_{v=w} \left\| \left(v^{\otimes 2}\bar{v}^{\otimes 2r-2} - w^{\otimes 2}\bar{w}^{\otimes 2r-2}\right) + \mathbb{E}_{w' \sim P}(w')^{\otimes 2}\bar{w}'^{\otimes 2r-2} - \sum_{i=1}^d a_i^*(w_i^*)^{\otimes 2r} \right\|_F^2$$

- The loss contains: Zero and second order term:

$$G = \sigma_0 \left\| \mathbb{E}_{w \sim P}\|w\|_2^2 - \sum_{i=1}^d a_i^* \right\|_F^2 + \sigma_2 \left\| \mathbb{E}_{w \sim P} w^{\otimes 2} - \sum_{i=1}^d a_i^*(w_i^*)^{\otimes 2} \right\|_F^2$$

- These are PCA-type terms, can not identify w_i^* uniquely.
- Forth order term:

$$H = \sigma_4 \left\| \mathbb{E}_{w \sim P} w^{\otimes 2}\bar{w}^{\otimes 2} - \sum_{i=1}^d a_i^*(w_i^*)^{\otimes 4} \right\|_F^2$$

Memorization Capacity of ReLU Nets

SUVRIT SRA
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

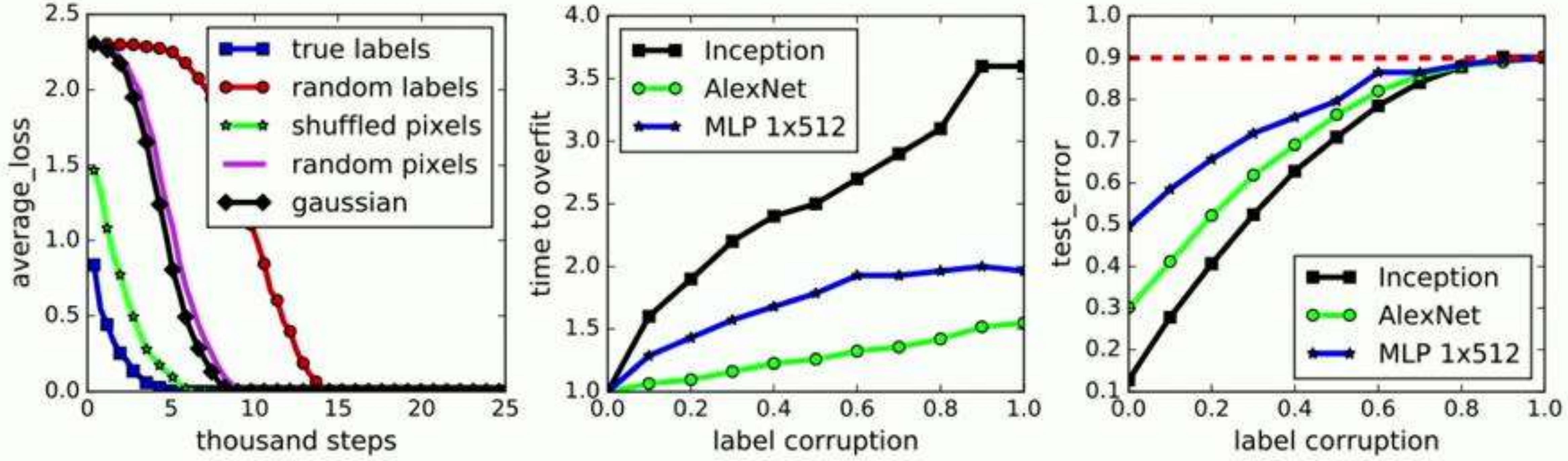
Chulhee Yun

Ali Jadbabaie

ml.mit.edu

The Memorization Phenomenon

- Overparametrized NNs trained with SGD can memorize even random noise *[Zhang et al., 2017]*

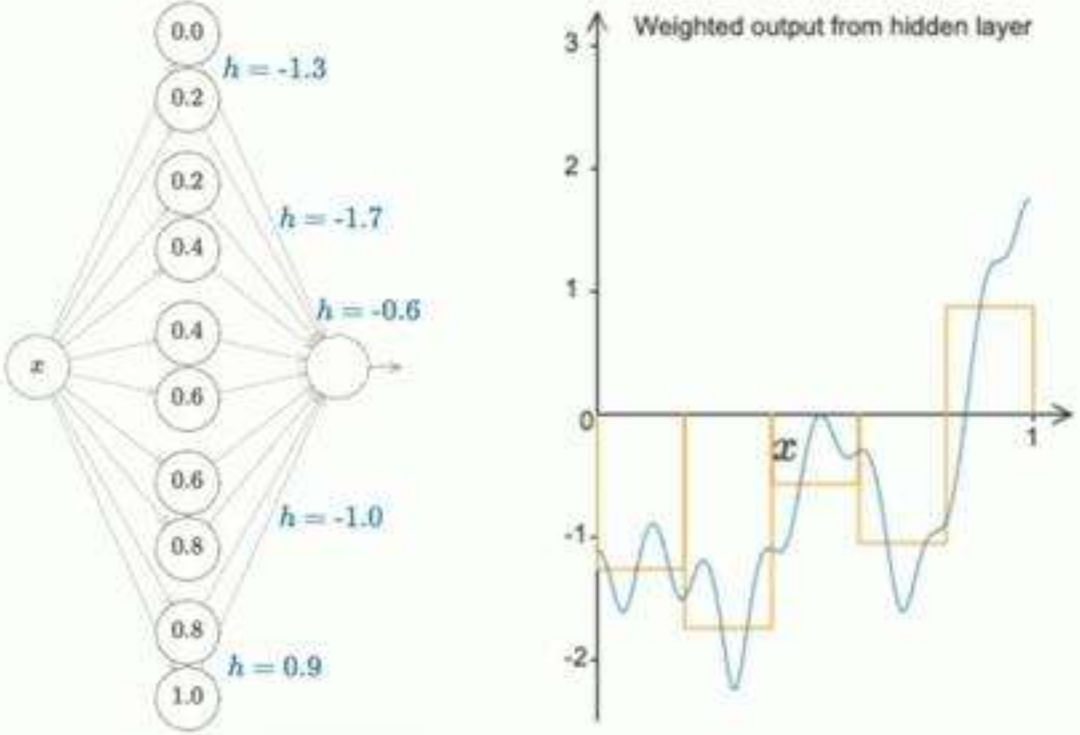


(a) learning curves (b) convergence slowdown (c) generalization error growth

Figure 1: Fitting random labels and random pixels on CIFAR10. (a) shows the training loss of various experiment settings decaying with the training steps. (b) shows the relative convergence time with different label corruption ratio. (c) shows the test error (also the generalization error since training error is 0) under different label corruptions.

Expressive power of NNs

- To understand memorization phenomenon, it is important to understand *expressive power*
- Expressive power a classic topic in NN theory; *universal approximation theory* *[Cybenko, '89, Hornik '91, Leshno '93, …]*

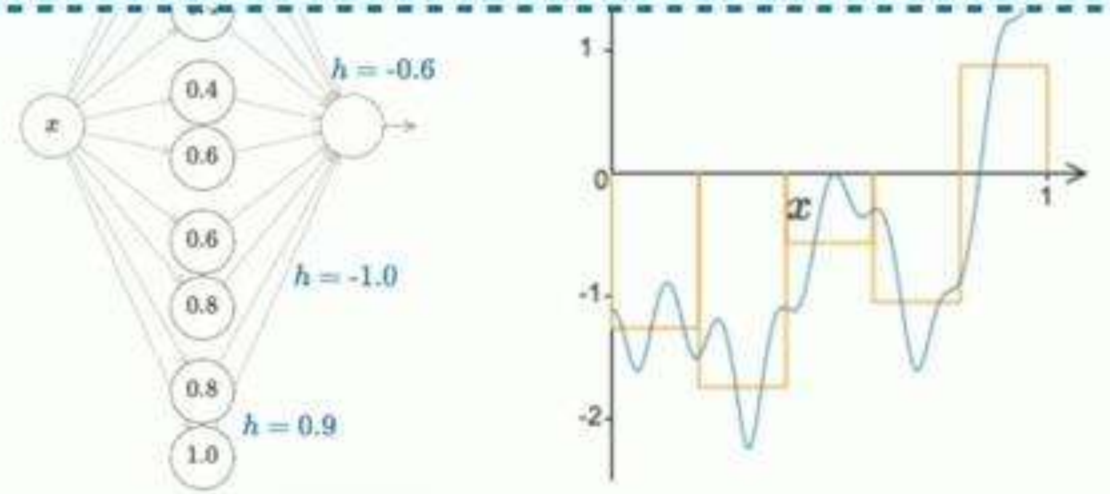


(http://neuralnetworksanddeeplearning.com/chap4.html)

Expressive power of NNs

- To understand memorization phenomenon, it is important to understand *expressive power*
- Expressive power a classic topic in NN theory; *universal approximation theory* *[Cybenko, '89, Hornik '91, Leshno '93, …]*

Majority of results consider function approximation (infinite points),
Not finite samples!



(http://neuralnetworksanddeeplearning.com/chap4.html)

Finite sample expressivity

Defn. We define (universal) **finite sample expressivity** of a neural network $f_{\boldsymbol{\theta}}(\cdot)$ as the network's ability to satisfy:

For arbitrary $\{(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}^p\}_{i=1}^N$ there exists a parameter $\boldsymbol{\theta}$ such that $f_{\boldsymbol{\theta}}(x_i) = y_i$ for $1 \leqslant i \leqslant N$.

Finite sample expressivity

Defn. We define (universal) **finite sample expressivity** of a neural network $f_{\boldsymbol{\theta}}(\,\cdot\,)$ as the network's ability to satisfy:

For arbitrary $\{(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}^p\}_{i=1}^N$ there exists a parameter $\boldsymbol{\theta}$ such that $f_{\boldsymbol{\theta}}(x_i) = y_i$ for $1 \leqslant i \leqslant N$.

That is, the net can memorize an arbitrary dataset with *(input, output)* points.

Memorization capacity

Defn. For $p = 1$, we define **memorization capacity** to be the **maximum value** of N for which the network has finite sample expressivity.

That is, the maximum N, s.t. for any $\{(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^N$, there exists a parameter $\boldsymbol{\theta}$ such that $f_{\boldsymbol{\theta}}(x_i) = y_i$.

Comparison to VC dimension

- **Memorization capacity**:

The maximum N such that **for all** $\{(x_i, y_i)\}_{i=1}^N$, there exists a parameter $\boldsymbol{\theta}$ such that $f_{\boldsymbol{\theta}}(x_i) = y_i$ *(recall 'p'=1 here).*

- **VC dimension:**

The maximum N for which **there exists** $\{x_i\}_{i=1}^N$, such that for any $(y_i \in \{\pm 1\})_{i=1}^N$, there exists a parameter $\boldsymbol{\theta}$ s.t. $f_{\boldsymbol{\theta}}(x_i) = y_i$.

Comparison to VC dimension

- **Memorization capacity**:

  The maximum N such that **for all** $\{(x_i, y_i)\}_{i=1}^{N}$, there exists a parameter $\boldsymbol{\theta}$ such that $f_{\boldsymbol{\theta}}(x_i) = y_i$ *(recall 'p'=1 here).*

- **VC dimension:**

  The maximum N for which **there exists** $\{x_i\}_{i=1}^{N}$, such that for any $(y_i \in \{\pm 1\})_{i=1}^{N}$, there exists a parameter $\boldsymbol{\theta}$ s.t. $f_{\boldsymbol{\theta}}(x_i) = y_i$.

⇒ **Memorization capacity ≤ VC dimension**

Related work on memorization

Classical works

- focus on memorization capacity of NNs with activations such as **linear threshold** or **sigmoid** *[Cover, 1965; Baum, 1988; Huang & Huang, 1991; Huang & Babri, 1998; Huang, 2003; etc…]*

Related work on memorization

Classical works

- focus on memorization capacity of NNs with activations such as **linear threshold** or **sigmoid** *[Cover, 1965; Baum, 1988; Huang & Huang, 1991; Huang & Babri, 1998; Huang, 2003; etc…]*

Recent Results

- ReLU fully-connected NNs (FNNs) *[Zhang et al., 2017]*
- Residual networks (ResNets) *[Hardt & Ma, 2017]*
- Convolutional neural networks (CNNs) *[Nguyen & Hein, 2017]*

Limitations of previous work

- Recent results impose strong assumptions on the number of hidden nodes!

Limitations of previous work

- Recent results impose strong assumptions on the number of hidden nodes!
- A 1-hidden-layer ReLU network with N **hidden nodes** can memorize any arbitrary dataset with N points.
  [Zhang et al., 2017]

Limitations of previous work

- Recent results impose strong assumptions on the number of hidden nodes!
- A 1-hidden-layer ReLU network with N **hidden nodes** can memorize any arbitrary dataset with N points.
  [Zhang et al., 2017]
- Results on ResNets and CNNs require N **hidden nodes**.
  [Hardt & Ma, 2017, Nguyen & Hein, 2017]

Limitations of previous work

- Recent results impose strong assumptions on the number of hidden nodes!

- A 1-hidden-layer ReLU network with N **hidden nodes** can memori
  [Zhang et

- Results **en nodes**.
  [Hardt & M

Can we use depth to memorize with fewer hidden nodes?

Outline

Tight memorization capacity of FCNNs

number of hidden nodes necessary and sufficient for universal memorization

Memorization capacity of Resnets

*number of hidden nodes **sufficient** for universal memorization*

Behavior of SGD near memorizing global min

*Analysis of **without replacement** SGD near global min*

Finite sample expressivity of FCNNs

Setup

Finite sample expressivity of FCNNs

Setup

Training data: $\{(x_i, y_i)\}_{i=1}^N, \;\; x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^p$

Finite sample expressivity of FCNNs

Setup

Training data: $\{(x_i, y_i)\}_{i=1}^{N}$, $x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^p$

Assumption: all x_i's are distinct and all $y_i \in [-1,1]$

Finite sample expressivity of FCNNs

Setup

Training data: $\{(x_i, y_i)\}_{i=1}^{N}, \;\; x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^p$

Assumption: all x_i's are distinct and all $y_i \in [-1,1]$

FCNN architecture:

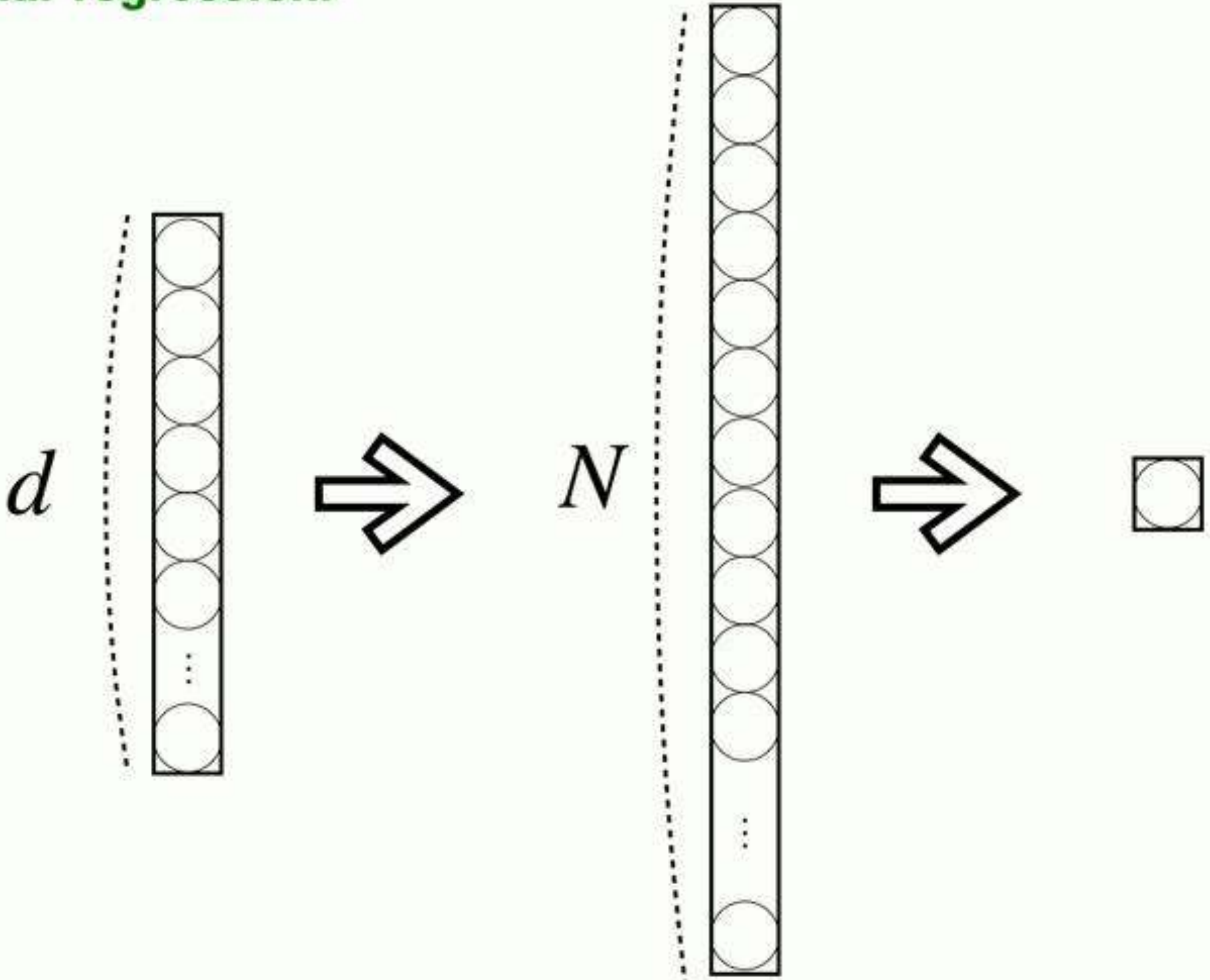
$$a^0(x) = x, \;\; a^l(x) = \sigma(W^l a^{l-1}(x) + b^l), \;\; l \in \{1,\ldots,L-1\}, \;\; f_{\theta}(x) = W^L a^{L-1}(x) + b^L$$

Activation function   Weight matrix   Bias vector

Activation $\sigma(t) = \max\{s_+ t, s_- t\}, \; s_+ > s_- \geq 0.$
(includes ReLU and Leaky ReLU)

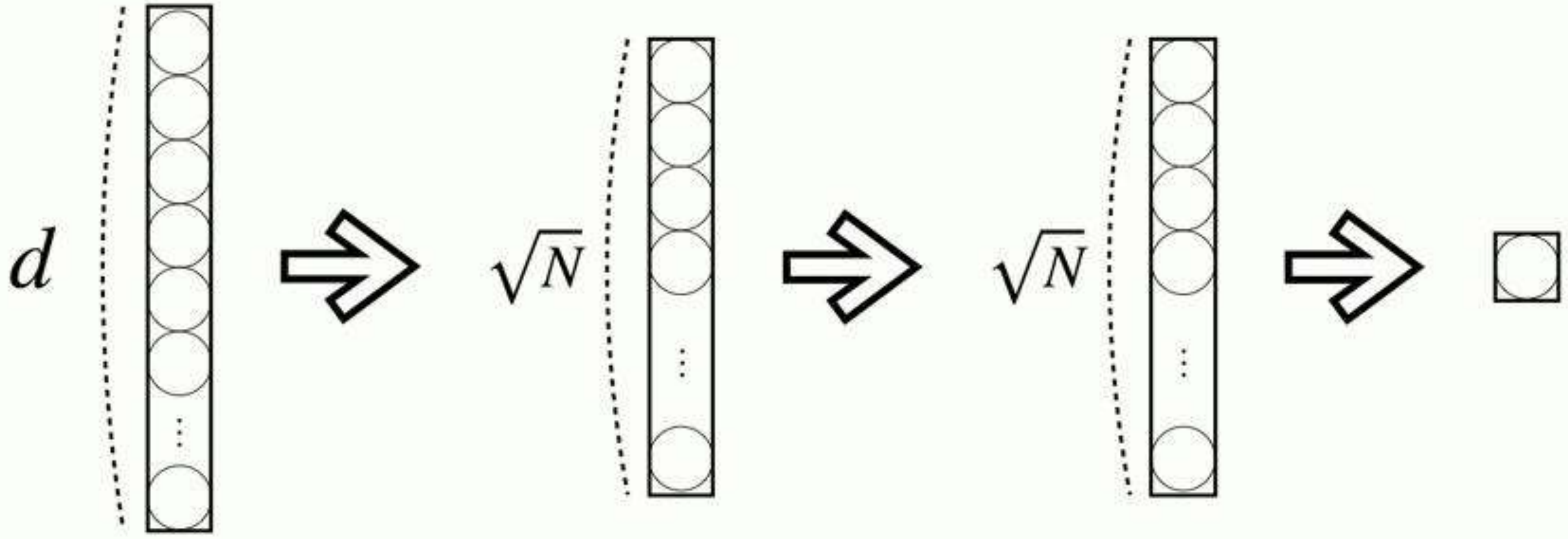
Finite sample expressivity of FCNNs

For scalar regression:



Finite sample expressivity of FCNNs

For scalar regression:



Memorization: sufficiency results

Theorem 1.

A 2-hidden-layer ReLU network with hidden layer dims $d_1 d_2 \geq 4Np$ can memorize arbitrary datasets with N distinct points. *(Recall 'p' is output dim.)*

Memorization: sufficiency results

Theorem 1.

A 2-hidden-layer ReLU network with hidden layer dims $d_1 d_2 \geq 4Np$ can memorize arbitrary datasets with N distinct points. *(Recall 'p' is output dim.)*

Proposition 2 (classification).

A 3-hidden-layer ReLU network with hidden layer dimensions $d_1 d_2 \geq 4N$ and $d_3 \geq 4p$ can memorize any arbitrary classification dataset with N distinct points *(here 'p' is the number of classes)*

Memorization: necessity result

Theorem 3.

A 1-hidden-layer ReLU network with $d_1 + 2 < N$, or
a 2-hidden-layer ReLU network with $2d_1d_2 + d_2 + 2 < N$
cannot memorize arbitrary datasets ($p = 1$) with N points.
(i.e., there exist datasets that they fail to memorize)

Discussion

Depth-width tradeoff in finite sample expressivity

Discussion

Depth-width tradeoff in finite sample expressivity

Necessary and sufficient width for memorizing ($p = 1$): 1-hidden-layer $\Theta(N)$ vs 2-hidden-layers $\Theta(\sqrt{N})$

Discussion

Depth-width tradeoff in finite sample expressivity

Necessary and sufficient width for memorizing ($p = 1$): 1-hidden-layer $\Theta(N)$ vs 2-hidden-layers $\Theta(\sqrt{N})$

For p-class classification, $\Theta(\sqrt{Np})$ requirement of 2-hidden-layer improves to $\Theta(\sqrt{N} + p)$ by adding another layer

Discussion

Depth-width tradeoff in finite sample expressivity

Necessary and sufficient width for memorizing ($p = 1$): 1-hidden-layer $\Theta(N)$ vs 2-hidden-layers $\Theta(\sqrt{N})$

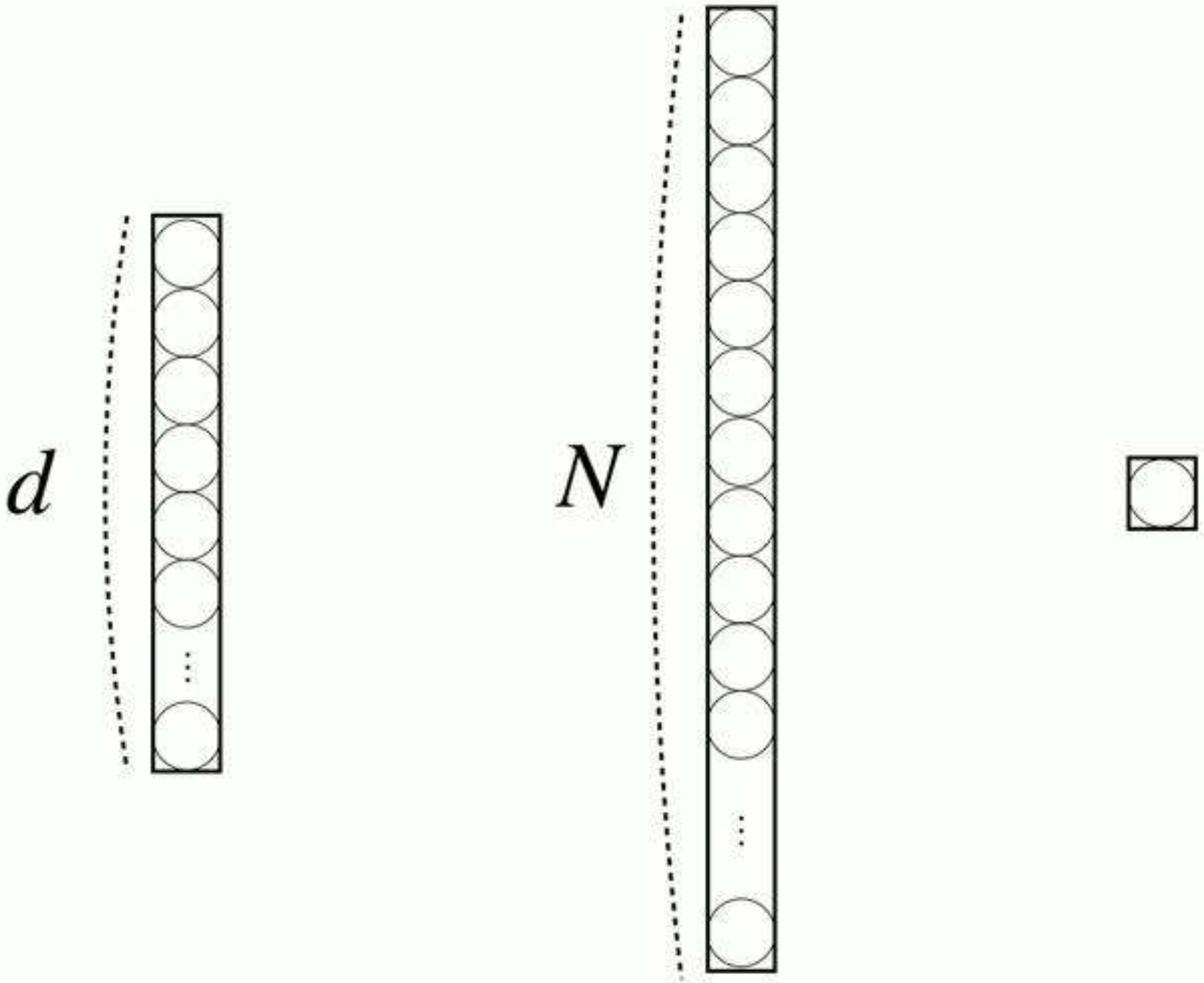
For p-class classification, $\Theta(\sqrt{Np})$ requirement of 2-hidden-layer improves to $\Theta(\sqrt{N} + p)$ by adding another layer

- ImageNet $N \approx 10^6, p = 10^3$ memorizable with 2k-2k-4k FCNN.
- Surprisingly small network size is required to memorize & <u>achieve zero training loss</u> at global minimum.

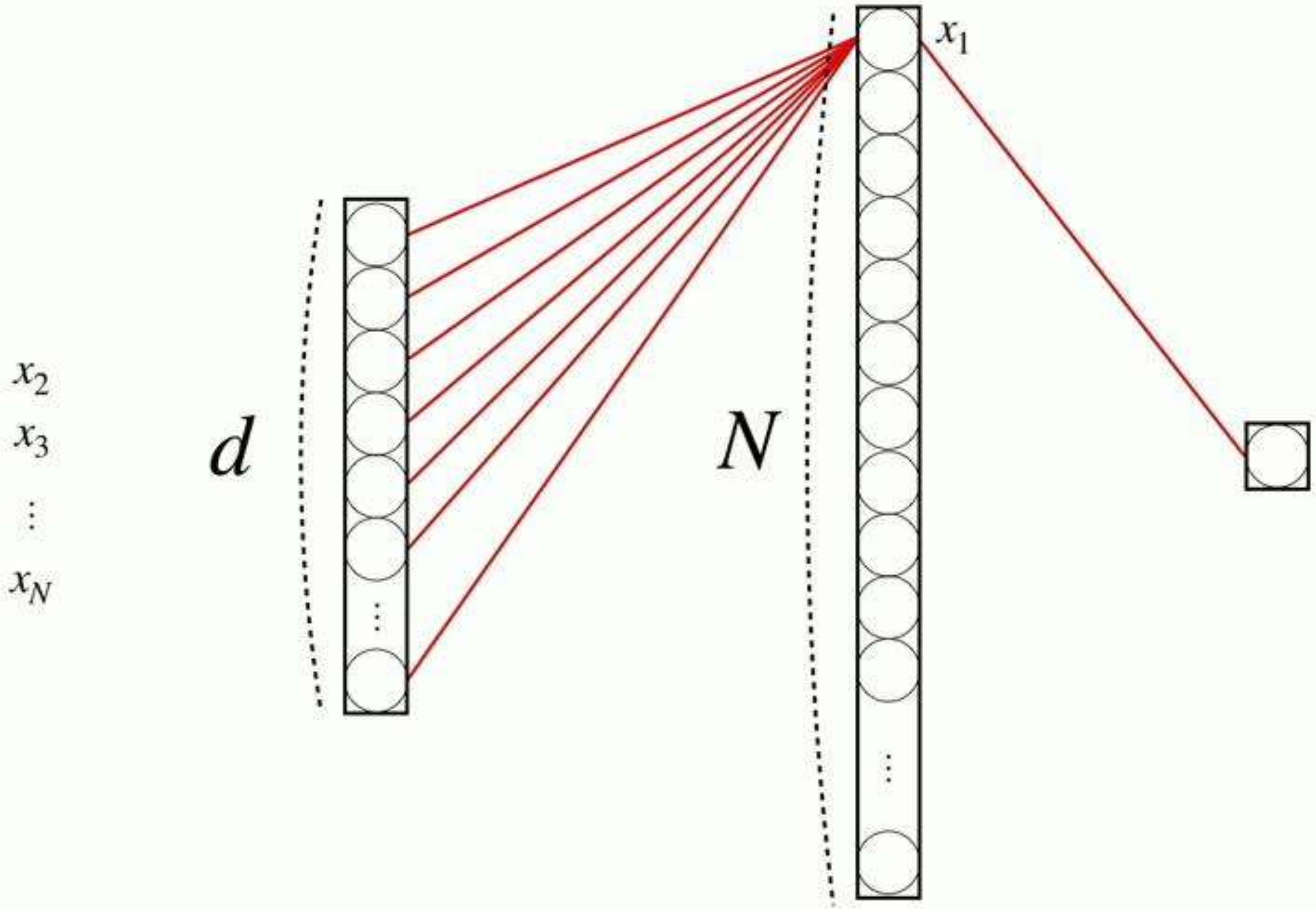
Proof ideas: setup

- **Training data**: $\{(x_i, y_i)\}_{i=1}^{N}, \;\; x_i \in \mathbb{R}^d, \;\; y_i \in \mathbb{R}$
- **Assumption**: all x_i's are distinct and all $y_i \in [-1,1]$.
- **Clipping region**: $\mathbb{R} \backslash [-1,1]$

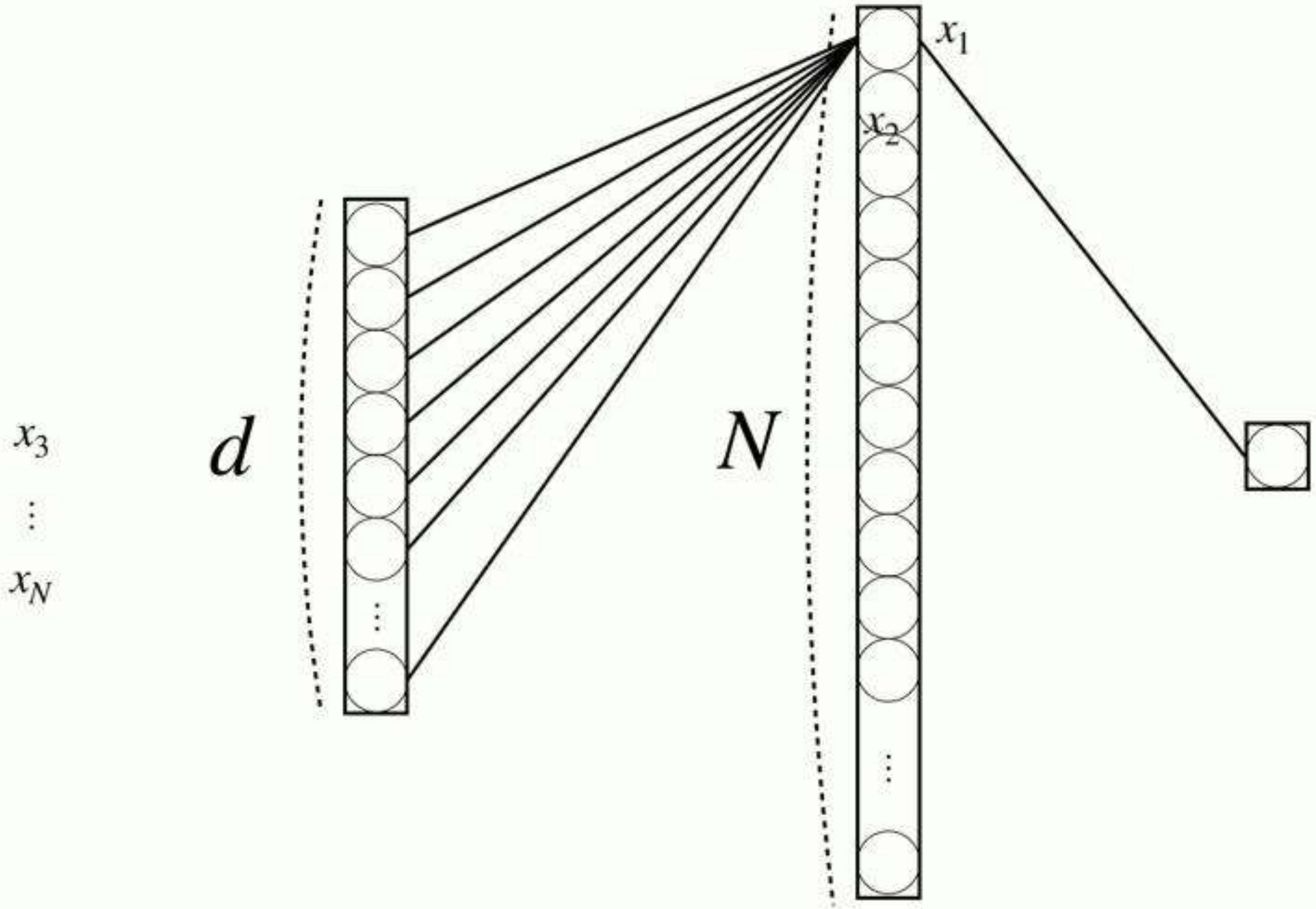
Key ideas of proof



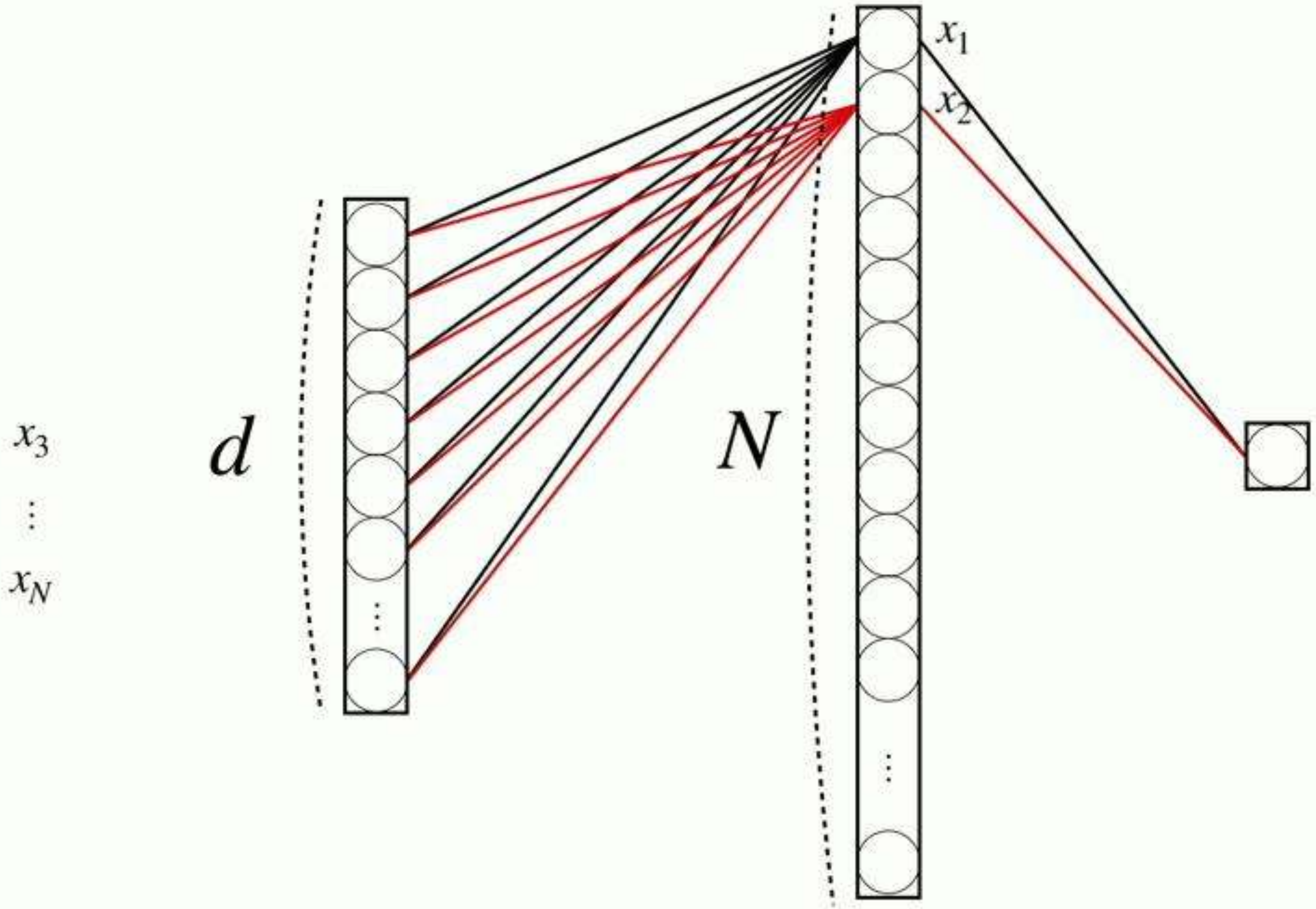
Key ideas of proof



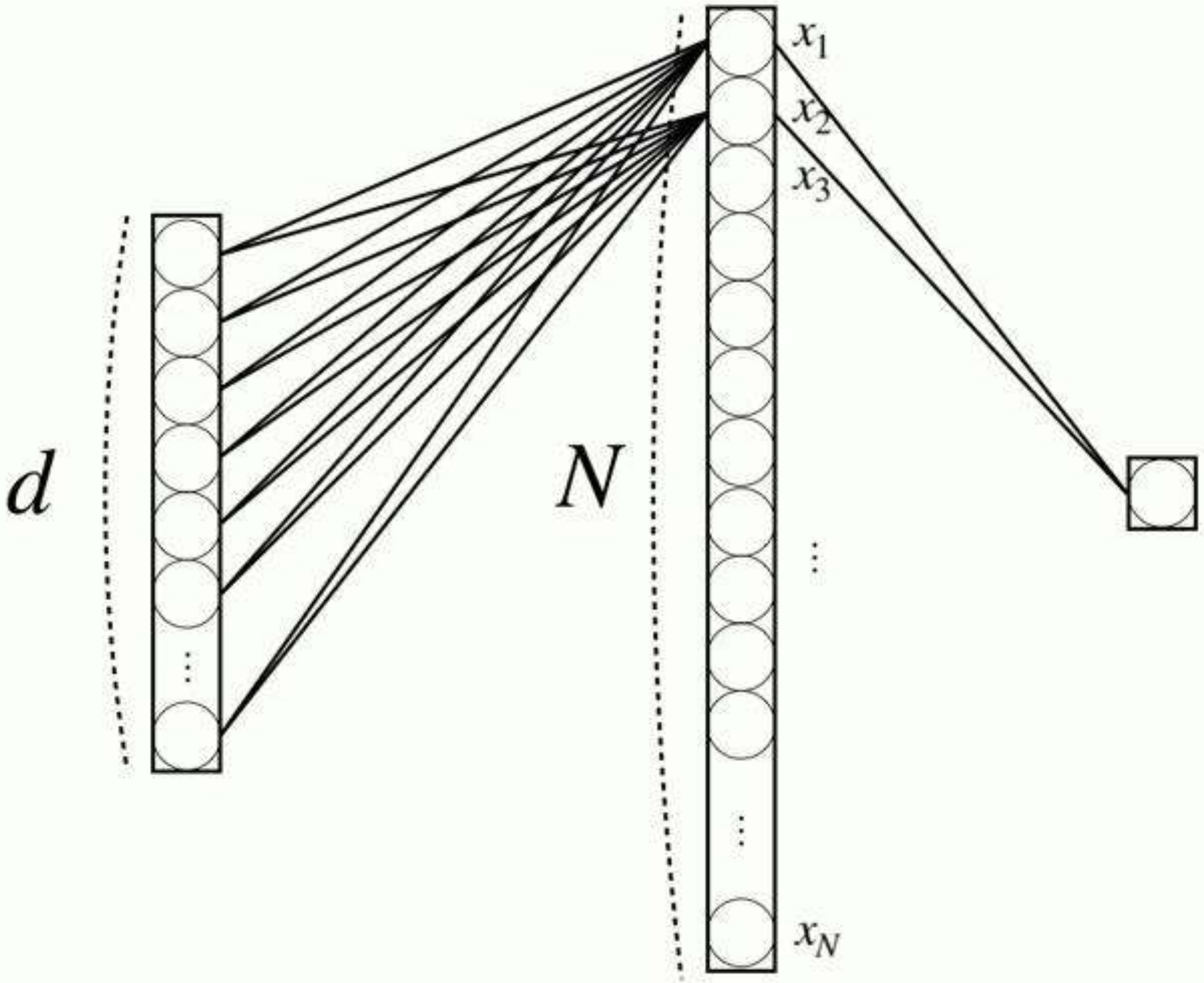
Key ideas of proof



Key ideas of proof

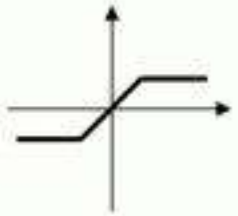


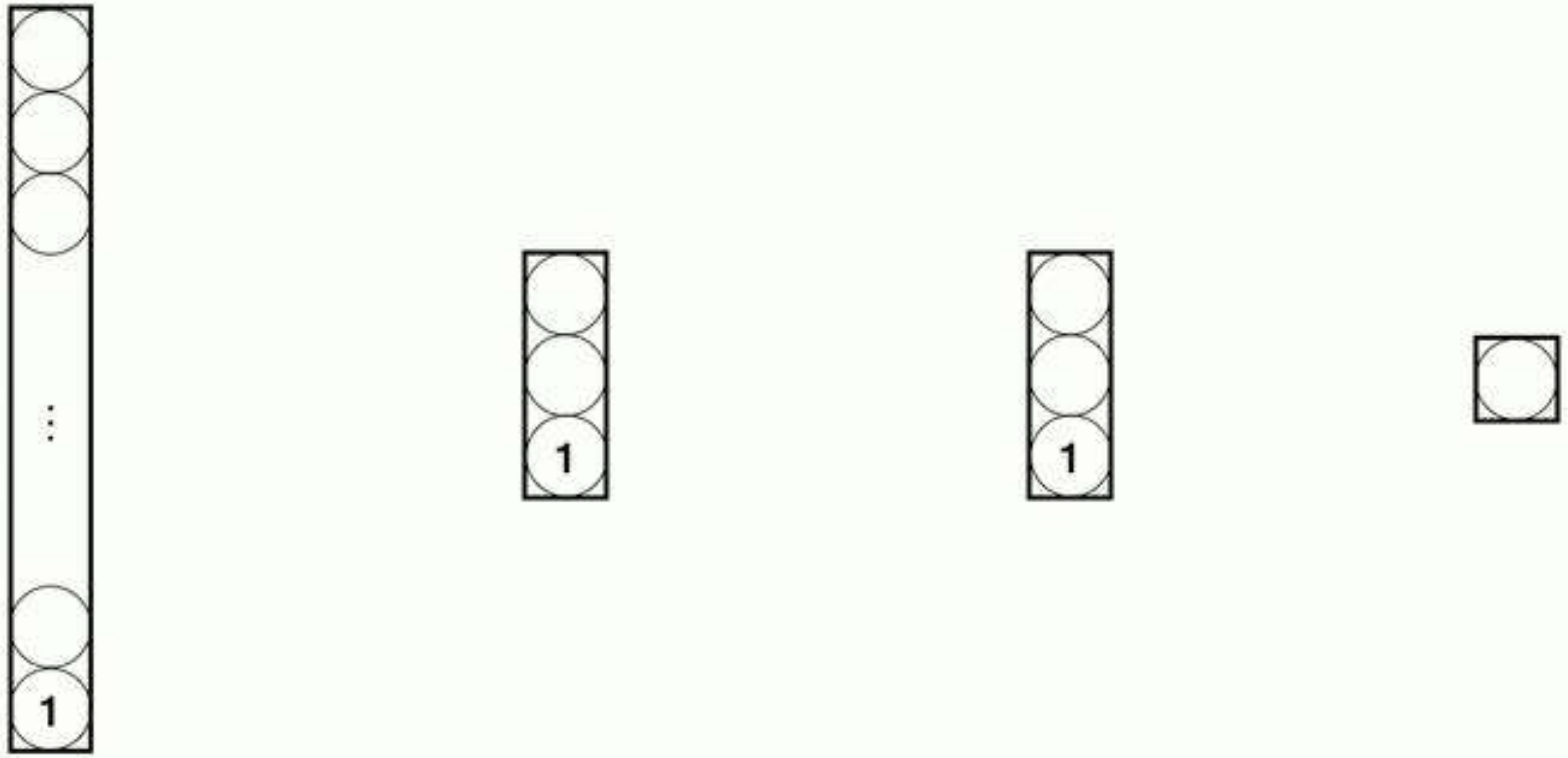
Key ideas of proof



Key ideas of the proof

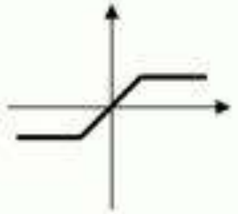
= hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

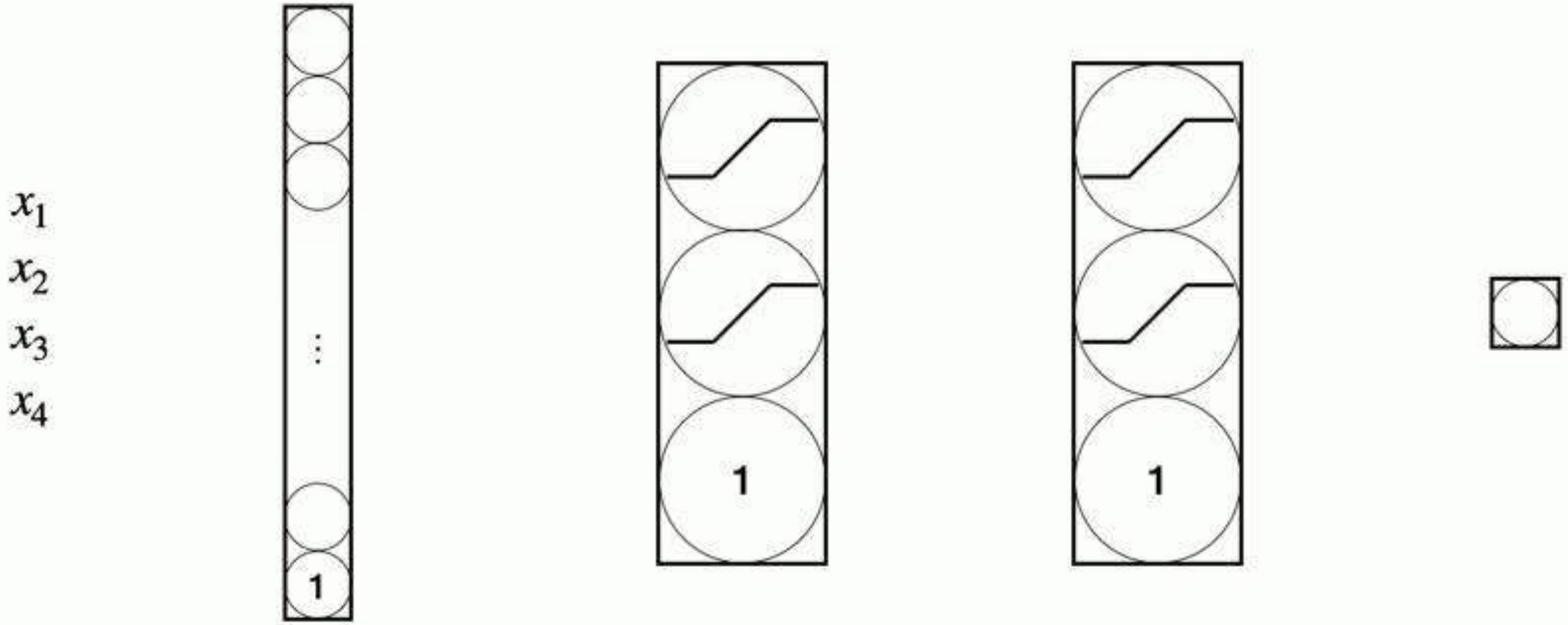




Key ideas of the proof

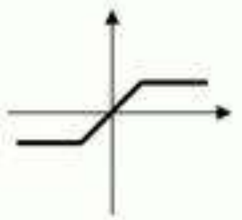
○ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

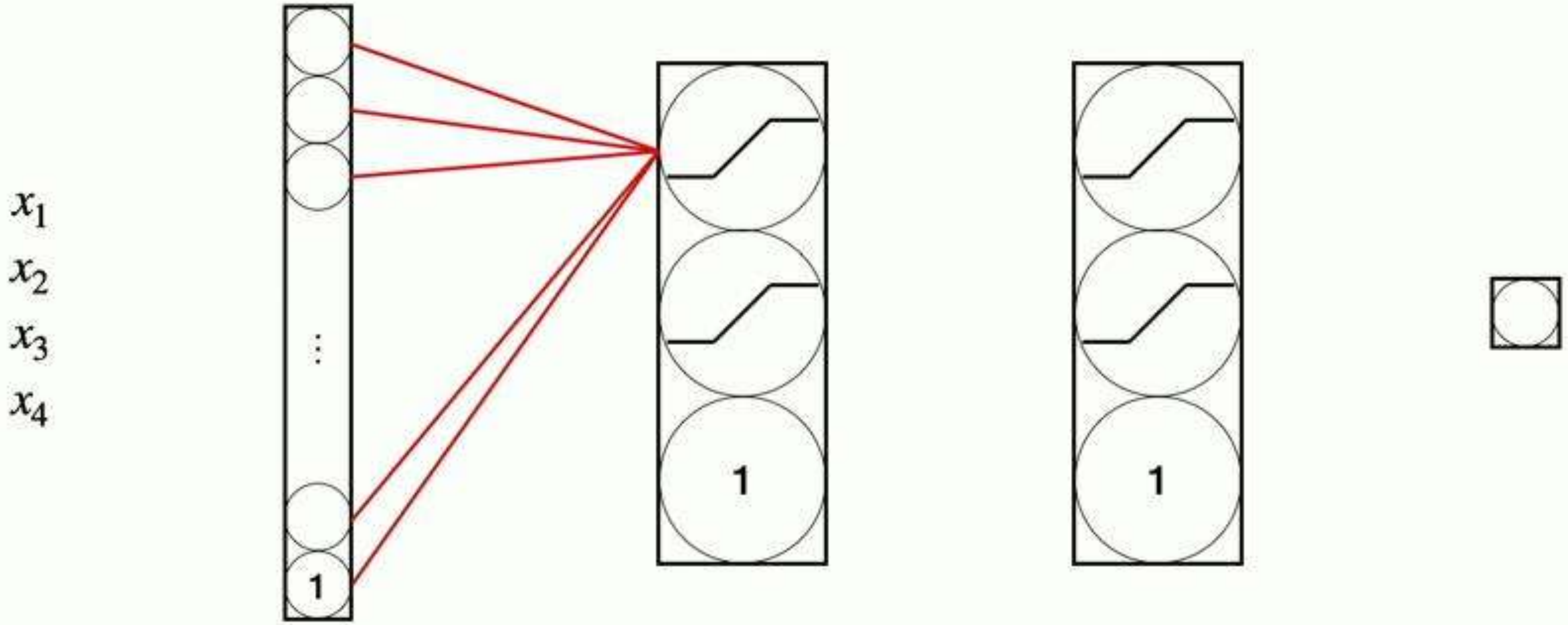




Key ideas of the proof

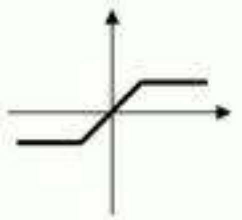
$\bigcirc$ = hard-tanh, $\sigma_H(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

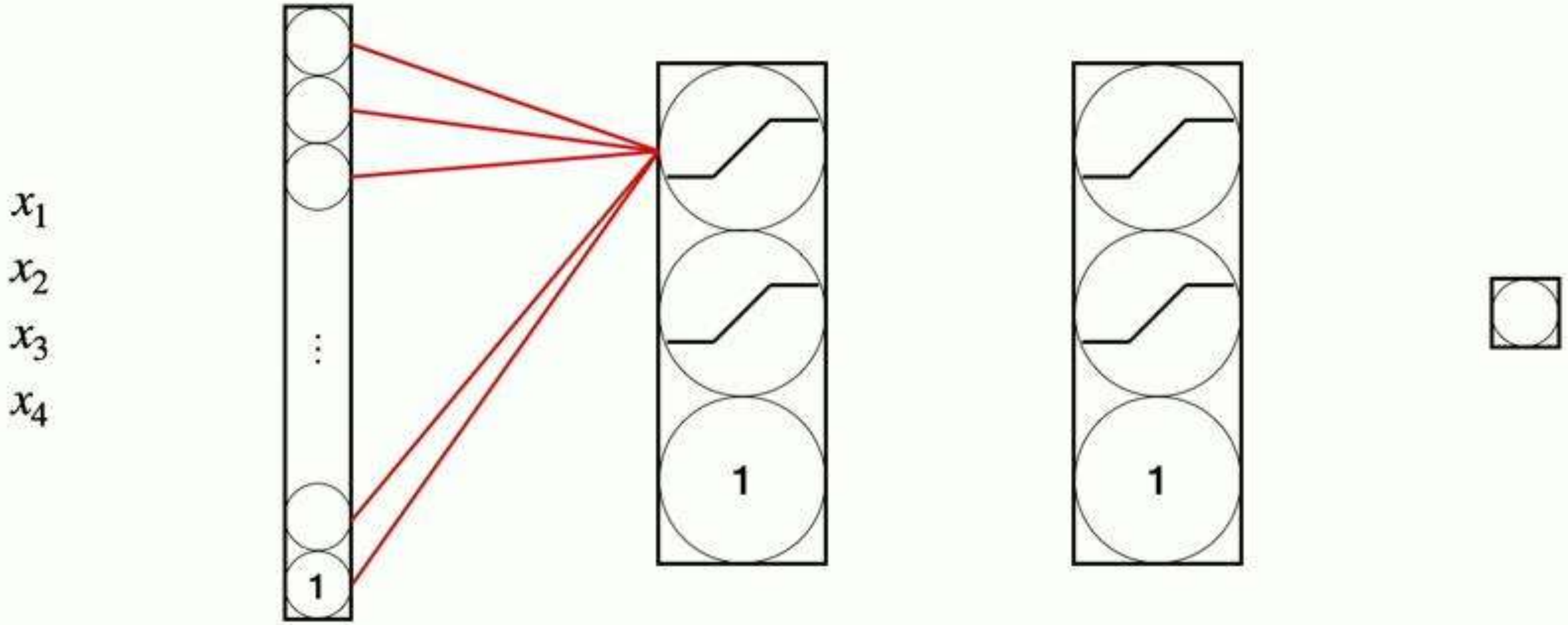




Key ideas of the proof

○ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

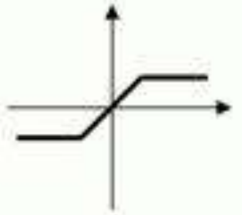


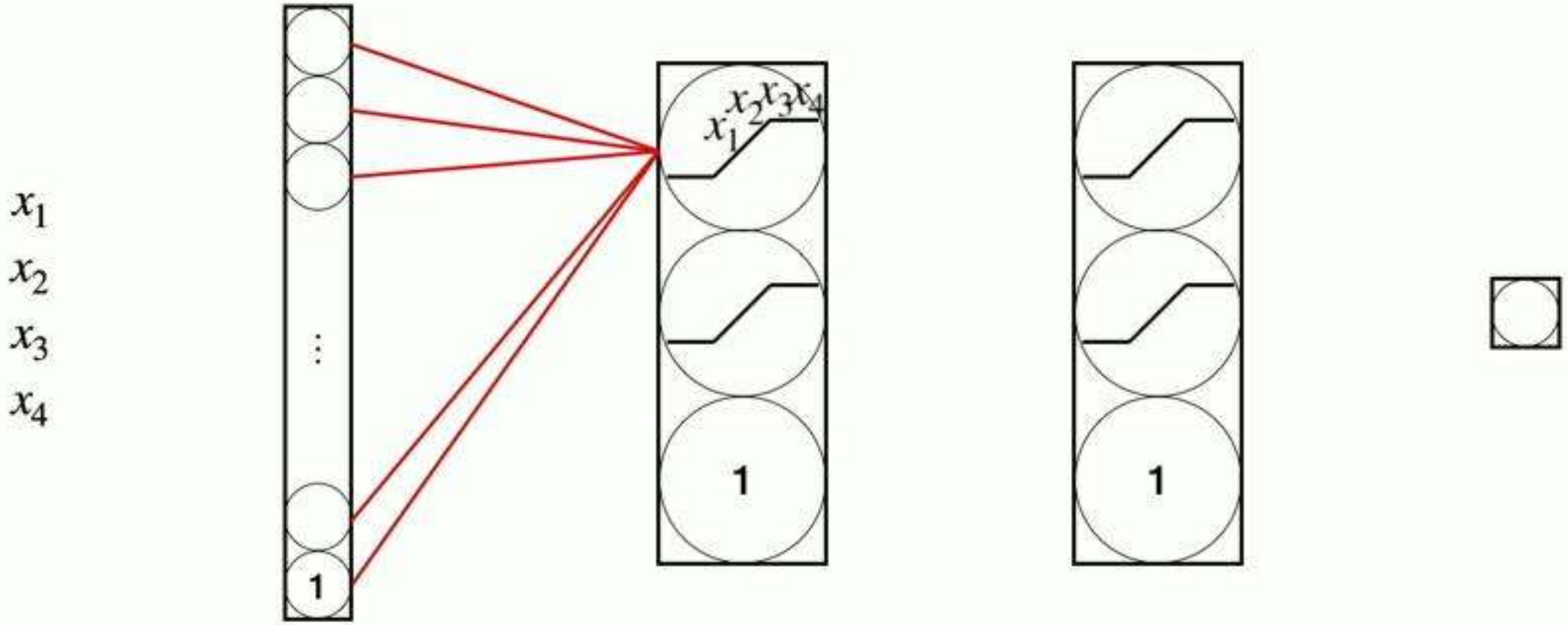


$u^T x_1 < u^T x_2 < u^T x_3 < u^T x_4$

Key ideas of the proof

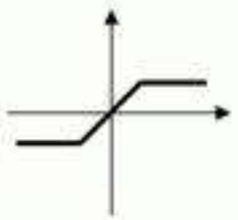
$\bigcirc$ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$



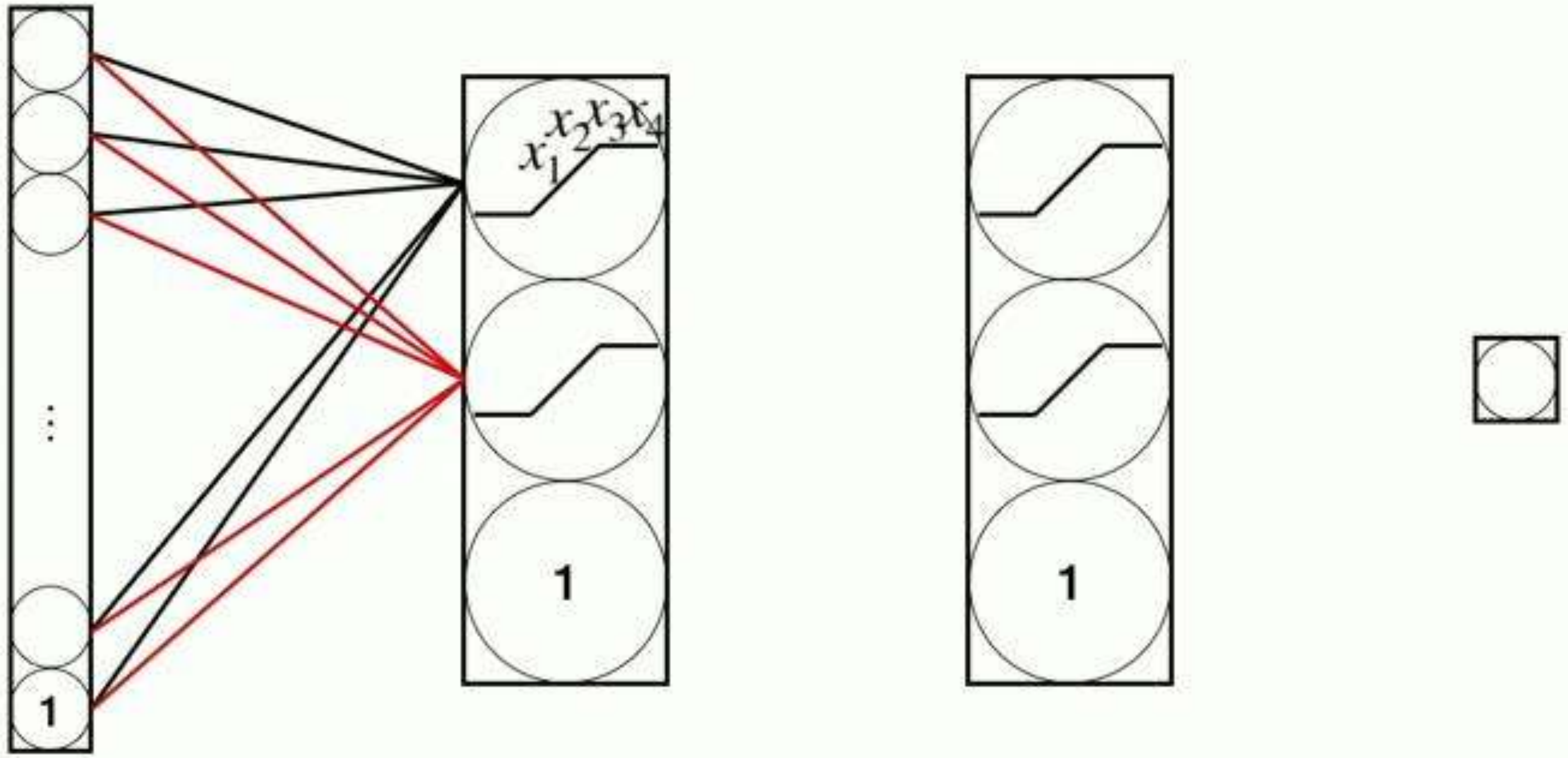


$u^T x_1 < u^T x_2 < u^T x_3 < u^T x_4$

Key ideas of the proof

 = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$ 

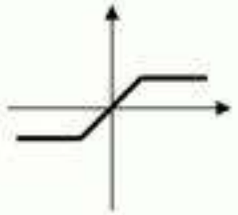
x_1
x_2
x_3
x_4

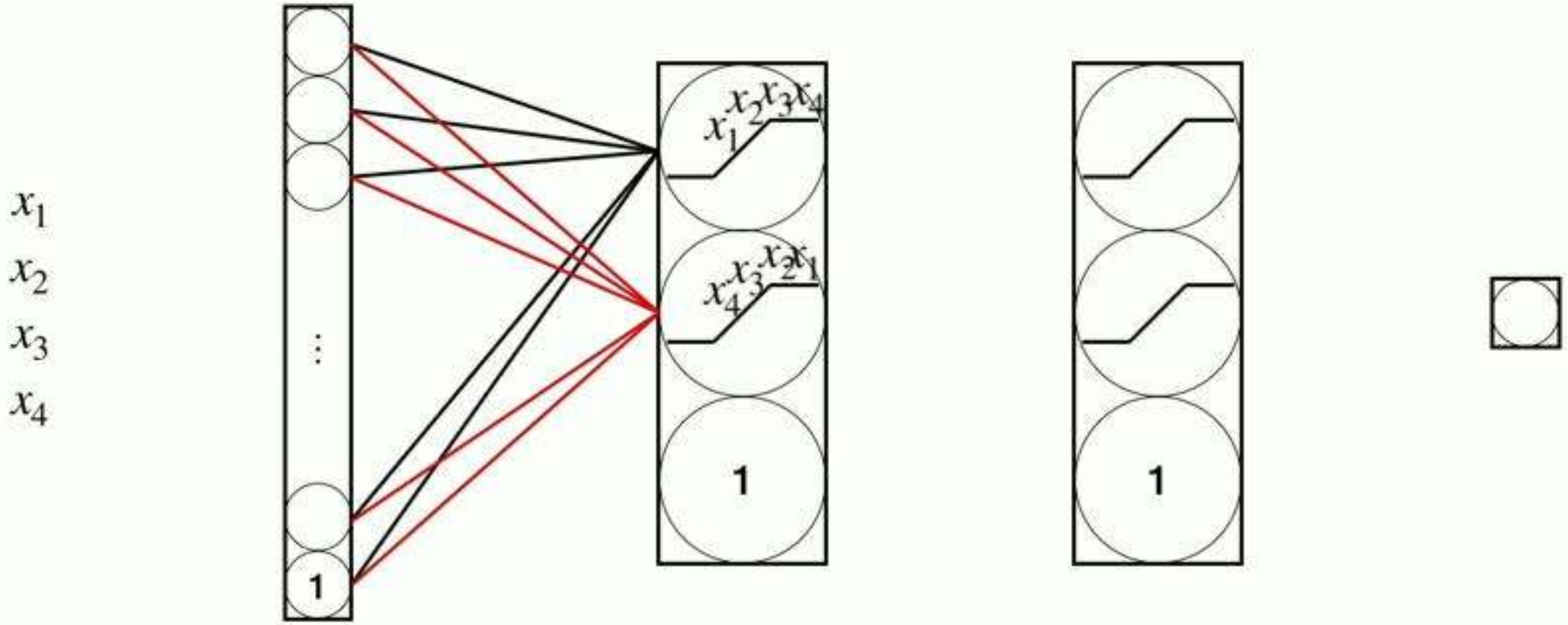


$u^T x_1 < u^T x_2 < u^T x_3 < u^T x_4$

Key ideas of the proof

○ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

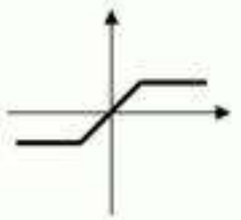


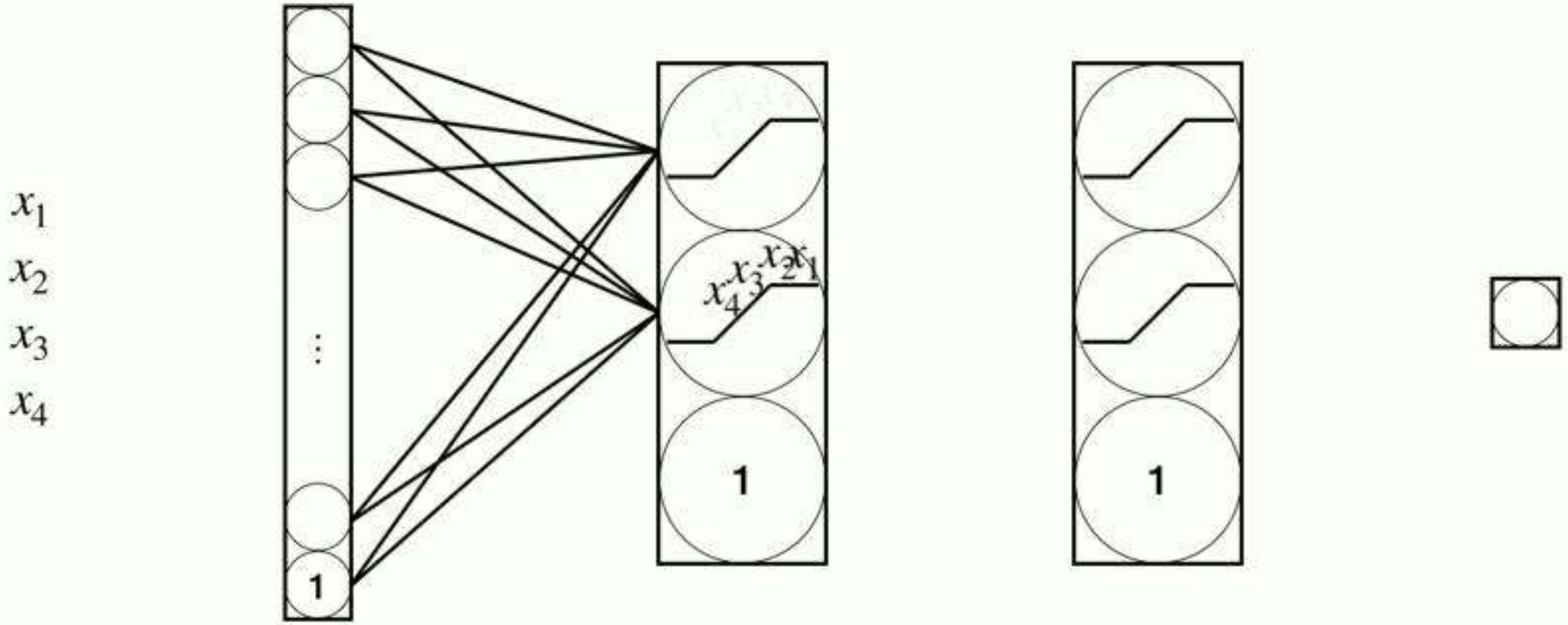


$u^T x_1 < u^T x_2 < u^T x_3 < u^T x_4$

Key ideas of the proof

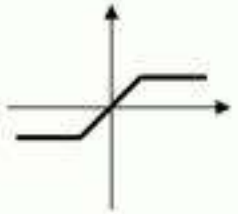
$\bigcirc$ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

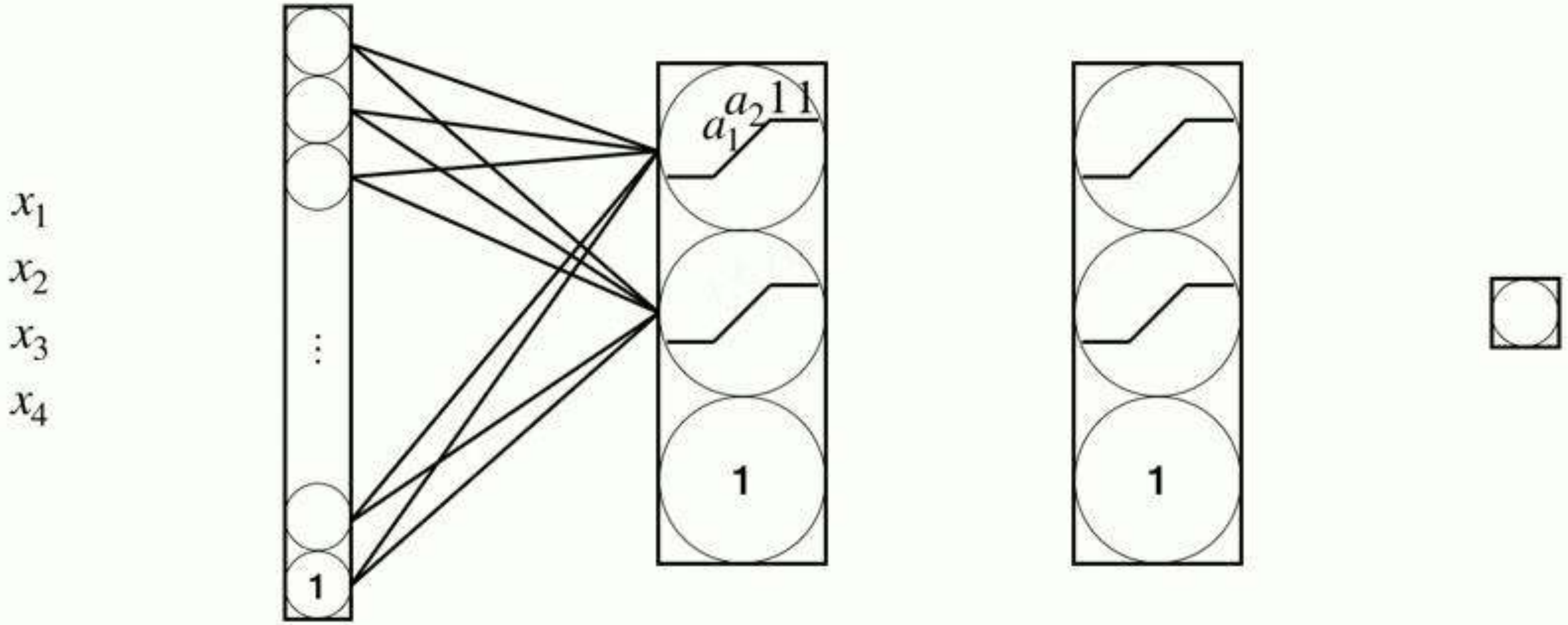




Key ideas of the proof

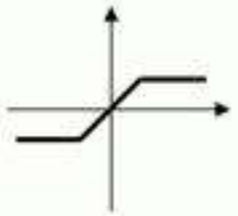
= hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

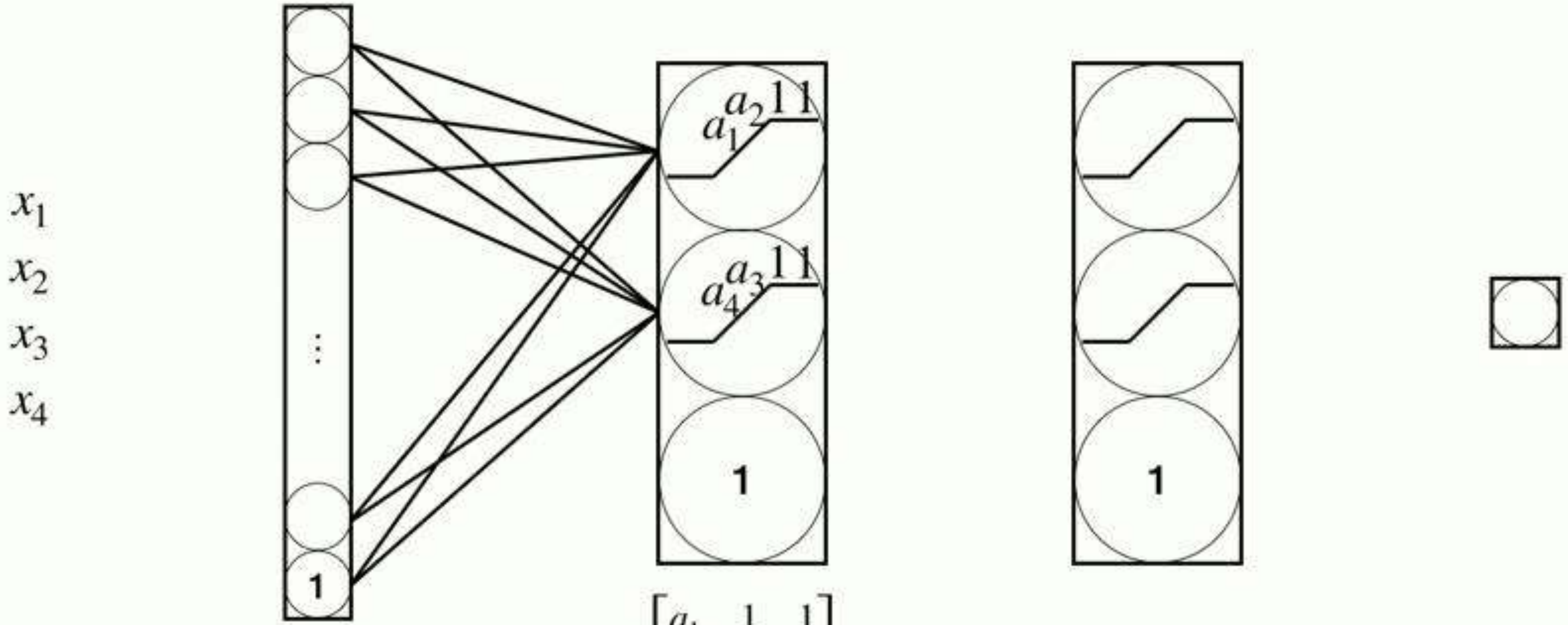




Key ideas of the proof

○ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

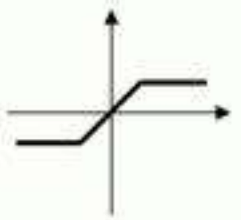


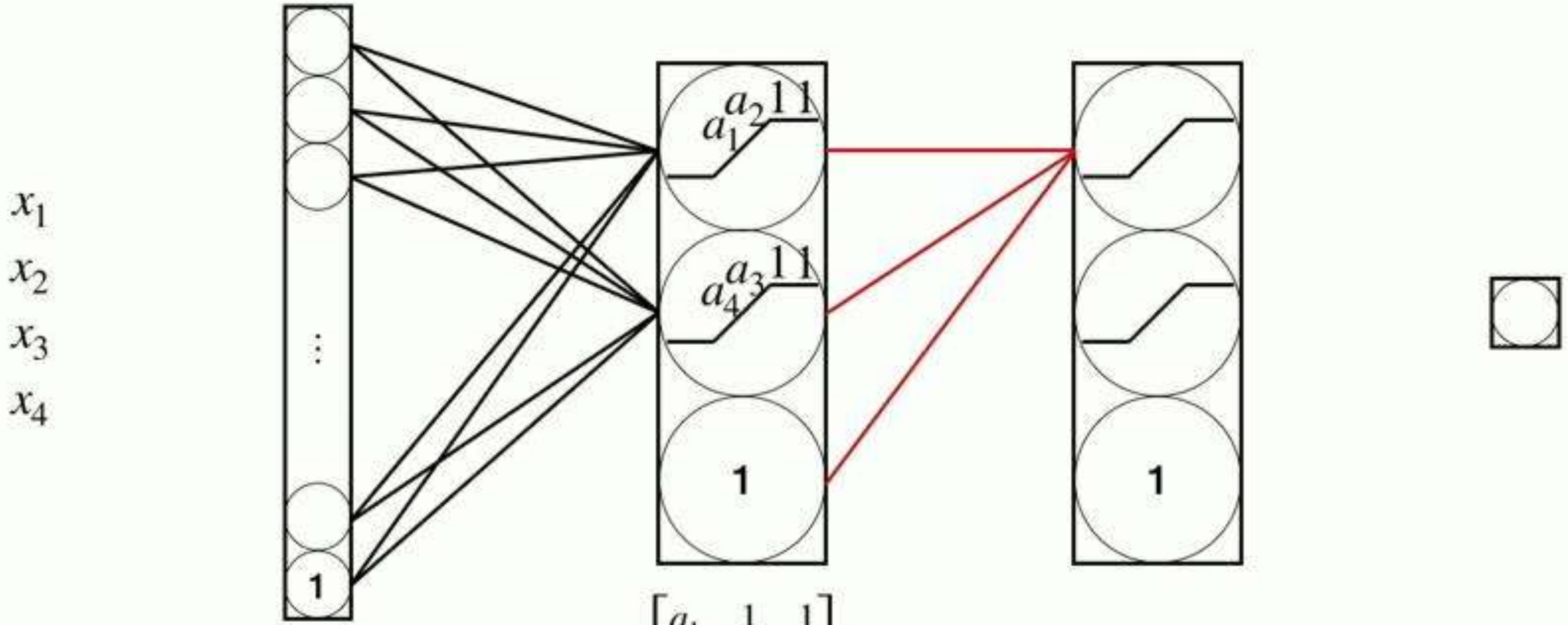


$$\begin{bmatrix} a_1 & 1 & 1 \\ a_2 & 1 & 1 \\ 1 & a_3 & 1 \\ 1 & a_4 & 1 \end{bmatrix}$$

Key ideas of the proof

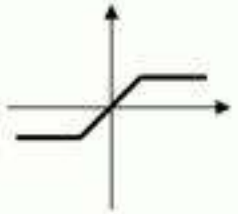
○ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

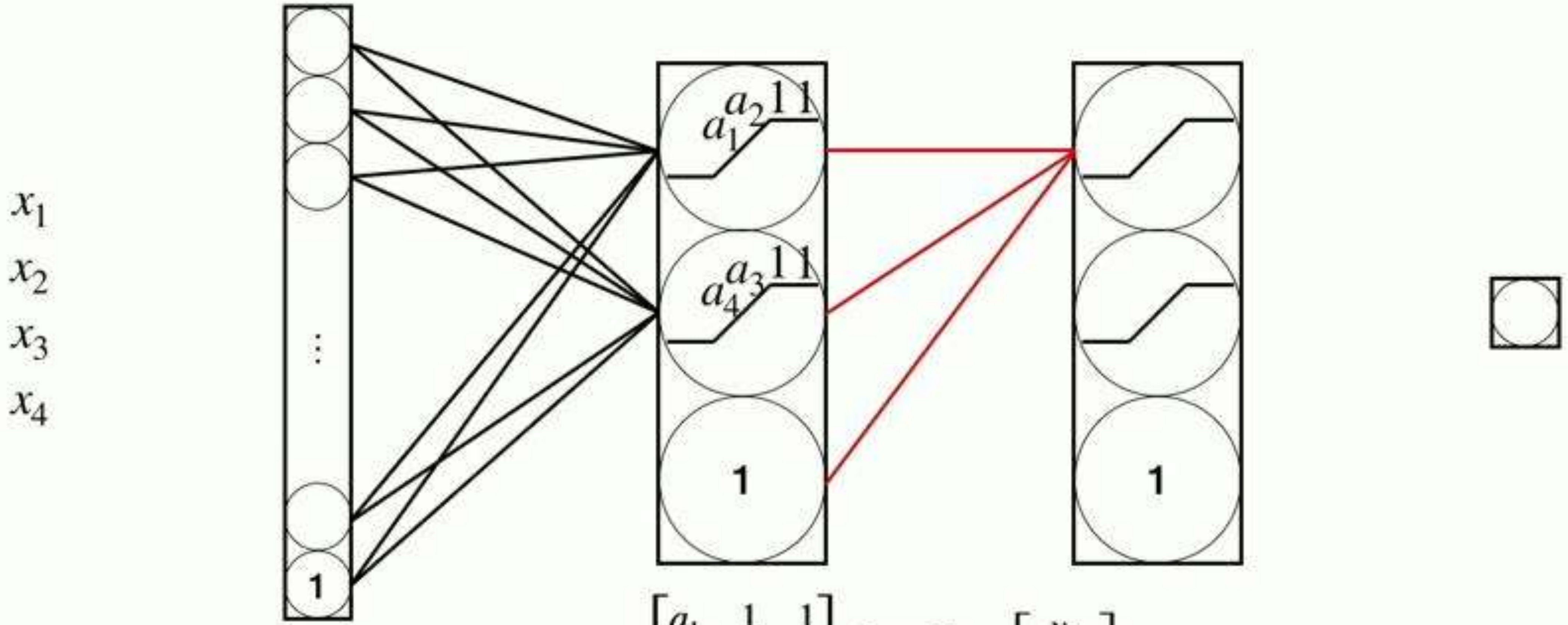




$$\begin{bmatrix} a_1 & 1 & 1 \\ a_2 & 1 & 1 \\ 1 & a_3 & 1 \\ 1 & a_4 & 1 \end{bmatrix}$$

Key ideas of the proof

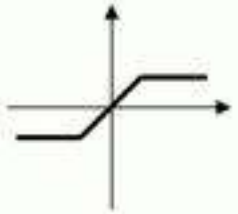
 = hard-tanh, $\sigma_H(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$ 

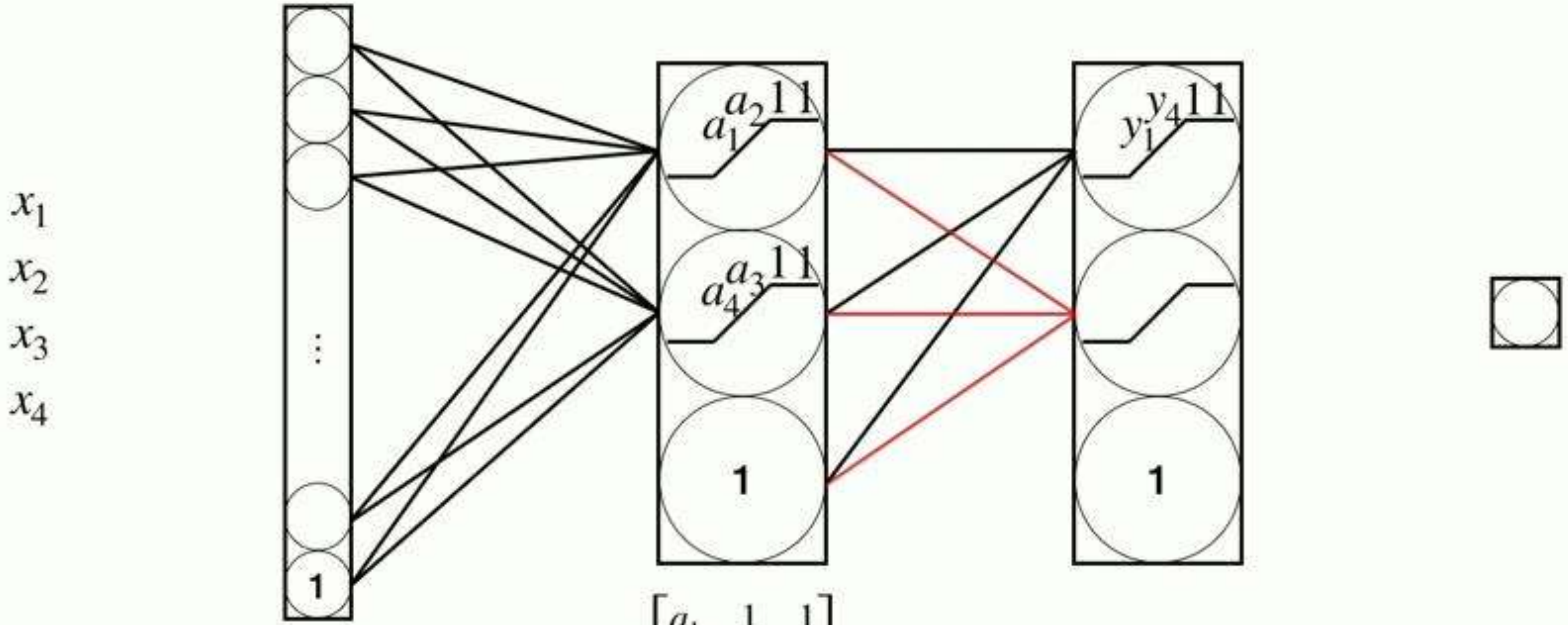


$$\begin{bmatrix} a_1 & 1 & 1 \\ a_2 & 1 & 1 \\ 1 & a_3 & 1 \\ 1 & a_4 & 1 \end{bmatrix} \begin{bmatrix} w_{11} \\ w_{12} \\ b_1 \end{bmatrix} = \begin{bmatrix} y_1 \\ > 1 \\ > 1 \\ y_4 \end{bmatrix}$$

Key ideas of the proof

○ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

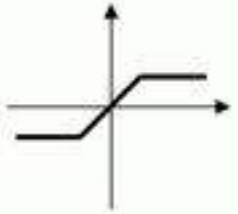




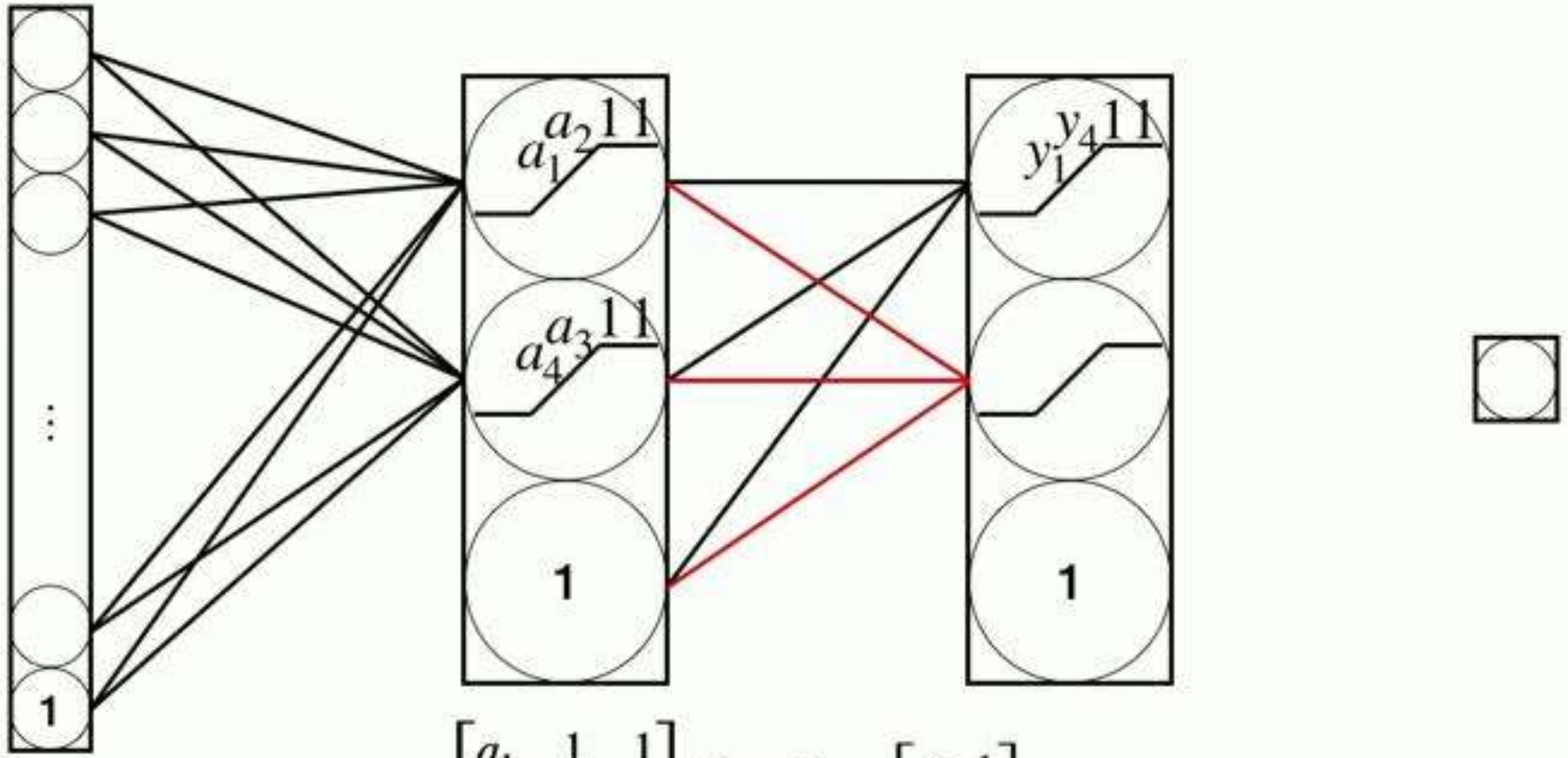
$$\begin{bmatrix} a_1 & 1 & 1 \\ a_2 & 1 & 1 \\ 1 & a_3 & 1 \\ 1 & a_4 & 1 \end{bmatrix}$$

Key ideas of the proof

○ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$



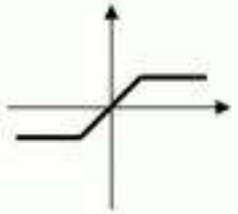
x_1
x_2
x_3
x_4

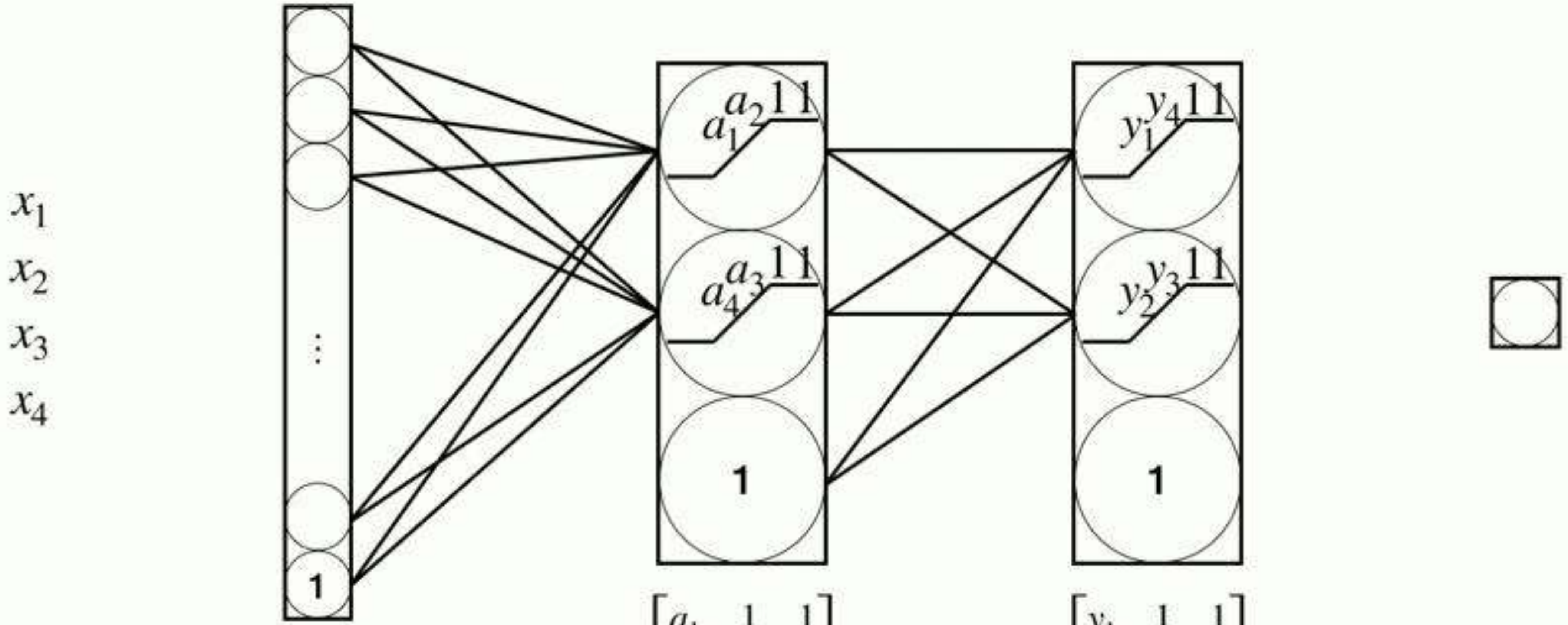


$$\begin{bmatrix} a_1 & 1 & 1 \\ a_2 & 1 & 1 \\ 1 & a_3 & 1 \\ 1 & a_4 & 1 \end{bmatrix} \begin{bmatrix} w_{21} \\ w_{22} \\ b_2 \end{bmatrix} = \begin{bmatrix} > 1 \\ y_2 \\ y_3 \\ > 1 \end{bmatrix}$$

Key ideas of the proof

○ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

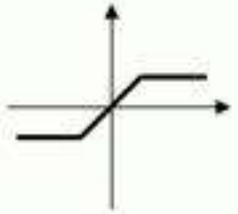


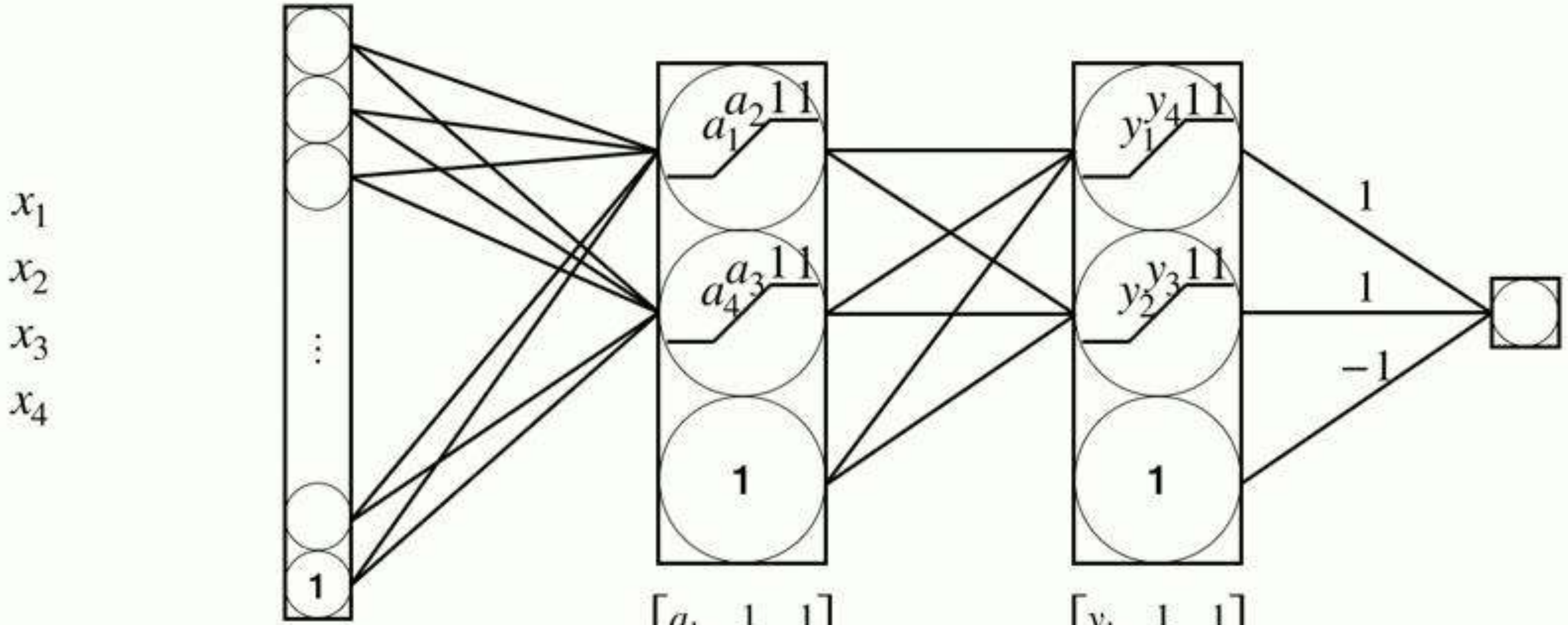


$$\begin{bmatrix} a_1 & 1 & 1 \\ a_2 & 1 & 1 \\ 1 & a_3 & 1 \\ 1 & a_4 & 1 \end{bmatrix} \qquad \begin{bmatrix} y_1 & 1 & 1 \\ 1 & y_2 & 1 \\ 1 & y_3 & 1 \\ y_4 & 1 & 1 \end{bmatrix}$$

Key ideas of the proof

○ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

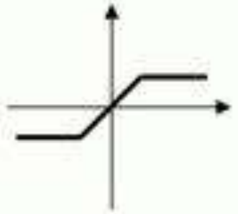


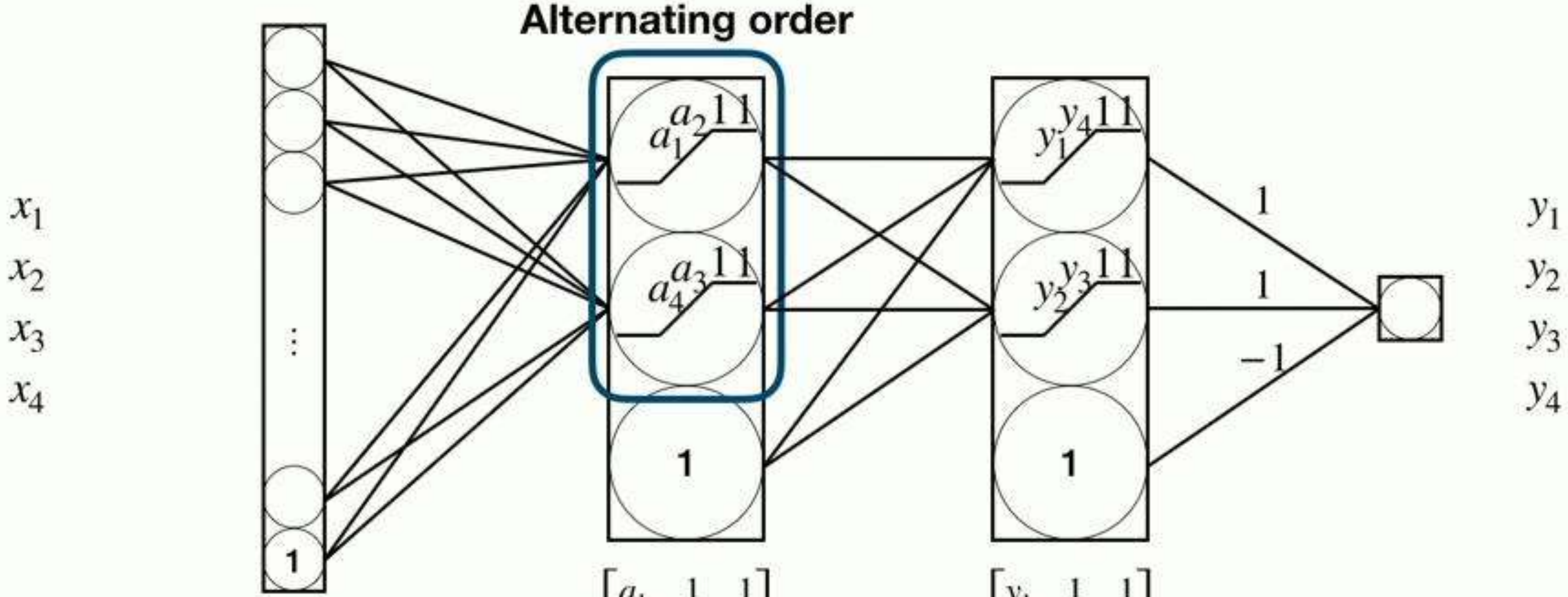


$$\begin{bmatrix} a_1 & 1 & 1 \\ a_2 & 1 & 1 \\ 1 & a_3 & 1 \\ 1 & a_4 & 1 \end{bmatrix} \qquad \begin{bmatrix} y_1 & 1 & 1 \\ 1 & y_2 & 1 \\ 1 & y_3 & 1 \\ y_4 & 1 & 1 \end{bmatrix}$$

Key ideas of the proof

$\bigcirc$ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1, 1] \\ 1 & t > 1 \end{cases}$

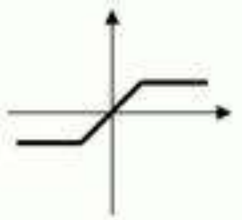


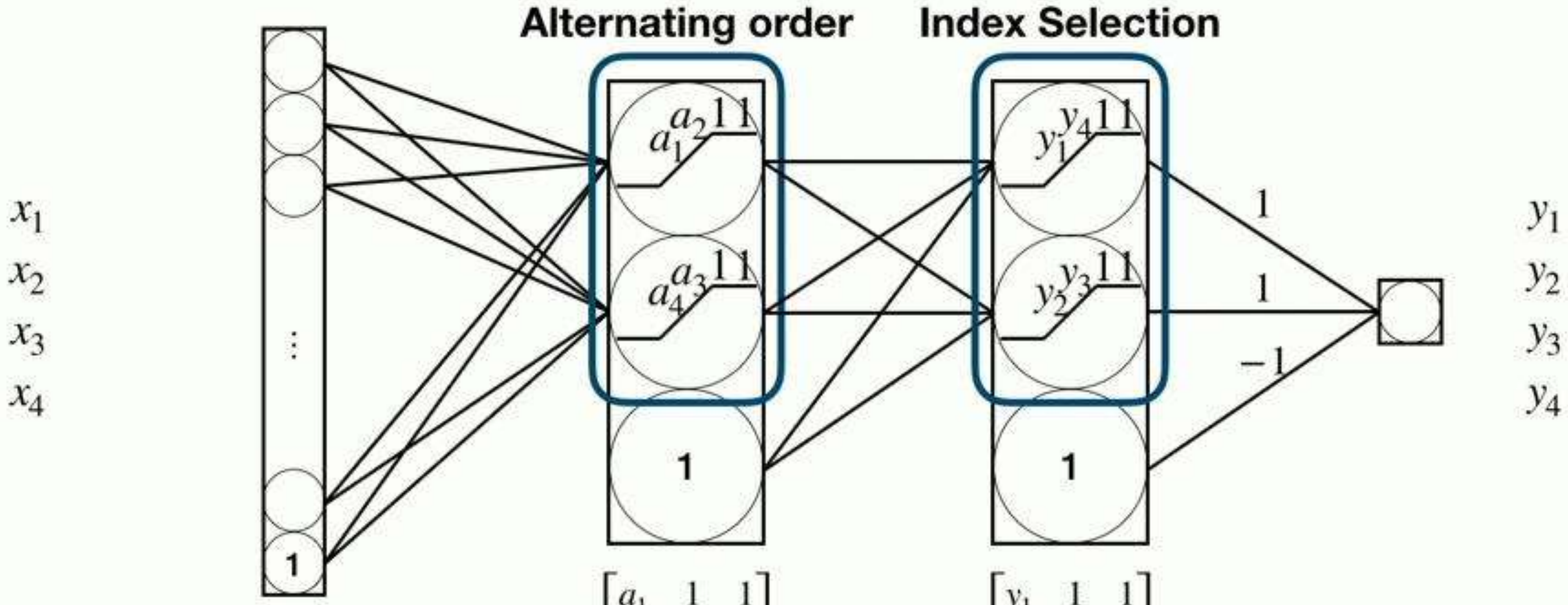


$$\begin{bmatrix} a_1 & 1 & 1 \\ a_2 & 1 & 1 \\ 1 & a_3 & 1 \\ 1 & a_4 & 1 \end{bmatrix} \qquad \begin{bmatrix} y_1 & 1 & 1 \\ 1 & y_2 & 1 \\ 1 & y_3 & 1 \\ y_4 & 1 & 1 \end{bmatrix}$$

Key ideas of the proof

$\bigcirc$ = hard-tanh, $\sigma_{\mathrm{H}}(t) = \begin{cases} -1 & t < -1 \\ t & t \in [-1,1] \\ 1 & t > 1 \end{cases}$

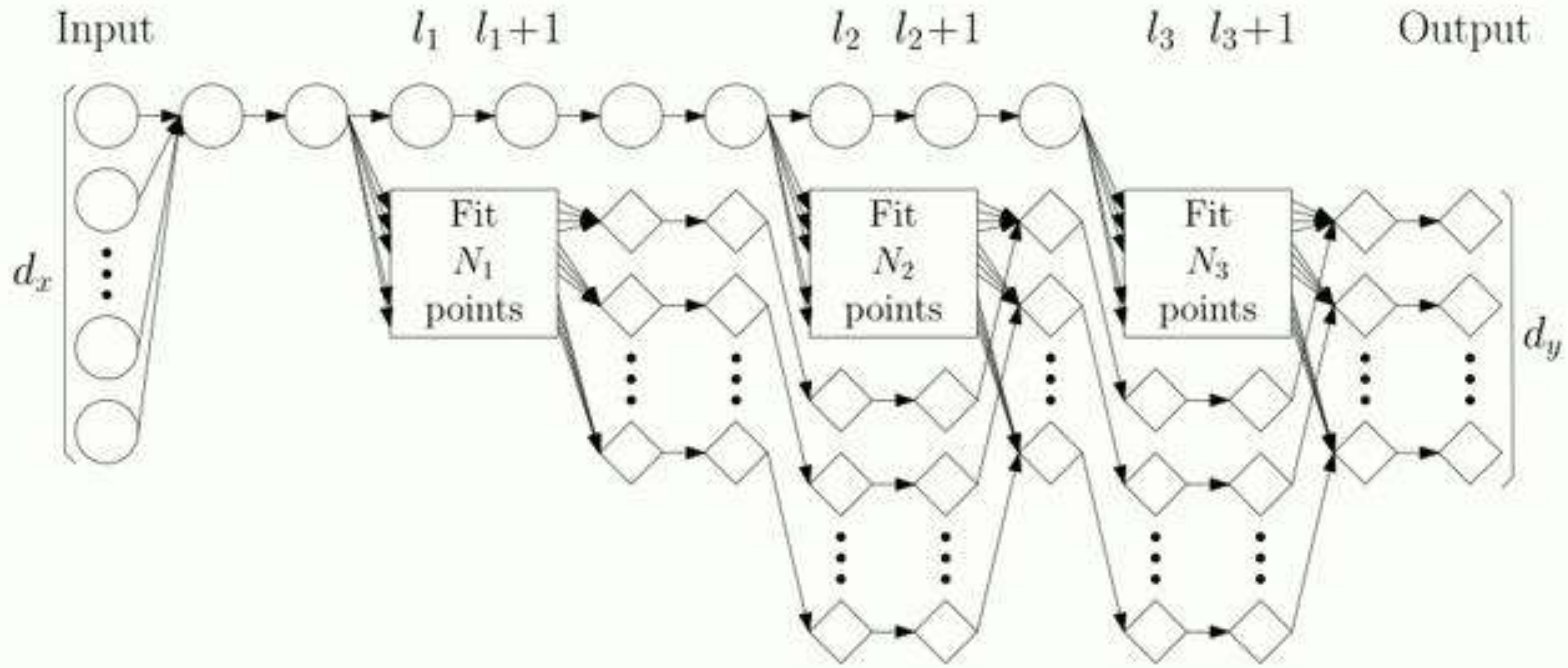




$$\begin{bmatrix} a_1 & 1 & 1 \\ a_2 & 1 & 1 \\ 1 & a_3 & 1 \\ 1 & a_4 & 1 \end{bmatrix} \qquad \begin{bmatrix} y_1 & 1 & 1 \\ 1 & y_2 & 1 \\ 1 & y_3 & 1 \\ y_4 & 1 & 1 \end{bmatrix}$$

Extension to deeper networks

- Extension to deeper networks possible:
  if there are $\Omega(Np)$ parameters between hidden layers, the network can memorize N points.



(schematic above shows only nonzero weights; it's still an FCNN)

Lower and upper bounds

- If $p = 1$, $\Theta(N)$ parameters suffice to memorize N points $\Longrightarrow$ lower bound $\Omega(W)$ on memorization capacity

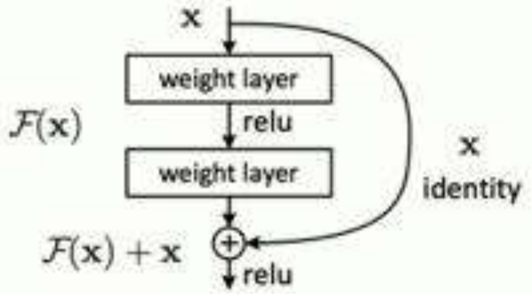
- Upper bound on VC-dim $O(WL \log W)$ *[Bartlett et al., 2019]* gives almost-tight upper bound on memorization capacity (assuming L is a constant).

Finite sample expressivity of ResNets

- **Assumption**: data points x_i's are in general position, i.e., no $d+1$ data points lie on the same affine hyperplane.

- **Assumption**: $y_i \in \{0,1\}^p$ is a one-hot encoding.

- Residual network (ResNet)



$$
\begin{aligned}
h^0(x) &= x, \\
h^l(x) &= h^{l-1}(x) + V^l \sigma(U^l h^{l-1}(x) + b^l) + c^l, \quad l \in \{1, \ldots, L-1\} \\
g_{\boldsymbol{\theta}}(x) &= V^L \sigma(U^L h^{L-1}(x) + b^L) + c^L
\end{aligned}
$$

- d_l is the number of hidden nodes in l-th residual layer

Sufficiency result for ResNets

Theorem 4.

ResNets with hidden layer dims $\sum_{l=1}^{L-1} d_l \geq \frac{4N}{d} + 4p$ and $d_L \geq 2p$ can memorize arbitrary N point classification datasets.

- Under a different assumption, we improve the requirement $N + p$ of *[Hardt & Ma, 2017]* to $\frac{4N}{d} + 6p$.
- For CIFAR-10 ($N = 50$k, $d = 3{,}072$, $p = 10$): 50,010 nodes vs 126 nodes

SGD near memorizers

- We want to solve the empirical risk minimization problem:

$$\text{minimize}_{\boldsymbol{\theta}} \quad \mathfrak{R}(\boldsymbol{\theta}) := \frac{1}{N}\sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(x_i); y_i)$$

- **Assumption.** The loss $\ell(z; y)$ is strictly convex and three times differentiable in z. For any y, there exists a global minimizer z of $\ell(z; y)$.

- **Defn.** A point $\boldsymbol{\theta}^*$ is a **memorizing global minimum** of $\mathfrak{R}(\boldsymbol{\theta})$ if $\nabla_z \ell(f_{\boldsymbol{\theta}^*}(x_i); y_i) = 0$ for all $1 \le i \le N$.

SGD near memorizers

- We analyze without-replacement SGD; mini-batch size B.

- At every $E = N/B$ steps, dataset reshuffled and partitioned into $B^{(kE)}, B^{(kE+1)}, \ldots, B^{(kE+E-1)}$

- SGD update

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \frac{\eta}{B} \sum_{i \in B^{(t)}} \nabla_{\boldsymbol{\theta}} \ell(f_{\boldsymbol{\theta}^{(t)}}(x_i); y_i)$$

SGD near memorizers

Theorem 5 (informal).

If the initialization $\boldsymbol{\theta}^{(0)}$ satisfies $\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\| \leq \rho$ for some memorizing global minimum $\boldsymbol{\theta}^*$ and small constant ρ, initialization satisfies $\mathfrak{R}(\boldsymbol{\theta}^{(0)}) - \mathfrak{R}(\boldsymbol{\theta}^*) \leq C\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|^2$.
If we run SGD with small enough η, it finds a point $\boldsymbol{\theta}$ that satisfies

$$\mathfrak{R}(\boldsymbol{\theta}) - \mathfrak{R}(\boldsymbol{\theta}^*) \leq C'\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|^4, \text{ and}$$
$$\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| \leq 2\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|.$$

SGD near memorizers

- Theorem restricted to initialization very close to memorizing global minima
- However, holds **without** any *width/depth requirement* on the network or *distributional assumption* on data— the only requirement: $\boldsymbol{\theta}^*$ **memorizes** the data.
- Completely **deterministic**, independent of the partition of dataset taken by SGD
- The behavior of SGD after finding $\boldsymbol{\theta}$ is not well understood

Theorem 5 (informal).

If the initialization $\boldsymbol{\theta}^{(0)}$ satisfies $\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\| \le \rho$ for some memorizing global minimum $\boldsymbol{\theta}^*$ and small constant ρ, initialization satisfies $\Re(\boldsymbol{\theta}^{(0)}) - \Re(\boldsymbol{\theta}^*) \le C\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|^2$. If we run SGD with small enough η, it finds a point $\boldsymbol{\theta}$ that satisfies

$$\Re(\boldsymbol{\theta}) - \Re(\boldsymbol{\theta}^*) \le C'\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|^4, \quad \text{and } \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| \le 2\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|.$$

References and other works

Small ReLU networks are powerful memorizers: a tight analysis of memorization capacity *(Yun, S., Jadbabaie, https://arxiv.org/abs/1810.07770)*

Are Deep-ResNets provably better than linear predictors? *(Yun, S., Jadbabaie, https://arxiv.org/abs/1907.03922)*

Why is gradient-clipping faster: an analysis of adaptive-gradient methods under a new smoothness condition weaker than usual Lipschitz gradients *(Zhang, S., Jadbabaie, https://arxiv.org/abs/1905.11881)*

T H A N K S!