Expressivity, sparsity and identifiability in deep neural networks

Rémi Gribonval - ENS Lyon & Inria - DANTE team

remi.gribonval@inria.fr

Quoc-Tung Le

Elisa Riccietti

Pierre Stock

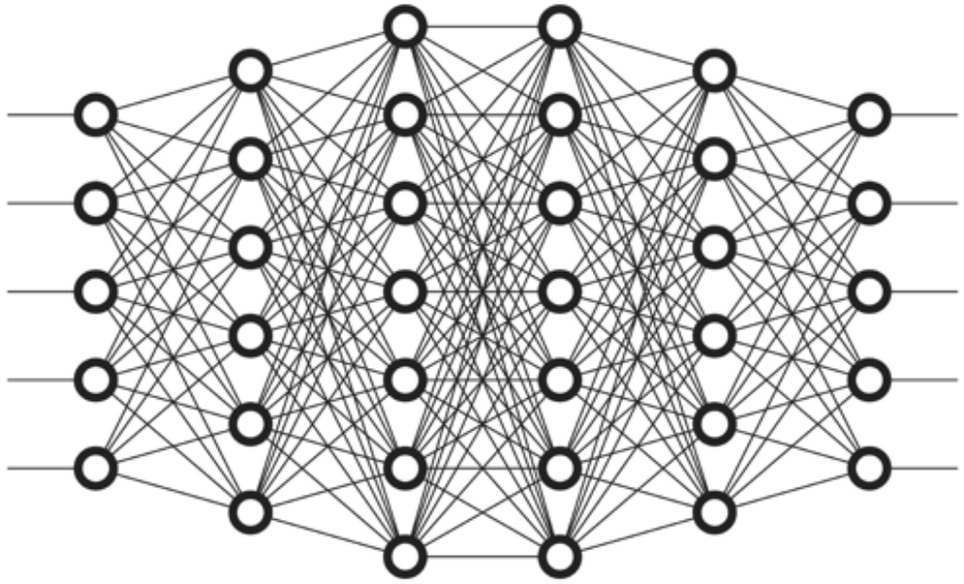
Hervé Jégou

Léon Zheng

and also: Hakim Hadj-Djilani, Nicolas Bellot, Thomas Gautrais, Adrien Leman, …

Introduction

- Deep neural networks = rich computational architecture
  - ***Expressive***: can implement classical “engineered” workflows
  - ***Tunable***: via optimization on (large) training datasets
  - ***Empirically successful*** … ***yet opaque & resource-hungry***



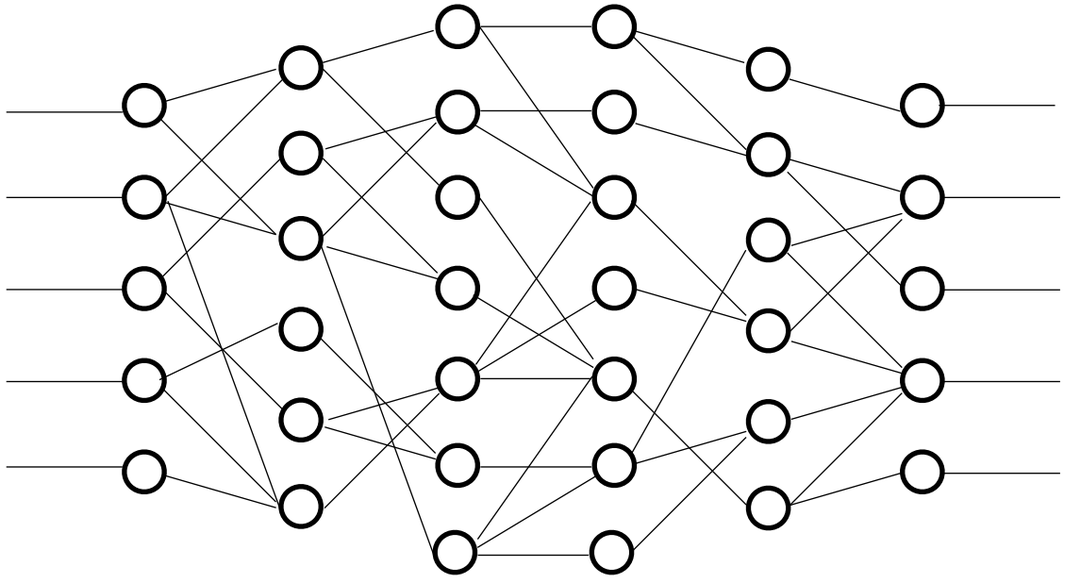
$$f : x \in \mathbb{R}^d \mapsto y = f(x) \in \mathbb{R}^k$$

Introduction

- Deep neural networks = rich computational architecture
  - ***Expressive***: can implement classical "engineered" workflows
  - ***Tunable***: via optimization on (large) training datasets
  - ***Empirically successful*** … ***yet opaque & resource-hungry***

Sparsity as a swiss-knife ?

Interpretability ?
Provably good learning ?
Resource-efficiency ?



$$f : x \in \mathbb{R}^d \mapsto y = f(x) \in \mathbb{R}^k$$

- **Sparsity: some lessons from inverse problems**
- Two-layer sparsity and beyond
- Rescaling-invariance in ReLU networks
- An embedding of ReLU networks

Sparsity in data processing

- As a natural **objective**: frugality / compression
- As a **prior**: identification of latent variables

  - *Prime example* = ***linear inverse problems***

    - linear model $y = \mathbf{A}x$ with known $m \times n$ matrix where $m \ll n$
    - under-determined / ill-posed / infinitely many solutions

  - under sparsity prior $\|x\|_0 \leq k$ and assumptions on $\mathbf{A}$

    - identifiability
    - guaranteed identification with tractable algorithms (e.g. L1 minimization)
    - stable and robust recovery

[S. Foucart & H. Rauhut, **A Mathematical Introduction to Compressive Sensing**, *Springer*, 2013]

Sparsity in data processing

- As a natural **objective**: frugality / compression
- As a **prior**: identification of latent variables

  - *Prime example* = ***linear inverse problems***

    - linear model $y = \mathbf{A}x$ with known $m \times n$ matrix where $m \ll n$
    - under-determined / ill-posed / infinitely many solutions

  - under sparsity prior $\|x\|_0 \leq k$ and assumptions on $\mathbf{A}$

    - identifiability
    - guaranteed identification with tractable algorithms (e.g. L1 minimization)
    - stable and robust recovery

✓ *Interpretable*
✓ *Provably good learning*
✓ *Resource-efficient*

[S. Foucart & H. Rauhut, **A Mathematical Introduction to Compressive Sensing**, *Springer*, 2013]

Sparsity in data processing

- As a natural **objective**: frugality / compression
- As a **prior**: identification of latent variables

  - *Prime example* = ***linear inverse problems***

    - linear model $y = \mathbf{A}x$ with known $m \times n$ matrix where $m \ll n$

      - under-determined / ill-posed / infinitely many solutions

    - under sparsity prior $\|x\|_0 \leq k$ and assumptions on $\mathbf{A}$

      - identifiability
      - guaranteed identification with tractable algorithms (e.g. L1 minimization)
      - stable and robust recovery

✓ *Interpretable*
✓ *Provably good learning*
✓ *Resource-efficient*

[S. Foucart & H. Rauhut, **A Mathematical Introduction to Compressive Sensing**, *Springer*, 2013]

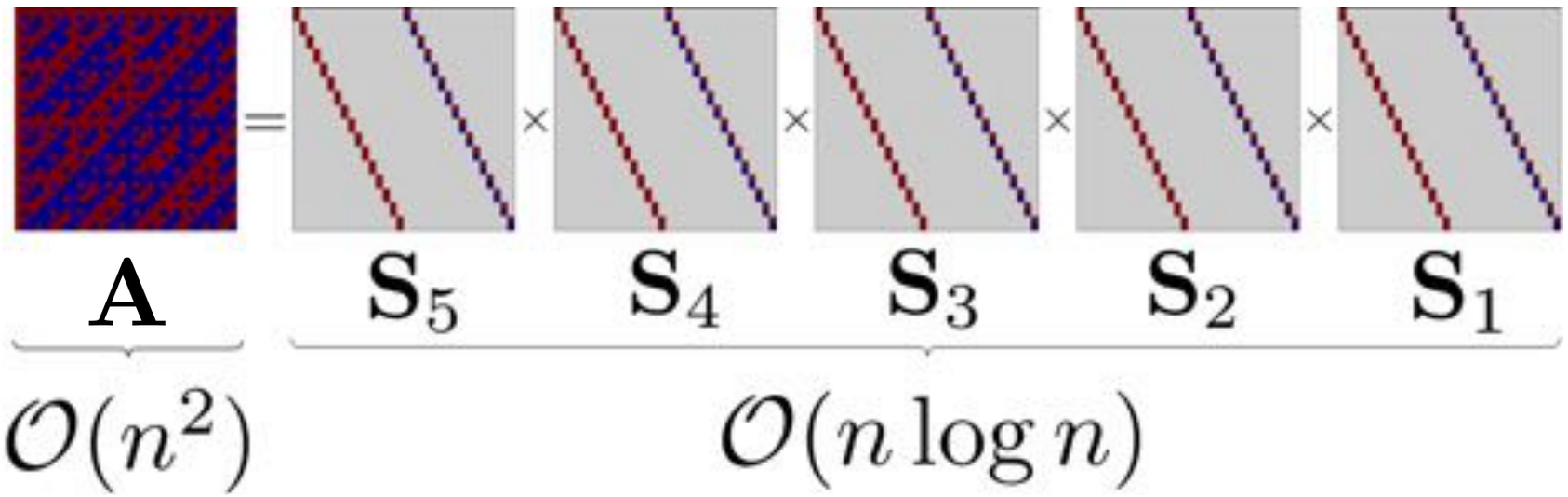
- ***Sparsity beyond linear inverse problems ?***

Fast transforms as sparse factorizations

- **Analytic transforms** (Fourier, wavelets, Hadamard, DCT …) are **fast** because …

Fast transforms as sparse factorizations

- **Analytic transforms** (Fourier, wavelets, Hadamard, DCT …) are **fast** because … **of factorizable structure**[1]



[1] [J. Morgenstern, **The Linear Complexity of Computation**, *J. ACM*, 1975]

Learning Multilayer-Sparse Transforms

- **Products of sparse factors**
  - covers standard fast transforms
  - immediate benefits:
    - **Speed**: *inverse problems and more*
    - **Storage**: *embedded implementation, efficient transmission*

$$\prod_{j=1}^{J} \mathbf{S}_j$$

Learning Multilayer-Sparse Transforms

- **Products of sparse factors**
  - covers standard fast transforms
  - immediate benefits:
    - **Speed**: *inverse problems and more*
    - **Storage**: *embedded implementation, efficient transmission*

$$\prod_{j=1}^{J} \mathbf{S}_j$$

- **Principle:** given matrix $\mathbf{A}$, find an approximation

$$\mathbf{A} \approx \prod_{j} \mathbf{S}_j$$

[O. Chabiron & al, **Toward Fast Transform Learning**, *IJCV*, 2014]

[L. Le Magoarou & R. G., **Flexible Multi-layer Sparse Approximations of Matrices and Applications**, *IEEE JSTSP*, 2016]

[T. Dao & al, **Learning Fast Algorithms for Linear Transforms Using Butterfly Factorizations**, *ICML,* 2019]

[Q.T.-Le & R.G., **Structured Support Exploration For Multilayer Sparse Matrix Factorization,**, *ICASSP,* 2021]

Factorization via optimization ?

- **Optimizing a factor given all others**

$$\min \|\mathbf{S}_j\|_1 \text{ s.t. } \mathbf{A} = \mathbf{L}\mathbf{S}_j\mathbf{R}$$

$$\mathbf{L} = \mathbf{S}_1 \dots \mathbf{S}_{j-1}$$
$$\mathbf{R} = \mathbf{S}_{j+1} \dots \mathbf{S}_J$$

- **Sparsity via a global optimization principle ?**

$$\min \sum_j \|\mathbf{S}_j\|_1 \text{ s.t. } \mathbf{A} \approx \Pi_j \mathbf{S}_j$$

  - guarantees ?

Example: blind sparse deconvolution

- **Goal = recover sparse signals from a convolution**

$$\mathbf{a} = \mathbf{s}_1 \star \mathbf{s}_2$$

  - Expressed as $\mathbf{A} = \mathbf{S}_1 \mathbf{S}_2$ (circulant/Toeplitz structure)
  - Easy if one signal is known, e.g. with

$$\min \|\mathbf{s}_2\|_1 \text{ s.t. } \mathbf{a} = \mathbf{s}_1 \star \mathbf{s}_2$$

  - « Natural » blind approach

$$\min \|\mathbf{s}_1\|_1 + \|\mathbf{s}_2\|_1 \text{ s.t. } \mathbf{a} = \mathbf{s}_1 \star \mathbf{s}_2$$

A pitfall

- **Theorem** (Benichoux, Vincent & G. 2013)
  - The trivial pair $\mathbf{s}_1 = \mathbf{a},\ \mathbf{s}_2 = \delta_0$ is always a global optimum of

$$\min \|\mathbf{s}_1\|_1 + \|\mathbf{s}_2\|_1 \text{ s.t. } \mathbf{a} = \mathbf{s}_1 \star \mathbf{s}_2$$

A pitfall

- **Theorem** (Benichoux, Vincent & G. 2013)
  - The trivial pair $\mathbf{s}_1 = \mathbf{a},\ \mathbf{s}_2 = \delta_0$ is always a global optimum of

$$\min \|\mathbf{s}_1\|_1 + \|\mathbf{s}_2\|_1 \text{ s.t. } \mathbf{a} = \mathbf{s}_1 \star \mathbf{s}_2$$

- **Under the hood**: intrinsic *scaling ambiguity* :
  - Equivalence classes of rescaling-equivalent parameters
    - if $\mathbf{S}_1, \mathbf{S}_2$ is feasible then so is $\lambda \mathbf{s}_1, \mathbf{s}_2/\lambda$ for any scalar
  - Implicit regularization $\min_\lambda \|\lambda \mathbf{s}_1\|_1 + \|\mathbf{s}_2/\lambda\|_1 \propto \sqrt{\|\mathbf{s}_1\|_1 \|\mathbf{s}_2\|_1}$
  - For any feasible pair $\|\mathbf{a}\|_1 \|\boldsymbol{\delta}_0\|_1 = \|\mathbf{a}\|_1 \leq \|\mathbf{s}_1\|_1 \|\mathbf{s}_2\|_1$

A pitfall

- **Theorem** (Benichoux, Vincent & G. 2013)
  - The trivial pair $\mathbf{s}_1 = \mathbf{a},\ \mathbf{s}_2 = \delta_0$ is always a global optimum of

$$\min \|\mathbf{s}_1\|_1 + \|\mathbf{s}_2\|_1 \text{ s.t. } \mathbf{a} = \mathbf{s}_1 \star \mathbf{s}_2$$

- **Under the hood**: intrinsic *scaling ambiguity* :
  - **Equivalence classes** of rescaling-equivalent parameters
    - if $\mathbf{S}_1, \mathbf{S}_2$ is feasible then so is $\lambda \mathbf{s}_1, \mathbf{s}_2/\lambda$ for any scalar
  - **Implicit regularization** $\min_\lambda \|\lambda \mathbf{s}_1\|_1 + \|\mathbf{s}_2/\lambda\|_1 \propto \sqrt{\|\mathbf{s}_1\|_1 \|\mathbf{s}_2\|_1}$
  - **For any feasible pair** $\|\mathbf{a}\|_1 \|\boldsymbol{\delta}_0\|_1 = \|\mathbf{a}\|_1 \leq \|\mathbf{s}_1\|_1 \|\mathbf{s}_2\|_1$

➡Lesson 1: beware of scaling ambiguities

[A. Benichoux, E. Vincent & R. G., **A fundamental pitfall in blind deconvolution with sparse and shift-invariant priors**, *ICASSP*, 2013]

- Sparsity: some lessons from inverse problems
- **Two-layer sparsity and beyond**
- Rescaling-invariance in ReLU networks
- An embedding of ReLU networks

Learning sparse networks

Inverse problems wisdom

- It is **NP-hard** to find

$$x^\star \in \arg\min \|x\|_0 \text{ s.t.} y = \mathbf{A}x$$

- … but this is **tractable** under sufficient (sparsity) assumptions

- Can be split into two sub-problems
  - ***hard***: select best support

$$I = \text{supp}(x^\star)$$

  - ***easy***: fit coefficients on support

$$\min_{x_I} \|y - \mathbf{A}_I x_I\|_2^2$$

Learning sparse networks

- Standard practice: dense learning & pruning
- « Lottery ticket hypothesis » heuristics
  - given support, SGD training made easy and stable

[J. Frankle & M. Carbin, **The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks**, *ICLR*, 2019]

[Zhou & al, **Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask**, *NeurIPS*, 2019]

Can this be grounded on some theory ?

➡ begin with ***shallow & linear*** (but **sparse**) networks

Sparse matrix factorization

Constrained optimization problem

$$\min L(\mathbf{X}, \mathbf{Y}) := \|\mathbf{A} - \mathbf{X}\mathbf{Y}\|_F^2$$

$$\mathtt{supp}(\mathbf{X}) \subseteq I, \ \mathtt{supp}(\mathbf{Y}) \subseteq J \qquad (I, J) \in \Sigma^{\mathtt{left}} \times \Sigma^{\mathtt{right}}$$

- Constraint = families of sparsity patterns $\Sigma^{\mathtt{left}}, \Sigma^{\mathtt{right}}$
- Examples
  - set Σ_k of k-sparse matrices
  - set $\Sigma_{k,\mathtt{col}}$ of matrices with k-sparse columns
  - etc.

Known to be NP-hard

- e.g., covers sparse PCA

What if support is fixed ?

- ***Easy*** for classical sparsity (e.g., X known, Y sparse unknown): ***least squares***

To be or not to be tractable

Theorem 1 (hardness)

- Sparse matrix factorization ***with fixed support*** (I, J) is NP-hard

$$\min L(\mathbf{X}, \mathbf{Y}) := \|\mathbf{A} - \mathbf{X}\mathbf{Y}\|_F^2$$

$$\mathrm{supp}(\mathbf{X}) \subseteq I, \ \mathrm{supp}(\mathbf{Y}) \subseteq J$$

➡ very different from least squares !

Detailed proofs and condition: see [Q.-T. Le, E. Riccietti & R. G., **Spurious Valleys, Spurious Minima and NP-hardness of Sparse Matrix Factorization With Fixed Support**, submitted]

[N. Gillis & F. Glineur, **Low-rank matrix approximation with weights or missing data is NP-hard**. *SIAM JMAA,* 2010]
[K. Kawaguchi. **Deep learning without poor local minima**. *NeurIPS,* 2016]
[X. Li & al. **Symmetry, saddle points, and global optimization landscape of nonconvex matrix factorization**. *IEEE TIT*, 2019]
[L. Venturi & al. **Spurious Valleys in One-hidden-layer Neural Network Optimization Landscapes**. *JMLR,* 2019]

To be or not to be tractable

Theorem 1 (hardness)

- Sparse matrix factorization ***with fixed support*** (I, J) is NP-hard

$$\min L(\mathbf{X}, \mathbf{Y}) := \|\mathbf{A} - \mathbf{X}\mathbf{Y}\|_F^2$$

$$\text{supp}(\mathbf{X}) \subseteq I, \ \text{supp}(\mathbf{Y}) \subseteq J$$

➡ very different from least squares !

Theorem 2 (easy instances)

- Under some condition on (I, J),
  - There is an ***efficient algorithm*** to compute a solution
  - For any $\mathbf{A}$, there is ***no spurious local valley***, and essentially ***no spurious local minimum***

Detailed proofs and condition: see [Q.-T. Le, E. Riccietti & R. G., **Spurious Valleys, Spurious Minima and NP-hardness of Sparse Matrix Factorization With Fixed Support**, submitted]

[N. Gillis & F. Glineur, **Low-rank matrix approximation with weights or missing data is NP-hard**. *SIAM JMAA,* 2010]
[K. Kawaguchi. **Deep learning without poor local minima**. *NeurIPS,* 2016]
[X. Li & al. **Symmetry, saddle points, and global optimization landscape of nonconvex matrix factorization**. *IEEE TIT*, 2019]
[L. Venturi & al. **Spurious Valleys in One-hidden-layer Neural Network Optimization Landscapes**. *JMLR,* 2019]

To be or not to be tractable

Theorem 1 (hardness)

- Sparse matrix factorization ***with fixed support*** (I, J) is NP-hard

$$\min L(\mathbf{X}, \mathbf{Y}) := \|\mathbf{A} - \mathbf{X}\mathbf{Y}\|_F^2$$

$$\mathrm{supp}(\mathbf{X}) \subseteq I, \ \mathrm{supp}(\mathbf{Y}) \subseteq J$$

➡ very different from least squares !

Theorem 2 (easy instances)

- Under some condition on (I, J),
  - There is an ***efficient algorithm*** to compute a solution
  - For any $\mathbf{A}$, there is ***no spurious local valley***, and essentially ***no spurious local minimum***

?

Detailed proofs and condition: see [Q.-T. Le, E. Riccietti & R. G., **Spurious Valleys, Spurious Minima and NP-hardness of Sparse Matrix Factorization With Fixed Support**, submitted]

[N. Gillis & F. Glineur, **Low-rank matrix approximation with weights or missing data is NP-hard**. *SIAM JMAA,* 2010]
[K. Kawaguchi. **Deep learning without poor local minima**. *NeurIPS,* 2016]
[X. Li & al. **Symmetry, saddle points, and global optimization landscape of nonconvex matrix factorization**. *IEEE TIT*, 2019]
[L. Venturi & al. **Spurious Valleys in One-hidden-layer Neural Network Optimization Landscapes**. *JMLR,* 2019]

Hardness *vs* landscape ?

Theorem 3

- There exists (I, J) and an explicit open set of matrices such that
  - For each $\mathbf{A}$ in the open set, ***the solution is known analytically***.
  - There exists $\mathbf{A}$ in the open set for which the problem ***has a spurious local valley***
  - There exists $\mathbf{A}$ in the open set for which the problem ***has a spurious local minimum***

➡ Lesson 2: *there may be life beyond gradient descent!*

Hardness *vs* landscape ?

Theorem 3

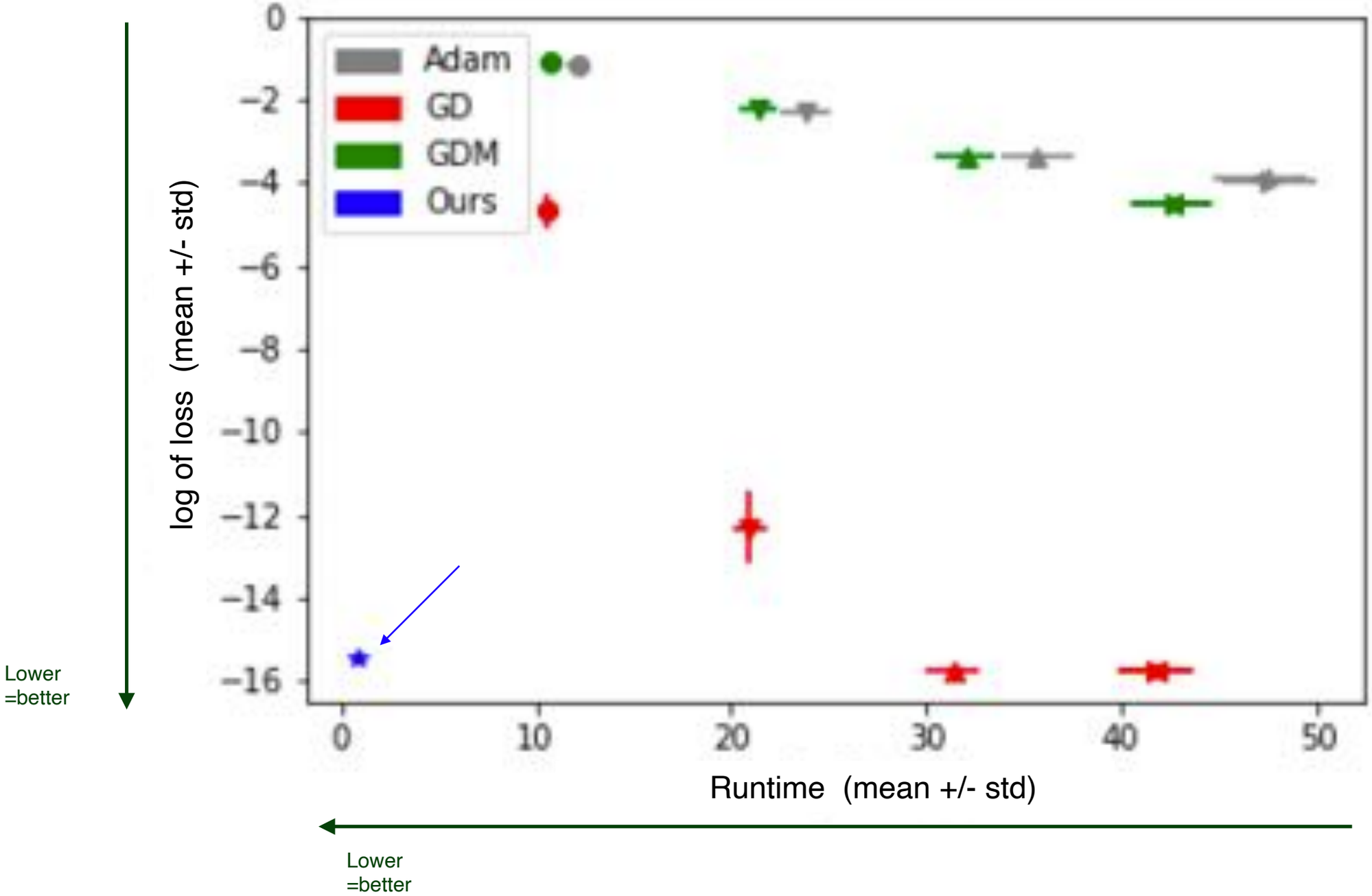
- There exists (I, J) and an explicit open set of matrices such that
  - For each $\mathbf{A}$ in the open set, ***the solution is known analytically***.
  - There exists $\mathbf{A}$ in the open set for which the problem ***has a spurious local valley***
  - There exists $\mathbf{A}$ in the open set for which the problem ***has a spurious local minimum***

➡ Lesson 2: *there may be life beyond gradient descent!*

***Indeed*, the algorithm of Theorem 2**

- is ***NOT*** based on gradient-descent
- exploits problem invariances and the SVD

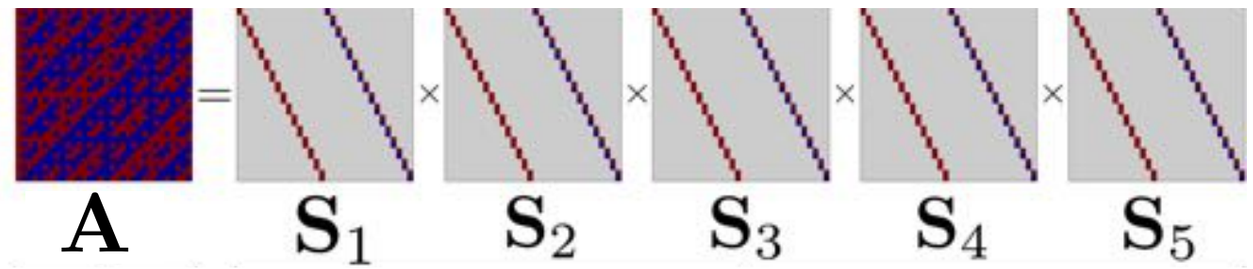
Precision *vs* runtime



GD(M) = gradient descent (with momentum)

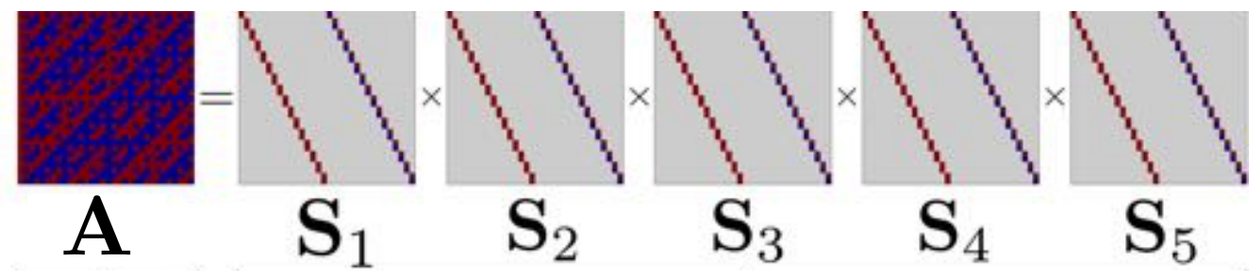
Butterfly factorization

- **Butterfly structure**
  - Common sparsity pattern for DCT, DST, DFT, Hadamard

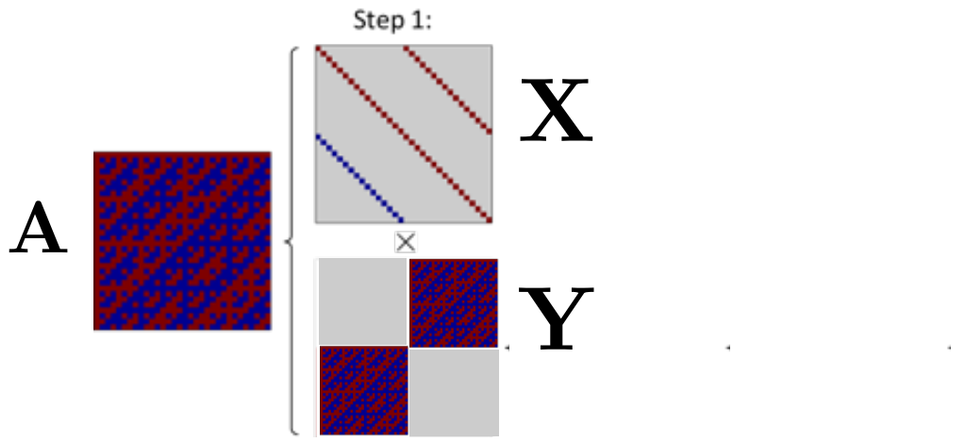


Butterfly factorization

- **Butterfly structure**
  - Common sparsity pattern for DCT, DST, DFT, Hadamard



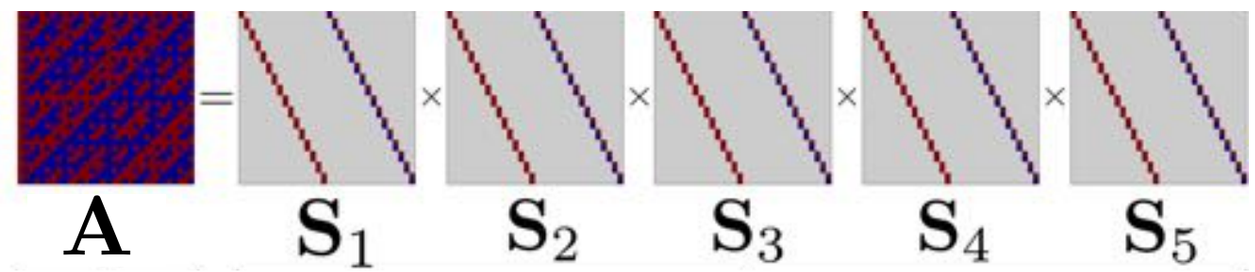
- **Hierarchical factorization approach**



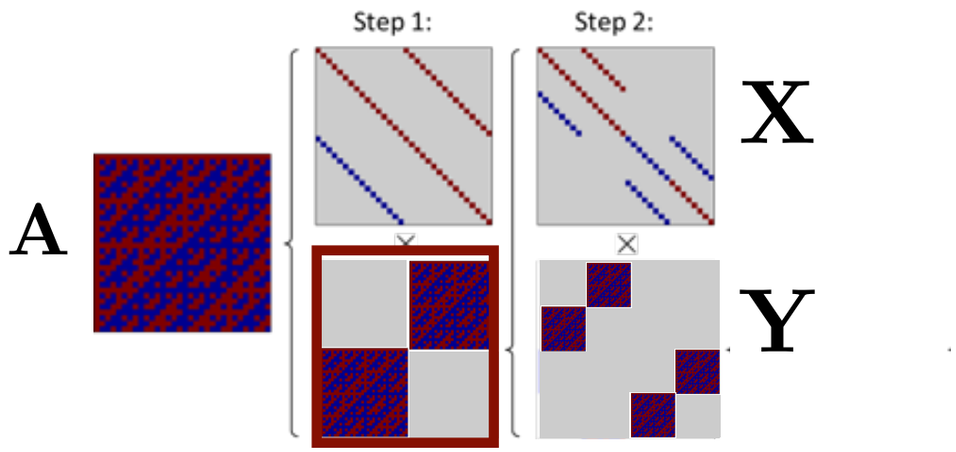
Butterfly factorization

Butterfly structure

- Common sparsity pattern for DCT, DST, DFT, Hadamard



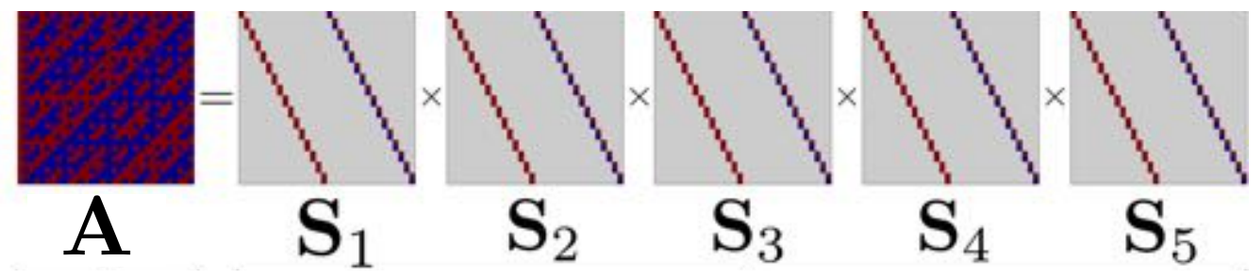
Hierarchical factorization approach



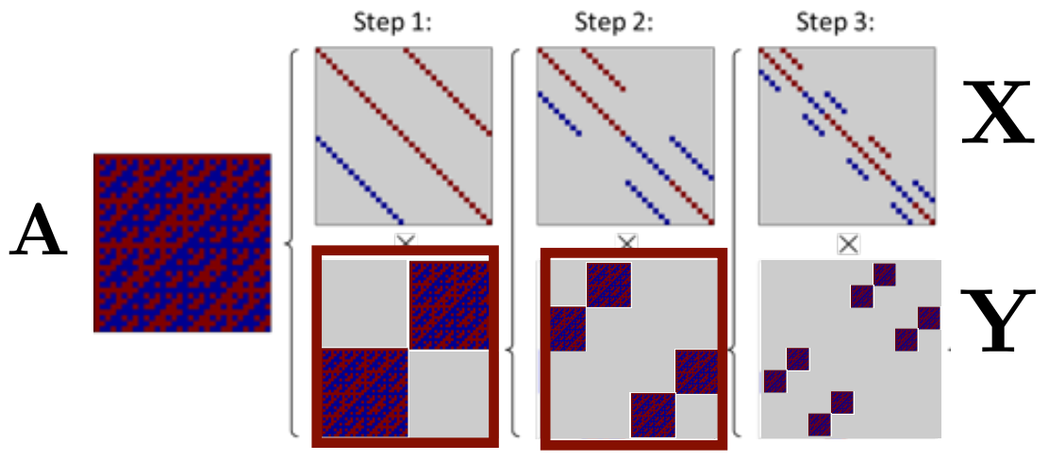
Butterfly factorization

Butterfly structure

- Common sparsity pattern for DCT, DST, DFT, Hadamard



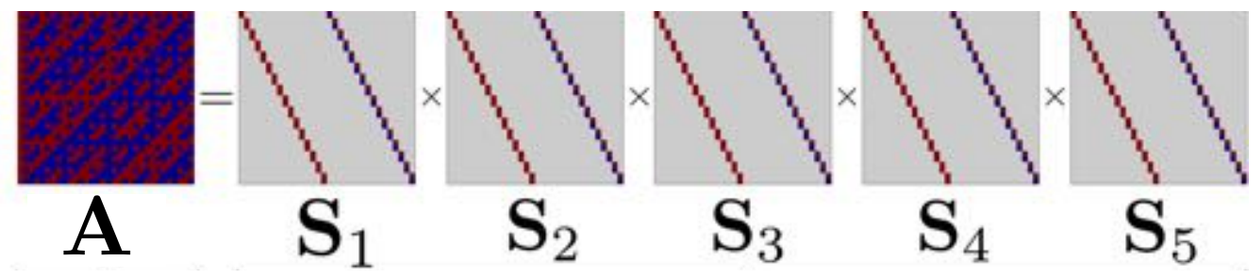
Hierarchical factorization approach



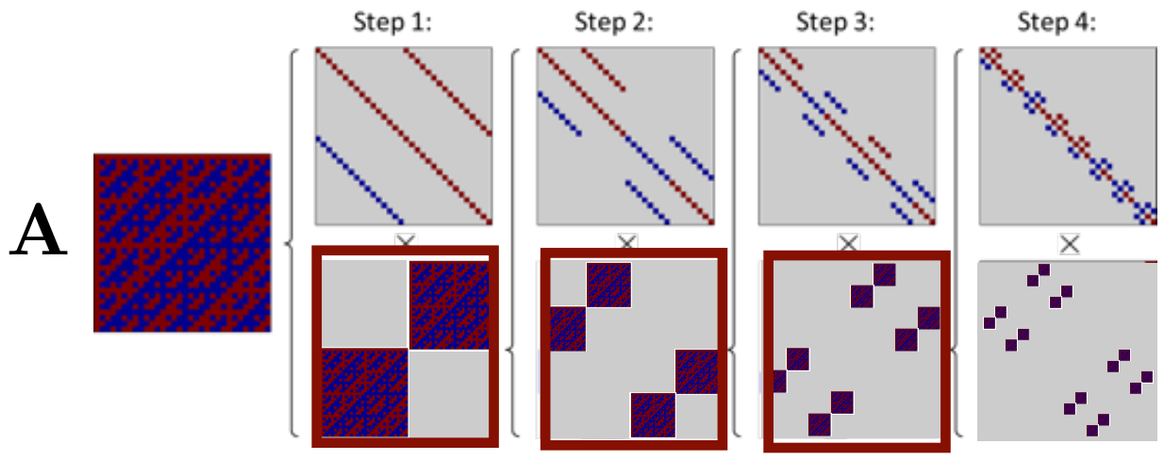
Butterfly factorization

Butterfly structure

- Common sparsity pattern for DCT, DST, DFT, Hadamard

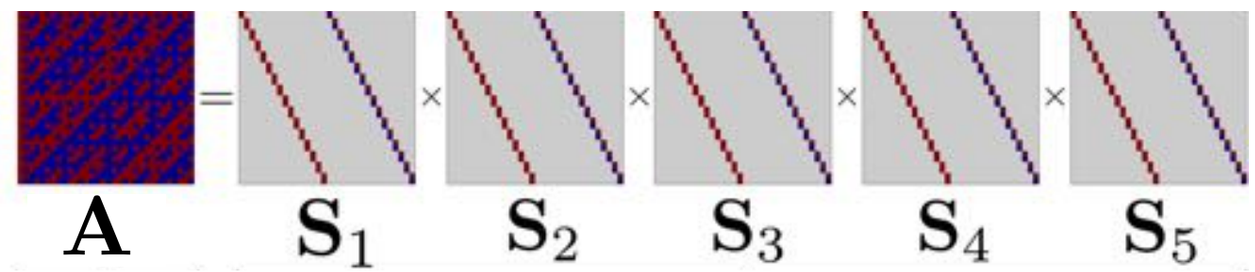


Hierarchical factorization approach

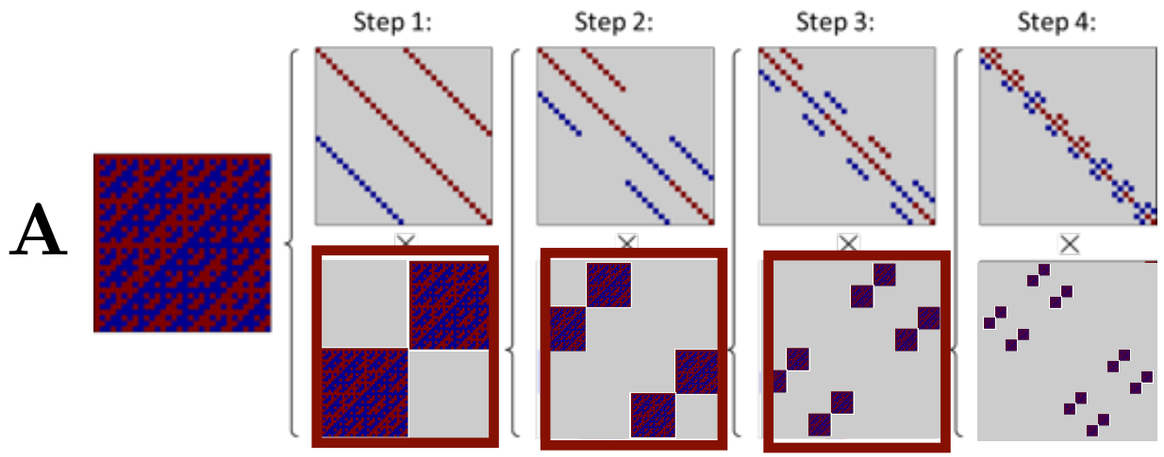


Butterfly factorization

- **Butterfly structure**
  - Common sparsity pattern for DCT, DST, DFT, Hadamard



- **Hierarchical factorization approach**



- **Tractability conditions on (I,J) hold at each step**

faust.inria.fr toolbox

-  &  wrappers (C++ core, GPU-compatible)

- PYPI install (or system packages)
  - https://pypi.org/project/pyfaust/
  - `pip install pyfaust`

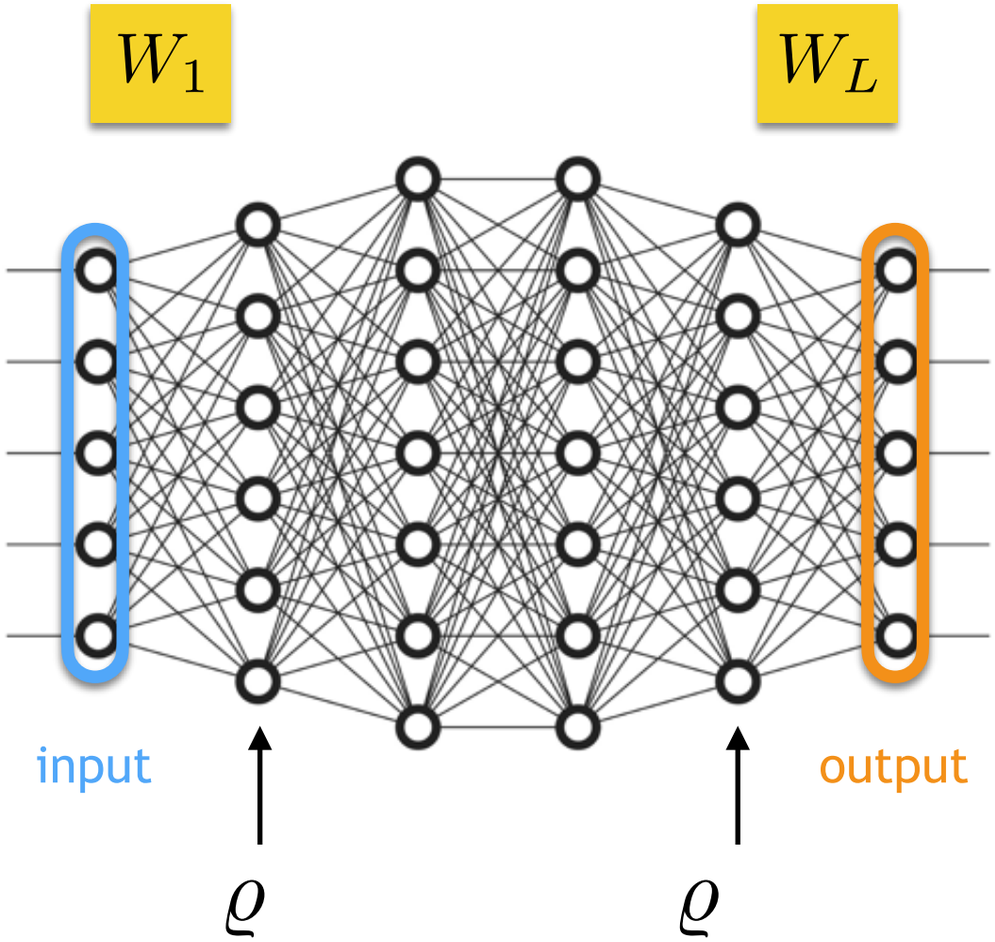
- Basic usage
  - Convert a matrix to a fast transform
    - e.g. `F = butterfly(A)`
  - Use it transparently (e.g. in inverse problems)
    - essentially replace `x = omp(y,A,k)` with `x = omp(y,F,k)`

    (for selected existing solvers)

- Sparsity: some lessons from inverse problems
- Two-layer sparsity and beyond
- **Rescaling-invariance in ReLU networks**
- An embedding of ReLU networks

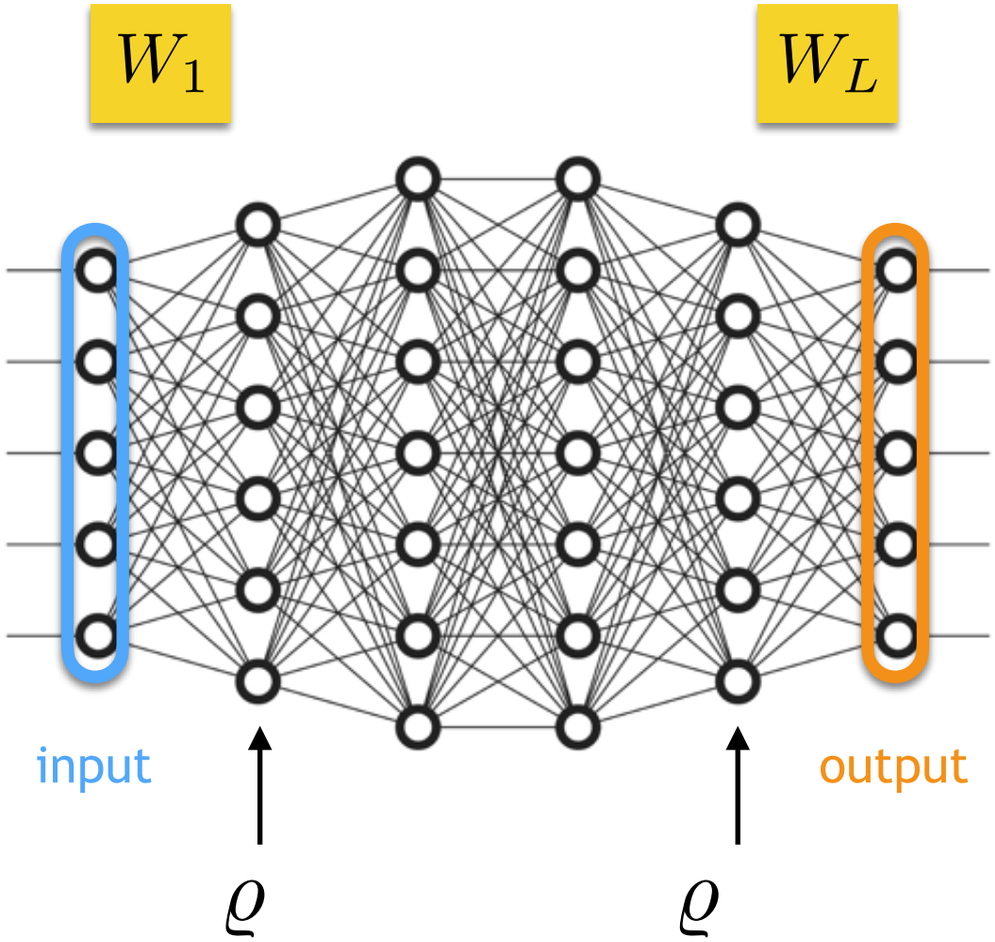
Feedforward neural networks

- **Feedforward network**
  - vector input $x \in \mathbb{R}^d$
  - parameters
    - L affine (“linear”) layers W_ℓ
    - L-1 (hidden) nonlinear layers
  - vector output $y \in \mathbb{R}^k$



Feedforward neural networks

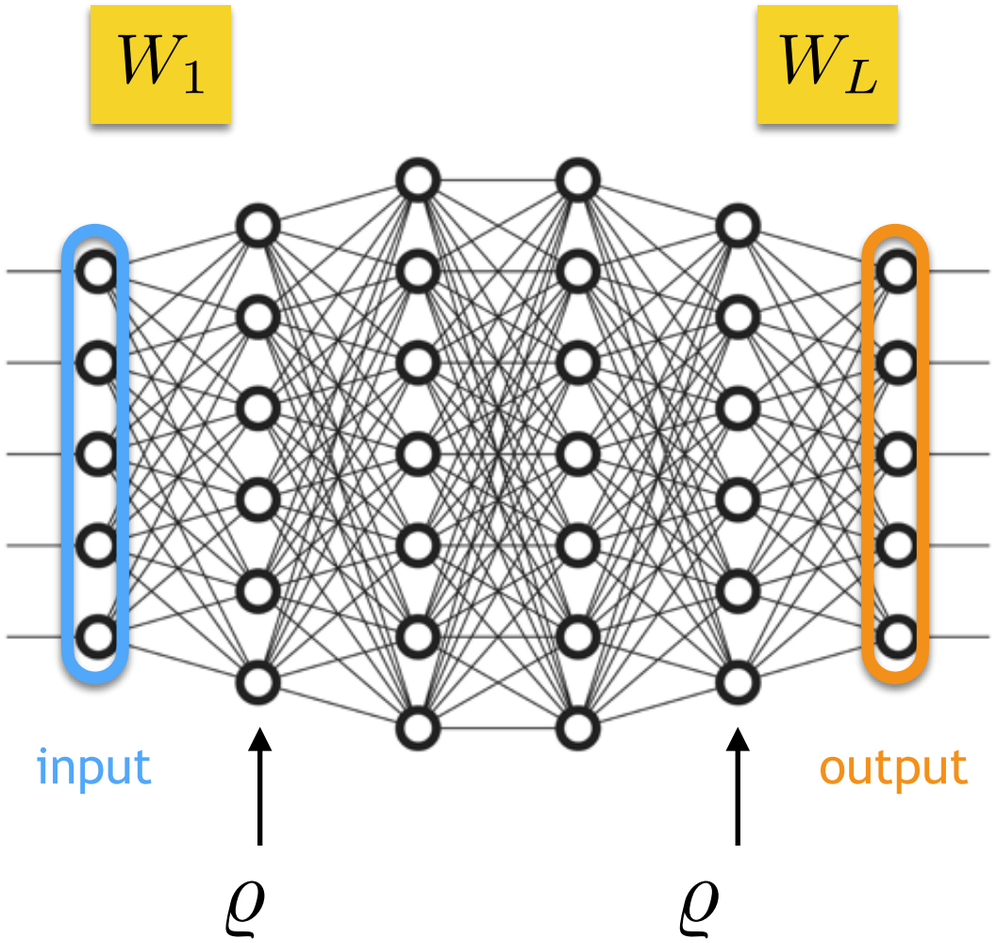
- **Feedforward network**
  - vector input $x \in \mathbb{R}^d$
  - parameters
    - L affine ("linear") layers W_ℓ
    - L-1 (hidden) nonlinear layers
  - vector output $y \in \mathbb{R}^k$
  - description $\theta = (W_\ell)_{\ell=1}^L$
  - ***realization*** $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$



$$f_\theta = W_L \circ \varrho \circ W_{L-1} \circ \cdots \circ \varrho \circ W_1$$

Feedforward neural networks

- **Feedforward network**
  - vector input $x \in \mathbb{R}^d$
  - parameters
    - L affine ("linear") layers W_ℓ
    - L-1 (hidden) nonlinear layers
  - vector output $y \in \mathbb{R}^k$
  - description $\theta = (W_\ell)_{\ell=1}^L$
  - ***realization*** $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$



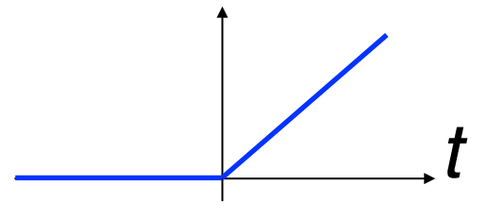
$$f_\theta = W_L \circ \varrho \circ W_{L-1} \circ \cdots \circ \varrho \circ W_1$$

- other ingredients: max-pooling, skip connections, conv … NOT IN THIS TALK

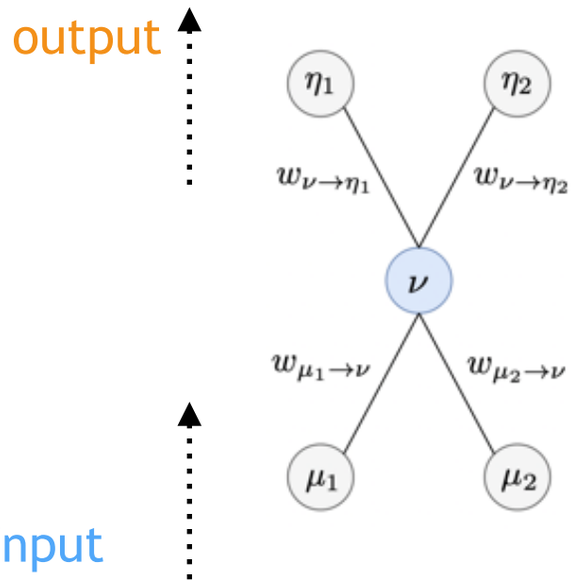
Invariance of ReLU-networks

- **The ReLU is positively homogeneous**

$$\varrho(t) = \texttt{ReLU}(t) = \max(t, 0) = t_+$$



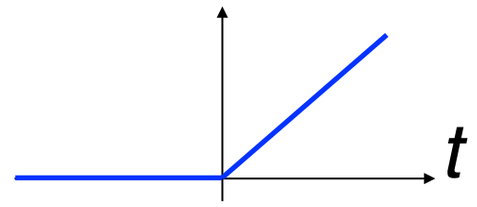
- **Rescaling** incoming/outgoing weights of a neuron



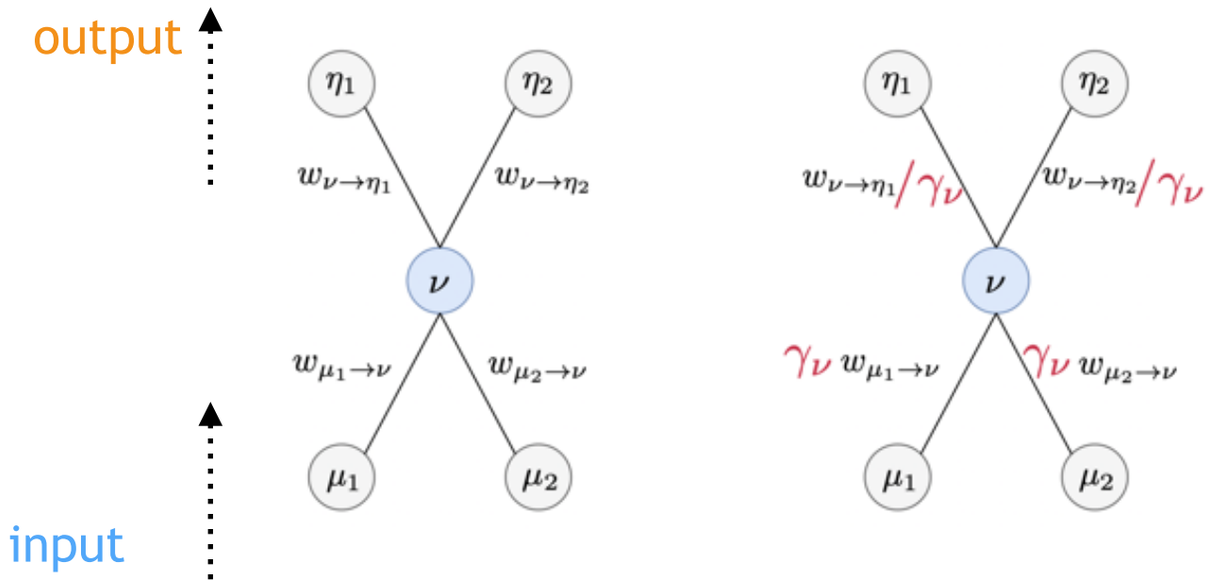
Invariance of ReLU-networks

- **The ReLU is positively homogeneous**

$$\varrho(t) = \texttt{ReLU}(t) = \max(t, 0) = t_+$$



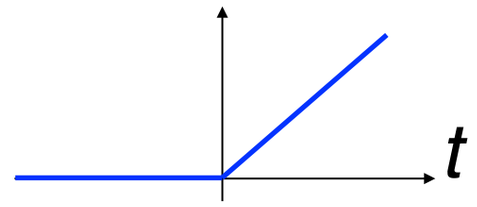
- **Rescaling** incoming/outgoing weights of a neuron



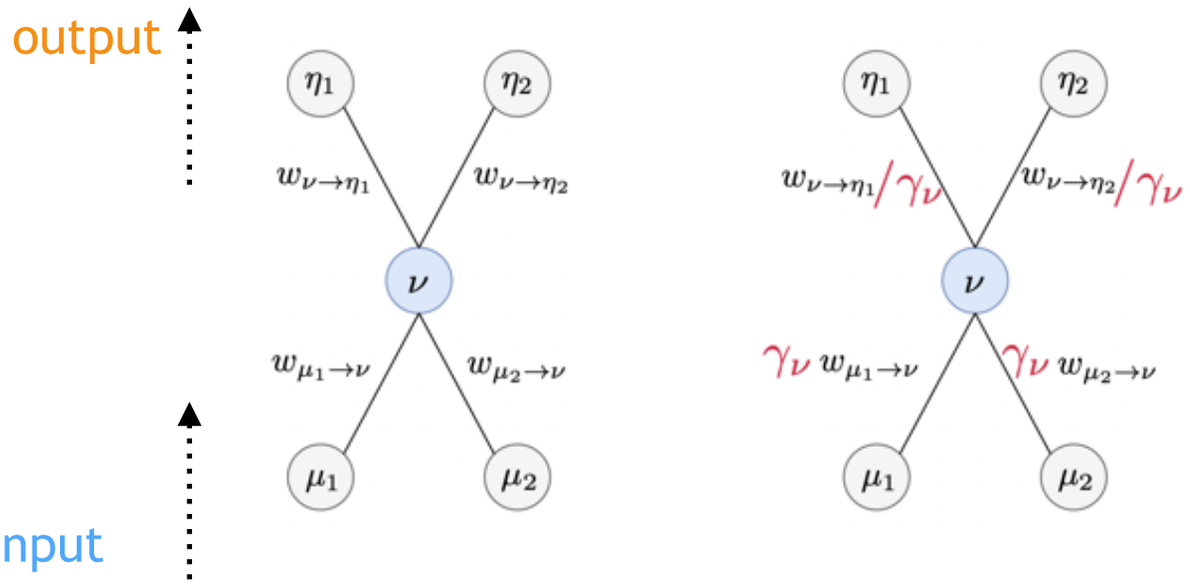
Invariance of ReLU-networks

- **The ReLU is positively homogeneous**

$$\varrho(t) = \texttt{ReLU}(t) = \max(t, 0) = t_+$$



- **Rescaling** incoming/outgoing weights of a neuron



- Notion of **scaling-equivalent parameters** $\theta' \sim_S \theta$
  - implies **functional equivalence** $f_\theta = f_{\theta'}$

Scaling-equivalent parameters …

- … are **always functionally equivalent**
- … but **can behave *wildly* differently**
  - during SGD optimization
    - see e.g. [B. Neyshabur & al, **Path-SGD - Path-Normalized Optimization in Deep Neural Networks**. *NeurIPS,* 2015]
  - under quantization
    - see e.g. [M. Nagel & al, **Data-Free Quantization Through Weight Equalization and Bias Correction**. *ICCV,* 2019]

*Can we **wander** in the equivalence class to improve some performance measure **without changing the network's prediction ?***

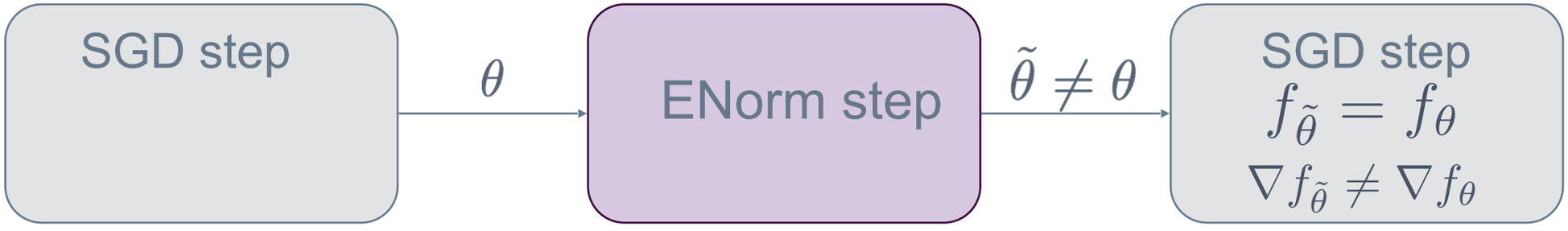
Benefits of scale-invariance: E-norm

- **Context**: stochastic gradient descent (SGD)

https://github.com/facebookresearch/enorm

Benefits of scale-invariance: E-norm

- **Context**: stochastic gradient descent (SGD)



- **Heuristic:** choose a « canonical » representer

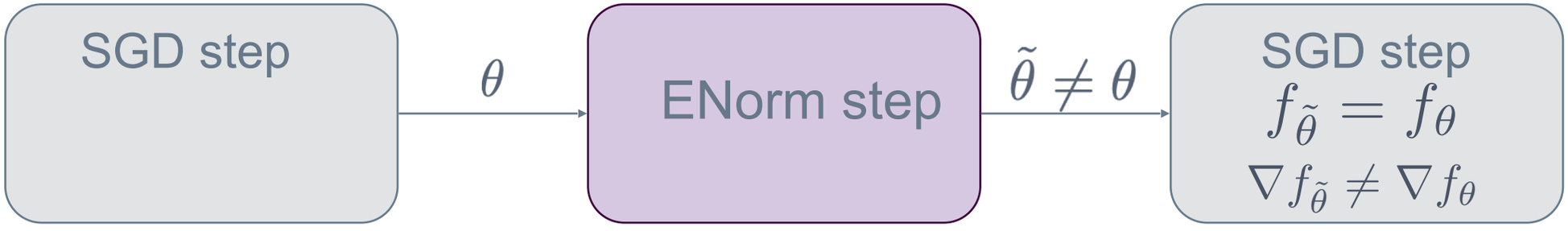
$$\tilde{\theta} = \arg \min_{\theta' \sim_S \theta} \|\theta'\|_p$$

  - Lp-norm *on weights* only, any p>0
  - Problem proved to ***equivalent to a convex one***

https://github.com/facebookresearch/enorm

Benefits of scale-invariance: E-norm

- **Context**: stochastic gradient descent (SGD)



- **Heuristic:** choose a « canonical » representer

$$\tilde{\theta} = \arg \min_{\theta' \sim_S \theta} \|\theta'\|_p$$

  - Lp-norm *on weights* only, any p>0
  - Problem proved to ***equivalent to a convex one***

- **ENorm algorithm**: Fast, Iterative, Sinkhorn*-like
  - block-coordinate descent
  - provably converges to « canonical » representant

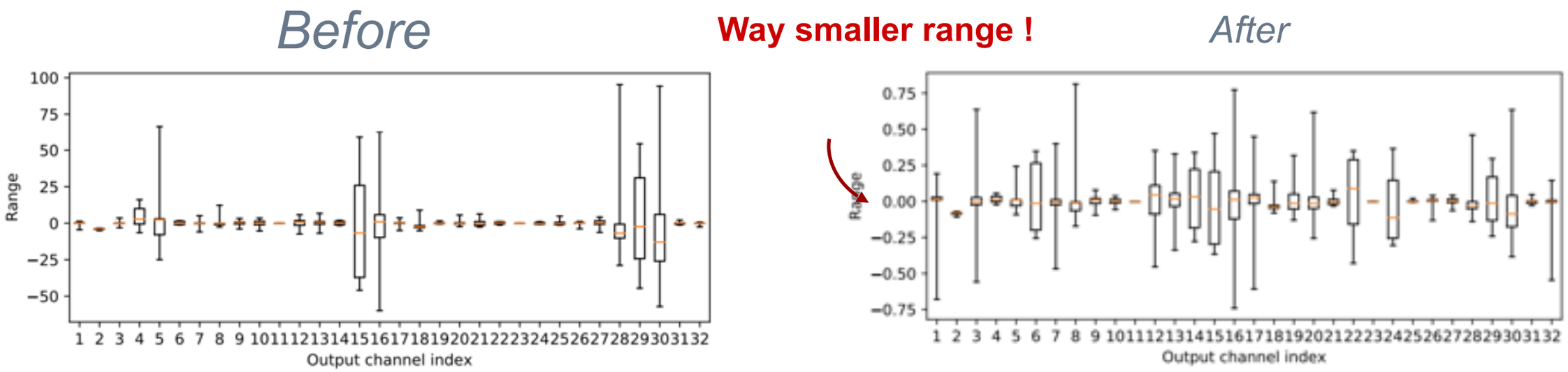
*[Sinkhorn & Knopp, 1967]

[P. Stock, B. Graham, R. G. & H. Jégou, **Equi-normalization of Neural Networks**. ICLR, 2019]

https://github.com/facebookresearch/enorm

Application to quantization

- [Nagel et al.] leverage ENorm for int8 quantization with Weight Equalization



[Nagel, van Baalen, Blankevoort & Welling, ICCV 2019]
Image credit: Nagel *et al.*

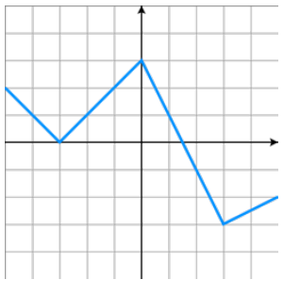
- Lesson 3: ~~beware of~~ scale-invariance !
  (try to) harness

- Sparsity: some lessons from inverse problems
- Two-layer sparsity and beyond
- Scale-invariance in ReLU networks
- **An embedding of ReLU networks**

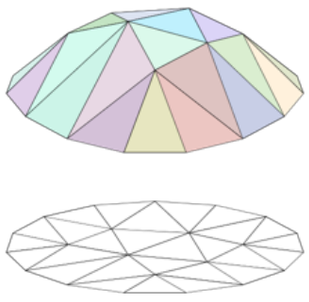
Realizations of ReLU-networks

- $f_\theta(x)$ **as a function of** x
  - Continuous and piecewise (affine) linear

- **d=1**



- **d>1**

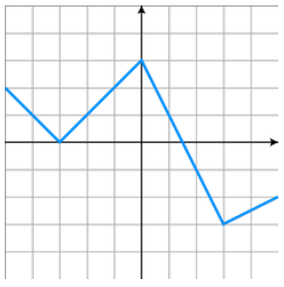


source: https://commons.wikimedia.org/wiki/File:Piecewise_linear_function2D.svg
(domaine public)

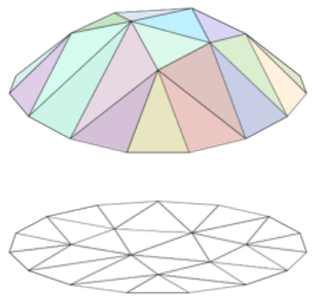
Realizations of ReLU-networks

- $f_\theta(x)$ **as a function of** x
  - Continuous and piecewise (affine) linear

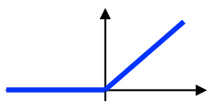
- **d=1**



- **d>1**



source: https://commons.wikimedia.org/wiki/File:Piecewise_linear_function2D.svg
(domaine public)



- **Activation vector (hidden neurons)**

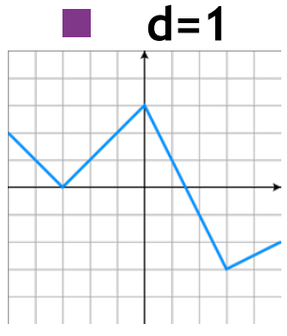
$$\mathbf{a}(\theta, x) := \{a_\nu(\theta, x)\}_{\nu \in H} \in \{0,1\}^H$$

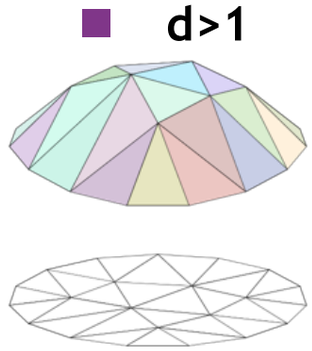
  - Linear regions: visible pieces
  - Activation region: *hidden* pieces
    - Do not always coincide

Realizations of ReLU-networks

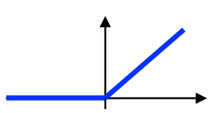
$f_\theta(x)$ as a function of x

- Continuous and piecewise (affine) linear





source: https://commons.wikimedia.org/wiki/File:Piecewise_linear_function2D.svg
(domaine public)



- **Activation vector (hidden neurons)**

$$\mathbf{a}(\theta, x) := \{a_\nu(\theta, x)\}_{\nu \in H} \in \{0,1\}^H$$

  - Linear regions: visible pieces
  - Activation region: *hidden* pieces
    - Do not always coincide

$f_\theta(x)$ as a function of θ

- Continuous and piecewise polynomial
- In the interior of an *activation region*
  - activation vector is ***locally constant***

$$\theta \mapsto \mathbf{a}(\theta, x)$$

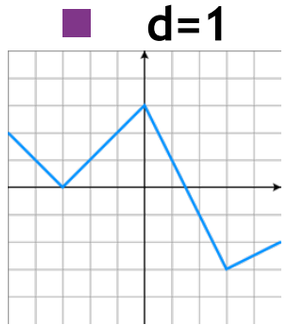
  - realization is *locally polynomial*

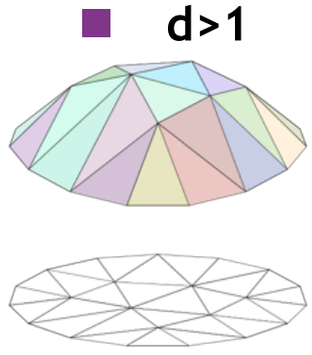
$$\theta \mapsto f_\theta(x)$$

Realizations of ReLU-networks

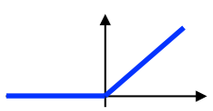
$f_\theta(x)$ as a function of x

- Continuous and piecewise (affine) linear





source: https://commons.wikimedia.org/wiki/File:Piecewise_linear_function2D.svg
(domaine public)



- **Activation vector (hidden neurons)**

$$\mathbf{a}(\theta, x) := \{a_\nu(\theta, x)\}_{\nu \in H} \in \{0,1\}^H$$

  - Linear regions: visible pieces
  - Activation region: *hidden* pieces
    - Do not always coincide

$f_\theta(x)$ as a function of θ

- Continuous and piecewise polynomial
- In the interior of an *activation region*
  - activation vector is ***locally constant***

$$\theta \mapsto \mathbf{a}(\theta, x)$$

  - realization is *locally polynomial*

$$\theta \mapsto f_\theta(x)$$

  - can be written as

$$f_\theta(x) = \langle \bar{\mathbf{a}}(\theta, x), \boldsymbol{\Phi}(\theta) \rangle$$

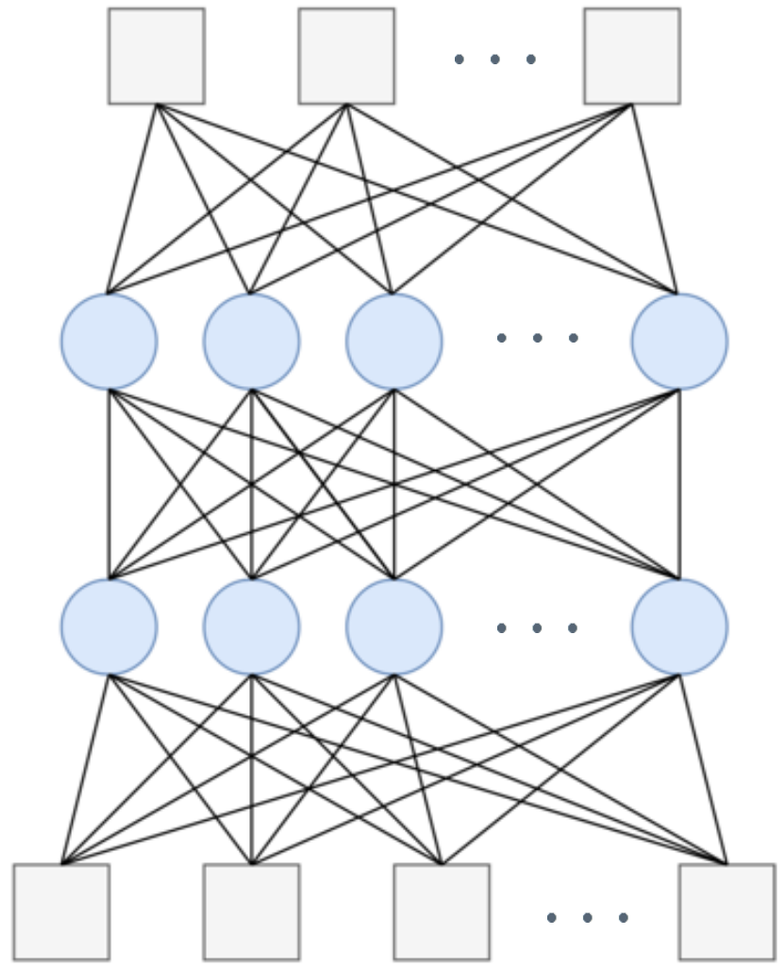
rescaling-invariant embedding

Embedding ReLU network parameters

Architecture

Neurons

Paths

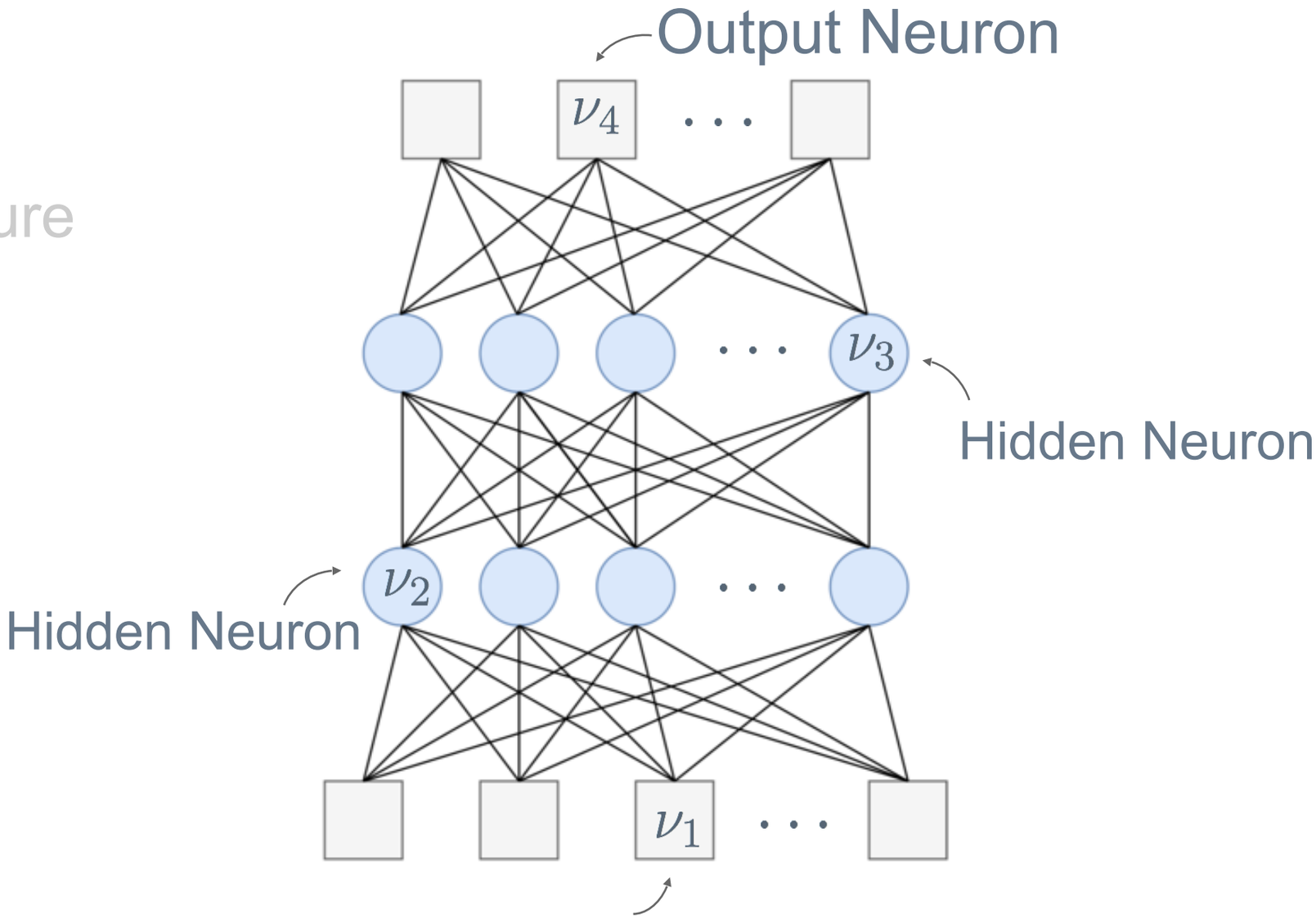


Embedding ReLU network parameters

Architecture

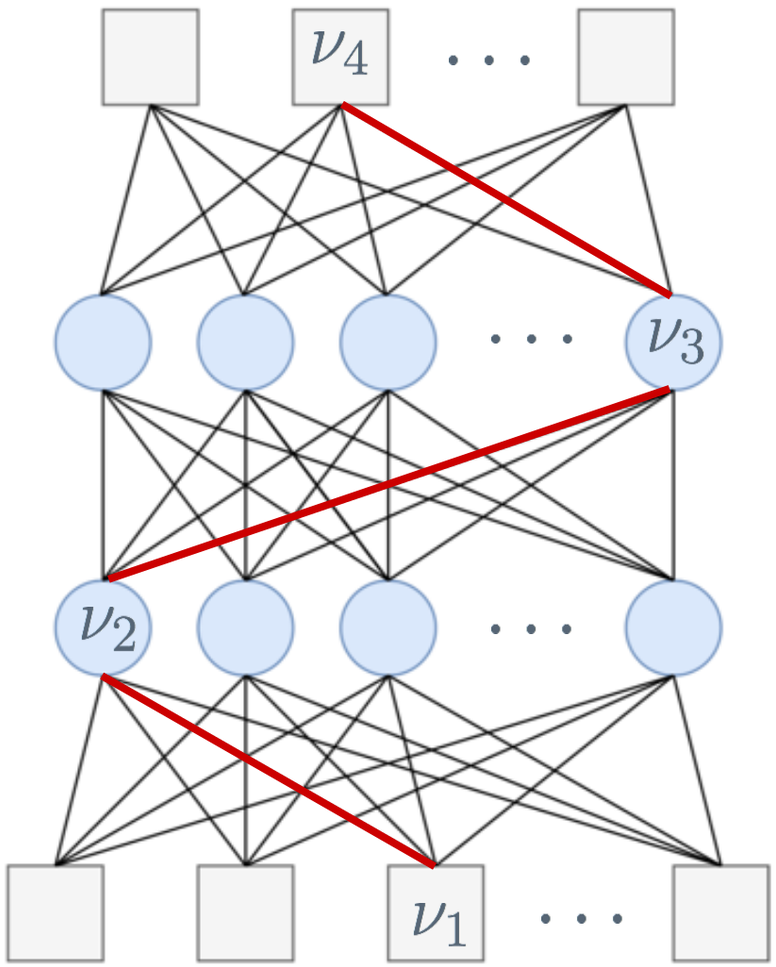
Neurons

Paths



Embedding ReLU network parameters

Architecture
Neurons
Paths



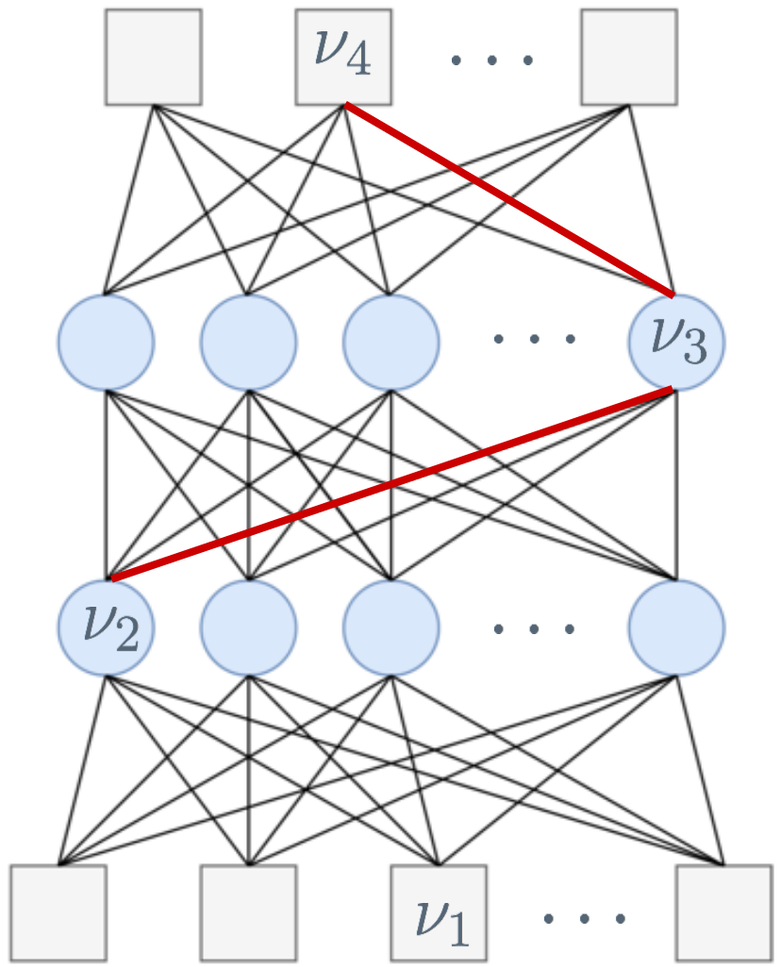
$$p = \nu_1 \to \nu_2 \to \nu_3 \to \nu_4$$

$p \in \mathcal{P}$ *full* path

$$\Phi_p(\theta) := \Pi_{e \in p} w_e$$

Embedding ReLU network parameters

Architecture
Neurons
Paths



$$p = \nu_1 \to \nu_2 \to \nu_3 \to \nu_4$$

$p \in \mathcal{P}$ *full* path

$$\Phi_p(\theta) := \Pi_{e \in p} w_e$$

$$q = \nu_\ell \to \ldots \to \nu_4$$

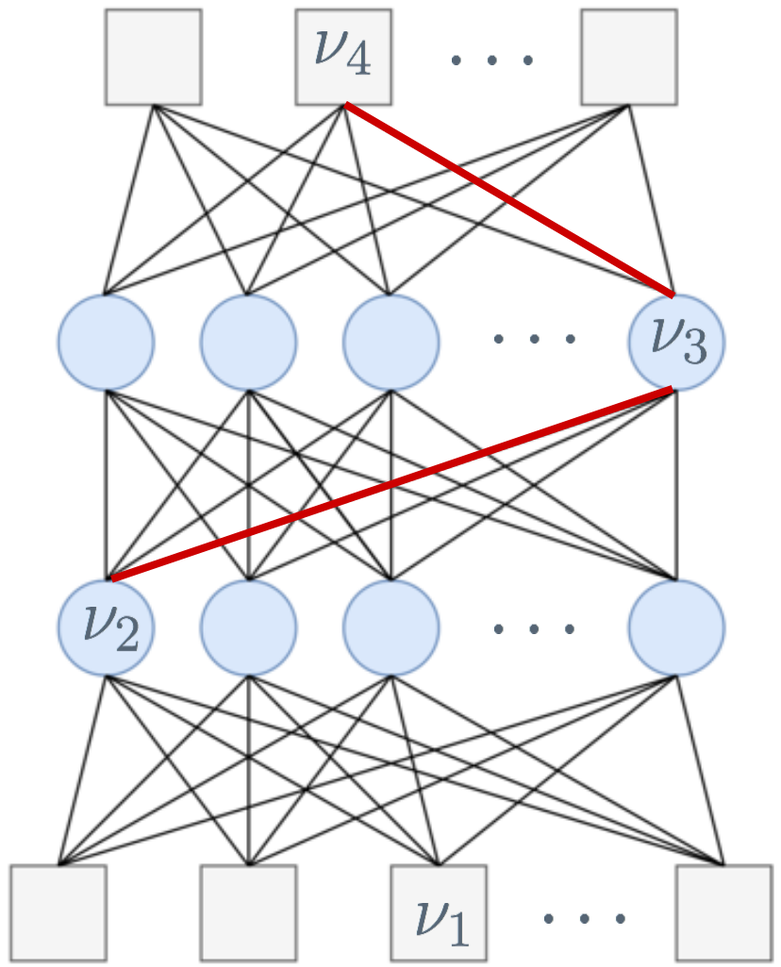
$q \in \mathcal{Q}$ *partial* path

$$\Phi_q(\theta) := b_\ell \Pi_{e \in q} w_e$$

appears implicitly, without biases,
in [Neyshabur & al 2015]
related idea: Segre embedding with *layerwise* scaling factor, see e.g.
[F. Malgouyres & J. Landsberg, **On the stable recovery of deep structured linear networks under sparsity constraints**. *PMLR,* 2020]
[F. Malgouyres & J. Landsberg, **Multilinear compressive sensing and an application to convolutional linear networks**. *SIAM,* 2018]

Embedding ReLU network parameters

Architecture
Neurons
Paths



$$p = \nu_1 \to \nu_2 \to \nu_3 \to \nu_4$$

$p \in \mathcal{P}$ *full* path

$$\Phi_p(\theta) := \Pi_{e \in p} w_e$$

$$q = \nu_\ell \to \ldots \to \nu_4$$

$q \in \mathcal{Q}$ *partial* path

$$\Phi_q(\theta) := b_\ell \Pi_{e \in q} w_e$$

and similarly: activations of full / partial paths $\bar{a}_p(\theta, x), \bar{a}_q(\theta, x)$

appears implicitly, without biases,
in [Neyshabur & al 2015]
related idea: Segre embedding with *layerwise* scaling factor, see e.g.
[F. Malgouyres & J. Landsberg, **On the stable recovery of deep structured linear networks under sparsity constraints**. *PMLR,* 2020]
[F. Malgouyres & J. Landsberg, **Multilinear compressive sensing and an application to convolutional linear networks**. *SIAM,* 2018]

Reverse-engineering networks ?

- **Scaling-equivalence *vs* functional equivalence ?**
  - One direction always true

$$\theta' \sim_S \theta \xrightarrow{\text{any parameters}} \boldsymbol{\Phi}(\theta') = \boldsymbol{\Phi}(\theta) \xrightarrow{\text{any parameters}} f_{\theta'} = f_\theta \quad \text{on } \mathcal{X} = \mathbb{R}^d$$

$$\texttt{sign}(\theta') = \texttt{sign}(\theta)$$

Reverse-engineering networks ?

- **Scaling-equivalence *vs* functional equivalence ?**
  - One direction always true

$$\theta' \sim_S \theta \xrightarrow{\text{any parameters}} \mathbf{\Phi}(\theta') = \mathbf{\Phi}(\theta) \xrightarrow{\text{any parameters}} f_{\theta'} = f_\theta \quad \text{on } \mathcal{X} = \mathbb{R}^d$$

$$\texttt{sign}(\theta') = \texttt{sign}(\theta)$$

- **Hard direction = converse**
  - see e.g. [C. Fefferman, **Reconstructing a neural net from its output**. *Revista Matemática Iberoamericana*, 1994]
    [D. Rolnick & K. Kording, **Reverse-engineering deep ReLU networks**. *ICML*, 2020]

Reverse-engineering networks ?

- **Scaling-equivalence *vs* functional equivalence ?**
  - One direction always true

$$\theta' \sim_S \theta$$

any parameters

$$\mathbf{\Phi}(\theta') = \mathbf{\Phi}(\theta)$$

$$\texttt{sign}(\theta') = \texttt{sign}(\theta)$$

« admissible » parameters

any parameters

$$\text{on } \mathcal{X} = \mathbb{R}^d$$
$$f_{\theta'} = f_\theta$$

- **Hard direction = converse**
  - see e.g. [C. Fefferman, **Reconstructing a neural net from its output**. *Revista Matemática Iberoamericana*, 1994]
    [D. Rolnick & K. Kording, **Reverse-engineering deep ReLU networks**. *ICML*, 2020]

Reverse-engineering networks ?

- **Scaling-equivalence *vs* functional equivalence ?**
  - One direction always true

$\theta' \sim_S \theta$ — *any* parameters → $\boldsymbol{\Phi}(\theta') = \boldsymbol{\Phi}(\theta)$ — « admissible » parameters

$\boldsymbol{\Phi}(\theta') = \boldsymbol{\Phi}(\theta)$ — *any* parameters → $f_{\theta'} = f_\theta$ on $\mathcal{X} = \mathbb{R}^d$ — locally, if « non-degenerate »

$\texttt{sign}(\theta') = \texttt{sign}(\theta)$

$f_{\theta'} = f_\theta$ on some finite $\mathcal{X} \subseteq \mathbb{R}^d$

- **Hard direction = converse**
  - see e.g. [C. Fefferman, **Reconstructing a neural net from its output**. *Revista Matemática Iberoamericana*, 1994]
    [D. Rolnick & K. Kording, **Reverse-engineering deep ReLU networks**. *ICML*, 2020]

Reverse-engineering networks ?

Scaling-equivalence *vs* functional equivalence ?

- One direction always true

$\theta' \sim_S \theta$ — *any* parameters → $\mathbf{\Phi}(\theta') = \mathbf{\Phi}(\theta)$ — *any* parameters → on $\mathcal{X} = \mathbb{R}^d$ $f_{\theta'} = f_\theta$ on some finite $\mathcal{X} \subseteq \mathbb{R}^d$

$\mathbf{\Phi}(\theta') = \mathbf{\Phi}(\theta)$ ⇐ « admissible » parameters

$\mathtt{sign}(\theta') = \mathtt{sign}(\theta)$

$f_{\theta'} = f_\theta$ ⇐ locally, if « non-degenerate »

$f_{\theta'} = f_\theta$ ⇐ globally, up to permutations?

Hard direction = converse

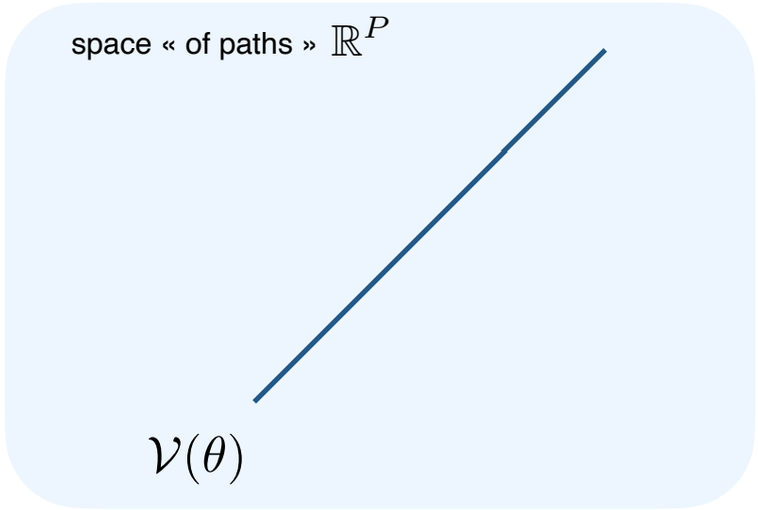
- see e.g. [C. Fefferman, **Reconstructing a neural net from its output**. *Revista Matemática Iberoamericana*, 1994]
  [D. Rolnick & K. Kording, **Reverse-engineering deep ReLU networks**. *ICML*, 2020]

Non-degeneracy

- **Recall** $f_\theta(x) = \langle \bar{\mathbf{a}}(\theta, x), \mathbf{\Phi}(\theta) \rangle$

  binary activation — path-space embedding

- **Definitions**
  - Activation space: $\mathcal{V}(\theta) \approx \mathtt{span}^{\perp}(\bar{\mathbf{a}}(\theta, x), x \in \mathbb{R}^d)$

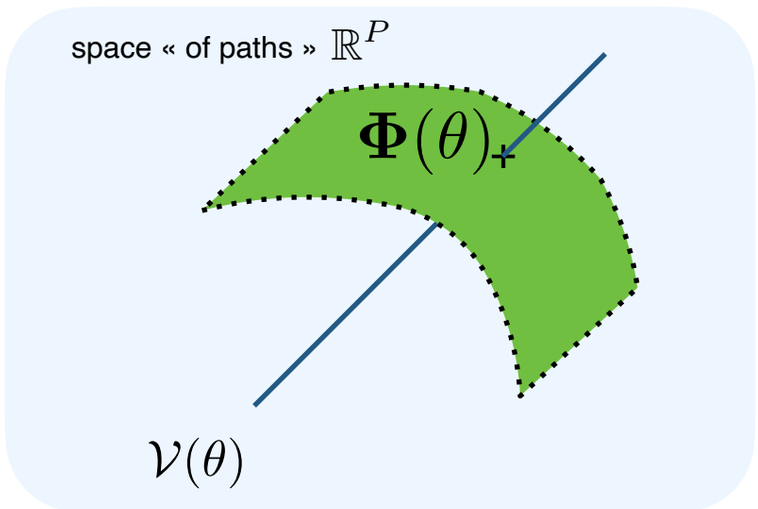


Non-degeneracy

binary activation

path-space embedding

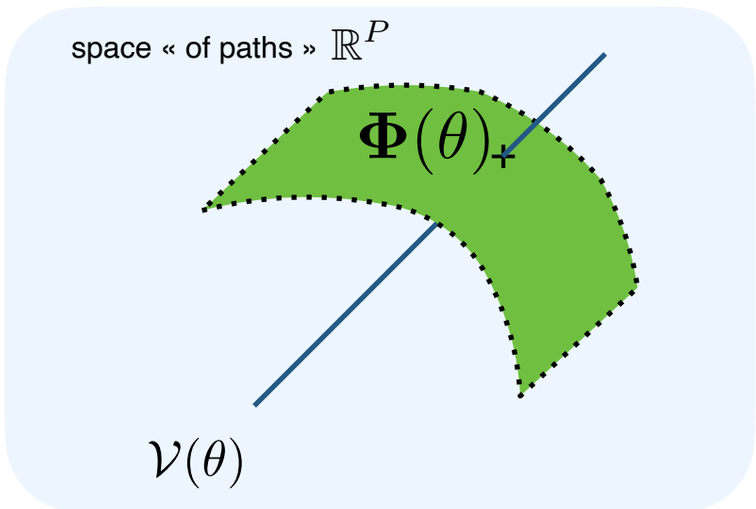
- **Recall** $f_\theta(x) = \langle \bar{\mathbf{a}}(\theta, x), \mathbf{\Phi}(\theta) \rangle$
- **Definitions**
  - Activation space: $\mathcal{V}(\theta) \approx \mathtt{span}^{\perp}(\bar{\mathbf{a}}(\theta, x), x \in \mathbb{R}^d)$
  - **Non-degenerate** parameter: *locally around* θ



$$\mathbf{\Phi}(\theta') - \mathbf{\Phi}(\theta) \in \mathcal{V}(\theta) \Rightarrow \mathbf{\Phi}(\theta') = \mathbf{\Phi}(\theta)$$

- **Theorem:**

Non-degeneracy



binary activation — path-space embedding

- **Recall** $f_\theta(x) = \langle \bar{\mathbf{a}}(\theta, x), \mathbf{\Phi}(\theta) \rangle$
- **Definitions**
  - Activation space: $\mathcal{V}(\theta) \approx \mathtt{span}^{\perp}(\bar{\mathbf{a}}(\theta, x), x \in \mathbb{R}^d)$
  - **Non-degenerate** parameter: *locally around* θ

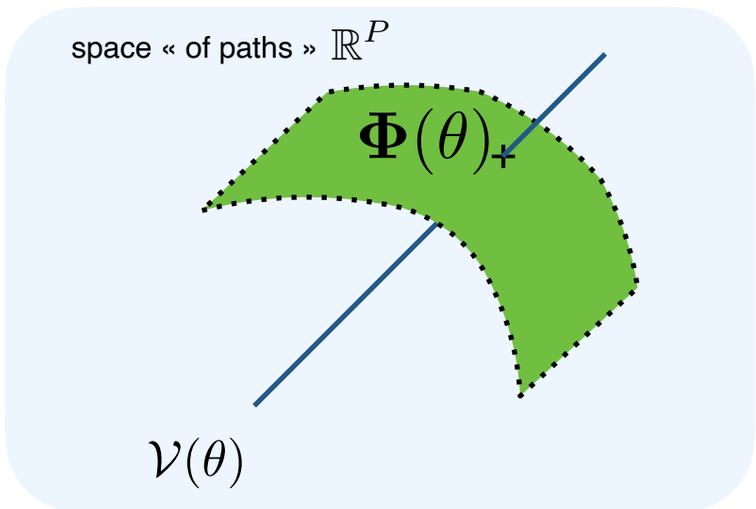
$$\mathbf{\Phi}(\theta') - \mathbf{\Phi}(\theta) \in \mathcal{V}(\theta) \Rightarrow \mathbf{\Phi}(\theta') = \mathbf{\Phi}(\theta)$$

- **Theorem:**
  - θ is non-degenerate *if, and only if* there is a **finite set** $\mathcal{X} \subseteq \mathbb{R}^d$ such that: for θ' close enough to θ

$$f_{\theta'} \overset{\text{on } \mathcal{X}}{=} f_\theta \quad \Longrightarrow \quad \theta' \sim_S \theta$$

  - **size** of the finite set $\mathcal{X}$ linked to dimension of $\mathcal{V}(\theta)$

Non-degeneracy



- **Recall** $f_\theta(x) = \langle \bar{\mathbf{a}}(\theta, x), \mathbf{\Phi}(\theta) \rangle$

  binary activation — path-space embedding

- **Definitions**
  - Activation space: $\mathcal{V}(\theta) \approx \mathtt{span}^{\perp}(\bar{\mathbf{a}}(\theta, x), x \in \mathbb{R}^d)$
  - **Non-degenerate** parameter *locally around* θ

$$\mathbf{\Phi}(\theta') - \mathbf{\Phi}(\theta) \in \mathcal{V}(\theta) \Rightarrow \mathbf{\Phi}(\theta') = \mathbf{\Phi}(\theta)$$

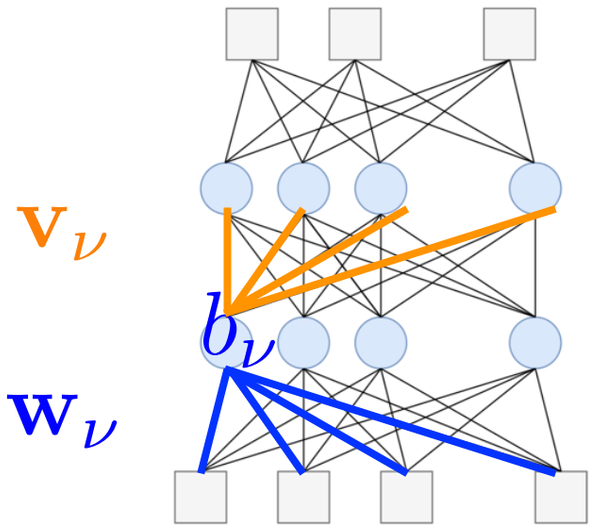
- **Theorem:**
  - θ is non-degenerate *if, and only if* there is a **finite set** $\mathcal{X} \subseteq \mathbb{R}^d$ such that: for θ' close enough to θ

$$f_{\theta'} \overset{\text{on } \mathcal{X}}{=} f_\theta \quad \Longrightarrow \quad \theta' \sim_S \theta$$

  - **size** of the finite set $\mathcal{X}$ linked to dimension of $\mathcal{V}(\theta)$

Twin neurons, and irreducibility

- **For each neuron**
  - bias
  - vectors of incoming / outgoing weights

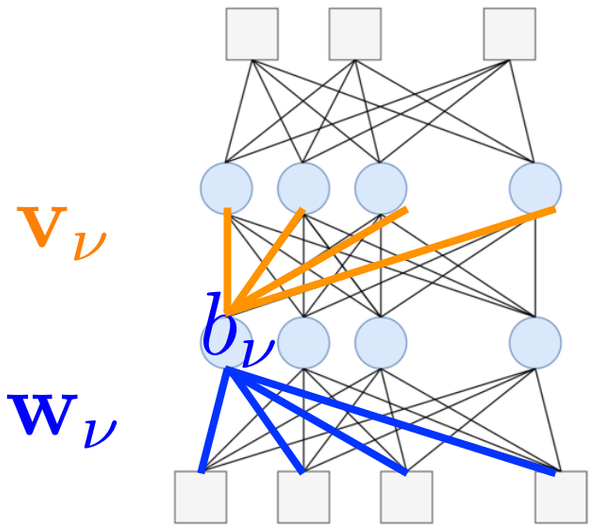


- **Definitions**
  - ***Twin neurons***
    - if $(\mathbf{w}_\nu, b_\nu) = \lambda(\mathbf{w}_{\nu'}, b_{\nu'})$
    - positive / negative twins depending on sign of λ
  - ***Irreducible***
    - if for every subset T of neurons of every layer we have

$$\sum_{\nu \in T} \mathbf{v}_\nu \mathbf{w}_\nu^\top \neq 0$$

Twin neurons, and irreducibility

- **For each neuron**
  - bias
  - vectors of incoming / outgoing weights

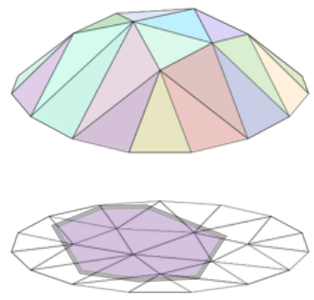


- **Definitions**
  - ***Twin neurons***
    - if $(\mathbf{w}_\nu, b_\nu) = \lambda(\mathbf{w}_{\nu'}, b_{\nu'})$
    - positive / negative twins depending on sign of λ
  - ***Irreducible*** θ
    - if for every subset T of neurons of every layer we have

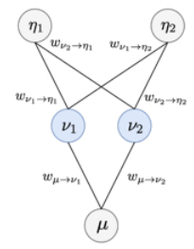
$$\sum_{\nu \in T} \mathbf{v}_\nu \mathbf{w}_\nu^\top \neq 0$$

Global identifiability ?

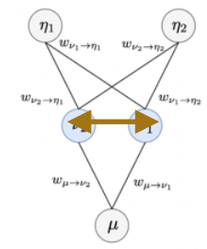
- **Definition: PS-identifiable parameter**
  - up to scaling & ***layerwise permutations***
  - from some set $\mathcal{X} \subseteq \mathbb{R}^d$



θ



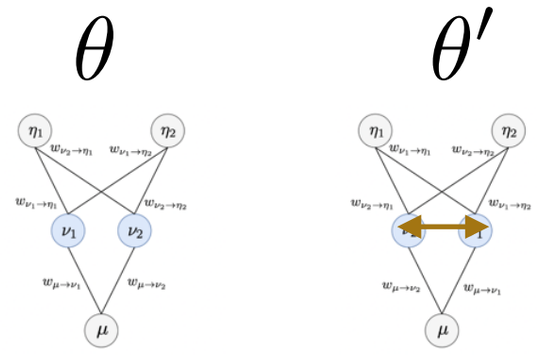
θ'

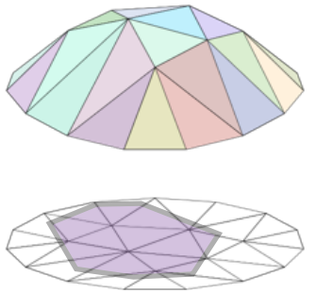


Global identifiability ?

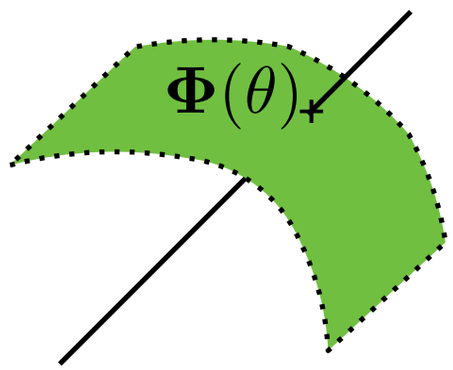
- **Definition: PS-identifiable parameter**
  - up to scaling & ***layerwise permutations***
  - from some set $\mathcal{X} \subseteq \mathbb{R}^d$

θ θ'



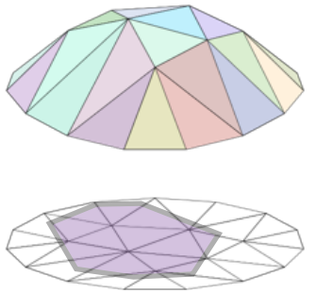


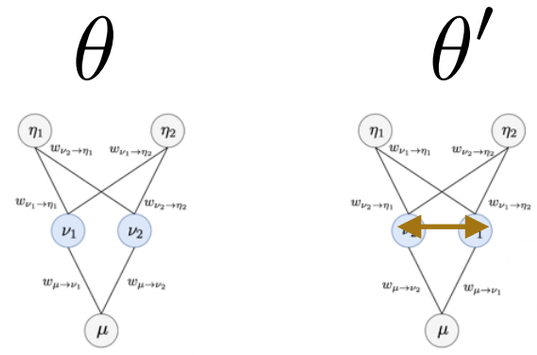
- **Theorem**
  - If θ is PS-identifiable from some $\mathcal{X} \subseteq \mathbb{R}^d$ then
    - It is irreducible
    - It is *locally* identifiable, *up to scaling only*
      - implies absence of *positive* twin neurons
    - Case of *bounded* $\mathcal{X}$: no negative twins either
    - Case of *finite* $\mathcal{X}$: θ is non-degenerate



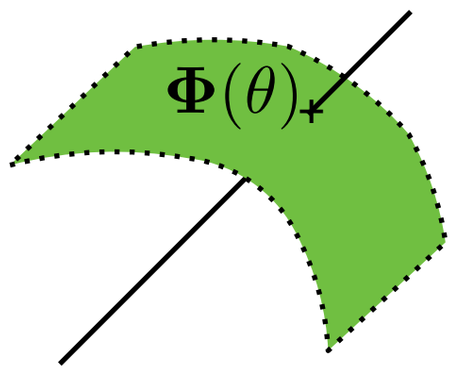
Global identifiability ?

- **Definition: PS-identifiable parameter**
  - up to scaling & ***layerwise permutations***
  - from some set $\mathcal{X} \subseteq \mathbb{R}^d$





- **Theorem**
  - If θ is PS-identifiable from some $\mathcal{X} \subseteq \mathbb{R}^d$ then
    - It is irreducible
    - It is *locally* identifiable, *up to scaling only*
      - implies absence of *positive* twin neurons
    - Case of *bounded* $\mathcal{X}$: no negative twins either
    - Case of *finite* $\mathcal{X}$: θ is non-degenerate



- **Converse for shallow networks:**
  - PS-identifiable from bounded $\mathcal{X}$ **iff** no twins & irreducible

- Sparsity: some lessons from inverse problems
- Two-layer sparsity and beyond
- Scale-invariance in ReLU networks
- An embedding of ReLU networks
- **Wrap-up**

Take Home Messages

- Sparse know-how can break in multilayer setting
  - forget L1 regularization ? beware of rescaling
  - NP-hardness even with fixed, known support
- Life beyond SGD
  - efficient sparse factorization with no gradient descent for certain fixed multilayer sparsity patterns
- Harnessing rescale equivalence in ReLU networks
  - a scale-invariant embedding
  - applications to identifiability analysis

Perspectives

- **Leveraging the full potential of the embedding**
  - Optimization on natural manifold (~path-SGD)
  - Quantization beyond heuristics
  - « Canonical » representer in equivalence class
- **Identifiability: a step towards explainability ?**
  - Characterizing PS-identifiability for deep networks
    - including with convolution layers and/or (structured) sparsity
      - two layer linear sparse: [L. Zheng, E. Riccietti & R. G., **Identifiability in Exact Sparse Matrix Factorization**. *working draft*, 2021]
  - Intuitive meaning of « irreducibility »
  - Sampling complexity and role of activation spaces
- **Beyond ReLU invariances**
  - e.g. in transformers

Quoc-Tung Le

Elisa Riccietti

Pierre Stock

Hervé Jégou

Léon Zheng

and also: Hakim Hadj-Djilani, Nicolas Bellot, Thomas Gautrais, Adrien Leman, …