TU Delft

Graph Filters

with applications to
Distributed Optimization
and Neural Networks

Geert Leus, Elvin Isufi, Mario Coutino

{ g.j.t.leus; e.isufi-1; m.a.coutinominguez } @tudelft.nl

Delft University of Technology

Acknowledgements

- Sundeep Chepuri (IISc)
- Paolo Di Lorenzo (Sapienza)
- Fernando Gamma (UPenn)
- Bianca Iancu (TU Delft)
- Jiani Liu (TU Delft)
- Andreas Loukas (EPFL)
- Antonio Marques (URJC)
- Matthew Morency (TU Delft)
- Alberto Natali (TU Delft)
- Alejandro Ribeiro (UPenn)
- Santiago Segarra (Rice)
- Andrea Simonetto (IBM)
- Tomas Sipco (TU Delft)

NWO

ASPIRE

KAUST Sensor Initiative

CONACYT

TU Delft Delft University of Technology

Tutorial break down

VIDEO 1:

- ◉ Part I Graph Signal Processing and Graph Filters
  - Introduction to GSP
  - Graph filters, applications, design and implementation aspects
- ◉ Part II Graph Filters for Distributed Optimization 1
  - Motivation and general concept
  - Applications and clarifying examples

Tutorial break down

VIDEO 1:

- ◉ Part I Graph Signal Processing and Graph Filters
  - Introduction to GSP
  - Graph filters, applications, design and implementation aspects
- ◉ Part II Graph Filters for Distributed Optimization 1
  - Motivation and general concept
  - Applications and clarifying examples

VIDEO 2:

- ◉Part III Graph Filters for Distributed Optimization 2
  - Asynchronous implementation
  - Cascaded implementation
- ◉Part IV Graph Filters for Neural Networks
  - Motivation and general concept
  - Applications and clarifying examples

part 1
graph signal processing and graph filters

part 1 :: overview

- ◉ Introduction to graph signal processing
  - Motivation
  - Mathematical formulation
  - Graph Fourier transform
  - Time-domain as a graph

part 1 :: overview

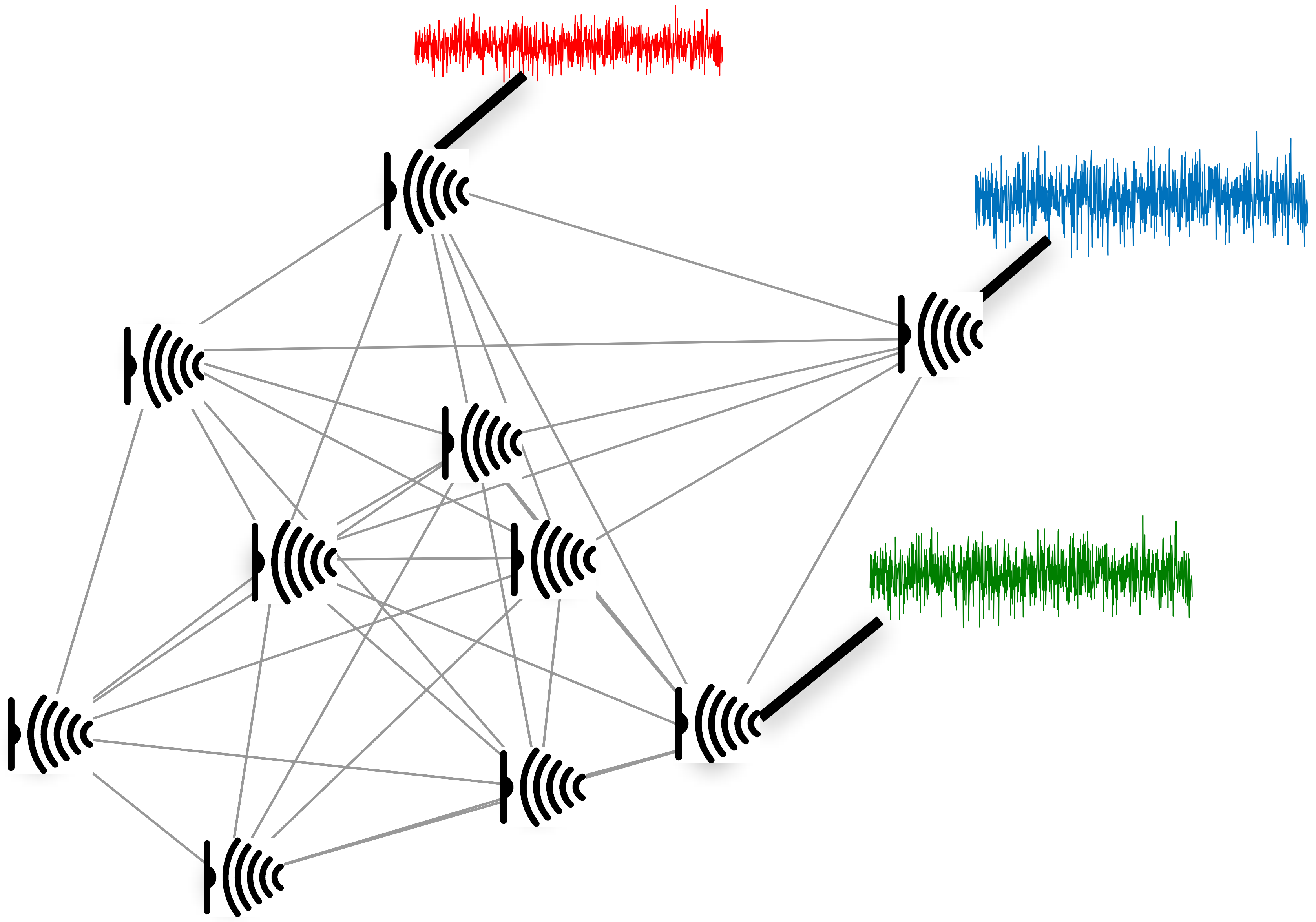
- ◉ Introduction to graph signal processing
  - Motivation
  - Mathematical formulation
  - Graph Fourier transform
  - Time-domain as a graph
- ◉ Graph filters
  - Definition and motivating applications
  - Design and implementation
  - FIR graph filters
  - ARMA graph filters

part 1 :: overview

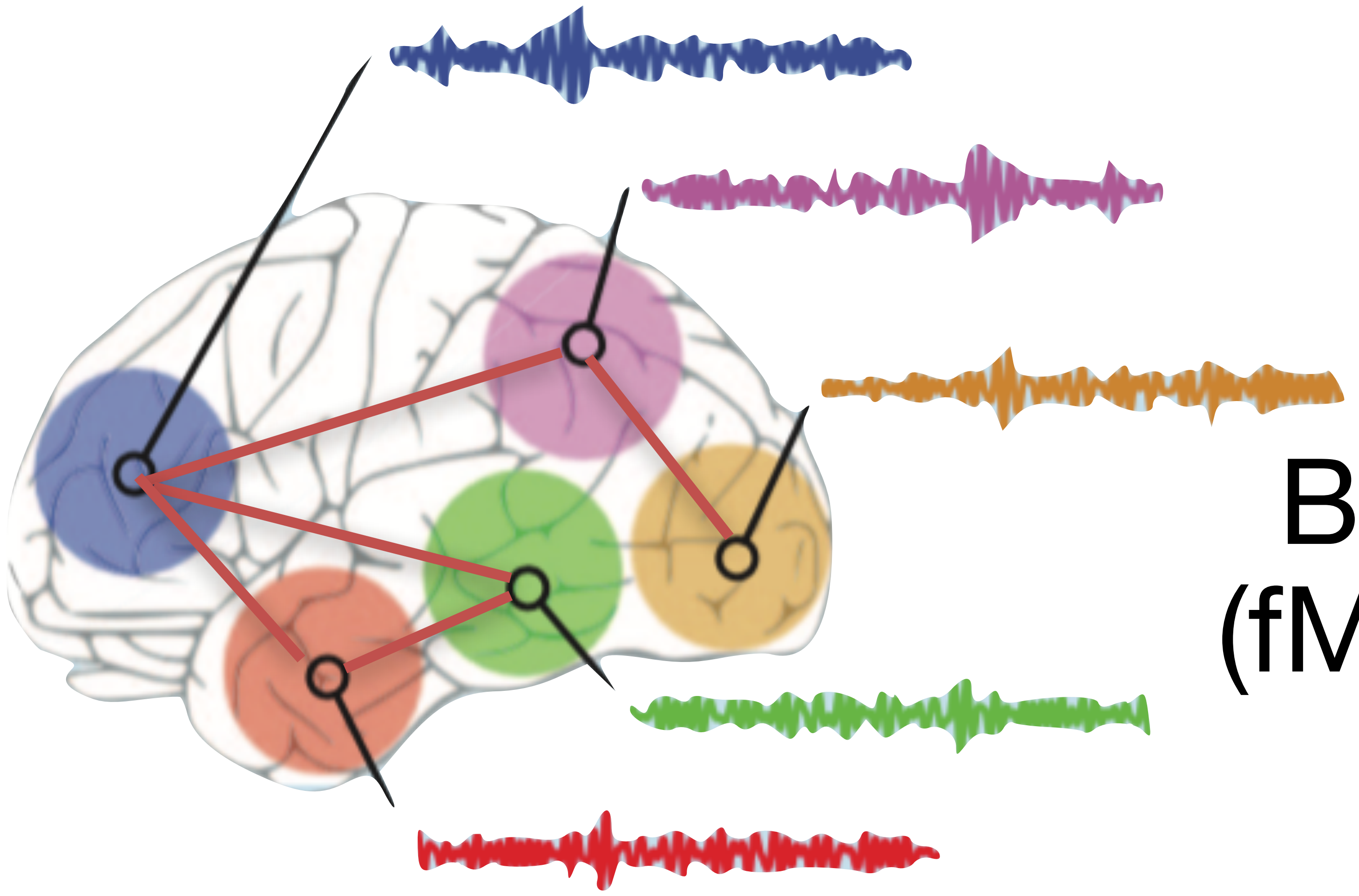
- ◉ Introduction to graph signal processing
  - Motivation
  - Mathematical formulation
  - Graph Fourier transform
  - Time-domain as a graph
- ◉ Graph filters
  - Definition and motivating applications
  - Design and implementation
  - FIR graph filters
  - ARMA graph filters
- ◉ Advanced graph filters (focus on FIR filters)
  - Node-varying graph filters
  - Edge-varying graph filters

Signals on graphs?

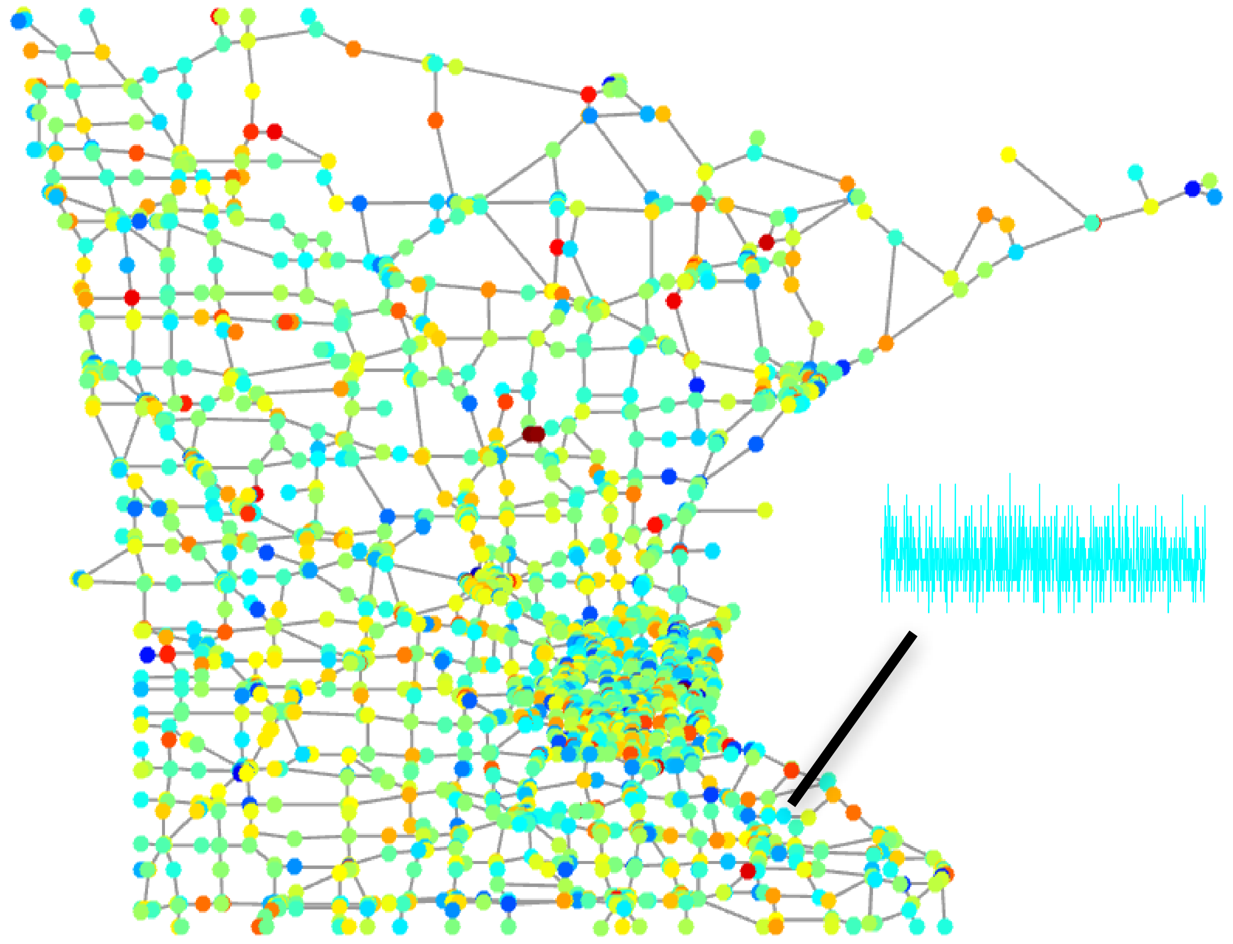
Motivation



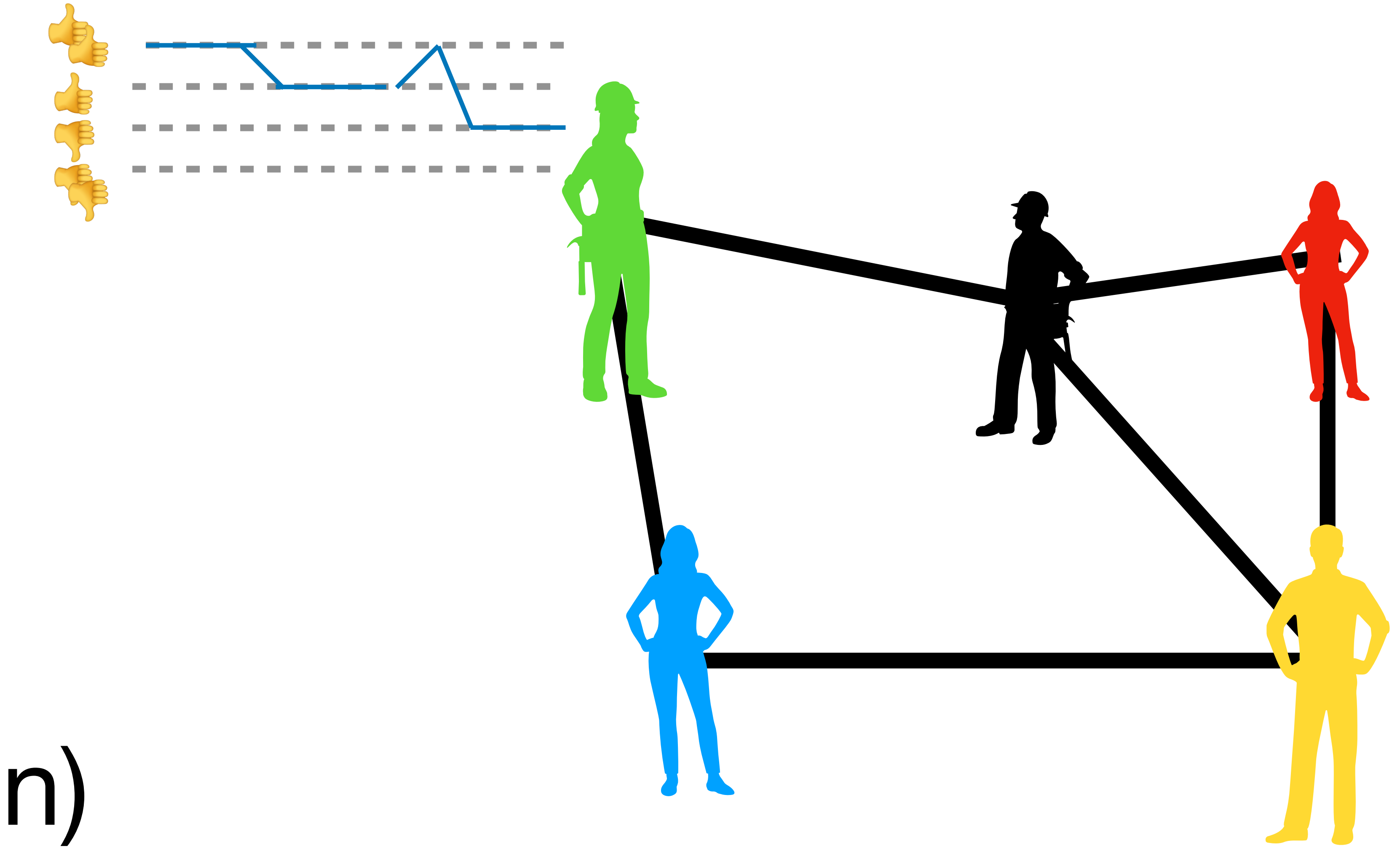
Sensor networks
(temperatures)



Brain networks
(fMRI time series)



Transport Networks
(# vehicles crossing the junction)

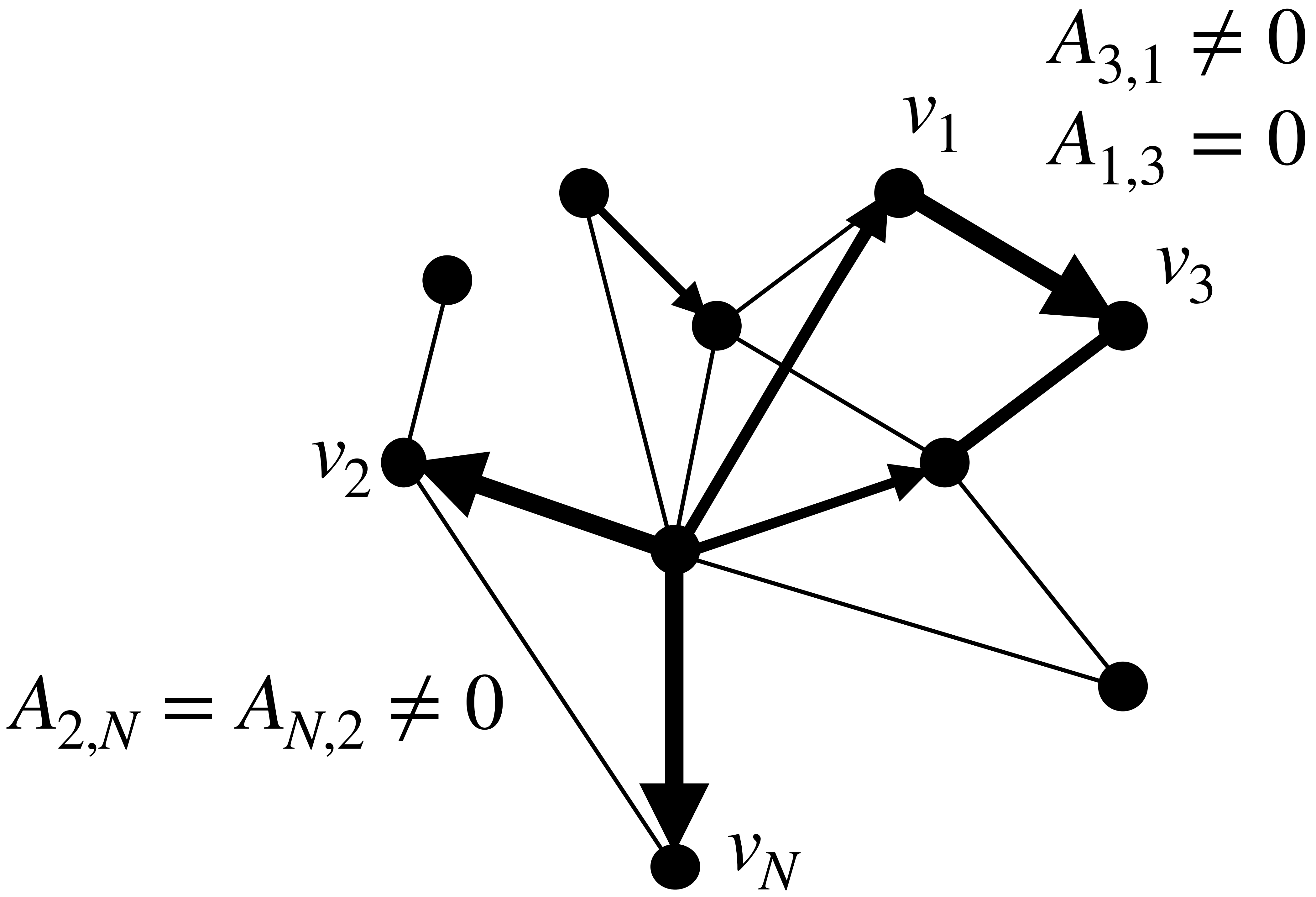


Social networks
(opinion profile)

Signal processing on graphs

Datasets with irregular support can be represented using a graph

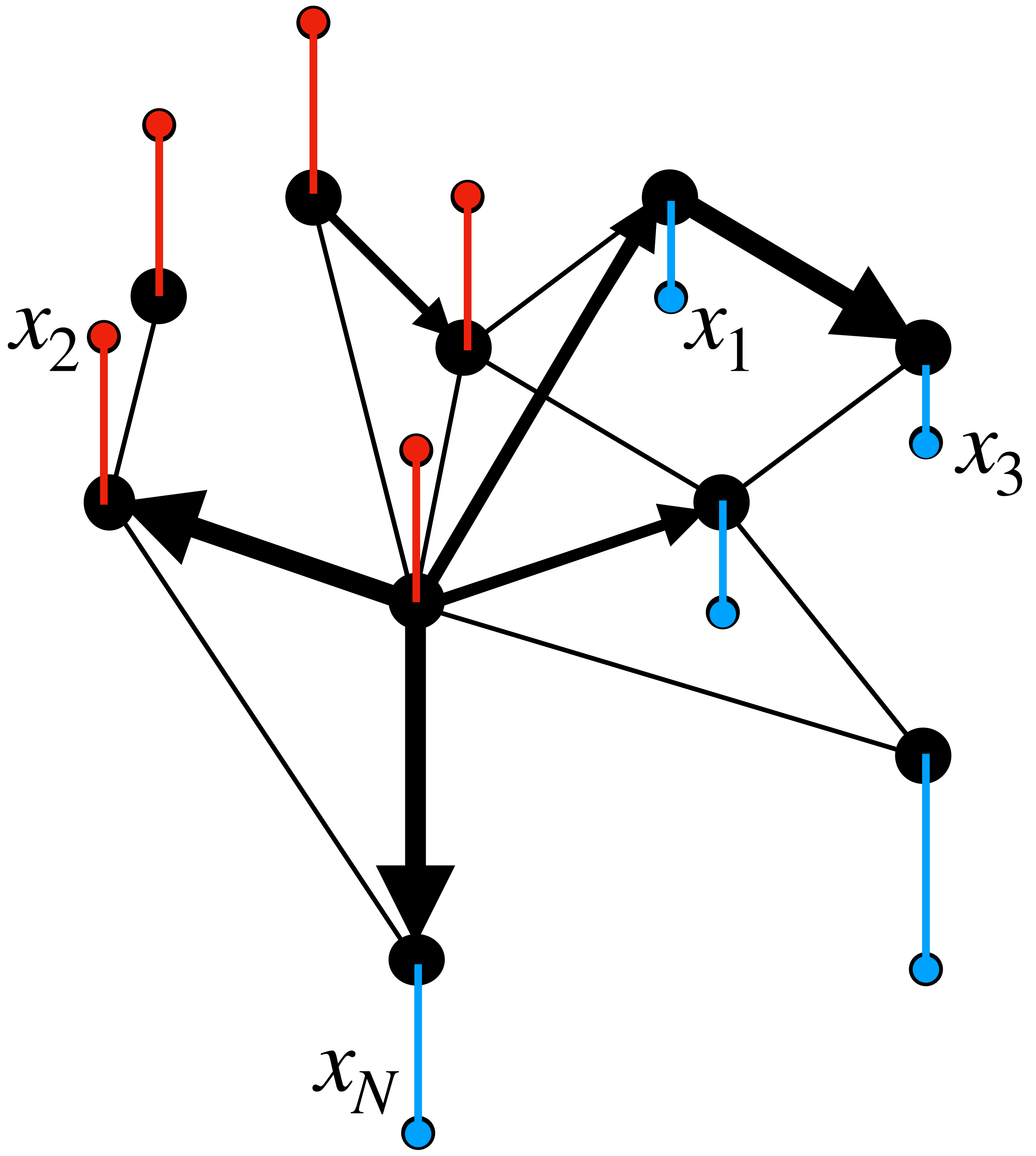
- Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- $\mathcal{V}$ is set of nodes $|\mathcal{V}| = N$
- $\mathcal{E}$ is the set of edges $|\mathcal{E}| = M$
- $\mathbf{A} \in \mathbb{R}_{+}^{N \times N}$ is the adjacency matrix



Signal processing on graphs

Datasets with irregular support can be represented using a graph

- Graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$
- $\mathscr{V}$ is set of nodes $|\mathscr{V}| = N$
- $\mathscr{E}$ is the set of edges $|\mathscr{E}| = M$
- $\mathbf{A} \in \mathbb{R}_{+}^{N \times N}$ is the adjacency matrix
- $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^N$ is the graph signal



Signal processing on graphs

- Local structure of the graph is captured by the graph-shift operator $\mathbf{S} \in \mathbb{R}^{N \times N}$

  $[\mathbf{S}]_{j,i}$ is nonzero only if $(i, j) \in \mathscr{E}$ and/or $i = j$.

  $\mathbf{S}$ could be the adjacency matrix, graph Laplacian, or modifications, …

Signal processing on graphs

- Local structure of the graph is captured by the graph-shift operator $\mathbf{S} \in \mathbb{R}^{N \times N}$

  $[\mathbf{S}]_{j,i}$ is nonzero only if $(i, j) \in \mathscr{E}$ and/or $i = j$.

  $\mathbf{S}$ could be the adjacency matrix, graph Laplacian, or modifications, …
- Adjacency matrix: $\mathbf{S} = \mathbf{A}$

Signal processing on graphs

- Local structure of the graph is captured by the graph-shift operator $\mathbf{S} \in \mathbb{R}^{N \times N}$

  $[\mathbf{S}]_{j,i}$ is nonzero only if $(i, j) \in \mathscr{E}$ and/or $i = j$.

  $\mathbf{S}$ could be the adjacency matrix, graph Laplacian, or modifications, …

- Adjacency matrix: $\mathbf{S} = \mathbf{A}$

- Graph Laplacian: $\mathbf{S} = \mathbf{L}_{\text{in/out}} = \mathbf{D}_{\text{in/out}} - \mathbf{A}$

$$[\mathbf{D}_{\text{in}}]_{i,i} = \sum_{j=1}^{N} [\mathbf{A}_{i,j}] \qquad [\mathbf{D}_{\text{out}}]_{i,i} = \sum_{j=1}^{N} [\mathbf{A}_{j,i}]$$

Signal processing on graphs

- Local structure of the graph is captured by the graph-shift operator $\mathbf{S} \in \mathbb{R}^{N\times N}$

  $[\mathbf{S}]_{j,i}$ is nonzero only if $(i,j) \in \mathscr{E}$ and/or $i = j$.

  $\mathbf{S}$ could be the adjacency matrix, graph Laplacian, or modifications, …
- Adjacency matrix: $\mathbf{S} = \mathbf{A}$
- Graph Laplacian: $\mathbf{S} = \mathbf{L}_{\text{in/out}} = \mathbf{D}_{\text{in/out}} - \mathbf{A}$

$$[\mathbf{D}_{\text{in}}]_{i,i} = \sum_{j=1}^{N} [\mathbf{A}_{i,j}] \qquad [\mathbf{D}_{\text{out}}]_{i,i} = \sum_{j=1}^{N} [\mathbf{A}_{j,i}]$$

- Symmetric graph Laplacian: $\mathbf{S} = \mathbf{L} = \mathbf{D} - \mathbf{A}, \quad \mathbf{D} = \mathbf{D}_{\text{in}} = \mathbf{D}_{\text{out}}$

Signal processing on graphs

- Local structure of the graph is captured by the graph-shift operator $\mathbf{S} \in \mathbb{R}^{N \times N}$

  $[\mathbf{S}]_{j,i}$ is nonzero only if $(i, j) \in \mathscr{E}$ and/or $i = j$.

  $\mathbf{S}$ could be the adjacency matrix, graph Laplacian, or modifications, …
- Adjacency matrix: $\mathbf{S} = \mathbf{A}$
- Graph Laplacian: $\mathbf{S} = \mathbf{L}_{\text{in/out}} = \mathbf{D}_{\text{in/out}} - \mathbf{A}$

$$[\mathbf{D}_{\text{in}}]_{i,i} = \sum_{j=1}^{N} [\mathbf{A}_{i,j}] \qquad [\mathbf{D}_{\text{out}}]_{i,i} = \sum_{j=1}^{N} [\mathbf{A}_{j,i}]$$

- Symmetric graph Laplacian: $\mathbf{S} = \mathbf{L} = \mathbf{D} - \mathbf{A}, \quad \mathbf{D} = \mathbf{D}_{\text{in}} = \mathbf{D}_{\text{out}}$
- Smoothness: $\mathbf{x}^\top \mathbf{L} \mathbf{x} = \sum_{i,j=1}^{N} [\mathbf{A}]_{i,j} (x_i - x_j)^2$

Spectral analysis of graph signals

Graph Fourier basis

Eigenvectors of graph shift represent frequency modes ($\mathbf{S}$ assumed to be normal)

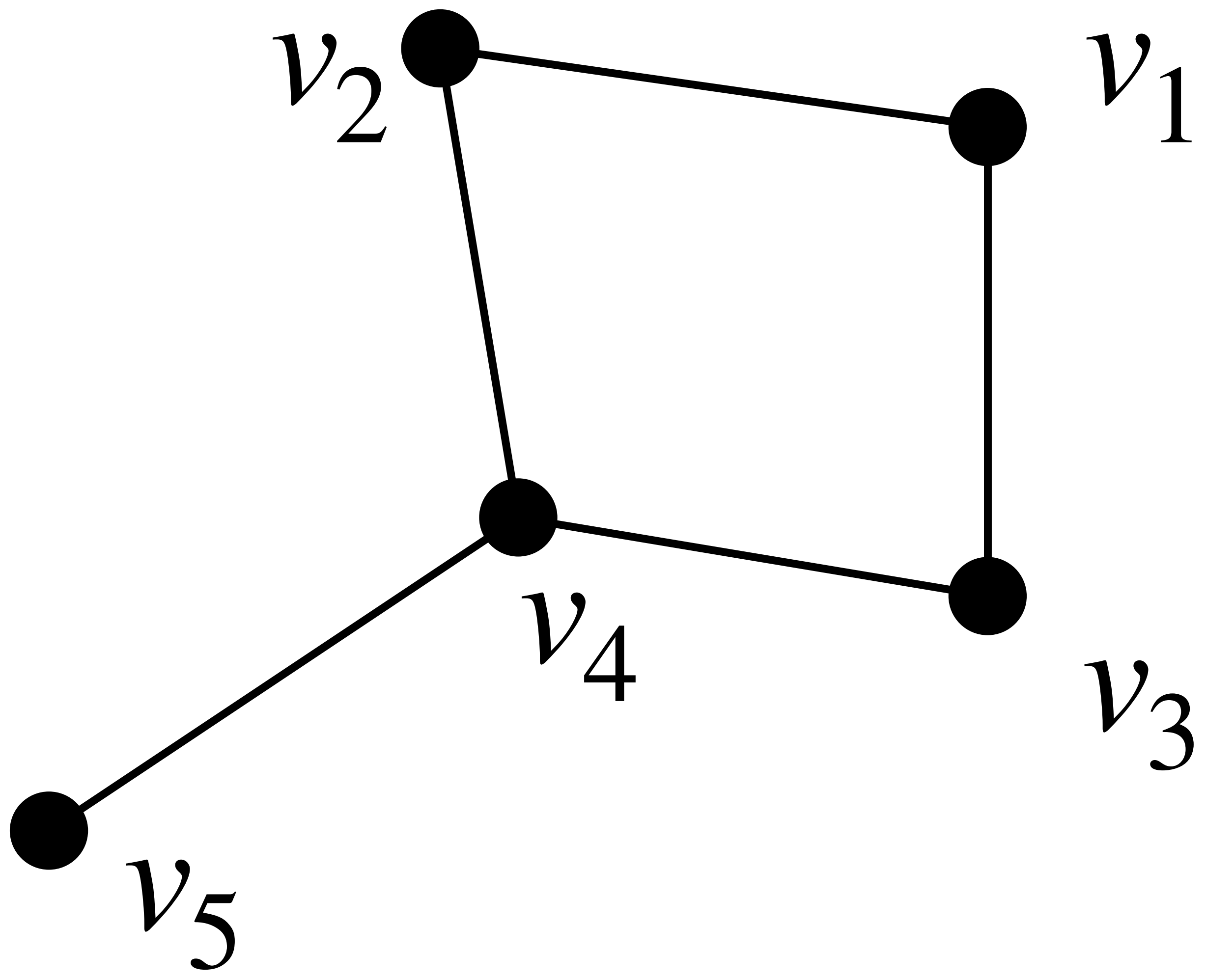
$$\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathrm{H}}$$

Graph Fourier basis

Eigenvectors of graph shift represent frequency modes ($\mathbf{S}$ assumed to be normal)

$$\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathrm{H}}$$

Example: Laplacian of undirected graph

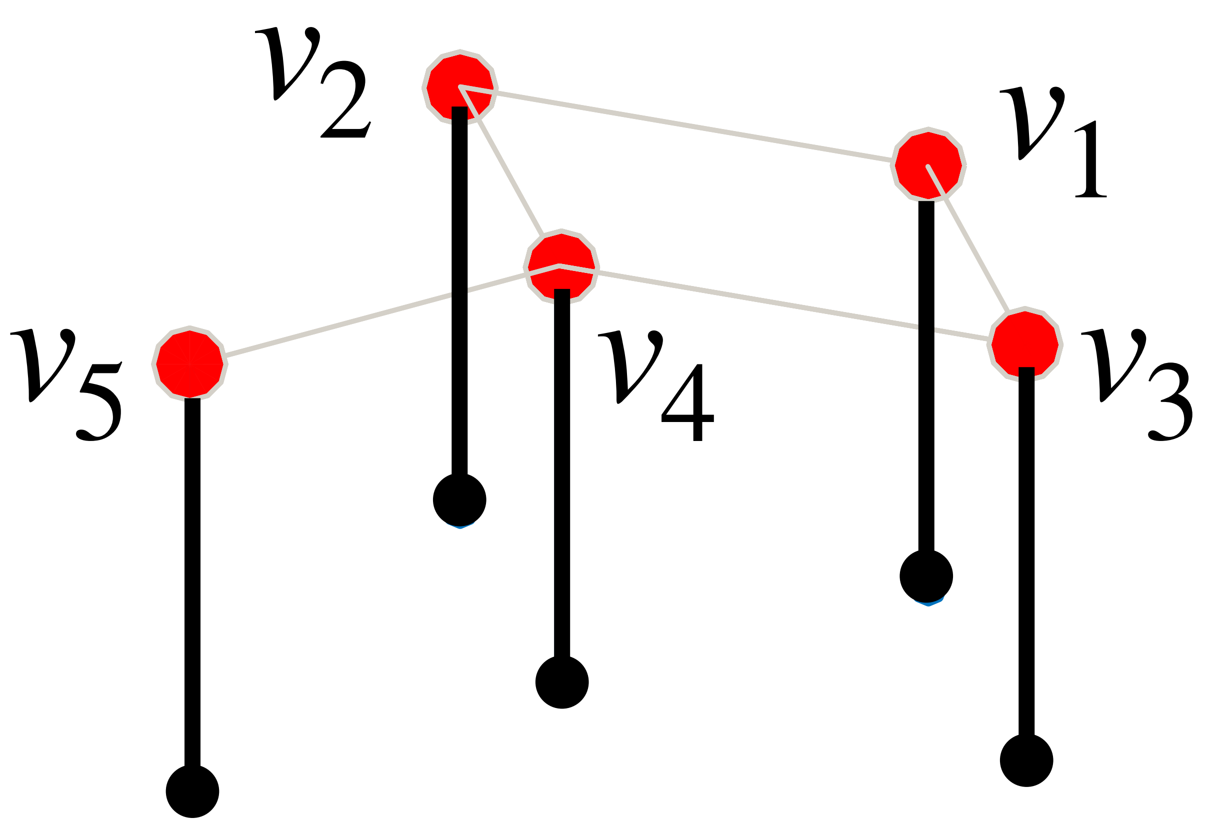


$$\mathbf{S} = \mathbf{D} - \mathbf{A} = \underbrace{\begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{diagonal degree matrix}} - \underbrace{\begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \end{array}}_{\text{adjacency matrix}}$$

Graph Fourier basis

Individual eigenvectors of Laplacian of undirected graph

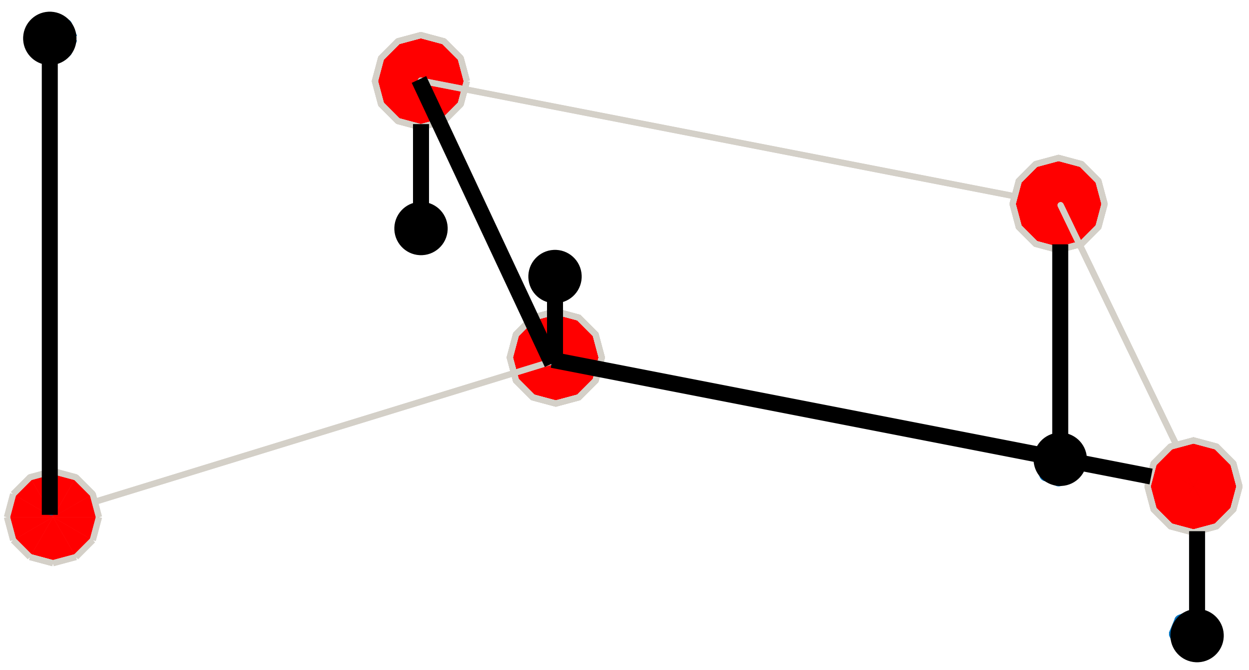
$\mathbf{u}_1$



DC (no zero crossings)

$$\mathbf{u}_1^\top \mathbf{S} \mathbf{u}_1 = \lambda_1 = 0$$

$\mathbf{u}_2$

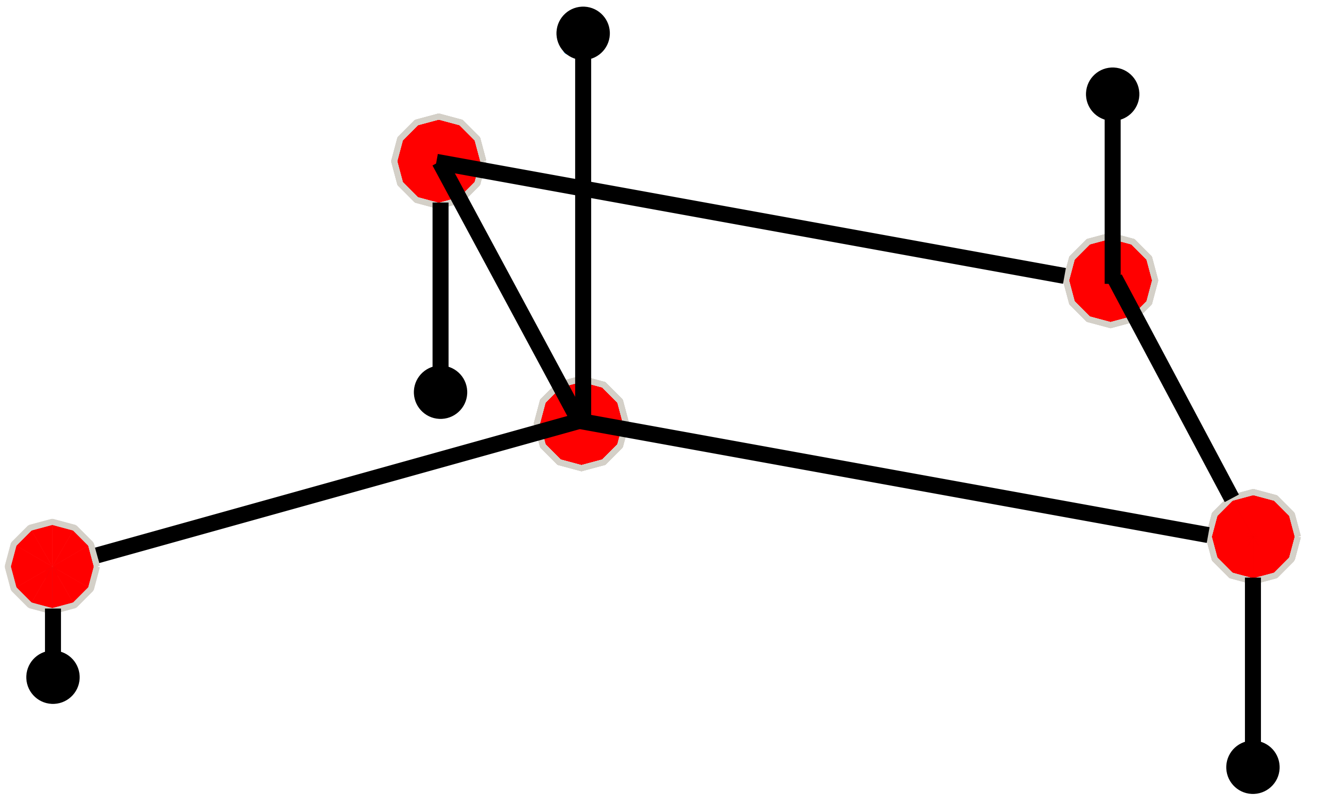


two zero crossings

$$\mathbf{u}_2^\top \mathbf{S} \mathbf{u}_2 = \lambda_2 = 0.8299$$

$\cdots$

$\mathbf{u}_5$



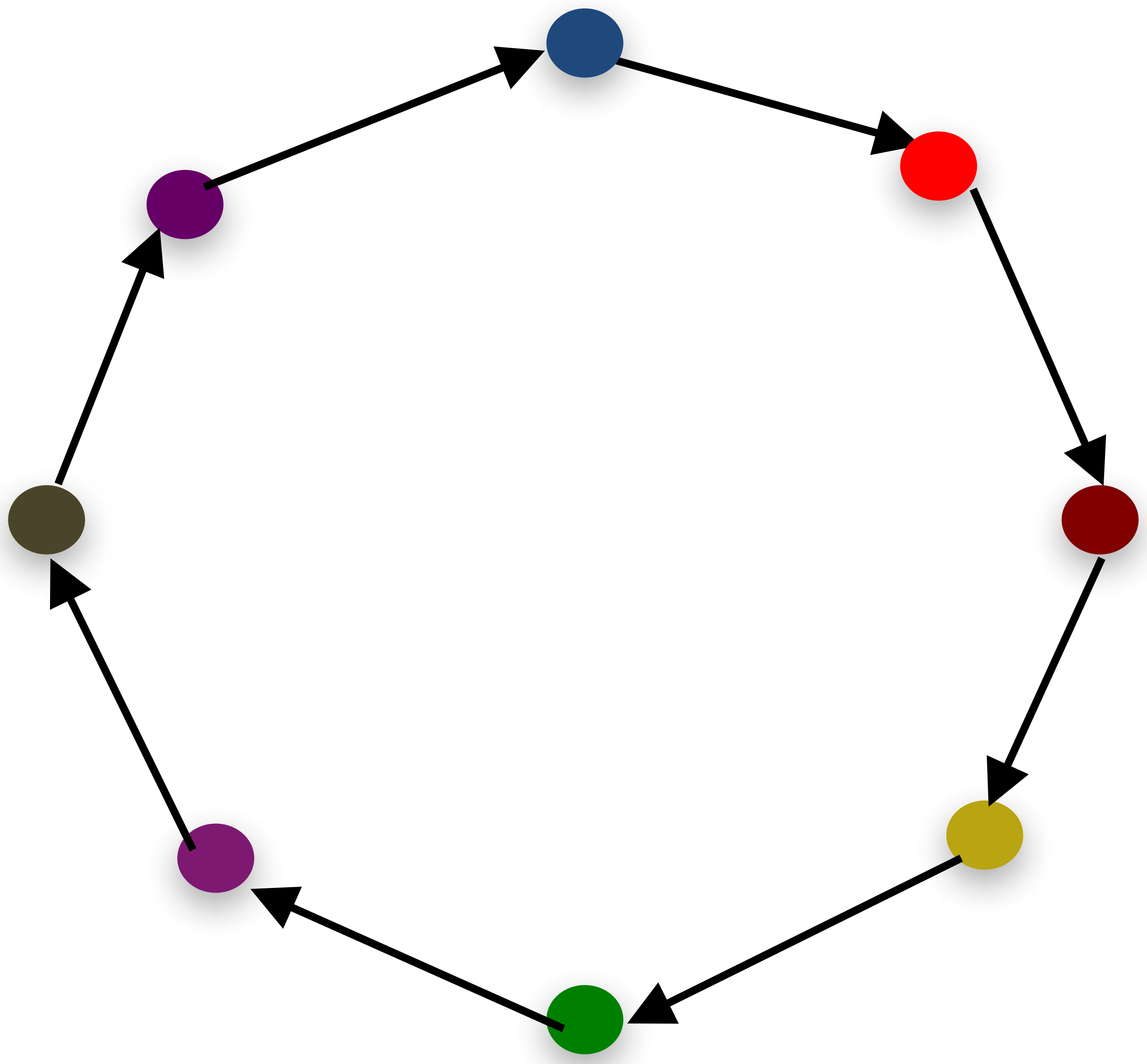
five zero crossings

$$\mathbf{u}_3^\top \mathbf{S} \mathbf{u}_3 = \lambda_5 = 4.4812$$

Time-domain as a graph

The DFT matrix and the traditional frequency grid is obtained by the adjacency matrix of the cycle graph

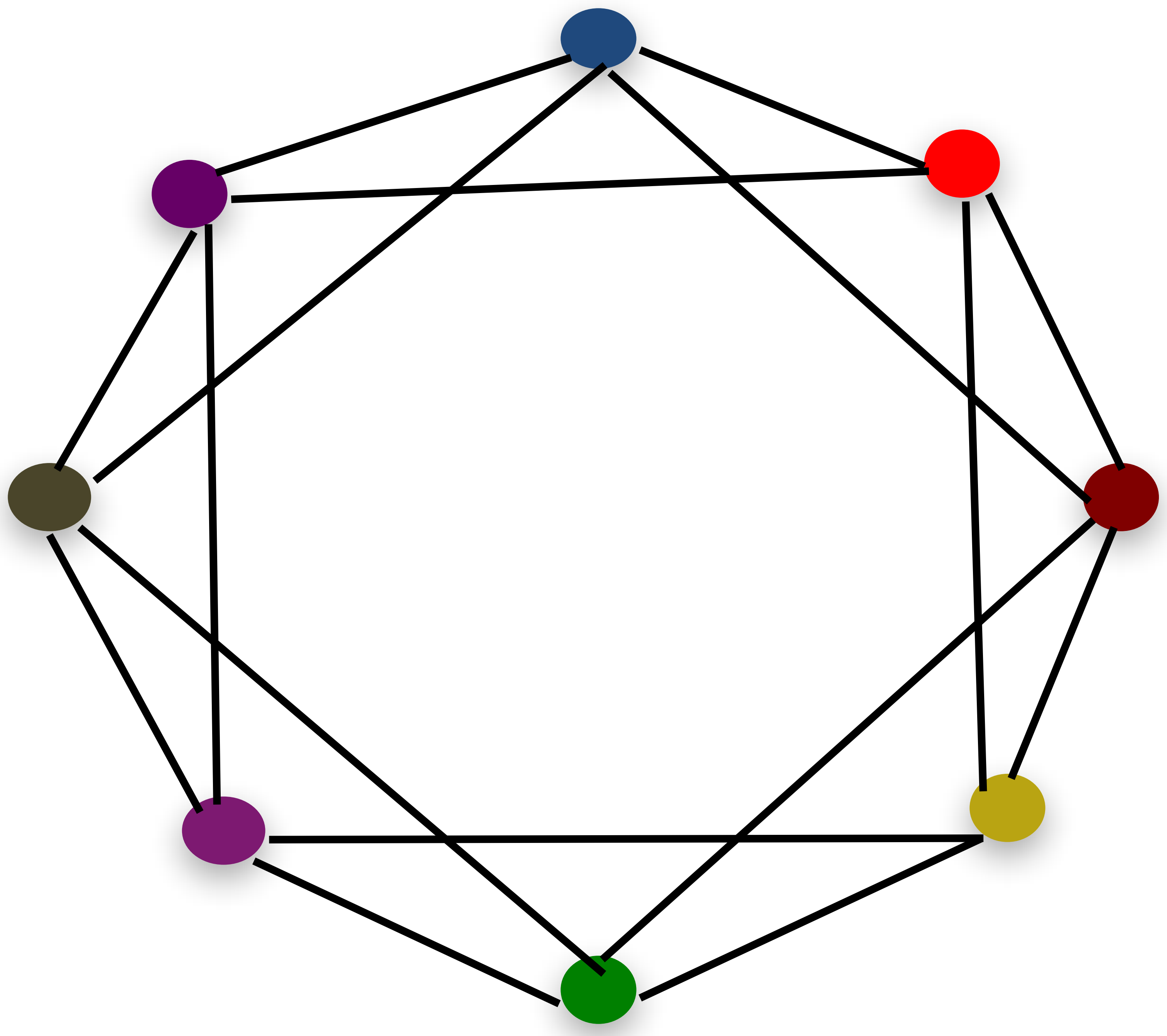
$$\mathbf{S} = \mathbf{F}^{-1}\mathbf{\Omega}\mathbf{F} : \ [\mathbf{\Omega}]_{i,i} = e^{2j\pi(i-1)/N}$$



$$\mathbf{S} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Time-domain as a graph

Any circulant graph (directed or not) in principle leads to the DFT as the matrix that diagonalises the shift operator



$$\mathbf{S} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

How do we “spectrally” shape signals?

Graph Fourier transform and graph filters

The graph Fourier transform is defined as

$$\hat{\mathbf{x}} = \mathbf{U}^{\mathrm{H}}\mathbf{x} \iff \mathbf{x} = \mathbf{U}\hat{\mathbf{x}}$$

Graph Fourier transform and graph filters

The graph Fourier transform is defined as

$$\hat{\mathbf{x}} = \mathbf{U}^{\mathrm{H}}\mathbf{x} \iff \mathbf{x} = \mathbf{U}\hat{\mathbf{x}}$$

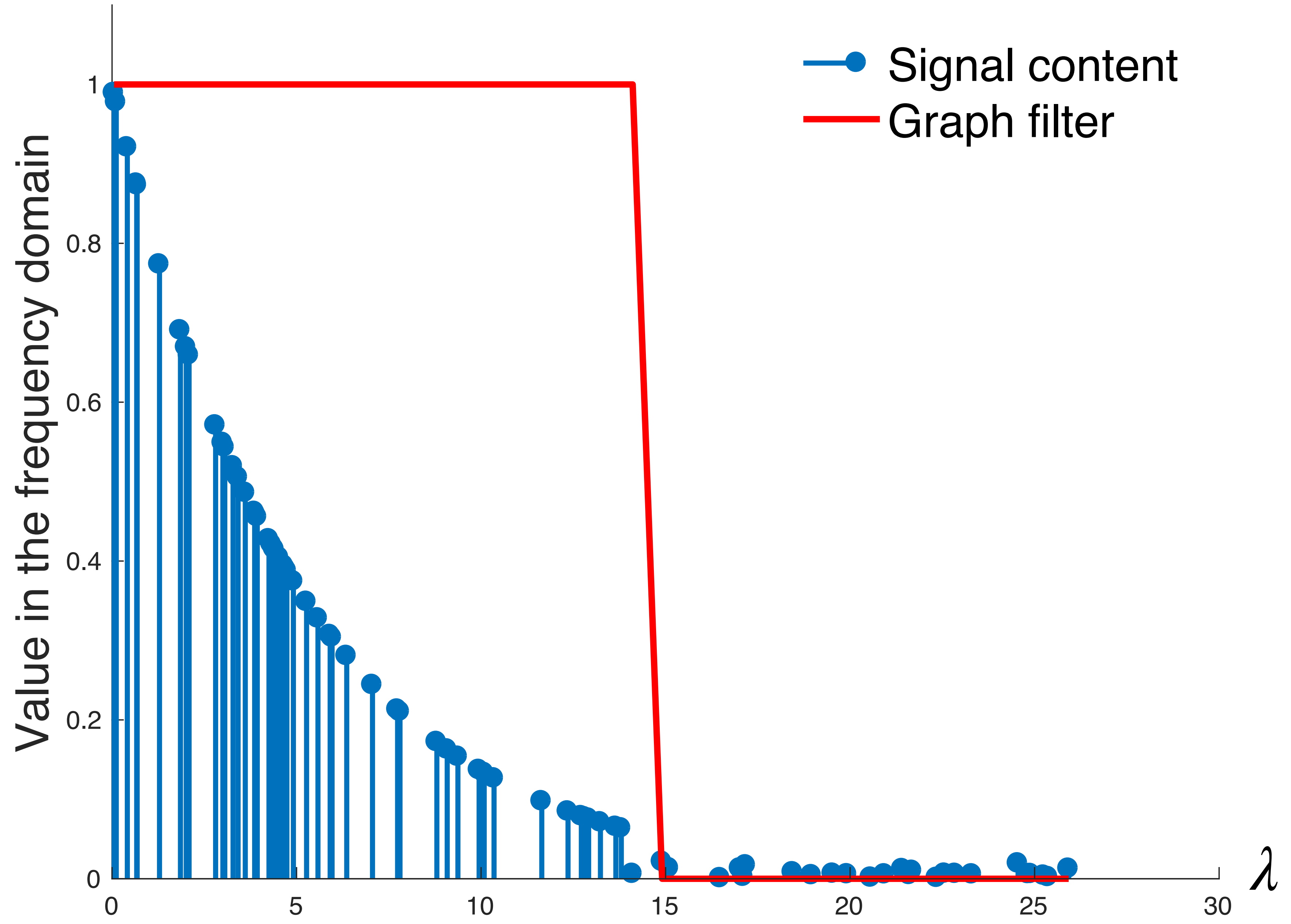
Graph filters can be used to modify the frequency content of graph signals

$$\hat{y}_n = h(\lambda_n)\hat{x}_n$$

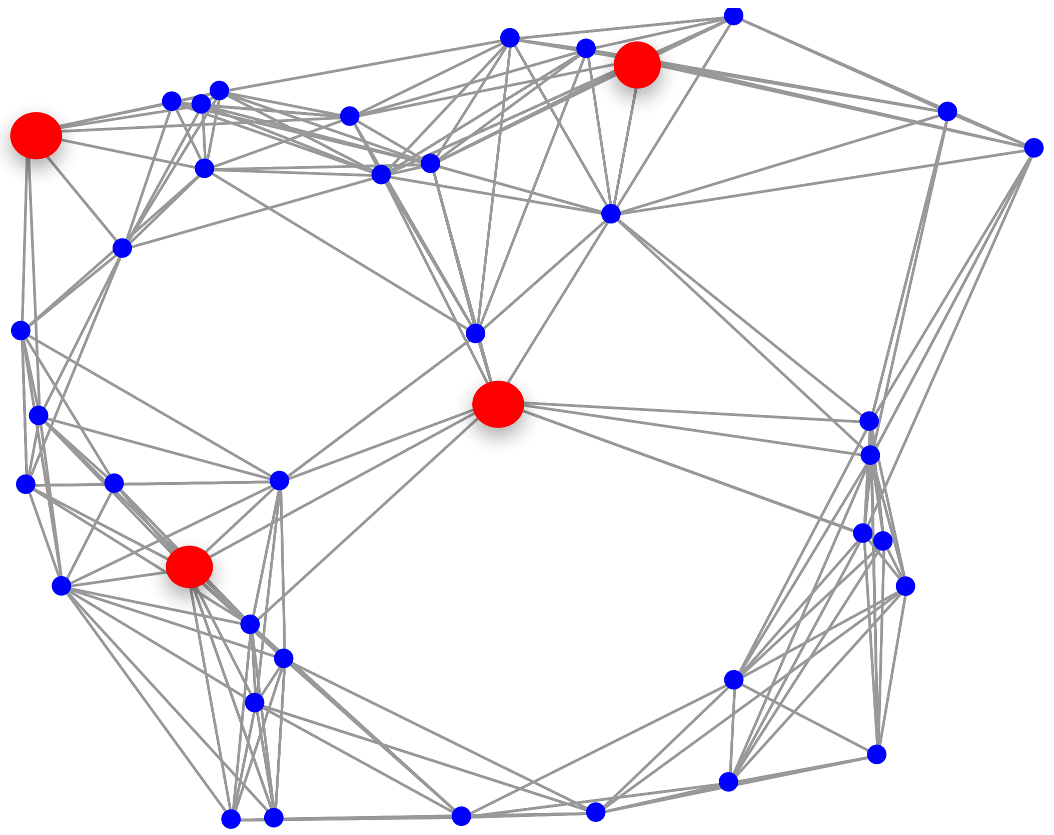
→ $$\begin{cases} \hat{\mathbf{y}} = h(\mathbf{\Lambda})\hat{\mathbf{x}} \\ h(\mathbf{\Lambda}) = \mathrm{diag}\{h(\lambda_n)\} \end{cases}$$

→ $$\mathbf{y} = \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^{\mathrm{H}}\mathbf{x} = \mathbf{H}\mathbf{x}$$

→ Shift invariance: $\mathbf{HS} = \mathbf{SH}$

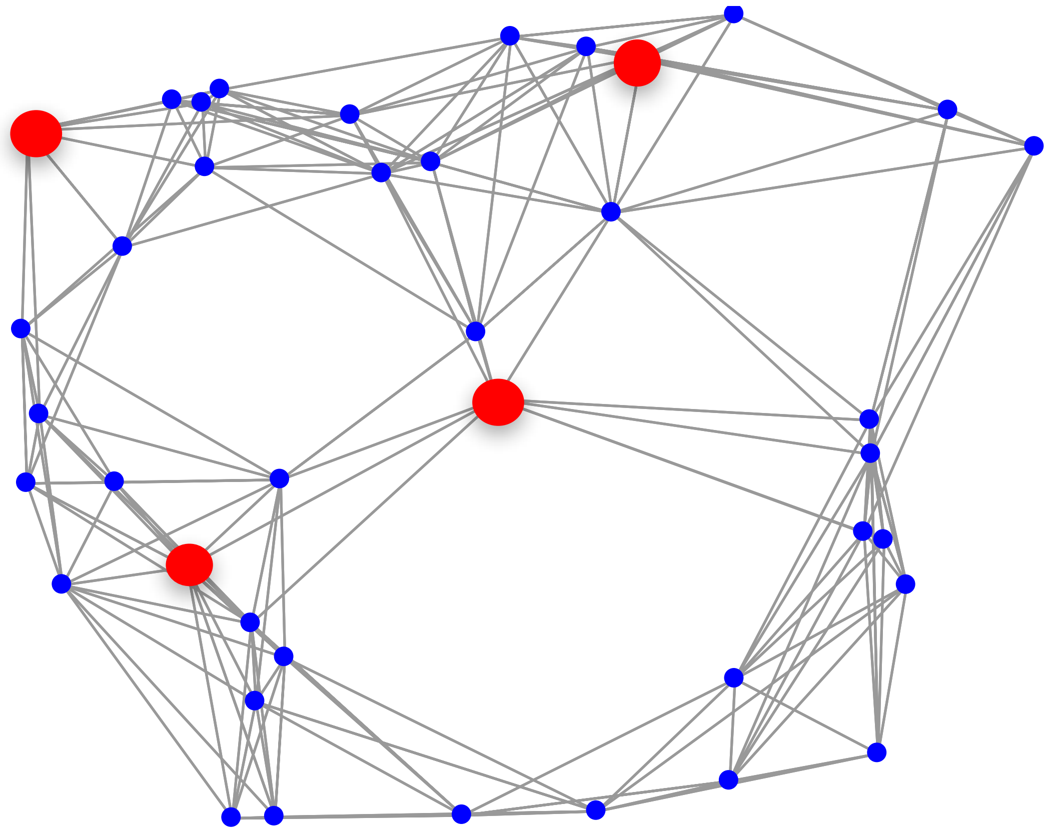


Applications of graph filters

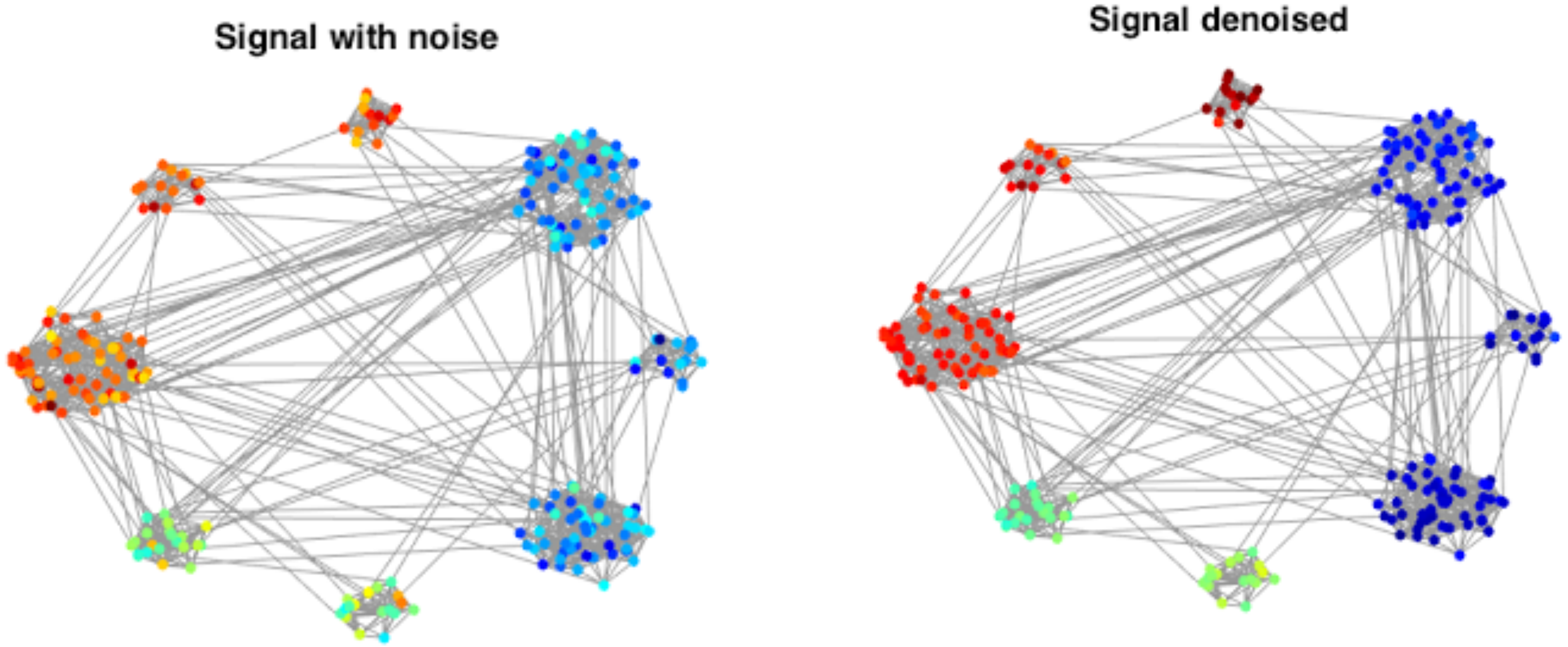


Interpolation (e.g., semi-supervised learning)

Applications of graph filters

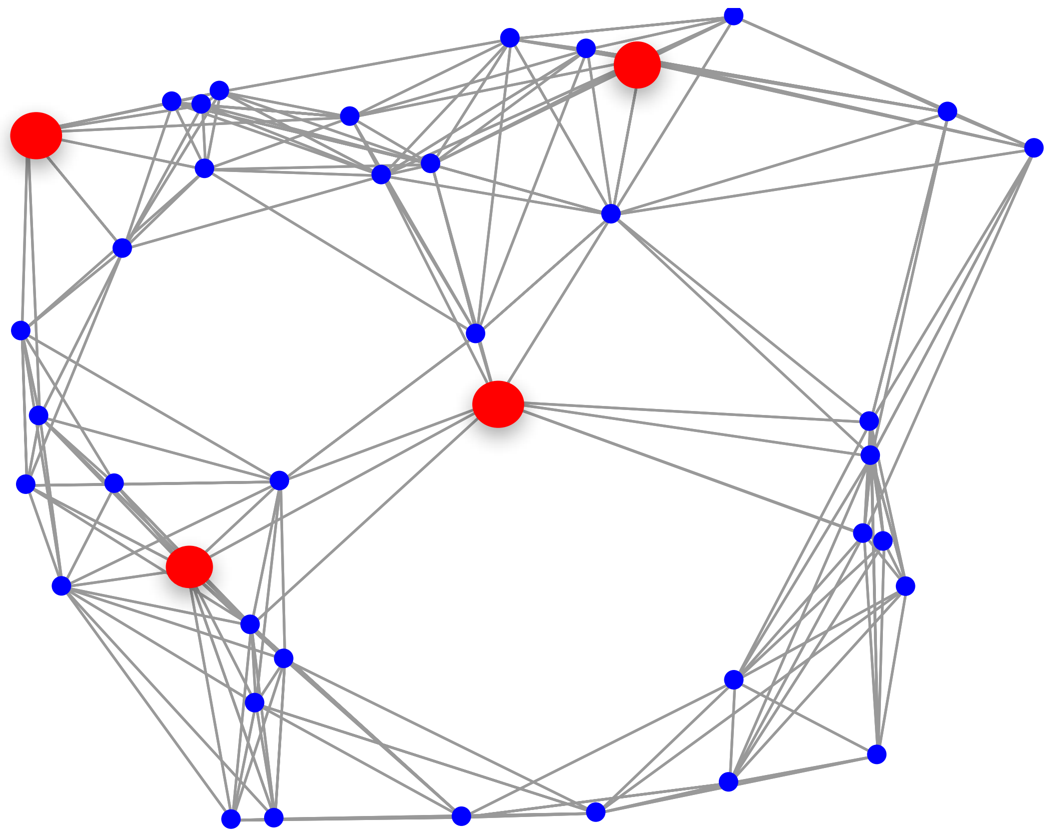


Interpolation (e.g., semi-supervised learning)

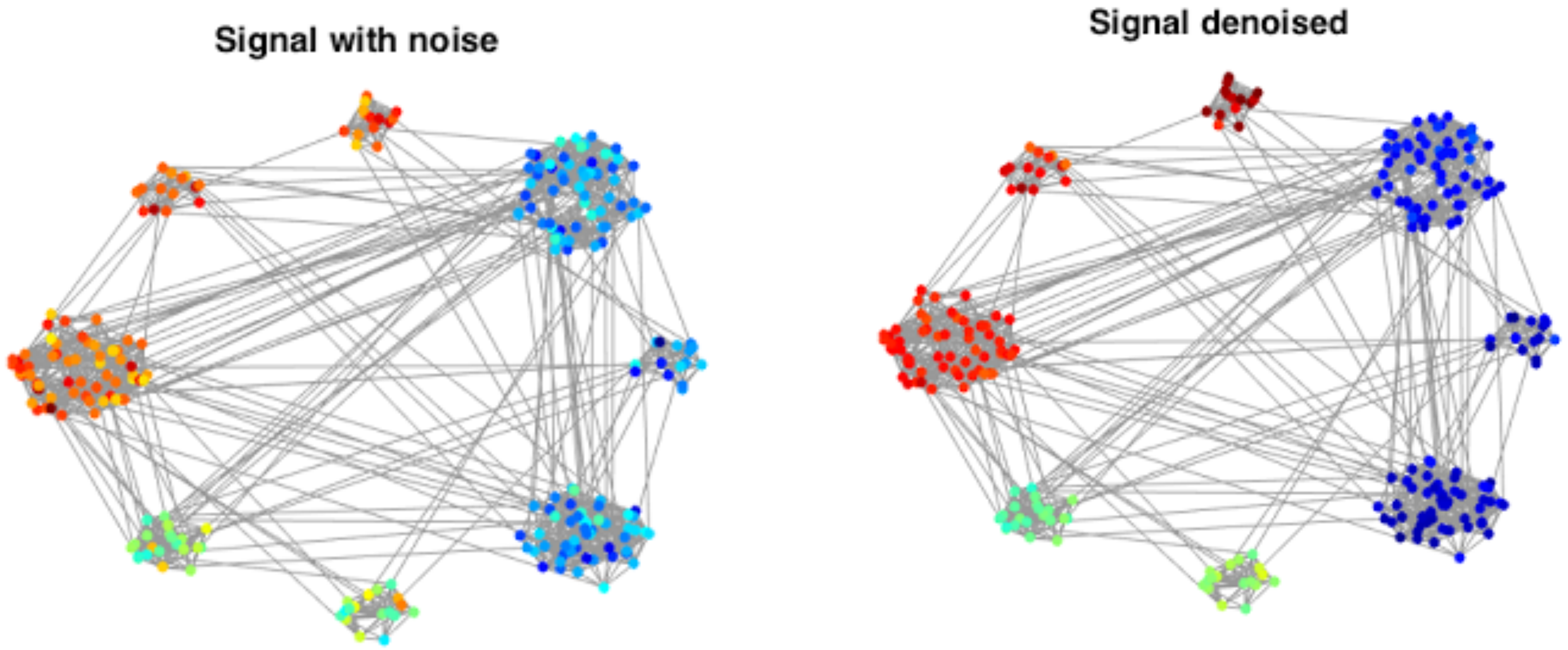


Denoising signals (e.g., Tikhonov)

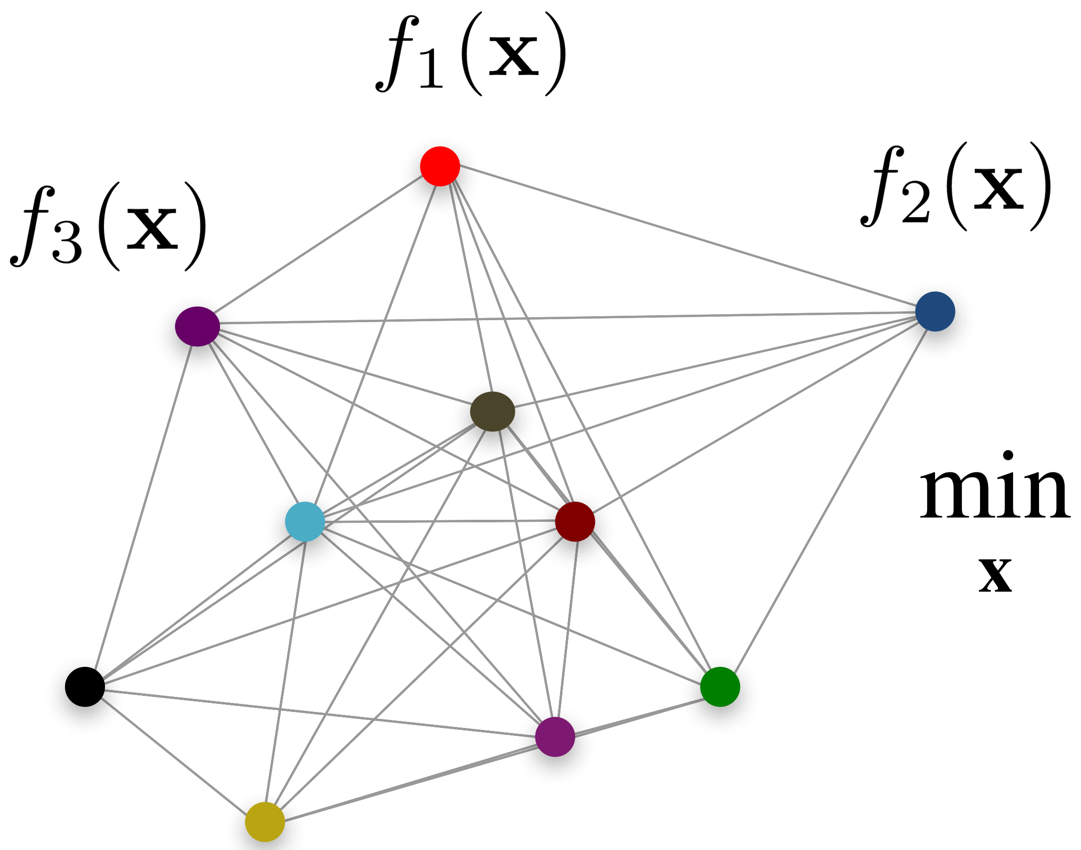
Applications of graph filters



Interpolation (e.g., semi-supervised learning)



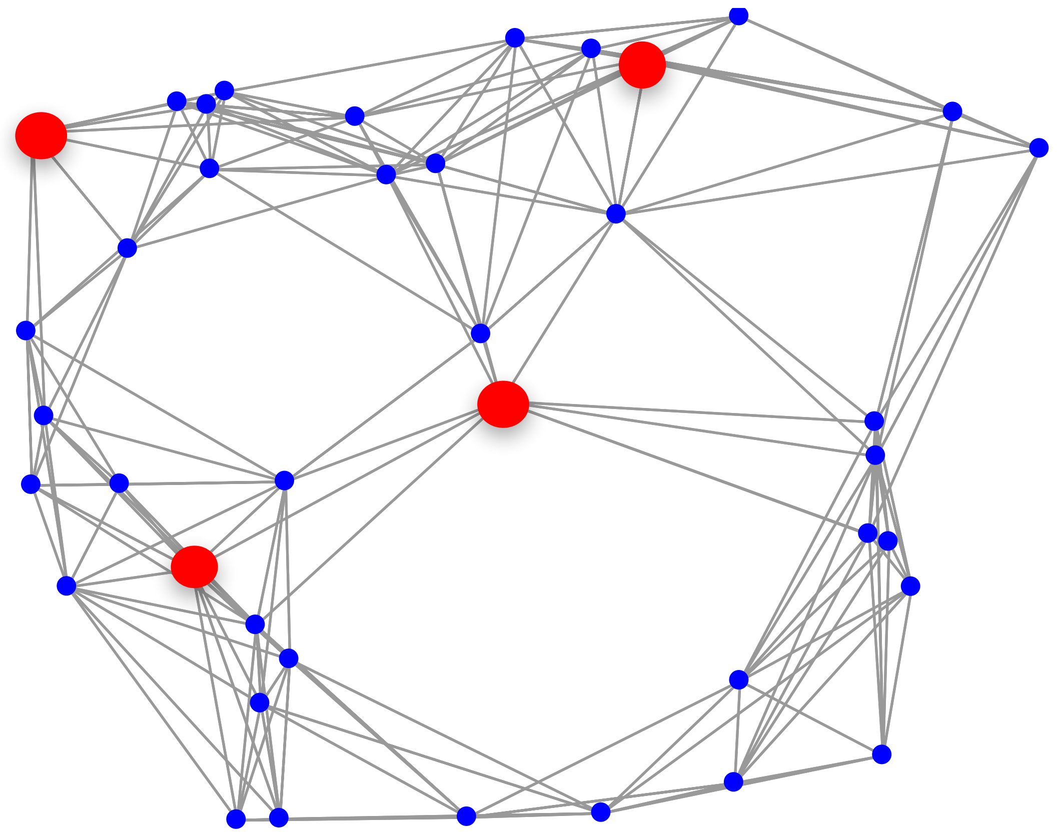
Denoising signals (e.g., Tikhonov)



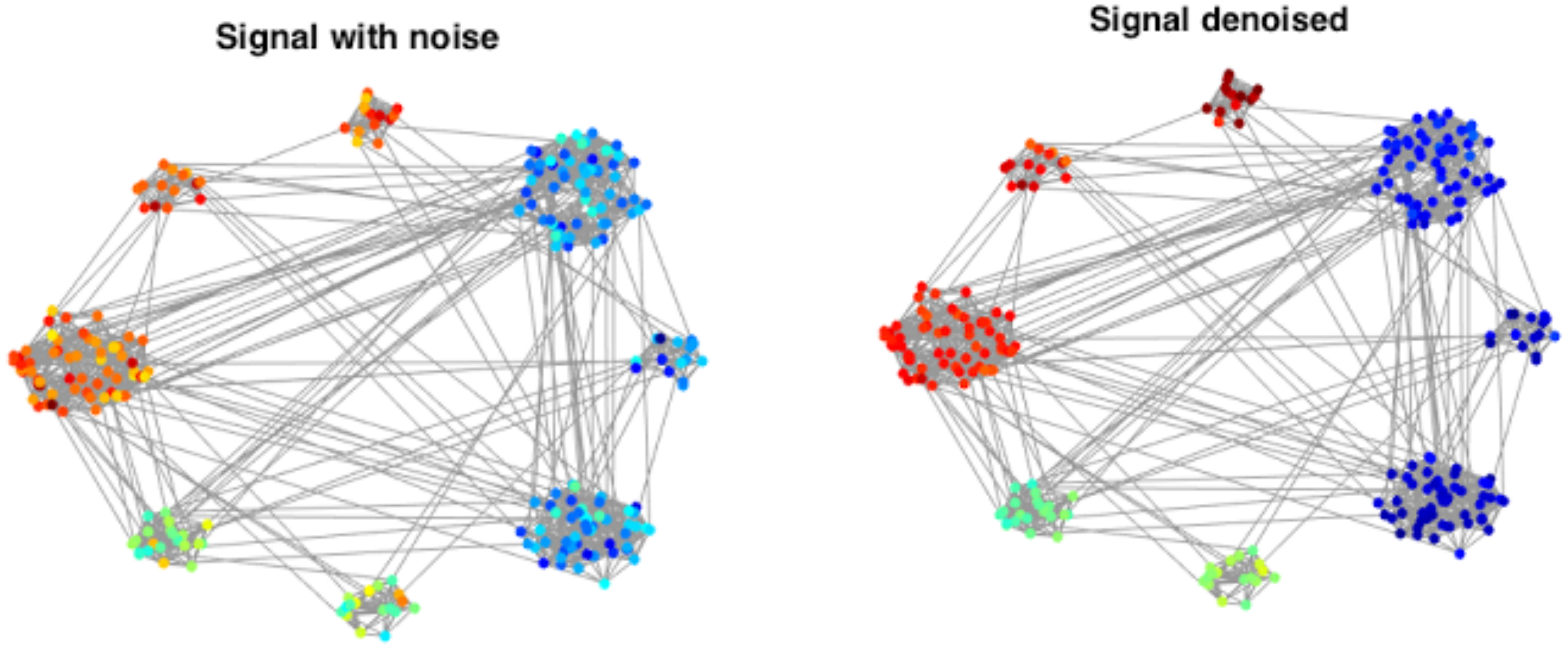
$$\min_{\mathbf{x}} \; f(x) = \sum_{i=1}^{N} f_i(\mathbf{x})$$

Distributed optimization

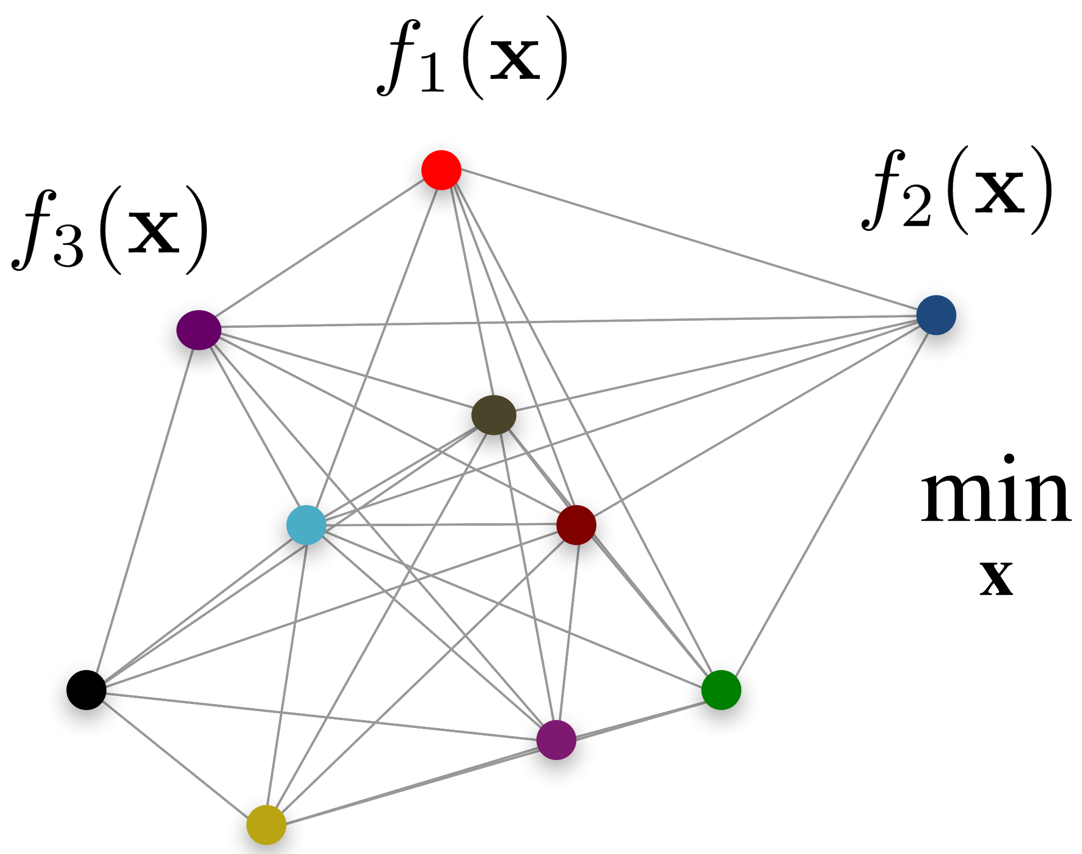
Applications of graph filters



Interpolation (e.g., semi-supervised learning)

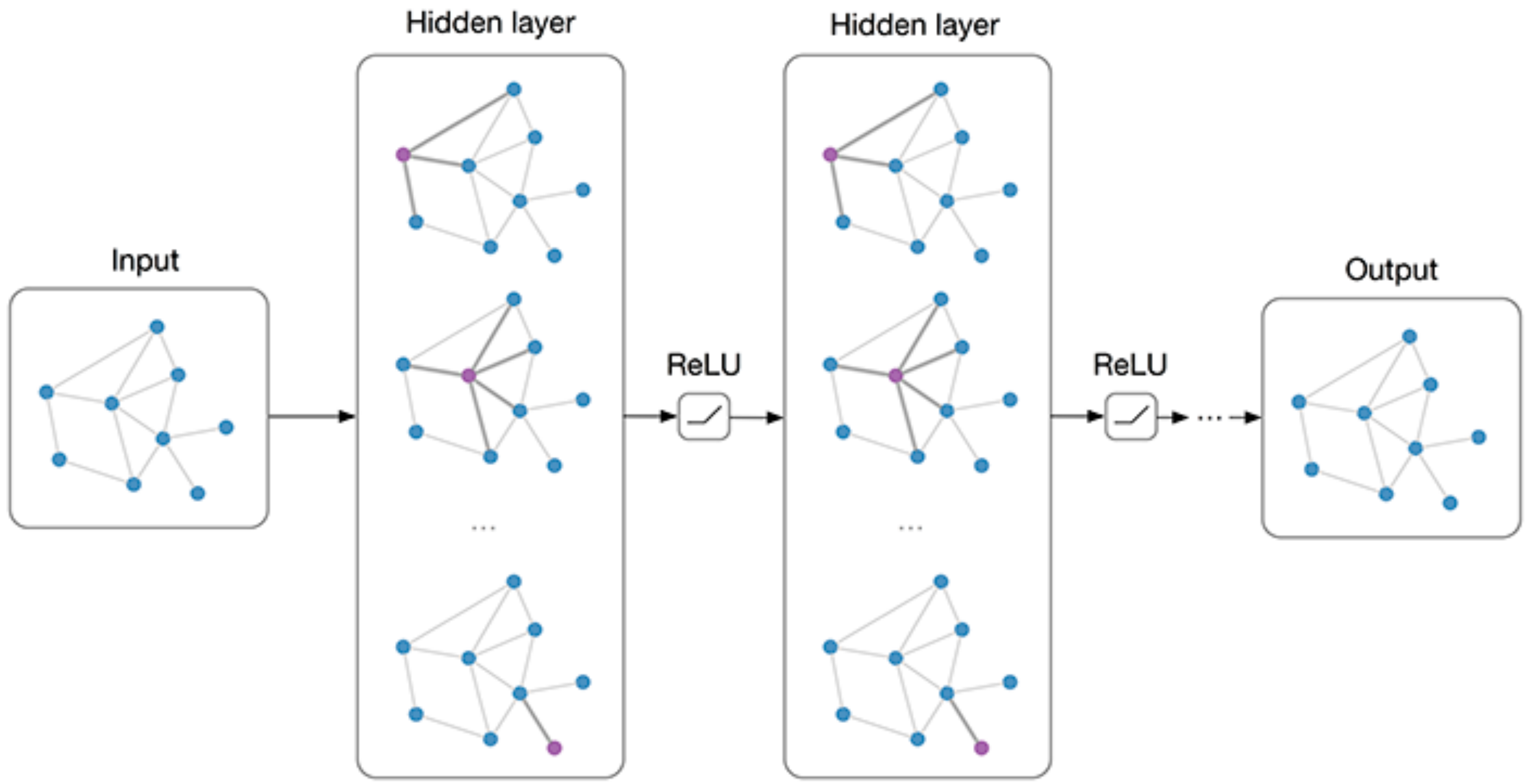


Denoising signals (e.g., Tikhonov)



$$\min_{\mathbf{x}} \; f(x) = \sum_{i=1}^{N} f_i(\mathbf{x})$$

Distributed optimization



Graph convolutional neural networks

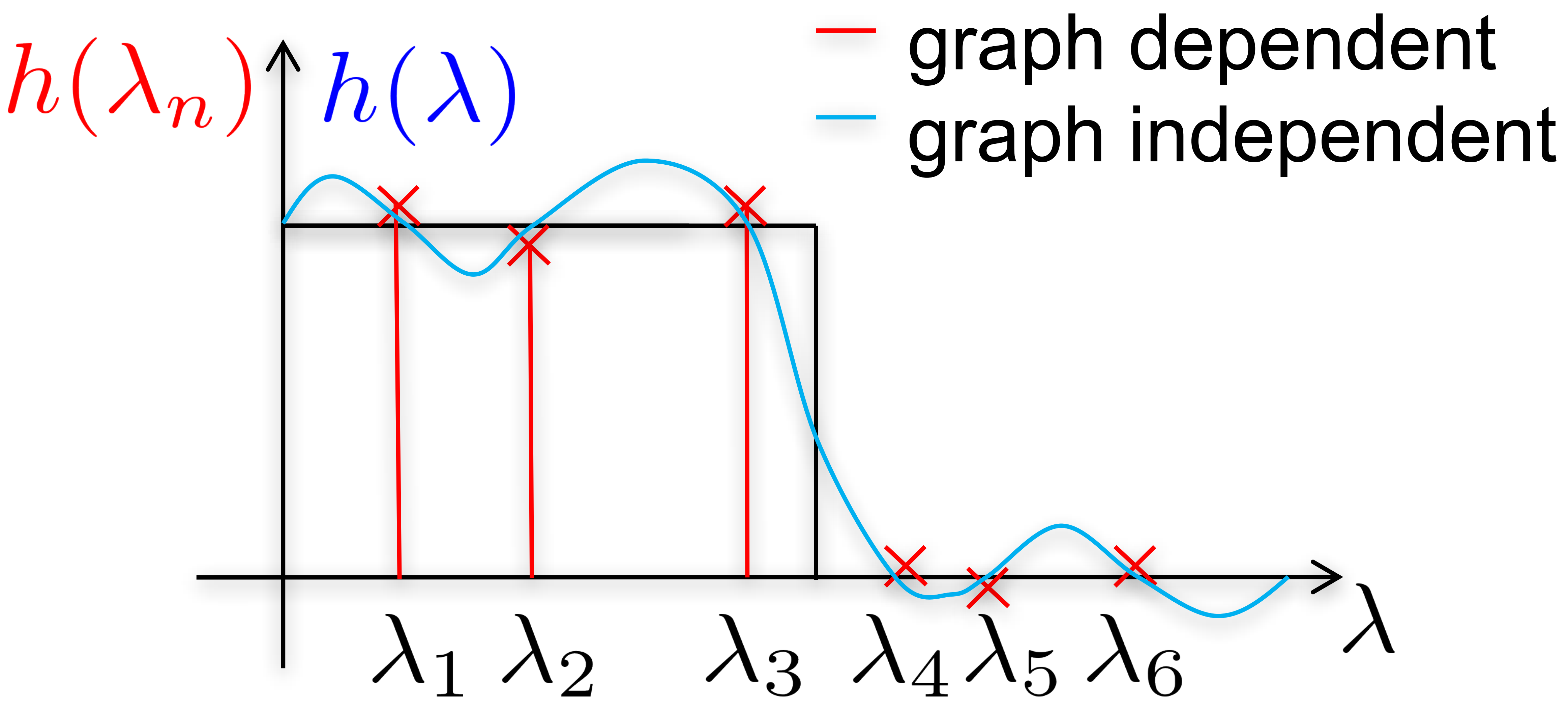
Graph filter design and implementation

$$\mathbf{y} = \mathrm{U}h(\boldsymbol{\Lambda})\mathrm{U}^{\mathrm{H}}\mathbf{x} = \mathbf{H}\mathbf{x} \iff \mathbf{H}\mathbf{S} = \mathbf{S}\mathbf{H}$$

Graph filter design and implementation

$$\mathbf{y} = \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^{\mathrm{H}}\mathbf{x} = \mathbf{H}\mathbf{x} \iff \mathbf{H}\mathbf{S} = \mathbf{S}\mathbf{H}$$

Graph-dependent vs graph-independent (universal) filter design



[Shuman'11, DCOSS]
[Sandryhaila'13, TSP]
Shuman'13, SPM]
[Segarra'18, TSP]

Graph filter design and implementation

$$\mathbf{y} = \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^{\mathrm{H}}\mathbf{x} = \mathbf{H}\mathbf{x} \iff \mathbf{H}\mathbf{S} = \mathbf{S}\mathbf{H}$$

Frequency-domain vs vertex-domain implementation

- No fast GFT implementations
- Need for parametrized filters in the vertex domain

FIR graph filters

Finite impulse response graph filters are expressible as matrix polynomials of the shift operator

$$\mathbf{y} = \mathbf{H}_{\text{FIR}}\mathbf{x} \quad \text{for} \quad \mathbf{H}_{\text{FIR}} = \sum_{k=0}^{K} \phi_k \mathbf{S}^k$$

with frequency response given by

$$h_{\text{FIR}}(\lambda_n) = \sum_{k=0}^{K} \phi_k \lambda_n^k$$

FIR graph filters

Finite impulse response graph filters are expressible as matrix polynomials of the shift operator

$$\mathbf{y} = \mathbf{H}_{\text{FIR}}\mathbf{x} \qquad \text{for} \qquad \mathbf{H}_{\text{FIR}} = \sum_{k=0}^{K} \phi_k \mathbf{S}^k$$

with frequency response given by

$$h_{\text{FIR}}(\lambda_n) = \sum_{k=0}^{K} \phi_k \lambda_n^k$$

number of parameters: $\mathcal{O}(K)$

computational complexity: $\mathcal{O}(MK)$

FIR graph filters

$$\mathbf{x}^{(k)} = \mathbf{S}^k \mathbf{x}$$

shifted graph signal

$$x(t-\tau) = z^{-\tau} x(t)$$

time-delayed signal

FIR graph filters

FIR graph filter

$$\mathbf{x}^{(k)} = \mathbf{S}^k \mathbf{x}$$

shifted graph signal

$$\mathbf{y} = \sum_{k=0}^{K} \phi_k \mathbf{x}^{(k)}$$

FIR time-domain filter

$$x(t-\tau) = z^{-\tau} x(t)$$

time-delayed signal

$$y(t) = \sum_{\tau=0}^{L} h(\tau) x(t-\tau)$$

FIR graph filters

$$\mathbf{x}^{(k)} = \mathbf{S}^k \mathbf{x}$$

shifted graph signal

FIR graph filter

$$\mathbf{y} = \sum_{k=0}^{K} \phi_k \mathbf{x}^{(k)}$$

$$x(t-\tau) = z^{-\tau} x(t)$$

time-delayed signal

FIR time-domain filter

$$y(t) = \sum_{\tau=0}^{L} h(\tau) x(t-\tau)$$

carries the notion of
convolution

↕

(shift-and-sum)

[graph convolution neural networks]

FIR graph filters

$$\mathbf{y} = \mathbf{H}_{\text{FIR}}\mathbf{x} = \sum_{k=0}^{K} \phi_k \mathbf{S}^k \mathbf{x} = \sum_{k=0}^{K} \phi_k \mathbf{x}^{(k)}$$

FIR graph filters

$$\mathbf{y} = \mathbf{H}_{\text{FIR}}\mathbf{x} = \sum_{k=0}^{K} \phi_k \mathbf{S}^k \mathbf{x} = \sum_{k=0}^{K} \phi_k \mathbf{x}^{(k)}$$

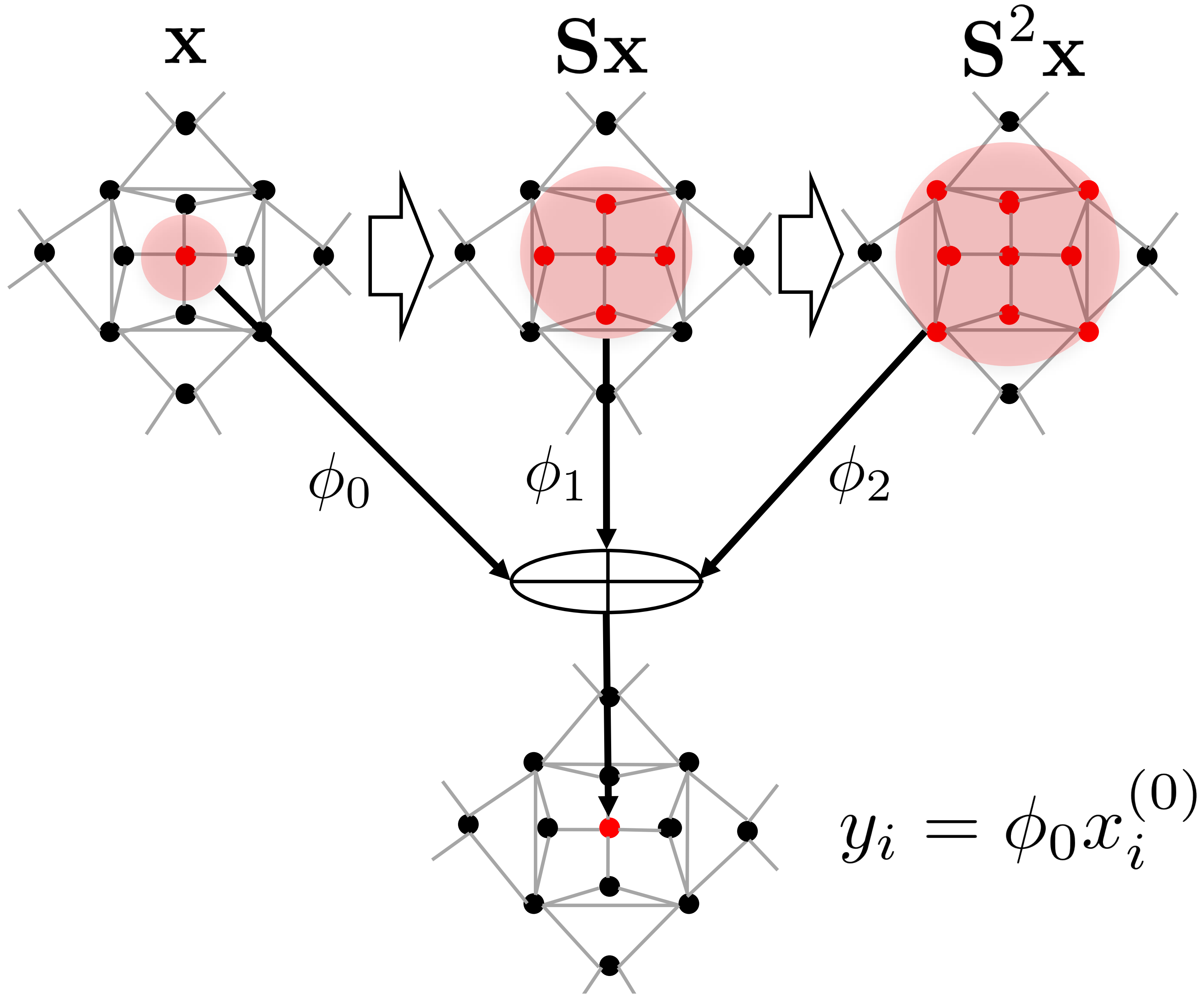
sum of shifted versions of
graph signal

FIR graph filters

$$\mathbf{y} = \mathbf{H}_{\text{FIR}}\mathbf{x} = \sum_{k=0}^{K} \phi_k \mathbf{S}^k \mathbf{x} = \sum_{k=0}^{K} \phi_k \mathbf{x}^{(k)}$$

sum of shifted versions of graph signal

Example: $\mathbf{y} = \phi_0\mathbf{x} + \phi_1\mathbf{S}\mathbf{x} + \phi_2\mathbf{S}^2\mathbf{x}$



$$x_i^{(0)} = x_i$$

$$x_i^{(1)} = \sum_{j\in\mathcal{N}_i} [\mathbf{S}]_{i,j} x_j^{(0)}$$

$$x_i^{(2)} = \sum_{j\in\mathcal{N}_i} [\mathbf{S}]_{i,j} x_j^{(1)}$$

$$y_i = \phi_0 x_i^{(0)} + \phi_1 x_i^{(1)} + \phi_2 x_i^{(2)}$$

FIR graph filters

$$\mathbf{H}_{\mathrm{FIR}} \triangleq \sum_{k=0}^{K} \phi_k \mathbf{S}^k$$

- ◉ Efficient and distributed implementation 😁

$$\begin{cases} x_i^{(0)} = x_i \\ x_i^{(k)} = \sum_{j \in \mathcal{N}_i} [\mathbf{S}]_{i,j} x_j^{(k-1)}, \quad k = 1, 2, \ldots, K \\ y_i = \sum_{k=0}^{K} \phi_k x_i^{(k)} \end{cases}$$

- ◉ Computational and communication cost of $\mathcal{O}(MK)$ 😁
- ◉ Good approximation requires high filter orders ☹️

FIR design

Minimization of error

$$e_n = \hat{h}_n - \sum_{k=0}^{K} \phi_k \lambda_n^k$$

frequency-domain
design

FIR design

Minimization of error

$$e_n = \hat{h}_n - \sum_{k=0}^{K} \phi_k \lambda_n^k$$

frequency-domain
design

- ◉ Least squares [Sandryhaila'13, TSP]

$$\mathbf{e} = \hat{\mathbf{h}} - \boldsymbol{\Xi}_{K+1}\boldsymbol{\phi}$$

$$\min_{\phi} ||\hat{\mathbf{h}} - \boldsymbol{\Xi}_{K+1}\boldsymbol{\phi}||_2^2$$

$$[\boldsymbol{\Xi}_{K+1}]_{n,k} = \lambda_n^{k-1} \qquad \boldsymbol{\Xi}_{K+1} \in \mathbb{R}^{N\times(K+1)}$$

FIR design

Minimization of error

$$e_n = \hat{h}_n - \sum_{k=0}^{K} \phi_k \lambda_n^k$$

frequency-domain
design

alternative data-driven
design

$$\min_{\{\phi_k\}} \sum_i \|\mathbf{y}_i - \mathbf{H}(\{\phi_k\})\mathbf{x}_i\|_2^2$$

- Least squares [Sandryhaila'13, TSP]

$$\Rightarrow \quad \mathbf{e} = \hat{\mathbf{h}} - \boldsymbol{\Xi}_{K+1}\boldsymbol{\phi}$$

$$\Rightarrow \quad \min_{\phi} \|\hat{\mathbf{h}} - \boldsymbol{\Xi}_{K+1}\boldsymbol{\phi}\|_2^2$$

$$[\boldsymbol{\Xi}_{K+1}]_{n,k} = \lambda_n^{k-1} \qquad \boldsymbol{\Xi}_{K+1} \in \mathbb{R}^{N\times(K+1)}$$

FIR design

- Chebyshev [Shuman'11, DCOSS]

$$\hat{h}(\lambda) = \sum_{k=0}^{\infty} c_k T_k(\lambda) \approx \sum_{k=0}^{K} c_k T_k(\lambda)$$

- ✦ $T_k(\lambda)$: modified Chebyshev polynomials; orthogonal over desired range
- ✦ Closed form expression for $\{c_k\}_{k=0}^{K}$

ARMA graph filters

Autoregresive moving average graph filters implement a fractional matrix polynomial of the shift operator

$$\mathbf{y} = \mathbf{H}_{\text{ARMA}}\mathbf{x} \qquad \text{for} \qquad \mathbf{H}_{\text{ARMA}} = \left(\mathbf{I} - \sum_{p=1}^{P} \psi_p \mathbf{S}^p\right)^{-1} \left(\sum_{q=0}^{Q} \varphi_q \mathbf{S}^q\right)$$

with frequency response given by

$$h_{\text{ARMA}}(\lambda_n) = \frac{\sum_{q=0}^{Q} \varphi_q \lambda_n^q}{1 + \sum_{p=1}^{P} \psi_p \lambda_n^p}$$

Isufi, Loukas, Simonetto, Leus, *Autoregressive Moving Average Graph Filtering*, IEEE Transactions on Signal Processing, 2017

ARMA graph filters

Autoregresive moving average graph filters implement a fractional matrix polynomial of the shift operator

$$\mathbf{y} = \mathbf{H}_{\text{ARMA}}\mathbf{x} \qquad \text{for} \qquad \mathbf{H}_{\text{ARMA}} = \left(\mathbf{I} - \sum_{p=1}^{P} \psi_p \mathbf{S}^p\right)^{-1} \left(\sum_{q=0}^{Q} \varphi_q \mathbf{S}^q\right)$$

with frequency response given by

$$h_{\text{ARMA}}(\lambda_n) = \frac{\sum_{q=0}^{Q} \varphi_q \lambda_n^q}{1 + \sum_{p=1}^{P} \psi_p \lambda_n^p}$$

number of parameters: $\mathcal{O}(P + Q)$

computational complexity: $\mathcal{O}(M)$ per it.

Isufi, Loukas, Simonetto, Leus, *Autoregressive Moving Average Graph Filtering*, IEEE Transactions on Signal Processing, 2017

ARMA graph filters

$$\mathbf{x}^{(k)} = \mathbf{S}^k \mathbf{x}$$

shifted graph signal

$$x(t-\tau) = z^{-\tau} x(t)$$

time-delayed input signal

ARMA graph filters

ARMA graph filter

$$\mathbf{x}^{(k)} = \mathbf{S}^k\mathbf{x}$$

shifted graph signal

$$\mathbf{y}^{(0)} = \sum_{q=0}^{Q} \varphi_k \mathbf{x}^{(q)} + \sum_{p=1}^{P} \psi_p \mathbf{y}^{(p)}$$

ARMA time-domain filter

$$x(t-\tau) = z^{-\tau}x(t)$$

time-delayed input signal

$$y(t) = \sum_{\tau=0}^{L} h(\tau)x(t-\tau) + \sum_{\kappa=1}^{R} g(\kappa)y(t-\kappa)$$

ARMA graph filters

ARMA graph filter

$$\mathbf{y}^{(0)} = \sum_{q=0}^{Q} \varphi_k \mathbf{x}^{(q)} + \sum_{p=1}^{P} \psi_p \mathbf{y}^{(p)}$$

$$\mathbf{x}^{(k)} = \mathbf{S}^k \mathbf{x}$$

shifted graph signal

not easy
to implement

requires shifted
version of the output

ARMA time-domain filter

$$y(t) = \sum_{\tau=0}^{L} h(\tau)x(t-\tau) + \sum_{\kappa=1}^{R} g(\kappa)y(t-\kappa)$$

$$x(t-\tau) = z^{-\tau}x(t)$$

time-delayed input signal

easy to implement

ARMA graph filters

$$\mathbf{H}_{\text{ARMA}} = \left(\mathbf{I} - \sum_{p=1}^{P} \psi_p \mathbf{S}^p\right)^{-1} \left(\sum_{q=0}^{Q} \varphi_q \mathbf{S}^q\right)$$

- 🟢 Stability is guaranteed by invertibility 😁
- 🟢 Good approximation for low filter orders 😁
- 🟢 Exact solution for denoising/interpolation/diffusion 😁
- 🔴 Filter design is more involved than for FIR ☹️
- 🔴 Does not admit trivial efficient/distributed implementation ☹️

ARMA implementation

$$\mathbf{H}_{\text{ARMA}} = \left(\mathbf{I} - \sum_{p=1}^{P} \psi_p \mathbf{S}^p\right)^{-1} \left(\sum_{q=0}^{Q} \varphi_q \mathbf{S}^q\right)$$

- Moving average part: similar to FIR
- Autoregressive part:
  - Gradient descent [Shi’15, SPL][Loukas’15, SPL]
  - Conjugate gradient [Liu’17, GlobalSIP]
  - Any other Krylov-based inversion can be used

Liu, Isufi, Leus, *Filter Design for Autoregressive Moving Average Graph Filters*, IEEETransactions on Signal and Information Processing over Networks, 2019

ARMA implementation

$$\mathbf{H}_{\text{ARMA}} = \left(\mathbf{I} - \sum_{p=1}^{P} \psi_p \mathbf{S}^p\right)^{-1} \left(\sum_{q=0}^{Q} \varphi_q \mathbf{S}^q\right)$$

- Distributed methods: (Jacobi)
  - Parallel or serial implementation of heat kernel

$$\mathbf{y}_{t+1} = \psi \mathbf{S} \mathbf{y}_t + \varphi \mathbf{x}$$

  - Direct implementation

$$\mathbf{y}_t = -\sum_{p=1}^{P} \psi_p \mathbf{S}^p \mathbf{y}_{t-1} + \sum_{q=0}^{Q} \varphi_q \mathbf{S}^q \mathbf{x}$$

Isufi, Loukas, Leus, *Autoregressive moving average graph filters a stable distributed implementation*, IEEE ICASSP, 2017

ARMA implementation

$$\mathbf{H}_{\text{ARMA}} = \left(\mathbf{I} - \sum_{p=1}^{P} \psi_p \mathbf{S}^p\right)^{-1} \left(\sum_{q=0}^{Q} \varphi_q \mathbf{S}^q\right)$$

- Distributed methods: (Jacobi)
  - Parallel or serial implementation of heat kernel

$$\mathbf{y}_{t+1} = \psi \mathbf{S} \mathbf{y}_t + \varphi \mathbf{x}$$

  - Direct implementation

$$\mathbf{y}_t = -\sum_{p=1}^{P} \psi_p \mathbf{S}^p \mathbf{y}_{t-1} + \sum_{q=0}^{Q} \varphi_q \mathbf{S}^q \mathbf{x}$$

cost per iteration

$\mathcal{O}(M)$

$\mathcal{O}(\max\{P, Q\}M)$

Isufi, Loukas, Leus, *Autoregressive moving average graph filters a stable distributed implementation*, IEEE ICASSP, 2017

ARMA design

Minimization of error

frequency-domain
design

$$e_n = \hat{h}_n - \frac{\sum_{q=0}^{Q} \varphi_q \lambda_n^q}{1 + \sum_{p=1}^{P} \psi_p \lambda_n^p}$$

β_n

α_n

Liu, Isufi, Leus, *Filter Design for Autoregressive Moving Average Graph Filters*, IEEE Transactions on Signal and Information Processing over Networks, 2019

ARMA design

Minimization of error

frequency-domain
design

$$e_n = \hat{h}_n - \frac{\sum_{q=0}^{Q} \varphi_q \lambda_n^q \;\longleftarrow\; \beta_n}{1 + \sum_{p=1}^{P} \psi_p \lambda_n^p \;\longleftarrow\; \alpha_n}$$

- Prony's method

➡

$$e'_n = \hat{h}_n \alpha_n - \beta_n$$

modified error

Liu, Isufi, Leus, *Filter Design for Autoregressive Moving Average Graph Filters*, IEEE Transactions on Signal and Information Processing over Networks, 2019

ARMA design

$$e'_n = \hat{h}_n \alpha_n - \beta_n$$

Liu, Isufi, Leus, *Autoregressive moving average graph filter design*, IEEE GlobalSIP, 2017

ARMA design

$$e'_n = \hat{h}_n \alpha_n - \beta_n$$

modified error is linear in $\{\psi, \varphi\}$

$$\Rightarrow \quad \mathbf{e}' = \hat{\mathbf{h}} \circ \boldsymbol{\alpha} - \boldsymbol{\beta}$$

Liu, Isufi, Leus, *Autoregressive moving average graph filter design*, IEEE GlobalSIP, 2017

ARMA design

$$e'_n = \hat{h}_n \alpha_n - \beta_n$$

modified error is linear in $\{\boldsymbol{\psi}, \boldsymbol{\varphi}\}$

$$\boldsymbol{\alpha} = \boldsymbol{\Xi}_{P+1}\boldsymbol{\psi}, \ \ \psi_0 = 1$$

$$[\boldsymbol{\Xi}_{P+1}]_{n,p} = \lambda_n^{p-1}$$

$$\boldsymbol{\beta} = \boldsymbol{\Xi}_{Q+1}\boldsymbol{\varphi}$$

$$[\boldsymbol{\Xi}_{Q+1}]_{n,q} = \lambda_n^{q-1}$$

$$\Rightarrow \quad \mathbf{e}' = \hat{\mathbf{h}} \circ \boldsymbol{\alpha} - \boldsymbol{\beta}$$

$$\Rightarrow \quad \min_{\boldsymbol{\psi},\boldsymbol{\varphi}} \|\hat{\mathbf{h}} \circ (\boldsymbol{\Xi}_{P+1}\boldsymbol{\psi}) - \boldsymbol{\Xi}_{Q+1}\boldsymbol{\varphi}\|_2^2$$

Liu, Isufi, Leus, *Autoregressive moving average graph filter design*, IEEE GlobalSIP, 2017

ARMA design

$$e'_n = \hat{h}_n \alpha_n - \beta_n$$

modified error is linear in $\{\boldsymbol{\psi}, \boldsymbol{\varphi}\}$

$$\mathbf{e}' = \hat{\mathbf{h}} \circ \boldsymbol{\alpha} - \boldsymbol{\beta}$$

$$\boldsymbol{\alpha} = \boldsymbol{\Xi}_{P+1}\boldsymbol{\psi}, \quad \psi_0 = 1$$

$$[\boldsymbol{\Xi}_{P+1}]_{n,p} = \lambda_n^{p-1}$$

$$\boldsymbol{\beta} = \boldsymbol{\Xi}_{Q+1}\boldsymbol{\varphi}$$

$$[\boldsymbol{\Xi}_{Q+1}]_{n,q} = \lambda_n^{q-1}$$

$$\min_{\boldsymbol{\psi},\boldsymbol{\varphi}} \|\hat{\mathbf{h}} \circ (\boldsymbol{\Xi}_{P+1}\boldsymbol{\psi}) - \boldsymbol{\Xi}_{Q+1}\boldsymbol{\varphi}\|_2^2$$

alternative data-driven design

$$\min_{\{\psi_p,\varphi_q\}} \sum_i \|\mathbf{A}(\{\psi_p\})\mathbf{y}_i - \mathbf{B}(\{\phi_q\}\mathbf{x}_i\|_2^2$$

Liu, Isufi, Leus, *Autoregressive moving average graph filter design*, IEEE GlobalSIP, 2017

ARMA design

- Iterative method

$$e_n = (\hat{h}_n \alpha_n - \beta_n)\gamma_n, \, \gamma_n = 1/\alpha_n$$

Liu, Isufi, Leus, *Filter Design for Autoregressive Moving Average Graph Filters*, IEEE Transactions on Signal and Information Processing over Networks, 2019

ARMA design

- Iterative method

for known $\boldsymbol{\gamma}$,
error is linear in $\{\boldsymbol{\psi}, \boldsymbol{\varphi}\}$

$$e_n = (\hat{h}_n \alpha_n - \beta_n)\gamma_n,\ \gamma_n = 1/\alpha_n$$

$$\mathbf{e} = (\hat{\mathbf{h}} \circ \boldsymbol{\alpha} - \boldsymbol{\beta}) \circ \boldsymbol{\gamma}$$

$$\boldsymbol{\alpha} = \boldsymbol{\Xi}_{P+1}\boldsymbol{\psi},\ \ \psi_0 = 1$$
$$[\boldsymbol{\Xi}_{P+1}]_{n,p} = \lambda_n^{p-1}$$
$$\boldsymbol{\beta} = \boldsymbol{\Xi}_{Q+1}\boldsymbol{\varphi}$$
$$[\boldsymbol{\Xi}_{Q+1}]_{n,q} = \lambda_n^{q-1}$$

Liu, Isufi, Leus, *Filter Design for Autoregressive Moving Average Graph Filters*, IEEE Transactions on Signal and Information Processing over Networks, 2019

ARMA design

- Iterative method

$$\boldsymbol{\alpha} = \boldsymbol{\Xi}_{P+1}\boldsymbol{\psi}, \ \ \psi_0 = 1$$

$$[\boldsymbol{\Xi}_{P+1}]_{n,p} = \lambda_n^{p-1}$$

$$\boldsymbol{\beta} = \boldsymbol{\Xi}_{Q+1}\boldsymbol{\varphi}$$

$$[\boldsymbol{\Xi}_{Q+1}]_{n,q} = \lambda_n^{q-1}$$

$$e_n = (\hat{h}_n\alpha_n - \beta_n)\gamma_n, \, \gamma_n = 1/\alpha_n$$

for known $\boldsymbol{\gamma}$,

error is linear in $\{\boldsymbol{\psi}, \boldsymbol{\varphi}\}$

$$\mathbf{e} = (\hat{\mathbf{h}} \circ \boldsymbol{\alpha} - \boldsymbol{\beta}) \circ \boldsymbol{\gamma}$$

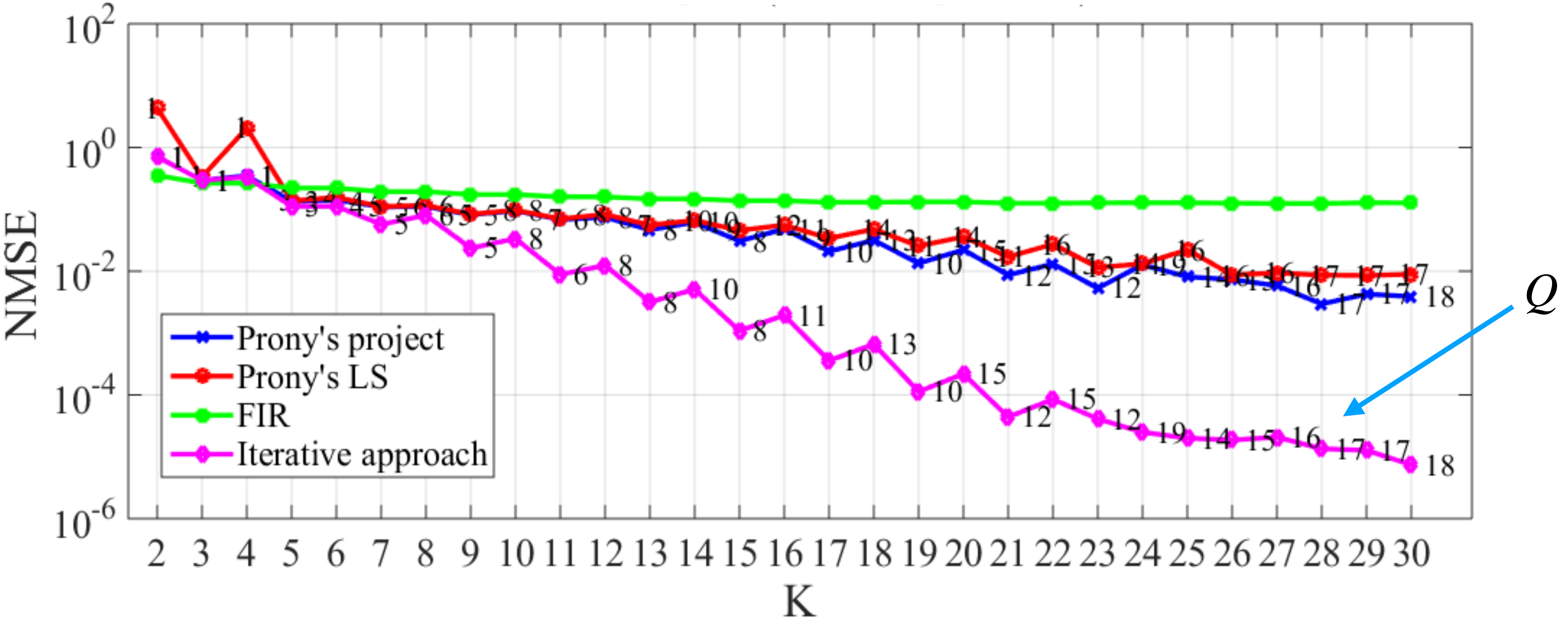
$$\min_{\psi_i, \varphi_i} \|\boldsymbol{\gamma}_{i-1} \circ \left[\hat{\mathbf{h}} \circ \left(\boldsymbol{\Xi}_{P+1}\boldsymbol{\psi}_i - \boldsymbol{\Xi}_{Q+1}\boldsymbol{\varphi}_i\right)\right]\|_2^2$$

Liu, Isufi, Leus, *Filter Design for Autoregressive Moving Average Graph Filters*, IEEE Transactions on Signal and Information Processing over Networks, 2019

ARMA design results

Approximate an ideal filter with $\mathbf{S} = \mathbf{L}_{\mathrm{n}}, \lambda_c = 1, P + Q \leq K$

e.g., graph spectral clustering [Tremblay'16, ICASSP]



Beyond classical graph filtering

FIR and IIR extensions

- Node-varying graph filters and edge-varying graph filters

[Segarra’17, TSP] [Coutino’17, CAMSAP]

FIR and IIR extensions

- Node-varying graph filters and edge-varying graph filters

  [Segarra’17, TSP] [Coutino’17, CAMSAP]

- Edge-varying for both FIR and ARMA graph filters [Coutino’19, TSP]

FIR and IIR extensions

- Node-varying graph filters and edge-varying graph filters

  [Segarra’17, TSP] [Coutino’17, CAMSAP]

- Edge-varying for both FIR and ARMA graph filters [Coutino’19, TSP]

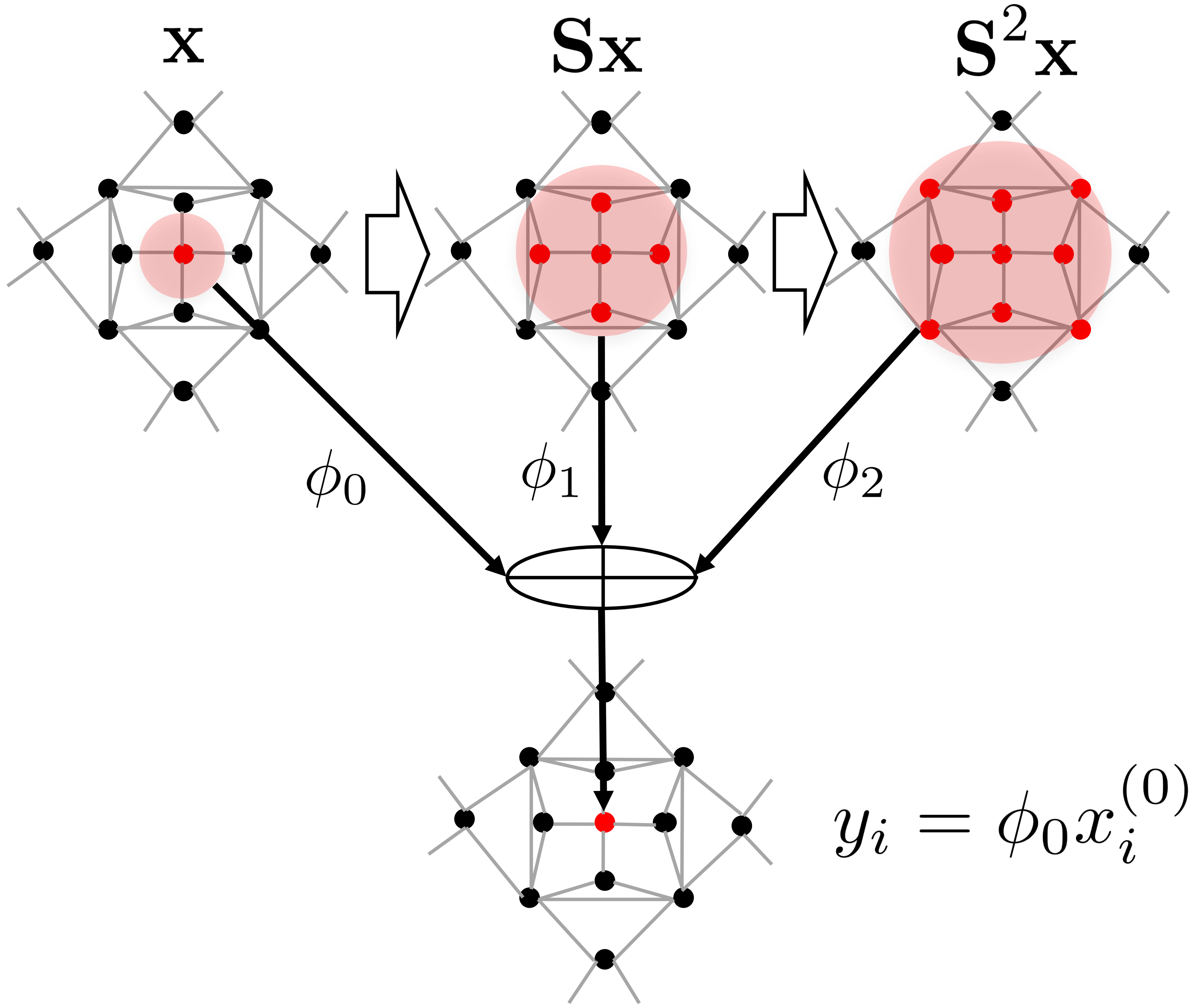
- Nonlinear graph filters
  - ✦Weighted median graph filters [Segarra’16, GlobalSIP]
  - ✦Activation functions (graph CNNs) [Bruna’13]

FIR graph filters

Example: $\mathbf{y} = \phi_0 \mathbf{x} + \phi_1 \mathbf{S}\mathbf{x} + \phi_2 \mathbf{S}^2 \mathbf{x}$

FIR graph filters

Example: $\mathbf{y} = \phi_0\mathbf{x} + \phi_1\mathbf{S}\mathbf{x} + \phi_2\mathbf{S}^2\mathbf{x}$



$$x_i^{(0)} = x_i$$

$$x_i^{(1)} = \sum_{j \in \mathcal{N}_i} [\mathbf{S}]_{i,j} x_j^{(0)}$$

$$x_i^{(2)} = \sum_{j \in \mathcal{N}_i} [\mathbf{S}]_{i,j} x_j^{(1)}$$

$$y_i = \phi_0 x_i^{(0)} + \phi_1 x_i^{(1)} + \phi_2 x_i^{(2)}$$

FIR graph filters

$$\mathbf{H}_{\mathrm{C}} \triangleq \sum_{k=0}^{K} \phi_k \mathbf{S}^k$$

- 🟢 Efficient and distributed implementation 😁

$$\begin{cases} x_i^{(0)} = x_i \\ x_i^{(k)} = \sum_{j \in \mathcal{N}_i} [\mathbf{S}]_{i,j} x_j^{(k-1)}, \quad k = 1, 2, \ldots, K \\ y_i = \sum_{k=0}^{K} \phi_k x_i^{(k)} \end{cases}$$

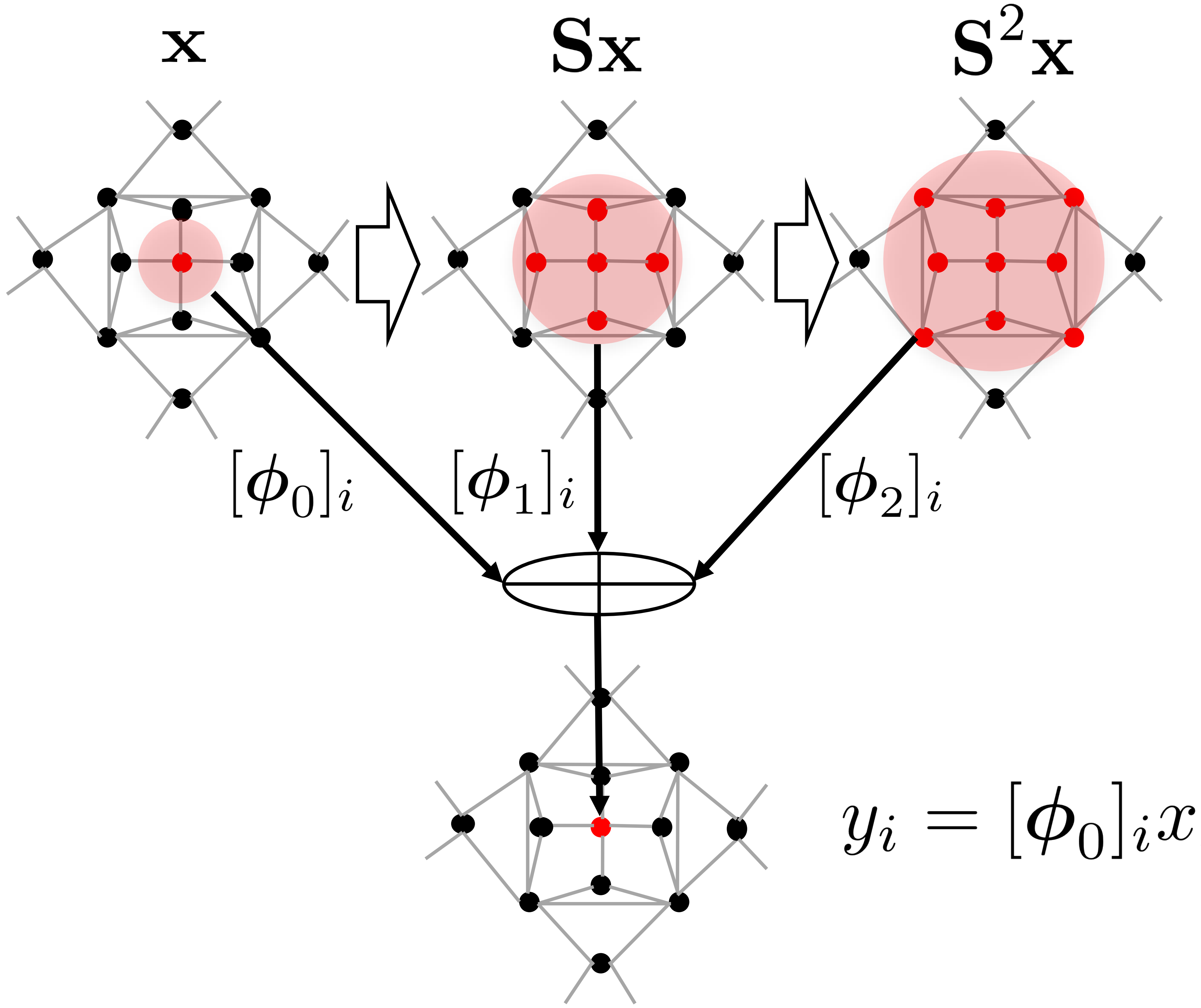
- 🟢 Computational and communication cost of $\mathcal{O}(MK)$ 😁
- 🟢 Linear in scalar coefficients $\{\phi_k\}$ 😁
- 🔴 Good approximation requires high filter orders ☹️

Node-varying graph filters

Example: $\mathbf{y} = \mathrm{diag}(\boldsymbol{\phi}_0)\mathbf{x} + \mathrm{diag}(\boldsymbol{\phi}_1)\mathbf{S}\mathbf{x} + \mathrm{diag}(\boldsymbol{\phi}_2)\mathbf{S}^2\mathbf{x}$

Node-varying graph filters

Example: $\mathbf{y} = \text{diag}(\boldsymbol{\phi}_0)\mathbf{x} + \text{diag}(\boldsymbol{\phi}_1)\mathbf{S}\mathbf{x} + \text{diag}(\boldsymbol{\phi}_2)\mathbf{S}^2\mathbf{x}$



$$x_i^{(0)} = x_i$$

$$x_i^{(k)} = \sum_{j \in \mathcal{N}_i} [\mathbf{S}]_{i,j} x_j^{(k-1)}$$

$$y_i = [\phi_0]_i x_i^{(0)} + [\phi_1]_i x_i^{(1)} + [\phi_2]_i x_i^{(2)}$$

Node-varying graph filters

$$\mathbf{H}_{\mathrm{NV}} \triangleq \sum_{k=0}^{K} \operatorname{diag}\{\boldsymbol{\phi}_k\}\mathbf{S}^k$$

- Efficient and distributed implementation 😁

$$\begin{cases} x_i^{(0)} = x_i \\ x_i^{(k)} = \sum_{j\in\mathcal{N}_i}[\mathbf{S}]_{i,j}x_j^{(k-1)}, \;\; k = 1, 2, \ldots, K \\ y_i = \sum_{k=0}^{K}[\boldsymbol{\phi}_k]_i x_i^{(k)} \end{cases}$$

- Computational and communication cost of $\mathcal{O}(MK)$ 😁
- Specializes to classical graph filter 😁
- Linear in vector coefficients $\{\boldsymbol{\phi}_k\}$ 😁

Segarra, Marques, Ribeiro, *Optimal Graph-Filter Design and Applications to Distributed Linear Network Operators*, IEEE Transactions on Signal Processing, 2017

Edge-varying graph filters

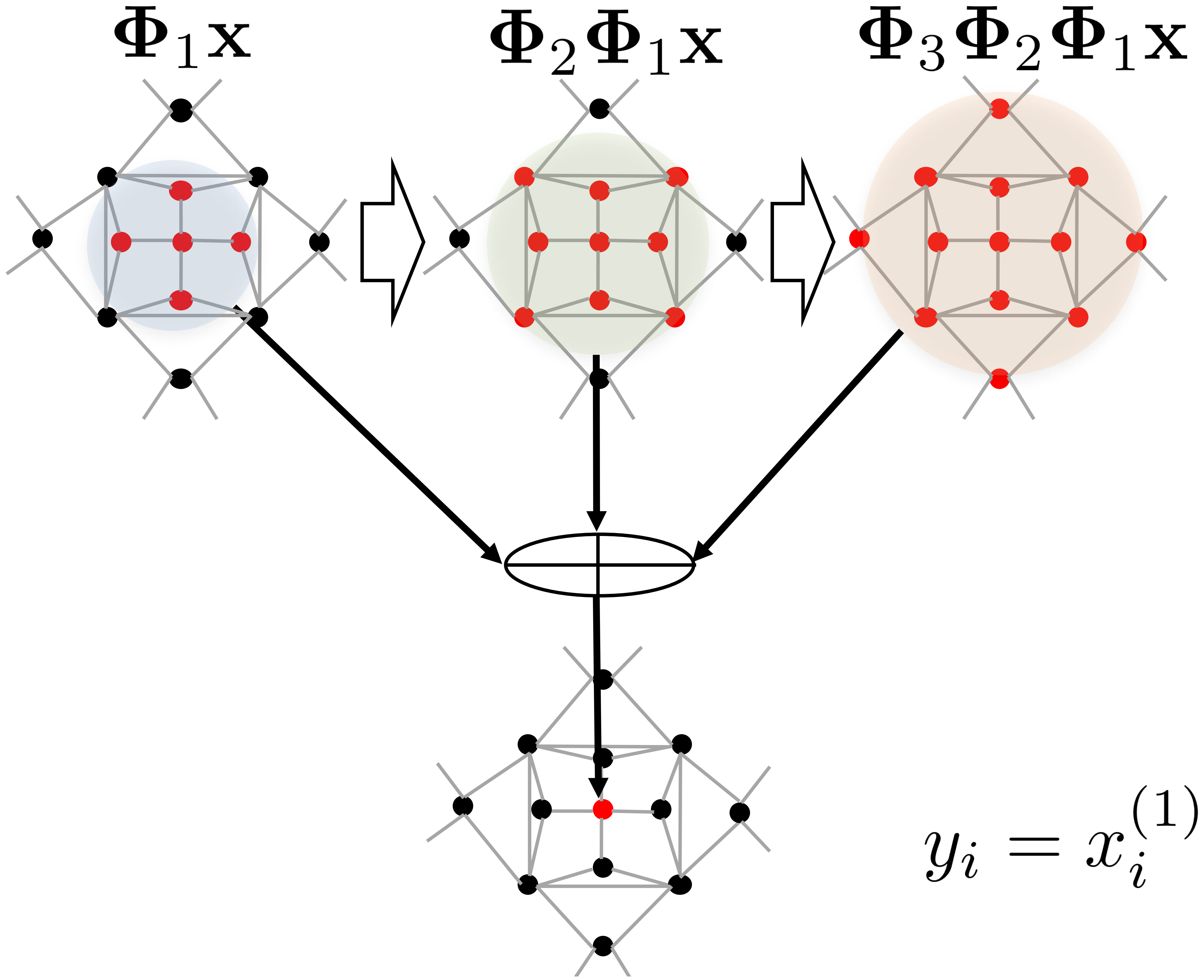
$\mathbf{\Phi}_k$ same support as $\mathbf{S} + \mathbf{I}$

Example: $\mathbf{y} = \mathbf{\Phi}_1\mathbf{x} + \mathbf{\Phi}_2\mathbf{\Phi}_1\mathbf{x} + \mathbf{\Phi}_3\mathbf{\Phi}_2\mathbf{\Phi}_1\mathbf{x}$

Edge-varying graph filters

$\mathbf{\Phi}_k$ same support as $\mathbf{S}+\mathbf{I}$

Example: $\mathbf{y} = \mathbf{\Phi}_1\mathbf{x} + \mathbf{\Phi}_2\mathbf{\Phi}_1\mathbf{x} + \mathbf{\Phi}_3\mathbf{\Phi}_2\mathbf{\Phi}_1\mathbf{x}$



$$x_i^{(0)} = x_i$$

$$x_i^{(k)} = \sum_{j \in \mathcal{N}_i} [\mathbf{\Phi}_k]_{i,j} x_j^{(k-1)}$$

$$y_i = x_i^{(1)} + x_i^{(2)} + x_i^{(3)}$$

Coutino, Isufi, Leus, *Advances in Distributed Graph Filtering*, IEEE Transactions on Signal Processing, 2019

Edge-varying graph filters

$$\mathbf{H}_{\text{EV}} \triangleq \sum_{k=1}^{K} \prod_{l=1}^{k} \boldsymbol{\Phi}_l$$

- Efficient and distributed implementation 😁

$$\begin{cases} x_i^{(0)} = x_i \\ x_i^{(k)} = \sum_{j \in \mathcal{N}_i} [\boldsymbol{\Phi}_k]_{i,j} x_j^{(k-1)}, \;\; k = 1, 2, \ldots, K \\ y_i = \sum_{k=1}^{K} x_i^{(k)} \end{cases}$$

- Computational and communication cost of $\mathcal{O}(MK)$ 😁
- Specializes to classical and node-varying graph filter 😁
- Non-linear in matrix coefficients $\{\boldsymbol{\Phi}_k\}$ ☹

Constrained edge-varying graph filter

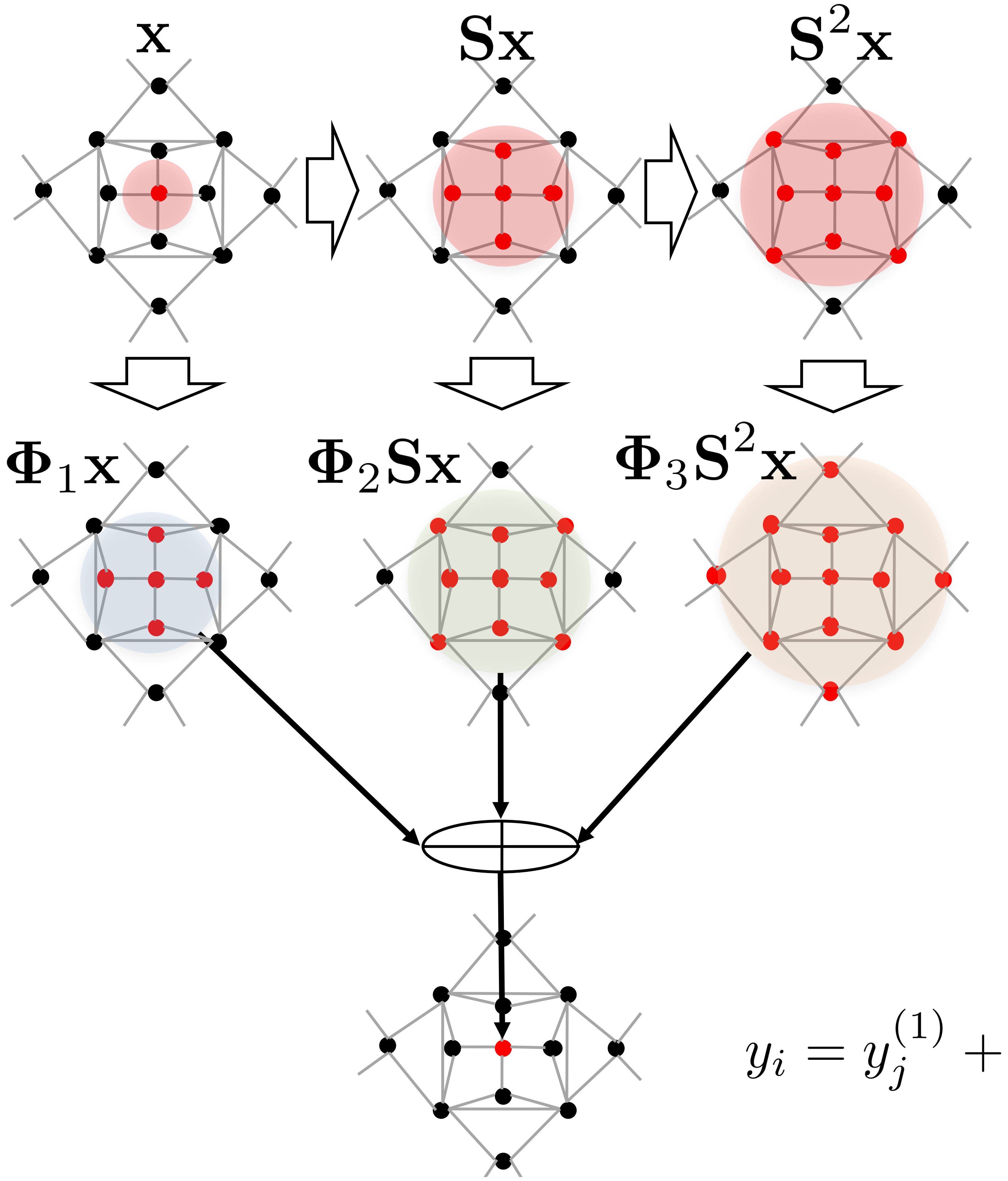
$\boldsymbol{\Phi}_k$ same support as $\mathbf{S} + \mathbf{I}$

Example: $\mathbf{y} = \boldsymbol{\Phi}_1\mathbf{x} + \boldsymbol{\Phi}_2\mathbf{S}\mathbf{x} + \boldsymbol{\Phi}_3\mathbf{S}^2\mathbf{x}$

Constrained edge-varying graph filter

$\mathbf{\Phi}_k$ same support as $\mathbf{S}+\mathbf{I}$

Example: $\mathbf{y} = \mathbf{\Phi}_1\mathbf{x} + \mathbf{\Phi}_2\mathbf{S}\mathbf{x} + \mathbf{\Phi}_3\mathbf{S}^2\mathbf{x}$



$$x_i^{(0)} = x_i$$

$$x_i^{(k)} = \sum_{j\in\mathcal{N}_i} [\mathbf{S}]_{i,j} x_j^{(k-1)}$$

$$y_i^{(k)} = \sum_{j\in\mathcal{N}_i} [\mathbf{\Phi}_k]_{i,j} x_j^{(k-1)}$$

$$y_i = y_j^{(1)} + y_j^{(2)} + y_j^{(3)}$$

Coutino, Isufi, Leus, *Distributed edge-variant graph filters*, IEEE CAMSAP, 2017

Constrained edge-varying graph filter

$$\mathbf{H}_{\text{CEV}} \triangleq \sum_{k=1}^{K} \mathbf{\Phi}_k \mathbf{S}^{k-1}$$

- Efficient and distributed implementation 😁

$$\begin{cases} x_i^{(0)} = x_i \\ x_i^{(k)} = \sum_{j \in \mathcal{N}_i} [\mathbf{S}]_{i,j} x_j^{(k-1)}, \;\; k = 1, 2, \ldots, K-1 \\ y_i^{(k)} = \sum_{j \in \mathcal{N}_i} [\mathbf{\Phi}_k]_{i,j} x_j^{(k-1)}, \;\; k = 1, 2, \ldots, K \\ y_i = \sum_{k=1}^{K} y_i^{(k)} \end{cases}$$

- Computational and communication cost of $\mathcal{O}(MK)$ 😁
- Specializes to classical and node-varying graph filter 😁
- Linear in matrix coefficients $\{\mathbf{\Phi}_k\}$ 😁

Node-domain graph filter design

Filter response fitting

$$\min_{\mathbf{\Theta}} \|\tilde{\mathbf{H}} - \mathbf{H}_{\text{fit}}(\mathbf{\Theta})\|^2$$

where

- $\tilde{\mathbf{H}}$ is he desired filter response
- $\|\cdot\|$ appropriate norm, e.g., Frobenius norm, spectral radius, etc.,

and

$$\mathbf{H}_{\text{fit}}(\mathbf{\Theta}) = \begin{cases} \mathbf{H}_{\text{C}}, & \mathbf{\Theta} = \{\phi_k\} \\ \mathbf{H}_{\text{NV}}, & \mathbf{\Theta} = \{\boldsymbol{\phi}_k\} \\ \mathbf{H}_{\text{CEV}}, & \mathbf{\Theta} = \{\mathbf{\Phi}_k\} \end{cases}$$

Node-domain graph filter design

Filter response fitting

$$\min_{\boldsymbol{\Theta}} \|\tilde{\mathbf{H}} - \mathbf{H}_{\text{fit}}(\boldsymbol{\Theta})\|^2$$

alternative data-driven design

$$\min_{\boldsymbol{\Theta}} \sum_i \|\mathbf{y}_i - \mathbf{H}_{\text{fit}}(\boldsymbol{\Theta})\mathbf{x}_i\|^2$$

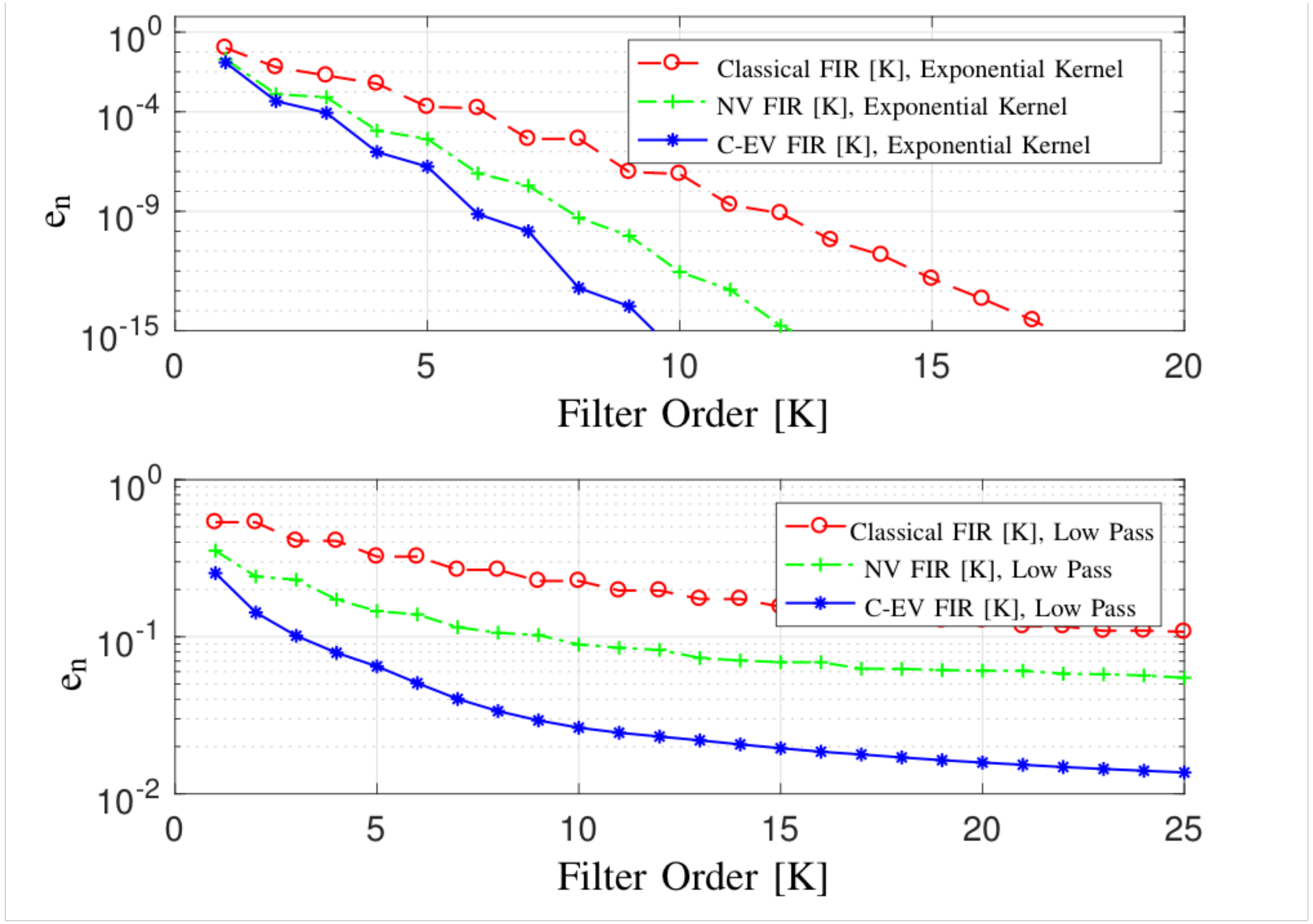
where

- $\tilde{\mathbf{H}}$ is he desired filter response
- $\|\cdot\|$ appropriate norm, e.g., Frobenius norm, spectral radius, etc.,

and

$$\mathbf{H}_{\text{fit}}(\boldsymbol{\Theta}) = \begin{cases} \mathbf{H}_{\text{C}}, & \boldsymbol{\Theta} = \{\phi_k\} \\ \mathbf{H}_{\text{NV}}, & \boldsymbol{\Theta} = \{\boldsymbol{\phi}_k\} \\ \mathbf{H}_{\text{CEV}}, & \boldsymbol{\Theta} = \{\boldsymbol{\Phi}_k\} \end{cases}$$

Fitting graph frequency response



$$e_{\mathrm{n}} = \|\tilde{\mathbf{H}} - \mathbf{H}_{\mathrm{fit}}\|_{\mathrm{F}}^{2} / \|\tilde{\mathbf{H}}\|_{\mathrm{F}}^{2}$$

Exponential kernel

$$\tilde{h}(\lambda) = e^{-3(\lambda-0.75)^2}$$

Ideal low-pass filter

$$\tilde{h}(\lambda) = \begin{cases} 1 & 0 \leq \lambda \leq \lambda_c \\ 0 & \text{otw} \end{cases}$$

part 1 :: conclusions

- Graph signal processing is an exciting new tool set for processing unstructured data

part 1 :: conclusions

- Graph signal processing is an exciting new tool set for processing unstructured data
- Graph filtering
  - Applications: denoising, interpolation, distributed optimization, neural networks
  - FIR and ARMA versions: simple (iterative) least squares design, efficient and/or distributed implementations

part 1 :: conclusions

- Graph signal processing is an exciting new tool set for processing unstructured data
- Graph filtering
  - Applications: denoising, interpolation, distributed optimization, neural networks
  - FIR and ARMA versions: simple (iterative) least squares design, efficient and/or distributed implementations
- Advanced graph filters (discussed for FIR)
  - Node-varying, edge-varying, constrained edge-varying

part 1 :: conclusions

- Graph signal processing is an exciting new tool set for processing unstructured data
- Graph filtering
  - Applications: denoising, interpolation, distributed optimization, neural networks
  - FIR and ARMA versions: simple (iterative) least squares design, efficient and/or distributed implementations
- Advanced graph filters (discussed for FIR)
  - Node-varying, edge-varying, constrained edge-varying
- Edge-varying graph filters
  - Most general form
  - Reduction in communication and computational cost
  - Constrained form allows for easy least squares design

part 2
graph filters for distributed optimization [1]

part 2 :: overview

part 2 :: overview

◉ Introduction

- Distributed optimization
- Connection to graph filtering

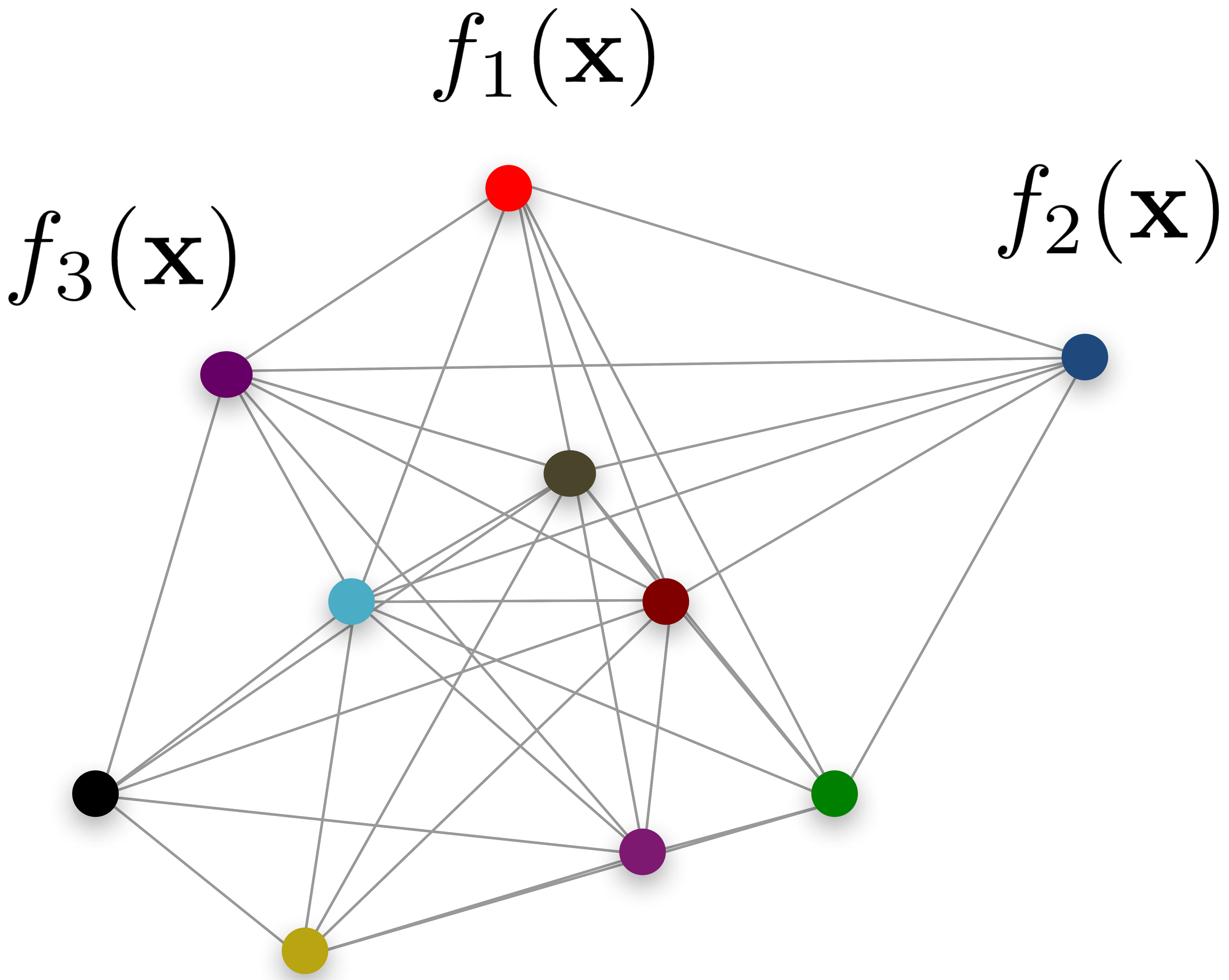
part 2 :: overview

- ◉ Introduction
  - Distributed optimization
  - Connection to graph filtering
- ◉ Applications
  - Average consensus
  - Distributed imaging
  - Distributed beamforming

Distributed optimization

Distributed optimization

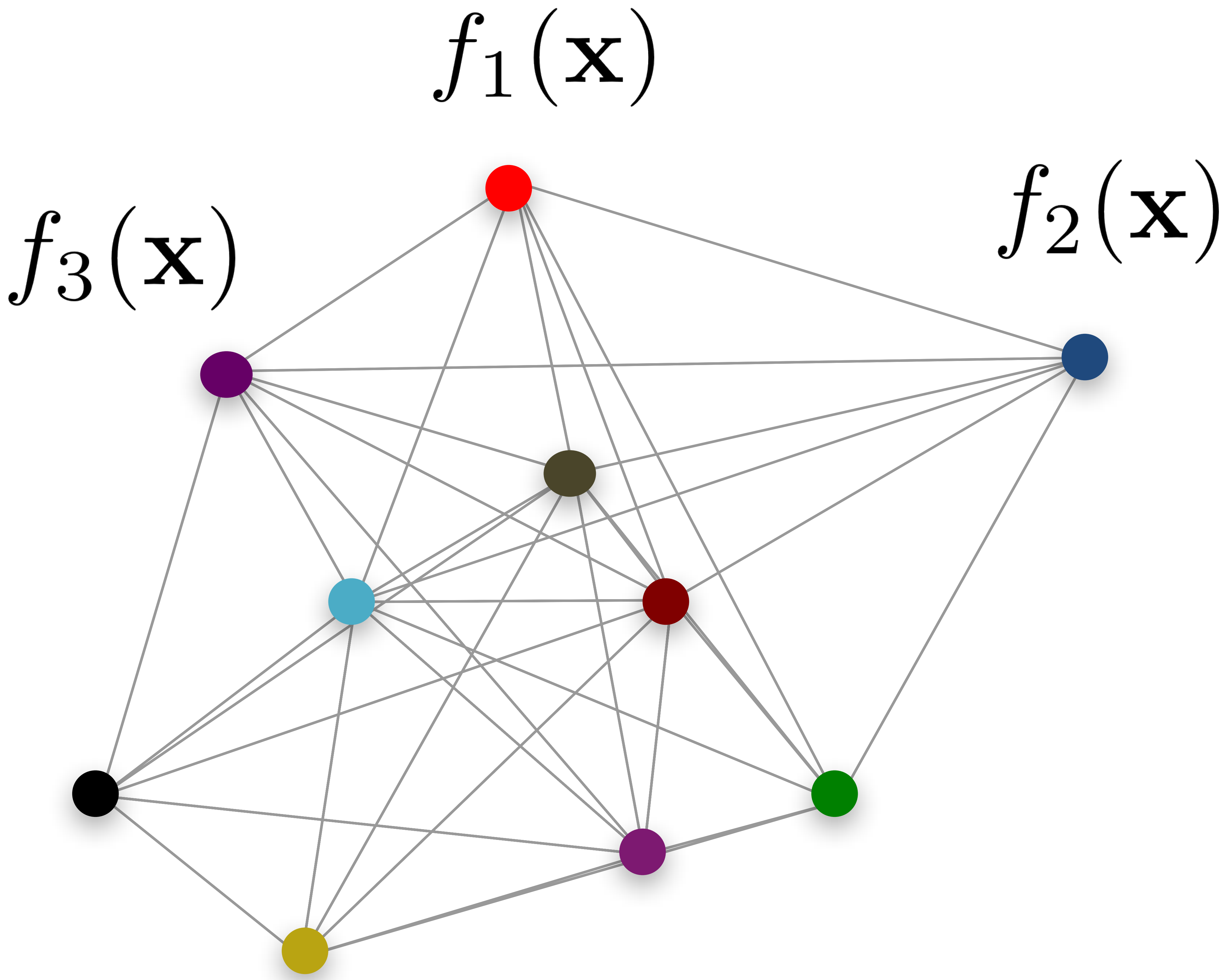
$$\min_{\mathbf{x}} \; f(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x})$$



Distributed optimization

$$\min_{\mathbf{x}} \ f(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x})$$

requires data exchanges within the network

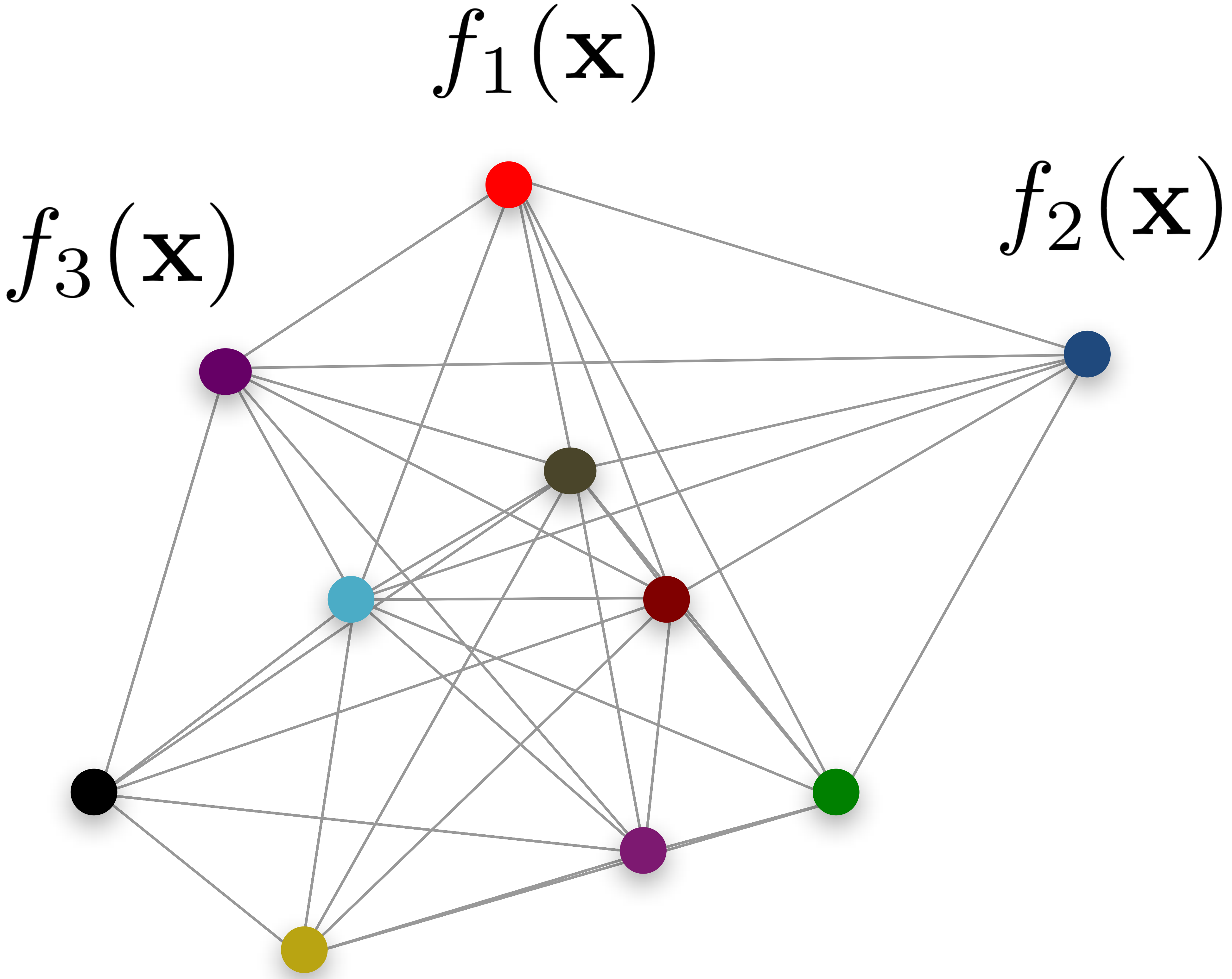


Distributed optimization

$$\min_{\mathbf{x}} \ f(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x})$$

requires data exchanges within the network

diffusions over the graph



Distributed optimization

We focus on problems

input data

$$\mathbf{x}^* \triangleq \arg\min_{\mathbf{x}} \sum_{i=1}^{N} f_i(\mathbf{y}; \mathbf{x})$$

Distributed optimization

We focus on problems

input data

$$\mathbf{x}^* \triangleq \arg\min_{\mathbf{x}} \sum_{i=1}^{N} f_i(\mathbf{y}; \mathbf{x})$$

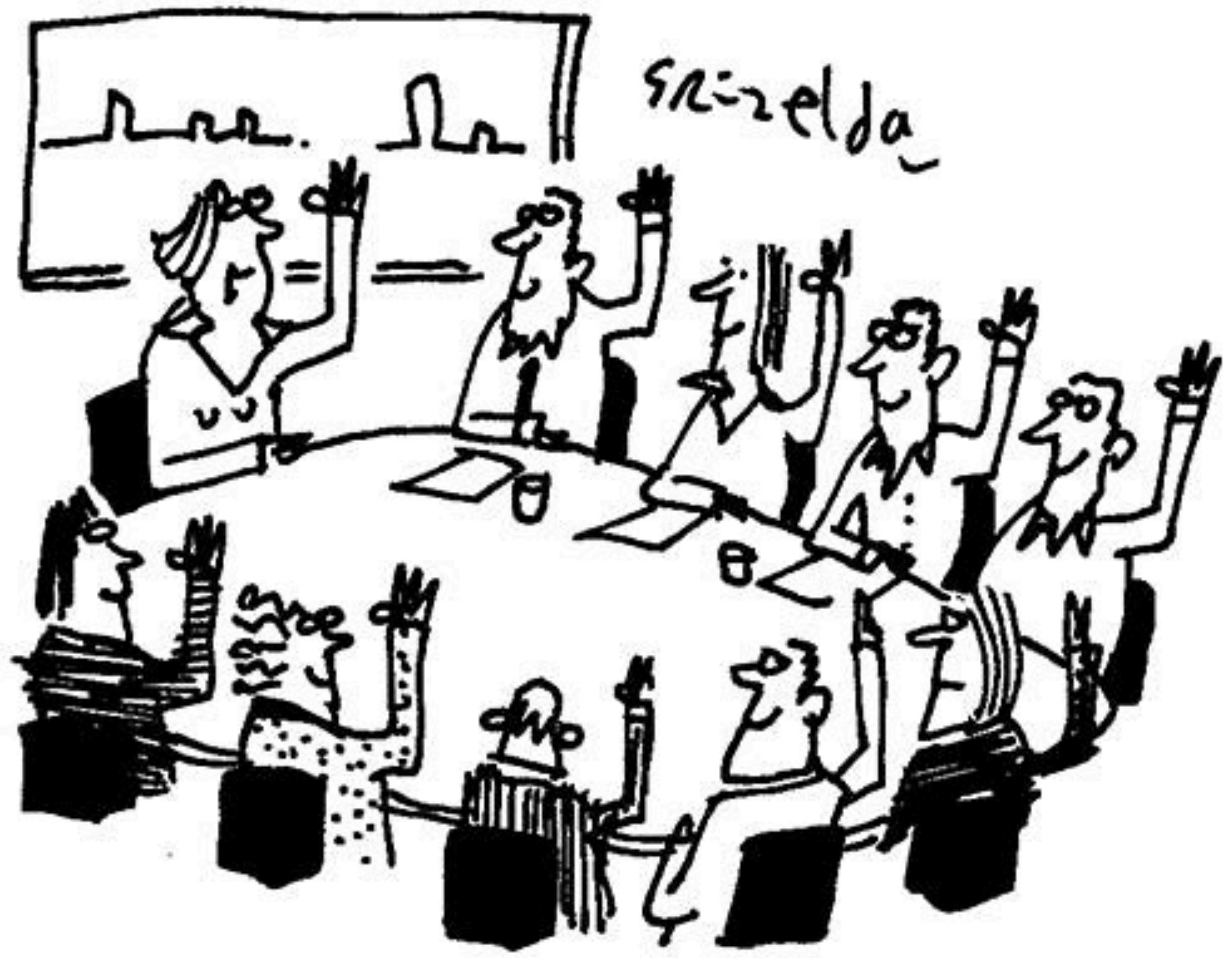
where

$$\mathbf{x}^* = \tilde{\mathbf{H}}\mathbf{y}$$

solution is a linear transformation
of the input data

Distributed optimization

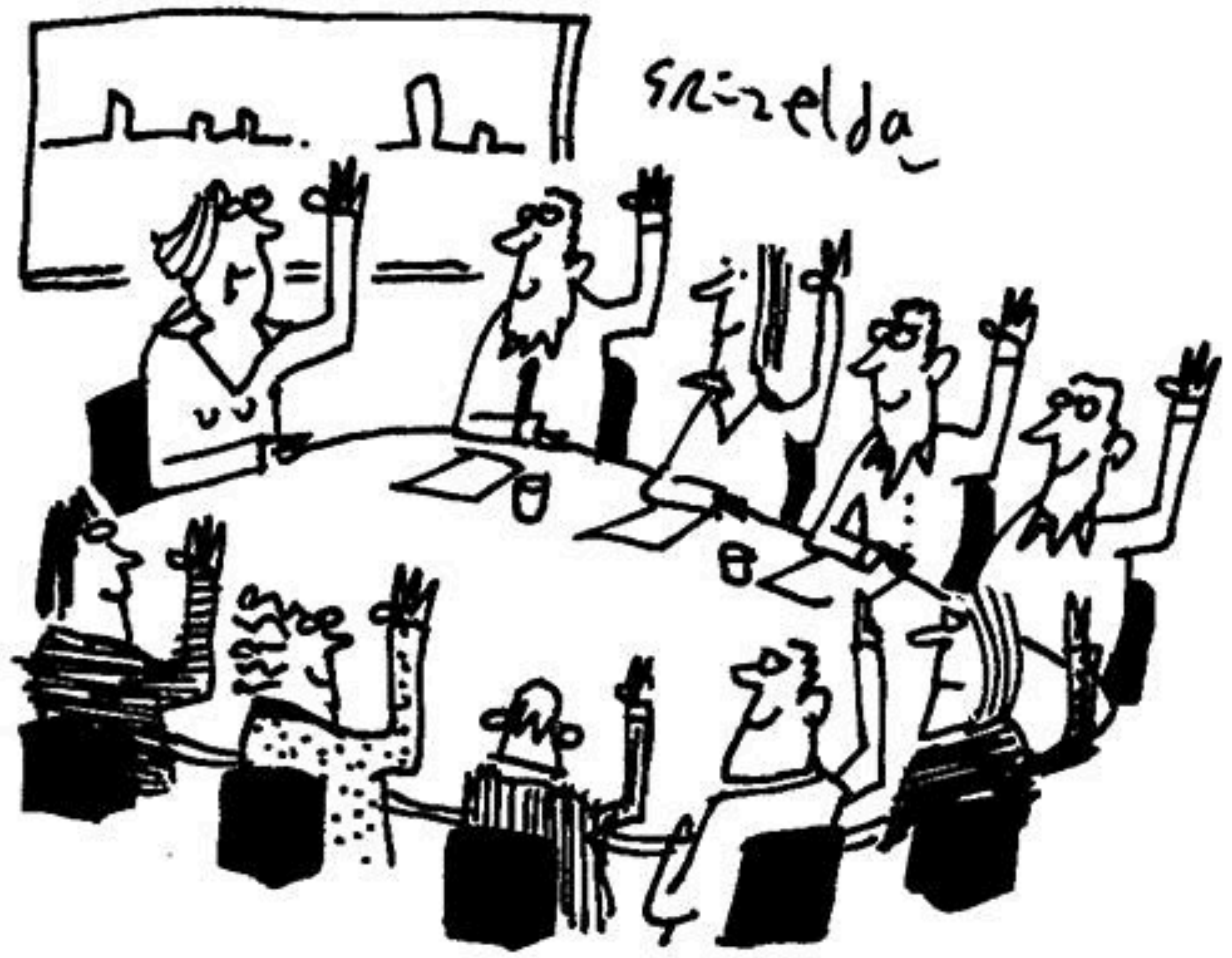
Average consensus



$$\min_{x} \sum_{i} (y_i - x)^2$$

Distributed optimization

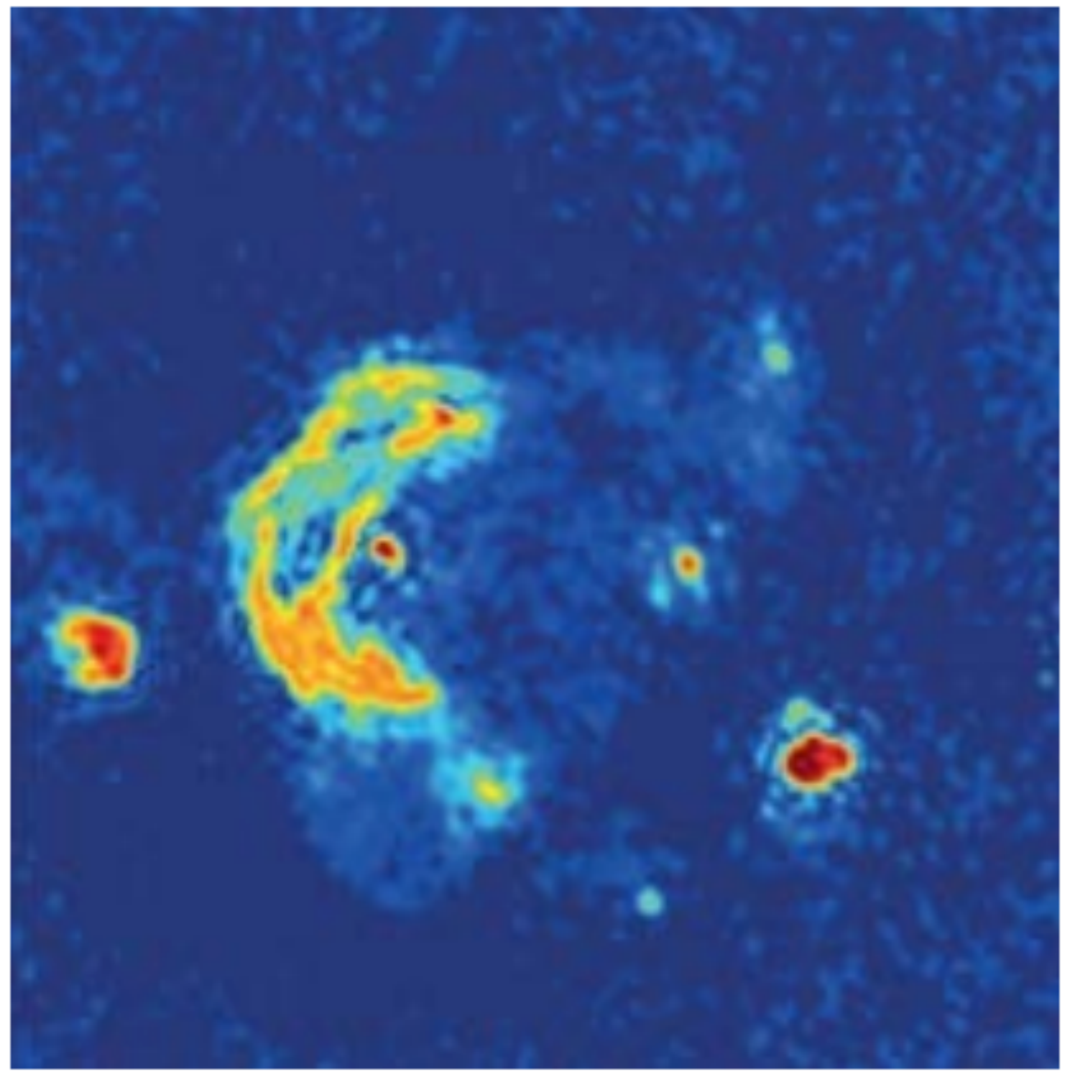
Average consensus



$$\min_x \sum_i (y_i - x)^2$$

Distributed imaging

[Naghibzadeh, ’19]



$$\min_x \sum_i (y_i - \mathbf{g}_i^\top \mathbf{x})^2$$

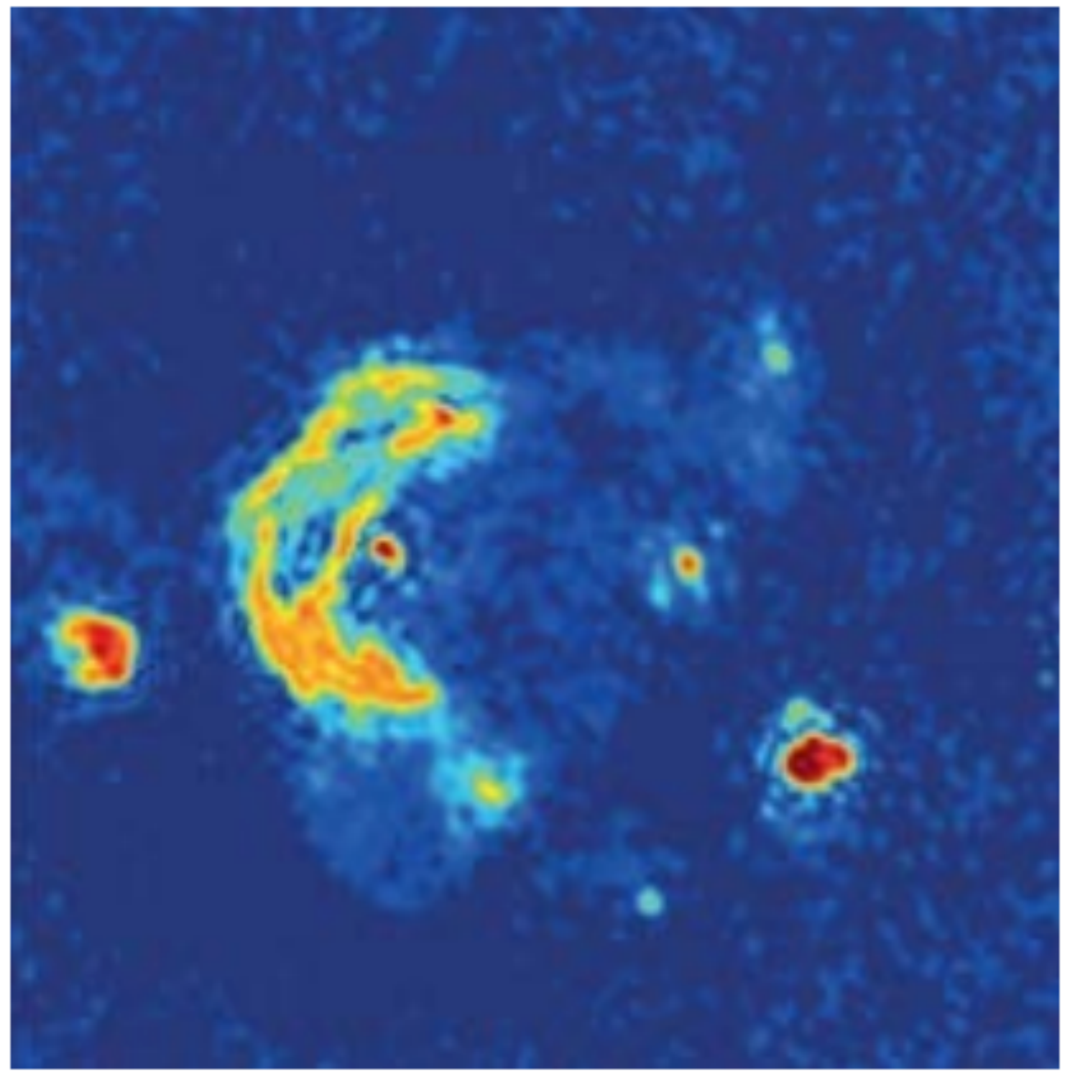
Distributed optimization

Average consensus

$$\min_x \sum_i (y_i - x)^2$$

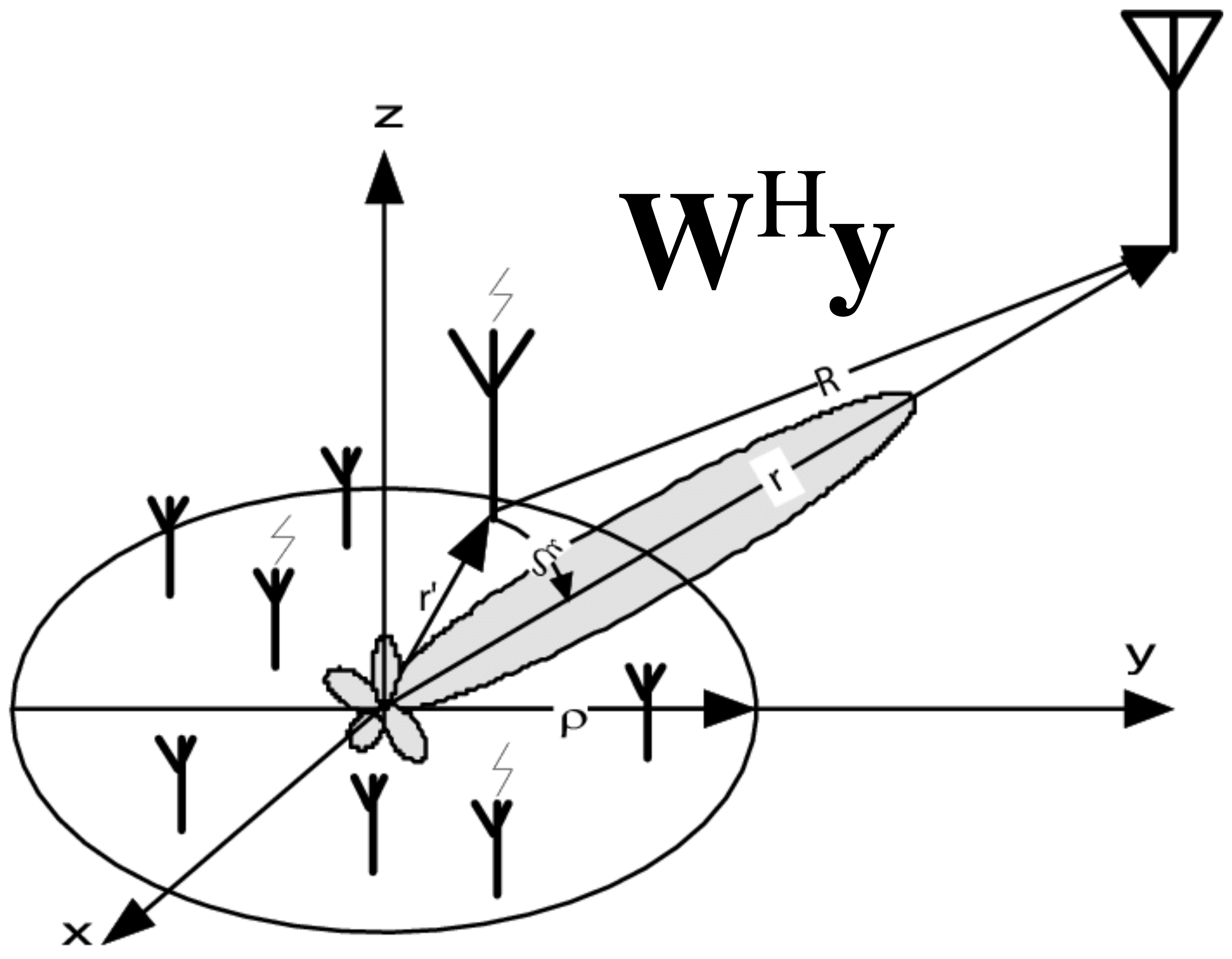
Distributed imaging

[Naghibzadeh, '19]



$$\min_x \sum_i (y_i - \mathbf{g}_i^\top \mathbf{x})^2$$

Distributed Beamforming



$$\min_{\mathbf{x}} \|\mathbf{y} - \left(\mathbf{W}^{\mathrm{H}}\right)^{\dagger}\mathbf{x}\|_2^2$$

How to leverage graph filters for distributed optimization?

Connection with graph filters

Goal: Implement known operation

$$\mathbf{x}^* = \tilde{\mathbf{H}}\mathbf{y}$$

Connection with graph filters

Goal: Implement known operation

$$\mathbf{x}^* = \tilde{\mathbf{H}}\mathbf{y}$$

in a distributed manner.

Connection with graph filters

Goal: Implement known operation

$$\mathbf{x}^* = \tilde{\mathbf{H}}\mathbf{y}$$

in a distributed manner.

distributable
by nature

Approach: Approximate $\tilde{\mathbf{H}}$ by means of graph filters

Global operator fitting

$$\min_{\boldsymbol{\Theta}} \|\tilde{\mathbf{H}} - \mathbf{H}_{\text{fit}}(\boldsymbol{\Theta})\|^2$$

where $\tilde{\mathbf{H}}$ is the solution of a centralized optimization problem

Global operator fitting

$$\min_{\boldsymbol{\Theta}} \|\tilde{\mathbf{H}} - \mathbf{H}_{\text{fit}}(\boldsymbol{\Theta})\|^2$$

where $\tilde{\mathbf{H}}$ is the solution of a centralized optimization problem

- ◉ Only works if global solution is linear (quadratic problems)

Global operator fitting

$$\min_{\boldsymbol{\Theta}} \|\tilde{\mathbf{H}} - \mathbf{H}_{\text{fit}}(\boldsymbol{\Theta})\|^2$$

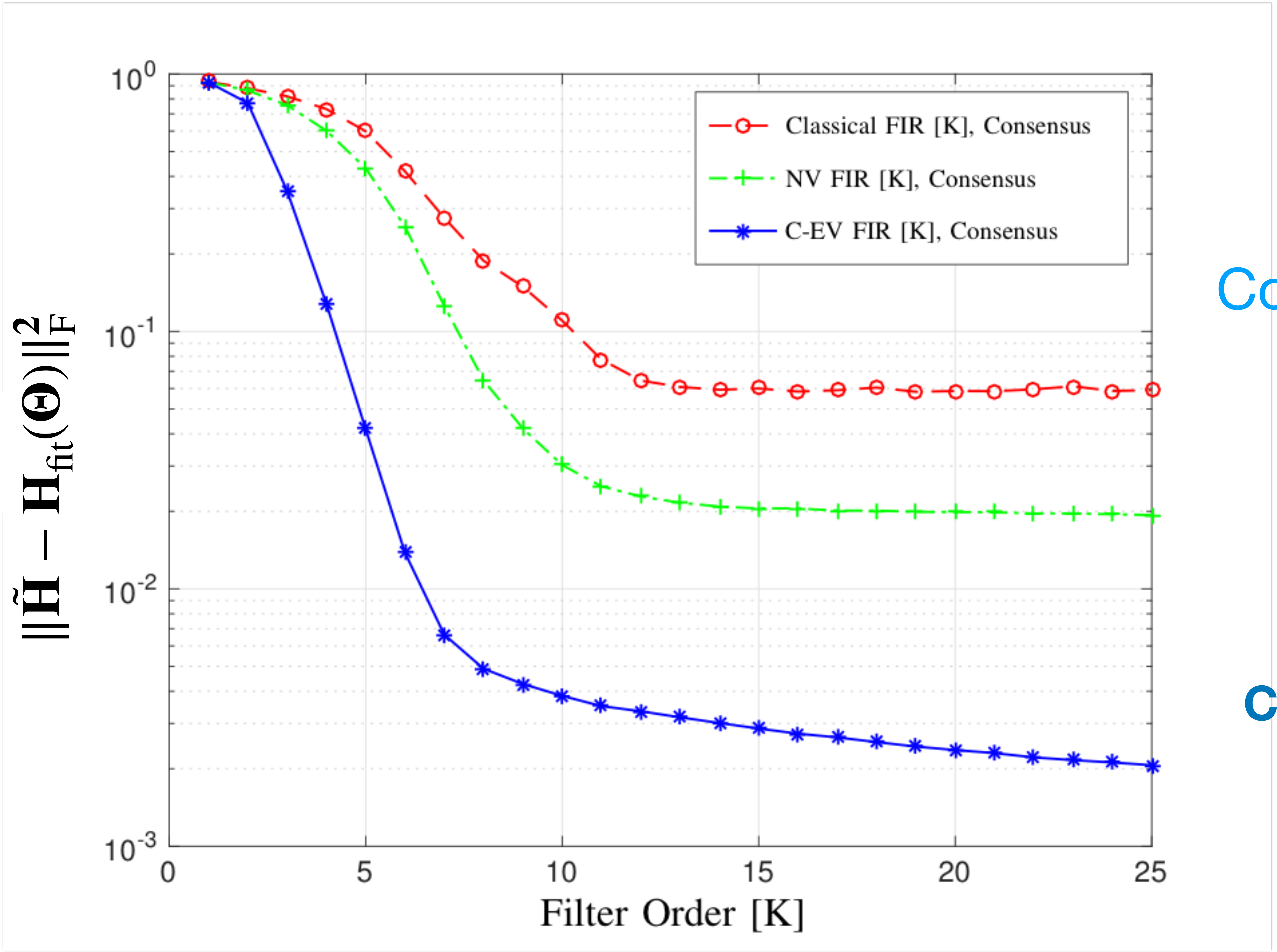
where $\tilde{\mathbf{H}}$ is the solution of a centralized optimization problem

- Only works if global solution is linear (quadratic problems)
- Two possible design approaches
  - A fusion centre designs the filter and distributes the coefficients
  - The nodes themselves carry out the filter design which requires knowledge of the global operator and the total graph

Coutino, Isufi, Leus, *Advances in Distributed Graph Filtering*, IEEE Transactions on Signal Processing, 2019

Some applications

Average consensus



Consensus matrix

$$\tilde{\mathbf{H}} = \mathbf{1}\mathbf{1}^\top / N$$

$$N = 256$$

Community graph

Distributed imaging

$$\min_{\mathbf{x}} ||\mathbf{y} - \mathbf{G}\mathbf{x}||_2^2$$

Distributed imaging

$$\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{G}\mathbf{x}\|_2^2$$

Distributed optimization approach

$$\min_{\mathbf{x}} \sum |[\mathbf{y}]_i - \mathbf{g}_i^\top \mathbf{x}|^2$$

$$\mathbf{G}^\top = [\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_N]$$

Distributed imaging

$$\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{G}\mathbf{x}\|_2^2$$

Distributed optimization approach

$$\min_{\mathbf{x}} \sum |[\mathbf{y}]_i - \mathbf{g}_i^\top \mathbf{x}|^2$$

$$\mathbf{G}^\top = [\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_N]$$

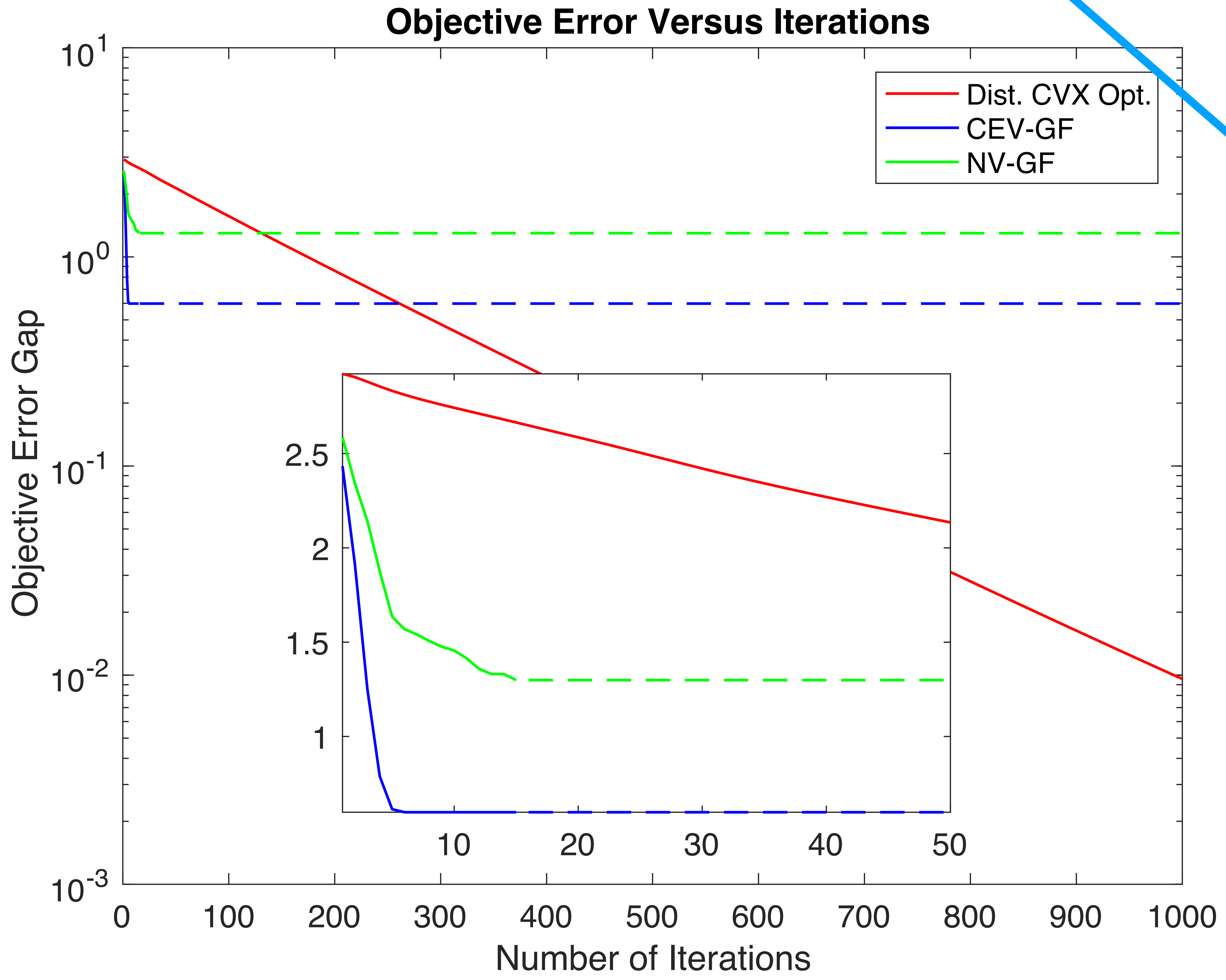
Graph filtering approach

$$\min_{\boldsymbol{\Theta}} \|\mathbf{1}\tilde{\mathbf{g}}_i^\top - \mathbf{H}_i(\boldsymbol{\Theta})\|_{\mathrm{F}}^2$$

$$(\mathbf{G}^\dagger)^\top = [\tilde{\mathbf{g}}_1, \tilde{\mathbf{g}}_2, \ldots, \tilde{\mathbf{g}}_d]$$

Distributed imaging

$$\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{G}\mathbf{x}\|_2^2$$



Distributed optimization approach

$$\min_{\mathbf{x}} \sum |[\mathbf{y}]_i - \mathbf{g}_i^\top \mathbf{x}|^2$$

$$\mathbf{G}^\top = [\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_N]$$

Graph filtering approach

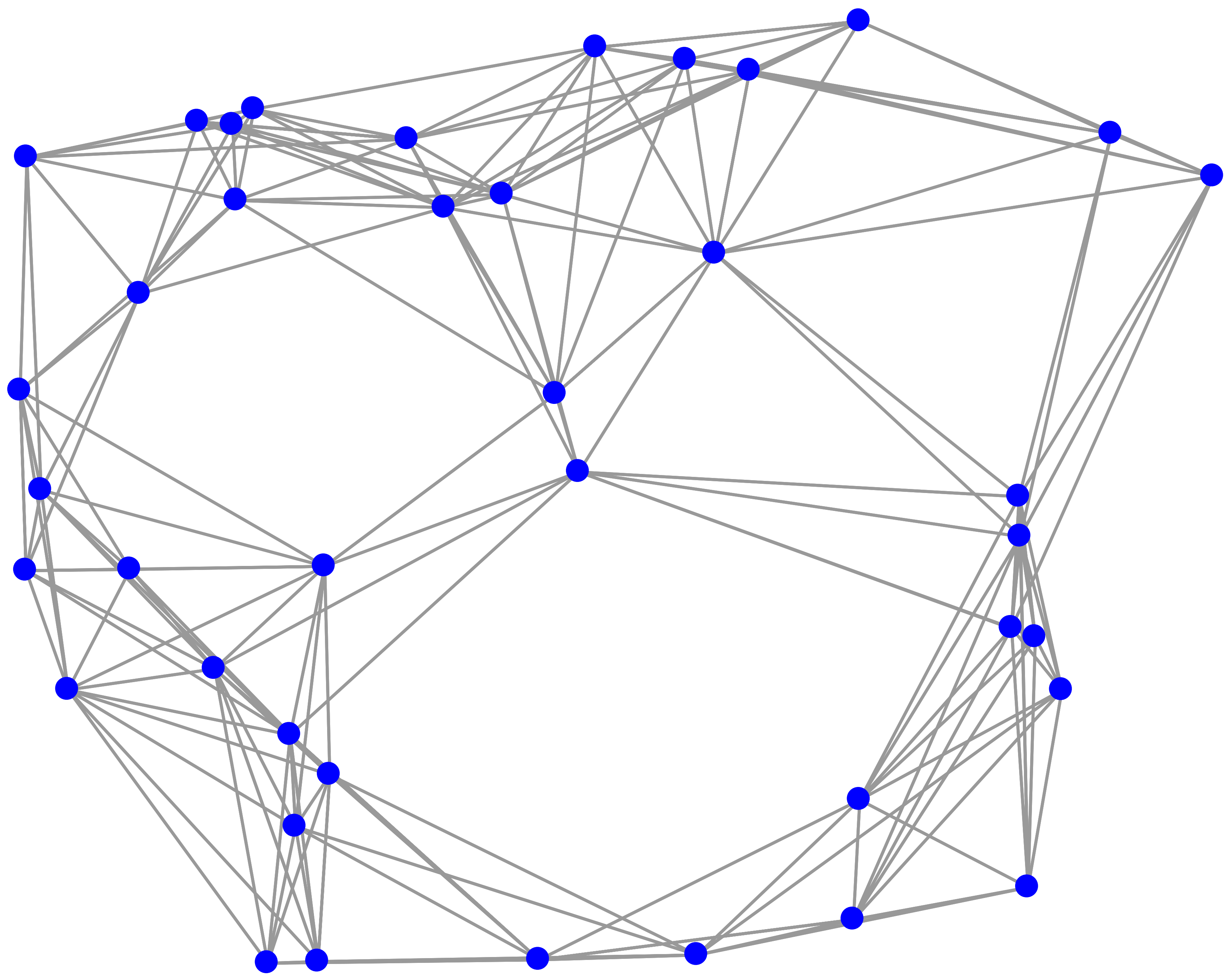
$$\min_{\boldsymbol{\Theta}} \|\mathbf{1}\tilde{\mathbf{g}}_i^\top - \mathbf{H}_i(\boldsymbol{\Theta})\|_F^2$$

$$(\mathbf{G}^\dagger)^\top = [\tilde{\mathbf{g}}_1, \tilde{\mathbf{g}}_2, \ldots, \tilde{\mathbf{g}}_d]$$

Distributed beamforming

Applying beaforming matrix

$$\mathbf{x} = \mathbf{W}^{\mathrm{H}}\mathbf{y}$$

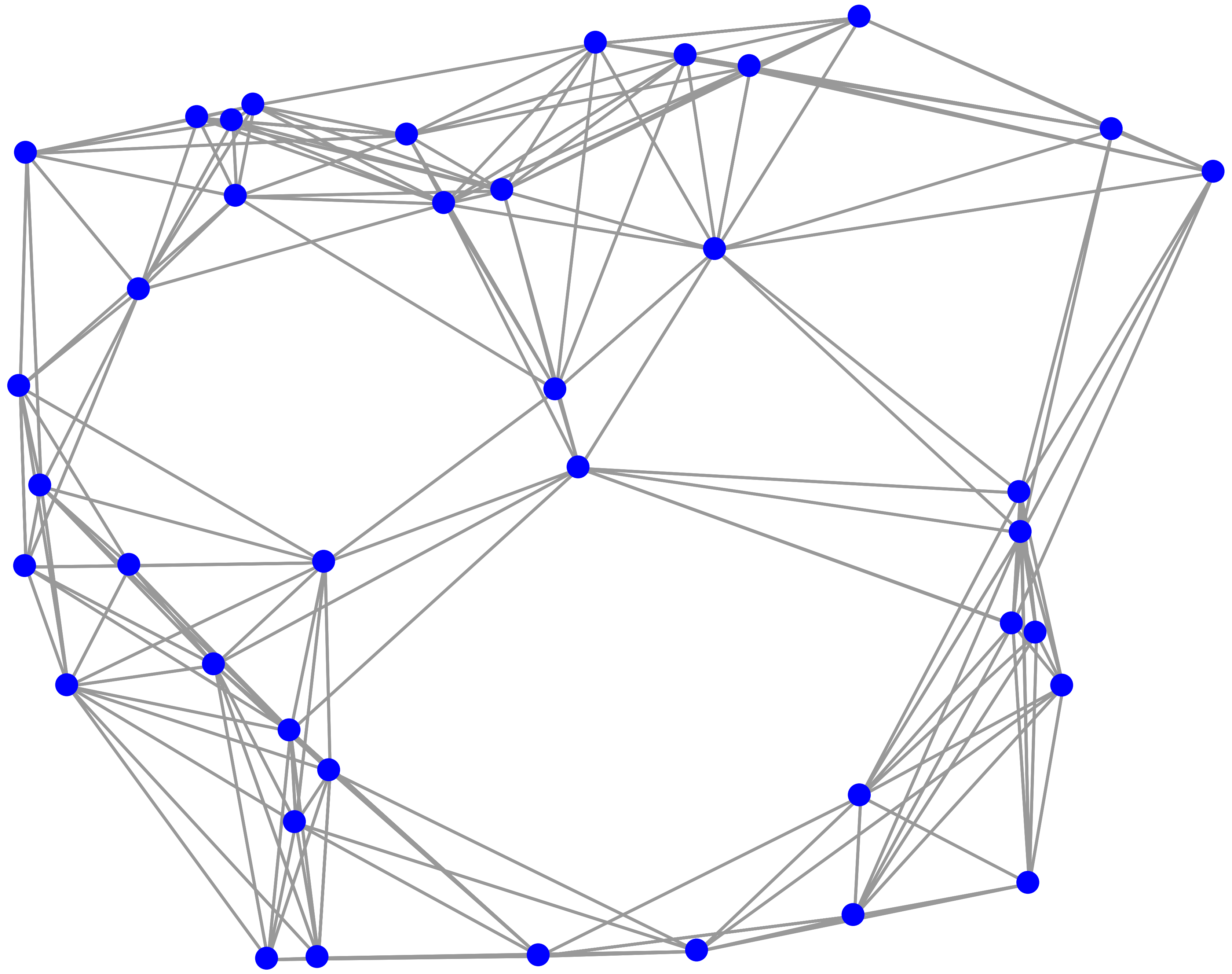


sensor array

Distributed beamforming

Applying beaforming matrix

$$\mathbf{x} = \mathbf{W}^{\mathrm{H}}\mathbf{y}$$

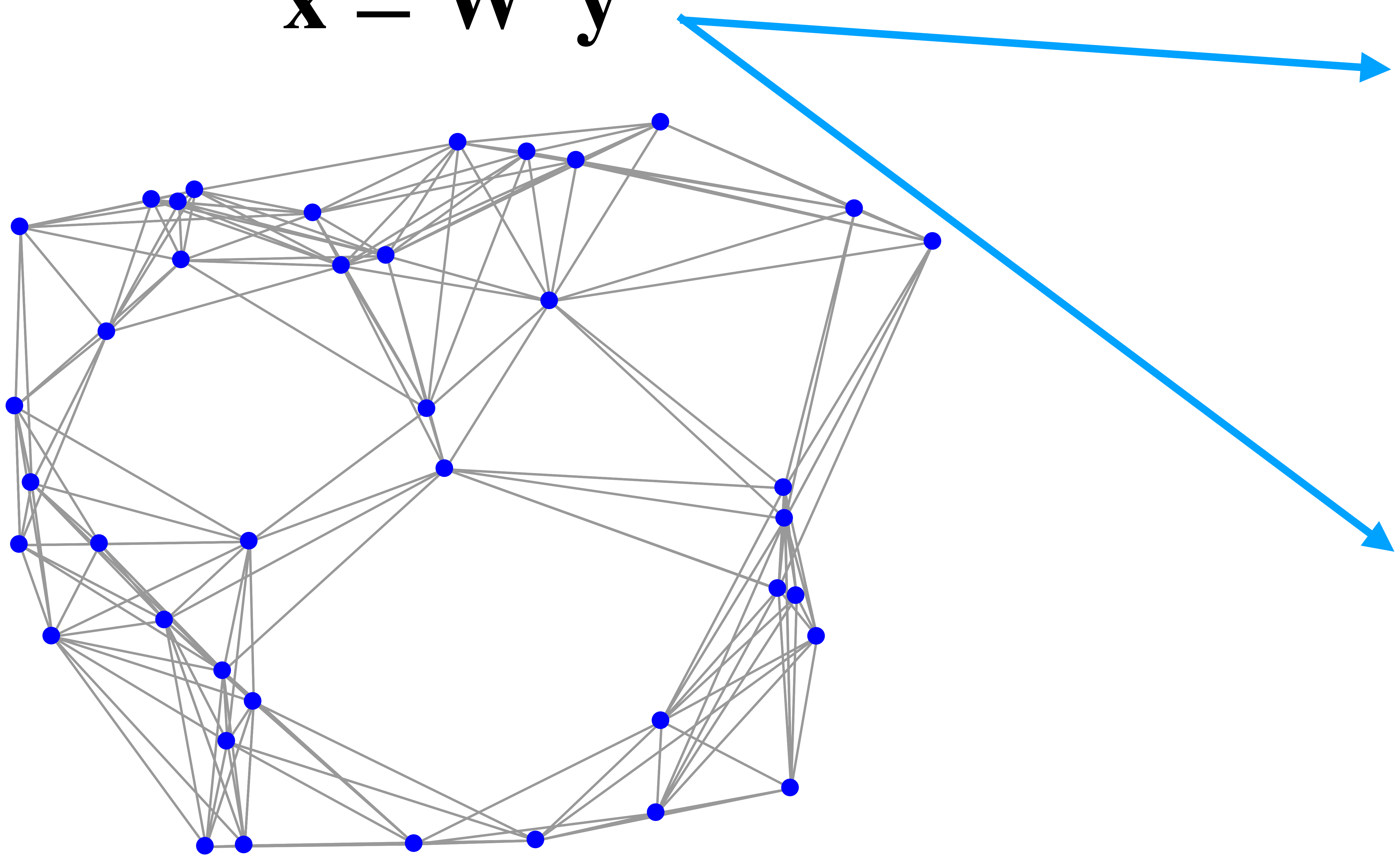


sensor array

Distributed optimization approach

$$\min_{\mathbf{x}} \|\mathbf{y} - (\mathbf{W}^{\mathrm{H}})^{\dagger}\mathbf{x}\|_2^2$$

Distributed beamforming



Applying beaforming matrix

$$\mathbf{x} = \mathbf{W}^{\mathrm{H}}\mathbf{y}$$

sensor array

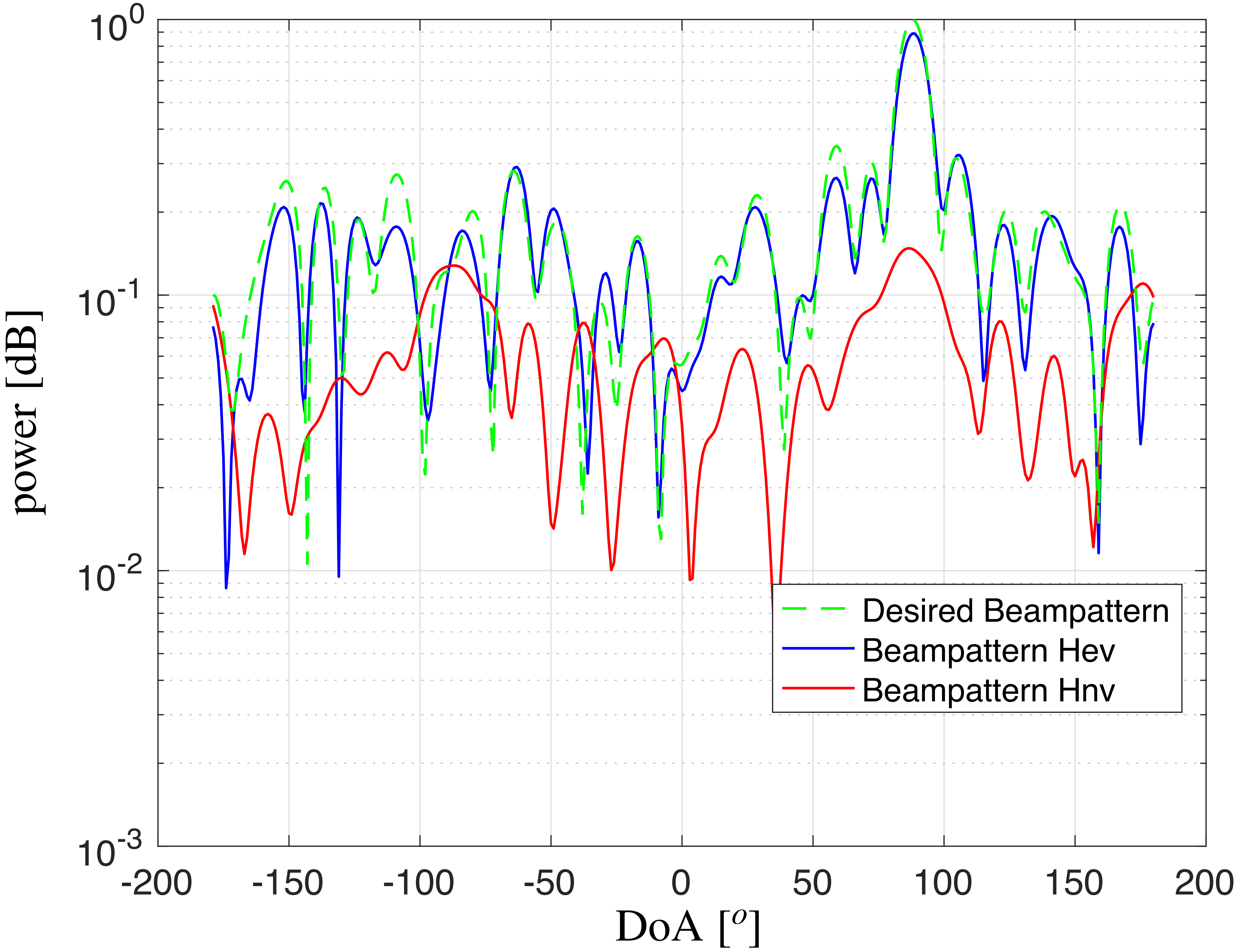
Distributed optimization approach

$$\min_{\mathbf{x}} \|\mathbf{y} - (\mathbf{W}^{\mathrm{H}})^{\dagger}\mathbf{x}\|_2^2$$

Graph filtering approach

$$\min_{\boldsymbol{\Theta}} \|\mathbf{W}^{\mathrm{H}} - \mathbf{H}(\boldsymbol{\Theta})\|_{\mathrm{F}}$$

Distributed beamforming



intermedio

part 3
graph filters for distributed optimization [2]

part 3 :: overview

part 3 :: overview

- ◉ Asynchronous implementation
  - Classical results
  - Results for classical graph filters
  - Extension to more advanced graph filters
  - Results

part 3 :: overview

- ◉ Asynchronous implementation
  - Classical results
  - Results for classical graph filters
  - Extension to more advanced graph filters
  - Results
- ◉ Cascaded implementation
  - Motivation
  - Cascaded problem formulation
  - Right-left iterative fitting (RELIEF)
  - Results for average consensus

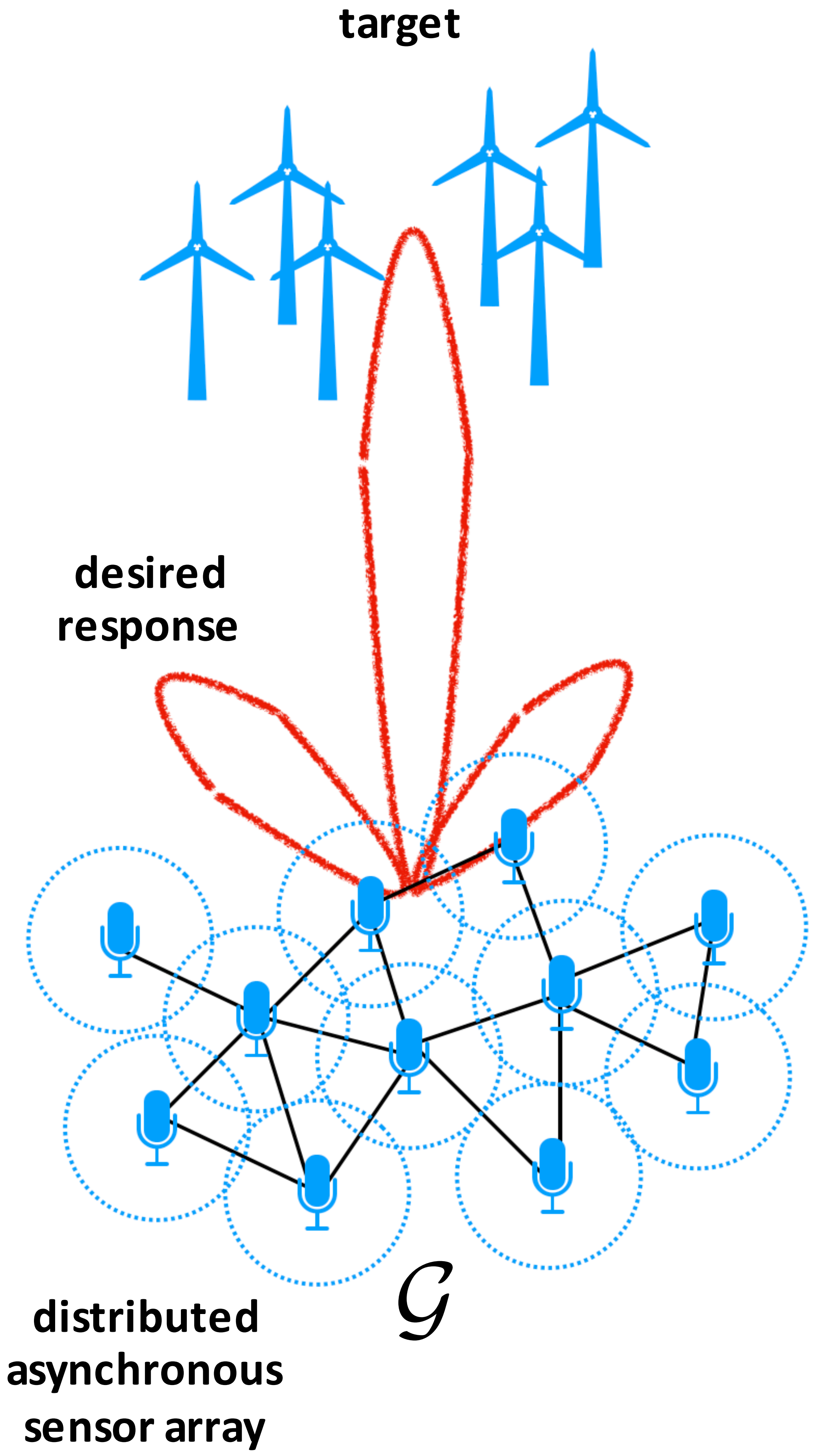
What about unsynchronized networks?

Asynchronous graph filtering

- Many applications require to compute, e.g., the beamforming

$$\mathbf{y} = \mathbf{W}^{\mathrm{H}}\mathbf{x} \approx \mathbf{H}(\boldsymbol{\Theta})\mathbf{x}$$

distributively over an unsynchronised network.



Asynchronous graph filtering

- Many applications require to compute, e.g., the beamforming

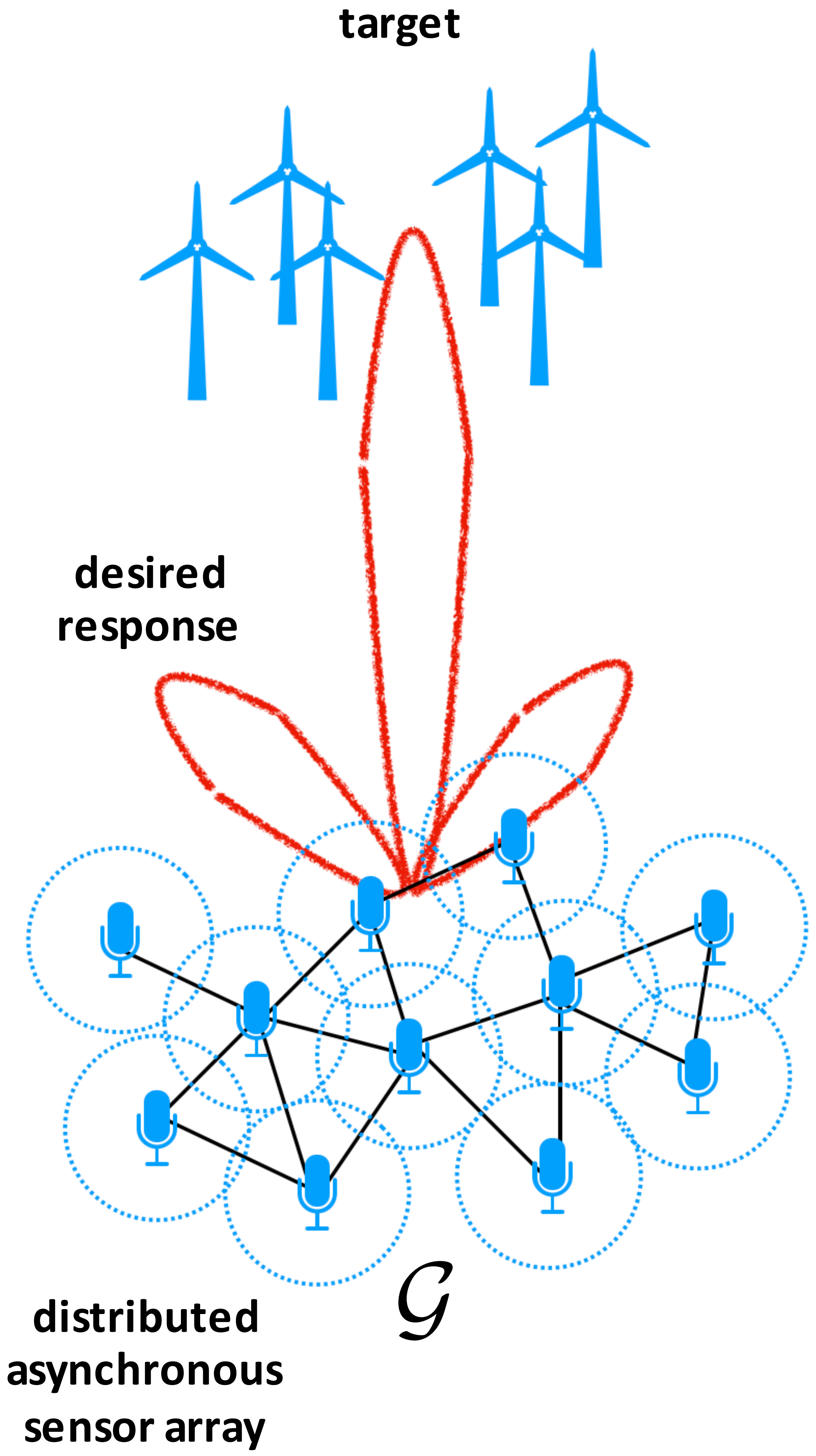
$$\mathbf{y} = \mathbf{W}^{\mathrm{H}}\mathbf{x} \approx \mathbf{H}(\boldsymbol{\Theta})\mathbf{x}$$

distributively over an unsynchronised network.

- Suppose we want to use general graph filter operations

$$\mathbf{y}_{\mathrm{GF}} = \mathbf{H}(\boldsymbol{\Theta})\mathbf{x} = \mathbf{H}_{\mathrm{B}}\mathbf{H}_{\mathrm{A}}$$

$$\mathbf{H}_B \triangleq \mathbf{I} - \sum_{k=1}^{Q} \boldsymbol{\Phi}_{k,b}\mathbf{S}^{k-1} \qquad \mathbf{H}_A \triangleq \left(\mathbf{I} - \sum_{k=1}^{K} \boldsymbol{\Phi}_{k,a}\mathbf{S}^{k-1}\right)^{-1}$$



Asynchronous graph filtering

- Many applications require to compute, e.g., the beamforming

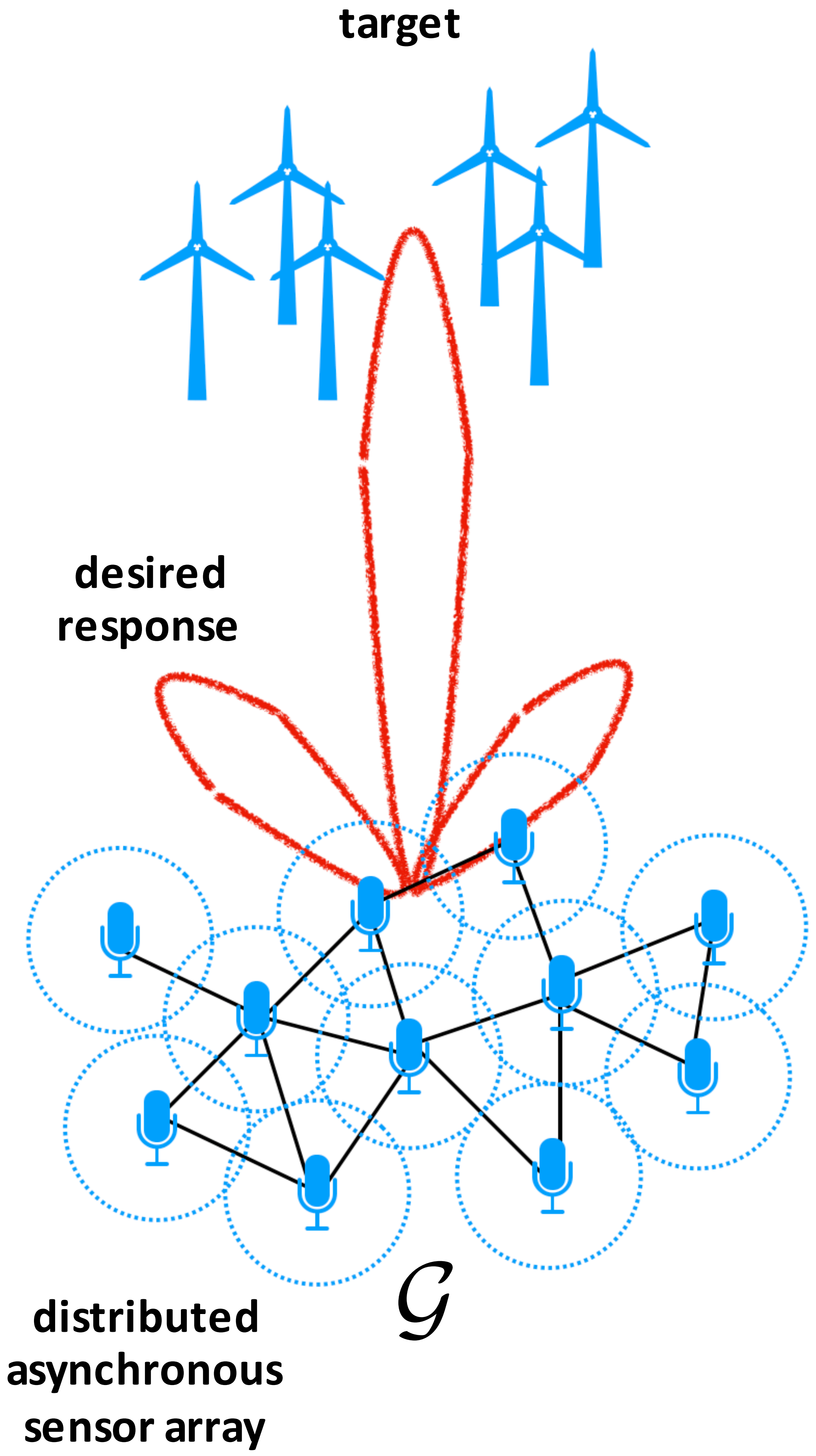
$$\mathbf{y} = \mathbf{W}^{\mathrm{H}}\mathbf{x} \approx \mathbf{H}(\boldsymbol{\Theta})\mathbf{x}$$

distributively over an unsynchronised network.

- Suppose we want to use general graph filter operations

$$\mathbf{y}_{\mathrm{GF}} = \mathbf{H}(\boldsymbol{\Theta})\mathbf{x} = \mathbf{H}_{\mathrm{B}}\mathbf{H}_{\mathrm{A}}$$

$$\mathbf{H}_B \triangleq \mathbf{I} - \sum_{k=1}^{Q} \boldsymbol{\Phi}_{k,b}\mathbf{S}^{k-1} \qquad \mathbf{H}_A \triangleq \left(\mathbf{I} - \sum_{k=1}^{K} \boldsymbol{\Phi}_{k,a}\mathbf{S}^{k-1}\right)^{-1}$$



Under which conditions is possible to implement the operator in the network?

Asynchronous graph filtering

- **Classical results.** Asynchronous implementation of the recurrence

$$\mathbf{y}_{t+1} = \mathbf{y}_t + (\mathbf{x} - \mathbf{C}\mathbf{y}_t) \qquad \text{(splitting method)}$$

convergence under mild conditions on $\mathbf{C}$.

[D. Chazan,’69][D. Bertsekas,’83][Y.Saad,’03]

Asynchronous graph filtering

- **Classical results.** Asynchronous implementation of the recurrence

$$\mathbf{y}_{t+1} = \mathbf{y}_t + (\mathbf{x} - \mathbf{C}\mathbf{y}_t)$$

(splitting method)

convergence under mild conditions on $\mathbf{C}$.

[D. Chazan,'69][D. Bertsekas,'83][Y.Saad,'03]

- **Recent results for classical GF.** Asynchronous implementation of a GF

$$\mathbf{y} = p(\gamma\mathbf{S})(q(\gamma\mathbf{S}))^{-1}\mathbf{x}$$

$$p(x) = \sum_{k=0}^{K-1} p_k x^k$$

$$q(x) = 1 + \sum_{k=1}^{K} q_k x^k$$

convergence under mild conditions on matrices involved.

[O. Teke, '19]

Asynchronous graph filtering

- From the model

$$\mathbf{y}_{\mathrm{GF}} = \mathbf{H}(\boldsymbol{\Theta})\mathbf{x} = \mathbf{H}_{\mathrm{B}}\mathbf{H}_{\mathrm{A}}\mathbf{x}$$

Asynchronous graph filtering

- From the model

$$\mathbf{y}_{\mathrm{GF}} = \mathbf{H}(\boldsymbol{\Theta})\mathbf{x} = \mathbf{H}_{\mathrm{B}}\mathbf{H}_{\mathrm{A}}\mathbf{x}$$

consider the partial output of the graph filter operation

$$\mathbf{y}_{\mathrm{A}} = \mathbf{H}_{\mathrm{A}}\mathbf{x}$$

Asynchronous graph filtering

- From the model

$$\mathbf{y}_{\mathrm{GF}} = \mathbf{H}(\boldsymbol{\Theta})\mathbf{x} = \mathbf{H}_{\mathrm{B}}\mathbf{H}_{\mathrm{A}}\mathbf{x}$$

consider the partial output of the graph filter operation

$$\mathbf{y}_{\mathrm{A}} = \mathbf{H}_{\mathrm{A}}\mathbf{x}$$

which can be written as the solution to the linear system

$$\left(\mathbf{I} - \sum_{k=1}^{K} \boldsymbol{\Phi}_k \mathbf{S}^{k-1}\right)\mathbf{y}_{\mathrm{A}} = \mathbf{x}$$

Asynchronous graph filtering

This system can be solved with the recursion

$$\mathbf{y}_{t+1} = \mathbf{x} + \mathbf{B}\mathbf{y}_t$$

Asynchronous graph filtering

This system can be solved with the recursion

$$\mathbf{y}_{t+1} = \mathbf{x} + \mathbf{B}\mathbf{y}_t$$

which converges to $\mathbf{y}_\infty \to \mathbf{H}_\mathrm{A}\mathbf{x}$ if

$$\rho(\mathbf{B}) < 1$$

Asynchronous graph filtering

This system can be solved with the recursion

$$\mathbf{y}_{t+1} = \mathbf{x} + \mathbf{B}\mathbf{y}_t$$

which converges to $\mathbf{y}_\infty \rightarrow \mathbf{H}_\mathrm{A}\mathbf{x}$ if

$$\rho(\mathbf{B}) < 1$$

Requires several exchanges before update!

However, $\mathbf{B} \triangleq \sum_{k=0}^{K} \mathbf{\Phi}_k \mathbf{S}^k$ is not suitable for asynchronous operation.

Asynchronous graph filtering

Instead, consider the kth shift of the recurrence vector

$$\mathbf{y}_t^{(k)} = \mathbf{S}\mathbf{y}_t^{(k-1)}$$

Asynchronous graph filtering

Instead, consider the kth shift of the recurrence vector

$$\mathbf{y}_t^{(k)} = \mathbf{S}\mathbf{y}_t^{(k-1)}$$

stacking the vectors $\{\mathbf{y}_t^{(k)}\}_{k=0}^{K-1}$ in $\bar{\mathbf{y}}_t$ we obtain the extended recurrence equation

$$\begin{aligned}\bar{\mathbf{y}}_{t+1} &= \begin{bmatrix}\mathbf{x}\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\end{bmatrix} + \begin{bmatrix}\mathbf{\Phi}_1 & \mathbf{\Phi}_2 & \cdots & & \mathbf{\Phi}_K\\ \mathbf{S} & & & & \mathbf{0}\\ & \ddots & & & \vdots\\ & & & \mathbf{S} & \mathbf{0}\end{bmatrix}\bar{\mathbf{y}}_{t+1}\\ &= \bar{\mathbf{x}} + \bar{\mathbf{B}}\,\bar{\mathbf{y}}_{t+1}\end{aligned}$$

Analogous to companion matrix for **linear recursive sequences**

Asynchronous graph filtering

Instead, consider the kth shift of the recurrence vector

$$\mathbf{y}_t^{(k)} = \mathbf{S}\mathbf{y}_t^{(k-1)}$$

stacking the vectors $\{\mathbf{y}_t^{(k)}\}_{k=0}^{K-1}$ in $\bar{\mathbf{y}}_t$ we obtain the extended recurrence equation

$$\begin{aligned}\bar{\mathbf{y}}_{t+1} &= \begin{bmatrix}\mathbf{x}\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\end{bmatrix} + \begin{bmatrix}\boldsymbol{\Phi}_1 & \boldsymbol{\Phi}_2 & \cdots & \boldsymbol{\Phi}_K\\ \mathbf{S} & & & \mathbf{0}\\ & \ddots & & \vdots\\ & & \mathbf{S} & \mathbf{0}\end{bmatrix}\bar{\mathbf{y}}_{t+1}\\ &= \bar{\mathbf{x}} + \bar{\mathbf{B}}\,\bar{\mathbf{y}}_{t+1}\end{aligned}$$

the recurrence asymptotically converges if $\rho(\bar{\mathbf{B}}) < 1.$

Asynchronous graph filtering

- For the inexact synchronous recurrence

$$\bar{\mathbf{y}}^{\text{in}}_{t+1} = \bar{\mathbf{x}} + \bar{\mathbf{B}}\bar{\mathbf{y}}_t + \mathbf{v}_t$$

Asynchronous graph filtering

- For the inexact synchronous recurrence

$$\bar{\mathbf{y}}_{t+1}^{\text{in}} = \bar{\mathbf{x}} + \bar{\mathbf{B}}\bar{\mathbf{y}}_t + \mathbf{v}_t$$

bounded
perturbations, e.g.,
fixed-precision

$$\|\mathbf{v}_l\| \leq \beta, \forall\, l \in \mathbb{N}$$

Asynchronous graph filtering

- For the inexact synchronous recurrence

$$\bar{\mathbf{y}}_{t+1}^{\text{in}} = \bar{\mathbf{x}} + \bar{\mathbf{B}}\bar{\mathbf{y}}_t + \mathbf{v}_t$$

bounded perturbations, e.g., fixed-precision

$$\|\mathbf{v}_l\| \leq \beta, \forall\, l \in \mathbb{N}$$

under conditions

$$C \quad \rho \leq 1, \qquad \gamma := a\frac{1-\rho^K}{1-\rho} < 1 \qquad a := \max_{k\in\{1,\ldots,K\}} \|\mathbf{\Phi}_k\|$$

Asynchronous graph filtering

- For the inexact synchronous recurrence

$$\bar{\mathbf{y}}^{\text{in}}_{t+1} = \bar{\mathbf{x}} + \bar{\mathbf{B}}\bar{\mathbf{y}}_t + \mathbf{v}_t$$

bounded perturbations, e.g., fixed-precision

$$\|\mathbf{v}_l\| \leq \beta, \forall\, l \in \mathbb{N}$$

under conditions

$$C \quad \rho \leq 1, \qquad \gamma := a\frac{1-\rho^K}{1-\rho} < 1 \qquad a := \max_{k\in\{1,\ldots,K\}} \|\mathbf{\Phi}_k\|$$

$$\Rightarrow \quad \boxed{\lim_{l\to\infty} \|(\bar{\mathbf{y}}^{\text{in}}_l)^{(0)} - \mathbf{y}_{\text{A}}\| \leq \frac{\beta}{1-\gamma}}$$

recurrence asymptotically converges within a norm ball

Asynchronous graph filtering

- For the inexact synchronous recurrence

$$\bar{\mathbf{y}}^{\text{in}}_{t+1} = \bar{\mathbf{x}} + \bar{\mathbf{B}}\bar{\mathbf{y}}_t + \mathbf{v}_t$$

bounded perturbations, e.g., fixed-precision

$$\|\mathbf{v}_l\| \leq \beta, \forall\, l \in \mathbb{N}$$

under conditions

$$C \quad \rho \leq 1, \qquad \gamma := a\frac{1-\rho^K}{1-\rho} < 1 \qquad a := \max_{k\in\{1,\ldots,K\}} \|\mathbf{\Phi}_k\|$$

⇒

$$\boxed{\lim_{l\to\infty} \|(\bar{\mathbf{y}}^{\text{in}}_l)^{(0)} - \mathbf{y}_{\text{A}}\| \leq \frac{\beta}{1-\gamma}}$$

exact convergence in noise-free case

recurrence asymptotically converges within a norm ball

Asynchronous graph filtering

- For the asynchronous recurrence

$$\bar{\mathbf{y}}^{\mathrm{a}}_{t+1} = [\mathbf{Z}_t\bar{\mathbf{x}} + \mathbf{Z}_t\bar{\mathbf{B}}\bar{\mathbf{y}}^{\mathrm{a}}_t] + [(\mathbf{I} - \mathbf{Z}_t)\bar{\mathbf{y}}^{\mathrm{a}}_t]$$

(updated entries) (unchanged entries)

Asynchronous graph filtering

- For the asynchronous recurrence

$$\bar{\mathbf{y}}^{\mathrm{a}}_{t+1} = [\mathbf{Z}_t\bar{\mathbf{x}} + \mathbf{Z}_t\bar{\mathbf{B}}\bar{\mathbf{y}}^{\mathrm{a}}_t] + [(\mathbf{I} - \mathbf{Z}_t)\bar{\mathbf{y}}^{\mathrm{a}}_t]$$

(updated entries) (unchanged entries)

$$\mathbf{Z}_t \triangleq \mathrm{diag}(\mathbf{w}_t) \in \{0,1\}^{KN\times KN}$$

(update-selection matrix)

Asynchronous graph filtering

- For the asynchronous recurrence

$$\bar{\mathbf{y}}^{\mathrm{a}}_{t+1} = [\mathbf{Z}_t\bar{\mathbf{x}} + \mathbf{Z}_t\bar{\mathbf{B}}\bar{\mathbf{y}}^{\mathrm{a}}_t] + [(\mathbf{I} - \mathbf{Z}_t)\bar{\mathbf{y}}^{\mathrm{a}}_t]$$

(updated entries) (unchanged entries)

$$\mathbf{Z}_t \triangleq \mathrm{diag}(\mathbf{w}_t) \in \{0,1\}^{KN\times KN}$$

(update-selection matrix)

$$\mathbf{w}_t = [(\mathbf{w}_t^{(0)})^\top, \ldots, (\mathbf{w}_t^{(K-1)})^\top]^\top$$

(kth shift update selection)

Asynchronous graph filtering

- For the asynchronous recurrence

$$\bar{\mathbf{y}}^{\mathrm{a}}_{t+1} = [\mathbf{Z}_t\bar{\mathbf{x}} + \mathbf{Z}_t\bar{\mathbf{B}}\bar{\mathbf{y}}^{\mathrm{a}}_t] + [(\mathbf{I} - \mathbf{Z}_t)\bar{\mathbf{y}}^{\mathrm{a}}_t]$$

(updated entries) (unchanged entries)

not all memory entries are updated

$$\mathbf{Z}_t \triangleq \mathrm{diag}(\mathbf{w}_t) \in \{0,1\}^{KN\times KN}$$

(update-selection matrix)

$$\mathbf{w}_t = [(\mathbf{w}_t^{(0)})^\top, \ldots, (\mathbf{w}_t^{(K-1)})^\top]^\top$$

(kth shift update selection)

Asynchronous graph filtering

- For the asynchronous recurrence

$$\bar{\mathbf{y}}^{\mathrm{a}}_{t+1} = [\mathbf{Z}_t\bar{\mathbf{x}} + \mathbf{Z}_t\bar{\mathbf{B}}\bar{\mathbf{y}}^{\mathrm{a}}_t] + [(\mathbf{I} - \mathbf{Z}_t)\bar{\mathbf{y}}^{\mathrm{a}}_t]$$

(updated entries) (unchanged entries)

not all memory entries are updated

$$\mathbf{Z}_t \triangleq \mathrm{diag}(\mathbf{w}_t) \in \{0,1\}^{KN\times KN}$$

(update-selection matrix)

$$\mathbf{w}_t = [(\mathbf{w}_t^{(0)})^\top, \ldots, (\mathbf{w}_t^{(K-1)})^\top]^\top$$

(kth shift update selection)

with

$$\sum_{t=0}^{\infty}[\mathbf{Z}_t]_{i,i} \gg 0 \quad \& \quad C$$

(sufficiently exciting condition)

Asynchronous graph filtering

- For the asynchronous recurrence

$$\bar{\mathbf{y}}^{\mathrm{a}}_{t+1} = [\mathbf{Z}_t\bar{\mathbf{x}} + \mathbf{Z}_t\bar{\mathbf{B}}\bar{\mathbf{y}}^{\mathrm{a}}_t] + [(\mathbf{I} - \mathbf{Z}_t)\bar{\mathbf{y}}^{\mathrm{a}}_t]$$

(updated entries) (unchanged entries)

not all memory entries are updated

$$\mathbf{Z}_t \triangleq \mathrm{diag}(\mathbf{w}_t) \in \{0,1\}^{KN \times KN}$$

(update-selection matrix)

$$\mathbf{w}_t = [(\mathbf{w}_t^{(0)})^\top, \ldots, (\mathbf{w}_t^{(K-1)})^\top]^\top$$

(kth shift update selection)

with

$$\sum_{t=0}^{\infty} [\mathbf{Z}_t]_{i,i} \gg 0 \quad \& \quad C$$

(sufficiently exciting condition)

$$\boxed{\lim_{l \to \infty} (\bar{\mathbf{y}}^{\mathrm{a}}_l)^{(0)} = \mathbf{y}_{\mathrm{A}}}$$

recurrence asymptotically converges

Coutino, Leus, *Asynchronous Distributed Edge-Variant Graph Filters,* IEEE DSW, 2019

Asynchronous graph filtering

- Let a network perform the filtering operation

$$\mathbf{y}_{\mathrm{GF}} = \mathbf{H}(\boldsymbol{\Theta})\mathbf{x} = \mathbf{H}_{\mathrm{B}}\mathbf{H}_{\mathrm{A}}\mathbf{x}$$

Asynchronous graph filtering

- Let a network perform the filtering operation

$$\mathbf{y}_{\mathrm{GF}} = \mathbf{H}(\boldsymbol{\Theta})\mathbf{x} = \mathbf{H}_{\mathrm{B}}\mathbf{H}_{\mathrm{A}}\mathbf{x}$$

if conditions

$$\sum_{t=0}^{\infty} [\mathbf{Z}_t]_{i,i} \gg 0 \quad \& \quad C$$

are met for the matrices involved in $\mathbf{H}_{\mathrm{A}}$

Asynchronous graph filtering

- Let a network perform the filtering operation

$$\mathbf{y}_{\mathrm{GF}} = \mathbf{H}(\boldsymbol{\Theta})\mathbf{x} = \mathbf{H}_{\mathrm{B}}\mathbf{H}_{\mathrm{A}}\mathbf{x}$$

if conditions

$$\sum_{t=0}^{\infty} [\mathbf{Z}_t]_{i,i} \gg 0 \quad \& \quad C$$

are met for the matrices involved in $\mathbf{H}_{\mathrm{A}}$

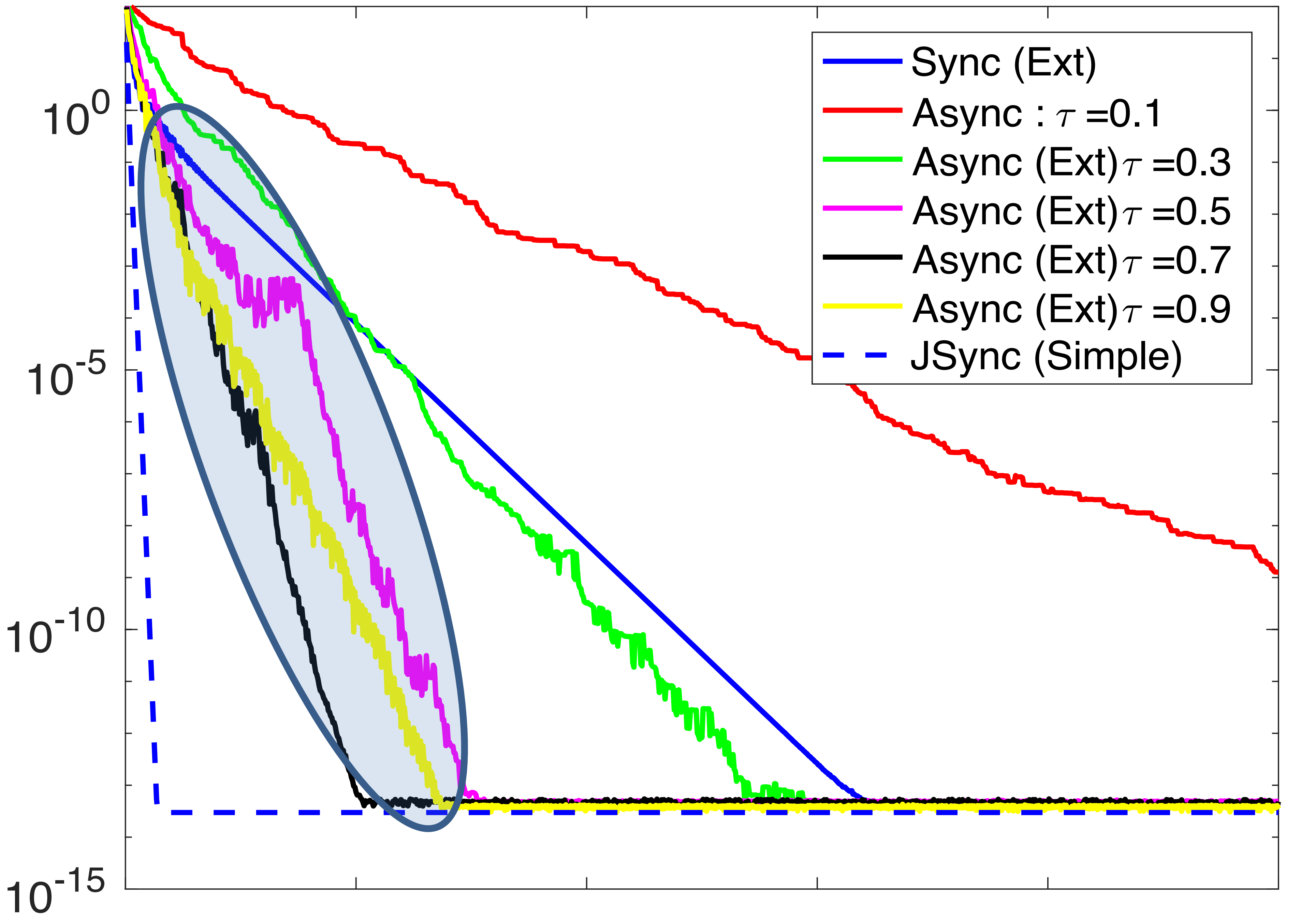
then the asynchronous implementation of $\mathbf{H}(\boldsymbol{\Theta})$ converges to $\mathbf{y}_{\mathrm{GF}}$.

Coutino, Leus, *Asynchronous Distributed Edge-Variant Graph Filters,* IEEE DSW, 2019

Asynchronous graph filtering

Example: ARMA CEV-GF

$$\mathbf{H} = \mathbf{H}_{\mathrm{B}}\mathbf{H}_{\mathrm{A}} = [\sum_{l=0}^{1} \phi_l \mathbf{S}^l][\sum_{k=1}^{3} \mathbf{\Phi}_k \mathbf{S}^{k-1}]^{-1}$$



$$\rho(\mathbf{H}) = 2.048$$

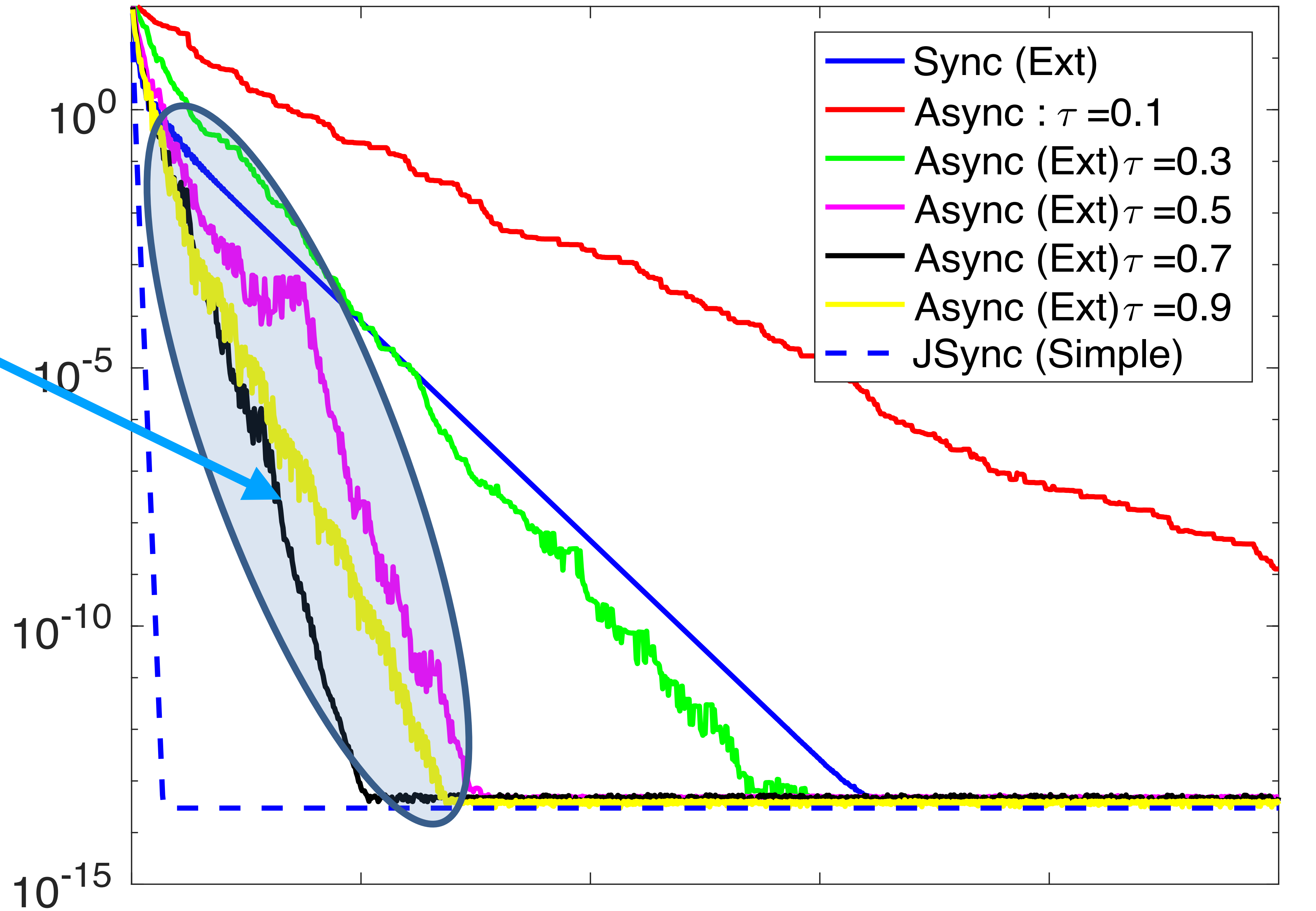
τ **: synchronization rate**

Asynchronous graph filtering

Example: ARMA CEV-GF

$$\mathbf{H} = \mathbf{H}_{\mathrm{B}}\mathbf{H}_{\mathrm{A}} = [\sum_{l=0}^{1} \phi_l \mathbf{S}^l][\sum_{k=1}^{3} \mathbf{\Phi}_k \mathbf{S}^{k-1}]^{-1}$$

non monotone convergence



$$\rho(\mathbf{H}) = 2.048$$

τ : synchronization rate

How to deal with large graph filter orders?

Cascaded graph filter implementation

- For large graph filter orders K

finding $\boldsymbol{\Theta}$ for $\mathbf{H}(\boldsymbol{\Theta})$ becomes severely ill-conditioned.

Cascaded graph filter implementation

- For large graph filter orders K

finding $\mathbf{\Theta}$ for $\mathbf{H}(\mathbf{\Theta})$ becomes severely ill-conditioned.

[most shift operators have poor spectral qualities]

Cascaded graph filter implementation

- For large graph filter orders K

finding $\boldsymbol{\Theta}$ for $\mathbf{H}(\boldsymbol{\Theta})$ becomes severely ill-conditioned.

[most shift operators have poor spectral qualities]

Cascaded Graph filters

Limits order of GF and use it as a building block (module)

$$\mathscr{H}(\mathbf{S};\boldsymbol{\Theta}) \triangleq \prod_{i=1}^{Q} \mathbf{H}(\boldsymbol{\Theta}_i)$$

Coutino, Leus, *On Distributed Consensus by a Cascade Of Generalized Graph Filters*, Asilomar, 2019

Cascaded graph filter implementation

- For large graph filter orders K

finding $\boldsymbol{\Theta}$ for $\mathbf{H}(\boldsymbol{\Theta})$ becomes severely ill-conditioned.

[most shift operators have poor spectral qualities]

Connections with GNNs

Cascaded Graph filters

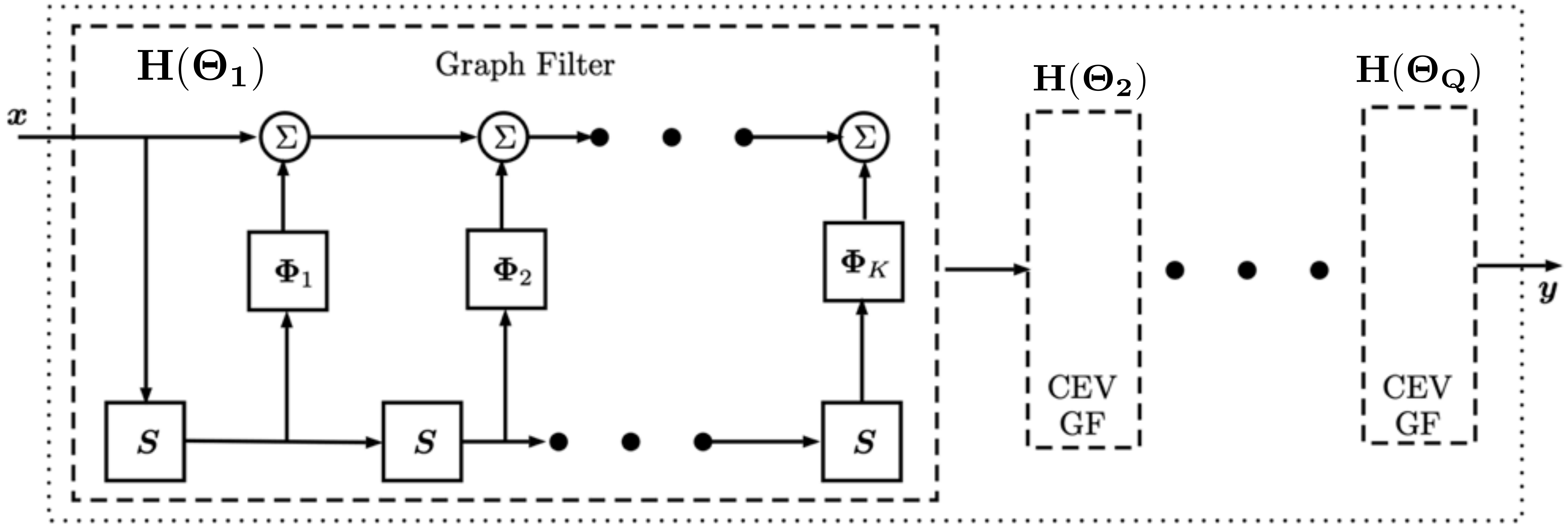
Limits order of GF and use it as a building block (module)

$$\mathscr{H}(\mathbf{S};\boldsymbol{\Theta}) \triangleq \prod_{i=1}^{Q} \mathbf{H}(\boldsymbol{\Theta}_i)$$

Coutino, Leus, *On Distributed Consensus by a Cascade Of Generalized Graph Filters*, Asilomar, 2019

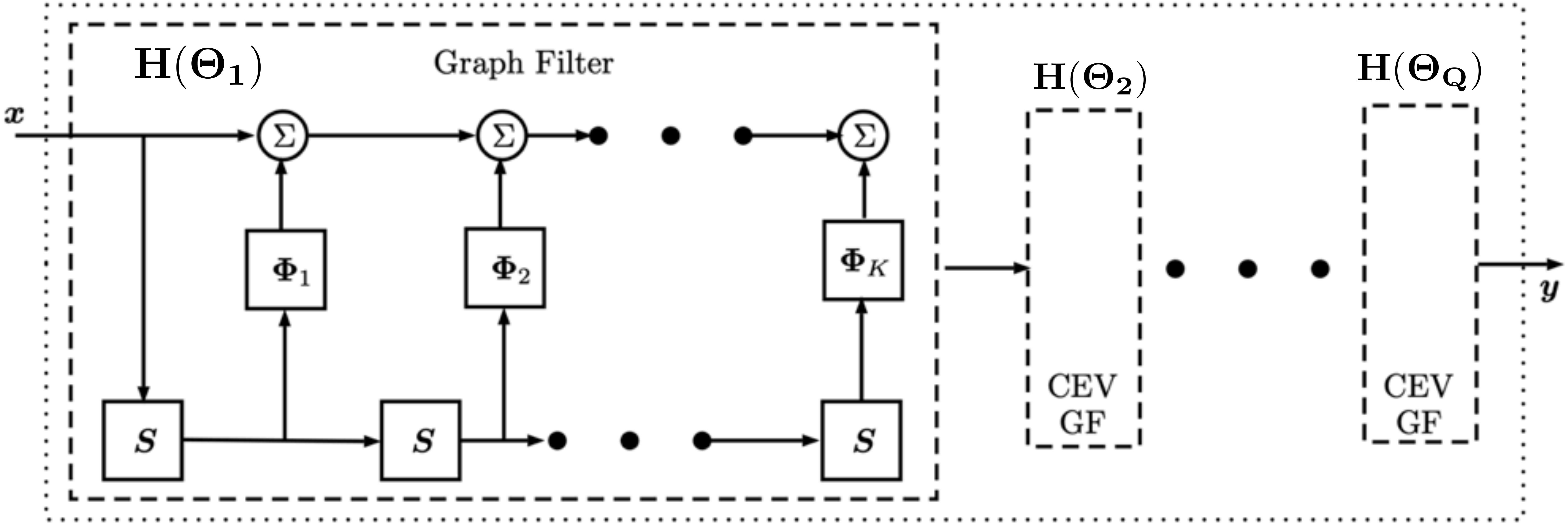
Cascaded graph filter implementation

$$\mathscr{H}(\mathbf{S};\boldsymbol{\Theta}) \triangleq \prod_{i=1}^{Q} \mathbf{H}(\boldsymbol{\Theta}_i)$$



Cascaded graph filter implementation

$$\mathscr{H}(\mathbf{S};\boldsymbol{\Theta}) \triangleq \prod_{i=1}^{Q} \mathbf{H}(\boldsymbol{\Theta}_i)$$

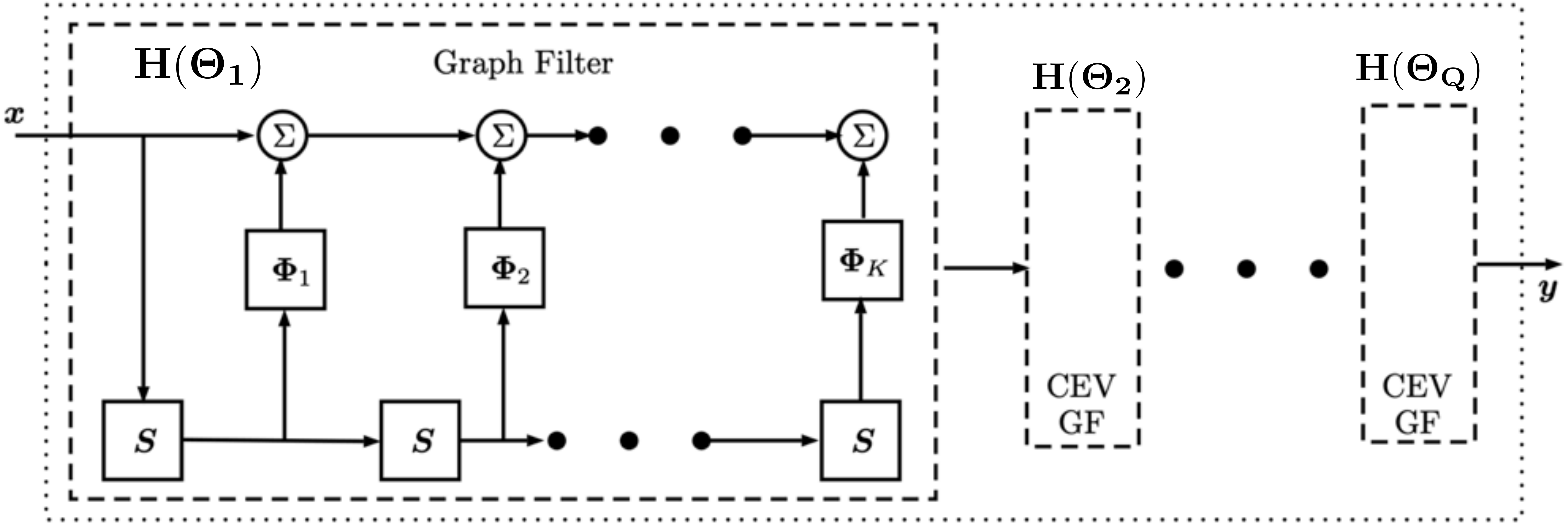


✦ This implementation aims to

- improve conditioning of design problem
- reduced-sized optimization problems
- obtain better performance with a reduced order

Cascaded graph filter implementation

$$\mathscr{H}(\mathbf{S};\boldsymbol{\Theta}) \triangleq \prod_{i=1}^{Q} \mathbf{H}(\boldsymbol{\Theta}_i)$$



✦ This implementation aims to

- improve conditioning of design problem
- reduced-sized optimization problems
- obtain better performance with a reduced order

However, this leads to a
non convex design problem
☹

Cascaded graph filter implementation

- Cascaded graph filter parameters can be found by the nonconvex problem

$$\{\boldsymbol{\Theta}_i^*\}_{i=1}^{Q} = \underset{\{\boldsymbol{\Theta}_i\}_{i=1}^{Q}}{\arg\min} \ \|\mathcal{H}(\mathbf{S}, \boldsymbol{\Theta}) - \mathbf{H}^*\|$$

$$\text{s.t.} \quad \boldsymbol{\Theta}_i \in \mathcal{C}_i, \forall\, i \in \{1, \ldots, Q\}$$

similar to the approach for learning parameters of GNNs.

Cascaded graph filter implementation

- Cascaded graph filter parameters can be found by the nonconvex problem

$$\{\boldsymbol{\Theta}_i^*\}_{i=1}^Q = \underset{\{\boldsymbol{\Theta}_i\}_{i=1}^Q}{\arg\min} \; \|\mathcal{H}(\mathbf{S}, \boldsymbol{\Theta}) - \mathbf{H}^*\|$$

$$\text{s.t.} \quad \boldsymbol{\Theta}_i \in \mathcal{C}_i, \forall\, i \in \{1, \ldots, Q\}$$

coefficients constraints, e.g., interval

similar to the approach for learning parameters of GNNs.

Cascaded graph filter implementation

- Cascaded graph filter parameters can be found by the nonconvex problem

$$\{\boldsymbol{\Theta}_i^*\}_{i=1}^{Q} = \underset{\{\boldsymbol{\Theta}_i\}_{i=1}^{Q}}{\arg\min} \ \|\mathcal{H}(\mathbf{S}, \boldsymbol{\Theta}) - \mathbf{H}^*\|$$

$$\text{s.t.} \quad \boldsymbol{\Theta}_i \in \mathcal{C}_i, \forall\, i \in \{1, \ldots, Q\}$$

coefficients constraints, e.g., interval

similar to the approach for learning parameters of GNNs.

- Alternatively, we can perform a sequential refitting process using a partition

$$\mathcal{H}(\mathbf{S}; \boldsymbol{\Theta}) = \mathbf{H}_{\mathrm{l}}\mathbf{M}\mathbf{H}_{\mathrm{r}}$$

$$\mathbf{H}_{\mathrm{l}} \triangleq \mathbf{H}(\boldsymbol{\Theta}_Q) \qquad \mathbf{M} \triangleq \prod_{q=2}^{Q-1} \mathbf{H}(\boldsymbol{\Theta}_q) \qquad \mathbf{H}_{\mathrm{r}} \triangleq \mathbf{H}(\boldsymbol{\Theta}_1)$$

which exploits the sparsity of the involved matrices.

Cascaded graph filter implementation

- For fixed $\mathbf{H}_{\mathrm{r}}$ and $\mathbf{M}$ the design for $\mathbf{H}_{\mathrm{l}}$ is given by

$$\underset{\boldsymbol{\theta}_q}{\arg\min} \|\boldsymbol{\Omega}_{\mathrm{l}}\boldsymbol{\theta}_q - \mathrm{vec}(\mathbf{H}^*)\|_2$$

[linSparseSolve]

$$\boldsymbol{\Omega}_{\mathrm{l}} \triangleq (\mathbf{H}_{\mathrm{r}}^\top \mathbf{M}^\top \otimes \mathbf{I})\boldsymbol{\Psi}$$

$$\boldsymbol{\Psi} \triangleq [(\mathbf{I} \otimes \mathbf{I})\mathbf{J}, (\mathbf{S}^\top \otimes \mathbf{I})\mathbf{J}, \cdots, ((\mathbf{S}^\top)^K \otimes \mathbf{I})\mathbf{J}] : \mathbf{J}$$

selection matrix for nonzero entries of $\boldsymbol{\Phi}_k$

Cascaded graph filter implementation

- For fixed $\mathbf{H}_\mathrm{r}$ and $\mathbf{M}$ the design for $\mathbf{H}_\mathrm{l}$ is given by

$$\arg\min_{\boldsymbol{\theta}_q} \|\boldsymbol{\Omega}_\mathrm{l}\boldsymbol{\theta}_q - \mathrm{vec}(\mathbf{H}^*)\|_2$$

[linSparseSolve]

$$\boldsymbol{\Omega}_\mathrm{l} \triangleq (\mathbf{H}_\mathrm{r}^\top \mathbf{M}^\top \otimes \mathbf{I})\boldsymbol{\Psi}$$

$$\boldsymbol{\Psi} \triangleq [(\mathbf{I} \otimes \mathbf{I})\mathbf{J}, (\mathbf{S}^\top \otimes \mathbf{I})\mathbf{J}, \cdots, ((\mathbf{S}^\top)^K \otimes \mathbf{I})\mathbf{J}] : \mathbf{J}$$

selection matrix for
nonzero entries of $\boldsymbol{\Phi}_k$

- ✦ $\boldsymbol{\Omega}_\mathrm{l}$ accepts efficient memory storage
- ✦ preconditioner for $\boldsymbol{\Omega}_\mathrm{l}$ can be obtained if its constructed explicitly
- ✦ Otherwise, $\boldsymbol{\Omega}_\mathrm{l}$ can be computed as an operator, i.e., matrix-vector operation

Cascaded graph filter implementation

- For fixed $\mathbf{H}_\mathrm{r}$ and $\mathbf{M}$ the design for $\mathbf{H}_\mathrm{l}$ is given by

$$\arg\min_{\boldsymbol{\theta}_q} \|\boldsymbol{\Omega}_\mathrm{l}\boldsymbol{\theta}_q - \mathrm{vec}(\mathbf{H}^*)\|_2$$

[linSparseSolve]

$$\boldsymbol{\Omega}_\mathrm{l} \triangleq (\mathbf{H}_\mathrm{r}^\top \mathbf{M}^\top \otimes \mathbf{I})\boldsymbol{\Psi}$$

$$\boldsymbol{\Psi} \triangleq [(\mathbf{I} \otimes \mathbf{I})\mathbf{J}, (\mathbf{S}^\top \otimes \mathbf{I})\mathbf{J}, \cdots, ((\mathbf{S}^\top)^K \otimes \mathbf{I})\mathbf{J}] : \mathbf{J}$$

selection matrix for nonzero entries of $\boldsymbol{\Phi}_k$

- ✦ $\boldsymbol{\Omega}_\mathrm{l}$ accepts efficient memory storage
- ✦ preconditioner for $\boldsymbol{\Omega}_\mathrm{l}$ can be obtained if its constructed explicitly
- ✦ Otherwise, $\boldsymbol{\Omega}_\mathrm{l}$ can be computed as an operator, i.e., matrix-vector operation

Linear solver exploiting such characteristics are readily available [Paige, ’82] [Fong, ’11]

Cascaded graph filter implementation

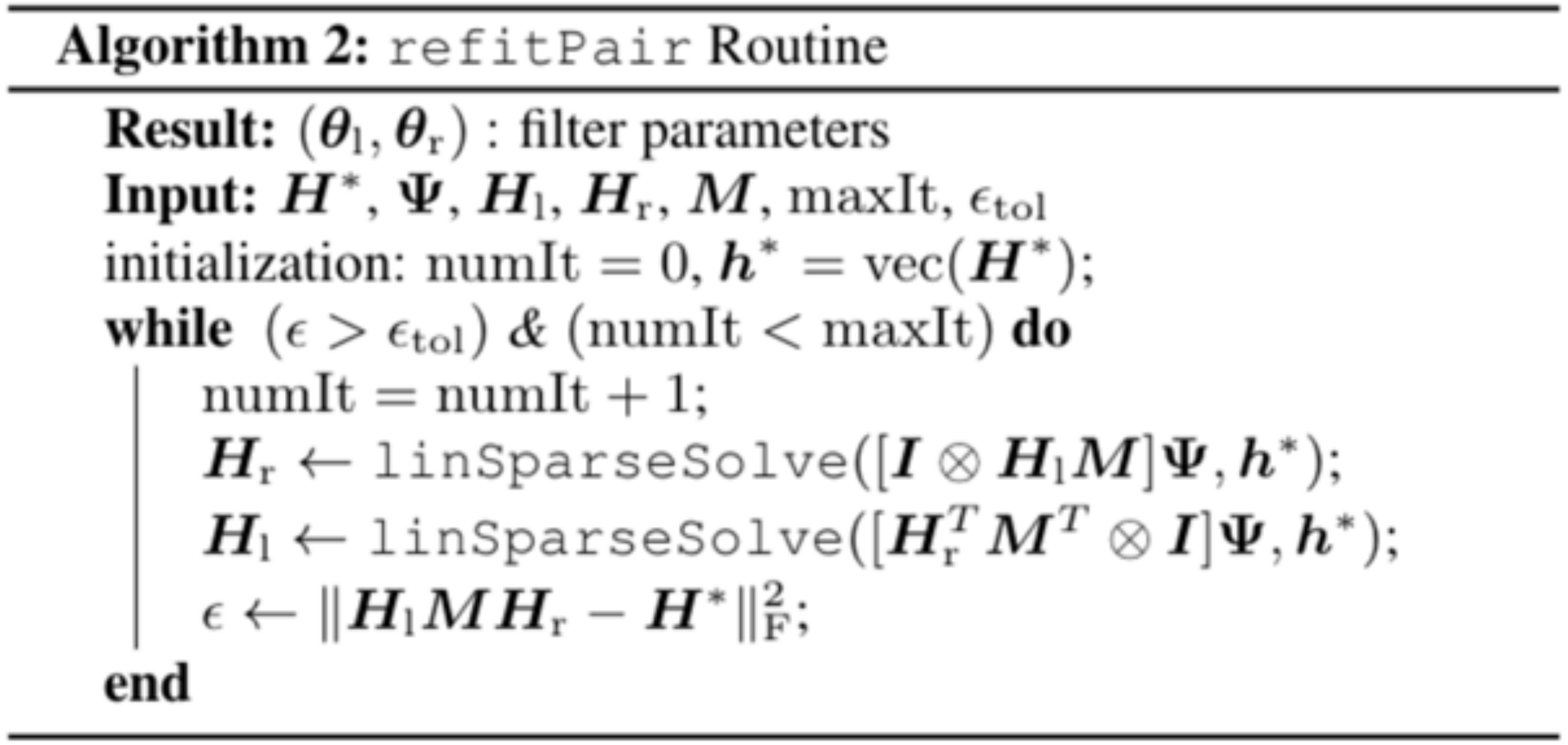
- Borrowing ideas from **RELAX** we fit $\{\mathbf{H}_{\mathrm{r}}, \mathbf{H}_{\mathrm{l}}\}$ until convergence.

  [Li, ’96]

  - as $\mathbf{M}$ is not included, sparsity of the system is preserved.
  - efficient sparse solvers can be used for each matrix
  - under mild conditions, two-block coordinate descent converges.

Cascaded graph filter implementation

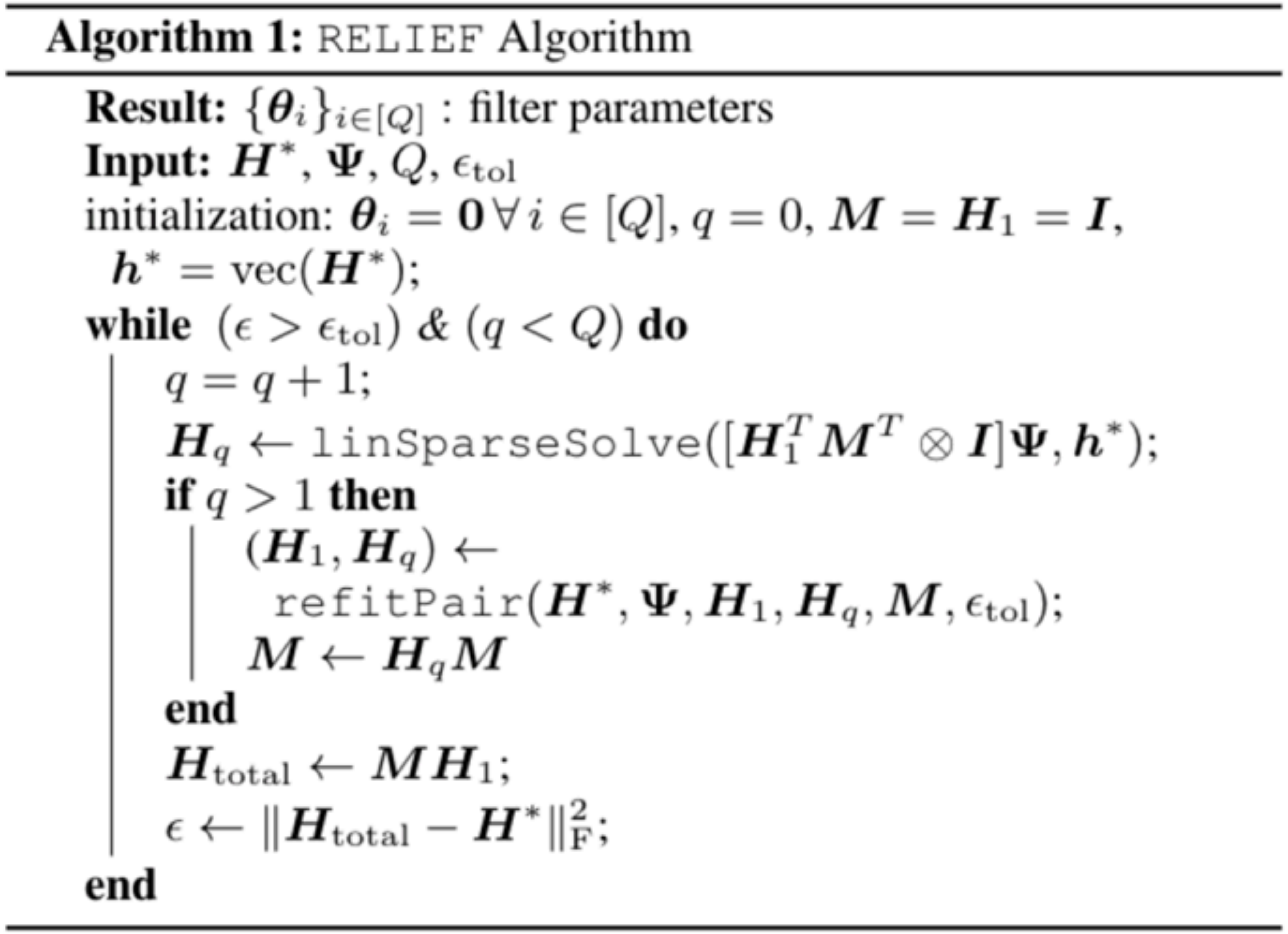
- Borrowing ideas from **RELAX** we fit $\{\mathbf{H}_\mathrm{r}, \mathbf{H}_\mathrm{l}\}$ until convergence.
  [Li, '96]
  - as $\mathbf{M}$ is not included, sparsity of the system is preserved.
  - efficient sparse solvers can be used for each matrix
  - under mild conditions, two-block coordinate descent converges.



```
Algorithm 2: refitPair Routine
  Result: $(\boldsymbol{\theta}_\mathrm{l}, \boldsymbol{\theta}_\mathrm{r})$ : filter parameters
  Input: $\boldsymbol{H}^*$, $\boldsymbol{\Psi}$, $\boldsymbol{H}_\mathrm{l}$, $\boldsymbol{H}_\mathrm{r}$, $\boldsymbol{M}$, maxIt, $\epsilon_\mathrm{tol}$
  initialization: numIt $= 0$, $\boldsymbol{h}^* = \mathrm{vec}(\boldsymbol{H}^*)$;
  while $(\epsilon > \epsilon_\mathrm{tol})$ & (numIt $<$ maxIt) do
      numIt $=$ numIt $+ 1$;
      $\boldsymbol{H}_\mathrm{r} \leftarrow$ linSparseSolve($[\boldsymbol{I} \otimes \boldsymbol{H}_\mathrm{l}\boldsymbol{M}]\boldsymbol{\Psi}, \boldsymbol{h}^*$);
      $\boldsymbol{H}_\mathrm{l} \leftarrow$ linSparseSolve($[\boldsymbol{H}_\mathrm{r}^T\boldsymbol{M}^T \otimes \boldsymbol{I}]\boldsymbol{\Psi}, \boldsymbol{h}^*$);
      $\epsilon \leftarrow \|\boldsymbol{H}_\mathrm{l}\boldsymbol{M}\boldsymbol{H}_\mathrm{r} - \boldsymbol{H}^*\|_\mathrm{F}^2$;
  end
```

Cascaded graph filter implementation

- Summary of the procedure :: **Right-Left Iterative Fitting [RELIEF]**

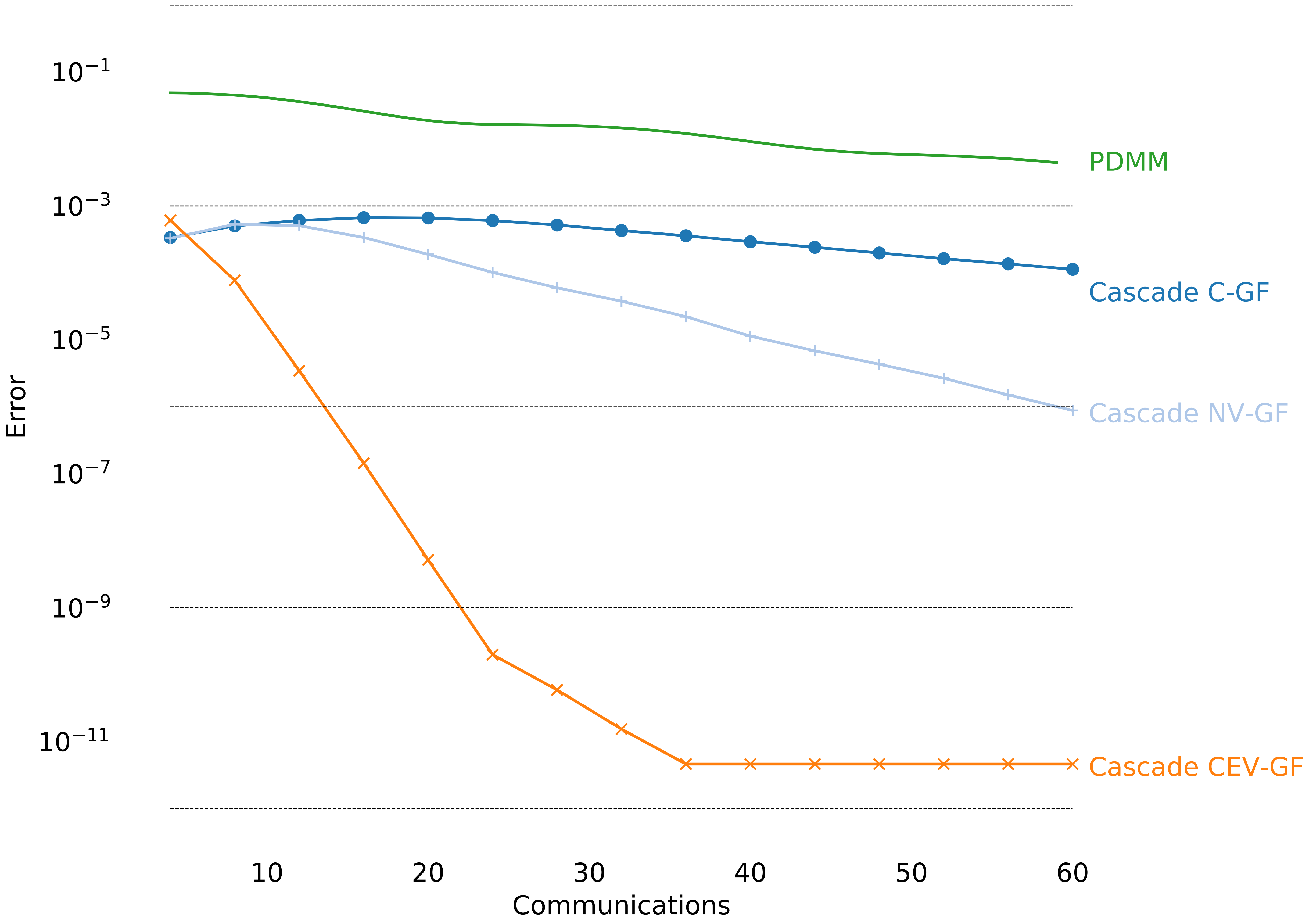


```
Algorithm 1: RELIEF Algorithm
Result: $\{\boldsymbol{\theta}_i\}_{i\in[Q]}$ : filter parameters
Input: $\boldsymbol{H}^*, \boldsymbol{\Psi}, Q, \epsilon_{\rm tol}$
initialization: $\boldsymbol{\theta}_i = \boldsymbol{0}\,\forall\, i \in [Q], q = 0, \boldsymbol{M} = \boldsymbol{H}_1 = \boldsymbol{I},$
 $\boldsymbol{h}^* = \mathrm{vec}(\boldsymbol{H}^*)$;
while $(\epsilon > \epsilon_{\rm tol})$ & $(q < Q)$ do
    $q = q + 1$;
    $\boldsymbol{H}_q \leftarrow$ linSparseSolve($[\boldsymbol{H}_1^T\boldsymbol{M}^T \otimes \boldsymbol{I}]\boldsymbol{\Psi}, \boldsymbol{h}^*$);
    if $q > 1$ then
        $(\boldsymbol{H}_1, \boldsymbol{H}_q) \leftarrow$
         refitPair($\boldsymbol{H}^*, \boldsymbol{\Psi}, \boldsymbol{H}_1, \boldsymbol{H}_q, \boldsymbol{M}, \epsilon_{\rm tol}$);
        $\boldsymbol{M} \leftarrow \boldsymbol{H}_q\boldsymbol{M}$
    end
    $\boldsymbol{H}_{\rm total} \leftarrow \boldsymbol{M}\boldsymbol{H}_1$;
    $\epsilon \leftarrow \|\boldsymbol{H}_{\rm total} - \boldsymbol{H}^*\|_{\rm F}^2$;
end
```

Coutino, Leus, *On Distributed Consensus by a Cascade Of Generalized Graph Filters*, Asilomar, 2019

Cascaded graph filter implementation

Example: Consensus over a network with 500 nodes



part 2 & 3 :: conclusions

- Graph signal processing arises as an alternative for distributed optimization
  - Significant benefits in terms of communication efficiency
  - Applications: distributed consensus, distributed imaging, beamforming
  - Requires knowledge of the data transformation
  - Data transform must be linear and data independent

part 2 & 3 :: conclusions

- Graph signal processing arises as an alternative for distributed optimization
  - Significant benefits in terms of communication efficiency
  - Applications: distributed consensus, distributed imaging, beamforming
  - Requires knowledge of the data transformation
  - Data transform must be linear and data independent
- Asynchronous graph filter is possible under mild conditions
  - Results hold for classical, node-varying, constrained edge-varying graph filters
  - For node-varying and constrained edge-varying filter order is critical

part 2 & 3 :: conclusions

- ◉ Graph signal processing arises as an alternative for distributed optimization
  - Significant benefits in terms of communication efficiency
  - Applications: distributed consensus, distributed imaging, beamforming
  - Requires knowledge of the data transformation
  - Data transform must be linear and data independent
- ◉ Asynchronous graph filter is possible under mild conditions
  - Results hold for classical, node-varying, constrained edge-varying graph filters
  - For node-varying and constrained edge-varying filter order is critical
- ◉ Cascaded graph filters alleviates ill-conditioning of large filter orders
  - Allows for an efficient sparse least squares design
  - Reduction in communication and computational cost
  - Implements only linear data transformations

part 4:: overview

- Role of graph filters in graph neural networks (GNNs)
  - GNNs ~ nonlinear graph filters
- For simplicity will discuss supervised learning
- How to go from neural networks to GNNs?
- Types of GNNs
  - What are graph convolutional neural networks?
  - How use edge varying GNNs?
- How to use GNNs for graph signal processing applications?
- For GNN pooling, transferability, and applications in control and resource allocation
  - T-9: Graph Neural Networks (F. Gama and A. Ribeiro)

Why we use filters in neural networks?

Supervised learning

- ◉ Relies on a dataset of R training examples
  $\mathscr{R} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_R, y_R)\}$
  - ✦ $\mathbf{x}_r$ the rth input data in space $\mathscr{X}$
  - ✦ y_r the rth output data in space $\mathscr{Y}$ (labels)
- ◉ Goal: learn a function f that maps $\mathbf{x}_r$ to y_r
  - ◉ we want f parametric: $f(\boldsymbol{\theta}) : \mathscr{X} \to \mathscr{Y}$

Supervised learning

- Deign parameters $\boldsymbol{\theta}$ such that
  - minimize a cost distance between $f(\boldsymbol{\theta}, \mathbf{x}_r)$ and $\mathbf{y}_r$ (e.g., MSE)

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \frac{1}{R} \sum_{r=1}^{R} \left(f(\boldsymbol{\theta}, \mathbf{x}_r) - y_r\right)^2$$

  - generalize well for test data $\mathbf{x}_r \notin \mathscr{R}$

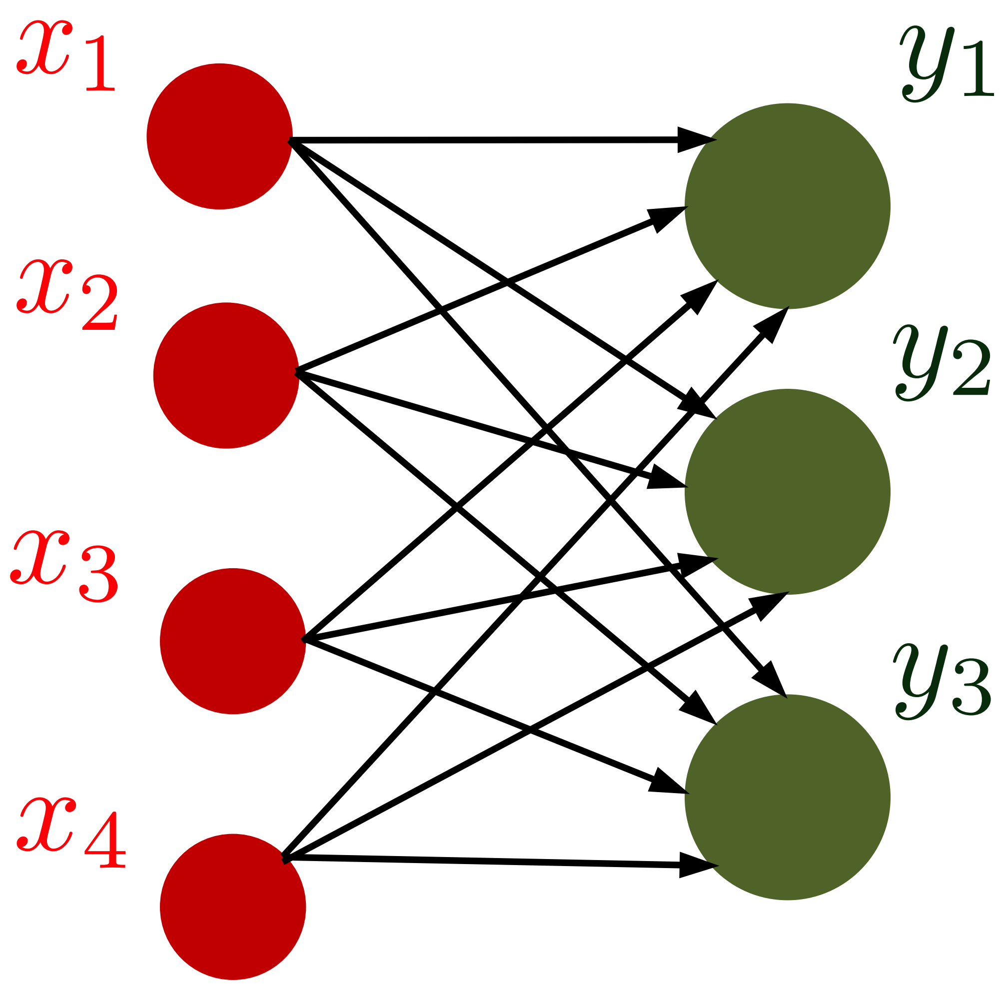
Neural networks

- Express function f as a cascade of layered functions

$$f(\boldsymbol{\theta}, \mathbf{x}) = f^3(\boldsymbol{\theta}^3, f^2(\boldsymbol{\theta}^2, f^1(\boldsymbol{\theta}^1, \mathbf{x})))$$

layer 1: parameters $\boldsymbol{\theta}^1$
layer 2: parameters $\boldsymbol{\theta}^2$
layer 3: parameters $\boldsymbol{\theta}^3$

- No structure in the data; perceptron



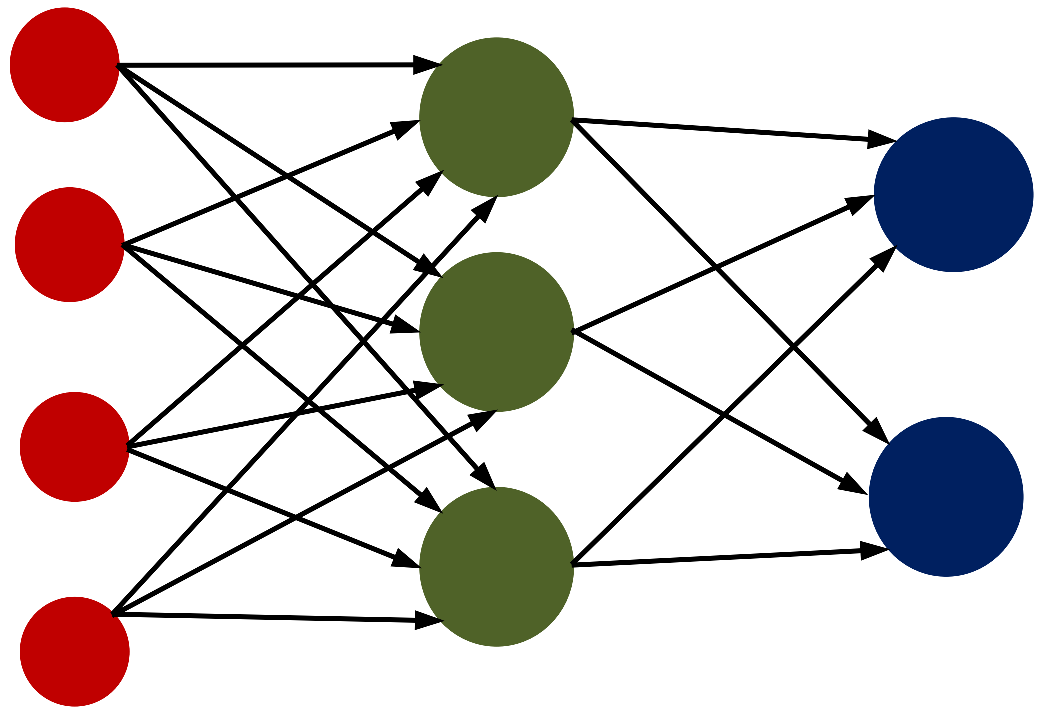
$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- Parameters $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}\}$
- Pointwise nonlinearity $\sigma(\,\cdot\,)$

$$\text{ReLU}(x) = \begin{cases} x & x > 0 \\ 0 & \text{otw} \end{cases}$$

Neural networks

- No structure in the data: multi-layer perceptron
  - Improves expressivity



$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_2$

$$\mathbf{x}^l = \sigma(\mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l)$$

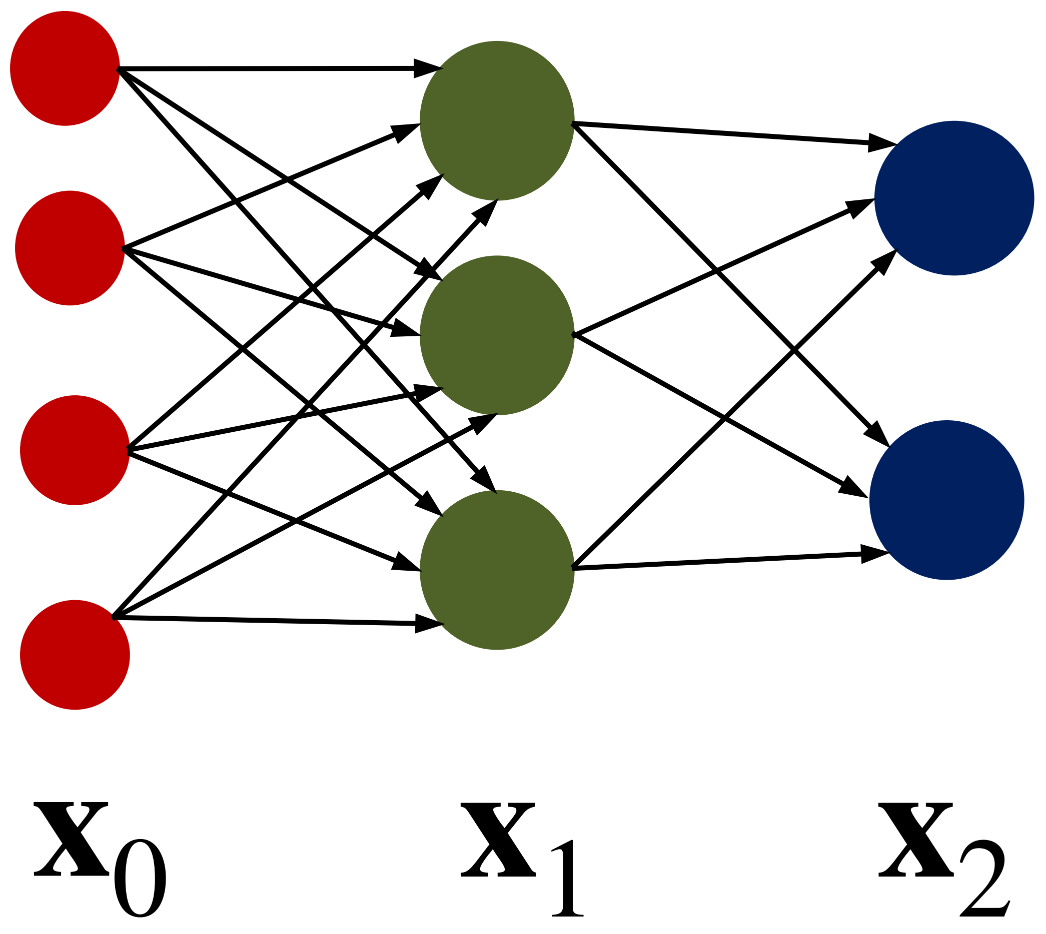
- Input features $\mathbf{x}^0 = \mathbf{x}_r$
- Output features $\mathbf{x}^L$
- Propagation rule at layer l

$\mathbf{x}^{l-1}$: input layer l = output layer $l-1$
$\mathbf{x}^{l}$: output layer l

$\boldsymbol{\theta}^l = \{\mathbf{W}^l, \mathbf{b}^l\}$: parameters layer l

Neural networks

- Unrolling recursion $\mathbf{x}^L = \sigma(\mathbf{W}^L \mathbf{x}^{L-1} + \mathbf{b}^L)$



$$\begin{aligned}\mathbf{x}^L &= \sigma\left(\mathbf{W}^L \mathbf{x}^{L-1} + \mathbf{b}^L\right)\\ &= \sigma\left(\mathbf{W}^L \sigma\left(\mathbf{W}^{L-1} \mathbf{x}^{L-2} + \mathbf{b}^{L-1}\right) + \mathbf{b}^L\right)\\ &= \sigma\left(\mathbf{W}^L \sigma\left(\mathbf{W}^{L-1} \sigma\left(\ldots \sigma\left(\mathbf{W}^1 \mathbf{x}^0 + \mathbf{b}^0\right) + \mathbf{b}^{L-1}\right) + \mathbf{b}^L\right)\right.\end{aligned}$$

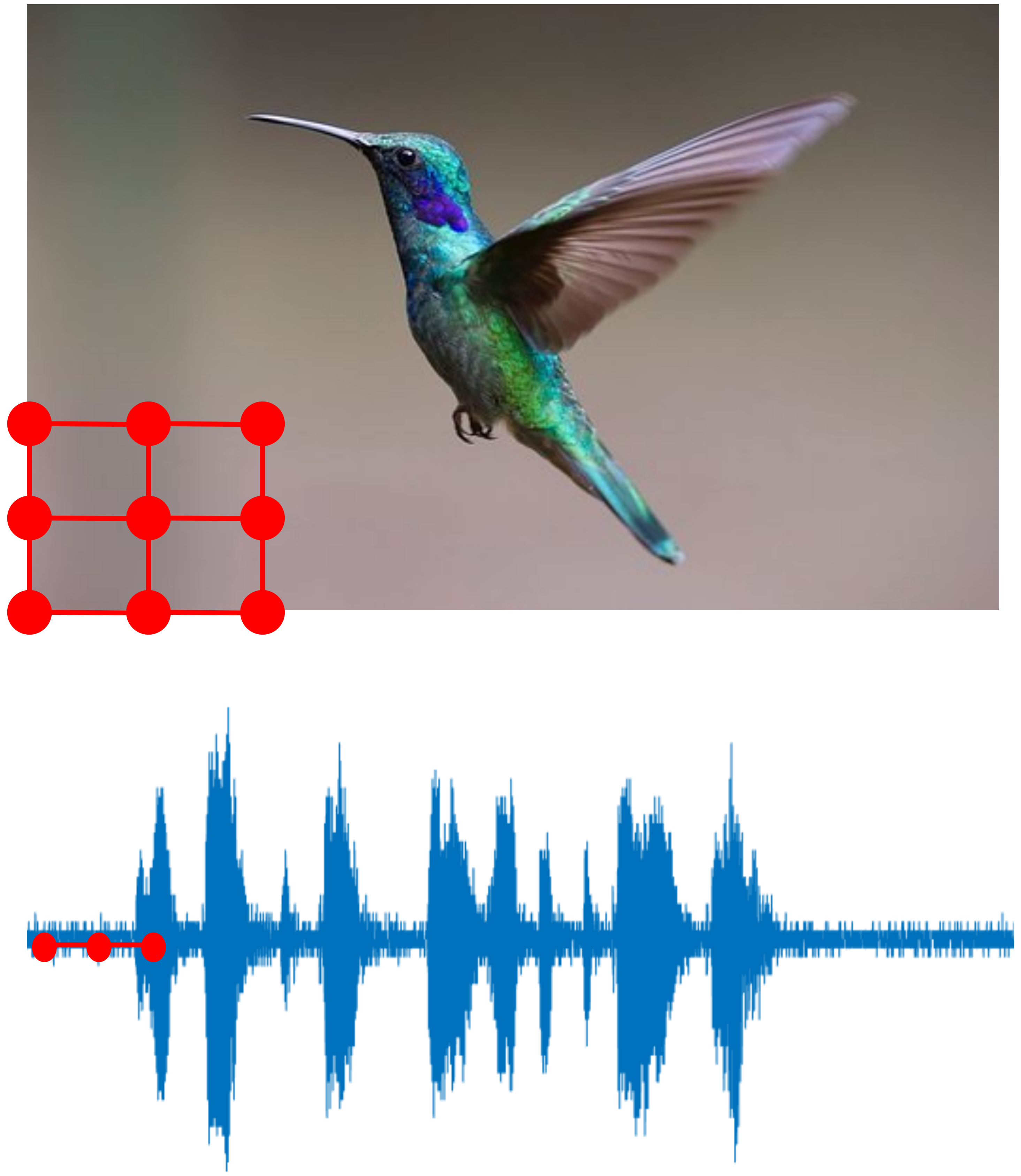
- $\mathbf{x}^L$ depends on $\mathbf{x}^0$ through a composition of linear functions and pointwise nonlinearities

Neural networks

- MLP fails in high dimensional data $\mathbf{x}^l = \sigma\left(\mathbf{W}^l\mathbf{x}^{l-1} + \mathbf{b}^l\right)$
  - if layers have dimensions $\dim(\mathbf{x}^l) = \dim(\mathbf{x}^{l-1}) \sim \mathcal{O}(N)$
    - $\dim\left(\mathbf{W}^l\right) \sim \mathcal{O}(N^2)$ parameters, e.g., $N = 1000 \rightarrow \mathcal{O}(10^6)$
    - complexity $\mathcal{O}(N^2)$
- need to exploit structure in data

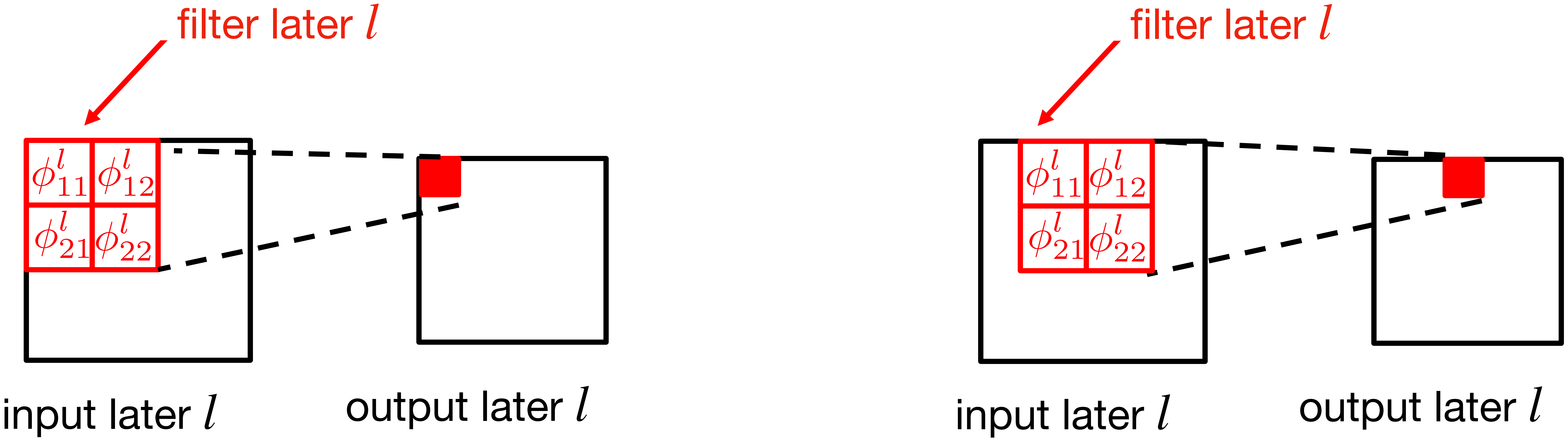
Neural networks

- structure in data
  - spatial data: pixel neighbors
  - temporal data: signal proximity
- reduce parameters by effective sharing
- reduce complexity by efficient implementation
- use spatial and temporal filters
  - no loose of discriminatory power



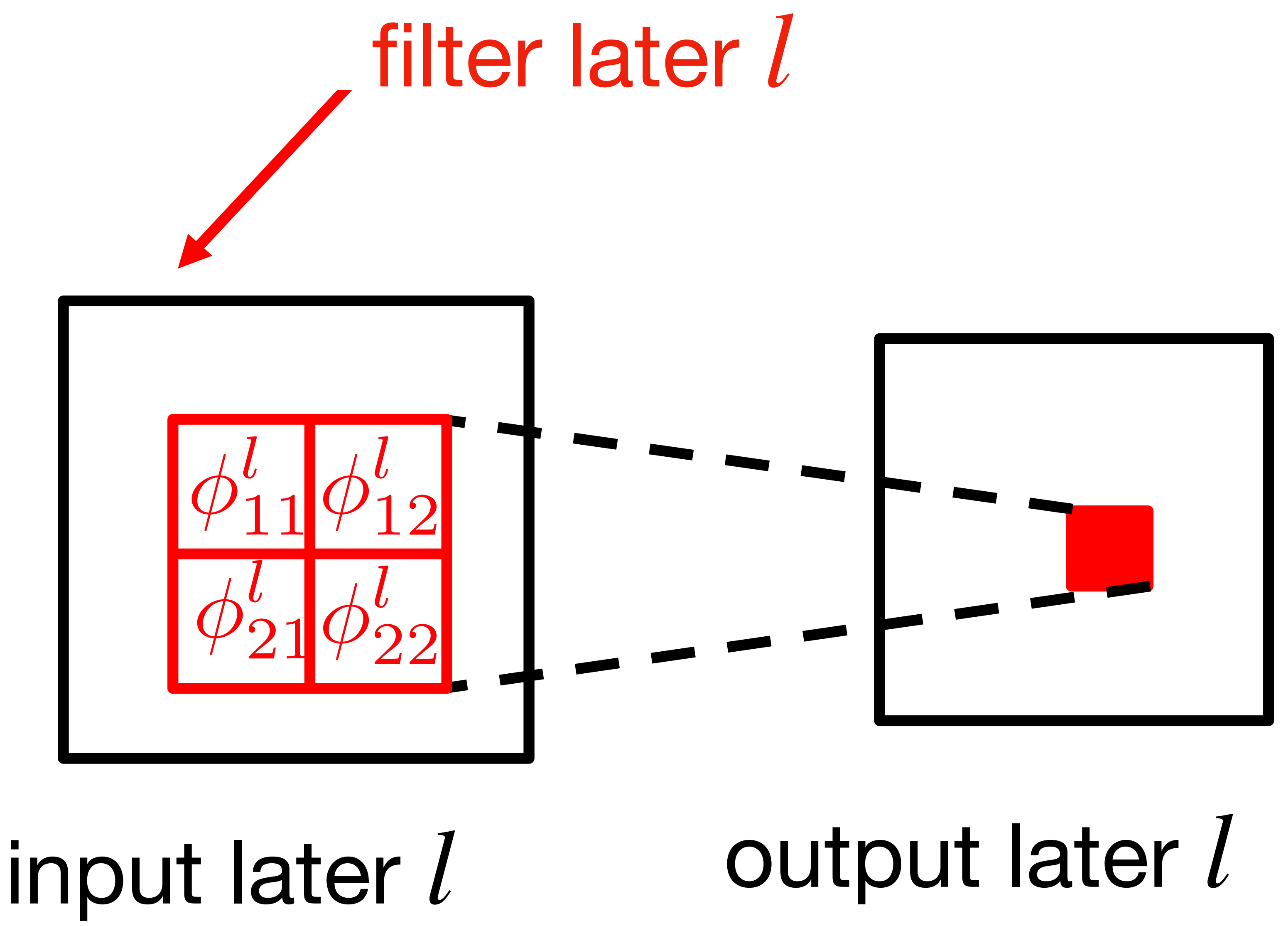
Filters in spatial convolutional layer

- MLP propagation rule $\mathbf{x}^l = \sigma\left(\mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l\right)$
- Spatial data: spatial convolution filter bank substitutes $\mathbf{W}^l$
  - filters apply the same parameters to different locations
  - bias $\mathbf{b}^l$ can be ignored or shared $\mathbf{b}^l = b^l \mathbf{1}$



Filters in spatial convolutional layer

- shift-and-sum convolves filter with input image



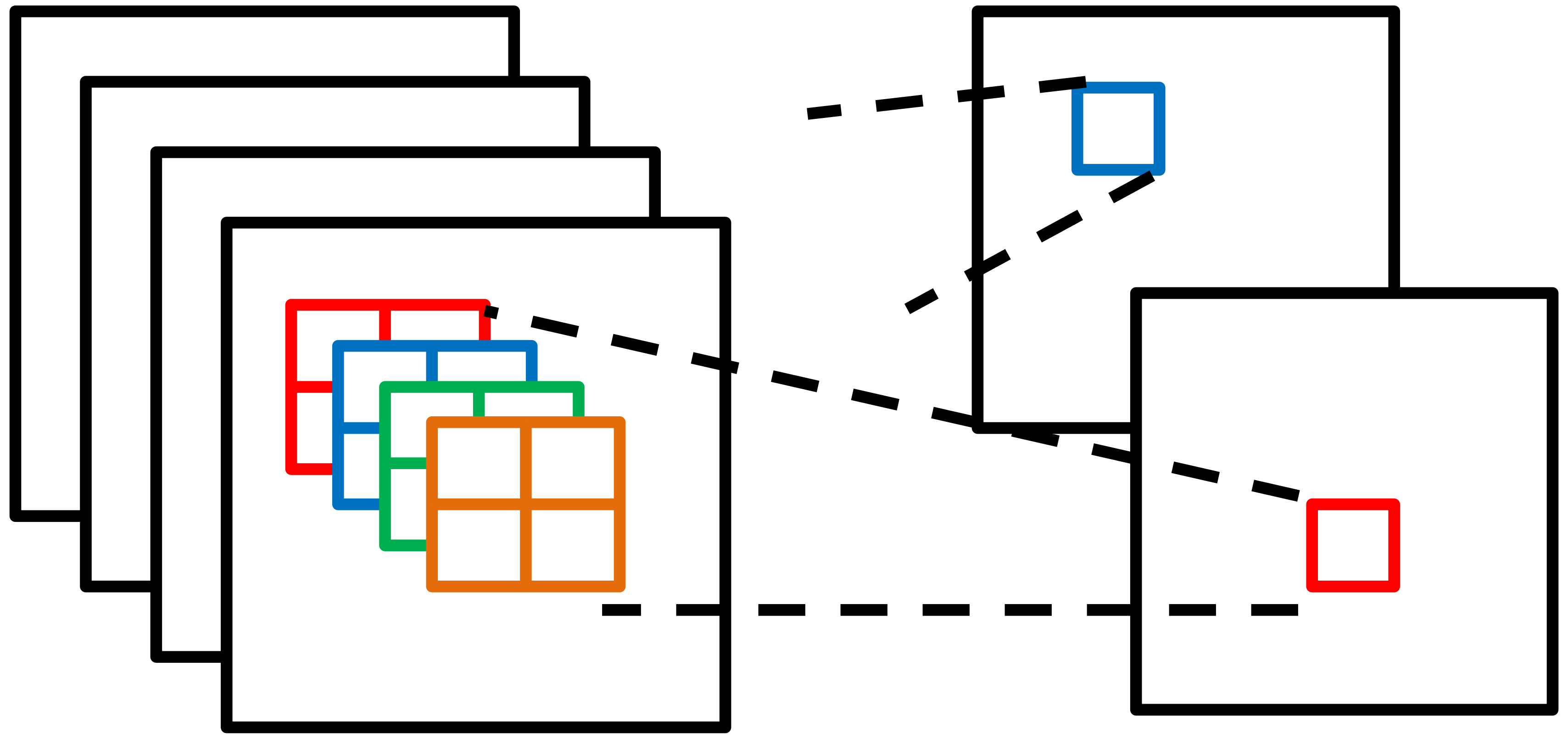
$$y_{ij}^l = \sum_{r=1}^{R} \sum_{c=1}^{C} \phi_{rc}^l x_{i-r,j-c}^{l-1}$$

vertical and horizontal input shifts

- spatial FIR convolutional filtering

Convolutional neural networks

- CNNs increase descriptive power with a parallel filter bank



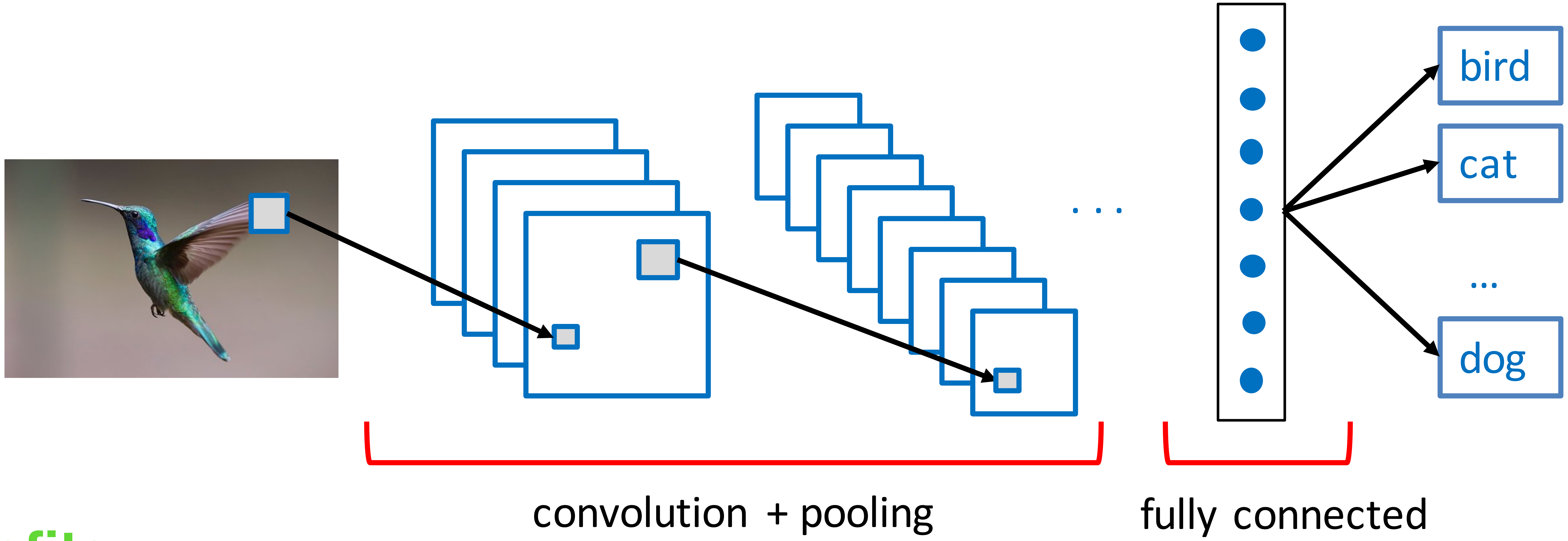
input layer l
has 4 inputs

output layer l
has 2 outputs

- input F images
- process each with a parallel bank of filters
- sum filter outputs to obtain higher-level features
- parameters are filter coefficients
- train with back propagation

CNN full stack

- Cascade of spatial filter bank and nonlinearities



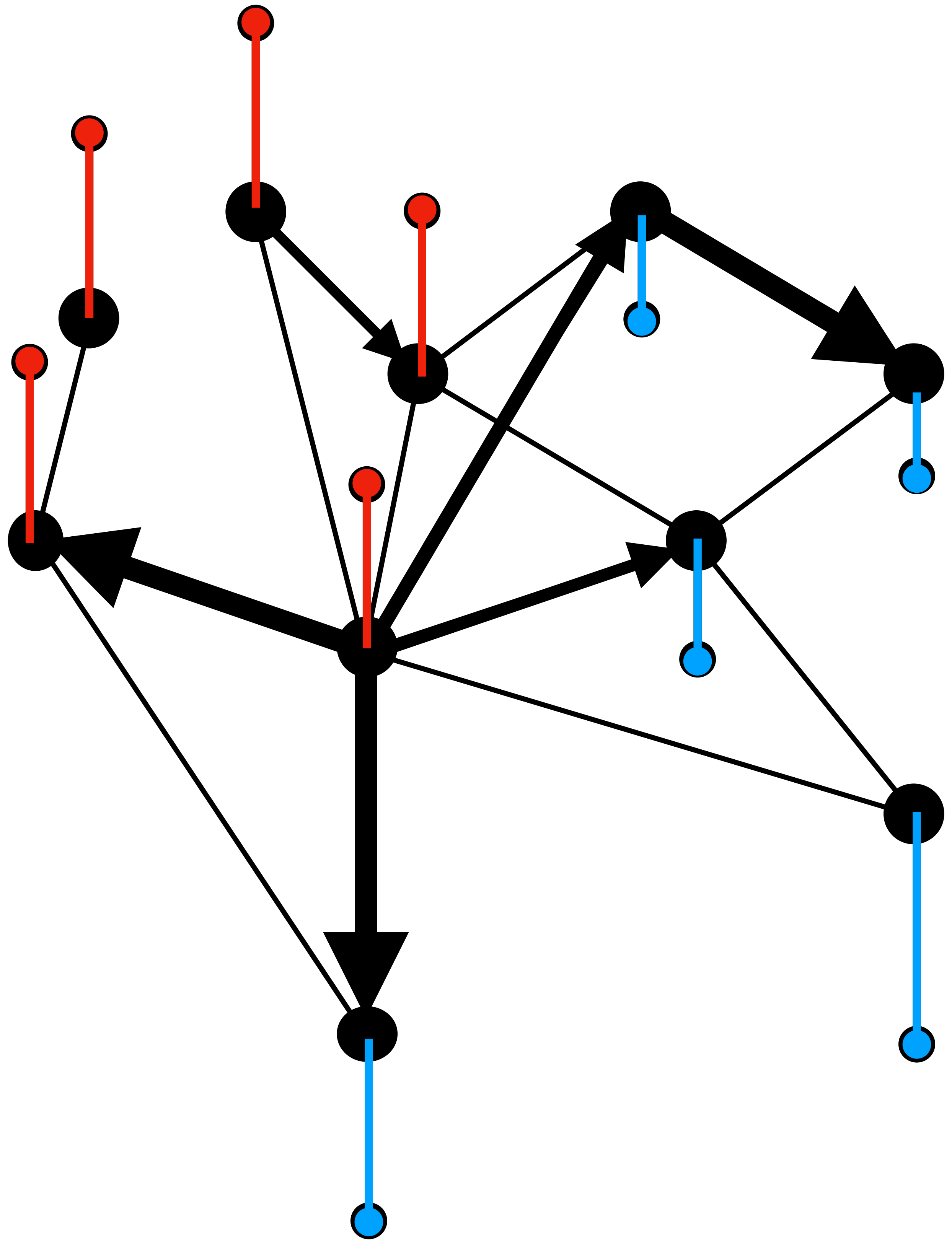
Benefits

- Parameters - independent on the image dimensions
- Complexity - spatial convolution (efficient)

What about data on graphs?

Learning from (ir)regular graph data

- Training samples $\mathbf{x}_r \in \mathbb{R}^N$ are graph signals
- Non-Euclidean structure
  - ✦conventional CNNs are inapplicable
- MLP can apply $\mathbf{x}^l = \sigma\left(\mathbf{W}^l\mathbf{x}^{l-1} + \mathbf{b}^l\right)$
  - ✦ignores the structure
  - ✦data demanding

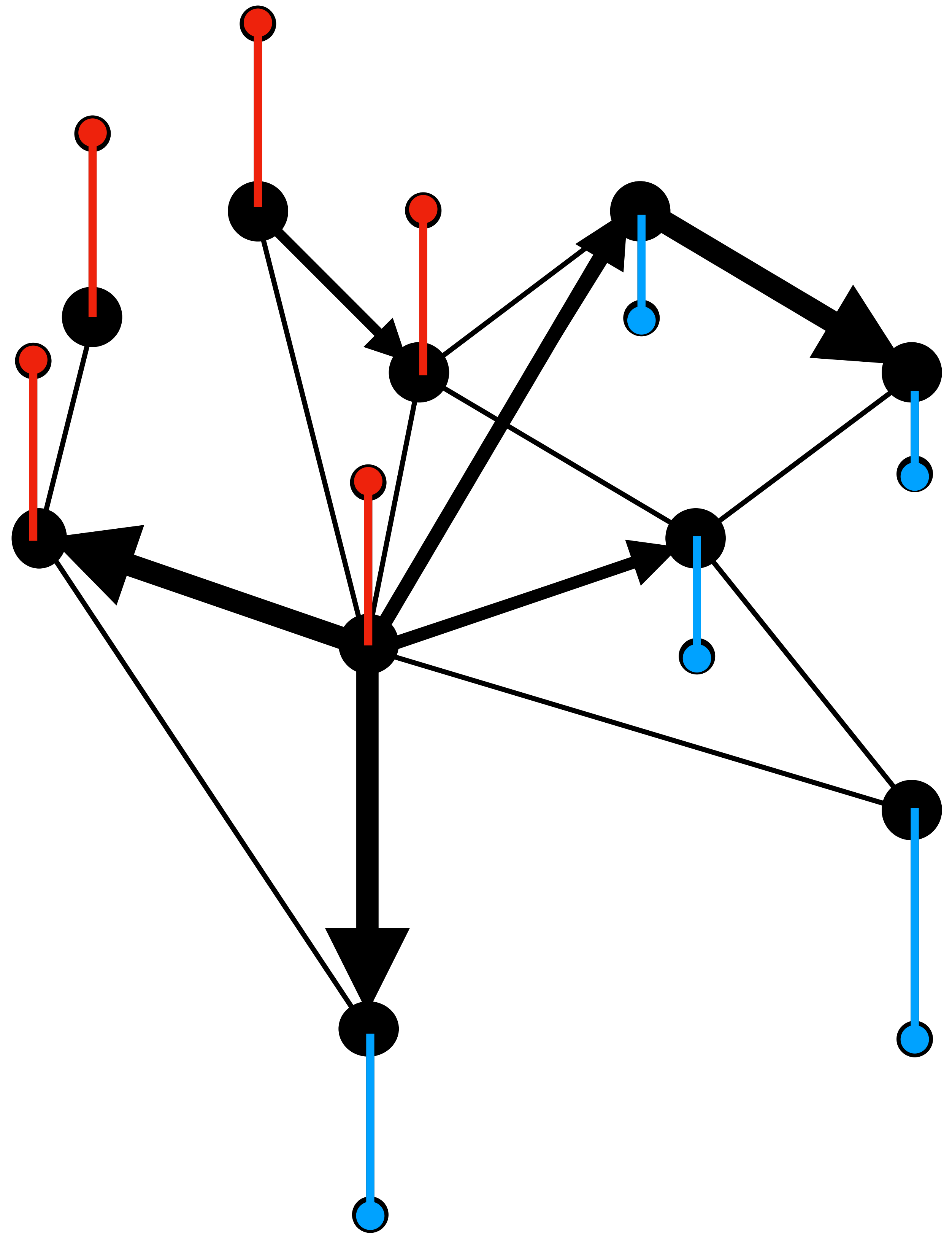


Learning from (ir)regular graph data

- Need a neural network solution to account to account for coupling signal-topology
- Graph as prior to estimate a parametric function

$$f(\boldsymbol{\theta}, \mathbf{S}) : \mathscr{X} \to \mathscr{Y}$$

  - ✦ $\mathbf{S}$ is the graph shift operator
  - ✦ $\boldsymbol{\theta}$ trainable parameters (i.e., filter coefficients)



Gama, Isufi, Leus, Ribeiro, *Graphs, Convolutions, and Neural Networks*, IEEE Signal Processing Magazine, under review, arXiv: 2003.03777

Graph neural networks

- Graph neural networks substitute $\mathbf{W}^l$ with graph filter bank
- Propagation rule through graph filters

$$\mathbf{x}^l = \sigma\left(\mathbf{H}^l \mathbf{x}^{l-1}\right)$$

- $\mathbf{H}^l$ graph filter at layer l for any shift operator $\mathbf{S}$
    - Edge-variant filter: EdgeNets
    - Node-variant filter: Node-variant GNNs [Gama'18 - DSW]
    - FIR filters: Graph convolutional neural networks [Gama'18 - TSP]
        - Chebyshev form: ChebNets [Defferrard'16 - NeurIPS]
    - ARMA filters: ARMANets
        - Direct, parallel, cascade [Wijesinghe'19 - NeurIPS] [Bianchi'19-arXiv]
        - Cayley form: CayleyNets [Levie'18 - TSP]

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

Graph convolutional neural networks

- Graph convolutional neural networks use a graph convolutional filter (FIR - ARMA)

Example: FIR

$$\mathbf{x}^{l} = \sigma\Big(\sum_{k=0}^{K} \phi_k^l \mathbf{S}^k \mathbf{x}^{l-1}\Big)$$

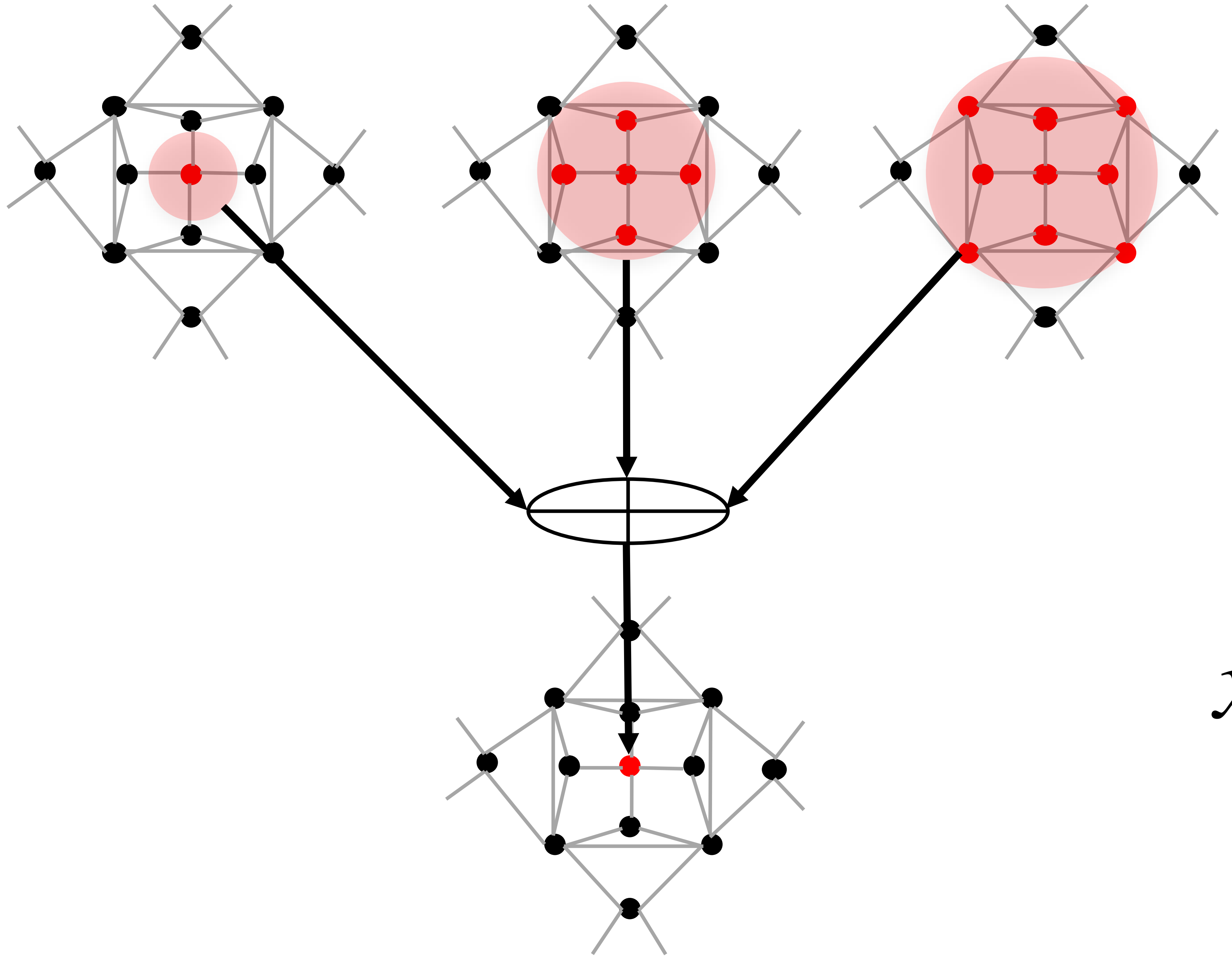
- parameters shared among all nodes and edges
- shift-and-sum convolves filter with graph signal

Gama, Marques, Leus, Ribeiro, *Convolutional Neural Networks Architectures for Signals Supported on Graphs*, IEEE Transactions on Signal Processing, 2018

Graph convolutional neural networks

- GCNN: shift-and-sum & shared parameters

$$\mathbf{x}^l = \sigma\left(\phi_o^l \mathbf{x}^{l-1} + \phi_1^l \mathbf{S}\mathbf{x}^{l-1} + \phi_2^l \mathbf{S}^2\mathbf{x}^{l-1}\right)$$



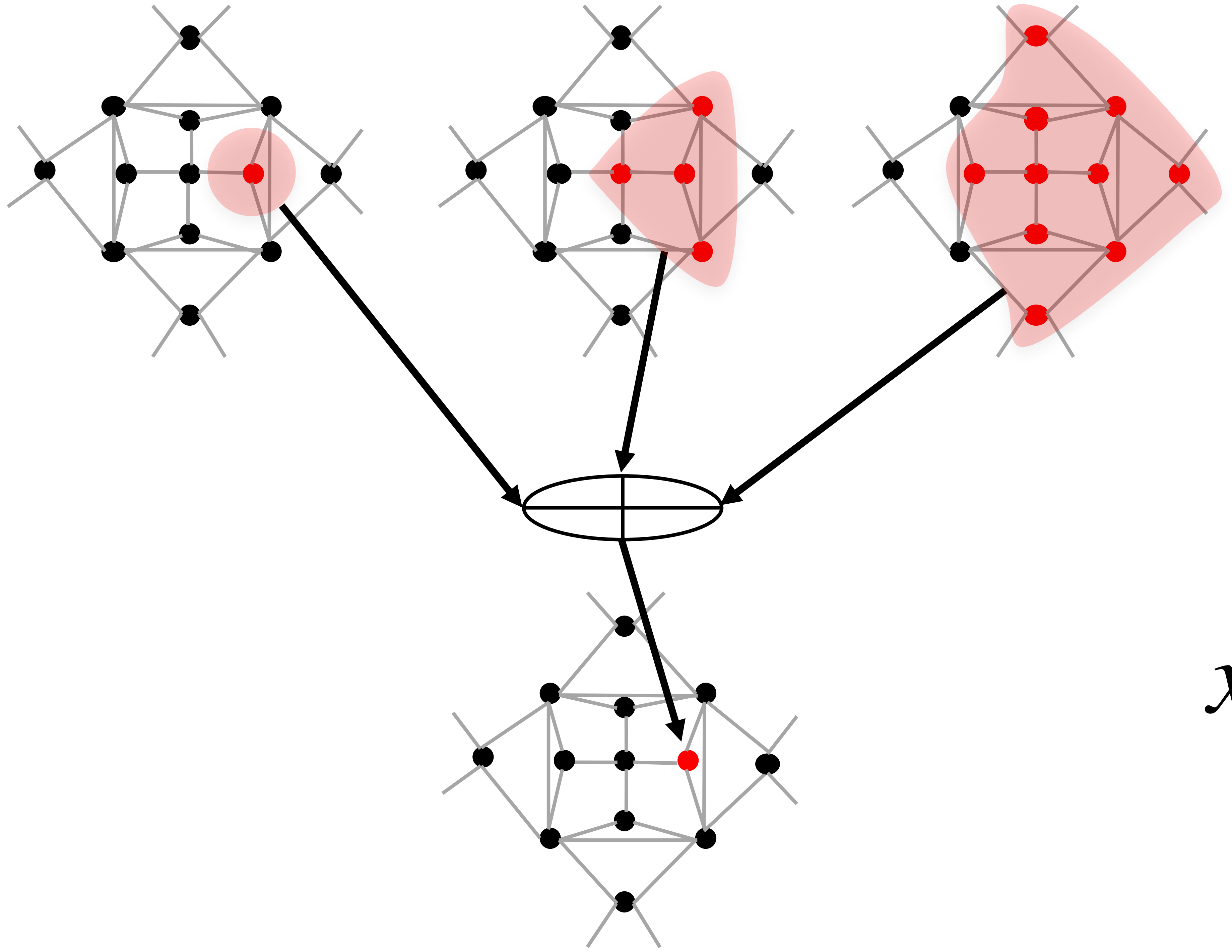
shift over the nodes

$$x_i^l = \sigma\left(\phi_o^l x_i^{l-1} + \phi_1^l [\mathbf{S}\mathbf{x}^{l-1}]_i + \phi_2^l [\mathbf{S}^2\mathbf{x}^{l-1}]_i\right)$$

Graph convolutional neural networks

- GCNN: shift-and-sum & shared parameters

$$\mathbf{x}^l = \sigma\left(\phi_o^l \mathbf{x}^{l-1} + \phi_1^l \mathbf{S}\mathbf{x}^{l-1} + \phi_2^l \mathbf{S}^2\mathbf{x}^{l-1}\right)$$



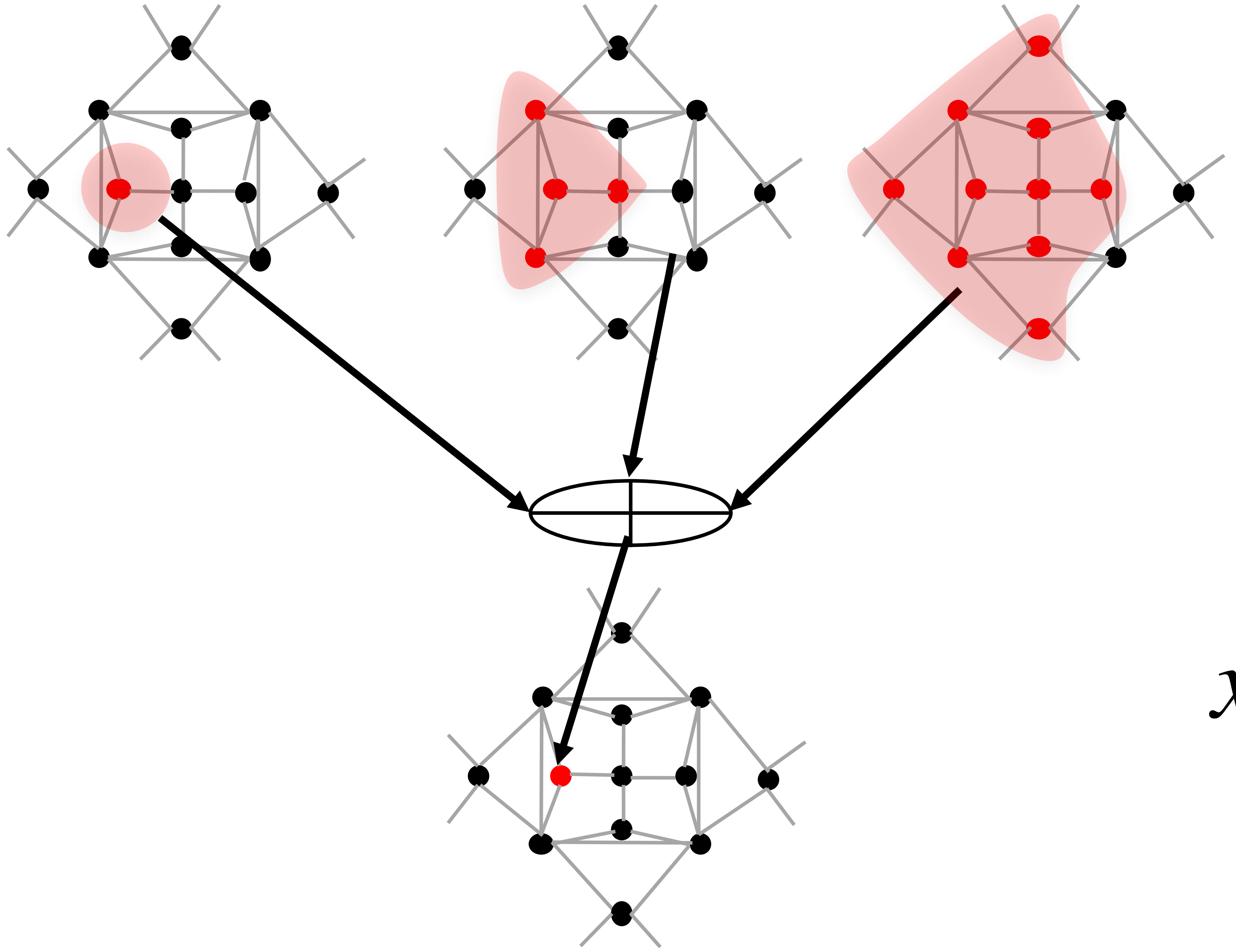
shift over the nodes

$$x_i^l = \sigma\left(\phi_o^l x_i^{l-1} + \phi_1^l [\mathbf{S}\mathbf{x}^{l-1}]_i + \phi_2^l [\mathbf{S}^2\mathbf{x}^{l-1}]_i\right)$$

Graph convolutional neural networks

- GCNN: shift-and-sum & shared parameters

$$\mathbf{x}^l = \sigma\left(\phi_o^l \mathbf{x}^{l-1} + \phi_1^l \mathbf{S}\mathbf{x}^{l-1} + \phi_2^l \mathbf{S}^2\mathbf{x}^{l-1}\right)$$

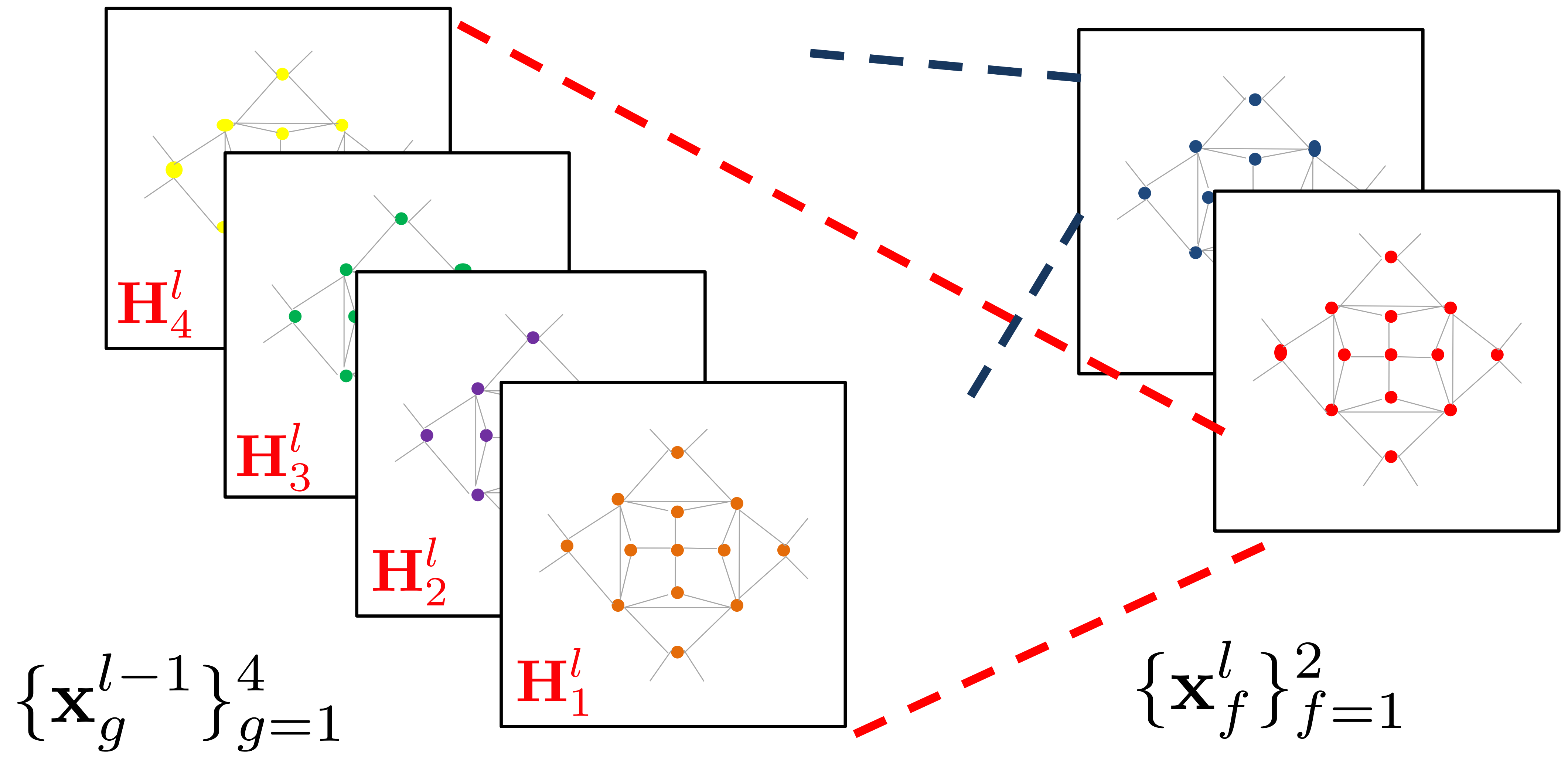


shift over the nodes

$$x_i^l = \sigma\left(\phi_o^l x_i^{l-1} + \phi_1^l [\mathbf{S}\mathbf{x}^{l-1}]_i + \phi_2^l [\mathbf{S}^2\mathbf{x}^{l-1}]_i\right)$$

Graph convolutional neural networks

- GCNNs increase descriptive power with a parallel filter bank



$\{\mathbf{x}_g^{l-1}\}_{g=1}^4$

input layer l
has 4 inputs

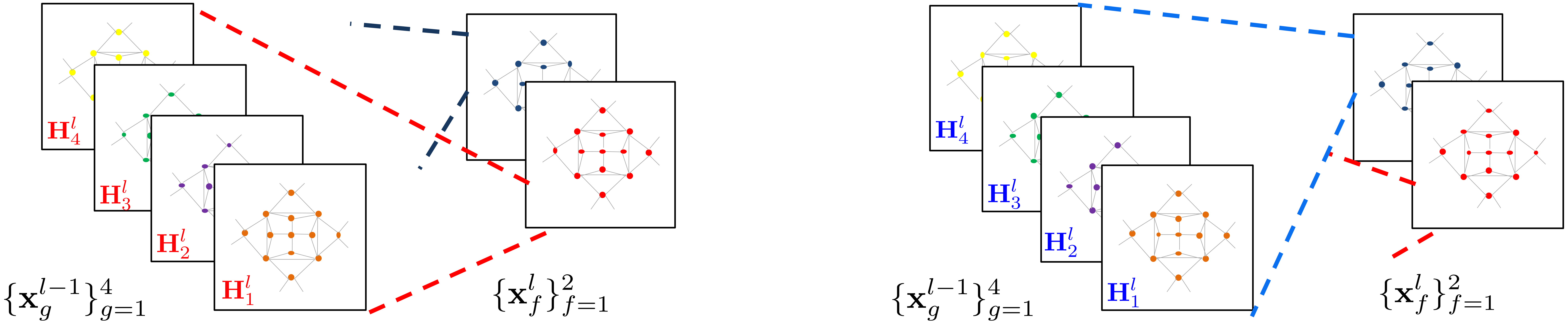
$\{\mathbf{x}_f^l\}_{f=1}^2$

output layer l
has 2 outputs

- F input graph signals $\{\mathbf{x}_g^{l-1}\}_{g=1}^F$
- process each signal with a graph filter
- sum filter outputs
- parameter are filter coefficients (backprop.)

Graph convolutional neural networks

- GCNNs increase descriptive power with a parallel filter bank



input feature

output feature

for all output features

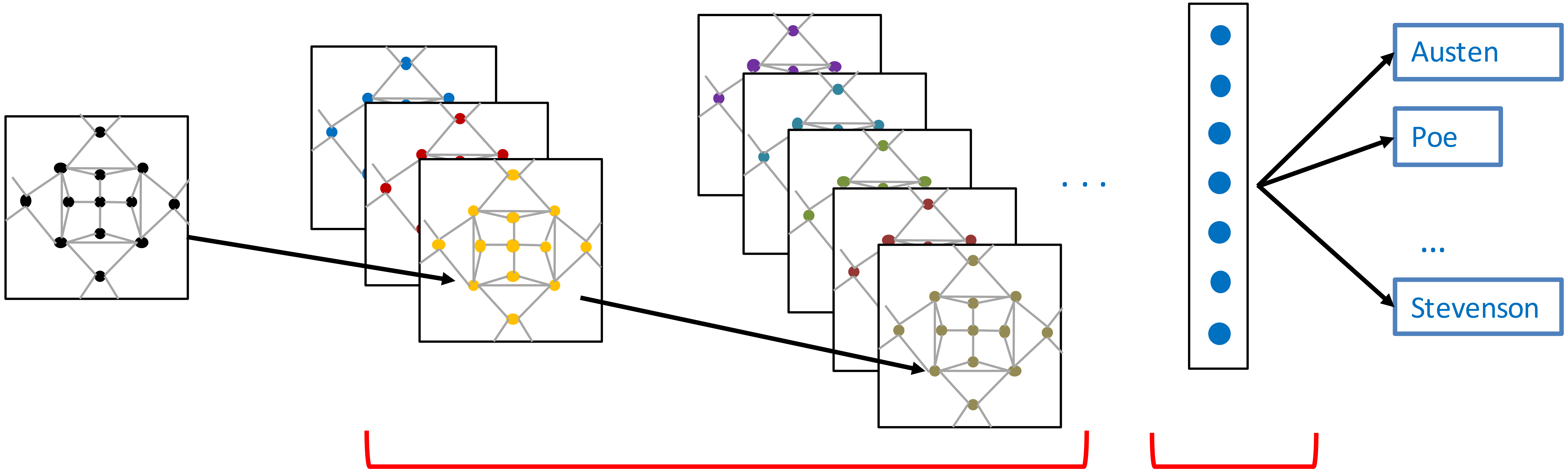
$$\mathbf{x}^l = \sigma\left(\sum_{g=1}^{F} \mathbf{H}_{fg}^l \mathbf{x}_g^{l-1}\right) = \sigma\left(\sum_{g=1}^{F}\sum_{k=0}^{K} \phi_{kfg}^l \mathbf{S}^l \mathbf{x}_g^{l-1}\right) \forall f \in \{1,\ldots,F\}$$

sum over all inputs

FIR filter

GCNN full stack

- Cascade graph filters and nonlinearities



Benefits

- Parameters $\mathcal{O}(KF^2L)$ - independent on the graph dimensions
- Complexity $\mathcal{O}(KMF^2L)$ - linear in number of edges

EdgeNet

- Substitutes FIR filters with edge-variant graph filter
- Propagation rule

$$\mathbf{x}_f^l = \sigma\left(\sum_{g=1}^{F} \mathbf{H}_{\mathrm{EV}fg}^{l}\mathbf{x}_g^{l-1}\right) \forall f \in \{1,\ldots,F\}$$

Edge-variant filter

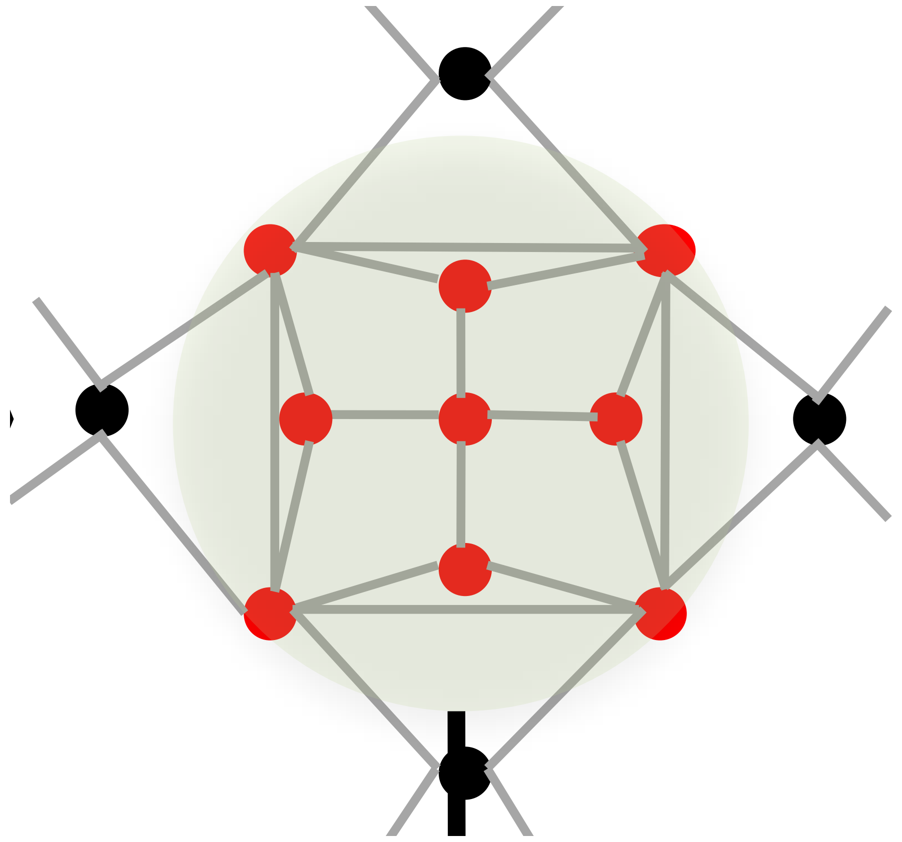
- The most general GNN
  - Includes all GCNN, all ARMANet, GIN, GAT

Isufi, Gama, Ribeiro, *Generalizing Graph Convolutional Neural Networks with Edge-Variant Recursions on Graphs*, EUSIPCO, 2019.

EdgeNet properties

- Different parameters per edge and node
  - ✦ Order $\mathcal{O}(MKF^2L)$
  - ✦ More flexibility
  - ✦ Requires only the support of $\mathbf{S}$
    - Adapts the edge weights to the task
    - Robust to uncertainties in edge weights
  - ✦ Requires fewer parallel filters and shallower networks
  - ✦ Can overfit and require more data than GCNN (FIR-filters)
- Complexity $\mathcal{O}(MKF^2L)$ - depends on edges

$\mathbf{\Phi}_k\mathbf{x}^{(k-1)}$



for kth iteration

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

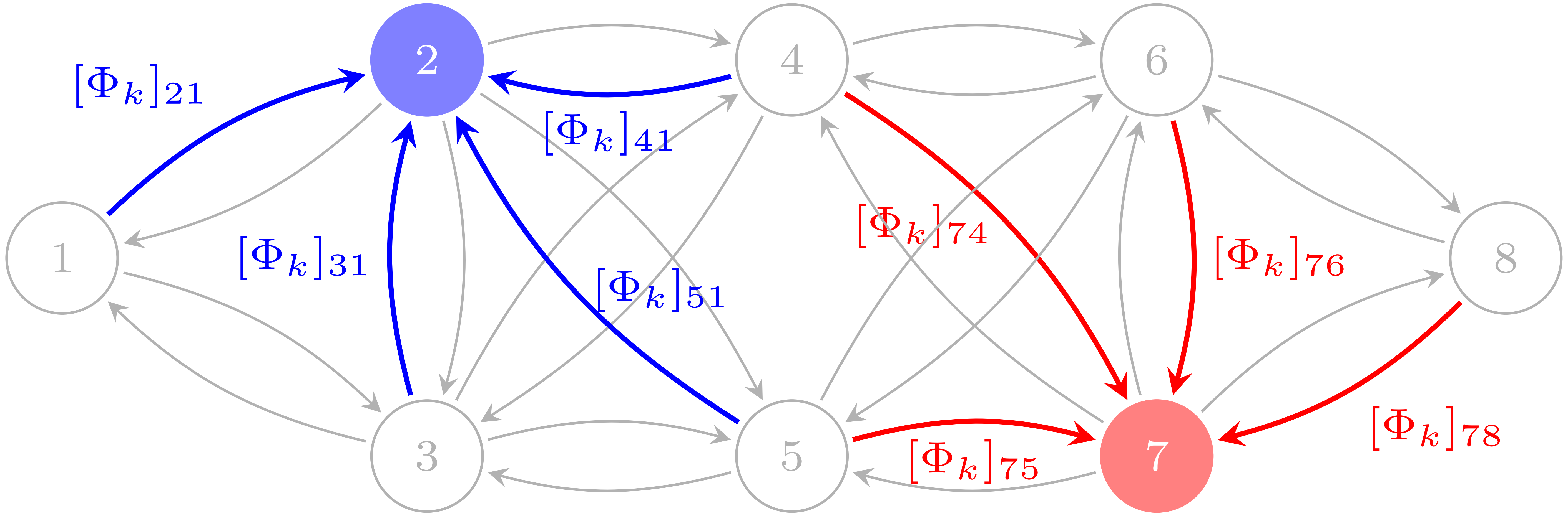
How to use EdgeNets?

- The full form may sometimes overfit
  - Penalize coefficients to sparse (i.e., $||\mathbf{\Phi}||_1$)
  - Impose parameter sharing
    - FIR : all nodes all edges same parameter
    - Node-variant : all edges same parameter for a node
    - Attention mechanism [Velickovic'18 - ICLR]
    - Hybrid : FIR + EV to particular nodes

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

How to use EdgeNets?

Example: Hybrid (FIR + EV)



- Nodes 2 and 7 use EV filter
- All other nodes use FIR filter
- More flexibility than GCNN
- Parameters independent on the graph dimensions

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

Where are GNNs useful?

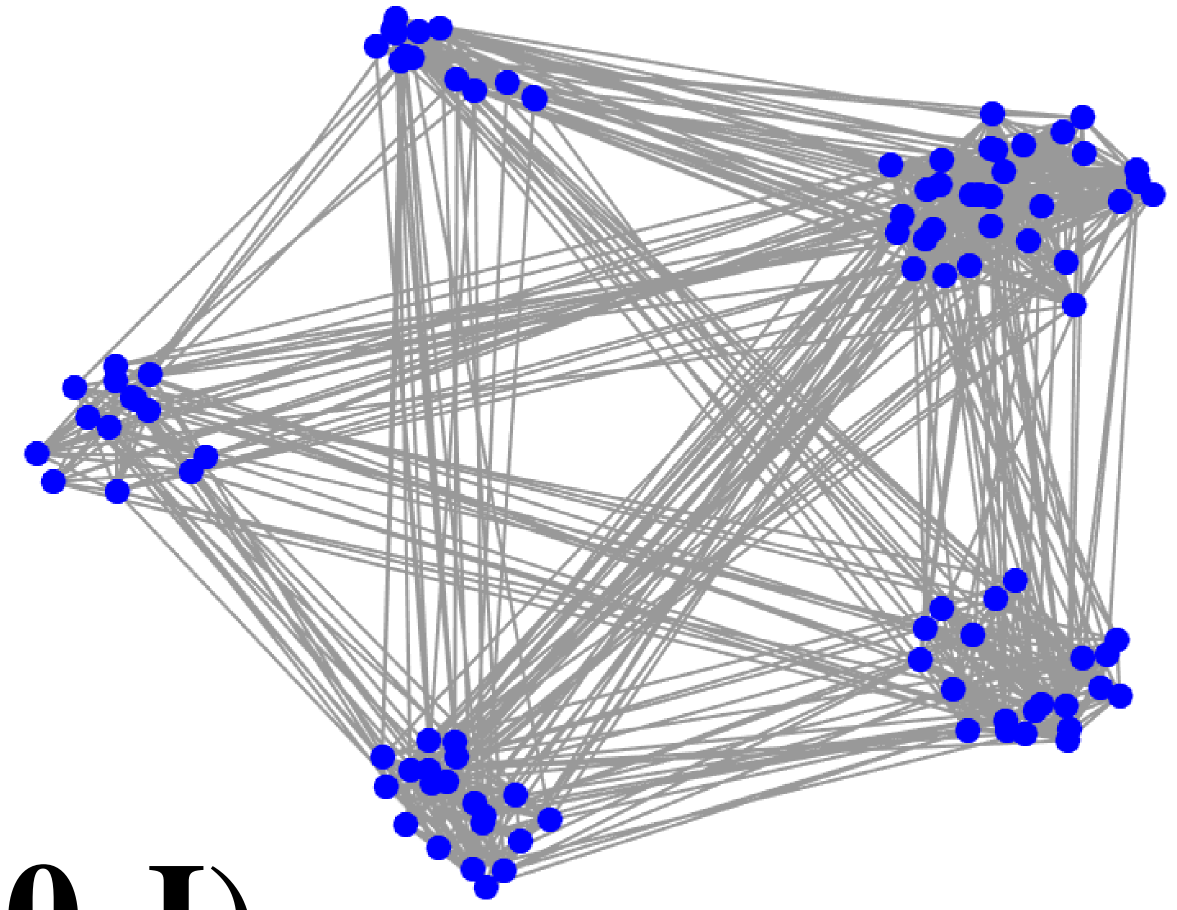
Applications

- Distributed finite-time consensus
- Distributed regression
- Authorship attribution
- Recommender systems

- For control, resource allocation and other SP applications [T-9]
- For semi-supervised learning, graph classification [Wu’20 -TNNLS]

Learning finite-time consensus

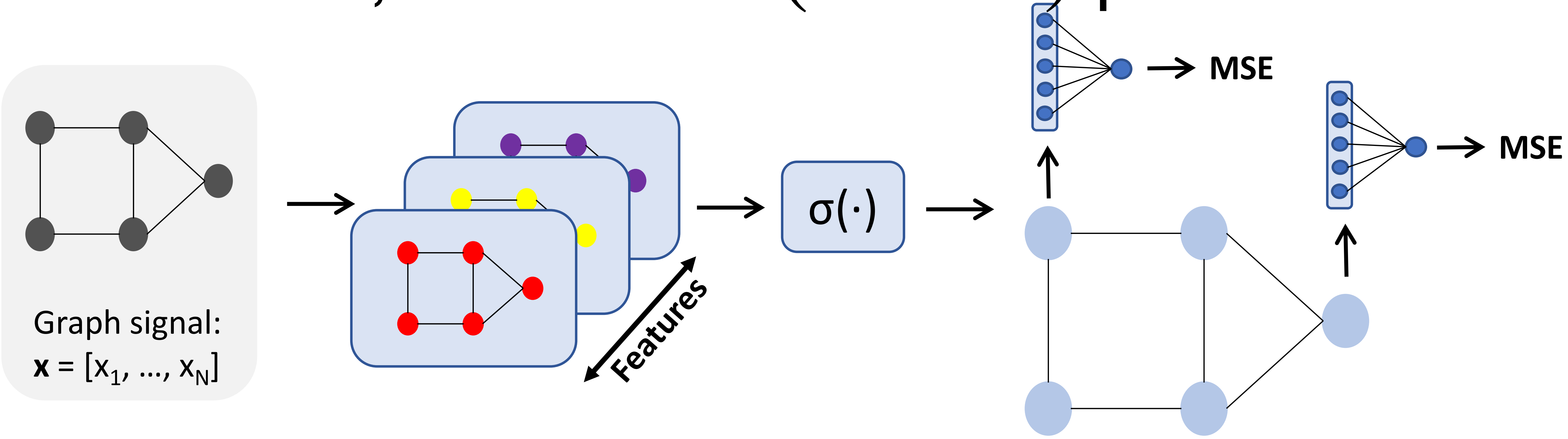
- Learn the consensus function for a specific graph
  - ✦ EV can do the job but all nodes need to know all graph
    - Feasible only in small setups



Stochastic block model

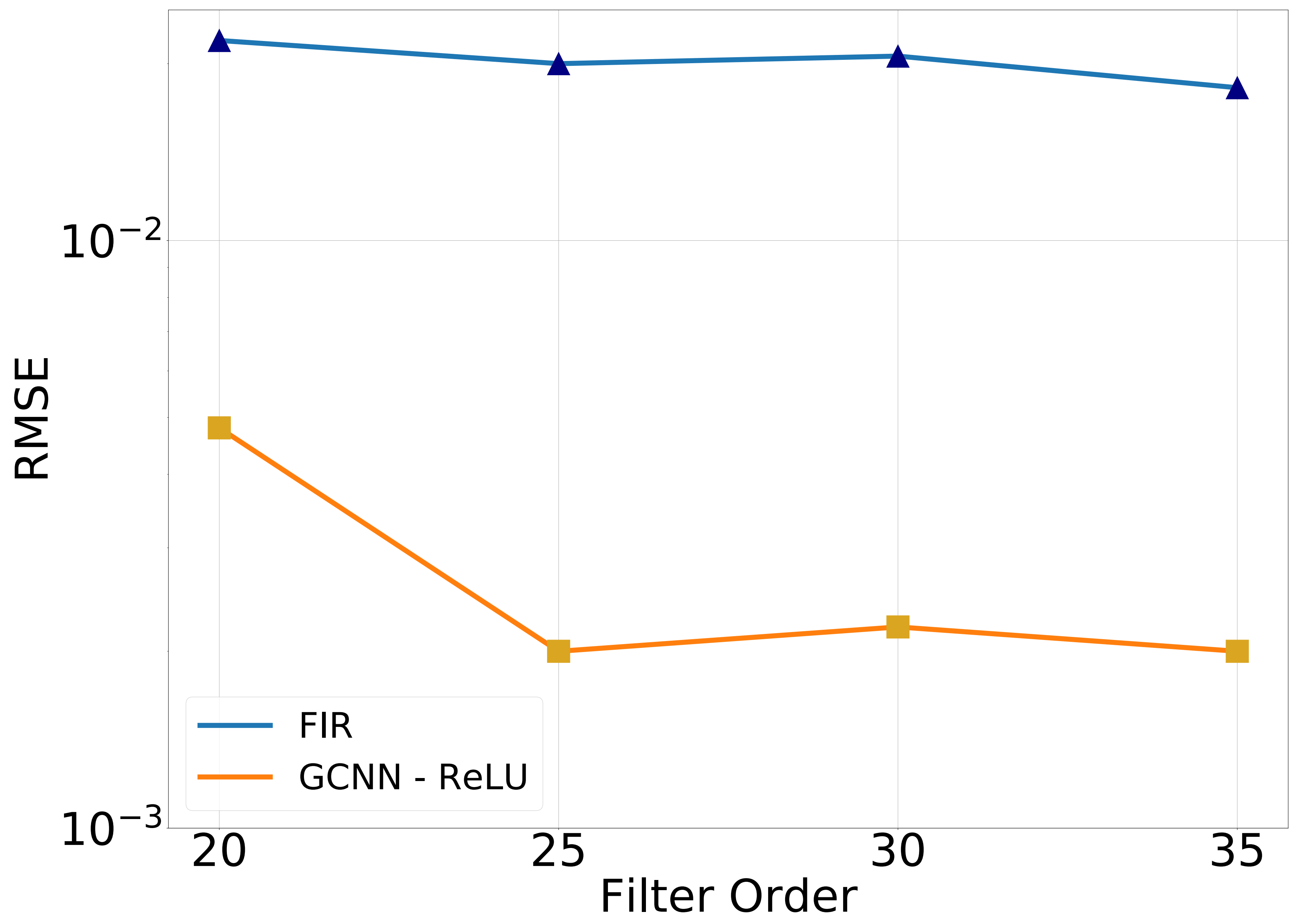
$N = 100$ and $C = 5$ communities; graph signals $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

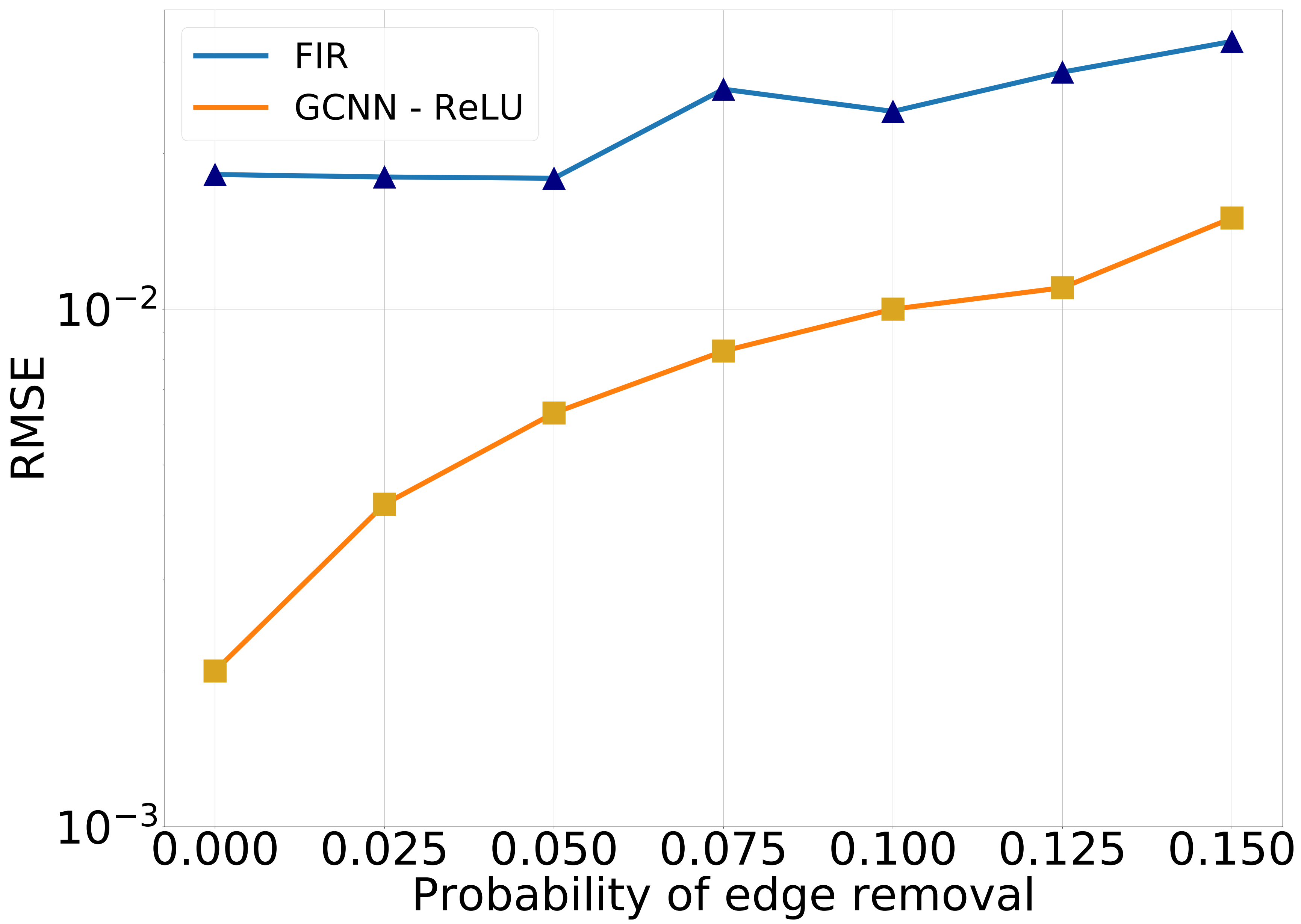
1 Layer, $F = 32$ features, shared FC (32×1) per node



Iancu, Isufi, *Towards Finite-Time Consensus with Graph Convolutional Neural Networks*, EUSIPCO 2020 (submitted)

Learning finite-time consensus





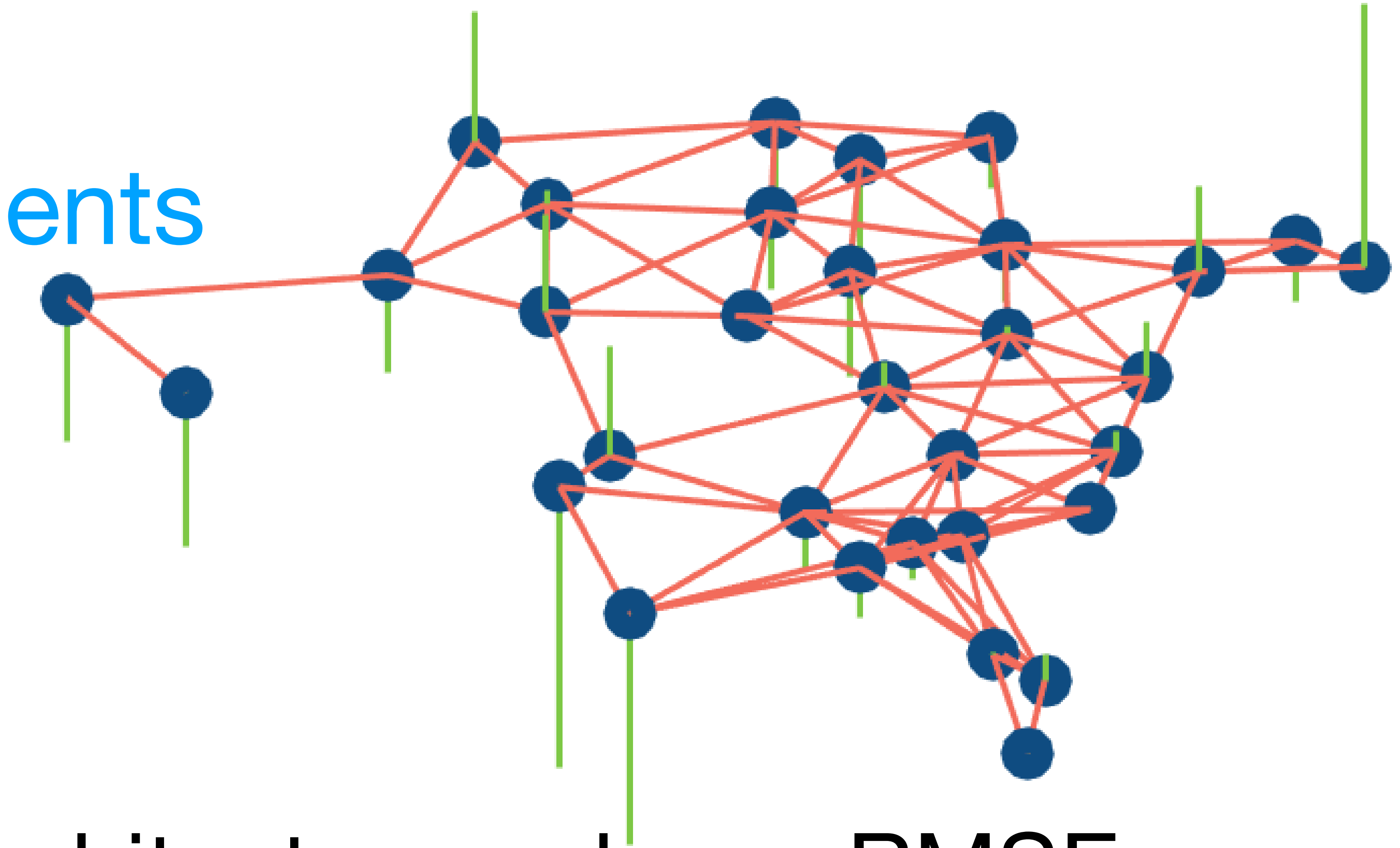
- Consensus is strictly low pass
- Better performance for high orders
- Machine precision needs EV

- Train and test on different graphs
- GCNN exploits better the connectivity
- GCNNs are better transferable

Levie, Isufi, Kutyniok, *On the Transferability of Spectral Graph Filters*, SAMPTA 2019.

Distributed regression

- Retrieve signal distributively from noisy measurements

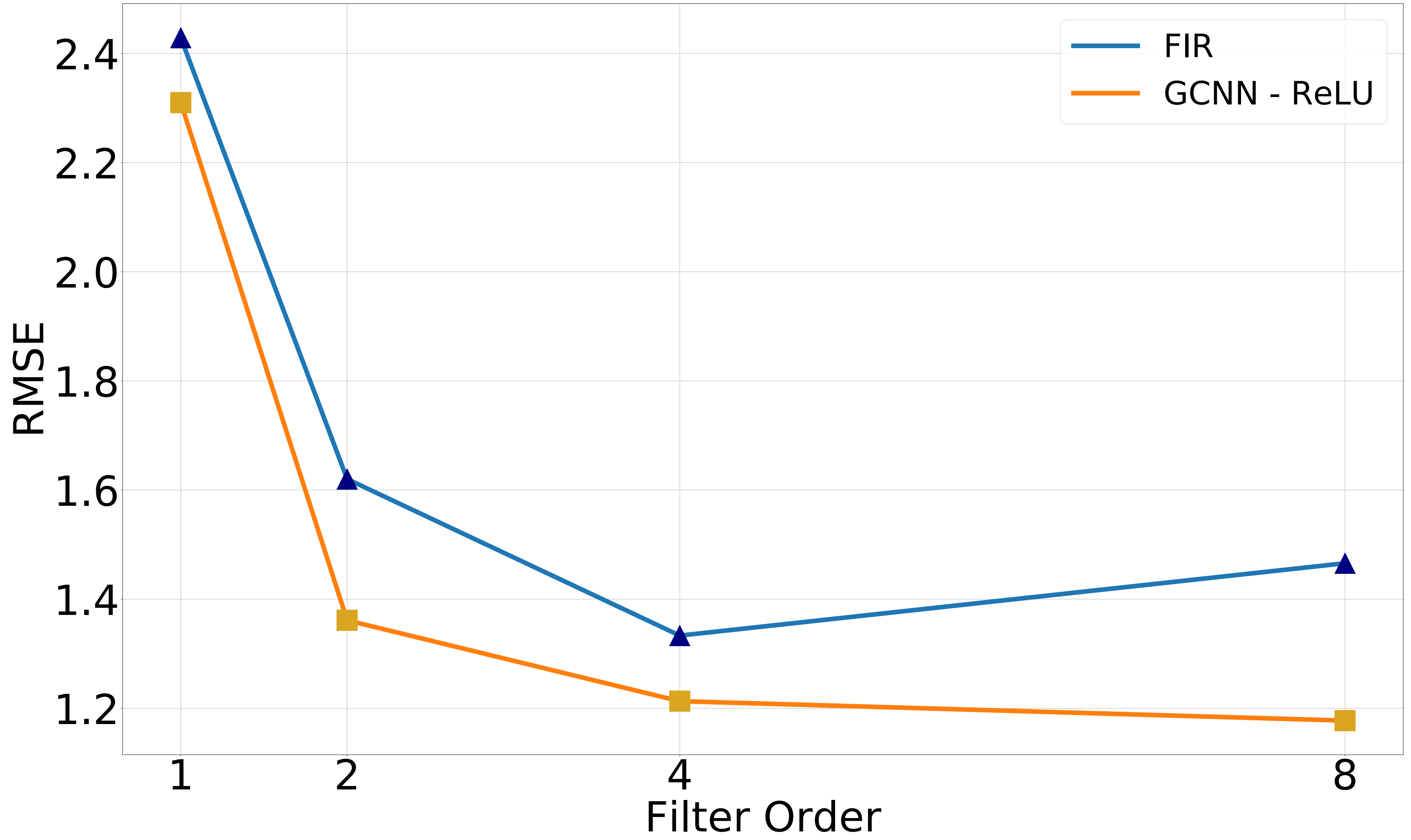


Molene weather dataset

Build a graph between stations $N = 32$

Graph signal: 744 temperature recording

$SNR = 3\text{dB}$ 1 layer; 4 features



- Nonlinear architecture reduces RMSE
- 4 times more communications
- Regression more challenging than classification
- Needs: more data/more graph prior

Iancu, Ruiz, Ribeiro Isufi, *Distributed Localized Nonlinearities For Graph Neural Networks*, MLSP 2020 (submitted)

Authorship attribution

- Attribute texts to an author [Segarra'15-TSP]

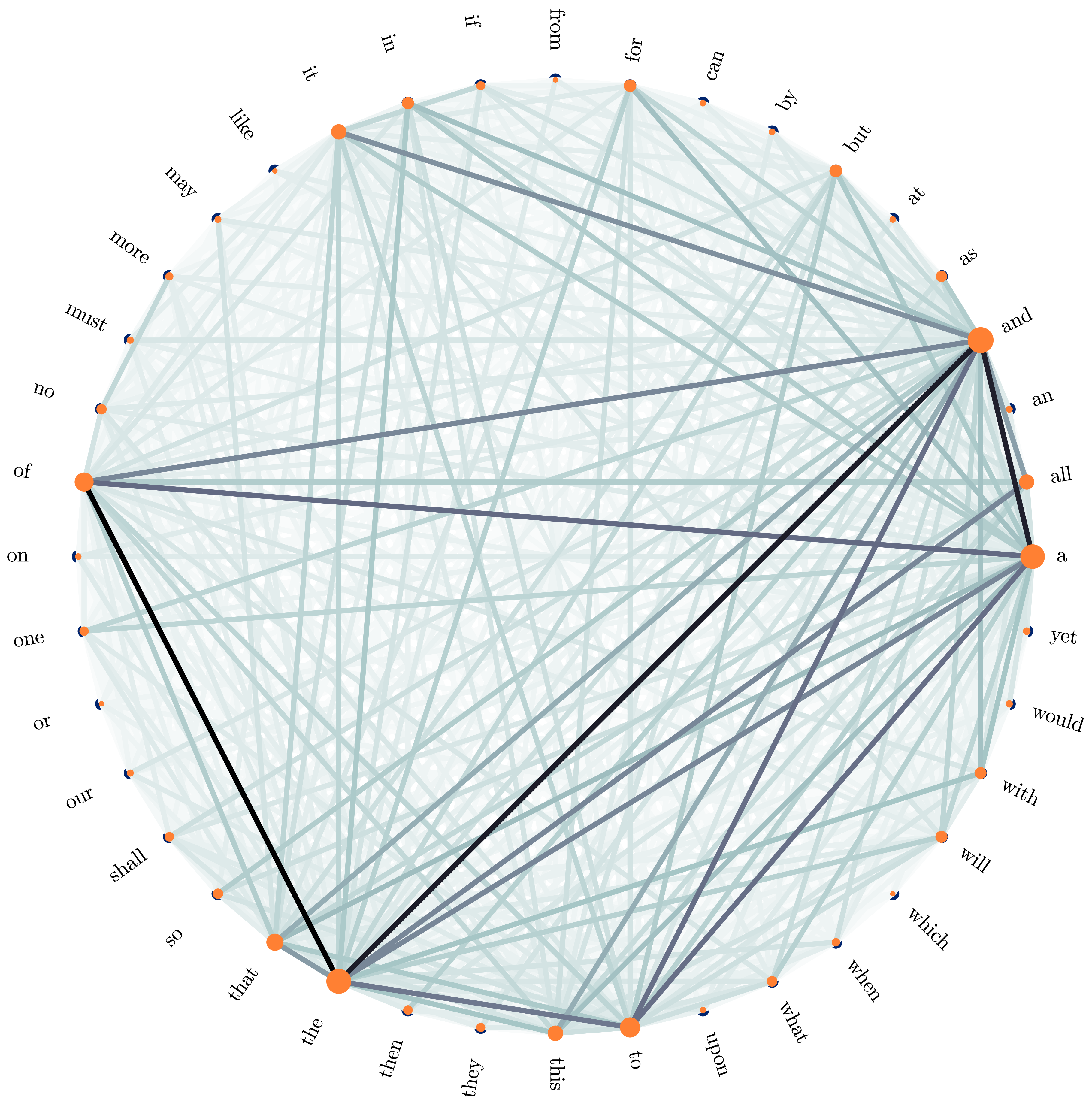
Build a word adjacency network

$N = 190 - 211$

Graph signal: word frequency count

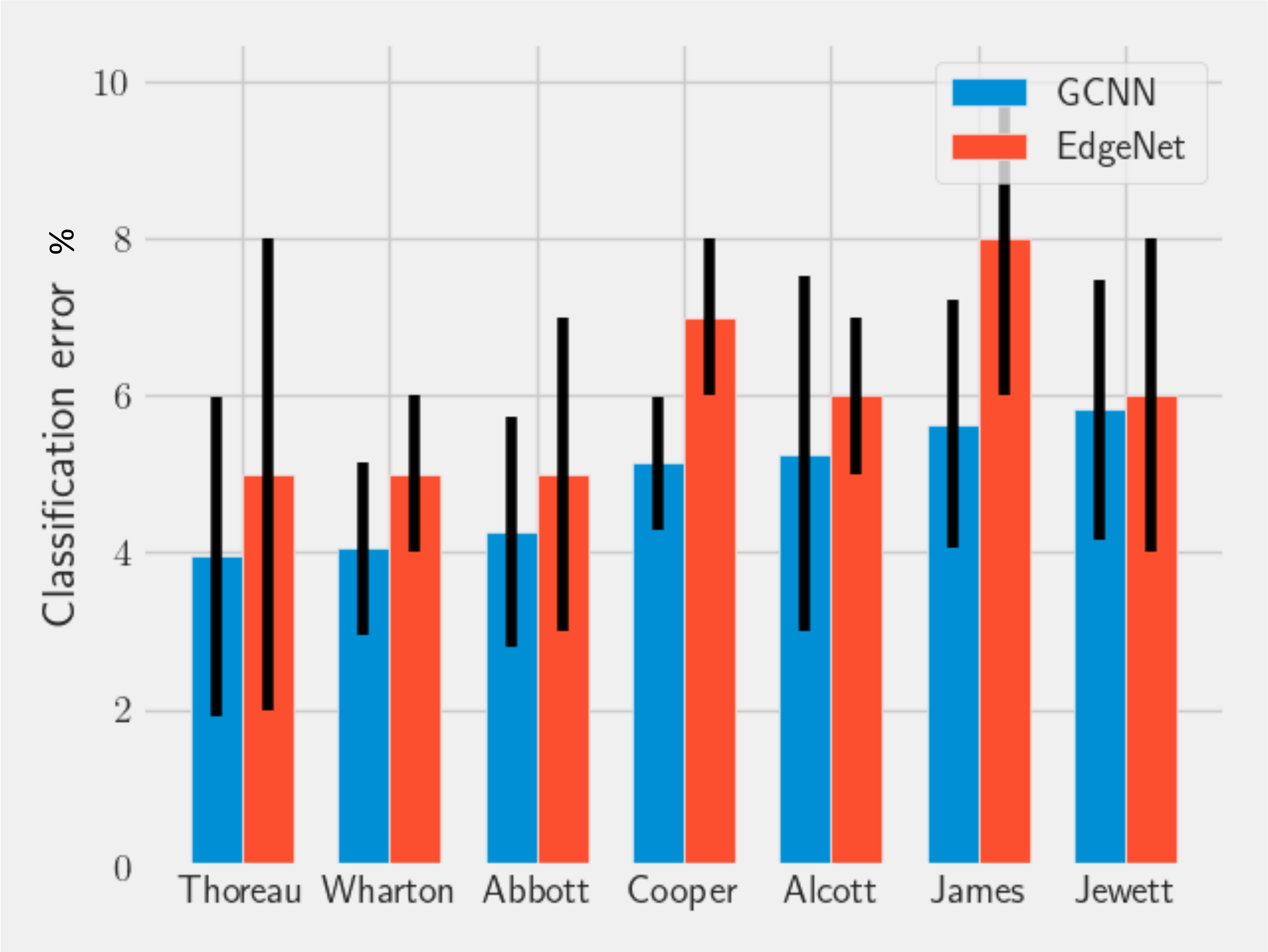
~ 1000 texts from the author of interest

~ 1000 from others



[© figure Ruiz'19-TSP]

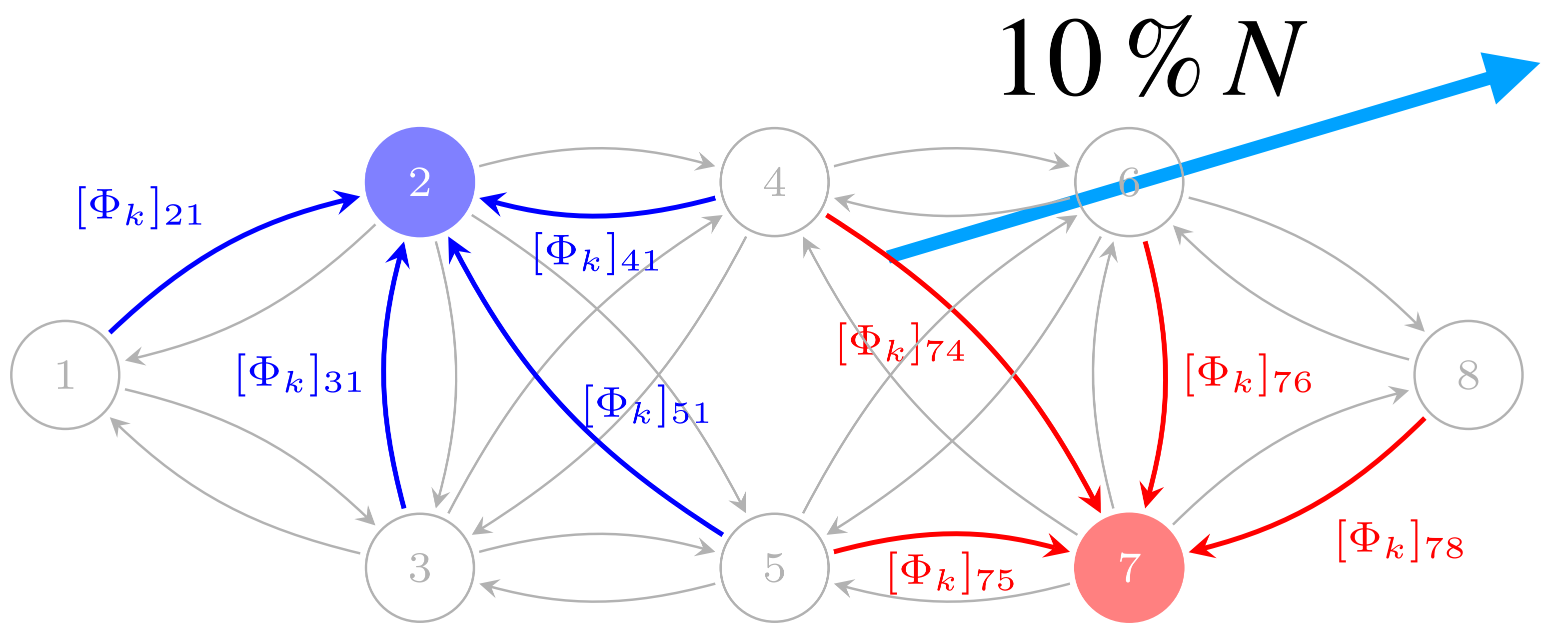
Authorship attribution (easier)



- EV hyperparameters (K, F, L) taken from the FIR
- Parameter sharing is beneficial

  1 layer, $K \in [2,10]$, $F \in \{16,32,64\}$

Authorship attribution (difficult)



Classification error

Architecture	Austen	Brontë	Poe
GCNN	7.2(±2.0)%	12.9(±3.5)%	14.3(±6.4)%
Edge varying	7.1(±2.2)%	13.1(±3.9)%	**10.7**(±**4.3**)%
Node varying	7.4(±2.1)%	14.6(±4.2)%	11.7(±4.9)%
Hybrid edge var.	**6.9**(±**2.6**)%	14.0(±3.7)%	11.7(±4.8)%
ARMANet	7.9(±2.3)%	**11.6**(±**5.0**)%	10.9(±3.7)%

1 layer, $F = 32$, $K = 4$

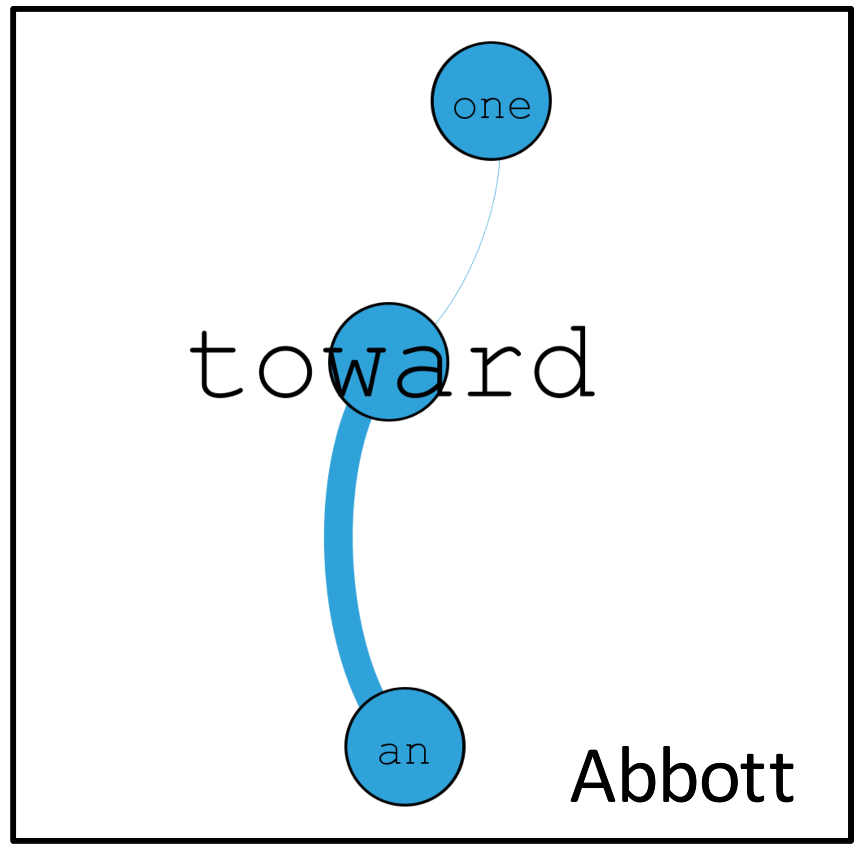
- EdgeNet requires its own hypertunning
- Better for more difficult scenarios
- Subclasses of the EV can perform better depending on problem difficulty

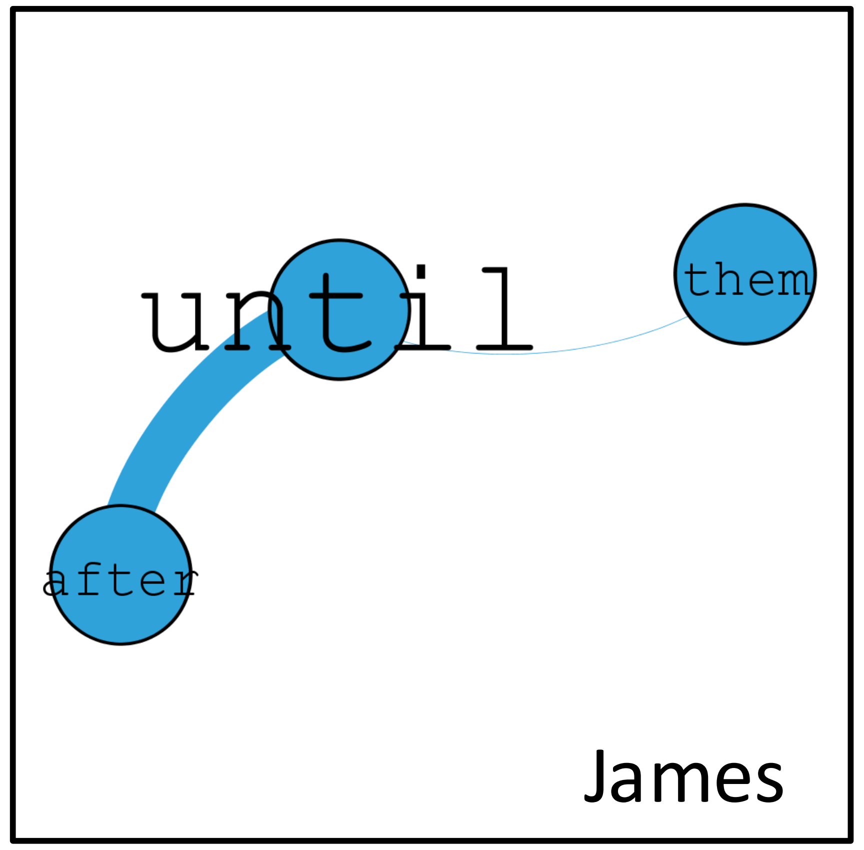
Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

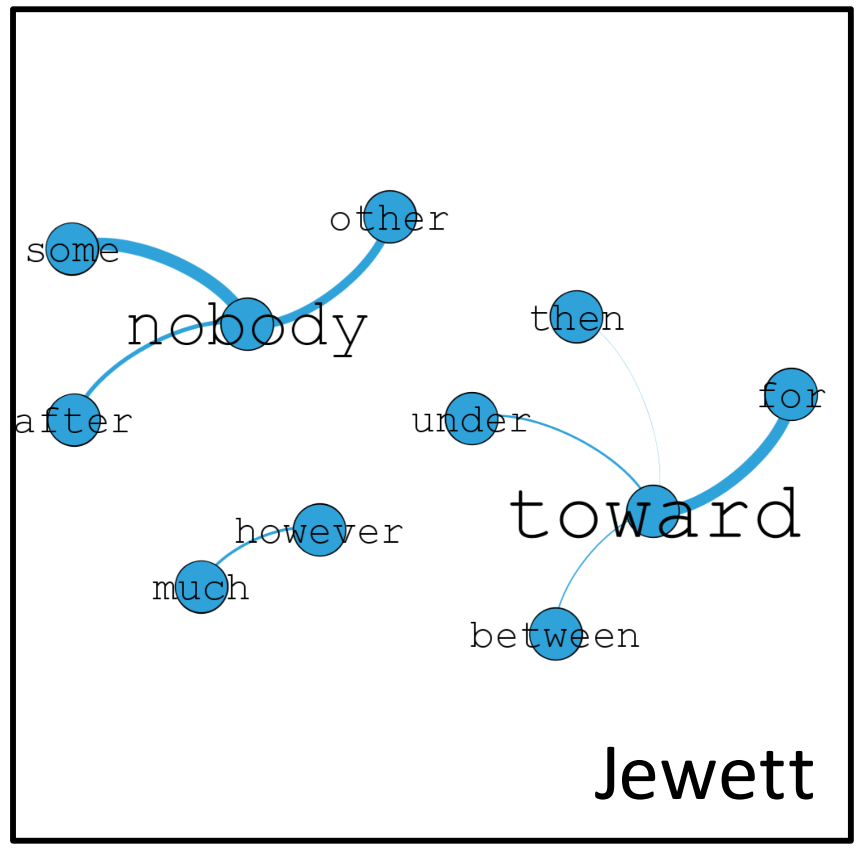
Authorship attribution (explain)

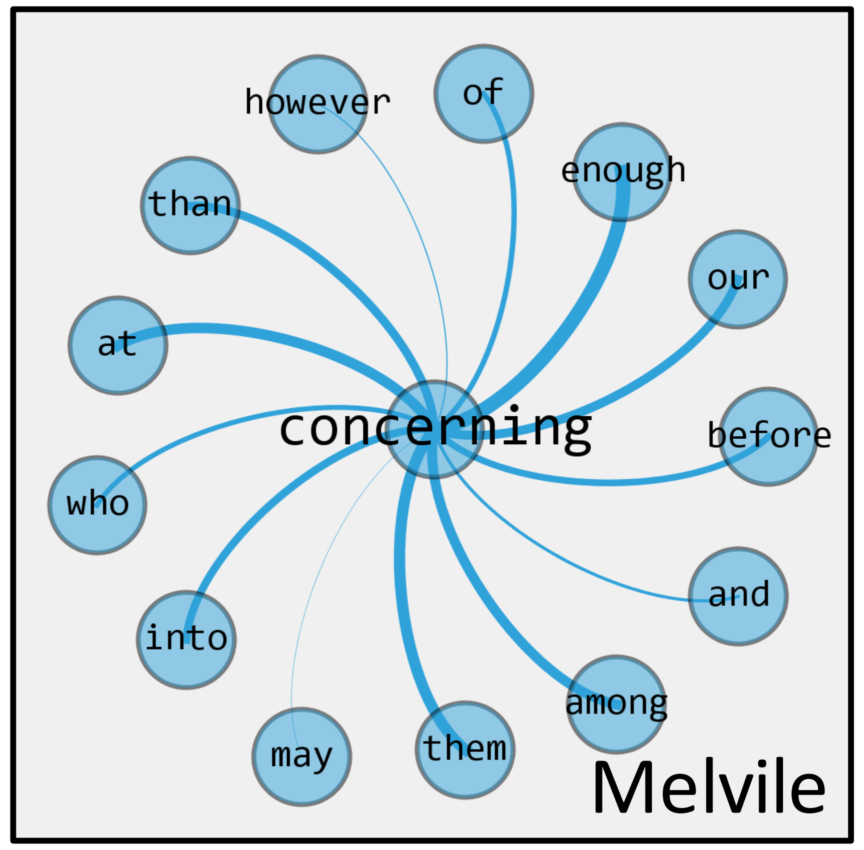
- Explain GNNs with EdgeNets
  - One layer EdgeNet with order $K = 1$
  - Training the EdgeNet = learning graph weights
    - removed small weight edges = accuracy drop $< 5\,\%$
    - identifies most relevant function words per author

$$\mathbf{x}_1^1 = \sigma\left(\mathbf{\Phi}\mathbf{x}_1^0\right)$$









- identify an author from 3 words

Authorship attribution

- Identifying author gender from texts
  - No NLP: shallow and fast training, no pretraining/corpus
  - Graphs + signals from female and male authors in train - test

Classification error

	EdgeNet	GCNN	EV-GCNN
Mean	**8.6%**	**10.1%**	**7.8%**
Std	**$6x10^{-3}$**	**$6x10^{-3}$**	**$5x10^{\wedge-3}$**

1 layer architectures, $F = 64$

Sparse WANs help classification

Sparse EV shift operator + GCNN

Recommender systems

- ◉ Fill missing entries in a user-item matrix

Movielens $100K$ dataset $U = 943; I = 1{,}582$

Build a similarity graph (principle of collaborative filter)

	Item 1	Item 2	Item 3	…	Item I
User 1	5	1	?	…	2
User 2	?	?	3	…	3
User 3	4	?	4	…	?
…	…	…	…	…	…
User U	?	4	3	…	1

user similarity graph

nodes : users

edges : Pearson/cosine similarity

between pairs of users

item similarity graph

nodes : items

edges : Pearson/cosine similarity

between pairs of items

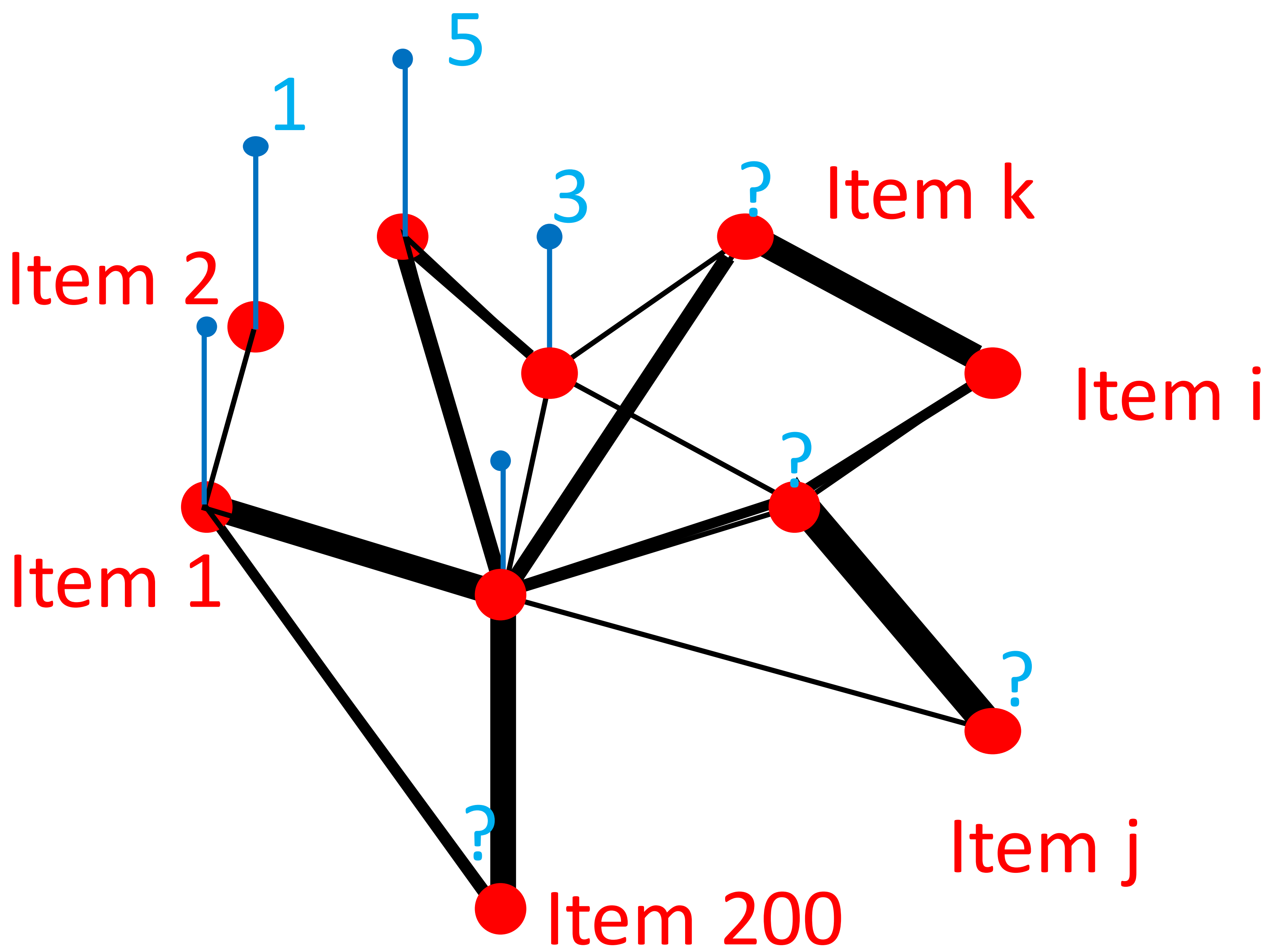
Recommender systems

- Here item similarity graph

$N = 200$ most rated items

subset of user ratings
to build the graph

	Item 1	Item 2	Item 3	…	Item I
User 1	5	1	?	…	2
User 2	?	?	3	…	3
User 3	4	?	4	…	?
…	…	…	…	…	…
User U	?	4	3	…	1



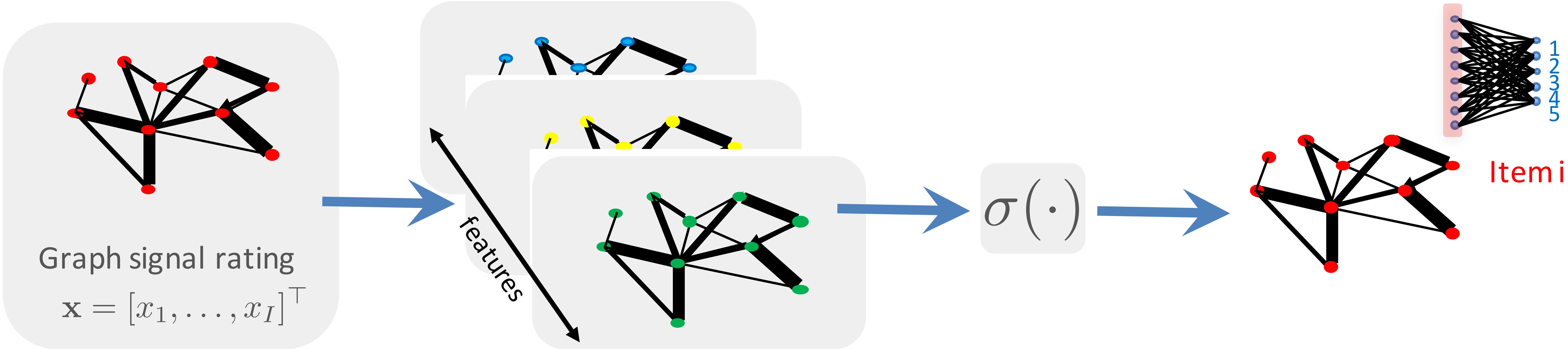
Graph signal : rating of user u to all items

- interpolation problem on graphs

Goal: find rating all users give to item i (fii ith column of matrix)

Recommender systems

- Use locality of the filters to build a GNN specific to item I



Frame as signal classification problem per node

1 layer, 32 features

EdgeNet suffers in general -requires parameterization

Archit./Movie-ID	50	258	100	181	294	Average
GCNN	**0.82**	1.08	**0.95**	0.86	1.04	0.95
Edge var.	0.93	**1.03**	1.00	0.88	1.24	1.02
Node var.	0.78	1.04	1.00	0.87	**1.00**	**0.94**
Hybrid edge var.	0.75	**1.02**	0.98	**0.82**	1.08	**0.93**

part 4 :: conclusions

- Graph filter are the building block of graph neural network (GNN)
  - Incorporate effectively the graph signal - graph topology into learning
  - Serve as a prior to reduce parameters and complexity
  - Graph convolutions through graph filters
- Different filter = different graph neural networks
  - FIR = GCNNs
  - ARMA = ARMANets
  - Edge varying = EdgeNets

part 4 :: conclusions

- EdgeNets provide the broadest GNN family
  - Particularize to all the others including GINs and GATs
  - Help explainability
- Applications in signal classification & regression
  - Authorship attribution
  - Recommender systems

GNN - next challenges

- More graph prior instead of more data
- Explainability
  - What topological information is more relevant?
  - What spectral information is more relevant?
  - EdgeNet can be a strong tool in this regard
- Robustness/Transferability
  - To topological perturbations
  - To input perturbations
- Distributed learning
  - Graph filters are distributable

part 4
graph neural networks

part 4:: overview

- Role of graph filters in graph neural networks (GNNs)
  - GNNs ~ nonlinear graph filters
- For simplicity will discuss supervised learning

part 4:: overview

- Role of graph filters in graph neural networks (GNNs)
  - GNNs ~ nonlinear graph filters
- For simplicity will discuss supervised learning
- How to go from neural networks to GNNs?
- Types of GNNs
  - What are graph convolutional neural networks?
  - How to use edge varying GNNs?
- How to use GNNs for graph signal processing applications?

part 4:: overview

- Role of graph filters in graph neural networks (GNNs)
  - GNNs ~ nonlinear graph filters
- For simplicity will discuss supervised learning
- How to go from neural networks to GNNs?
- Types of GNNs
  - What are graph convolutional neural networks?
  - How to use edge varying GNNs?
- How to use GNNs for graph signal processing applications?
- For GNN pooling, transferability, and applications in control and resource allocation
  - T-9: Graph Neural Networks (F. Gama and A. Ribeiro)

Why we use filters in neural networks?

Supervised learning

- Relies on a dataset of R training examples
  $\mathscr{R} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_R, y_R)\}$
  - $\mathbf{x}_r$ the rth input data in space $\mathscr{X}$
  - y_r the rth output data in space $\mathscr{Y}$ (labels)

Supervised learning

- Relies on a dataset of R training examples
  $\mathscr{R} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_R, y_R)\}$
  - ✦ $\mathbf{x}_r$ the rth input data in space $\mathscr{X}$
  - ✦ y_r the rth output data in space $\mathscr{Y}$ (labels)
- Goal: learn a function f that maps $\mathbf{x}_r$ to y_r
  - we want f parametric: $f(\boldsymbol{\theta}) : \mathscr{X} \to \mathscr{Y}$

Supervised learning

- ◉ Deign parameters $\boldsymbol{\theta}$ such that
  - ✦minimize a cost distance between $f(\boldsymbol{\theta}, \mathbf{x}_r)$ and y_r (e.g., MSE)

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \frac{1}{R} \sum_{r=1}^{R} \left(f(\boldsymbol{\theta}, \mathbf{x}_r) - y_r\right)^2$$

  - ✦generalize well for test data $\mathbf{x}_r \notin \mathscr{R}$

Neural networks

- Express function f as a cascade of layered functions

$$f(\boldsymbol{\theta}, \mathbf{x}) = f^3(\boldsymbol{\theta}^3, f^2(\boldsymbol{\theta}^2, f^1(\boldsymbol{\theta}^1, \mathbf{x})))$$

layer 1: parameters $\boldsymbol{\theta}^1$
layer 2: parameters $\boldsymbol{\theta}^2$
layer 3: parameters $\boldsymbol{\theta}^3$

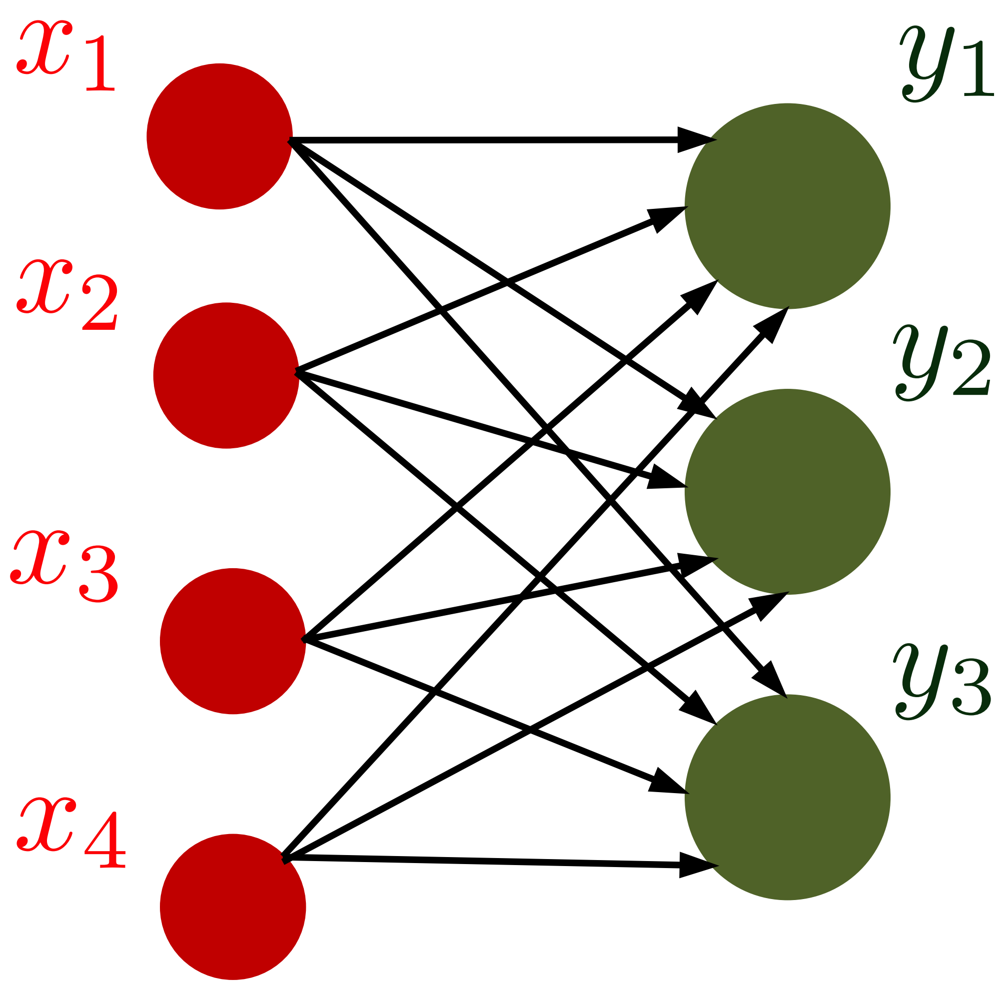
Neural networks

- Express function f as a cascade of layered functions

$$f(\boldsymbol{\theta}, \mathbf{x}) = f^3(\boldsymbol{\theta}^3, f^2(\boldsymbol{\theta}^2, f^1(\boldsymbol{\theta}^1, \mathbf{x})))$$

layer 1: parameters $\boldsymbol{\theta}^1$
layer 2: parameters $\boldsymbol{\theta}^2$
layer 3: parameters $\boldsymbol{\theta}^3$

- No structure in the data: perceptron



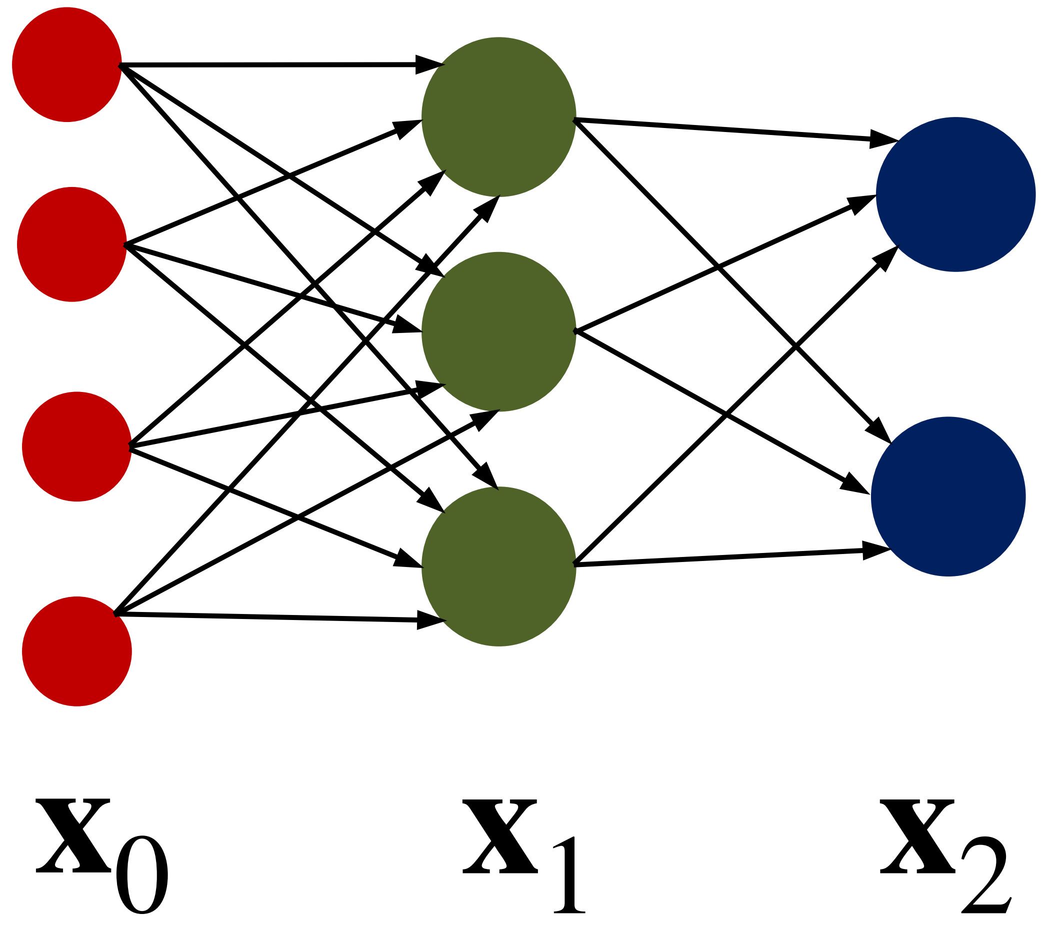
$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- Parameters $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}\}$
- Pointwise nonlinearity $\sigma(\,\cdot\,)$

$$\mathrm{ReLU}(x) = \begin{cases} x & x > 0 \\ 0 & \mathrm{otw} \end{cases}$$

Neural networks

- No structure in the data: multi-layer perceptron
  - Improves expressivity



- Input features $\mathbf{x}^0 = \mathbf{x}_r$
- Output features $\mathbf{x}^L$

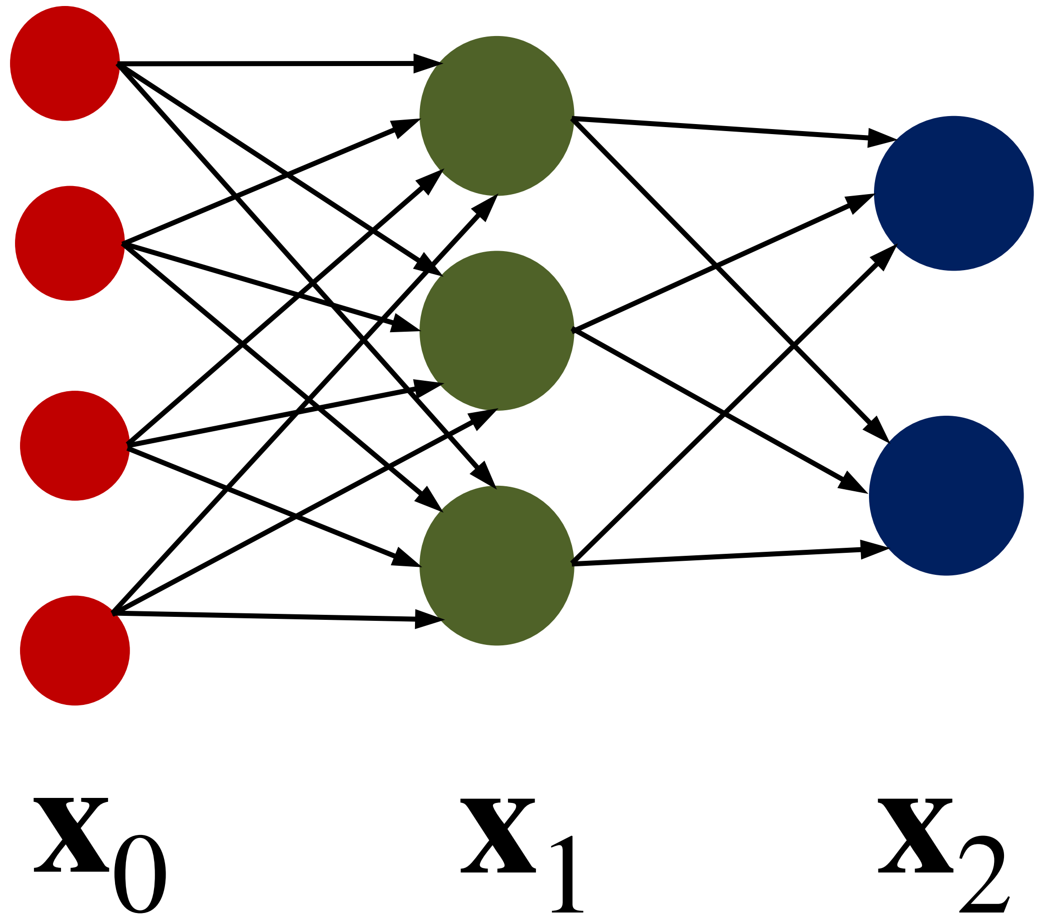
$$\mathbf{x}^l = \sigma(\mathbf{W}^l\mathbf{x}^{l-1} + \mathbf{b}^l)$$

$\mathbf{x}^{l-1}$: input layer l = output layer $l-1$
$\mathbf{x}^l$: output layer l

$\boldsymbol{\theta}^l = \{\mathbf{W}^l, \mathbf{b}^l\}$: parameters layer l

Neural networks

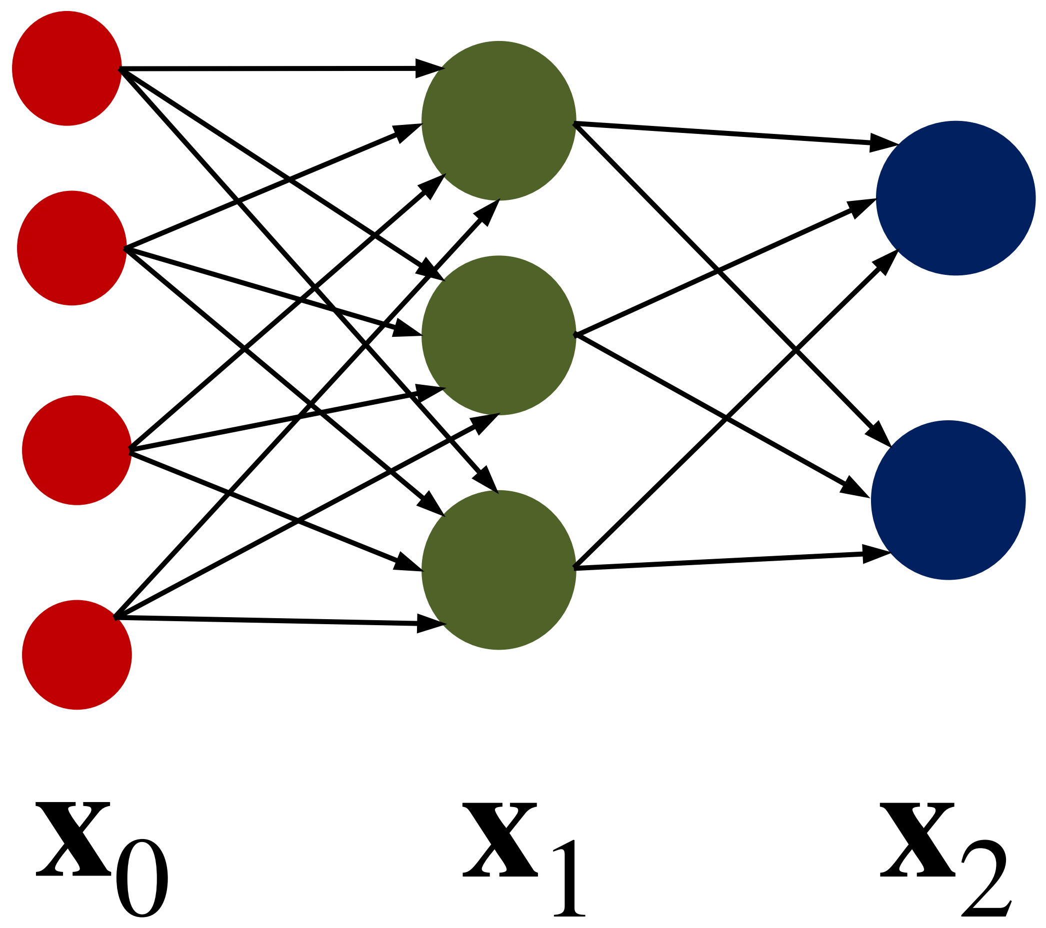
- Unrolling recursion $\mathbf{x}^L = \sigma(\mathbf{W}^L \mathbf{x}^{L-1} + \mathbf{b}^L)$



$$\begin{aligned}\mathbf{x}^L &= \sigma\left(\mathbf{W}^L \mathbf{x}^{L-1} + \mathbf{b}^L\right) \\ &= \sigma\left(\mathbf{W}^L \sigma\left(\mathbf{W}^{L-1} \mathbf{x}^{L-2} + \mathbf{b}^{L-1}\right) + \mathbf{b}^L\right)\end{aligned}$$

Neural networks

- Unrolling recursion $\mathbf{x}^L = \sigma(\mathbf{W}^L \mathbf{x}^{L-1} + \mathbf{b}^L)$



$$\begin{aligned}\mathbf{x}^L &= \sigma\left(\mathbf{W}^L \mathbf{x}^{L-1} + \mathbf{b}^L\right)\\ &= \sigma\left(\mathbf{W}^L \sigma\left(\mathbf{W}^{L-1}\mathbf{x}^{L-2} + \mathbf{b}^{L-1}\right) + \mathbf{b}^L\right)\\ &= \sigma\left(\mathbf{W}^L \sigma\left(\mathbf{W}^{L-1}\sigma\left(\ldots\sigma\left(\mathbf{W}^1\mathbf{x}^0 + \mathbf{b}^1\right) + \mathbf{b}^{L-1}\right) + \mathbf{b}^L\right)\right)\end{aligned}$$

- $\mathbf{x}^L$ depends on $\mathbf{x}^0$ through a composition of linear functions and pointwise nonlinearities

Neural networks

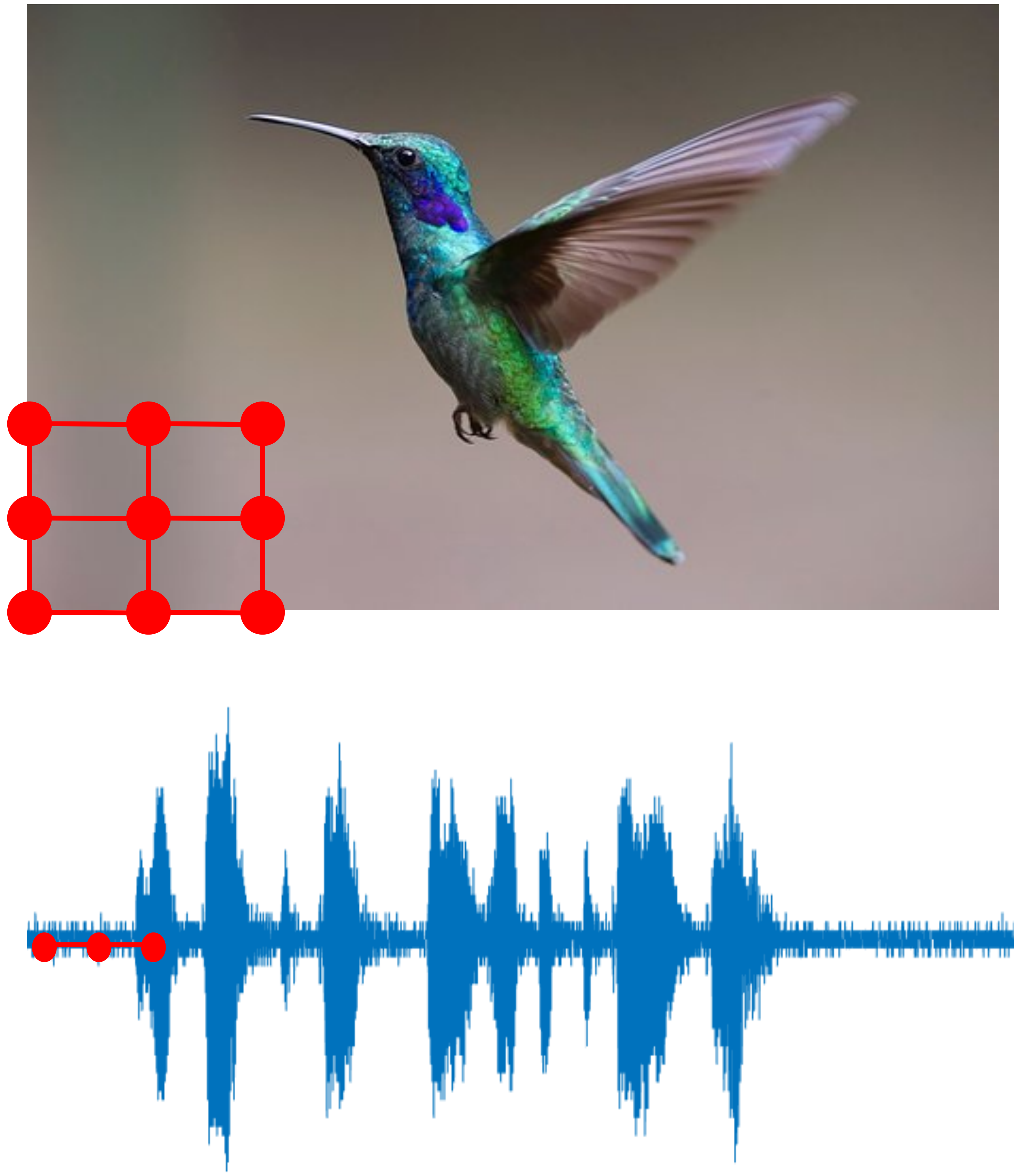
- MLP fails in high dimensional data $\mathbf{x}^l = \sigma\left(\mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l\right)$
  - if layers have dimensions $\dim(\mathbf{x}^l) = \dim(\mathbf{x}^{l-1}) \sim \mathcal{O}(N)$
    - $\dim\left(\mathbf{W}^l\right) \sim \mathcal{O}(N^2)$ parameters, e.g., $N = 1000 \rightarrow \mathcal{O}(10^6)$
    - complexity $\mathcal{O}(N^2)$

Neural networks

- MLP fails in high dimensional data $\mathbf{x}^l = \sigma\left(\mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l\right)$
  - if layers have dimensions $\dim(\mathbf{x}^l) = \dim(\mathbf{x}^{l-1}) \sim \mathcal{O}(N)$
    - $\dim\left(\mathbf{W}^l\right) \sim \mathcal{O}(N^2)$ parameters, e.g., $N = 1000 \rightarrow \mathcal{O}(10^6)$
    - complexity $\mathcal{O}(N^2)$
- need to exploit structure in data

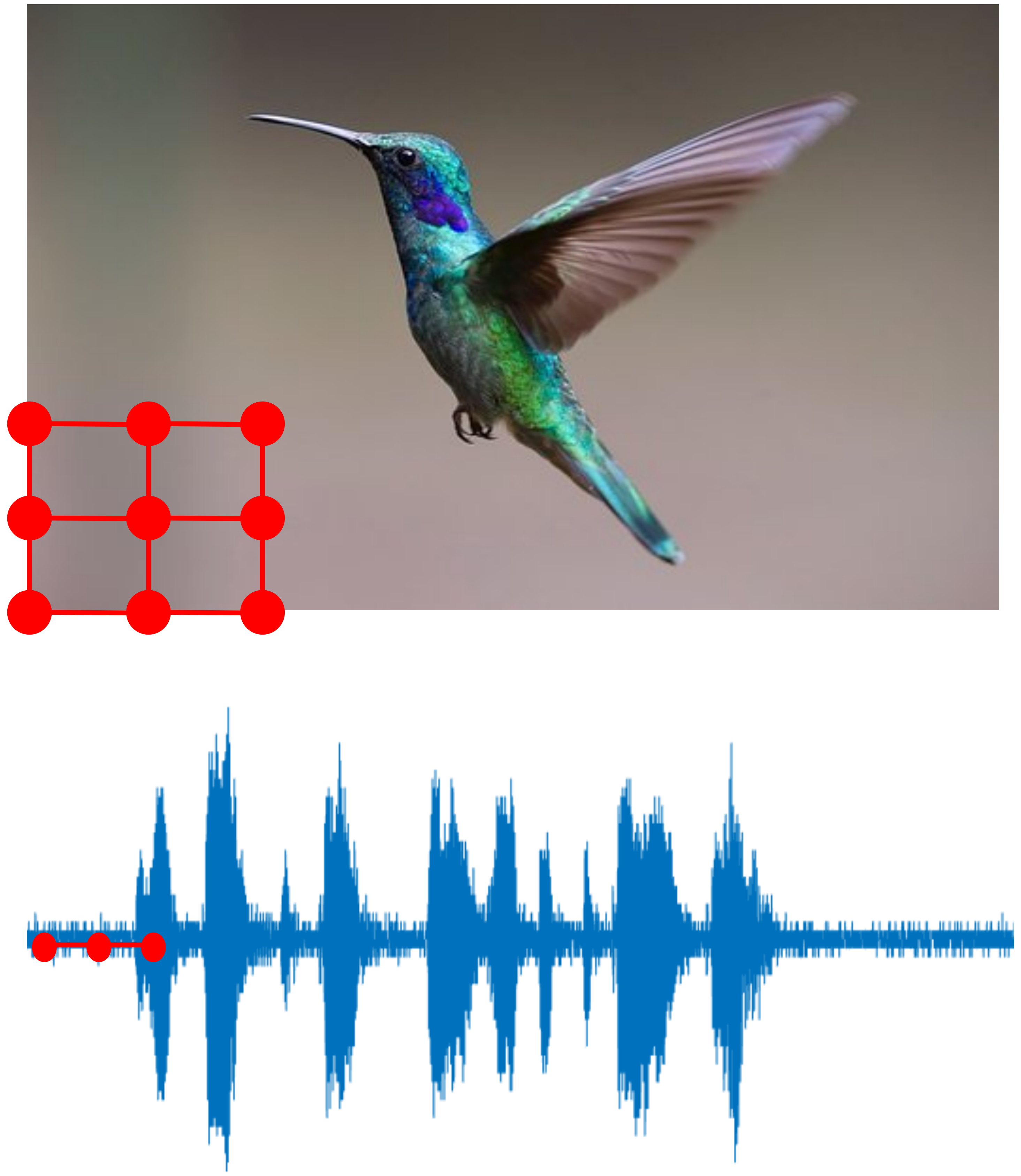
Neural networks

- structure in data
    - spatial data: pixel neighbors
    - temporal data: signal proximity



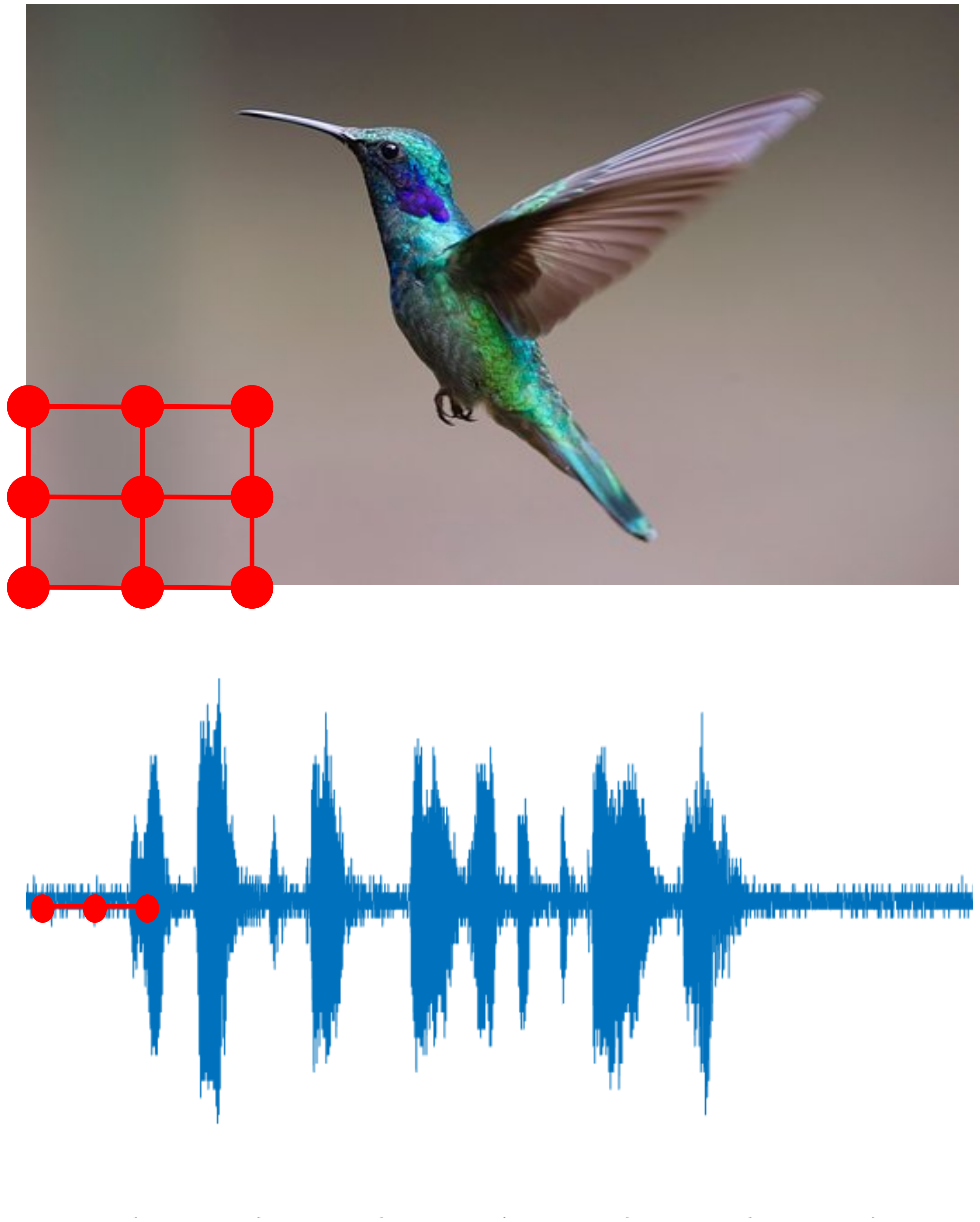
Neural networks

- structure in data
  - spatial data: pixel neighbors
  - temporal data: signal proximity
- reduce parameters by effective sharing
- reduce complexity by efficient implementation



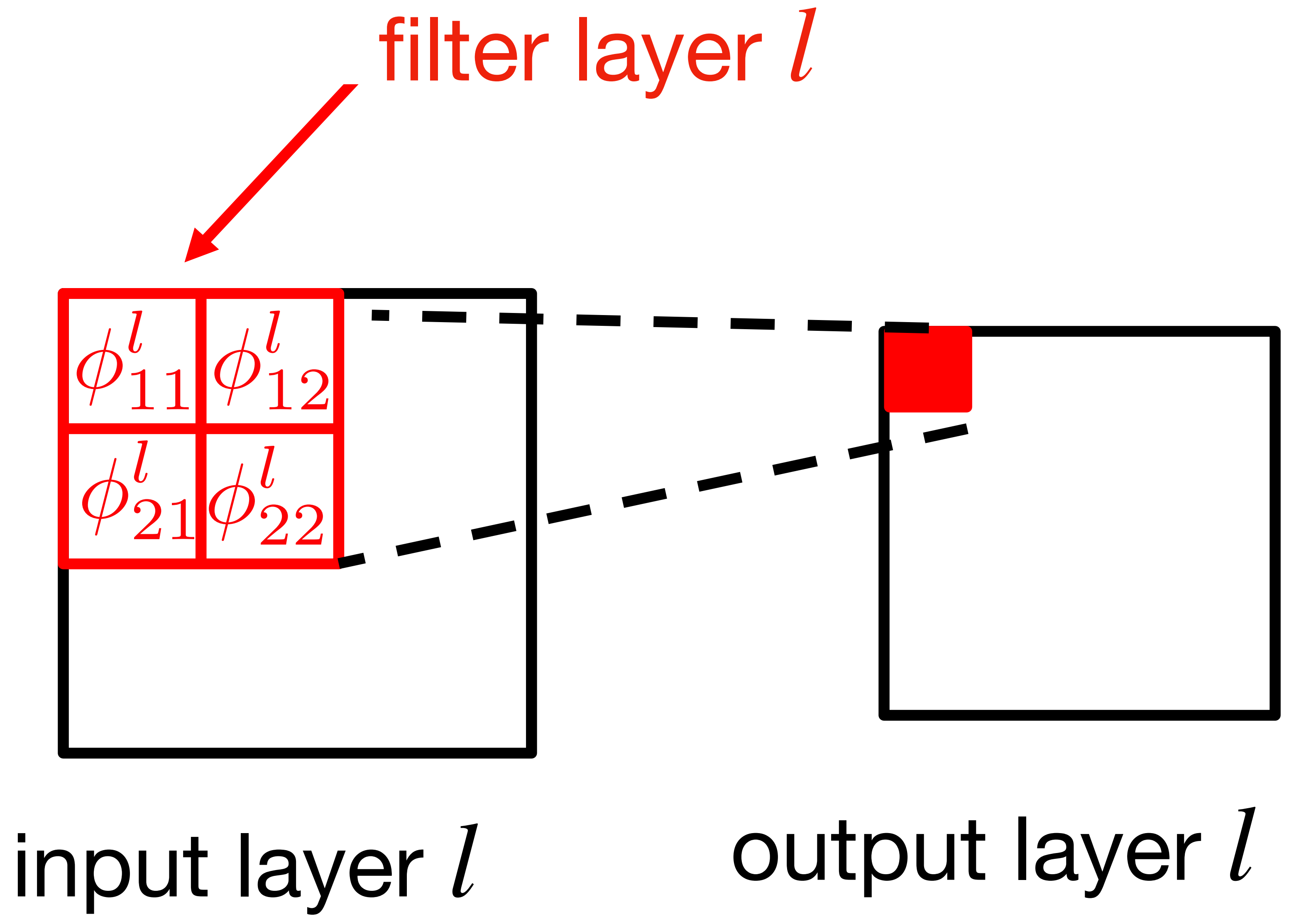
Neural networks

- structure in data
  - spatial data: pixel neighbors
  - temporal data: signal proximity
- reduce parameters by effective sharing
- reduce complexity by efficient implementation
- use spatial and temporal filters
  - no loose of discriminatory power



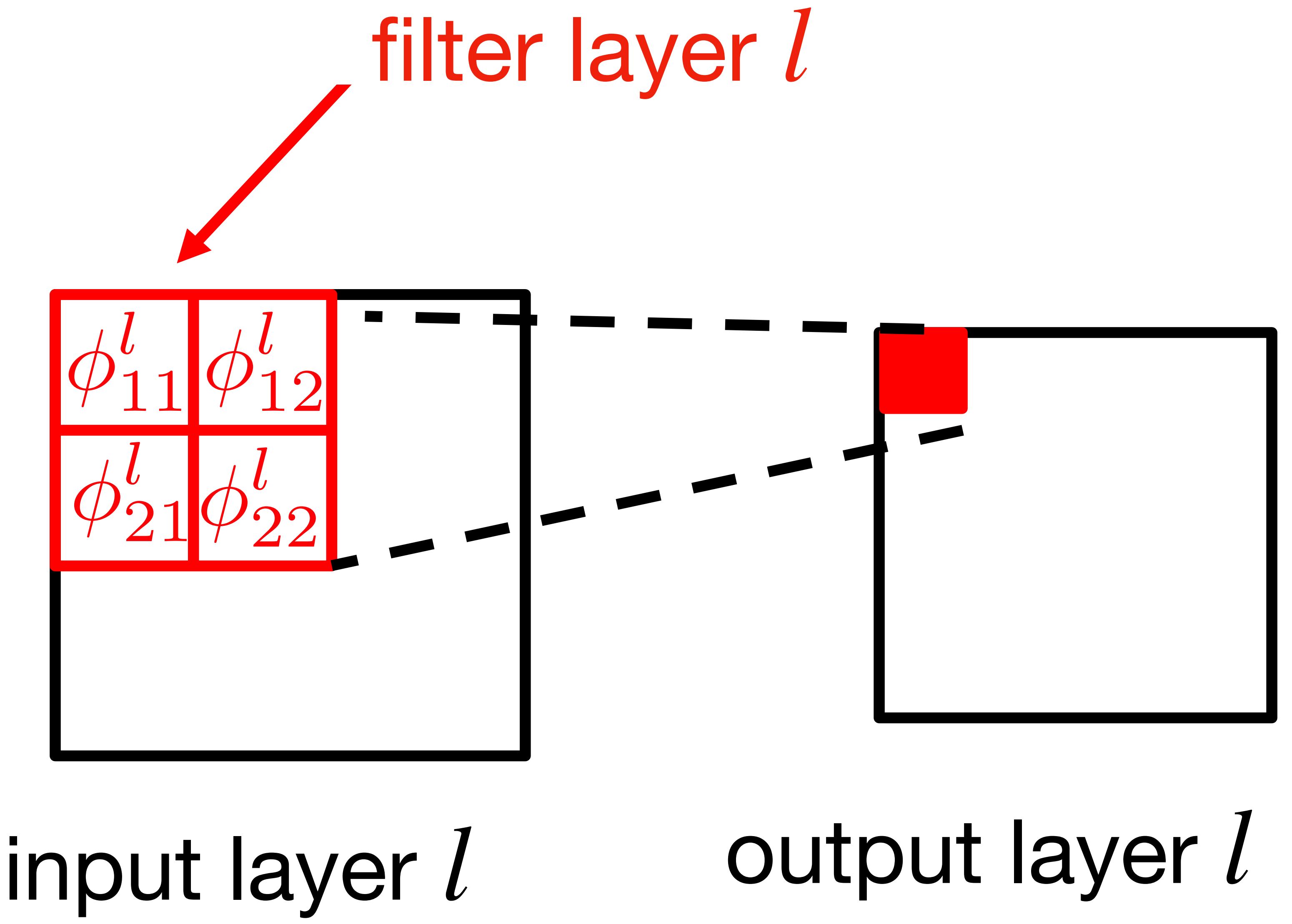
Filters in spatial convolutional layer

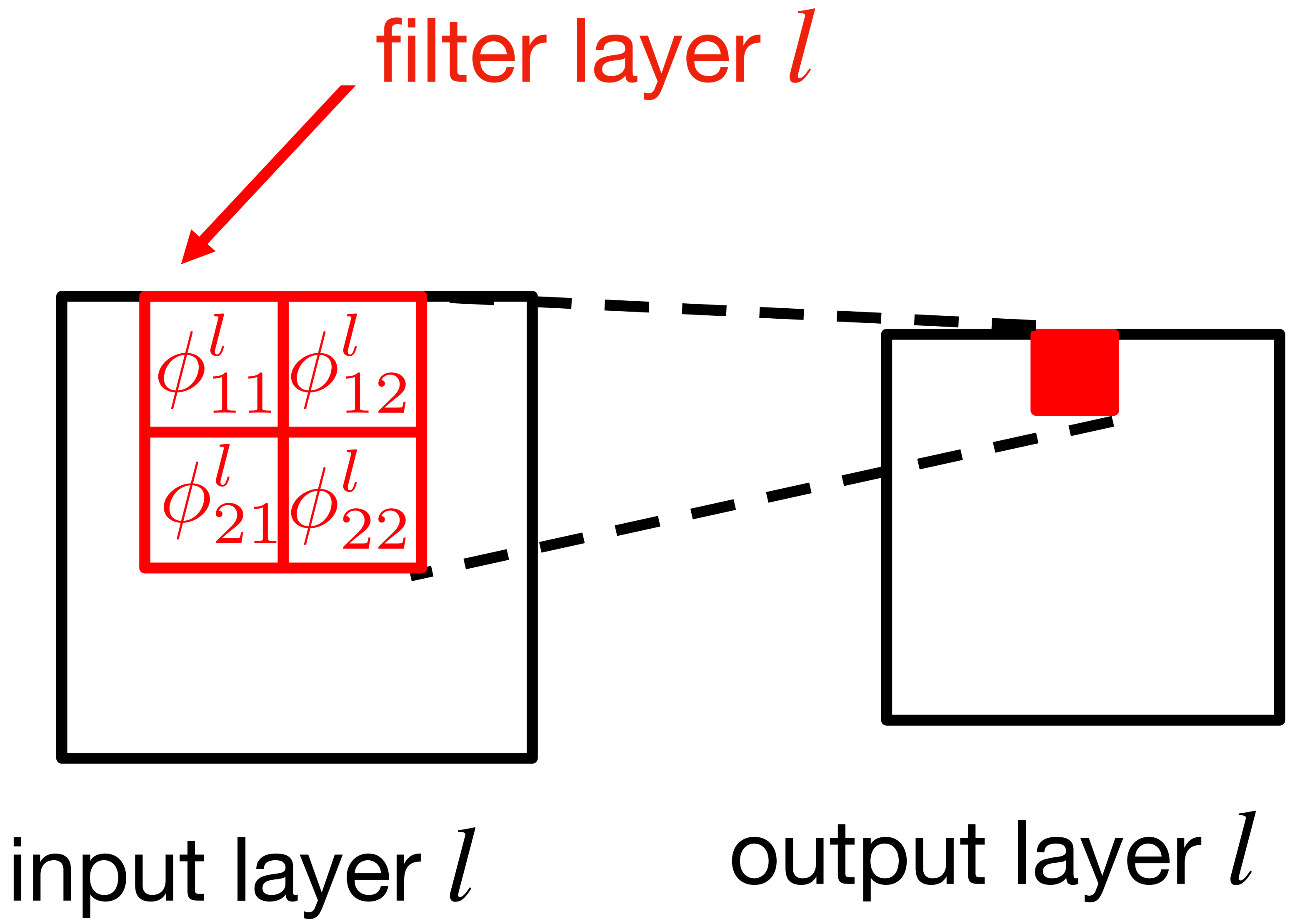
- MLP propagation rule $\mathbf{x}^l = \sigma\left(\mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l\right)$
- Spatial data: spatial convolution filter bank substitutes $\mathbf{W}^l$
  - filters apply the same parameters to different locations
  - bias $\mathbf{b}^l$ can be ignored or shared $\mathbf{b}^l = b^l \mathbf{1}$



Filters in spatial convolutional layer

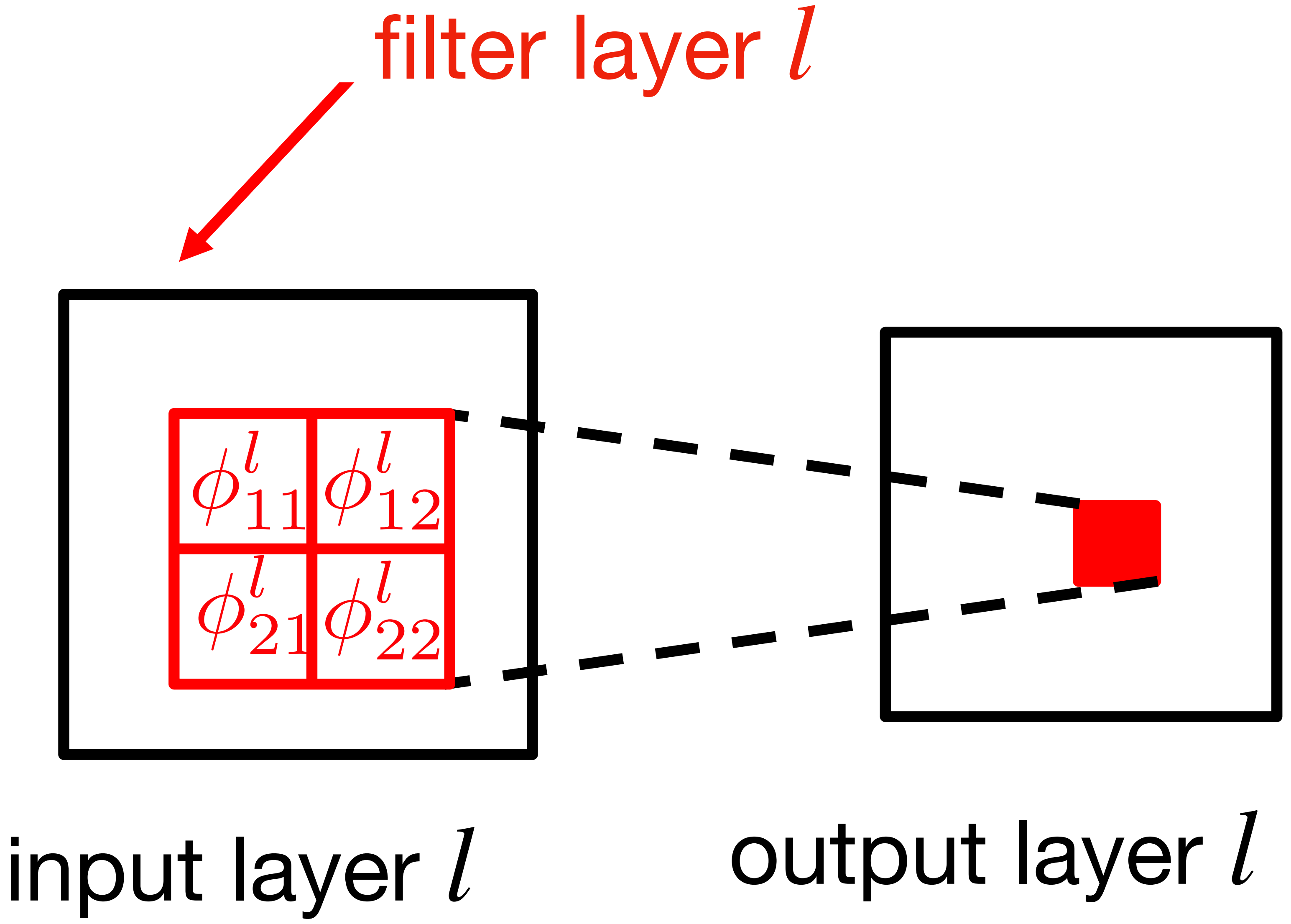
- MLP propagation rule $\mathbf{x}^l = \sigma\big(\mathbf{W}^l\mathbf{x}^{l-1} + \mathbf{b}^l\big)$
- Spatial data: spatial convolution filter bank substitutes $\mathbf{W}^l$
  - filters apply the same parameters to different locations
  - bias $\mathbf{b}^l$ can be ignored or shared $\mathbf{b}^l = b^l\mathbf{1}$





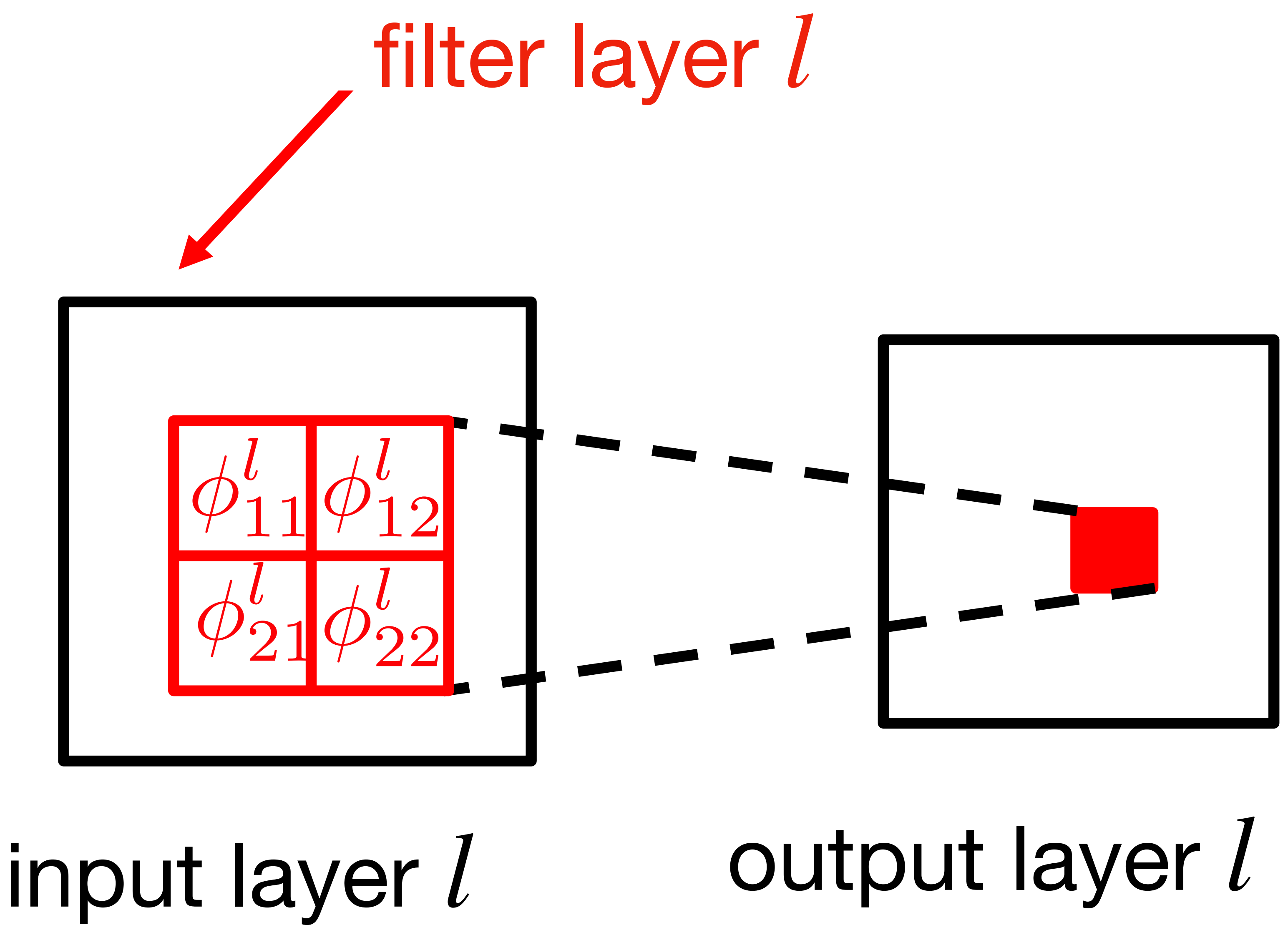
Filters in spatial convolutional layer

- shift-and-sum convolves filter with input image



Filters in spatial convolutional layer

- shift-and-sum convolves filter with input image

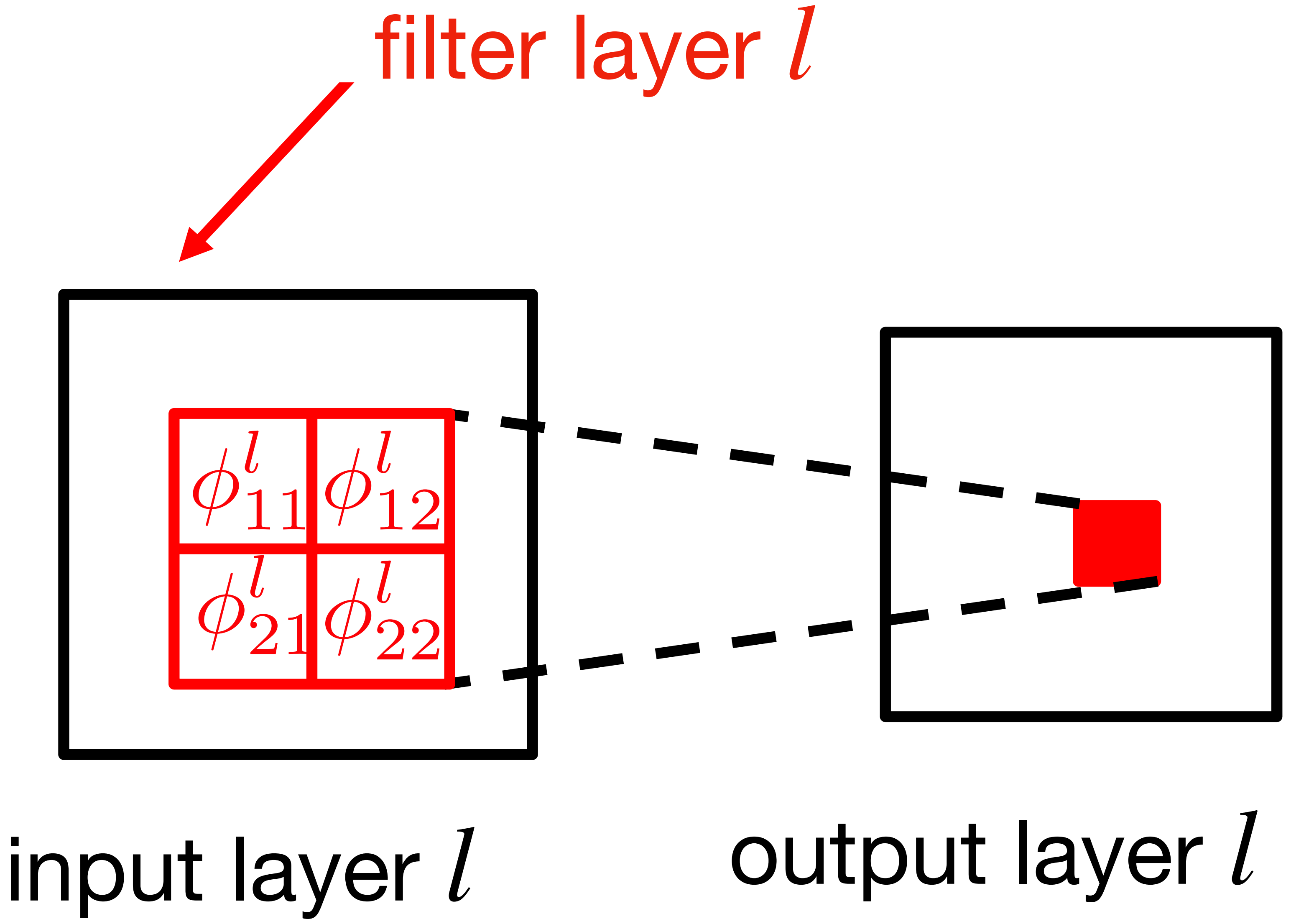


$$y_{ij}^l = \sum_{r=1}^{R} \sum_{c=1}^{C} \phi_{rc}^l x_{i-r,j-c}^{l-1}$$

vertical and horizontal input shifts

Filters in spatial convolutional layer

- shift-and-sum convolves filter with input image



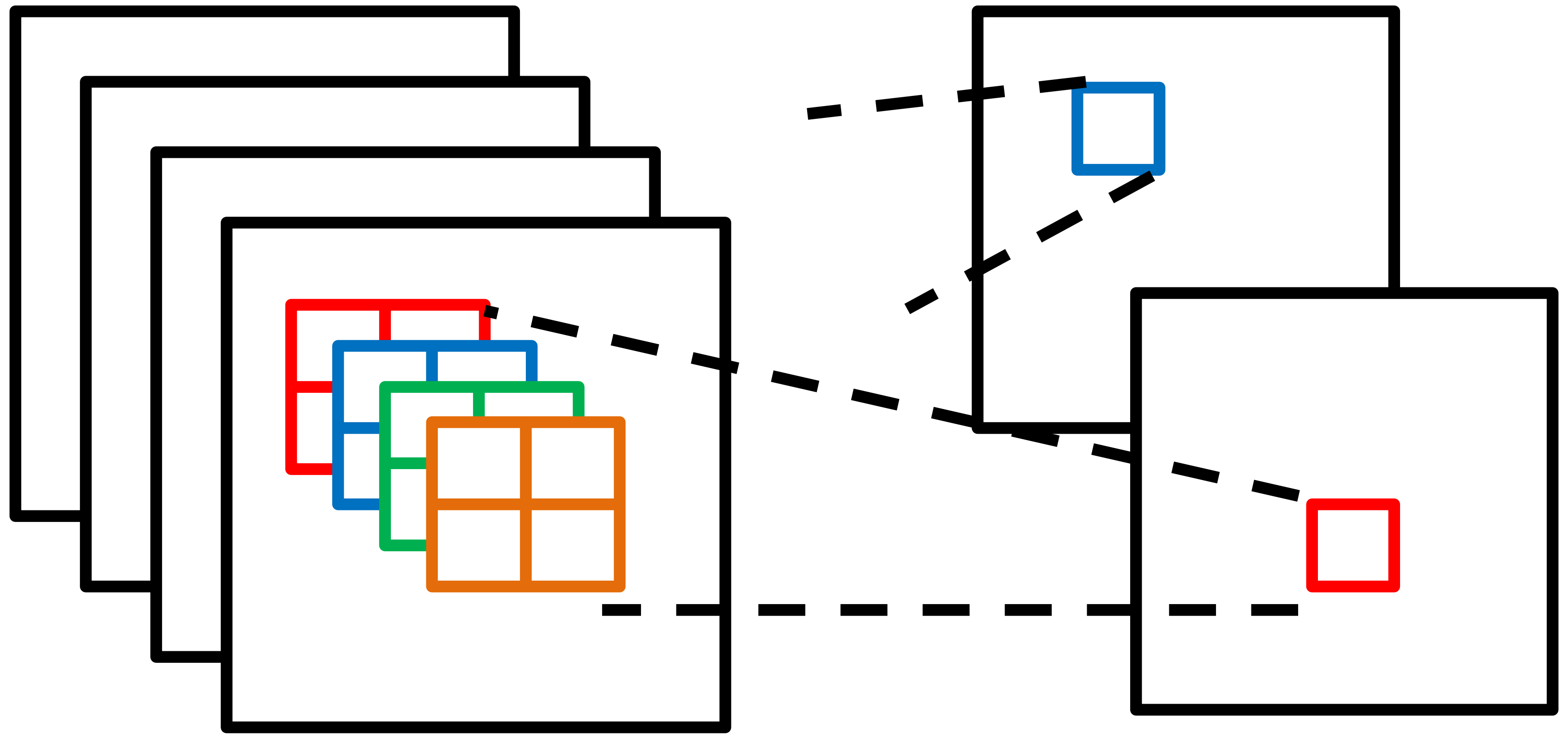
$$y^l_{ij} = \sum_{r=1}^{R} \sum_{c=1}^{C} \phi^l_{rc} x^{l-1}_{i-r,j-c}$$

vertical and horizontal input shifts

- spatial FIR convolutional filtering

Convolutional neural networks

- CNNs increase descriptive power with a parallel filter bank



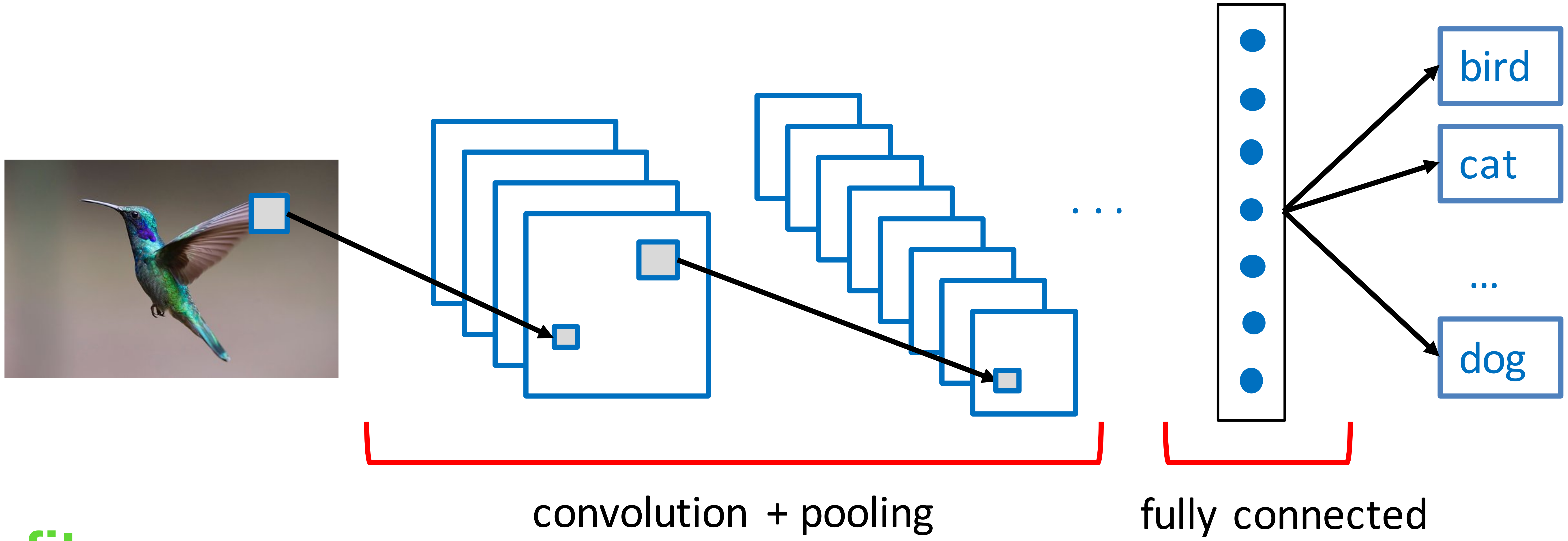
input layer l
has 4 inputs

output layer l
has 2 outputs

- ✦ input F images
- ✦ process each with a parallel bank of filters
- ✦ sum filter outputs to obtain higher-level features
- ✦ parameters are filter coefficients (backprop.)

CNN full stack

- Cascade of spatial filter bank and nonlinearities



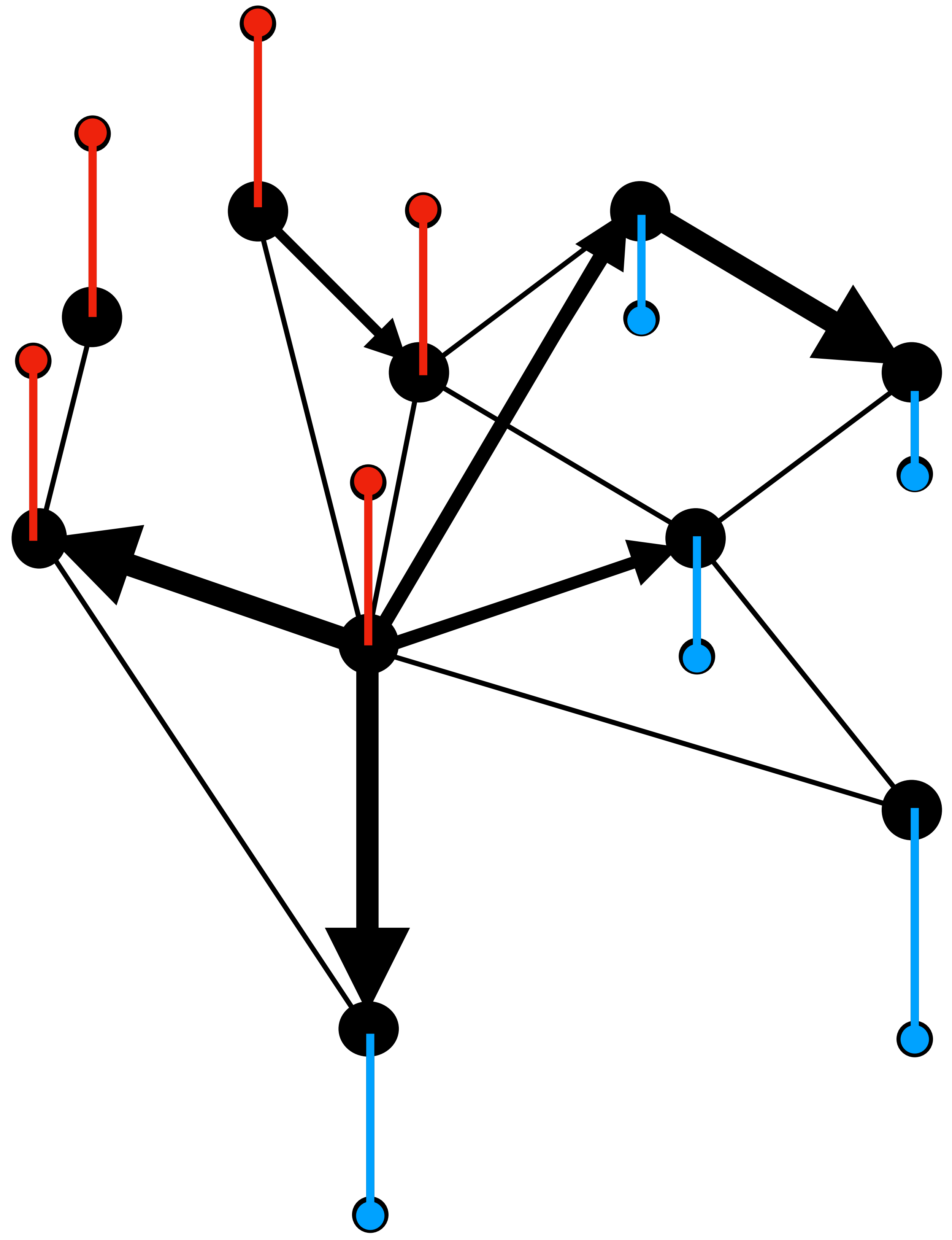
Benefits

- Parameters - independent on the image dimensions
- Complexity - spatial convolution

What about data on graphs?

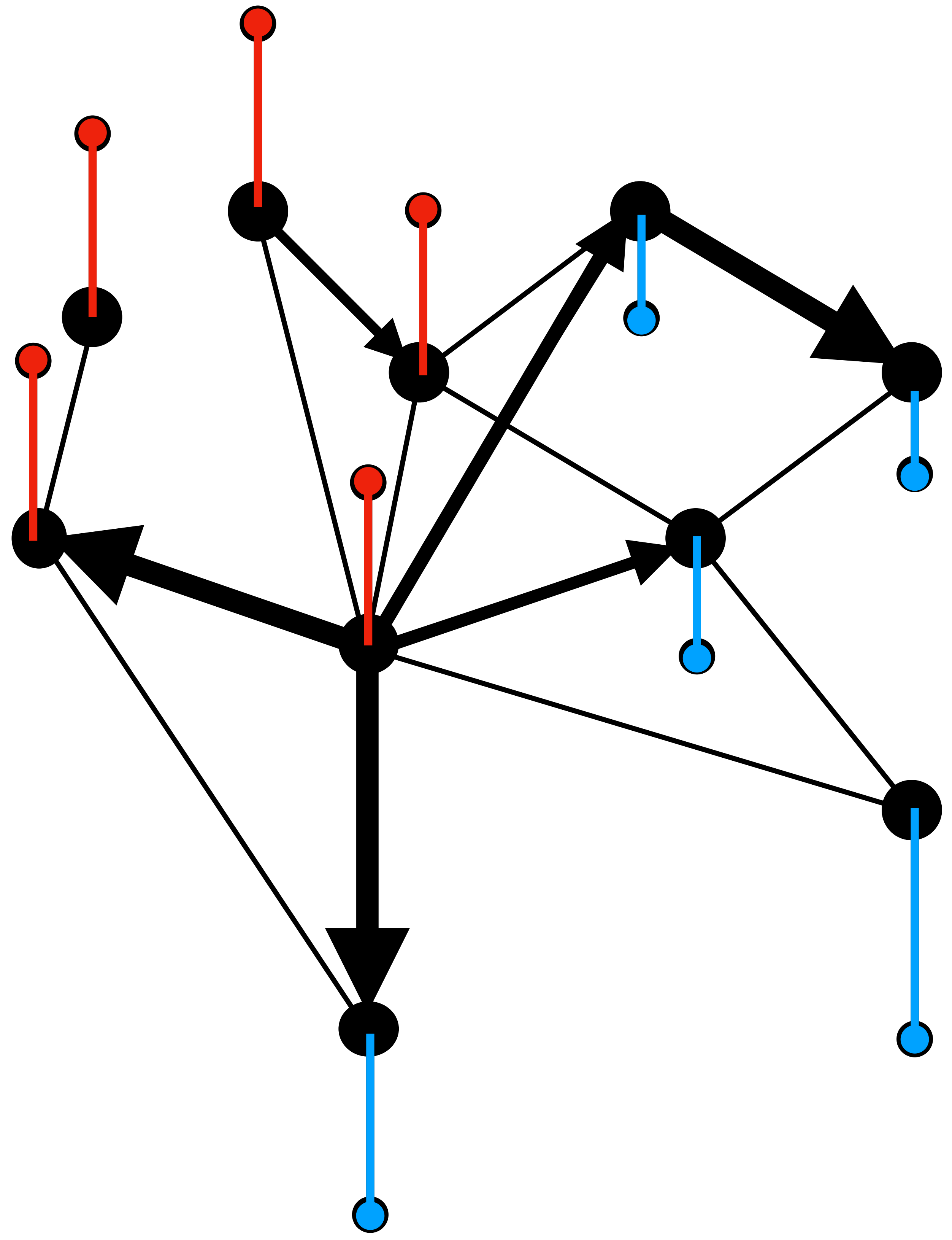
Learning from (ir)regular graph data

- Training samples $\mathbf{x}_r \in \mathbb{R}^N$ are graph signals
- Non-Euclidean structure
  - ✦ conventional CNNs are inapplicable



Learning from (ir)regular graph data

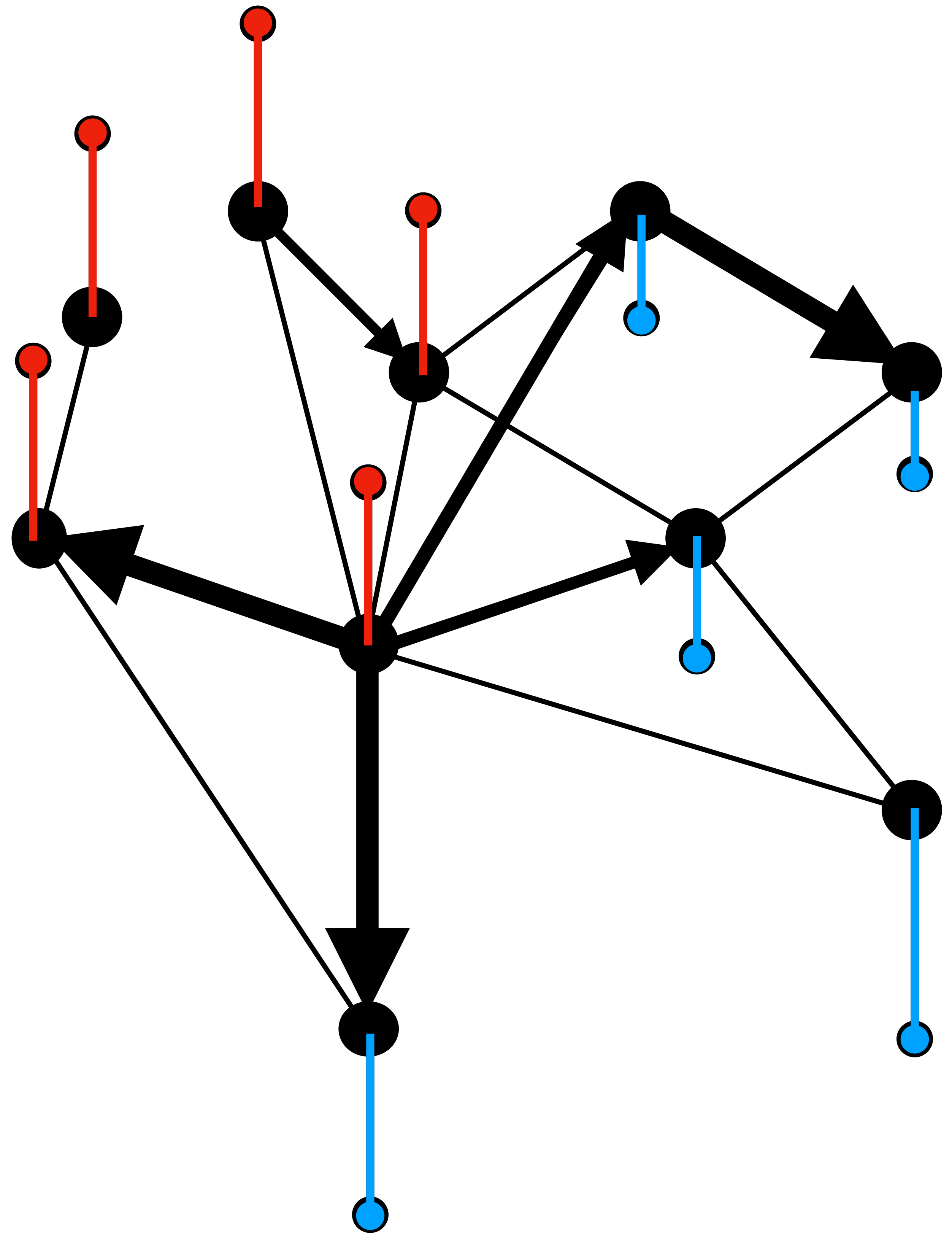
- Training samples $\mathbf{x}_r \in \mathbb{R}^N$ are graph signals
- Non-Euclidean structure
  - conventional CNNs are inapplicable
- MLP can apply $\mathbf{x}^l = \sigma\left(\mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l\right)$
  - ignores the structure
  - data demanding



Learning from (ir)regular graph data

- Need a neural network solution

  to account for coupling signal-topology



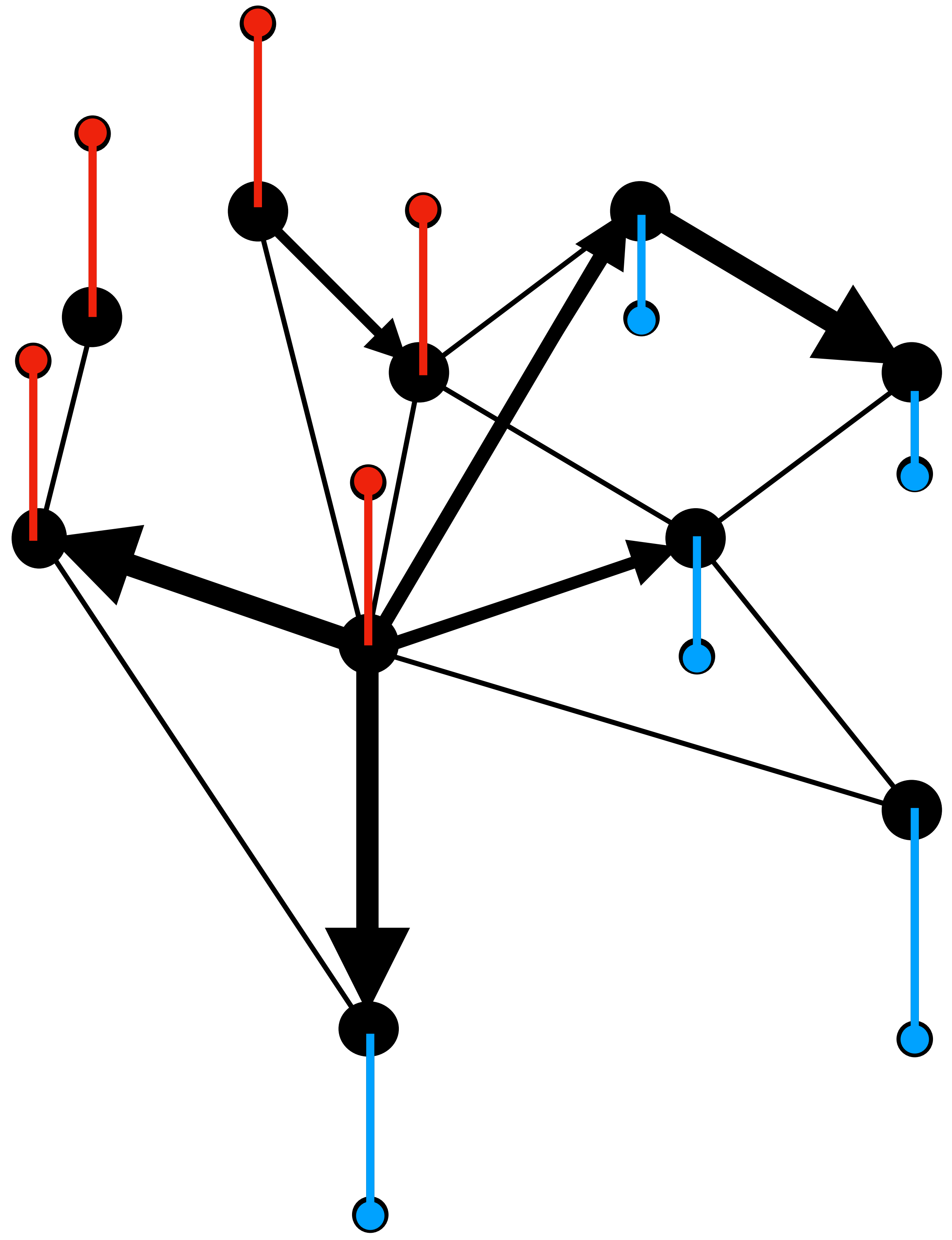
Learning from (ir)regular graph data

- ◉ Need a neural network solution

  to account for coupling signal-topology
- ◉ Graph as prior to estimate a parametric function

$$f(\boldsymbol{\theta}, \mathbf{S}) : \mathscr{X} \rightarrow \mathscr{Y}$$

  - ✦ $\mathbf{S}$ is the graph shift operator
  - ✦ $\boldsymbol{\theta}$ trainable parameters (i.e., filter coefficients)



Gama, Isufi, Leus, Ribeiro, *Graphs, Convolutions, and Neural Networks*, IEEE Signal Processing Magazine, under review, arXiv: 2003.03777

Graph neural networks

- Graph neural networks substitute $\mathbf{W}^l$ with graph filter bank

Graph neural networks

- Graph neural networks substitute $\mathbf{W}^l$ with graph filter bank
- Propagation rule through graph filters

$$\mathbf{x}^l = \sigma\left(\mathbf{H}^l\mathbf{x}^{l-1}\right)$$

- $\mathbf{H}^l$ graph filter at layer l for any shift operator $\mathbf{S}$

Graph neural networks

- Graph neural networks substitute $\mathbf{W}^l$ with graph filter bank
- Propagation rule through graph filters

$$\mathbf{x}^l = \sigma\left(\mathbf{H}^l\mathbf{x}^{l-1}\right)$$

- $\mathbf{H}^l$ graph filter at layer l for any shift operator $\mathbf{S}$
  - Edge-variant filter: EdgeNets
  - Node-variant filter: Node-variant GNNs [Gama'18 - DSW]

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

Graph neural networks

- Graph neural networks substitute $\mathbf{W}^l$ with graph filter bank
- Propagation rule through graph filters

$$\mathbf{x}^l = \sigma\left(\mathbf{H}^l \mathbf{x}^{l-1}\right)$$

- $\mathbf{H}^l$ graph filter at layer l for any shift operator $\mathbf{S}$
  - ✦ Edge-variant filter: EdgeNets
  - ✦ Node-variant filter: Node-variant GNNs [Gama'18 - DSW]
  - ✦ FIR filters: Graph convolutional neural networks[Gama'18 - TSP]
    - Chebyshev form: ChebNets [Defferrard'16 - NeurIPS]

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

Graph neural networks

- Graph neural networks substitute $\mathbf{W}^l$ with graph filter bank
- Propagation rule through graph filters

$$\mathbf{x}^l = \sigma\big(\mathbf{H}^l\mathbf{x}^{l-1}\big)$$

- $\mathbf{H}^l$ graph filter at layer l for any shift operator $\mathbf{S}$
  - Edge-variant filter: EdgeNets
  - Node-variant filter: Node-variant GNNs [Gama'18 - DSW]
  - FIR filters: Graph convolutional neural networks [Gama'18 - TSP]
    - Chebyshev form: ChebNets [Defferrard'16 - NeurIPS]
  - ARMA filters: ARMANets
    - Direct, parallel, cascade [Wijesinghe'19 - NeurIPS] [Bianchi'19-arXiv]
    - Cayley form: CayleyNets [Levie'18 - TSP]

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

Graph convolutional neural networks

- Graph convolutional neural networks use a graph convolutional filter (FIR - ARMA)

Example: FIR

$$\mathbf{x}^{l} = \sigma\Big(\sum_{k=0}^{K} \phi_k^l \mathbf{S}^k \mathbf{x}^{l-1}\Big)$$

Gama, Marques, Leus, Ribeiro, *Convolutional Neural Networks Architectures for Signals Supported on Graphs*, IEEE Transactions on Signal Processing, 2018

Graph convolutional neural networks

- Graph convolutional neural networks use a graph convolutional filter (FIR - ARMA)

Example: FIR

$$\mathbf{x}^l = \sigma\Big(\sum_{k=0}^{K} \phi_k^l \mathbf{S}^k \mathbf{x}^{l-1}\Big)$$

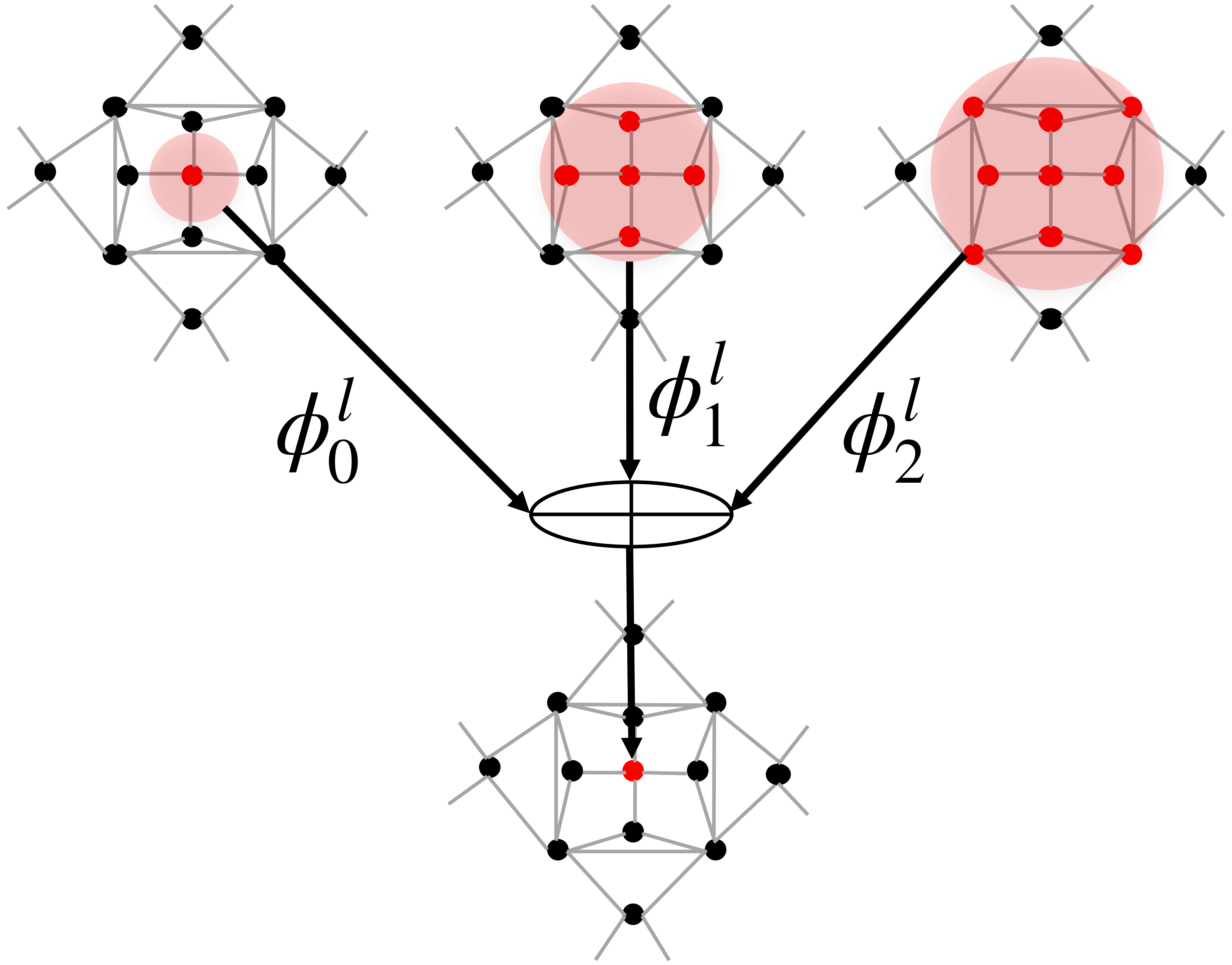
- parameters shared among all nodes and edges
- shift-and-sum convolves filter with graph signal

Gama, Marques, Leus, Ribeiro, *Convolutional Neural Networks Architectures for Signals Supported on Graphs*, IEEE Transactions on Signal Processing, 2018

Graph convolutional neural networks

- GCNN: shift-and-sum & shared parameters

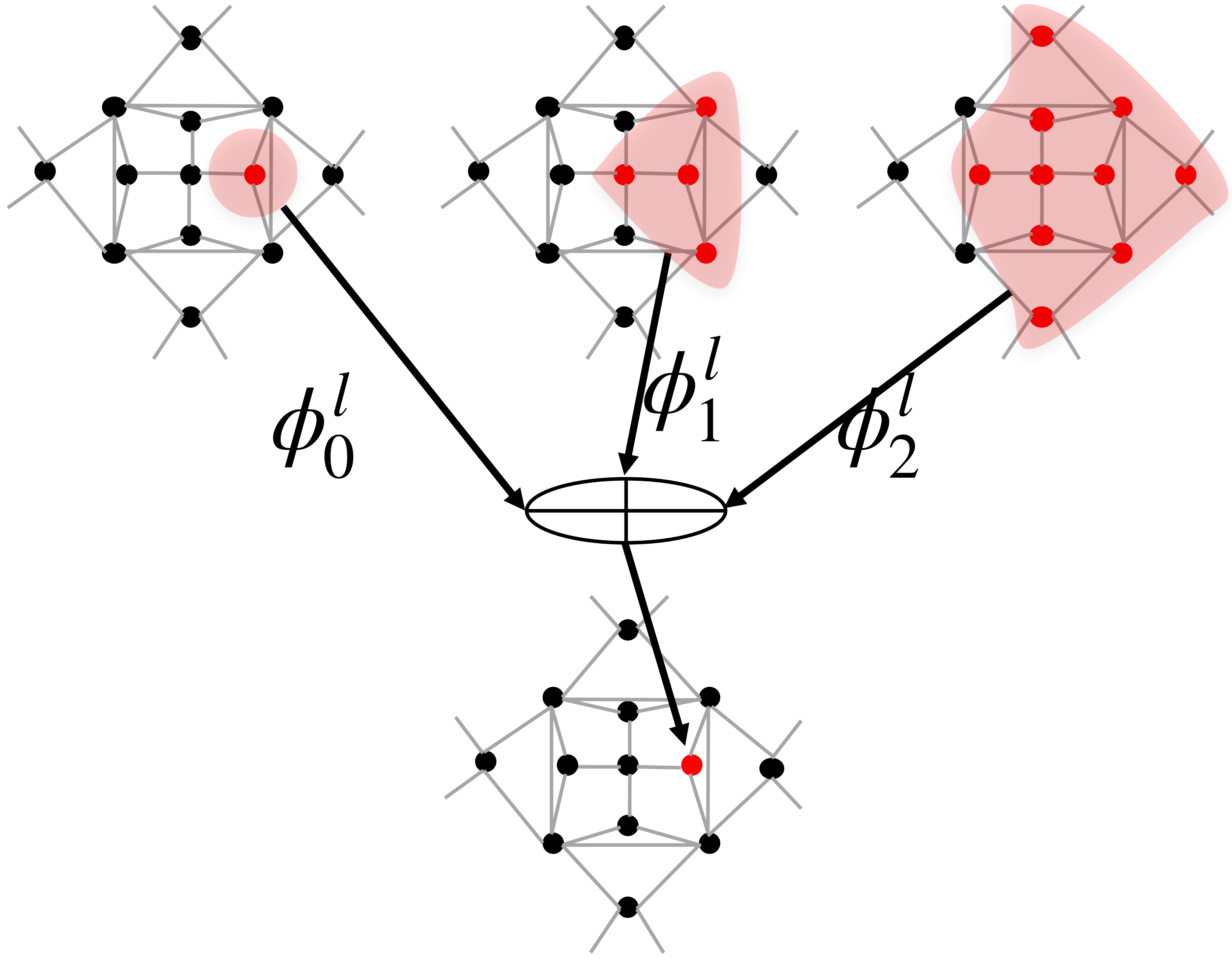
$$\mathbf{x}^l = \sigma\left(\phi_0^l \mathbf{x}^{l-1} + \phi_1^l \mathbf{S}\mathbf{x}^{l-1} + \phi_2^l \mathbf{S}^2 \mathbf{x}^{l-1}\right)$$



Graph convolutional neural networks

- GCNN: shift-and-sum & shared parameters

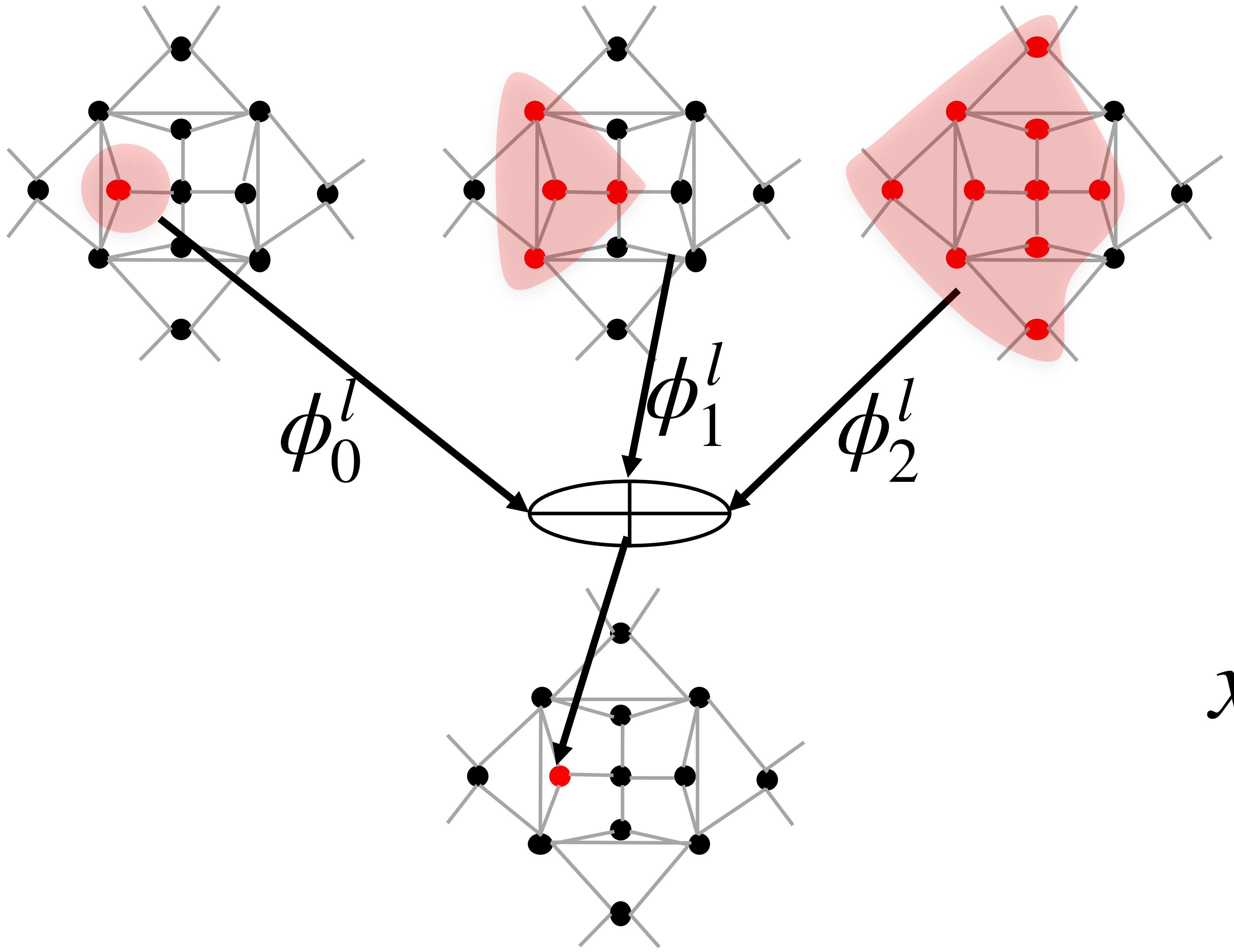
$$\mathbf{x}^l = \sigma\big(\phi_0^l \mathbf{x}^{l-1} + \phi_1^l \mathbf{S}\mathbf{x}^{l-1} + \phi_2^l \mathbf{S}^2\mathbf{x}^{l-1}\big)$$



Graph convolutional neural networks

- GCNN: shift-and-sum & shared parameters

$$\mathbf{x}^{l} = \sigma\left(\phi_0^l \mathbf{x}^{l-1} + \phi_1^l \mathbf{S}\mathbf{x}^{l-1} + \phi_2^l \mathbf{S}^2\mathbf{x}^{l-1}\right)$$

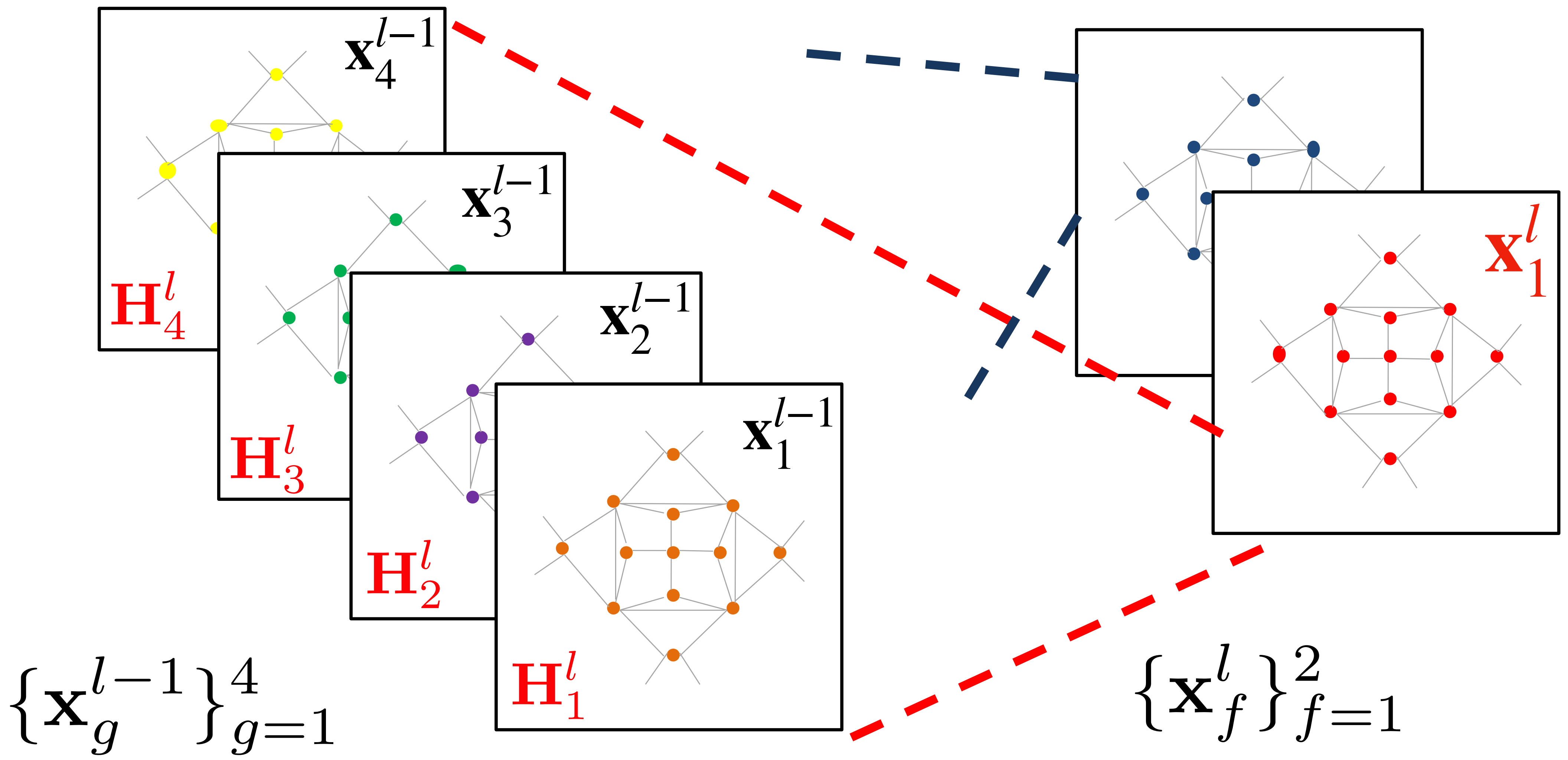


shift over the nodes

$$x_i^{l} = \sigma\left(\phi_0^l x_i^{l-1} + \phi_1^l [\mathbf{S}\mathbf{x}^{l-1}]_i + \phi_2^l [\mathbf{S}^2\mathbf{x}^{l-1}]_i\right)$$

Graph convolutional neural networks

- GCNNs increase descriptive power with a parallel filter bank



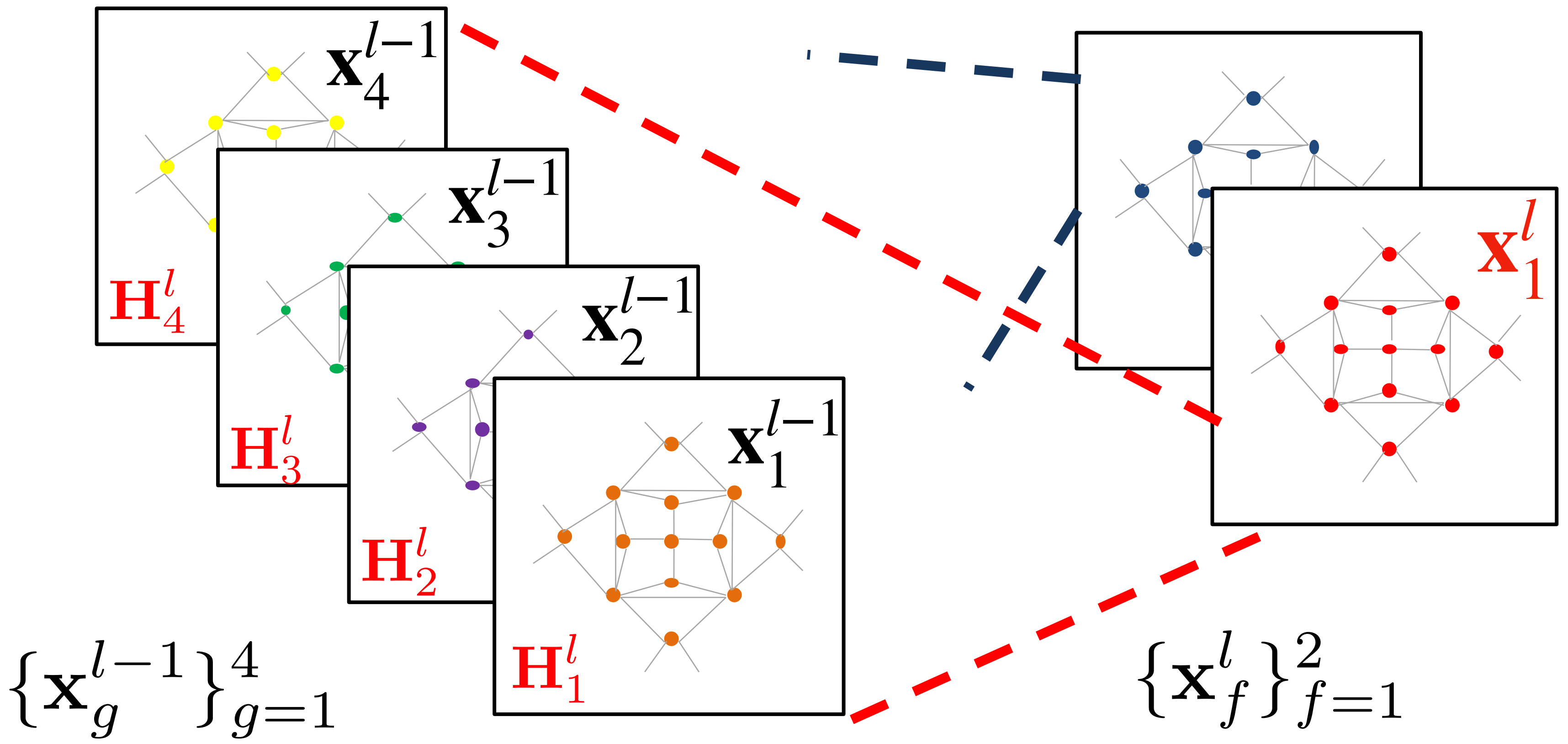
input layer l
has 4 inputs

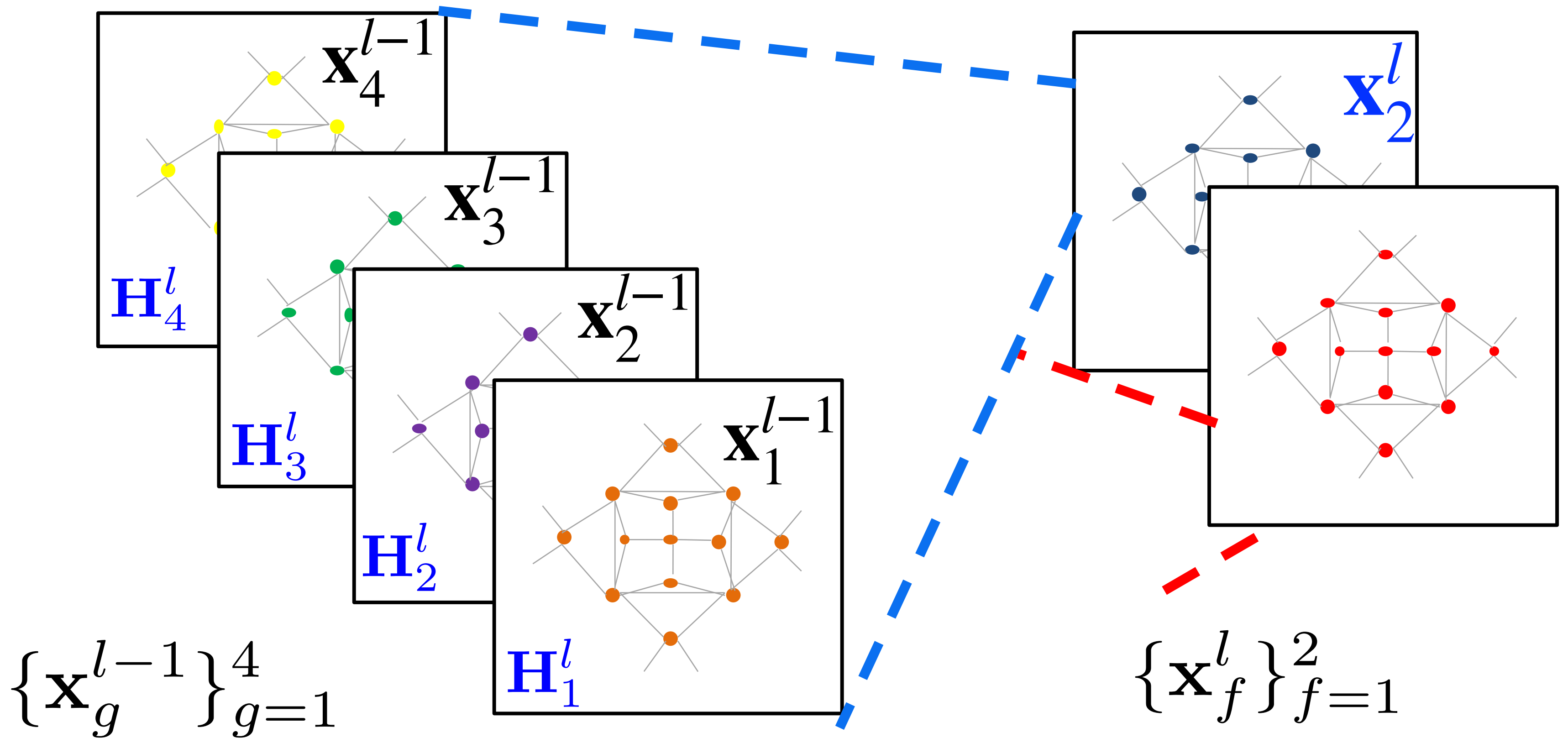
output layer l
has 2 outputs

- F input graph signals $\{\mathbf{x}_g^{l-1}\}_{g=1}^F$
- process each signal with a graph filter
- sum filter outputs
- parameter are filter coefficients (backprop.)

Graph convolutional neural networks

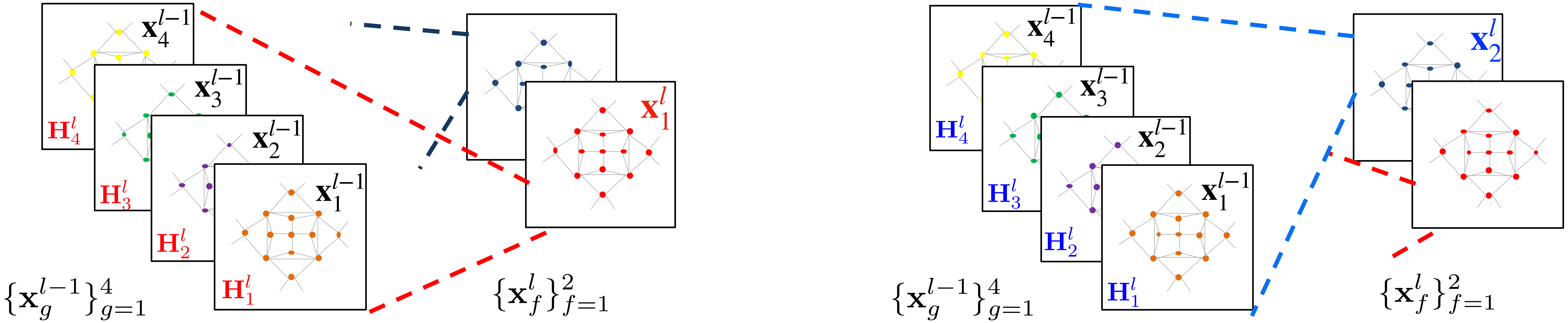
- GCNNs increase descriptive power with a parallel filter bank





Graph convolutional neural networks

- GCNNs increase descriptive power with a parallel filter bank

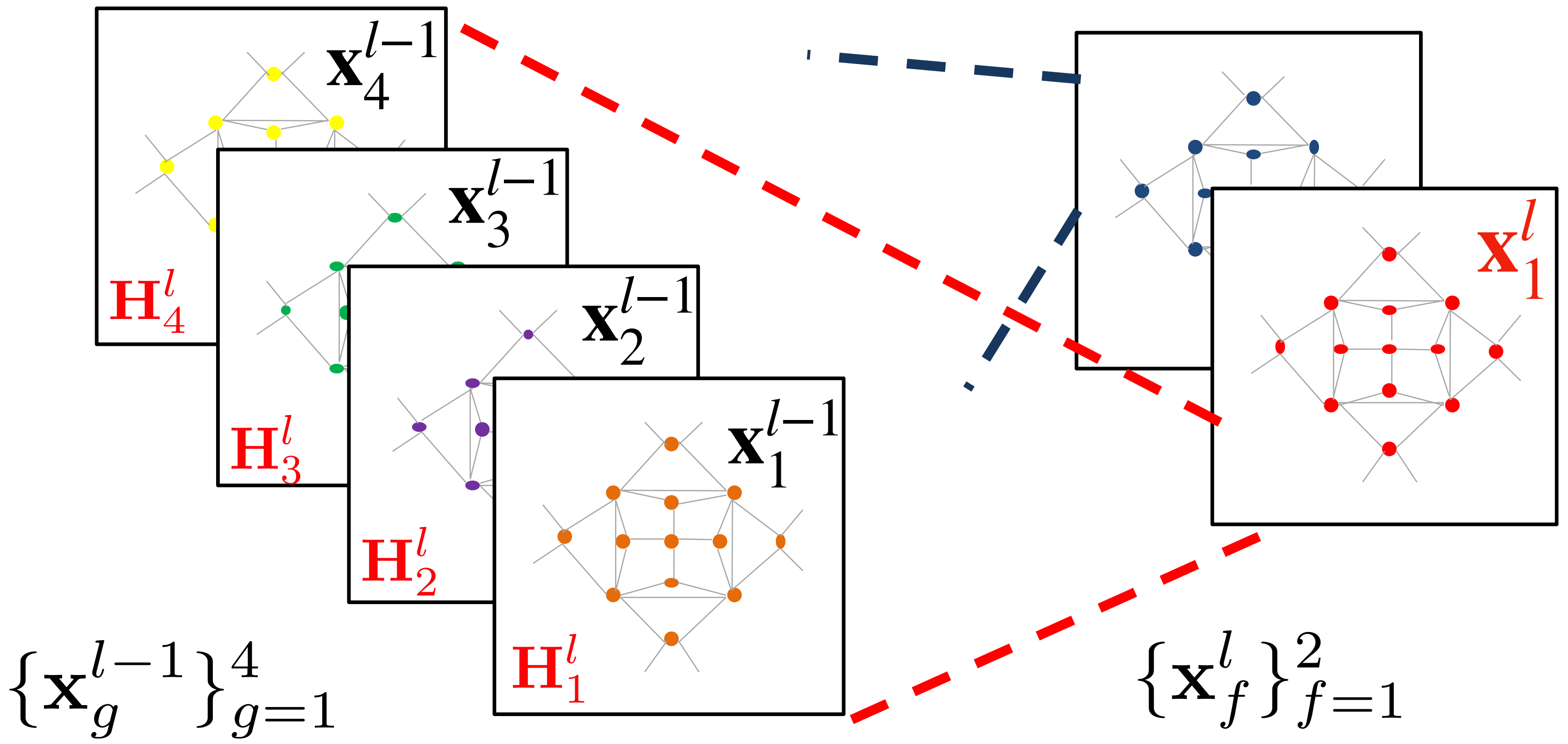


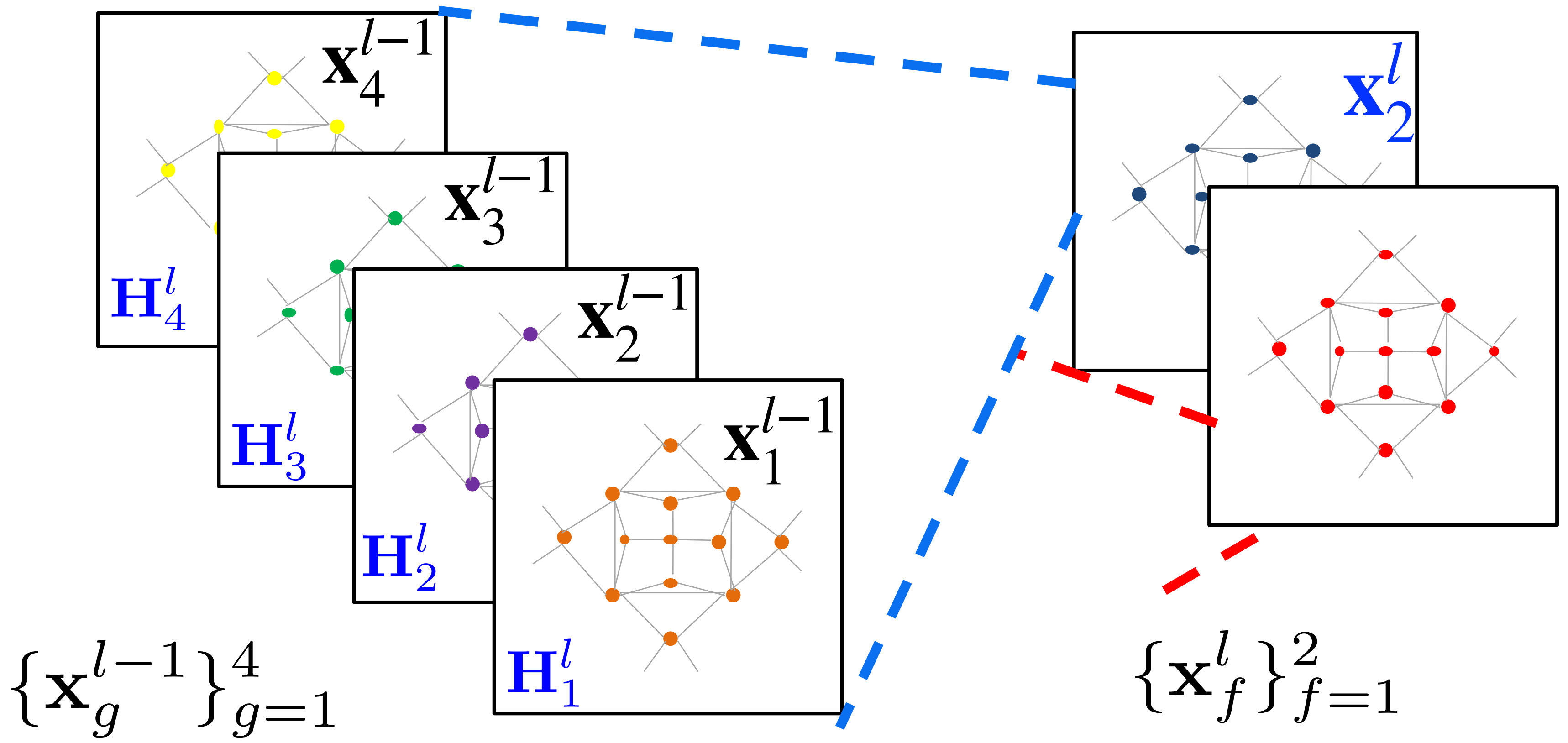
input feature

$$\mathbf{x}_f^l = \sigma\left(\sum_{g=1}^{F} \mathbf{H}_{fg}^l \mathbf{x}_g^{l-1}\right)$$

Graph convolutional neural networks

- GCNNs increase descriptive power with a parallel filter bank





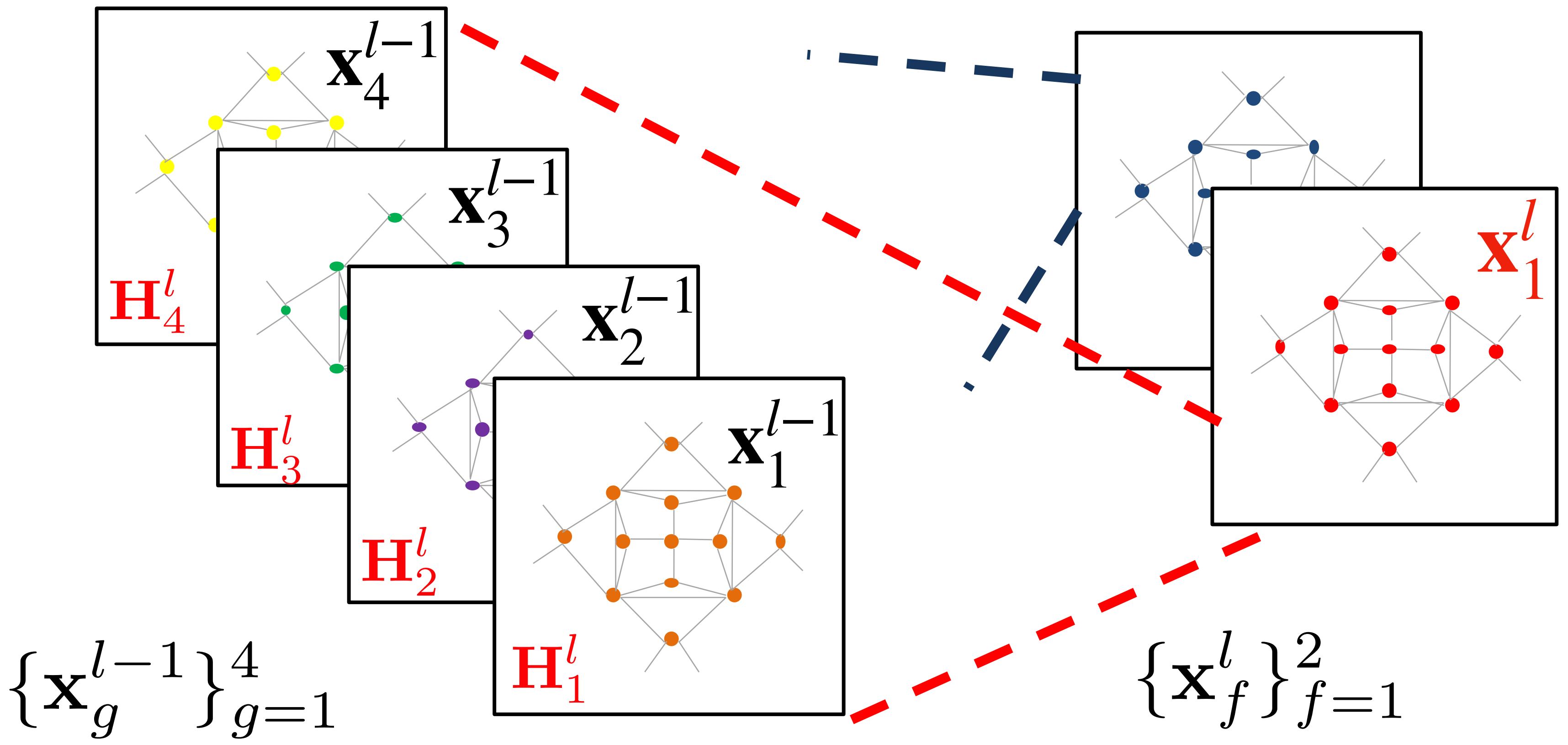
input feature

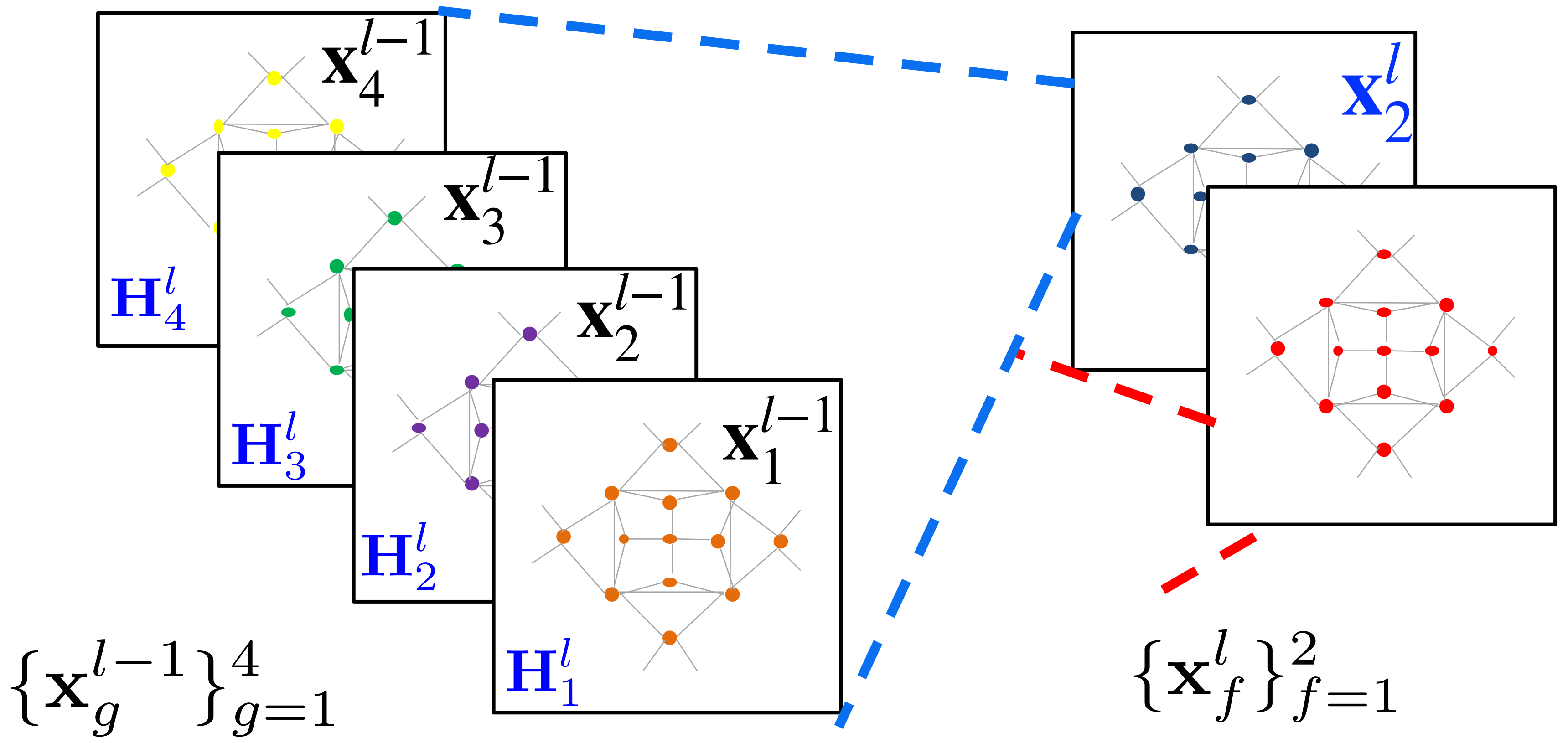
$$\mathbf{x}_f^l = \sigma\left(\sum_{g=1}^{F} \mathbf{H}_{fg}^l \mathbf{x}_g^{l-1}\right)$$

FIR filter

Graph convolutional neural networks

- GCNNs increase descriptive power with a parallel filter bank





sum over inputs

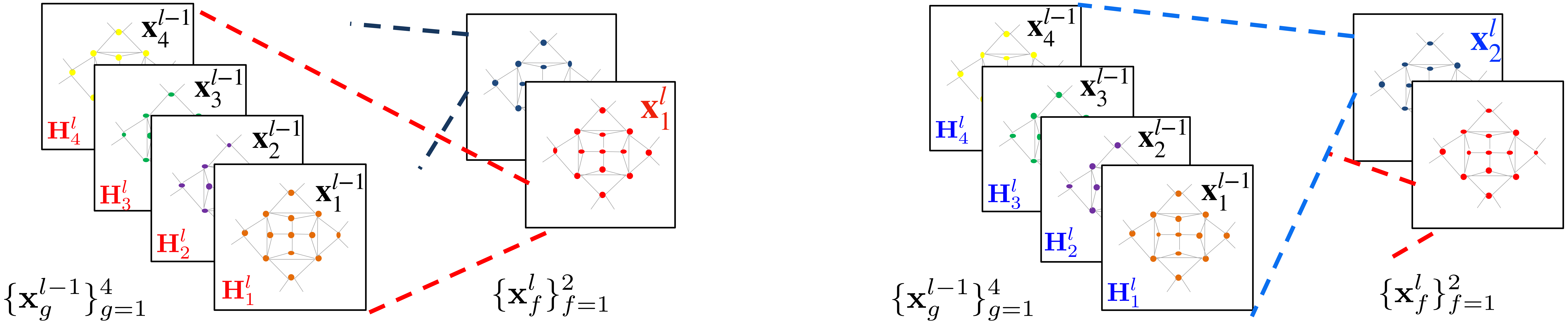
input feature

$$\mathbf{x}_f^l = \sigma\left(\sum_{g=1}^{F} \mathbf{H}_{fg}^l \mathbf{x}_g^{l-1}\right)$$

FIR filter

Graph convolutional neural networks

- GCNNs increase descriptive power with a parallel filter bank



sum over inputs

input feature

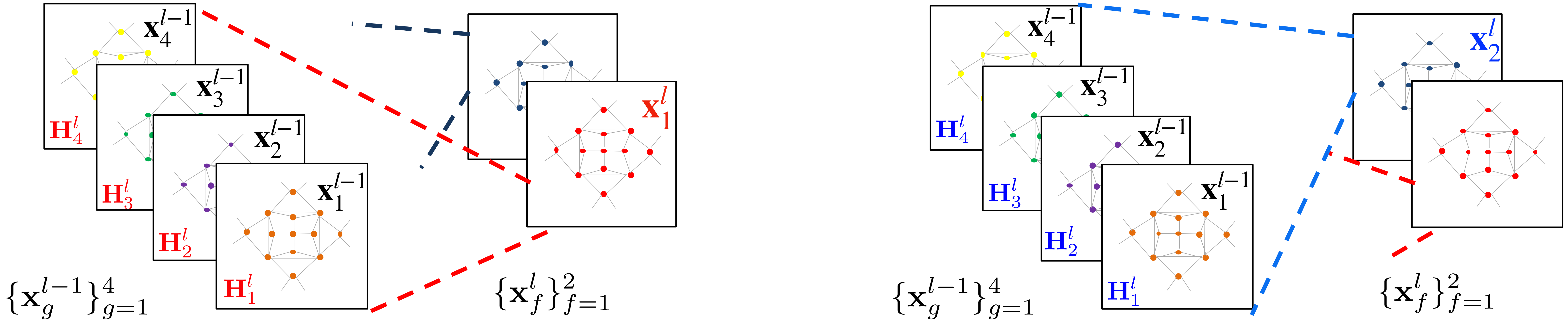
output feature

$$\mathbf{x}_f^l = \sigma\left(\sum_{g=1}^{F} \mathbf{H}_{fg}^l \mathbf{x}_g^{l-1}\right)$$

FIR filter

Graph convolutional neural networks

- GCNNs increase descriptive power with a parallel filter bank



sum over inputs

input feature

output feature

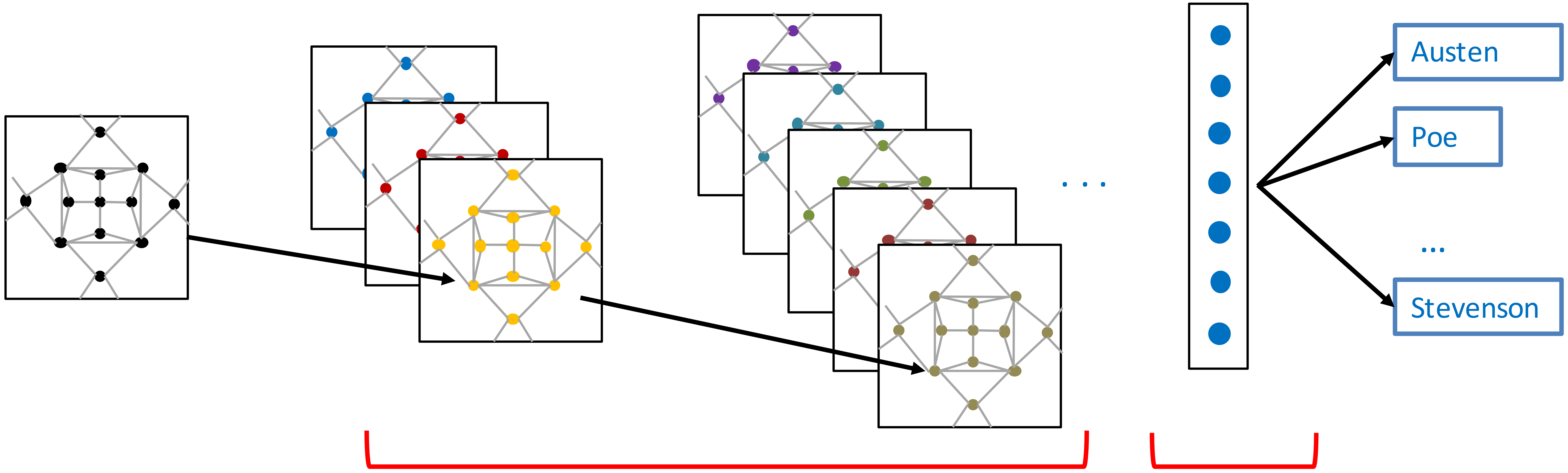
for all output features

$$\mathbf{x}_f^l = \sigma\left(\sum_{g=1}^{F} \mathbf{H}_{fg}^l \mathbf{x}_g^{l-1}\right) \qquad \forall f \in \{1,\ldots,F\}$$

FIR filter

GCNN full stack

- Cascade graph filters and nonlinearities



graph convolution **fully connected**

Benefits

- Parameters $\mathcal{O}(KF^2L)$ - independent on the graph dimensions
- Complexity $\mathcal{O}(KMF^2L)$ - linear in number of edges (~nodes)

EdgeNet

- Substitutes FIR filters with edge-variant graph filter
- Propagation rule

$$\mathbf{x}_f^l = \sigma\left(\sum_{g=1}^{F} \boxed{\mathbf{H}_{\mathrm{EV}fg}^l}\, \mathbf{x}_g^{l-1}\right) \forall f \in \{1,\ldots,F\}$$

Edge-variant filter

Isufi, Gama, Ribeiro, *Generalizing Graph Convolutional Neural Networks with Edge-Variant Recursions on Graphs*, EUSIPCO, 2019.

EdgeNet

- Substitutes FIR filters with edge-variant graph filter
- Propagation rule

$$\mathbf{x}_f^l = \sigma\left(\sum_{g=1}^{F} \mathbf{H}_{\mathrm{EV}fg}^l \mathbf{x}_g^{l-1}\right) \forall f \in \{1,\ldots,F\}$$

Edge-variant filter

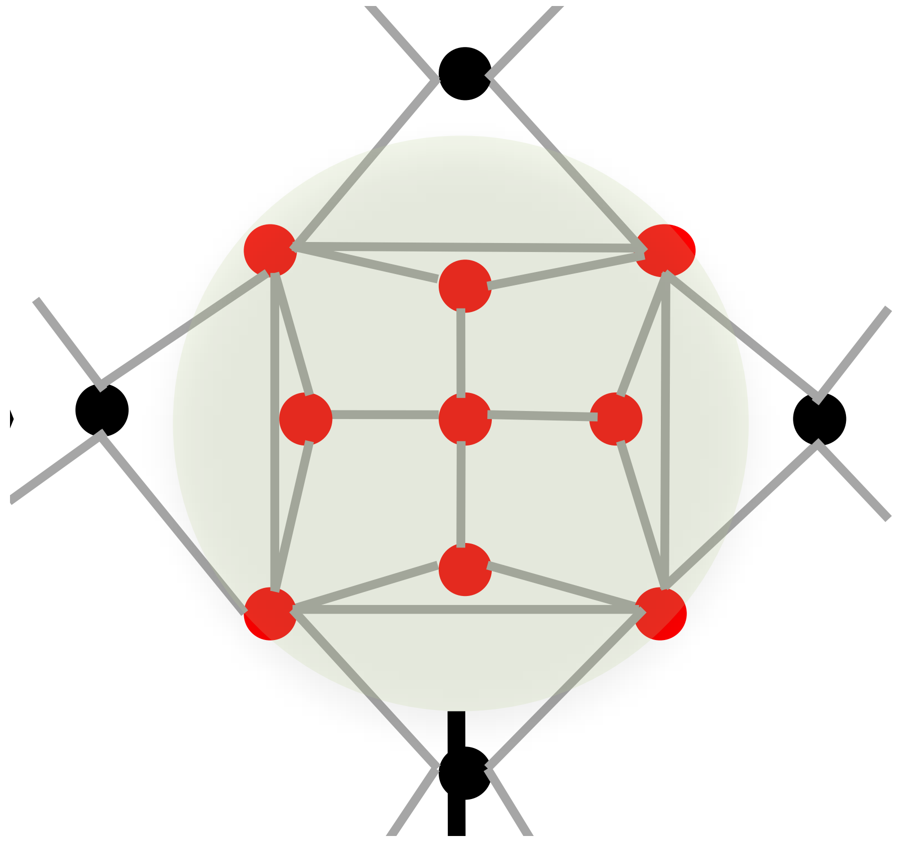
- The most general GNN
  - ✦ Includes all GCNN, all ARMANet, GIN, GAT

Isufi, Gama, Ribeiro, *Generalizing Graph Convolutional Neural Networks with Edge-Variant Recursions on Graphs*, EUSIPCO, 2019.

EdgeNet properties

- Different parameters per edge and node

$\Phi_k \mathbf{x}^{(k-1)}$



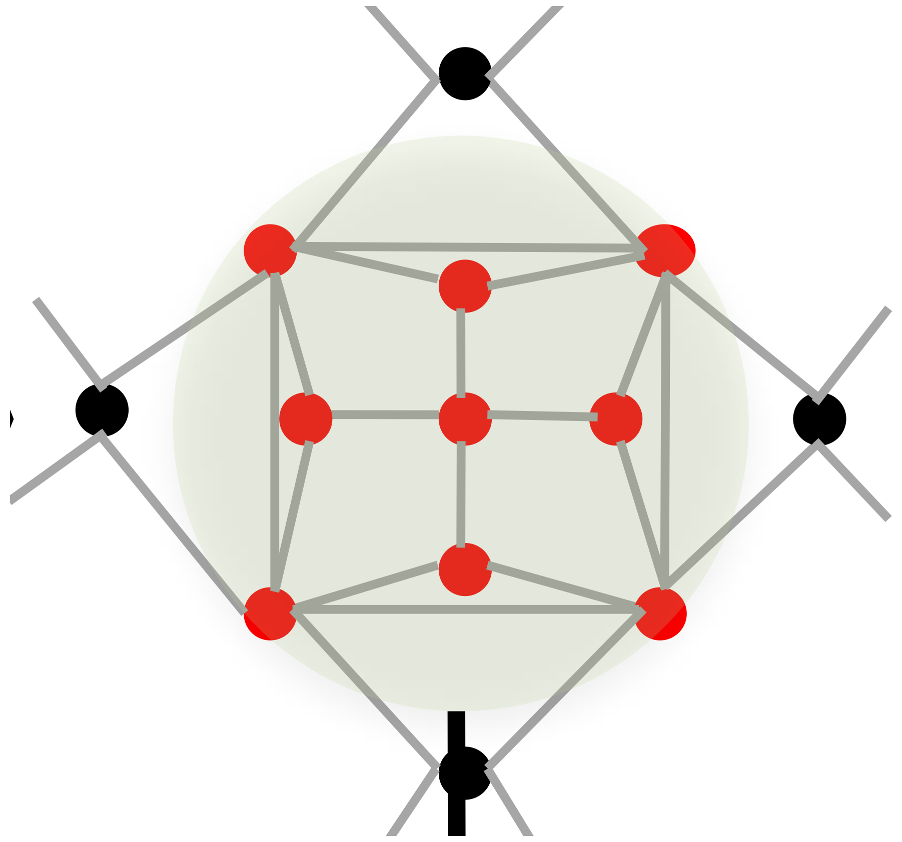
for kth iteration

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

EdgeNet properties

- Different parameters per edge and node
  - Order $\mathcal{O}(MKF^2L)$
  - More flexibility

$\mathbf{\Phi}_k \mathbf{x}^{(k-1)}$



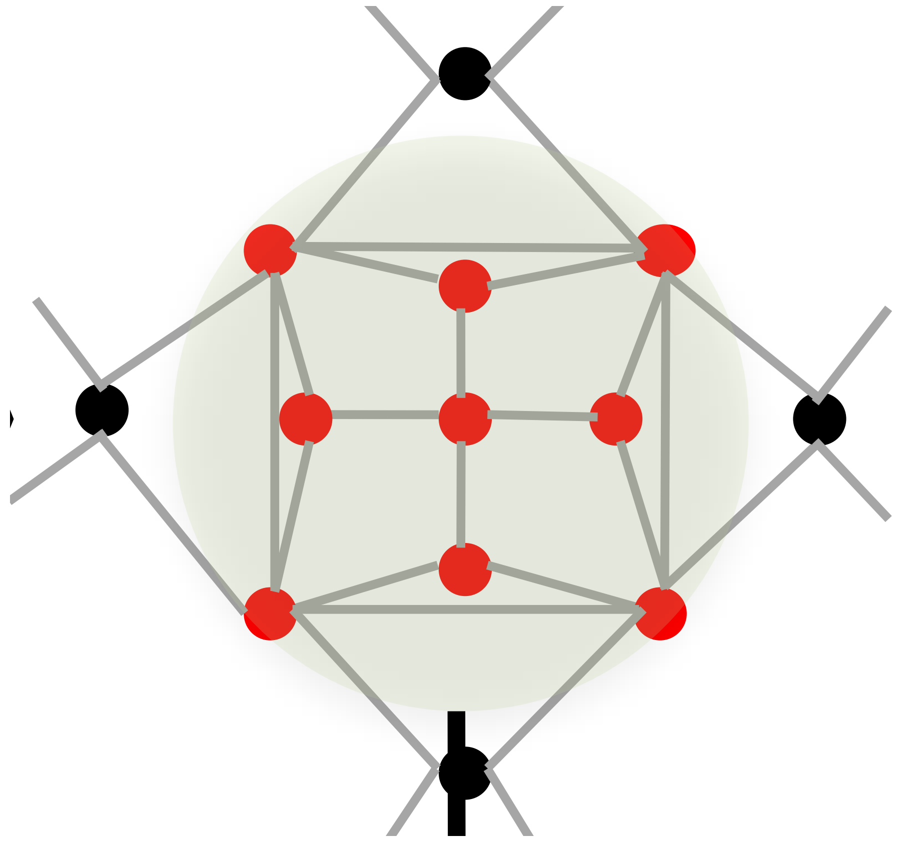
for kth iteration

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

EdgeNet properties

- Different parameters per edge and node
  - ✦ Order $\mathcal{O}(MKF^2L)$
  - ✦ More flexibility
  - ✦ Requires only the support of $\mathbf{S}$
    - Adapts the edge weights to the task
    - Robust to uncertainties in edge weights

$\mathbf{\Phi}_k\mathbf{x}^{(k-1)}$



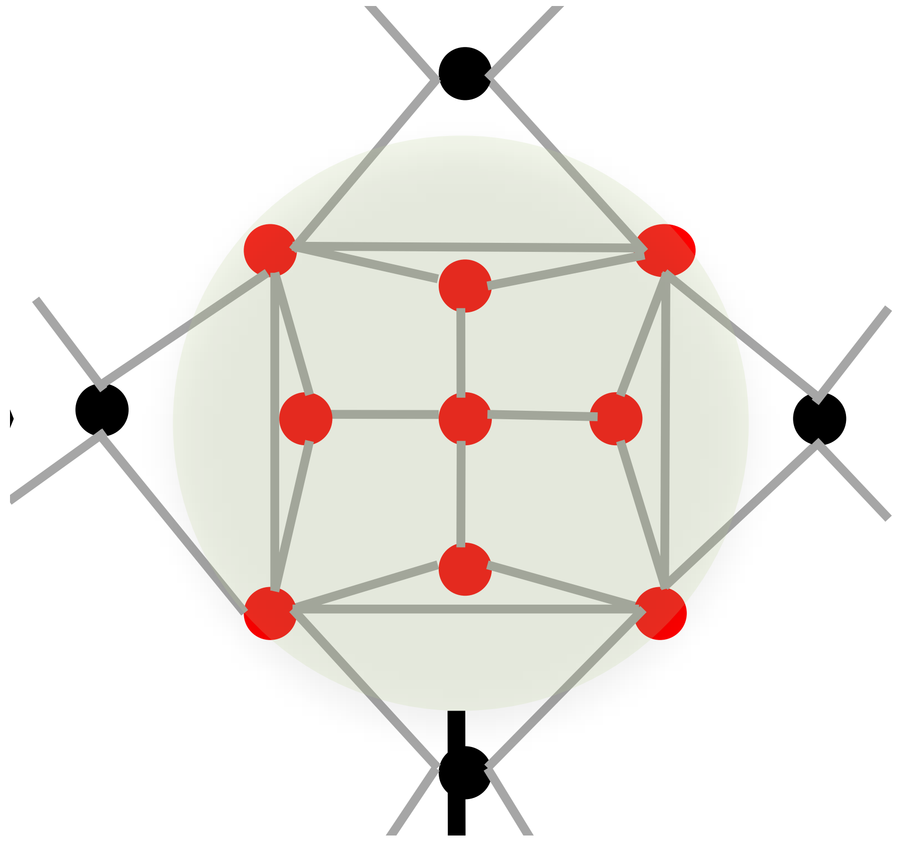
for kth iteration

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

EdgeNet properties

- Different parameters per edge and node
  - ✦ Order $\mathcal{O}(MKF^2L)$
  - ✦ More flexibility
  - ✦ Requires only the support of $\mathbf{S}$
    - Adapts the edge weights to the task
    - Robust to uncertainties in edge weights
  - ✦ Requires fewer parallel filters and shallower networks
  - ✦ Can overfit and require more data than GCNN (FIR-filters)

$\mathbf{\Phi}_k\mathbf{x}^{(k-1)}$



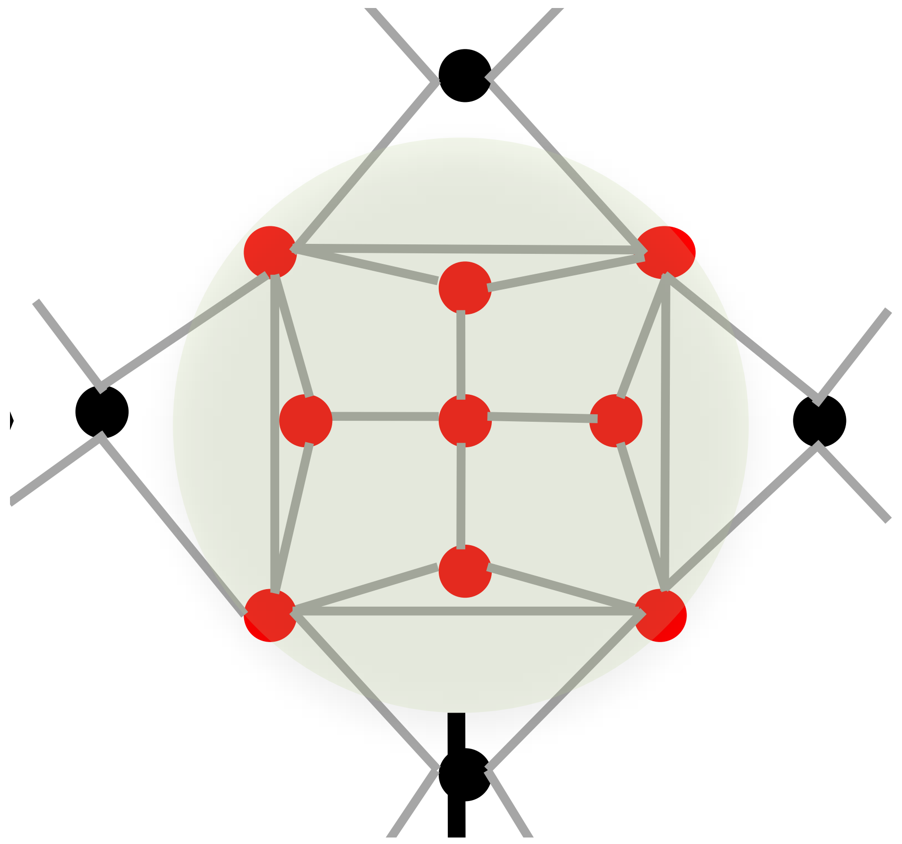
for kth iteration

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

EdgeNet properties

- Different parameters per edge and node
  - Order $\mathcal{O}(MKF^2L)$
  - More flexibility
  - Requires only the support of $\mathbf{S}$
    - Adapts the edge weights to the task
    - Robust to uncertainties in edge weights
  - Requires fewer parallel filters and shallower networks
  - Can overfit and require more data than GCNN (FIR-filters)
- Complexity $\mathcal{O}(MKF^2L)$ - depends on edges (like GCNN)

$\mathbf{\Phi}_k\mathbf{x}^{(k-1)}$



for kth iteration

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

How to use EdgeNets?

- The full form may sometimes overfit
    - Penalize coefficients to sparse (i.e., $||\mathbf{\Phi}||_1$)

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

How to use EdgeNets?

- The full form may sometimes overfit
  - Penalize coefficients to sparse (i.e., $||\mathbf{\Phi}||_1$)
  - Impose parameter sharing
    - FIR : all nodes all edges same parameter

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

How to use EdgeNets?

- The full form may sometimes overfit
  - ✦ Penalize coefficients to sparse (i.e., $||\mathbf{\Phi}||_1$)
  - ✦ Impose parameter sharing
    - FIR : all nodes all edges same parameter
    - Node-variant : all edges same parameter for a node

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

How to use EdgeNets?

- The full form may sometimes overfit
  - Penalize coefficients to sparse (i.e., $||\mathbf{\Phi}||_1$)
  - Impose parameter sharing
    - FIR : all nodes all edges same parameter
    - Node-variant : all edges same parameter for a node
    - Attention mechanism [Velickovic'18 - ICLR]

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

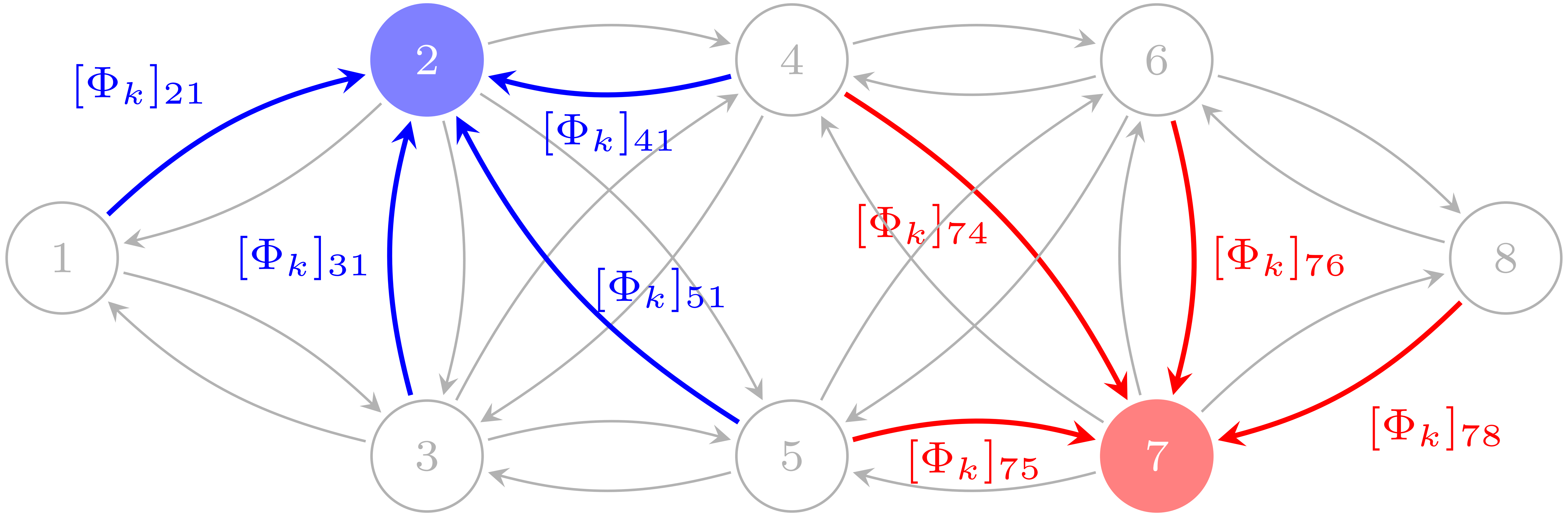
How to use EdgeNets?

- The full form may sometimes overfit
  - Penalize coefficients to sparse (i.e., $||\mathbf{\Phi}||_1$)
  - Impose parameter sharing
    - FIR : all nodes all edges same parameter
    - Node-variant : all edges same parameter for a node
    - Attention mechanism [Velickovic'18 - ICLR]
    - Hybrid : FIR + EV to particular nodes

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

How to use EdgeNets?

Example: Hybrid (FIR + EV)

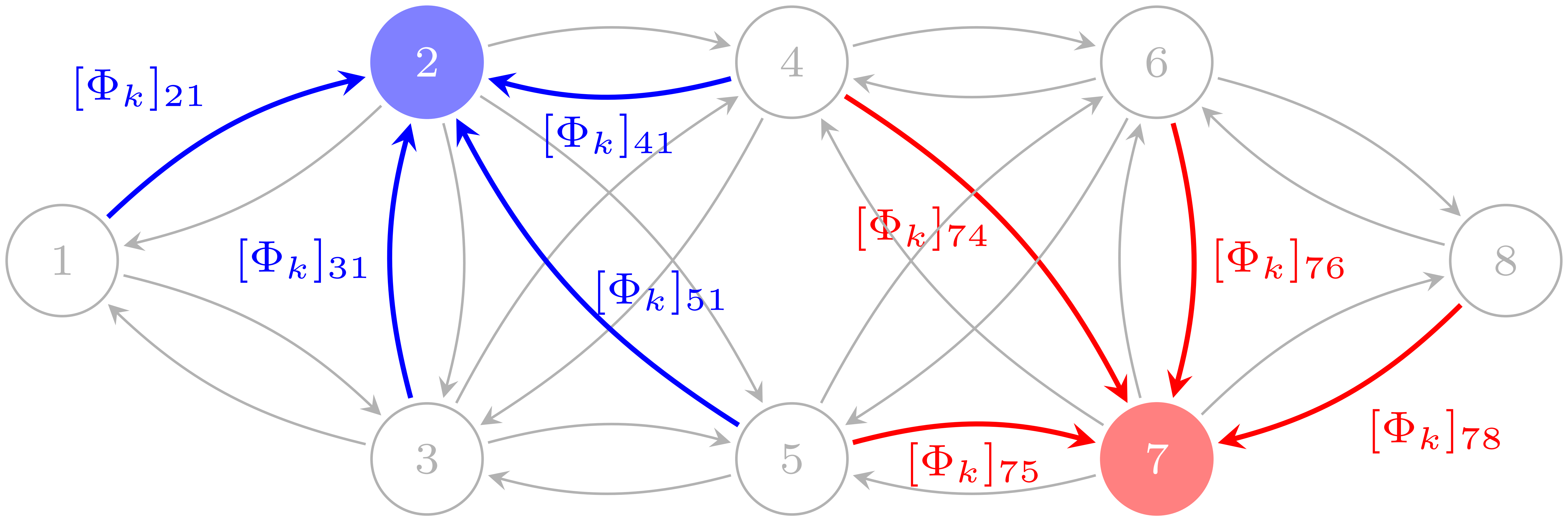


- Nodes 2 and 7 use EV filter
- All other nodes use FIR filter

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

How to use EdgeNets?

Example: Hybrid (FIR + EV)



- Nodes 2 and 7 use EV filter
- All other nodes use FIR filter
- More flexibility than GCNN
- Parameters independent on the graph dimensions

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

How to apply GNNs?

Applications

- Distributed finite-time consensus
- Distributed regression
- Authorship attribution
- Recommender systems

Applications

- Distributed finite-time consensus
- Distributed regression
- Authorship attribution
- Recommender systems

- For control, resource allocation and other SP applications [T-9]
- For semi-supervised learning, graph classification [Wu'20 -TNNLS]

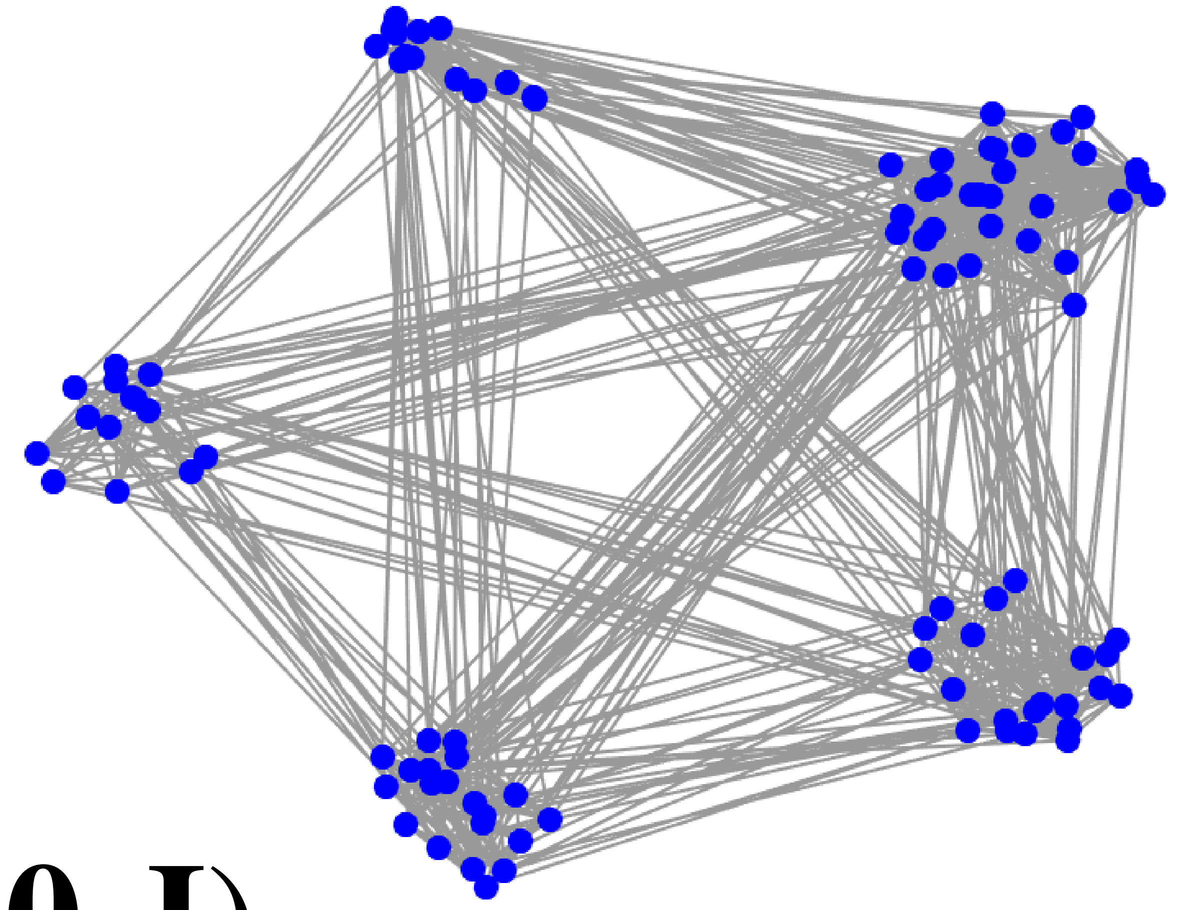
Learning finite-time consensus

- Learn the consensus function for a specific graph
  - EV can do the job but all nodes need to know all graph
    - Feasible only in small setups

Iancu, Isufi, *Towards Finite-Time Consensus with Graph Convolutional Neural Networks*, EUSIPCO 2020 (submitted)

Learning finite-time consensus

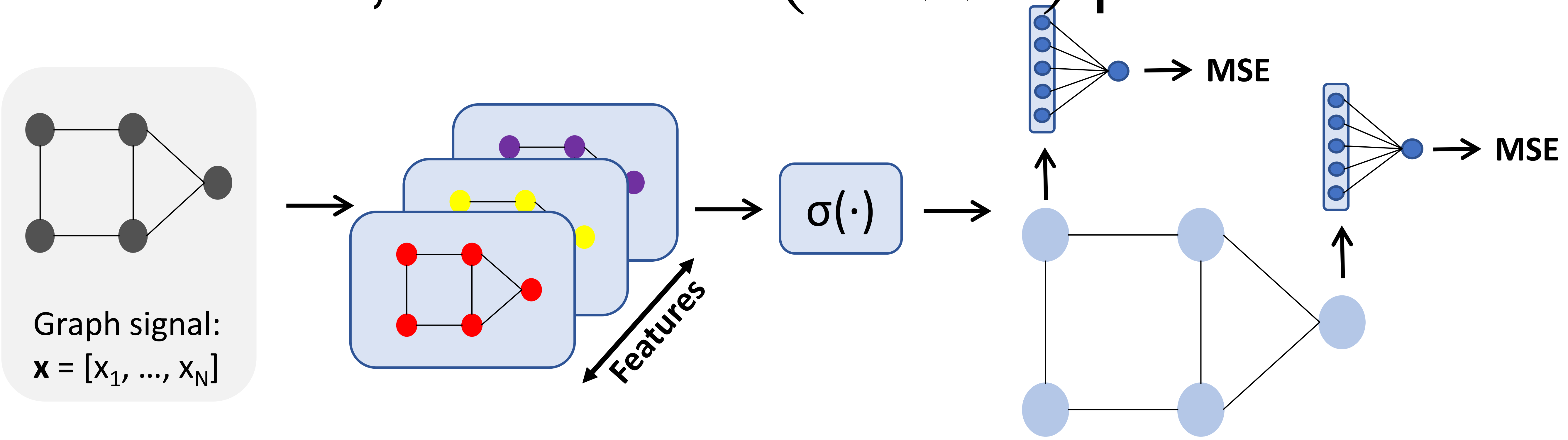
- Learn the consensus function for a specific graph
  - EV can do the job but all nodes need to know all graph
    - Feasible only in small setups



Stochastic block model

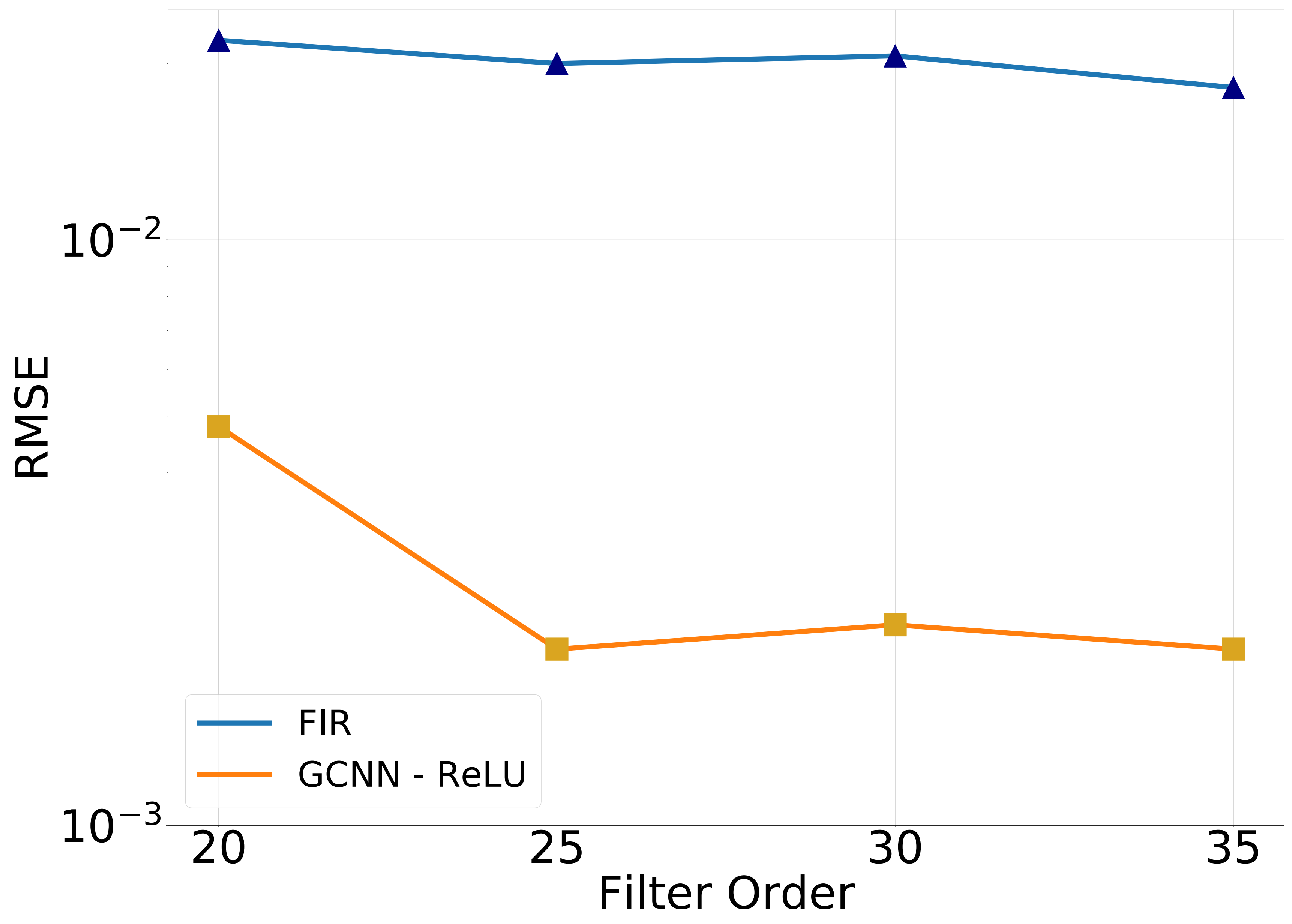
$N = 100$ and $C = 5$ communities; graph signals $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

1 Layer, $F = 32$ features, shared FC (32×1) per node



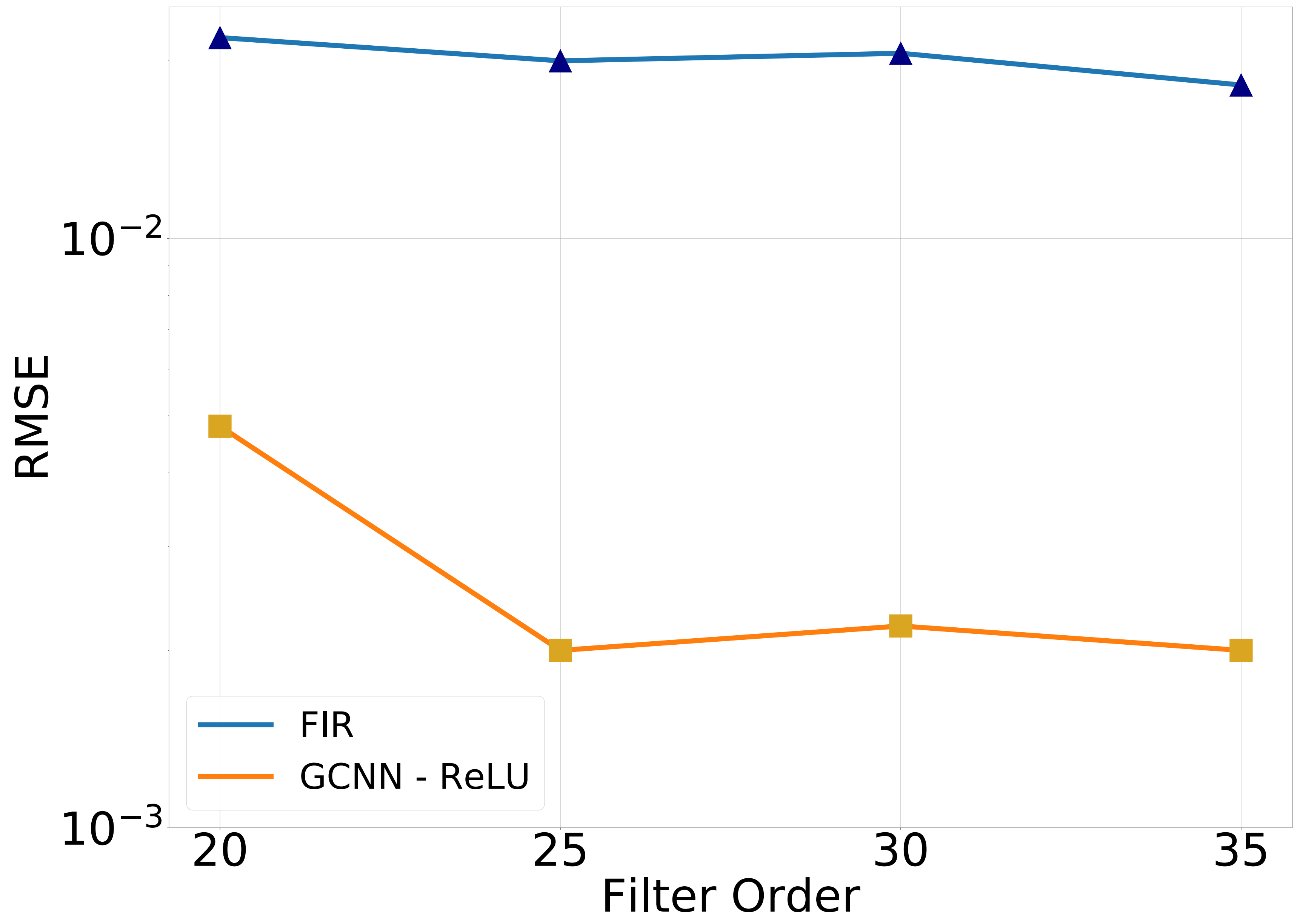
Iancu, Isufi, *Towards Finite-Time Consensus with Graph Convolutional Neural Networks*, EUSIPCO 2020 (submitted)

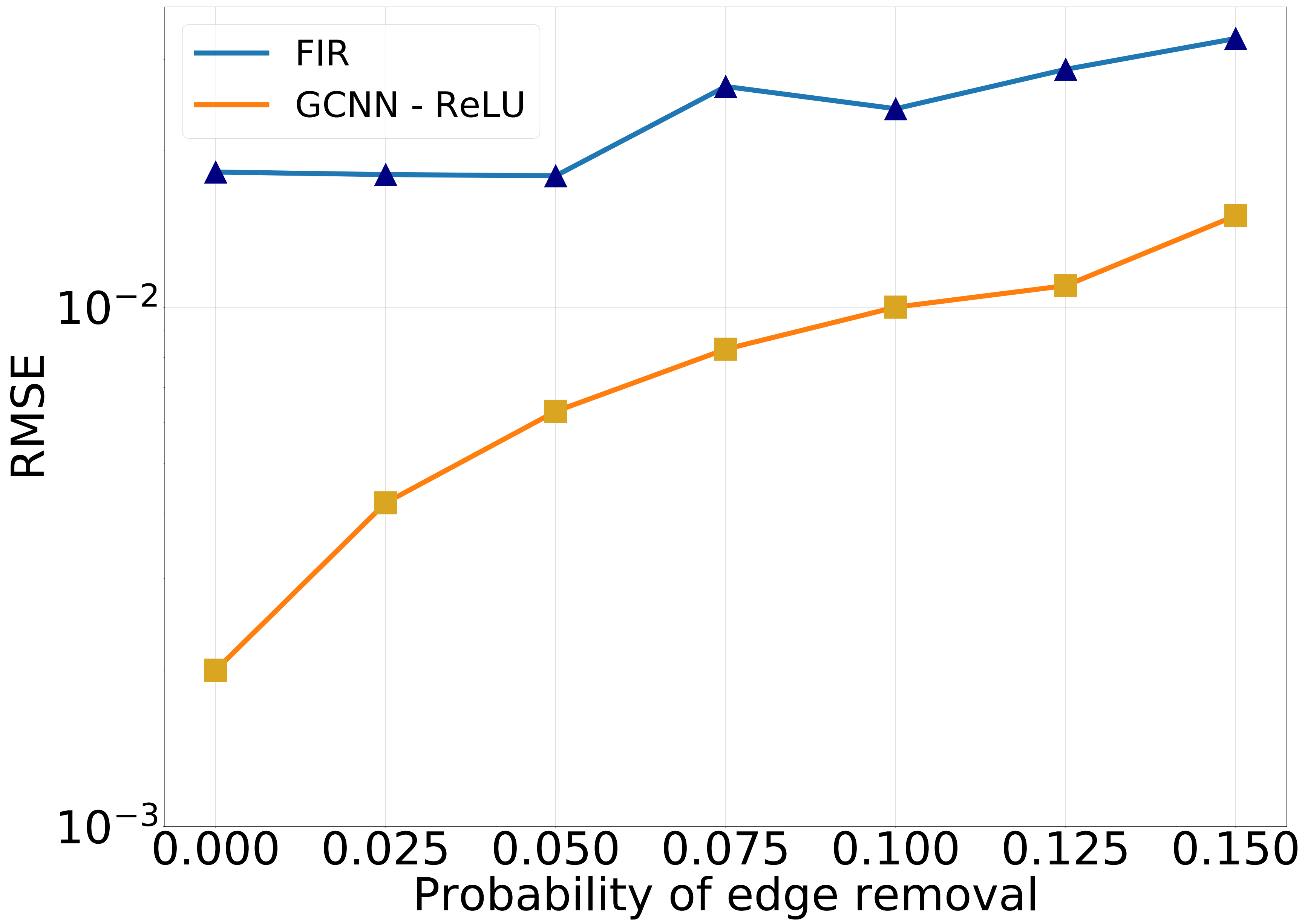
Learning finite-time consensus



- Consensus is strictly low pass
- Better performance for high orders
- Machine precision needs EV

Learning finite-time consensus





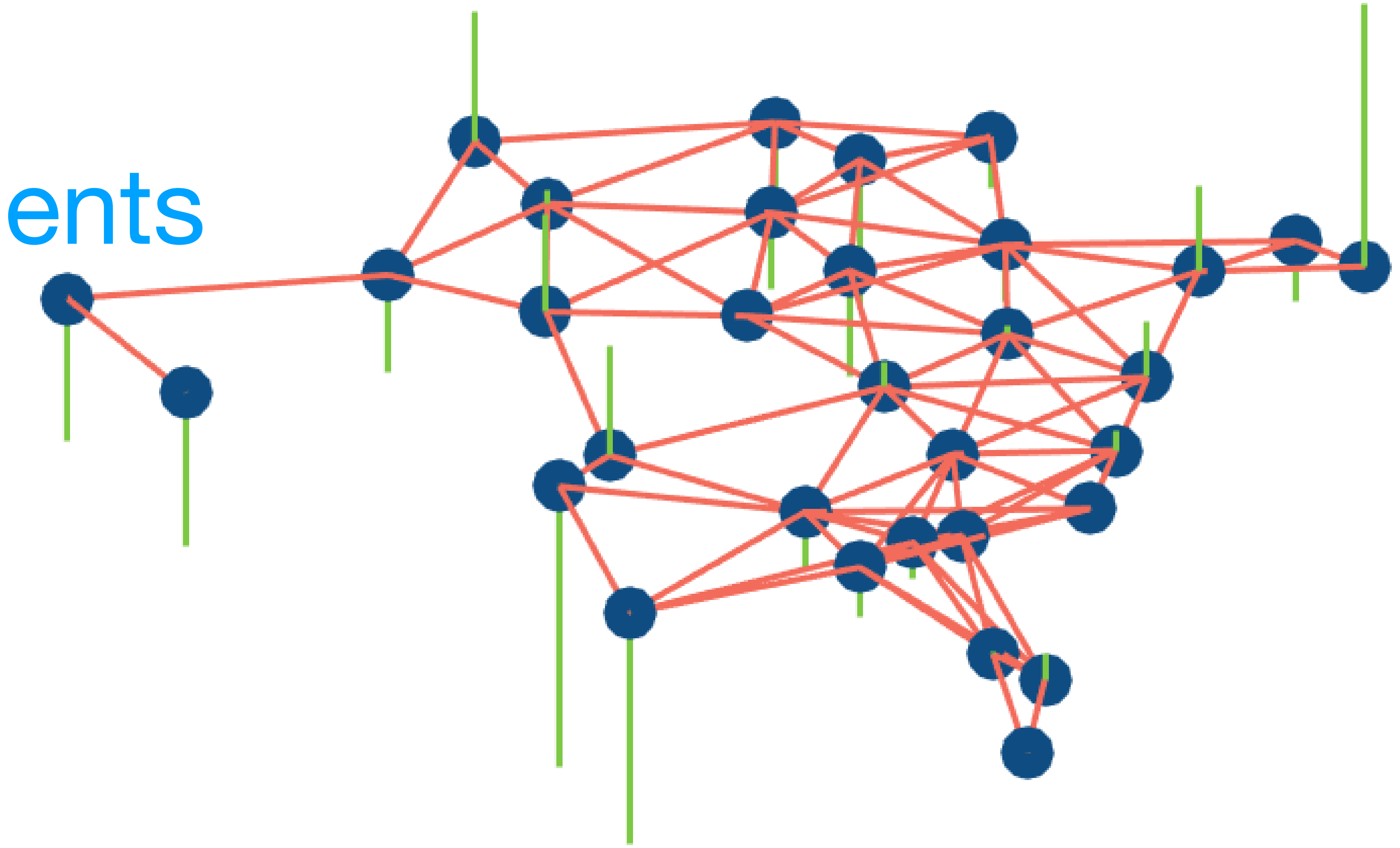
- Consensus is strictly low pass
- Better performance for high orders
- Machine precision needs EV

- Train and test on different graphs
- GCNN exploits better the connectivity
- GCNNs are better transferable

Levie, Isufi, Kutyniok, *On the Transferability of Spectral Graph Filters*, SAMPTA 2019.

Distributed regression

- Retrieve signal distributively from noisy measurements



Molene weather dataset

Build a graph between stations $N = 32$

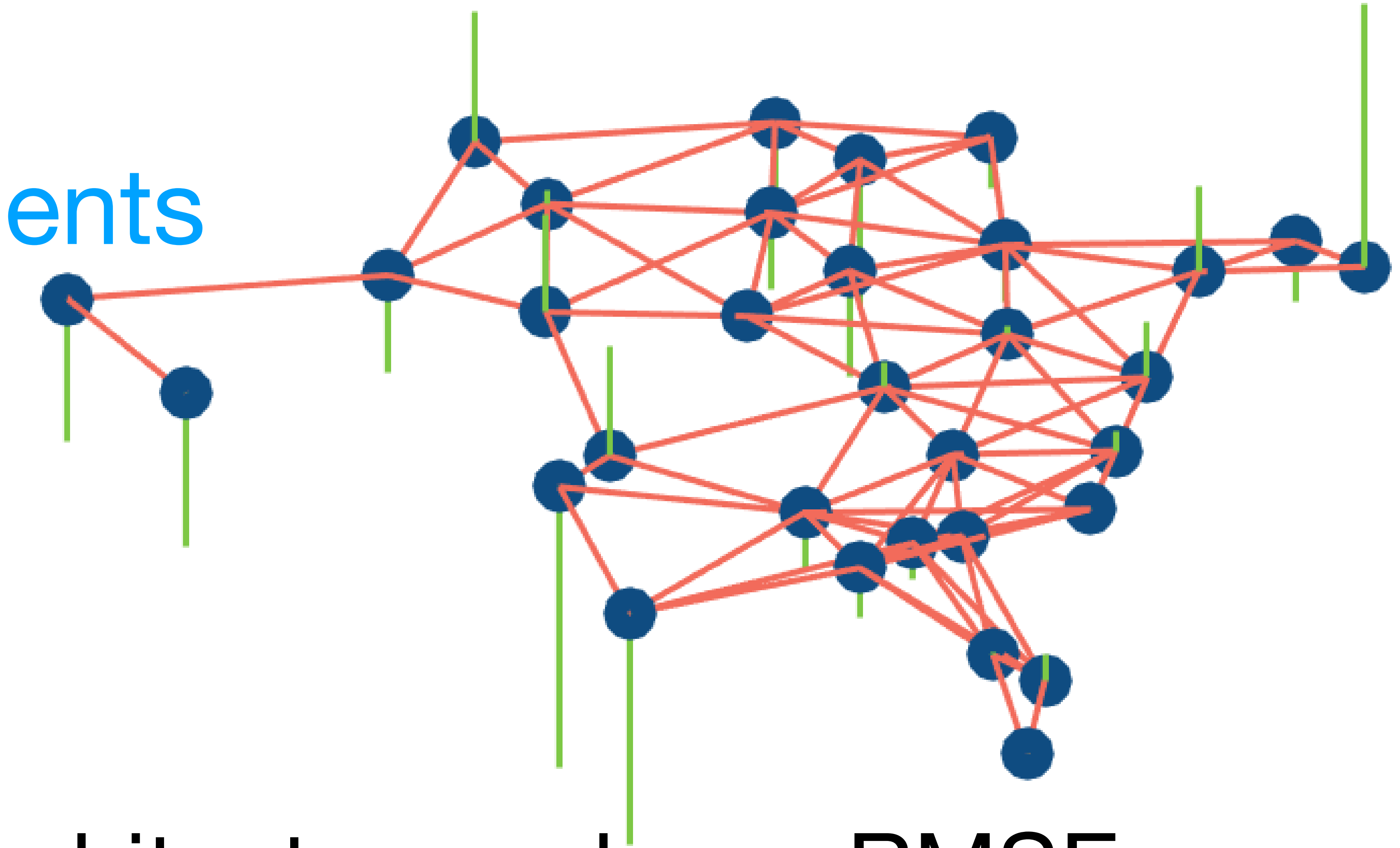
Graph signal: 744 temperature recording

$SNR = 3\text{dB}$ 1 layer; 4 features

Iancu, Ruiz, Ribeiro Isufi, *Distributed Localized Nonlinearities For Graph Neural Networks*, MLSP 2020 (submitted)

Distributed regression

◉ Retrieve signal distributively from noisy measurements

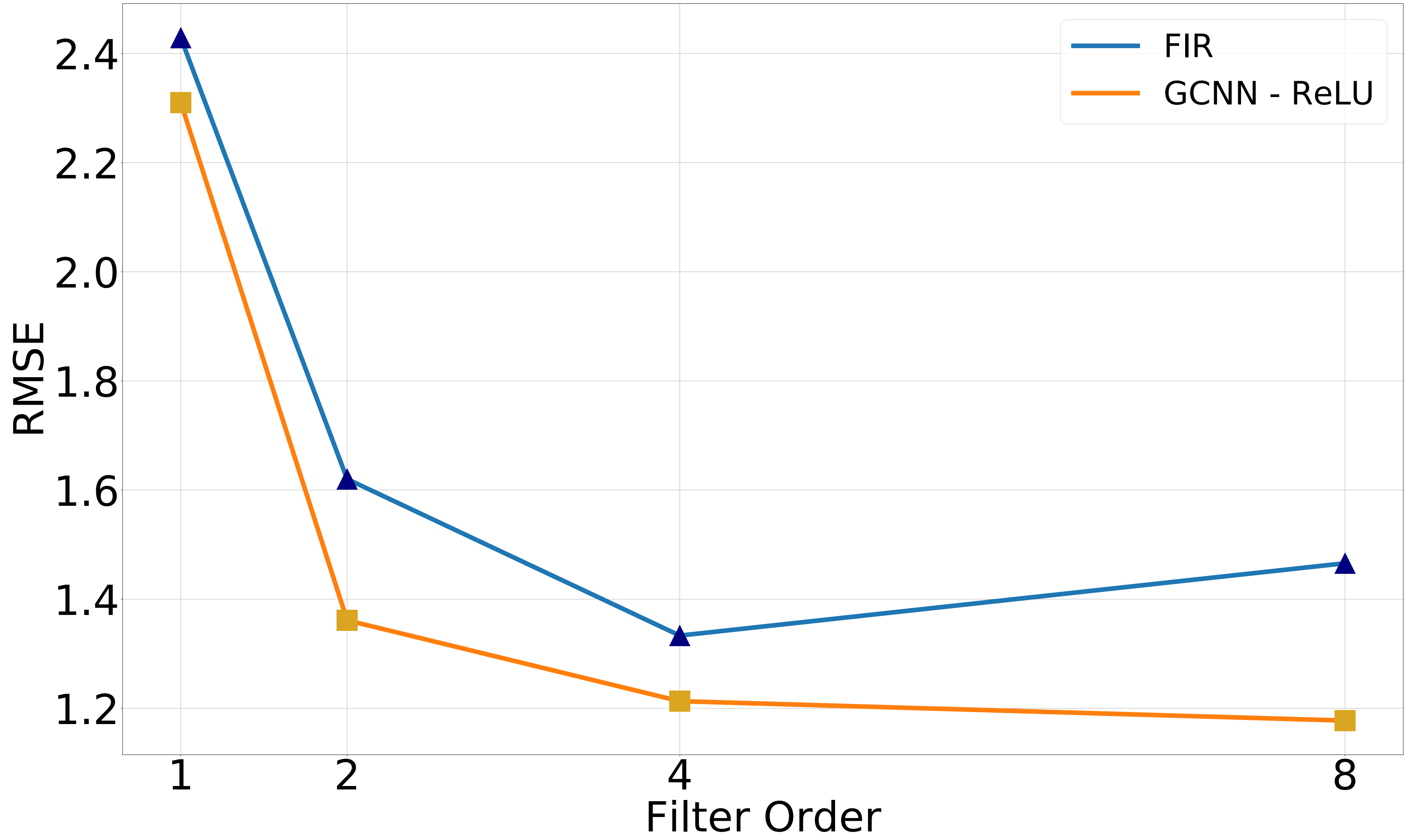


Molene weather dataset

Build a graph between stations $N = 32$

Graph signal: 744 temperature recording

$SNR = 3\text{dB}$ 1 layer; 4 features



- Nonlinear architecture reduces RMSE
- 4 times more communications
- Regression more challenging than classification
- Needs: more data/more graph prior

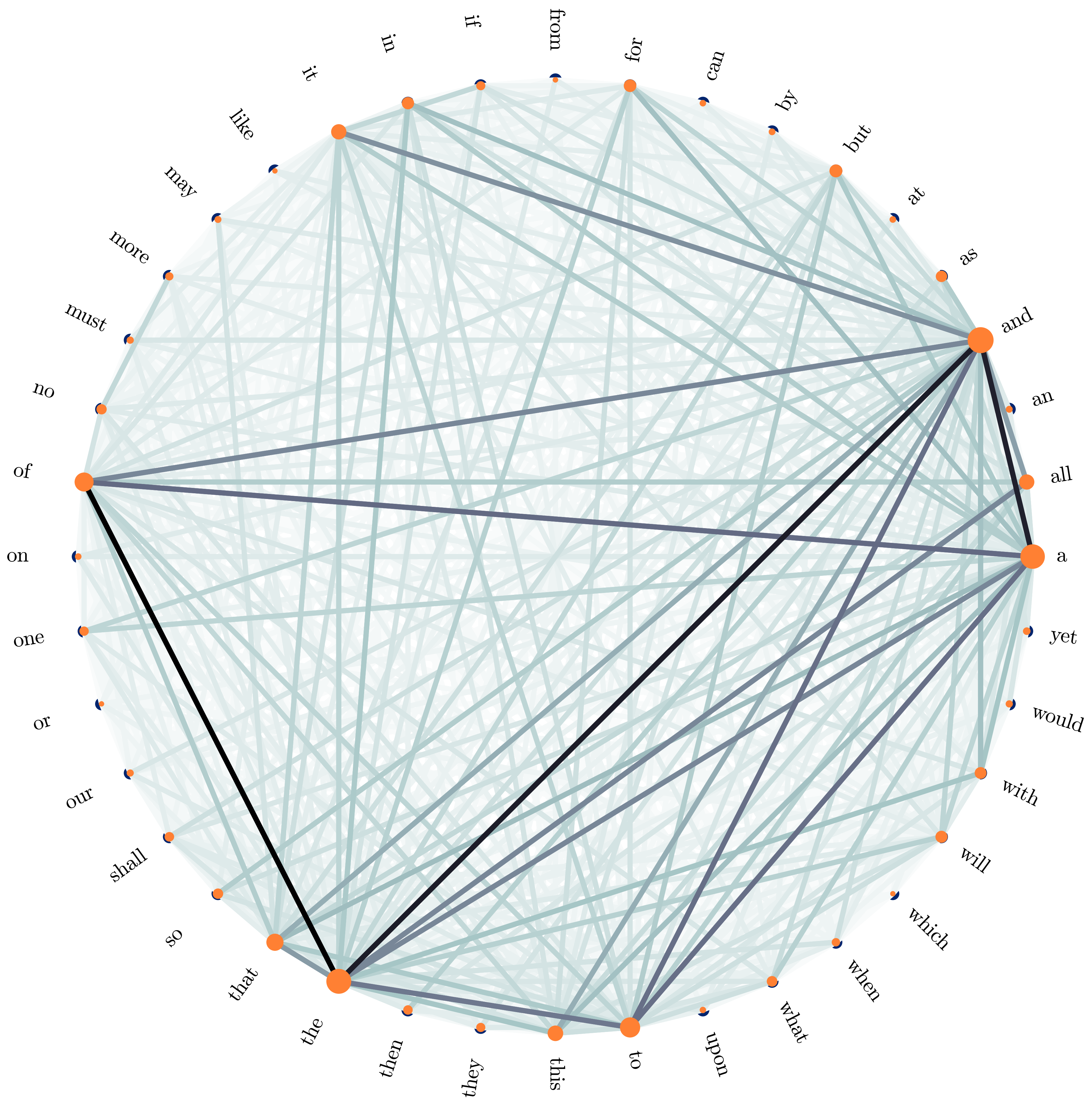
Iancu, Ruiz, Ribeiro Isufi, *Distributed Localized Nonlinearities For Graph Neural Networks*, MLSP 2020 (submitted)

Authorship attribution

- Attribute texts to an author [Segarra’15-TSP]

Build a word adjacency network

$$N = 190 - 211$$



[© figure Ruiz’19-TSP]

Authorship attribution

- Attribute texts to an author [Segarra'15-TSP]

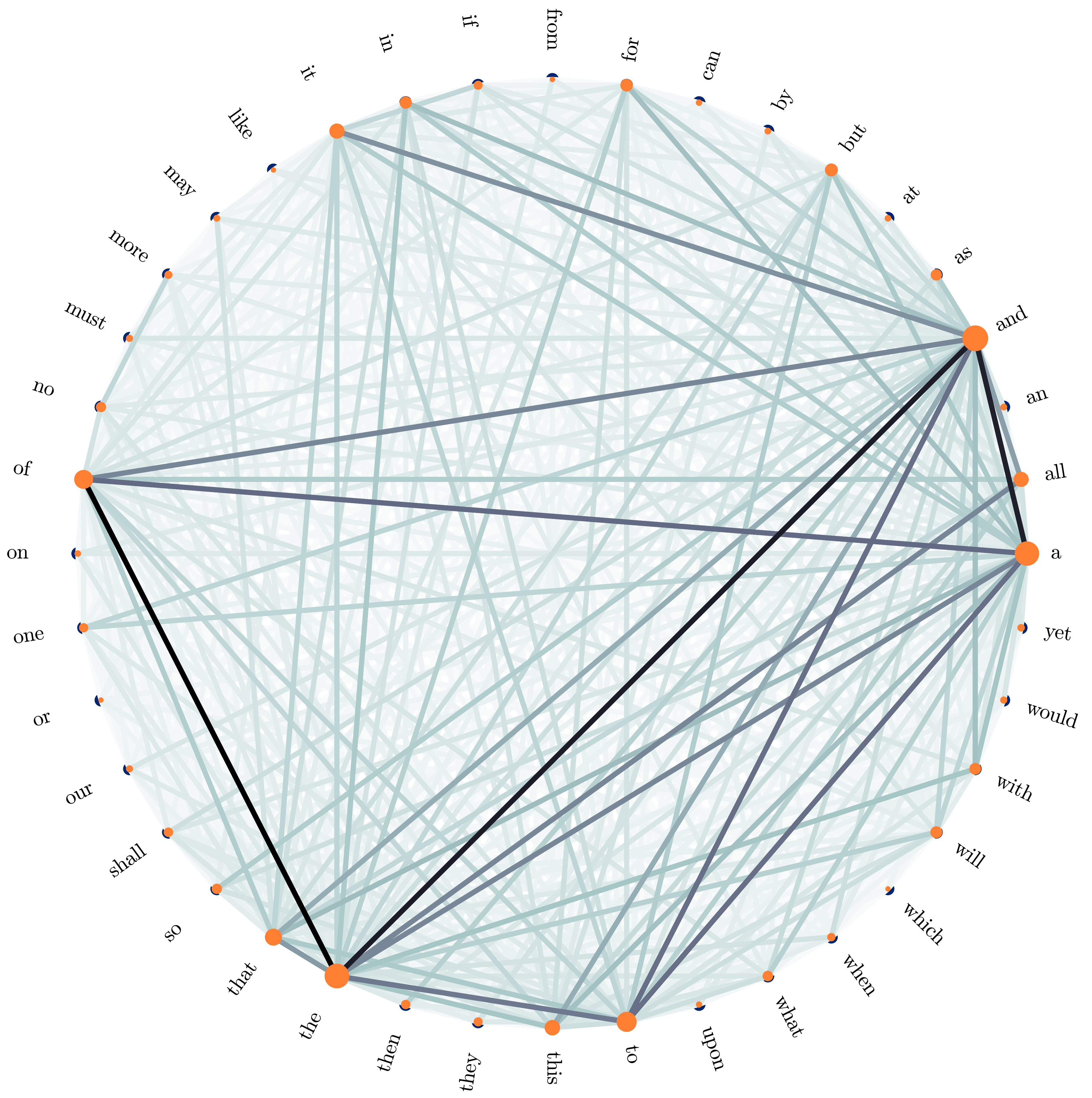
Build a word adjacency network

$N = 190 - 211$

Graph signal: word frequency count

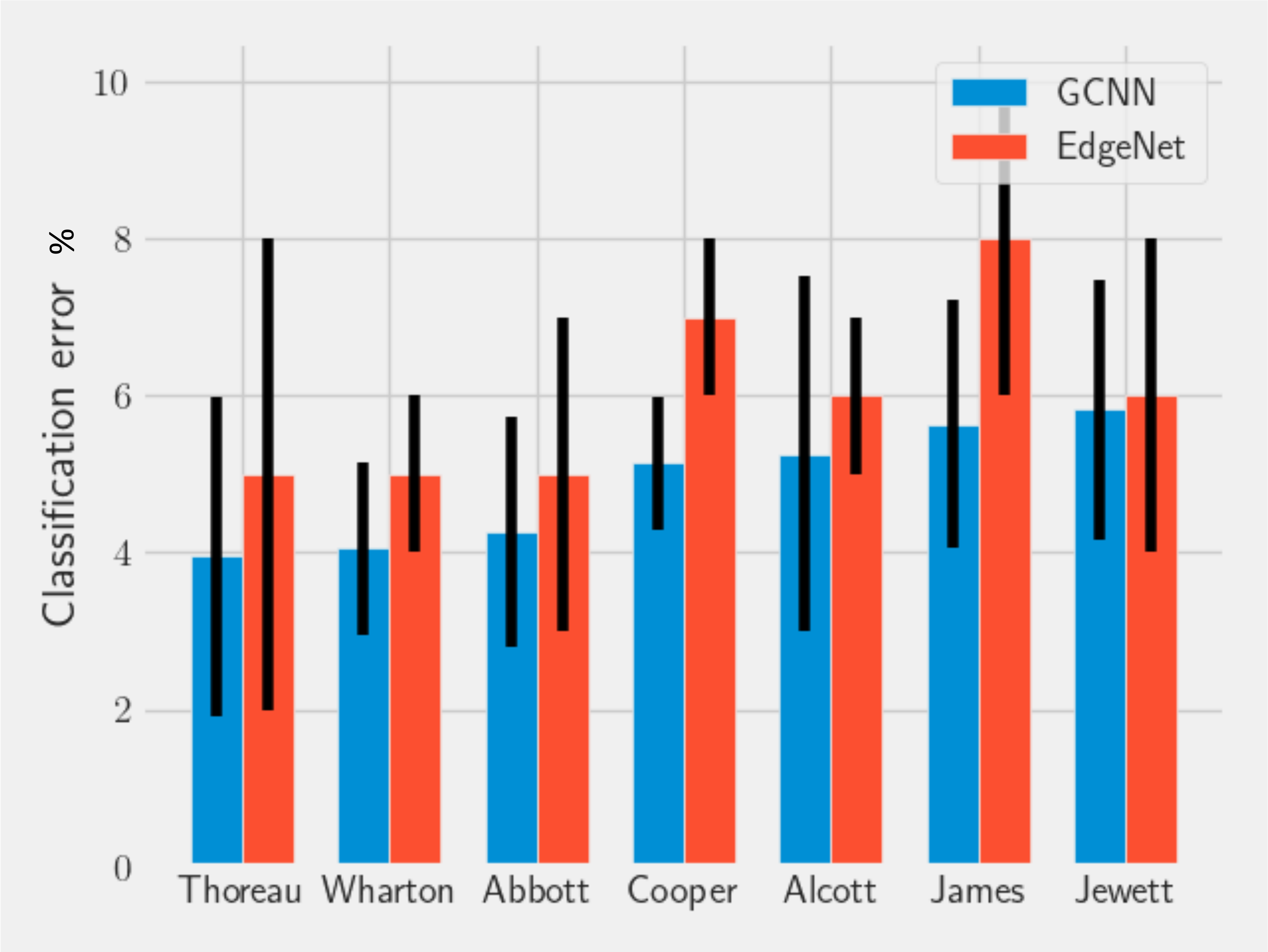
~ 1000 texts from the author of interest

~ 1000 from others



[© figure Ruiz'19-TSP]

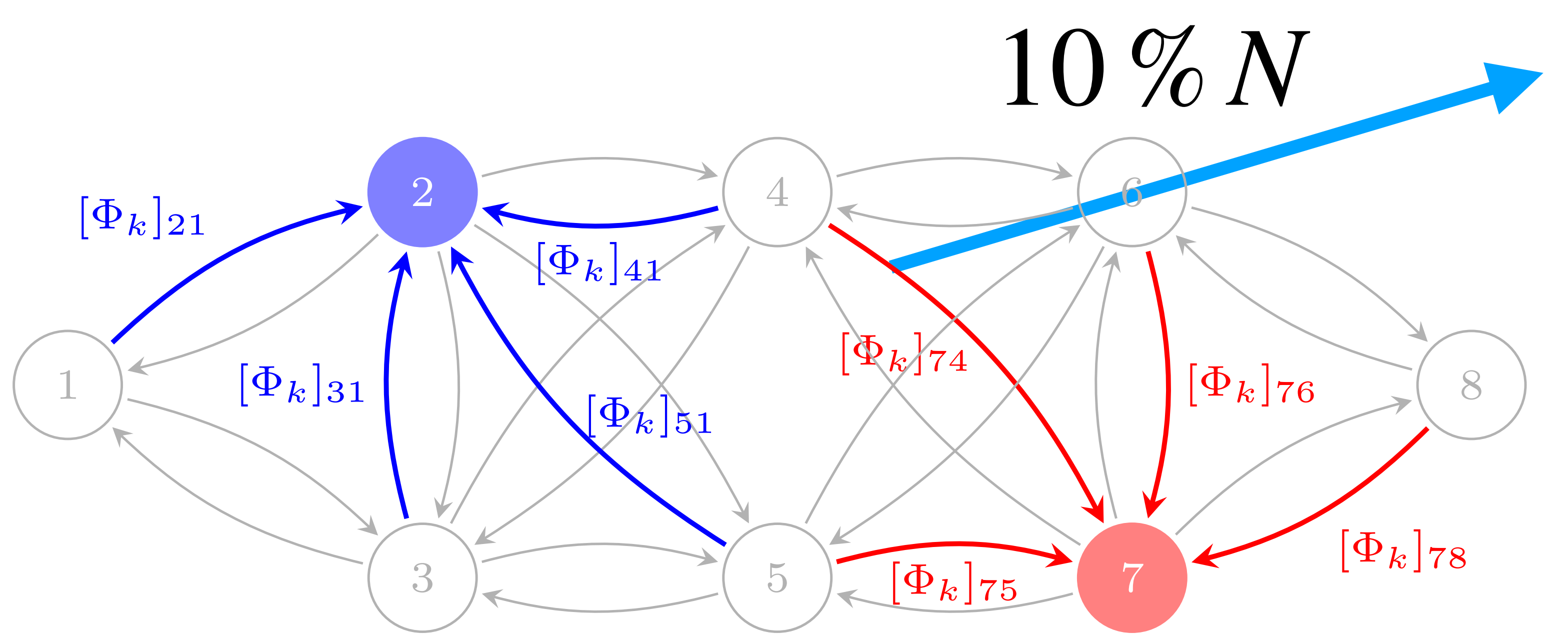
Authorship attribution (easier)



- EV hyperparameters (K, F, L) taken from the FIR
- Parameter sharing is beneficial

  1 layer, $K \in [2,10]$, $F \in \{16,32,64\}$

Authorship attribution (difficult)



Classification error

Architecture	Austen	Brontë	Poe
GCNN	7.2(±2.0)%	12.9(±3.5)%	14.3(±6.4)%
Edge varying	7.1(±2.2)%	13.1(±3.9)%	**10.7**(±**4.3**)%
Node varying	7.4(±2.1)%	14.6(±4.2)%	11.7(±4.9)%
Hybrid edge var.	**6.9**(±**2.6**)%	14.0(±3.7)%	11.7(±4.8)%
ARMANet	7.9(±2.3)%	**11.6**(±**5.0**)%	10.9(±3.7)%

1 layer, $F = 32$, $K = 4$

- EdgeNet requires its own hypertunning
- Better for more difficult scenarios
- Subclasses of the EV can perform better depending on problem difficulty

Isufi, Gama, Ribeiro, *EdgeNets: Edge Varying Graph Neural Networks*, arXiv: 2001.07620

Authorship attribution (explain)

- Explain GNNs with EdgeNets
  - One layer EdgeNet with order $K = 1$
  - Training the EdgeNet = learning graph weights

$$\mathbf{x}_1^1 = \sigma\left(\mathbf{\Phi}\mathbf{x}_1^0\right)$$

Authorship attribution (explain)

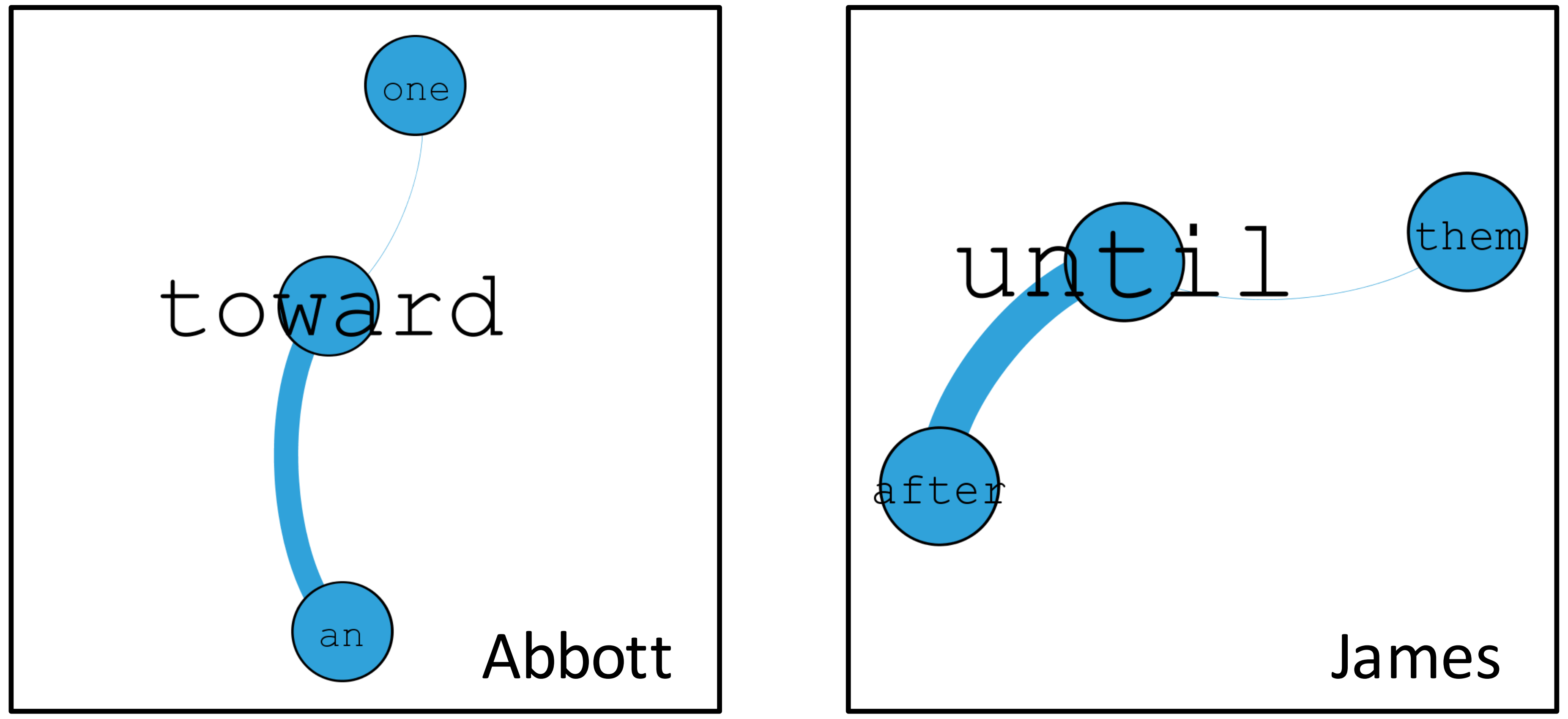
- Explain GNNs with EdgeNets
  - One layer EdgeNet with order $K = 1$
  - Training the EdgeNet = learning graph weights
    - removed small weight edges = accuracy drop $< 5\,\%$
    - identifies most relevant function words per author

$$\boxed{\mathbf{x}_1^1 = \sigma\left(\boldsymbol{\Phi}\mathbf{x}_1^0\right)}$$

Authorship attribution (explain)

$$\mathbf{x}_1^1 = \sigma\left(\mathbf{\Phi}\mathbf{x}_1^0\right)$$

- Explain GNNs with EdgeNets
  - One layer EdgeNet with order $K = 1$
  - Training the EdgeNet = learning graph weights
    - removed small weight edges = accuracy drop $< 5\,\%$
    - identifies most relevant function words per author

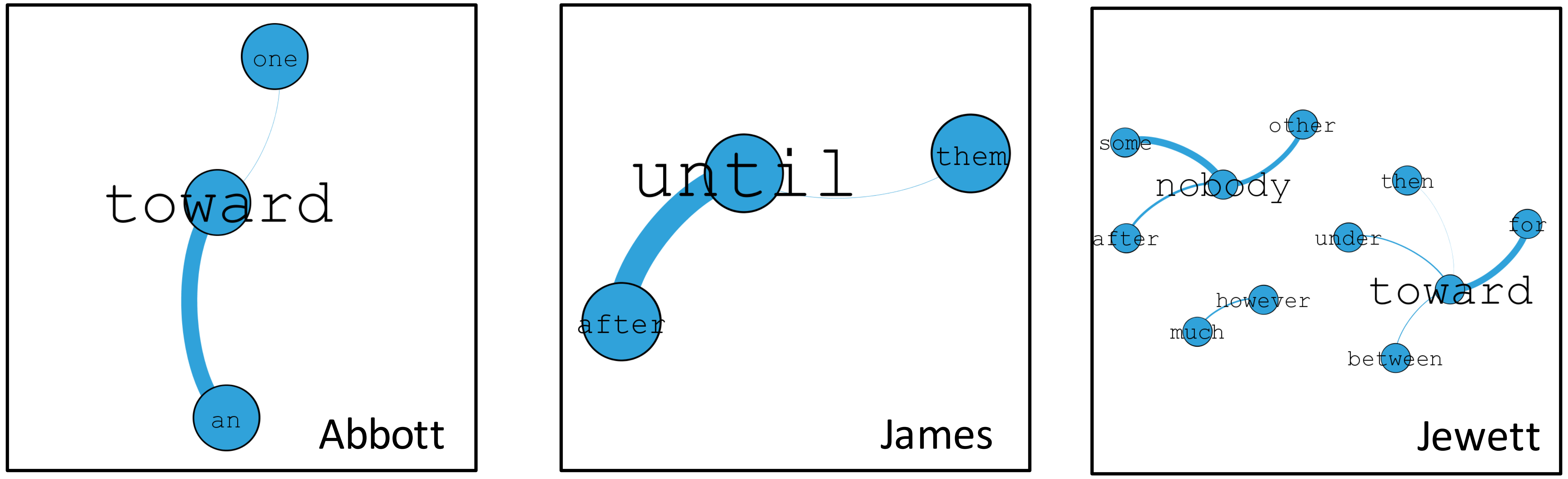


- identify an author from 3 words

Authorship attribution (explain)

- Explain GNNs with EdgeNets
  - One layer EdgeNet with order $K = 1$
  - Training the EdgeNet = learning graph weights
    - removed small weight edges = accuracy drop $< 5\,\%$
    - identifies most relevant function words per author

$$\mathbf{x}_1^1 = \sigma\big(\mathbf{\Phi}\mathbf{x}_1^0\big)$$

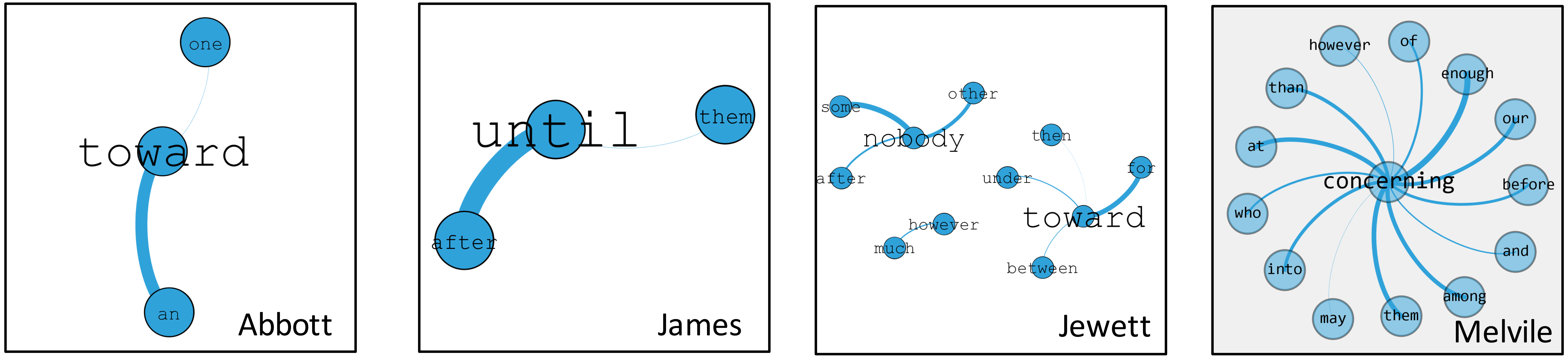


- identify an author from 3 words

Authorship attribution (explain)

$$\mathbf{x}_1^1 = \sigma\left(\boldsymbol{\Phi}\mathbf{x}_1^0\right)$$

- Explain GNNs with EdgeNets
  - One layer EdgeNet with order $K = 1$
  - Training the EdgeNet = learning graph weights
    - removed small weight edges = accuracy drop $< 5\,\%$
    - identifies most relevant function words per author



    - identify an author from 3 words

Authorship attribution

- Identifying author gender from texts
  - No NLP: shallow and fast training, no pretraining/corpus
  - Graphs + signals from female and male authors in train - test

Authorship attribution

- Identifying author gender from texts
  - No NLP: shallow and fast training, no pretraining/corpus
  - Graphs + signals from female and male authors in train - test

Classification error

	EdgeNet	GCNN	EV-GCNN
Mean	8.6%	10.1%	7.8%
Std	6x10^{-3}	6x10^{-3}	5x10$^{\wedge -3}$

1 layer architectures, $F = 64$

Sparse WANs help classification

Sparse EV shift operator + GCNN

Recommender systems

- Fill missing entries in a user-item matrix

Movielens $100K$ dataset $U = 943; I = 1{,}582$

Build a similarity graph (principle of collaborative filter)

	Item 1	**Item 2**	**Item 3**	**…**	**Item I**
User 1	**5**	**1**	**?**	**…**	**2**
User 2	**?**	**?**	**3**	**…**	**3**
User 3	**4**	**?**	**4**	**…**	**?**
…	**…**	**…**	**…**	**…**	**…**
User U	**?**	**4**	**3**	**…**	**1**

Recommender systems

- Fill missing entries in a user-item matrix

Movielens $100K$ dataset $U = 943; I = 1{,}582$

Build a similarity graph (principle of collaborative filter)

	Item 1	Item 2	Item 3	…	Item I
User 1	5	1	?	…	2
User 2	?	?	3	…	3
User 3	4	?	4	…	?
…	…	…	…	…	…
User U	?	4	3	…	1

user similarity graph

nodes : users

edges : Pearson/cosine similarity

between pairs of users

Recommender systems

- Fill missing entries in a user-item matrix

Movielens $100K$ dataset $U = 943; I = 1{,}582$

Build a similarity graph (principle of collaborative filter)

	Item 1	Item 2	Item 3	…	Item I
User 1	5	1	?	…	2
User 2	?	?	3	…	3
User 3	4	?	4	…	?
…	…	…	…	…	…
User U	?	4	3	…	1

user similarity graph

nodes : users

edges : Pearson/cosine similarity

between pairs of users

item similarity graph

nodes : items

edges : Pearson/cosine similarity

between pairs of items

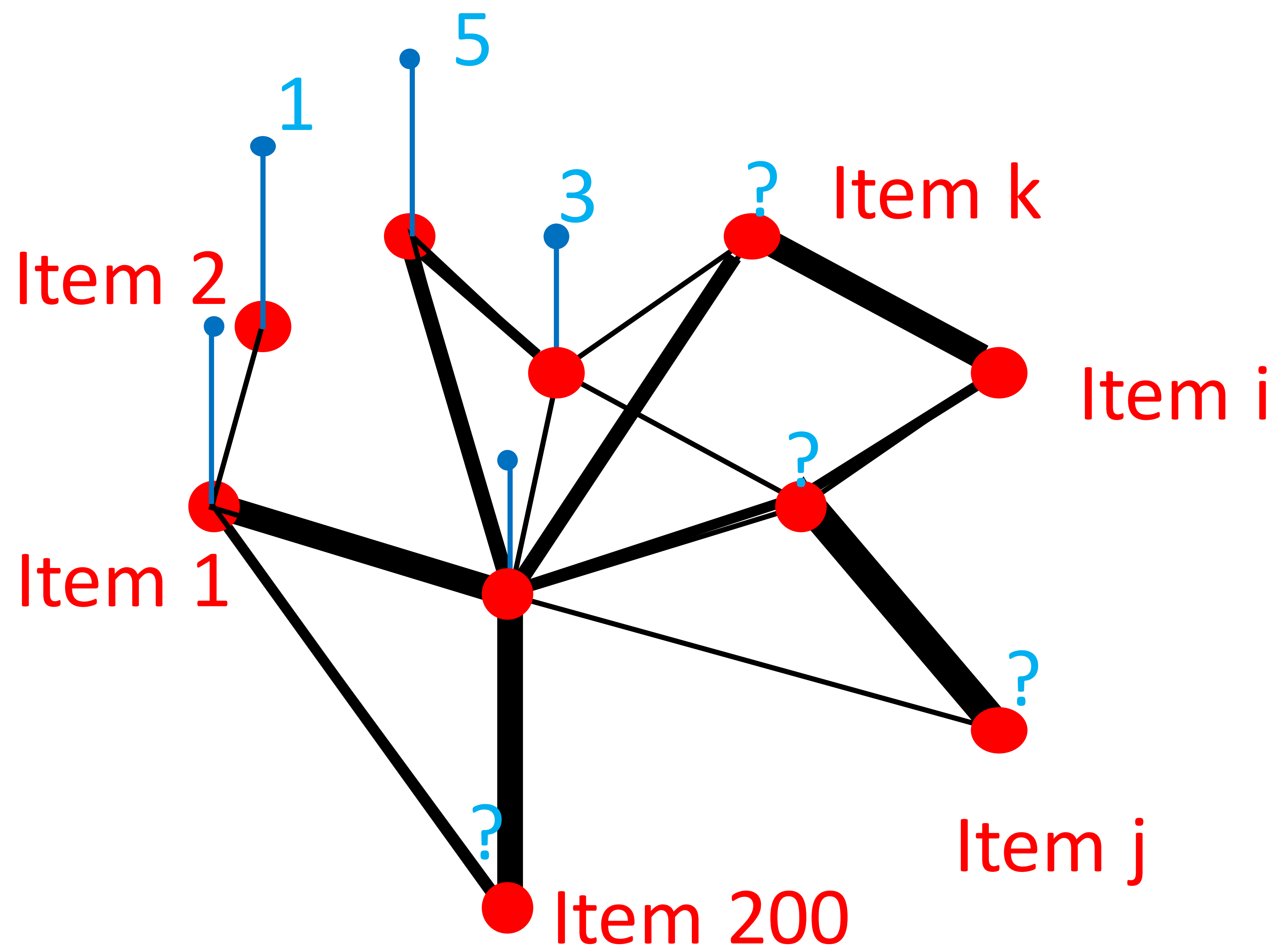
Recommender systems

- Here item similarity graph

$N = 200$ most rated items

	Item 1	Item 2	Item 3	…	Item I
User 1	5	1	?	…	2
User 2	?	?	3	…	3
User 3	4	?	4	…	?
…	…	…	…	…	…
User U	?	4	3	…	1

subset of user ratings
to build the graph



Graph signal : rating of user u to all items

- interpolation problem on graphs

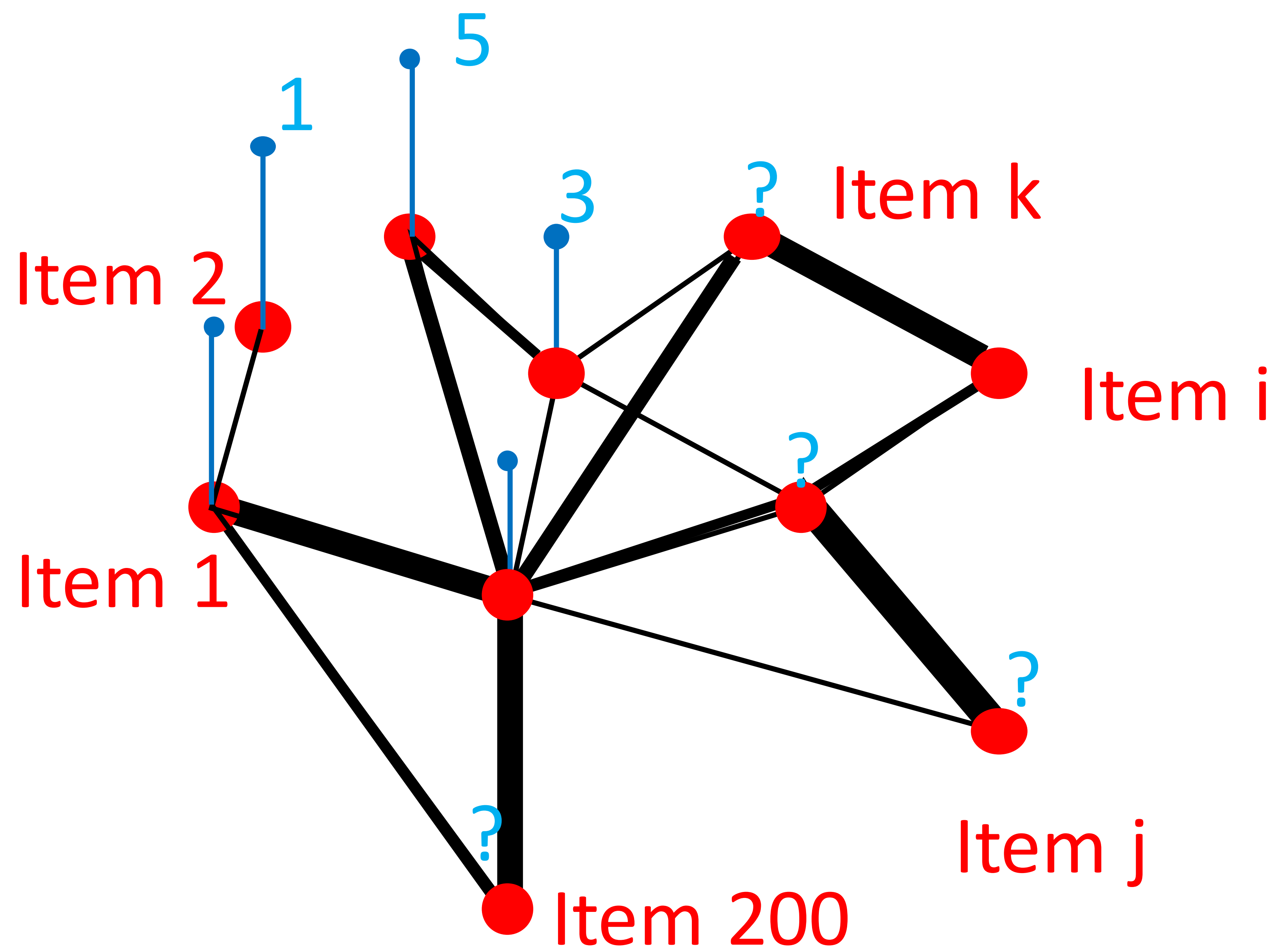
Recommender systems

- Here item similarity graph

$N = 200$ most rated items

	Item 1	Item 2	Item 3	…	Item I
User 1	5	1	?	…	2
User 2	?	?	3	…	3
User 3	4	?	4	…	?
…	…	…	…	…	…
User U	?	4	3	…	1

subset of user ratings
to build the graph



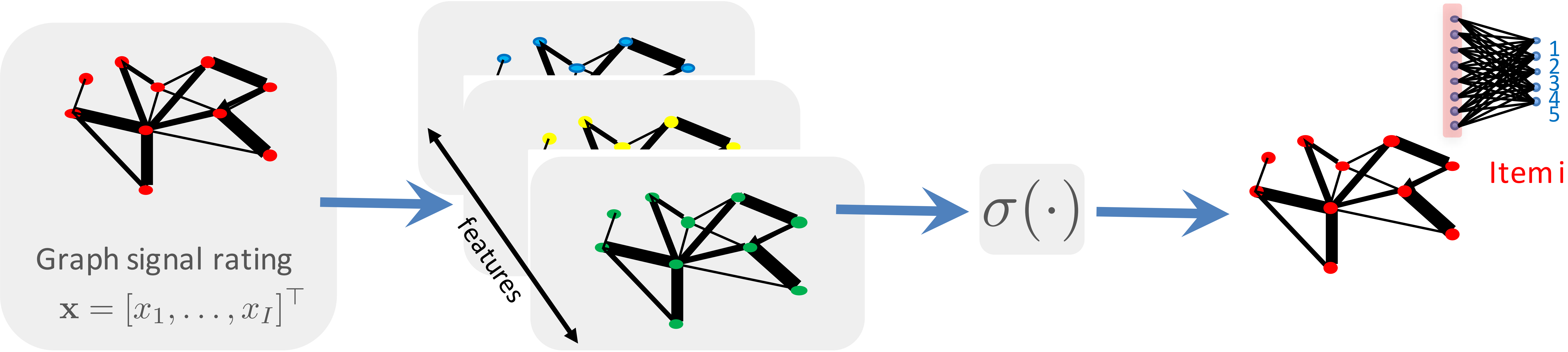
Graph signal : rating of user u to all items

- interpolation problem on graphs

Goal: find rating all users give to item i (fii ith column of matrix)

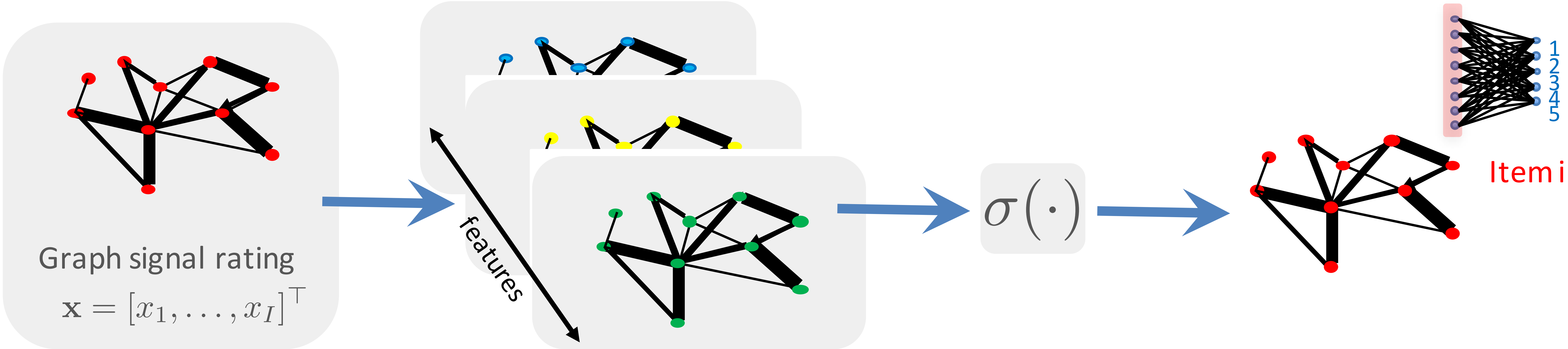
Recommender systems

- Use locality of the filters to build a GNN specific to item I



Recommender systems

- Use locality of the filters to build a GNN specific to item I



Frame as signal classification problem per node

1 layer, 32 features

EdgeNet suffers in general -requires parameterization

Archit./Movie-ID	50	258	100	181	294	Average
GCNN	**0.82**	1.08	**0.95**	0.86	1.04	0.95
Edge var.	0.93	**1.03**	1.00	0.88	1.24	1.02
Node var.	0.78	1.04	1.00	0.87	**1.00**	**0.94**
Hybrid edge var.	0.75	**1.02**	0.98	**0.82**	1.08	**0.93**

part 4 :: conclusions

- Graph filter are the building block of graph neural network (GNN)
  - Incorporate effectively the graph signal - graph topology into learning
  - Serve as a prior to reduce parameters and complexity
  - Graph convolutions through graph filters

part 4 :: conclusions

- Graph filter are the building block of graph neural network (GNN)
  - Incorporate effectively the graph signal - graph topology into learning
  - Serve as a prior to reduce parameters and complexity
  - Graph convolutions through graph filters
- Different filter = different graph neural networks
  - FIR = GCNNs
  - ARMA = ARMANets
  - Edge varying = EdgeNets

part 4 :: conclusions

- EdgeNets provide the broadest GNN family
  - Particularize to all the others including GINs and GATs
  - Help explainability

part 4 :: conclusions

- EdgeNets provide the broadest GNN family
  - Particularize to all the others including GINs and GATs
  - Help explainability
- Applications in signal classification & regression
  - Authorship attribution
  - Recommender systems

GNN - next challenges

- More graph prior instead of more data

GNN - next challenges

- More graph prior instead of more data
- Explainability
  - What topological information is more relevant?
  - What spectral information is more relevant?
  - EdgeNet can be a strong tool in this regard

GNN - next challenges

- More graph prior instead of more data
- Explainability
  - What topological information is more relevant?
  - What spectral information is more relevant?
  - EdgeNet can be a strong tool in this regard
- Robustness/Transferability
  - To topological perturbations
  - To input perturbations

GNN - next challenges

- More graph prior instead of more data
- Explainability
    - What topological information is more relevant?
    - What spectral information is more relevant?
    - EdgeNet can be a strong tool in this regard
- Robustness/Transferability
    - To topological perturbations
    - To input perturbations
- Distributed learning
    - Graph filters are distributable

Conclusions

- **Graph filtering** for denoising, interpolation, distributed optimization, GNNs
  - FIR graph filters
  - IIR - ARMA graph filters
- **Extensions** of FIR filters (can be used for IIR as well)
- **Edge-variant** graph filters **generalize** classical graph filters
  - reduction in communication and computation complexity
- **Easy design** using least squares
- **Applications**
  - Design of **low-order graph filters**
  - **Distributed optimization** solutions
  - **Graphical neural network** implementations

Questions?

{ g.j.t.leus; e.isufi-1; m.a.coutinominguez } @tudelft.nl

Thank you

References

Start with graph signal processing

- Ortega et al., Graph signal processing: Overview, challenges, and applications, Proceedings of the IEEE, 2018.
- Shuman et al., The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains, IEEE Signal Processing Magazine, 2013.
- Sandryhaila and Moura, Discrete signal processing on graphs, IEEE Transactions on Signal Processing, 2013.
- Sandryhaila and Moura, Discrete signal processing on graphs: Frequency analysis, IEEE Transactions on sSignal Processing, 2014.

Distributed aspects of graph filters

- Shuman et al., Chebyshev polynomial approximation for distributed signal processing, International Conference on Distributed Computing in Sensor Systems and Workshops, 2011.
- Sandryhaila et al, Finite-time distributed consensus through graph filters, IEEE International Conference on Acoustic, Speech and Signal Processing, 2014.
- Loukas et al., Distributed autoregressive moving average graph filters, IEEE Signal Processing Letters, 2015.
- Shi et al., Infinite impulse response graph filters in wireless sensor networks, IEEE Signal Processing Letters, 2015.
- Segarra et al., Center-weighted median graph filters, IEEE Global Conference on Signal and Information Processing, 2016.

References

- Isufi et al., Autoregressive moving average graph filters, IEEE Transactions on Signal Processing, 2017.
- Isufi et al., Autoregressive moving average graph filters – A stable distributed implementation, IEEE International Conference on Acoustic, Speech and Signal Processing, 2017.
- Isufi et al., Distributed sparsified graph filters for denoising and diffusion tasks, IEEE International Conference on Acoustic, Speech and Signal Processing, 2017.
- Segarra et al., Optimal grap-filter design and applications to distributed linear network operators, IEEE Transactions on Signal Processing, 2017.
- Coutino et al., Distributed edge-variant graph filters, IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, 2017.
- Chamon and Ribeiro, Finite-precision effects on graph filters, IEEE Global Conference on Signal Processing, 2017.
- Liu et al., Filter design for autoregressive moving average graph filters, IEEE Transaction on Signal and Information Processing over Networks, 2018.
- Isufi et al., Distributed Wiener-based reconstruction of graph signals, IEEE Statistical Signal Processing Workshop, 2018.
- Coutino et al., On the limits of finite-time distributed consensus through successive local linear operations, Asilomar Conference on Signals, Systems, and Computers, 2018.
- Coutino et al., Advances in distributed graph filtering, IEEE Transactions on Signal Processing, 2019.
- Ben Saad et al., Quantization analysis and robust design for distributed graph filters, IEEE Transactions on Signal Processing, 2019.

References

Start with graph neural networks

- Bronstein et al, Geometric deep learning: going beyond Euclidean data, IEEE Signal Processing Magazine, 2017.
- Wu et al, A comprehensive survey on graph neural networks, IEEE Transactions on Neural Networks and Learning Systems, 2020.
- Gama et al, Graphs, Convolutions, and Neural Networks, submitted to IEEE Signal Processing Magazine, 2020, arXiv: 2003.03777

Graph filters in graph neural networks

- Bruna et al., Spectral networks and locally connected networks on graphs, arXiv:1312.6203.
- Defferrard et al. Convolutional neural networks on graphs with fast localized spectral filtering, Conference on Neural Information Processing Systems, 2016.
- Kipf and Welling, Semi-supervised classification with graph convolutional neural networks, International Conference on Learning Representations, 2016.
- Velickovic et al, Graph attention networks, International Conference on Learning Representations, 2018.
- Levie et al., Cayleynets: Graph convolutional neural networks with complex rational spectral filters, IEEE Transactions on Signal Processing, 2018.
- Gama et al., Convolutional neural network architectures for signals supported on graphs, IEEE Transactions on Signal Processing, 2018.
- Gama et al., Convolutional neural networks via node-varying graph filters, IEEE Data Science Workshop, 2018.

References

- Bianchi et al., Graph neural networks with convolutional ARMA filters, arXiv: 1901.01343, 2019.
- Wijesinghe et al., DFNets: Spectral CNNs for graphs with feedback-looped filters, Conference on Neural Information Processing Systems, 2019.
- Ruiz et al., Invariance-preserving localized activation functions for graph neural networks, IEEE Transactions on Signal Processing, 2019.
- Isufi et al., Generalizing graph convolutional neural networks with edge-variant recursions on graphs, European Signal Processing Conference, 2019.
- Isufi et al., EdgeNets: Edge varying graph neural networks, arXiv: 2001.07620, 2020.
- Iancu and Isufi, Towards finite-time consensus with graph convolutional neural networks, submitted to European Signal Processing Conference 2020.
- Iancu et al., Distributed Localized Nonlinearities For Graph Neural Networks, submitted to IEEE International Workshop on Machine Learning for Signal Processing, 2020.

Robustness of graph filters and graph neural networks

- Isufi et al., Stochastic graph filtering on time-varying graphs, IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, 2015.
- Isufi et al., Filtering random graph processes over random time-varying graphs, IEEE Transactions on Signal Processing, 2017.
- Levie et al., On the transferability of spectral graph filters, International Conference on Sampling Theory and Applications, 2019.

References

- Gama et al., Stability properties of graph neural networks, arXiv:1905.04497, 2020.
- Gao et al., Stochastic graph neural networks, IEEE International Conference on Acoustic, Speech and Signal Processing, 2020.
- Ioannidis et al., Edge Dithering for Robust Adaptive Graph Convolutional Networks, arXiv:1910.09590, 2020

Other applications of graph filters

- Hammond et al., Wavelets on graphs via spectral graph theory, Applied and Computational Harmonic Analysis, 2011.
- Girault et al., Semi-supervised learning for graph to signal mapping: A graph signal Wiener filter interpretation, IEEE International Conference on Acoustic, Speech and Signal Processing, 2014.
- Eglimez and Ortega, Spectral anomaly detection using graph-based filtering for wireless sensor networks, IEEE International Conference on Acoustic, Speech and Signal Processing, 2014.
- Tremblay et al., Compressive spectral clustering, International Conference on Machine Learning, 2016.
- Isufi et al., 2-Dimensional finite impulse response graph-temporal filters, IEEE Global Conference on Signal and Information Processing, 2016.
- Marques et al., Stationary graph processes and spectral estimation, IEEE Transactions on Signal Processing, 2017.
- Ioannidis et al., Graph Neural Networks for Predicting Protein Functions, IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, 2017.
- Berberidis et al., Adaptive Diffusions for Scalable Learning over Graphs, IEEE Transactions on Signal Processing, 2018.
- Huang et al., Rating prediction via graph signal processing, IEEE Transactions on Signal Processing, 2018.

References

- Isufi et al., Forecasting time series with VARMA recursions on graphs, IEEE Transactions on Signal Processing, 2019.
- Sun et al., A graph signal processing framework for atrial activity extraction, European Signal Processing Conference, 2019.
- Nikolakopoulos et al., Personalized Diffusions for Top-N Recommendation, International Conference on Recommender Systems , 2019
- Sun et al., Graph-time spectral analysis for atrial fibrillation, Biomedical Signal Processing and Control, 2020.

Graph topology inference

- Segarra et al., Network topology inference from spectral templates, IEEE Transactions on Signal and Information Processing over Networks, 2017.
- Giannakis et al., Topology identification and learning over graphs: Accounting for nonlinearities and dynamics, Proceedings of the IEEE, 2018
- Mateos et al., Connecting the dots: Identifying network structure via graph signal processing, IEEE Signal Processing Magazine, 2019
- Coutino et al., State-space network topology identification from partial observations, IEEE Transactions on Signal and Information Processing over Networks, 2020.

Other referred papers

- Segarra et al., Authorship attribution through function word adjacency networks, IEEE Transactions on Signal Processing, 2015.