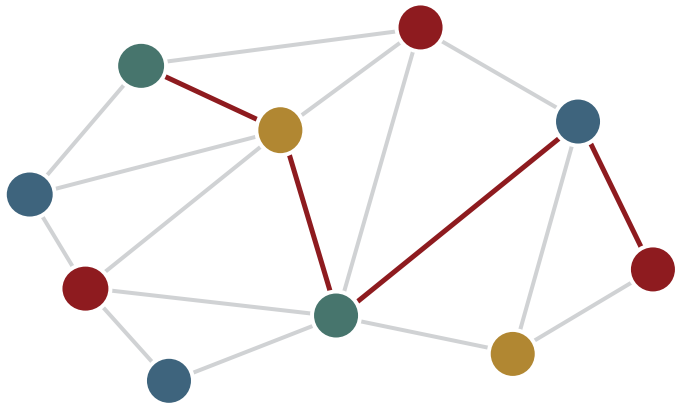
Graph Convolutions
Locality, shift, and permutation symmetries



September 1

Zhiyang Wang · Electrical & Systems Engineering · WashU

Class 3 · Graph convolutions and properties

Three parts: from the grid recipe to graph symmetries

01 **FROM $\boldsymbol{T}$ TO $\boldsymbol{S}$**

Classical convolution is **shift–scale–sum** on the **cycle graph**; swap with any graph shift operator $\boldsymbol{S}$.

02 **DEFINITION AND LOCALITY**

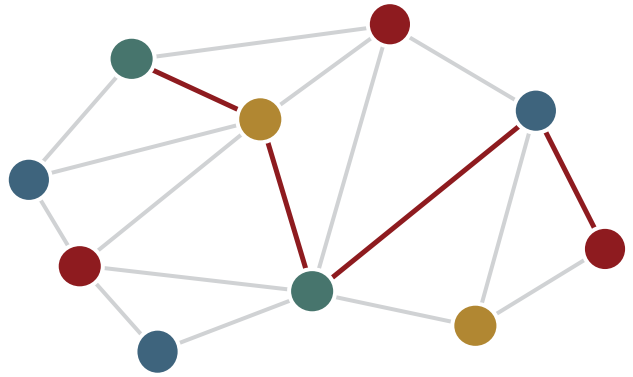
Polynomials in $\boldsymbol{S}$: K taps reach at most $K-1$ hops $(h_0, h_1 \ldots h_{K-1})$ — local behavior follows graph matrix

03 **SYMMETRIES, THEN SPECTRUM**

Commutation with $\boldsymbol{S}$, permutation equivariance, and the convolution theorem on graphs.

PART I

Convolution is shift, scale, sum.



01 Recall the grid recipe with shift T

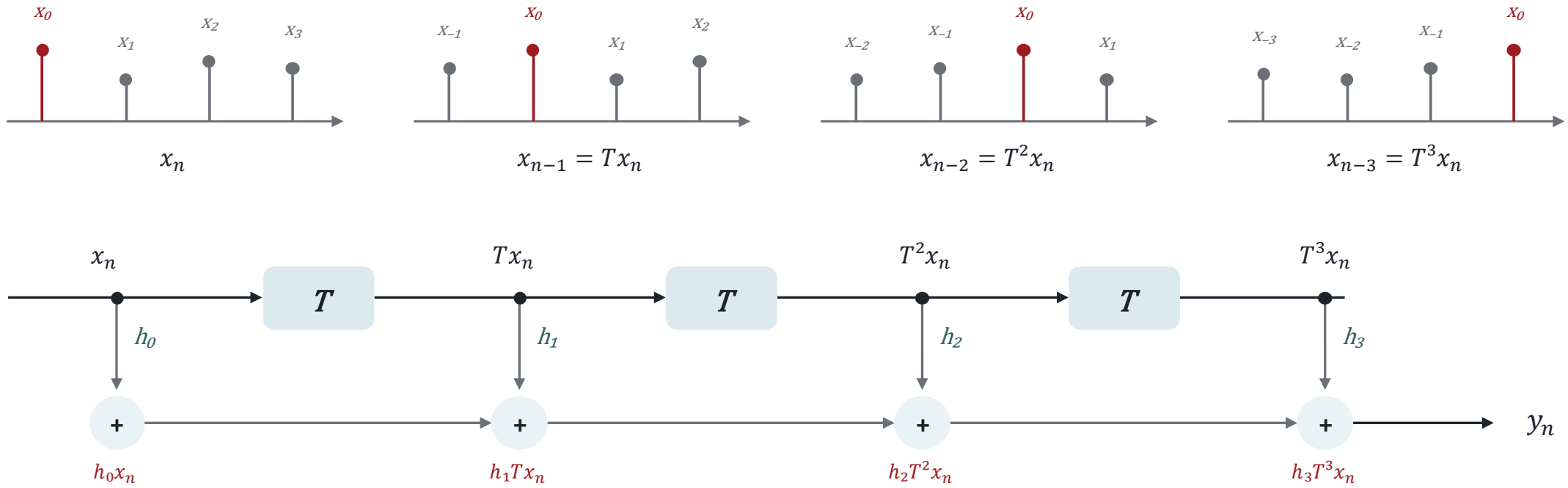
02 See T as a graph adjacency in a cycle graph

03 Swap in the graph shift S

04 Recognize smoothing as our first filter

Classical convolution is a special case of graph signal processing.

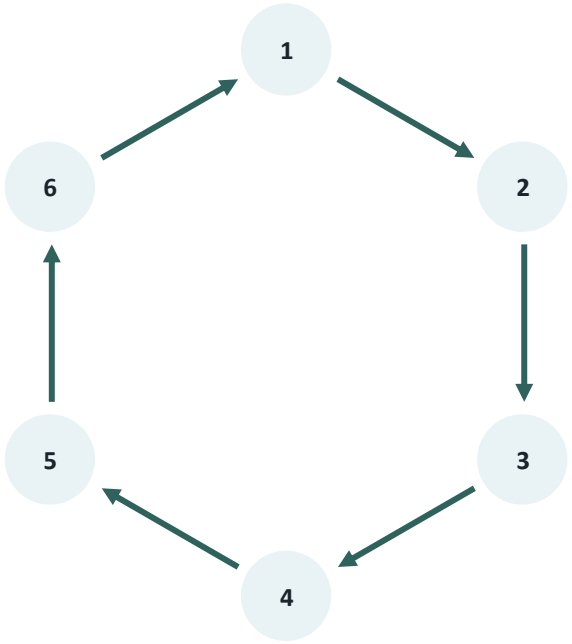
A standard convolution is shift, scale, sum



- $\boldsymbol{T}$ shifts the signal one step; the taps $h_0, \ldots, h_{k-1}$ weigh the shifted copies; the sum mixes them.
- A filter is a shared short list of taps ($h_0, \ldots, h_{k-1}$) — not one parameter per input–output pair.
- The same taps are reused at every position: that is the parameter sharing that makes convolution scale.

Three moves — shift, scale, sum — and one small shared filter.

The time shift is a graph shift



the directed cycle

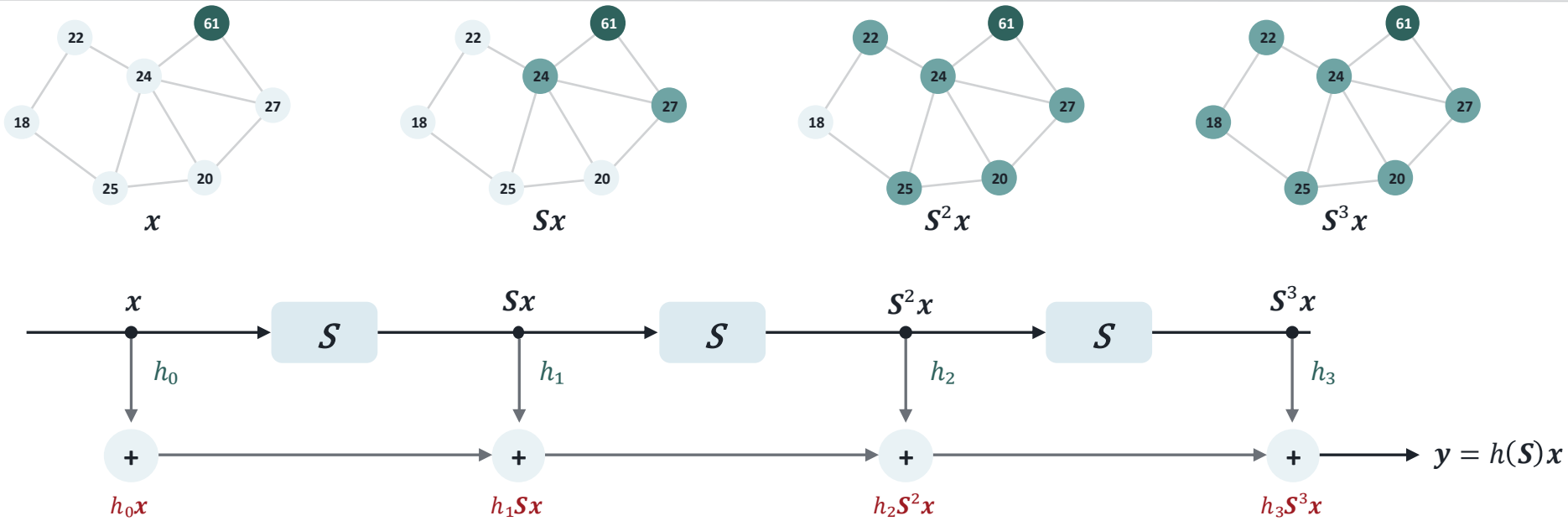
$$(\boldsymbol{T}\boldsymbol{x})_i = x_{i-1}$$

$$\boldsymbol{T} = \boldsymbol{A}_{cycle} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

- The one-step delay is exactly the adjacency matrix of this graph: each node hands its value to the next.
- So circular convolution, LTI filters, the DFT — all of it is ***graph signal processing on one special graph***.

Classical convolution = a polynomial in one particular graph shift.

Keep the recipe, generalize the graph



- S = any graph shift operator: adjacency A, Laplacian L, or a normalized variant
- Offset k on the line becomes hop radius k on the graph
- "same taps at every index" becomes "same taps at every node"
- **The coefficients define the filter. The operator defines the geometry.**

Erase the cycle assumption; keep everything else.

You have already run one

Last class's smoothing filter, reread with today's definitions:

$$(\boldsymbol{I} - \epsilon \boldsymbol{L})\boldsymbol{x} = h_0\boldsymbol{x} + h_1\boldsymbol{L}\boldsymbol{x}, \qquad (h_0, h_1) = (1, -\epsilon)$$

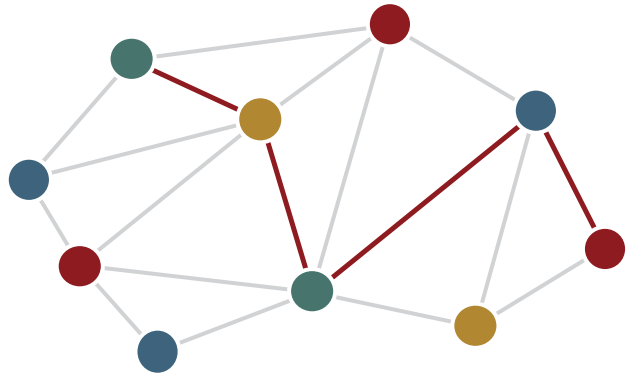
- A two-tap graph convolution with $\boldsymbol{S} = \boldsymbol{L}$.
- Smoothing as an example of filtering: aggregation, weighted and summed.

Aim of graph filtering:

- Process graph signals to desired outputs
- Learn representations of graphs or graph signals -- embeddings

PART II

Polynomials in S stay local.



01 Powers of S reach farther hop by hop

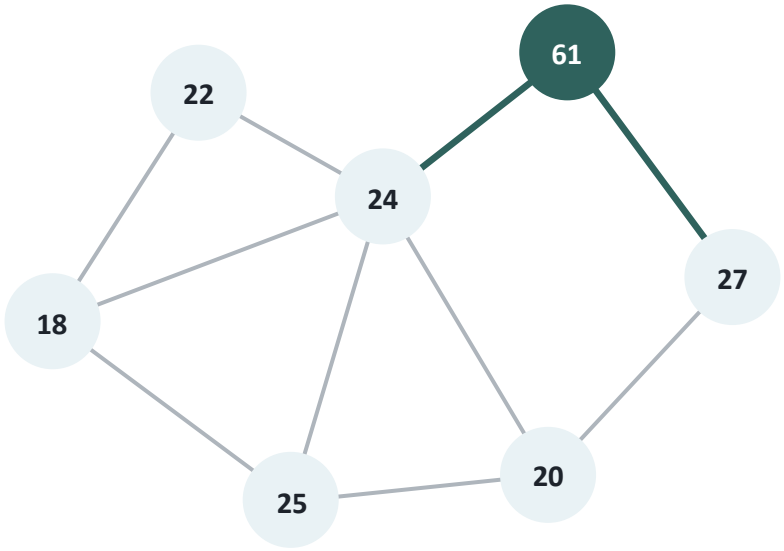
02 Define the graph convolution

03 Compute one by hand on the path

04 Read locality straight from graph matrix

K taps ⇒ information travels at most $K-1$ hops.

Powers of S expand the receptive field



$$\left(\mathbf{S}^k\right)_{ij} \neq 0 \Rightarrow d(i,j) \leq k$$

- $k = 0$: each node sees itself
- $k = 1$: one-hop neighbors — a single aggregation round
- $k = 2$: neighbors of neighbors — for the dark node, two hops already cover most of this graph.

$$(\boldsymbol{S}^2)_{\mathrm{ij}} = \sum_m \boldsymbol{S}_{\mathrm{im}} \boldsymbol{S}_{\mathrm{mj}} \qquad \text{Node } i \leftrightarrow \text{Node } m \leftrightarrow \text{Node } j$$

Each multiplication by S crosses at most one edge.

One rule, three popular shifts

Any matrix that respects the edges qualifies. On the path with $x = (1, 2, 2, 5)$, one step of each:

COLLECT · $S = A$

$$(\boldsymbol{Ax})_i = \sum_{j \in \mathcal{N}_i} x_j$$

(2, 3, 7, 2)

sum what your neighbors have

AVERAGE · $S = D^{-1}A$

$$D^{-1/2}AD^{-1/2}$$

$$\left(\boldsymbol{D^{-1}Ax}\right)_i = \frac{1}{d_i} \sum_{j \in \mathcal{N}_i} x_j$$

(2, 1.5, 3.5, 2)

their mean opinion

COMPARE · $S = L$

$$(\boldsymbol{Lx})_i = \sum_{j \in \mathcal{N}_i} (x_i - x_j)$$

(−1, +1, −3, +3)

your disagreement with them

All three obey $S_{ij} \neq 0$ only across edges — so permutation theorem today applies to each.

The theory is shift-agnostic; the semantics is your choice.

Weighted edges change while locality survives

$$(\boldsymbol{S}\boldsymbol{x})_i = \sum_{j \in \mathcal{N}_i} w_{ij} x_j$$

- A strong edge is a loud neighbor: w_{ij} scales how much of x_{j} node i hears.
- On the road network: lane count or capacity as the weight — a highway link counts more than a side street.
- Locality and the symmetry proofs hold unchanged if we have weighted graphs.

Weights change the arithmetic while theorem holds.

Compute one by hand: the path again

$$\boldsymbol{Sx} = (2, 3, 7, 2)$$
$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{Sx} = (3, 5, 9, 7)$$

- Taps $(h_0, h_1) = (1,1)$, shift $\boldsymbol{S}$ = adjacency.
- Check node 3: $2 + (2 + 5) = 9$.

Same graph, same signal (1, 2, 2, 5) as the Laplacian class — one running example, three operators.

One line to compute the convolutions.

The same computation, in matrix form

Write the path's shift out entry by entry — row i answers "who does node i listen to?":

$$\boldsymbol{Sx} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 2 \\ 5 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 7 \\ 2 \end{pmatrix}, \qquad \boldsymbol{y} = \boldsymbol{x} + \boldsymbol{Sx} = \begin{pmatrix} 3 \\ 5 \\ 9 \\ 7 \end{pmatrix}$$

- Row 3 has ones in columns 2 and 4: $(\boldsymbol{Sx})_3 = x_2 + x_4 = 7$ — re-read as one matrix–vector product.
- When we have F input features per node, $\boldsymbol{S}$ would still multiply across nodes.
- One matrix-vector product can be seen as one synchronous communication round.

A graph convolution is a matrix–vector product.

Add a tap, reach a hop

Grow the filter on the same input, taps all equal to 1:

FILTER	FORMULA	OUTPUT	REACH
K = 1	$y = x$	$(1,\ 2,\ 2,\ 5)$	self only
K = 2	$y = x + Sx$	$(3,\ 5,\ 9,\ 7)$	one hop of context
K = 3	$y = x + Sx + S^2x$	$(6, 14, 14, 14)$	two hops — nearly the whole path

- By $K = 3$ the outputs are almost level: on a 4-node path, two hops means nearly everyone has heard nearly everyone. More taps trade local detail for consensus.

K is the knob between local detail and global consensus. h_k decides what's emphasized inside.

Aggregating past the graph's diameter

Shift = $\boldsymbol{L = D - A}$, the combinatorial Laplacian.
Diameter is 3, so $K = 4$ reaches all; iterate $\boldsymbol{y} \leftarrow (\boldsymbol{I} - \varepsilon \boldsymbol{L})\boldsymbol{y}$, $\varepsilon = 0.25$:

k = 1	*(1.75, 1.75, 2.75, 3.75)*
k = 4	*(1.89, 2.20, 2.73, 3.17)*
k = 8	*(2.16, 2.36, 2.64, 2.84)*
k = 16	*(2.40, 2.46, 2.54, 2.60)*
k → ∞	*(2.50, 2.50, 2.50, 2.50)*

the plain mean ($\lambda = 0$ mode) — the only surviving mode

- **Reach saturates**: a K-tap filter mixes exactly the K-hop neighborhood, so once $K \geq$ diameter every pair is coupled.
- **Powers concentrate**: the gains $(1 - \varepsilon\lambda)^{\mathrm{k}}$ collapse to a spike at $\lambda = 0$ — at $k = 8$ the four modes read $1, 0.28, 0.004, 0$.
- No new directions past n powers: with n nodes, $\boldsymbol{L}^{\mathrm{n}}$ is a linear combination of $\boldsymbol{I}, \boldsymbol{L}, \ldots, \boldsymbol{L}^{\mathrm{n}-1}$(Cayley–Hamilton).

Foreshadow: stacked aggregation layers **collapse node representations** — oversmoothing, returning in future lectures.

More taps than diameter buys concentration, not reach.

Local behavior follows graph matrix

$$d(i,j) > K-1 \Rightarrow \left(h(\mathbf{S})\right)_{ij} = 0$$

- Node i's output depends on nodes within $K-1$ hops — a finite neighborhood, like a finite kernel window.
- No dense $n \times n$ parameter table anywhere: K numbers serve a graph of any size.
  - One of the biggest advantage brought by GNNs – parameter efficiency
- Filter order is design choice: each extra tap is one more round of local communication over the edges.

A K-tap filter sees K−1 hops — and those same K numbers serve a graph of any size.

Boundaries handle themselves

- Classical convolution has **a boundary problem**:

  at the ends of the signal the kernel hangs off, and you must invent data

  — zero-padding, reflection, wrap-around.

- A graph convolution -- the neighbor list *is* the boundary handling.

  End nodes of the path simply have one neighbor; the sum runs over whoever exists.

- Degree plays the role of boundary geometry — which you have already seen in action:

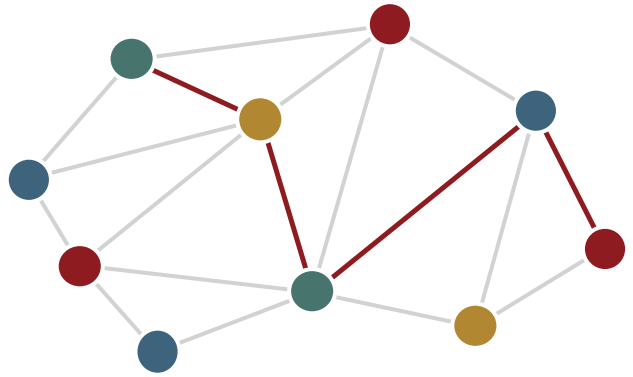
  lower-degree nodes have fewer neighbors to consult

Padding was never a fact about signals — it was a patch for pretending a line has no ends.

No padding, ever: the graph already says who exists.

PART III

Permutation symmetry is the whole point.



01 Watch the dense alternative fails

02 Commutation: filters move with the shift

03 Relabeling permutes; readouts forget

04 Diagonalize: the convolution theorem

Constraints are features: symmetry is why convolution wins.

The dense alternative memorizes names

Why not just learn a full linear map $y = Wx$?

Try it on a road network: our 7 nodes are congestion sensors, one noisy reading each; predict each sensor's next-hour value.

- W needs $n^2 = 49$ parameters for 7 sensors — the two-tap filter needs 2.
- The vendor re-exports the CSV with rows in a new order: $\boldsymbol{W}$ multiplies the wrong entries and every sensor's prediction scrambles.
- The city wires up a new district: $\boldsymbol{W}$ has the wrong shape entirely — nothing transfers.

A model that depends on node identities learns the labeling, not the structure.

The failures of $\boldsymbol{W}$ are the specification for $h(\boldsymbol{S})$.

Filters commute with the shift

$$h(\boldsymbol{S})\boldsymbol{S} = \boldsymbol{S}h(\boldsymbol{S})$$

Shift then filter = filter then shift — the output moves with the input.

One-line proof: powers of $\boldsymbol{S}$ commute with $\boldsymbol{S}$.

CHECK ON THE CYCLE

$$\boldsymbol{x} = (1,2,2,5), \qquad \boldsymbol{T}\boldsymbol{x} = (5,1,2,2)$$

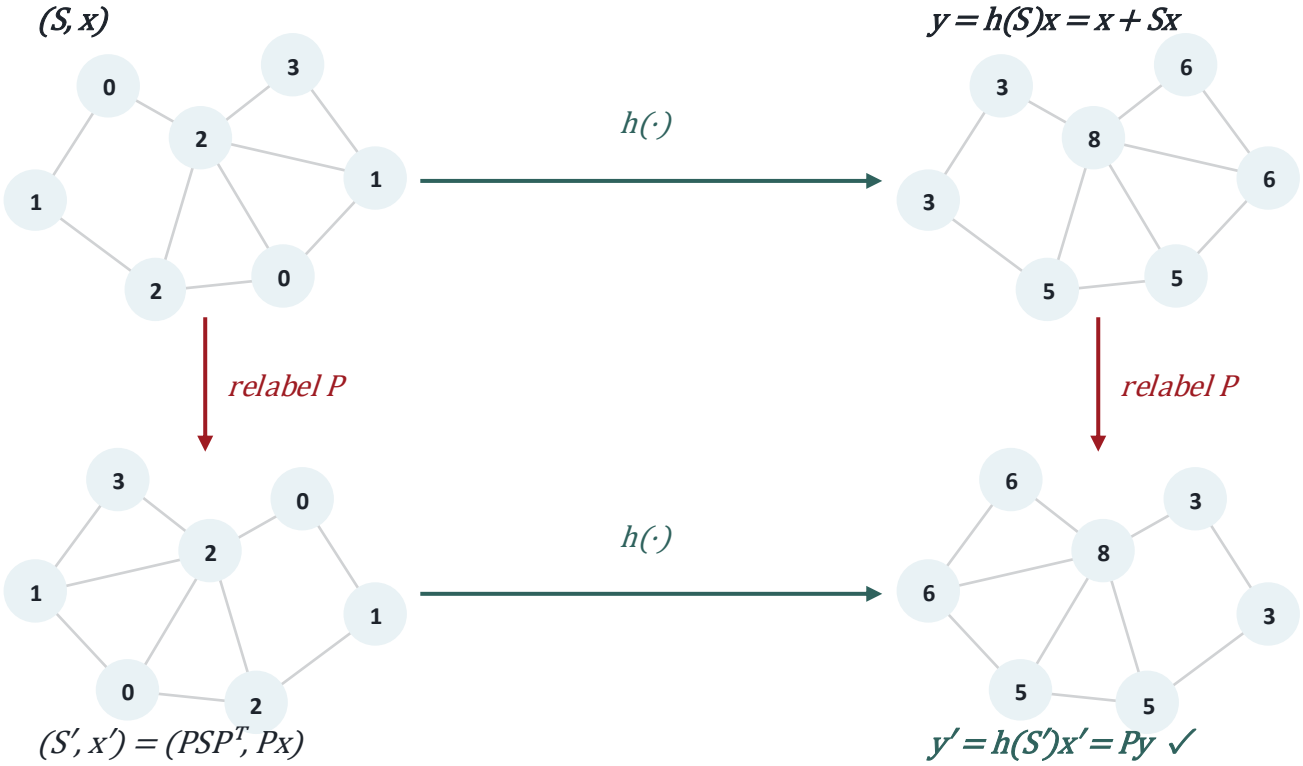
$$\boldsymbol{T}[(\boldsymbol{I} + \boldsymbol{T})\boldsymbol{x}] = (7,6,3,4) = (\boldsymbol{I} + \boldsymbol{T})[\boldsymbol{T}\boldsymbol{x}]$$

Exactly the LTI theorem from signals class – ***Shift invariance for graphs***

The graph analogue of time-invariance.

Relabeling cannot change the computation

$$S' = PSP^\top, x' = Px \quad \Rightarrow \quad h(S')x' = Ph(S)x$$



Permutation equivariance, by construction.

THE WHOLE PROOF

$$(PSP^\top)^k = PS^kP^\top \qquad (P^\top P = I)$$

The taps never mention a node name — outputs travel with their nodes.

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

See it: reverse the labels

- The output vector reverses exactly as the input did: (3, 5, 9, 7) → (7, 9, 5, 3).
- A symmetric readout forgets the order: the sum is 24 on both sides.

This is exactly the CSV re-export: *same sensors, new row order — every prediction travels with its sensor.*

Node outputs permute; graph-level readouts do not move at all.

Equivariance moves; invariance forgets

EQUIVARIANT · NODE LEVEL

Relabel the input and the nodewise output follows the same relabeling.

This is what a convolution layer gives you:

$$\boldsymbol{h}(\boldsymbol{S}')\boldsymbol{x}' = h\left(\boldsymbol{P}\boldsymbol{S}\boldsymbol{P}^{\boldsymbol{T}}\right)\boldsymbol{P}\boldsymbol{x} = \boldsymbol{P}\boldsymbol{h}(\boldsymbol{S})\boldsymbol{x}$$

Traffic: per-sensor congestion estimates.

INVARIANT · GRAPH LEVEL

A symmetric readout — sum, mean, max — discards node order and returns one unchanged answer.

Invariance = equivariance + symmetric readout.

$$f(\boldsymbol{h}(\boldsymbol{S}')\boldsymbol{x}') = f\left(h\left(\boldsymbol{P}\boldsymbol{S}\boldsymbol{P}^{\boldsymbol{T}}\right)\boldsymbol{P}\boldsymbol{x}\right) = f(\boldsymbol{h}(\boldsymbol{S})\boldsymbol{x})$$

Traffic: the city-wide congestion index.

Node tasks want equivariance; graph tasks want invariance.

What the graph version gives up

- On the grid, the kernel tells left from right: h_{-1} and h_{+1} are different numbers.
- A graph neighborhood is a set. One tap per hop distance, not one per neighbor.
- Graph filters are isotropic: neighbors at the same hop are treated identically; finer resolution costs more hops.

Buying back direction-awareness — attention is all you need

In traffic terms this bites: a sensor's upstream and downstream neighbors are genuinely different — and the filter provably cannot tell them apart.

- Simple polynomial gains locality, a small parameter count, and clean symmetry.
- May be too rigid for problems where different edge types should be treated differently

The price of arbitrary topology: per-hop weights, not per-neighbor weights.

Richer expressiveness does not require giving up permutation equivariance

The convolution theorem survives

Diagonalize $\boldsymbol{S} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}}$ (real and symmetric) and every power diagonalizes with it:

$$\boldsymbol{y} = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \boldsymbol{x} = \sum_{k=0}^{K-1} h_k \boldsymbol{V}\boldsymbol{\Lambda}^k\boldsymbol{V}^T\boldsymbol{x} \qquad \boldsymbol{V}^T\boldsymbol{y} = \boldsymbol{V}^T \sum_{k=0}^{K-1} h_k \boldsymbol{V}\boldsymbol{\Lambda}^k\boldsymbol{V}^T\boldsymbol{x} \qquad \hat{\boldsymbol{y}} = \sum_{k=0}^{K-1} h_k \boldsymbol{\Lambda}^k \hat{\boldsymbol{x}}$$

- Convolution in the vertex domain = pointwise multiplication in the graph frequency domain — word for word the DSP theorem.

$$\hat{y}_i = \sum_{k=0}^{K-1} h_k \lambda_i^k \hat{x}_i$$

- "One knob per frequency" -- a polynomial sets the knobs to $h(\lambda_k)$ (frequency response of graph filter) while touching only edges — K local rounds, no eigendecomposition.

$$\hat{h}(\lambda) = \sum_{k=0}^{K-1} h_k \lambda^k$$

Case: denoise the sensors with two taps

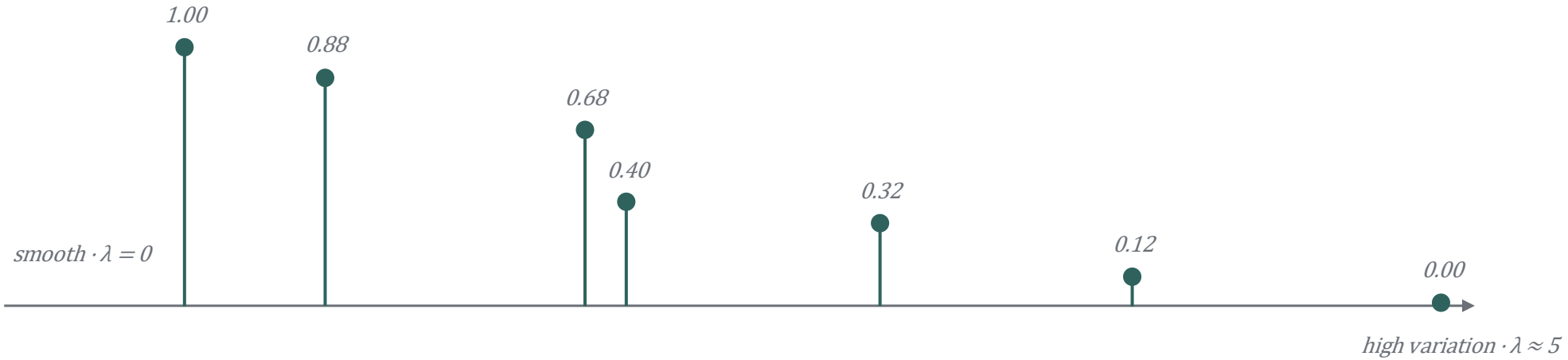
- One pass of the two-tap filter pulls each sensor toward its neighbors' average — the red errors shrink to the small residuals on the right.
- Why it works: the true field varies smoothly across the network, while noise is jagged — it flips from neighbor to neighbor. Averaging keeps the smooth part and cancels the jagged part.

The two low-degree sensors improve least — fewer neighbors to consult.

A low-pass graph filter is consensus with your neighbors — and it already denoises.

Why those two taps denoise, in frequency

The case's filter was $y = x - \varepsilon Lx$, i.e. $h(\lambda) = 1 - \varepsilon\lambda$.
Evaluate it at the network's frequencies ($\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$: smooth = 0, $\lambda_{max} \approx 5$):

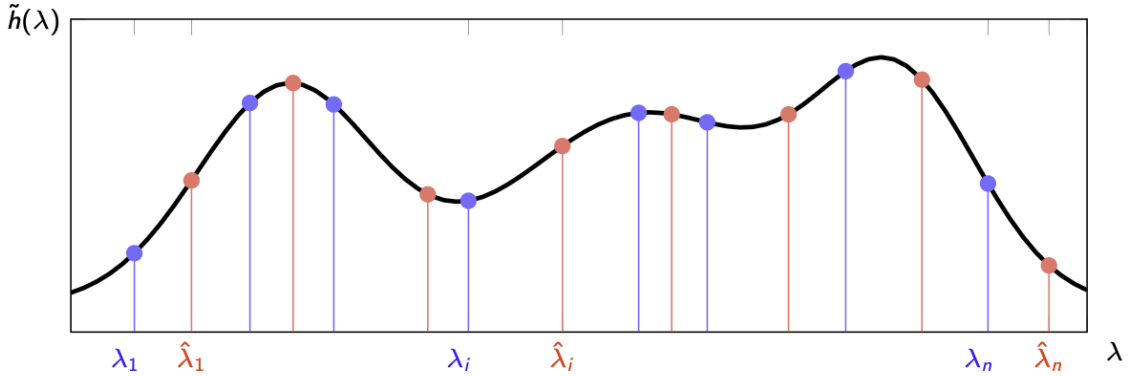


- Gain ≈ 1 at the smooth end: the signal lives here and passes through untouched.
- Gain → 0 at the high variation end: high-frequency noise lands here and is filtered away.

A low-pass frequency response keeps the smooth signal and removes high-frequency noise.

Frequency response is independent of the graph

$$h(\lambda) = h_0 + h_1\lambda + \cdots + h_{K-1}\lambda^{K-1}$$



- **Same rule, other districts, no retraining:** $h(\lambda) = 1 - \varepsilon\lambda$cuts expected noise by factor ≈ 0.60 on 7 sensors, 0.60 on 12, 0.61 on 20 — essentially identical everywhere.

Fine print: measure "smooth" with the same Laplacian L = D − A on every graph — a mismatched shift redefines smoothness and can undo the gain.

This graph-free h(λ) is why filters transfer: train the curve once — each graph merely samples it.

What carries over from grid to graph

	STANDARD CONVOLUTION	GRAPH CONVOLUTION
Domain	regular line or grid	arbitrary graph
Shift	$\boldsymbol{T}$ · one-step translation	S · one-hop aggregation
Filter	polynomial in $\boldsymbol{T}$	polynomial in $\boldsymbol{S}$
Locality	finite window	finite hop neighborhood
Shift symmetry	commutes with translation	commutes with $\boldsymbol{S}$
Label symmetry	index translation	permutation equivariance
Conv. theorem	pointwise in DFT basis	pointwise in GFT basis

Same algebraic recipe; topology replaces coordinates.

Checkpoint: predict the property

01 **A filter has $K = 4$ taps. How far can information travel?**

Think: the largest power is the third.

at most 3 hops

02 **The input is shifted by $\boldsymbol{S}$. What happens to the nodewise output?**

Think: polynomials commute with $\boldsymbol{S}$.

the output shifts by $\boldsymbol{S}$

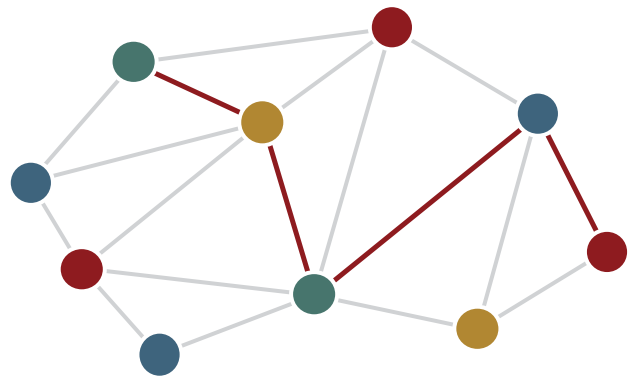
03 **After relabeling, should each output coordinate stay fixed?**

Think: equivariant first, invariant only after a readout.

no — coordinates permute; only a symmetric readout is invariant

TAKEAWAYS

Local operations. Global proofs.



01 Sparse shifts make graph filters local

02 Polynomials commute with the graph shift

03 Joint relabeling permutes node outputs

04 Symmetric readouts remove the labels

Next: stability, filters + nonlinearities, stacked — GNNs.