Neural Graph Processing

KIOS Summer School 2023

Prof. Cesare Alippi

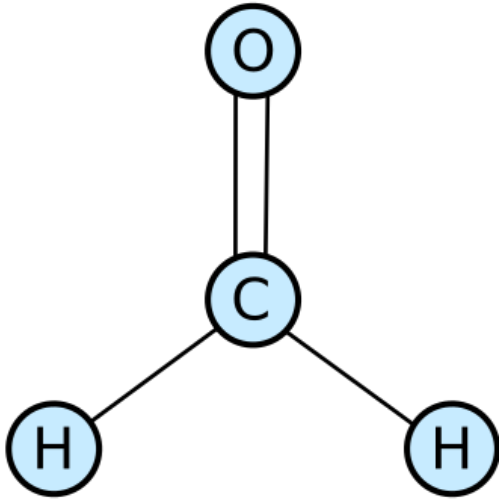
Università della Svizzera italiana

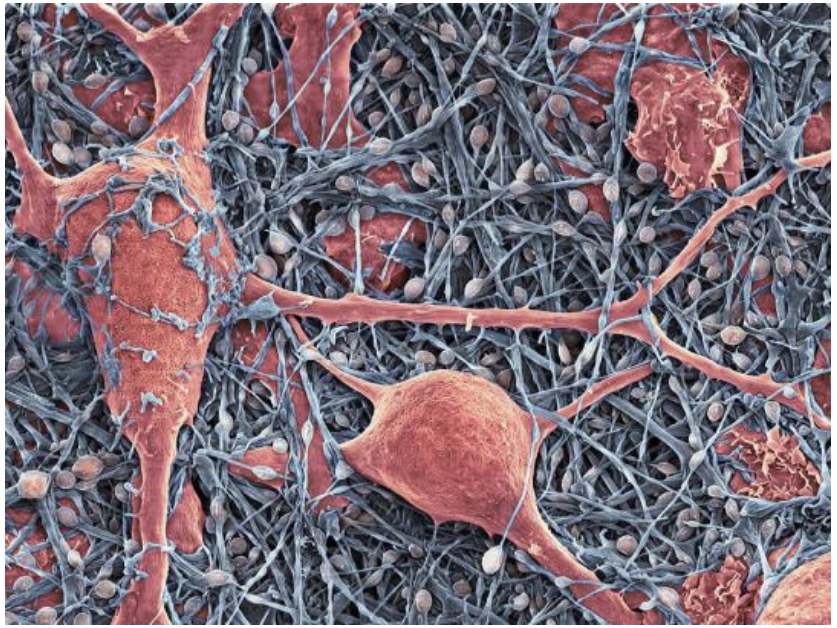
Outline

- **We are going to address three major topics**
  - **Graph representation: why and how**
  - **Processing operators**
  - **Spatiotemporal graphs**

Why graphs

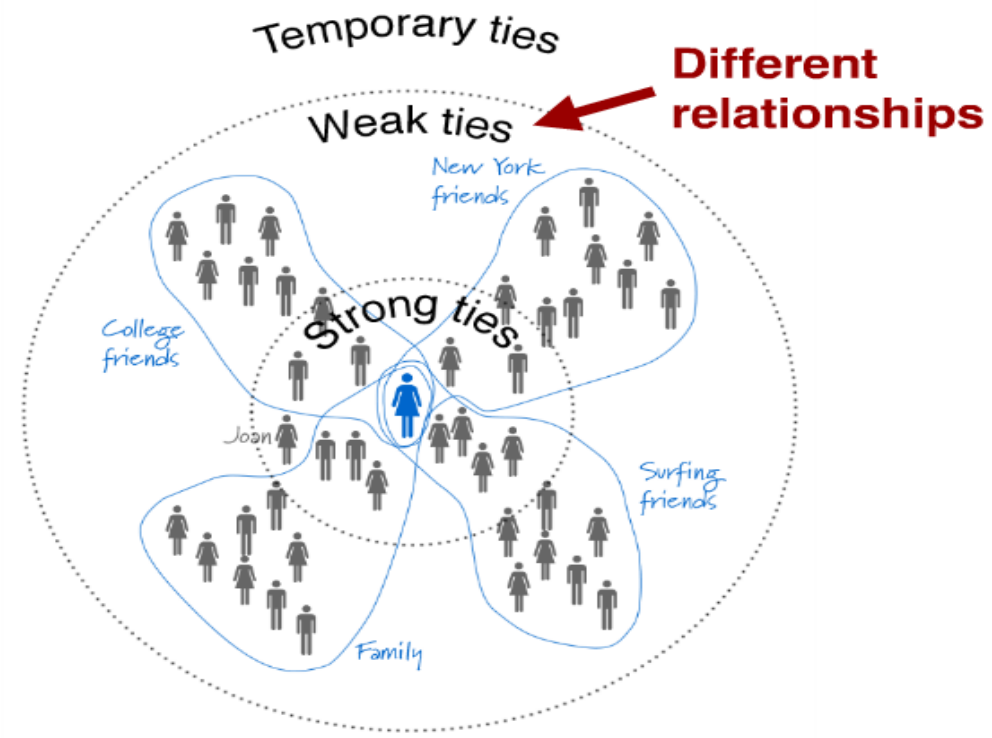
In many applications graphs come naturally





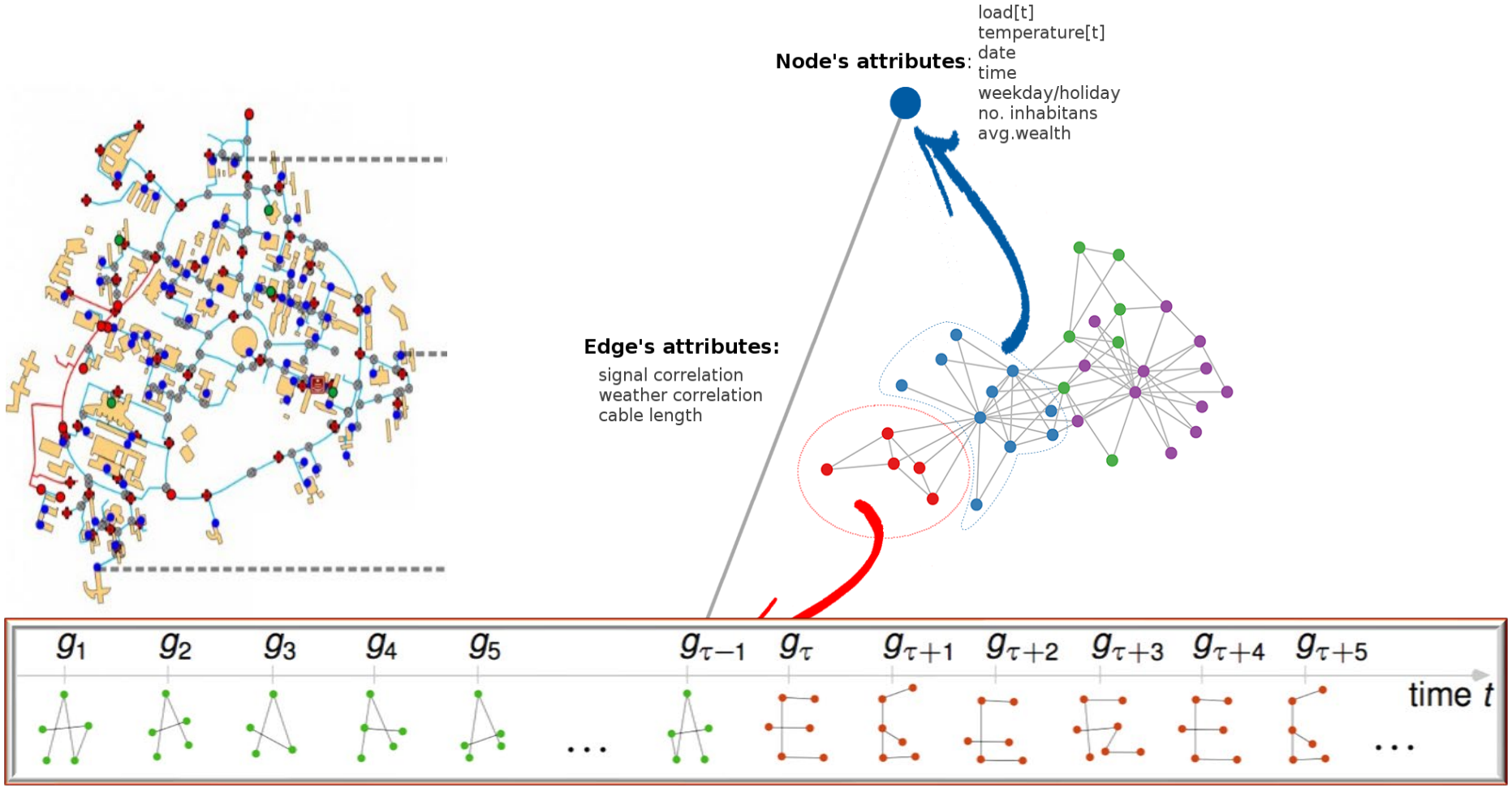
Why graphs

In some cases are explicit. In others are latent

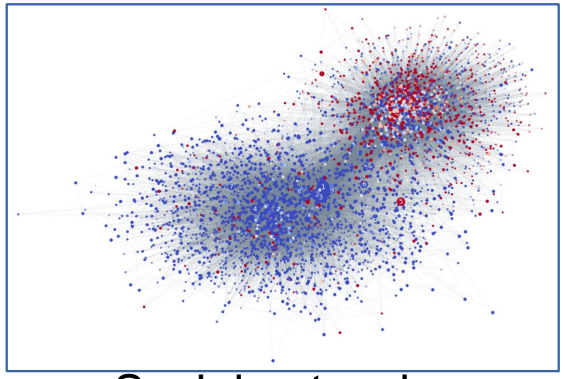


Data streams and graph streams

In others, we derive graphs from timeseries (signals)



A plethora of applications

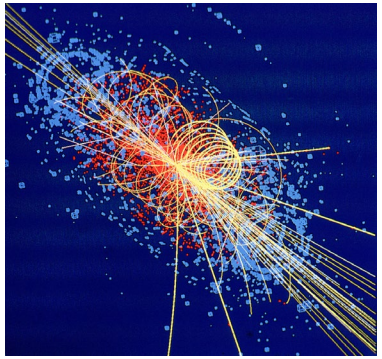


Social networks

Monti et al. "Fake news detection on social media using geometric deep learning."

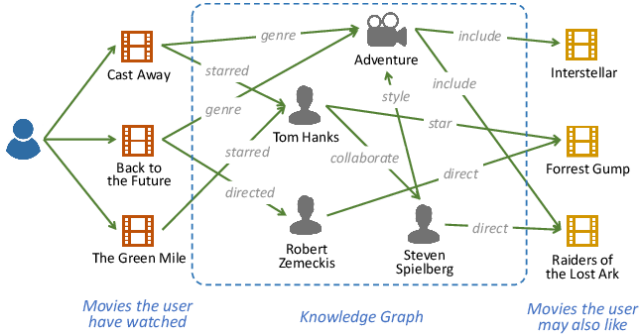
Traffic prediction

"Traffic prediction with advanced Graph Neural Networks" https://deepmind.com/blog/article/traffic-prediction-with-advanced-graph-neural-networks



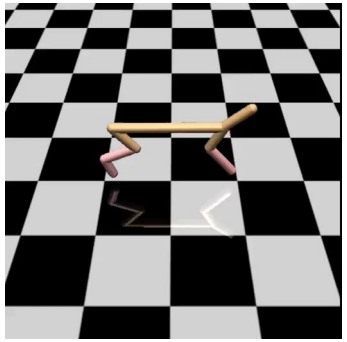
Physics

Shlomi, Jonathan, and Peter Battaglia. "Graph neural networks in particle physics."



Recommender systems

Ying et al. "Graph convolutional neural networks for web-scale recommender systems."

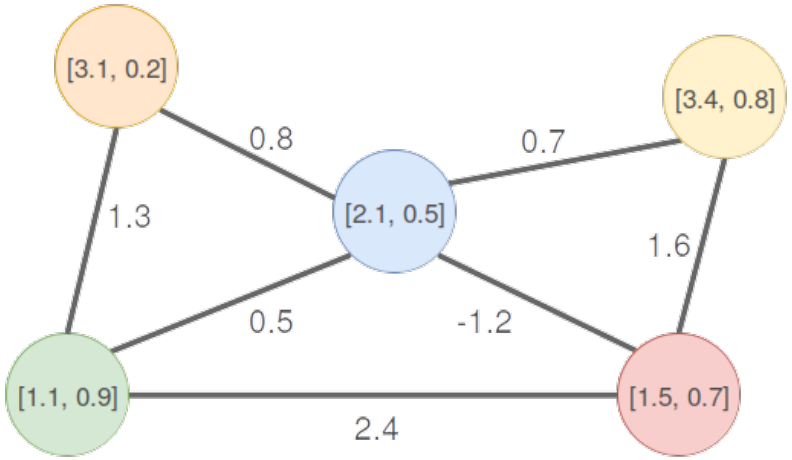


Reinforcement learning

Zambaldi et al. "Relational deep reinforcement learning."

Do we really need to represent and process graphs in a different way?

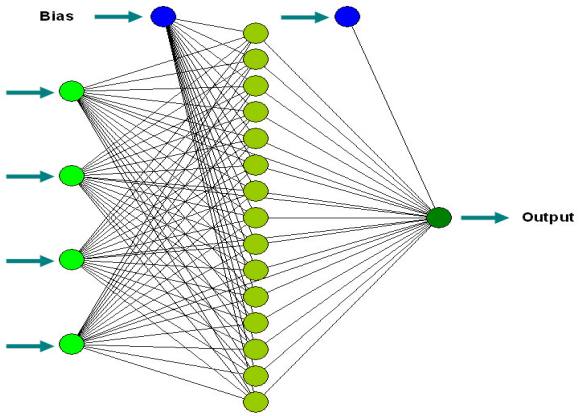
We might argue that having node features is enough provided that you have a strong inference engine (say) able to extract functional interdependencies.



e.g., Pearson's correlation coefficient

X =

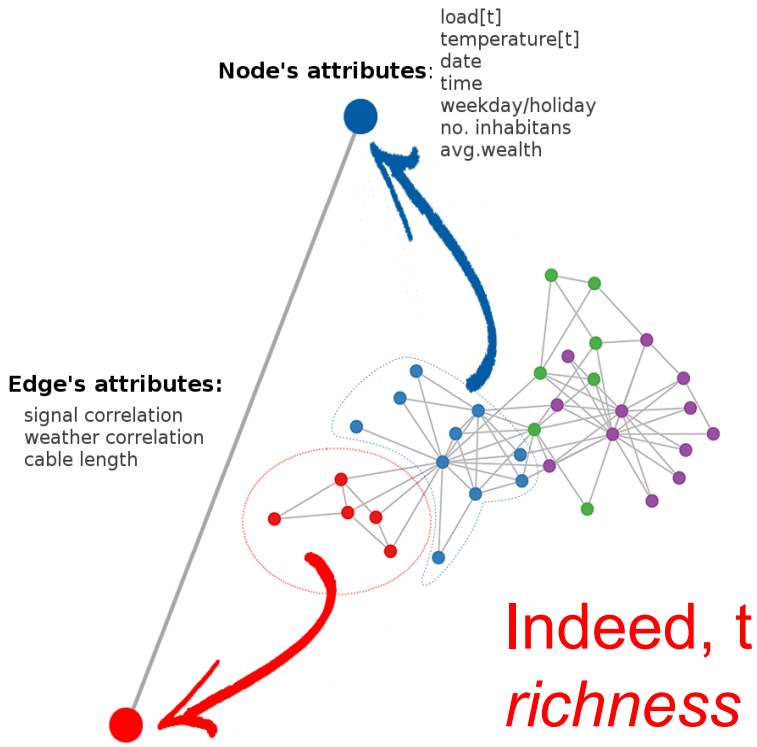
3.1	0.2
3.4	0.8
2.1	0.5
1.1	0.9
1.5	0.7



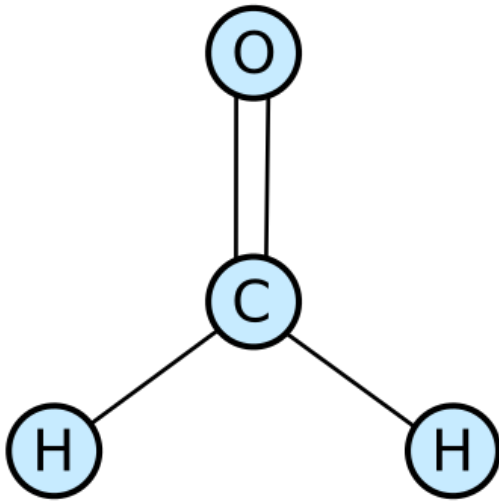
Do we really need to represent and process graphs in a different way?

That is true *in principle* in many cases and *partly* – whereas not – in others

in principle



not true



Indeed, this is related the *information richness of node features*

Some philosophical issues

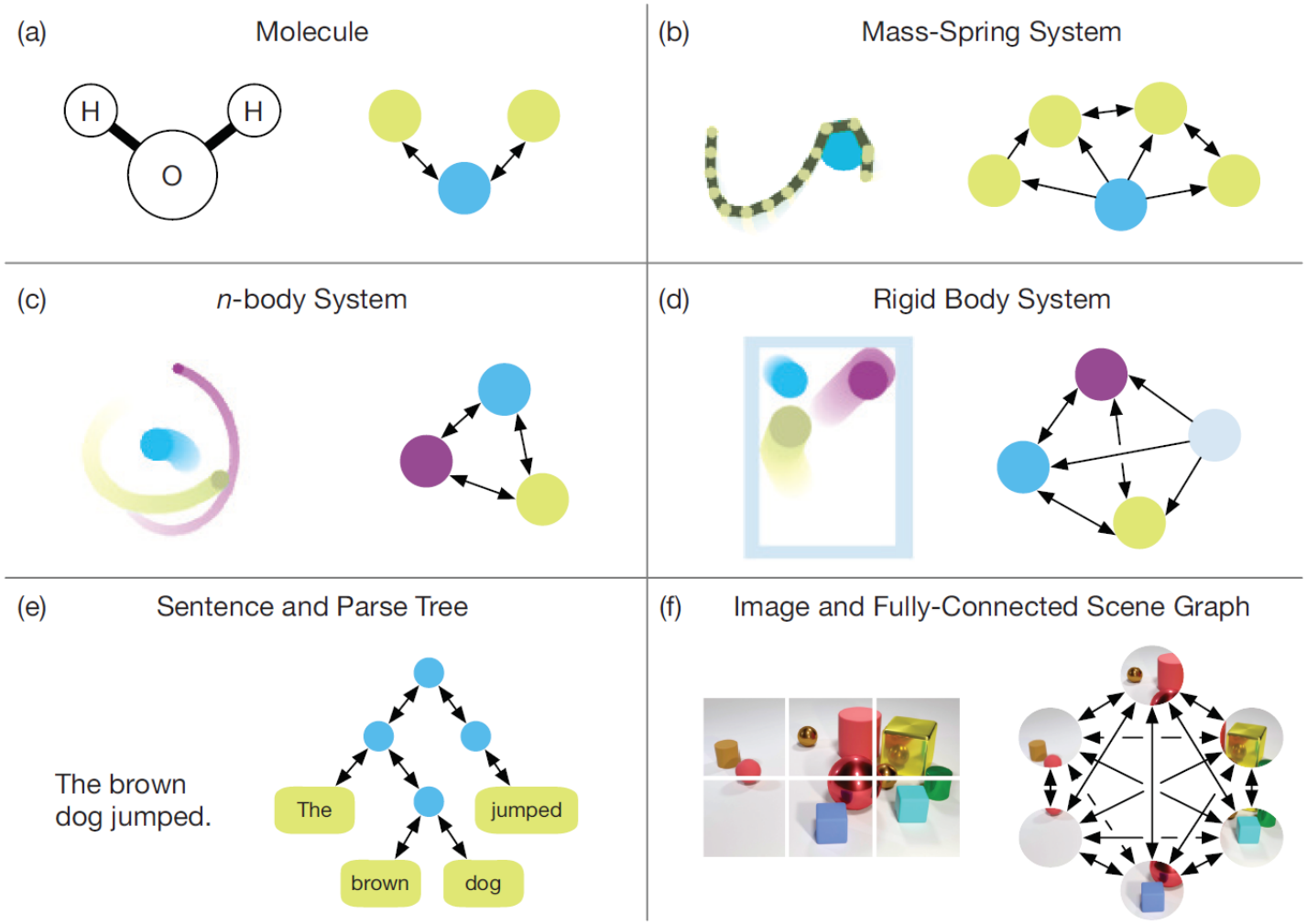
- Previous comments are related to the way we design features
  - End-to-end learned (e.g., DL)
  - Hand-engineered

- A school believes that we should place ourselves in-between by taking advantage of both
  - Rich representations
  - Inductive bias

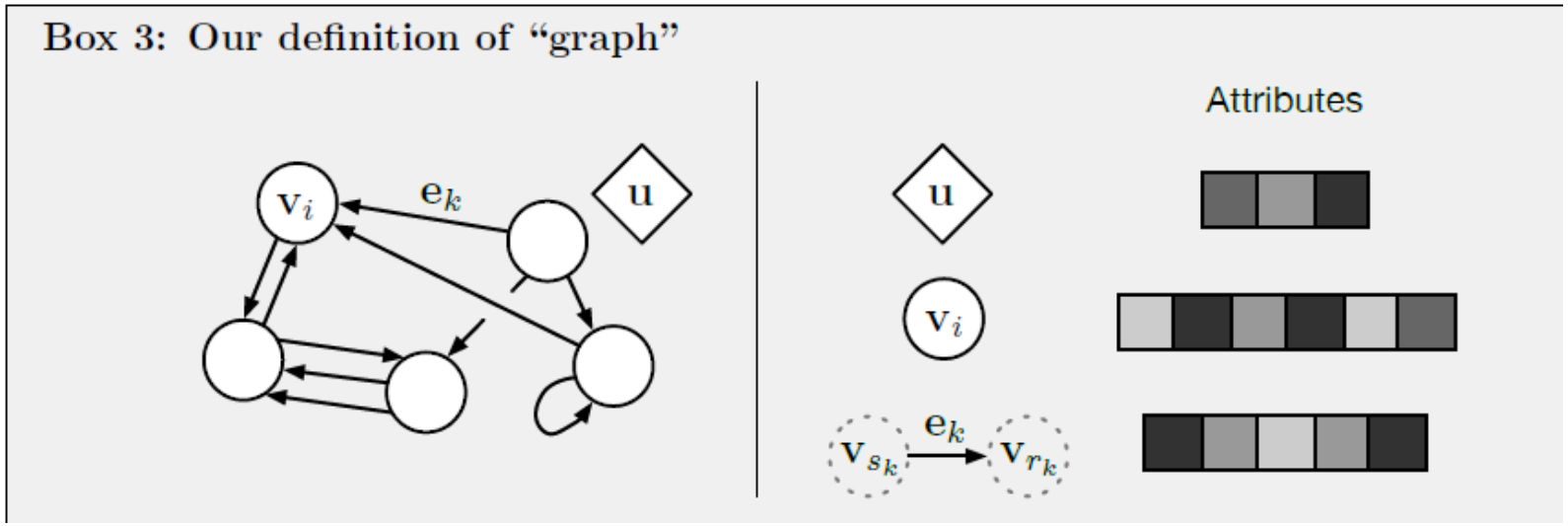
Inductive bias

- *An inductive bias is an artifact that allows a learning algorithm to prioritize one solution over another*, say efficiently driving learning towards particular regions of the search space
- Prior information can be encoded in the architecture of the solution itself (e.g., we believe that our model is linear).
- Inductive bias improves the search for solutions, in general without diminishing performance.

Graphs and graph representations

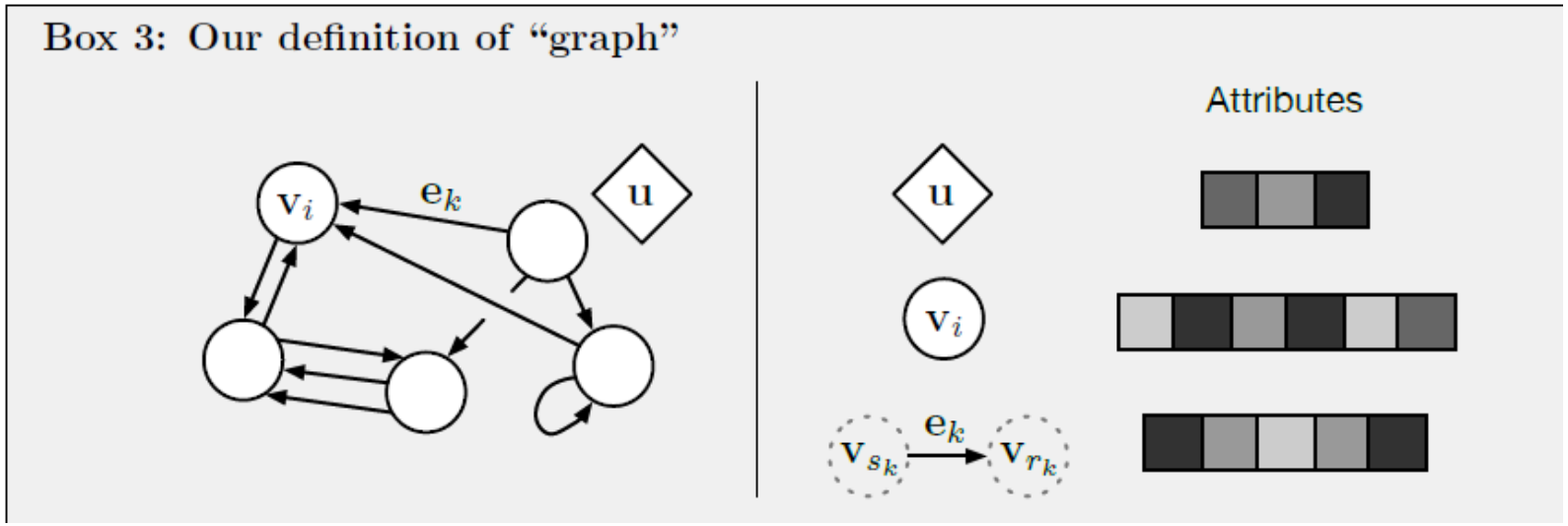


Graphs and graphs…



- Directed graph: one-way edges, from a sender node to a receiver node;
- Undirected graph: bidirectional edges;
- Multi-graph: there can be more than one edge between vertices (including self-edges).

Graphs and graphs…



- Attributes: properties that can be encoded e.g., vector, tensor, set, another graph, a model.
- Attributed graphs: edges and vertices have attributes associated with them.
- Global attribute: an attribute at the graph-level.

Graph processing

- In node-focused tasks features of nodes are our output, e.g., to reason about physical systems
- In edge-focused task edges represent the output we are interested in, e.g., to make decisions about interactions among entities
- In graph-focused tasks the entire network attributes constitute the output, e.g., to predict the potential energy of a physical system, the properties of a molecule, or answers to questions about a visual scene

How to represent a graph?

Adj

0	0	1	1	0
0	0	1	0	1
1	1	0	1	1
1	0	1	0	1
0	1	1	1	0

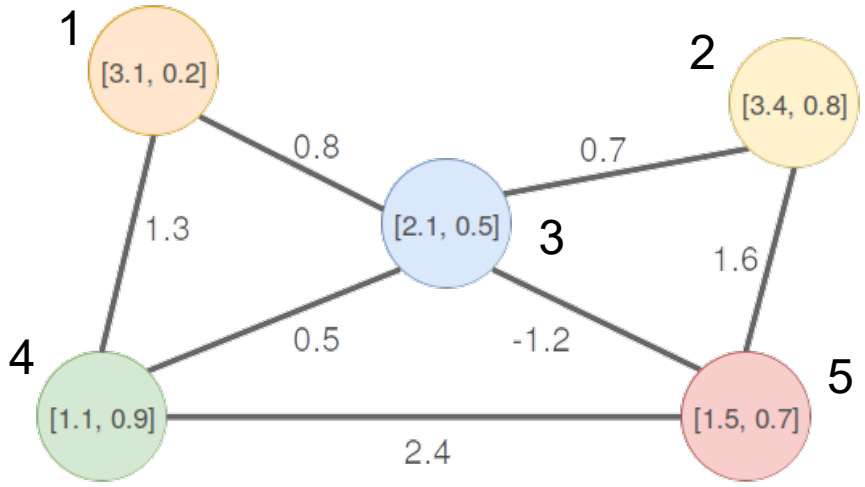
X

3.1	0.2
3.4	0.8
2.1	0.5
1.1	0.9
1.5	0.7

E

0.	0.	0.8	1.3	0.
0.	0.	0.7	0.	1.6
0.8	0.7	0.	0.5	-1.2
1.3	0.	0.5	0.	2.4
0.	1.6	-1.2	2.4	0.

- **Adj**: binary adjacency matrix
- **X**: node features
- **E**: edge features



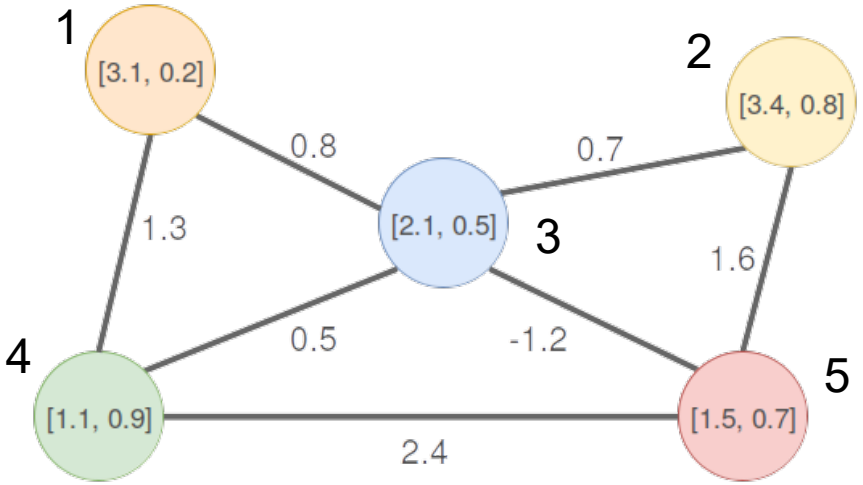
How to represent a graph?

X

3.1	0.2
3.4	0.8
2.1	0.5
1.1	0.9
1.5	0.7

Adj *in* E

0.	0.	0.8	1.3	0.
0.	0.	0.7	0.	1.6
0.8	0.7	0.	0.5	-1.2
1.3	0.	0.5	0.	2.4
0.	1.6	-1.2	2.4	0.



- **X**: node features
- **E**: edge features

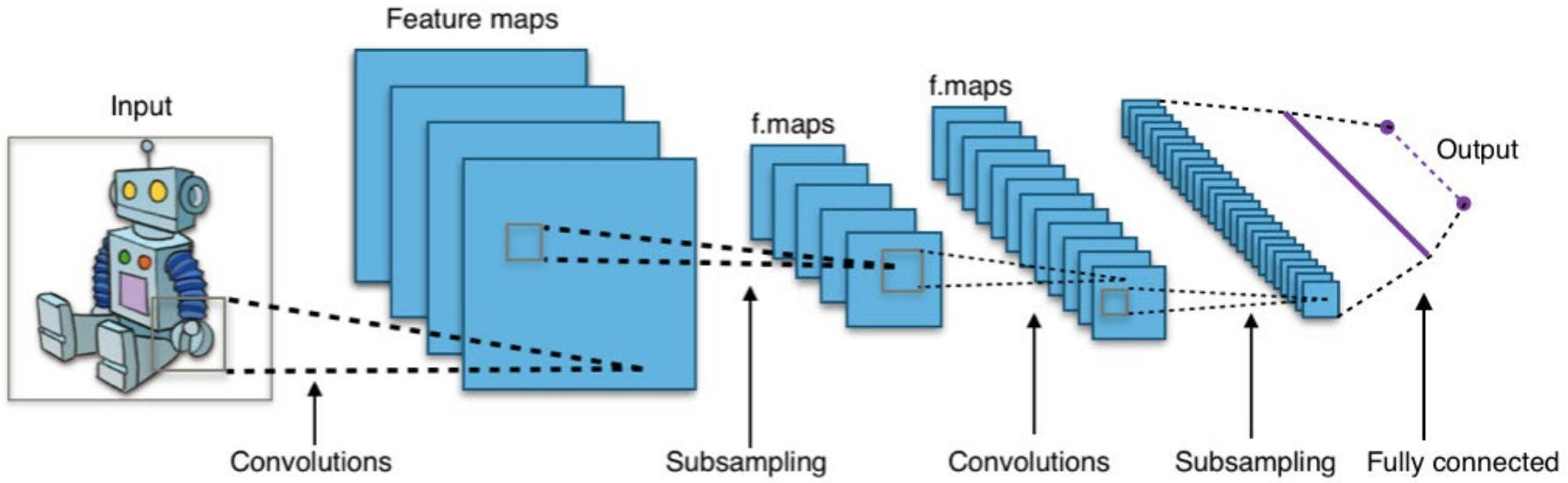
Yes, finally, we have matrices (or tensors)

Processing blocks

“a smooth transition to graph processing operators and architectures”

CNN: Deep Convolutional Networks

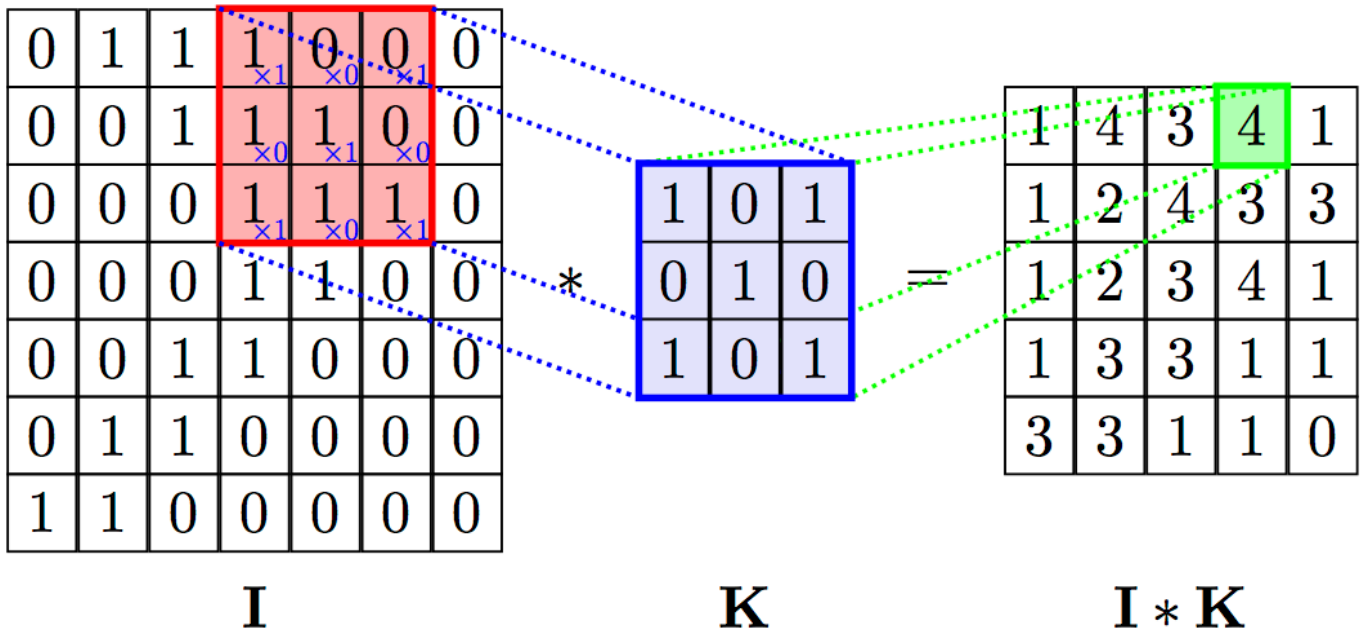
- A CNN takes advantage of the space locality principle (inductive bias)



- Subsequent steps of **convolutional** and **pooling** layers.
- Each layer computes a higher abstract representation w.r.t. the previous one; the image size shrinks at each step.

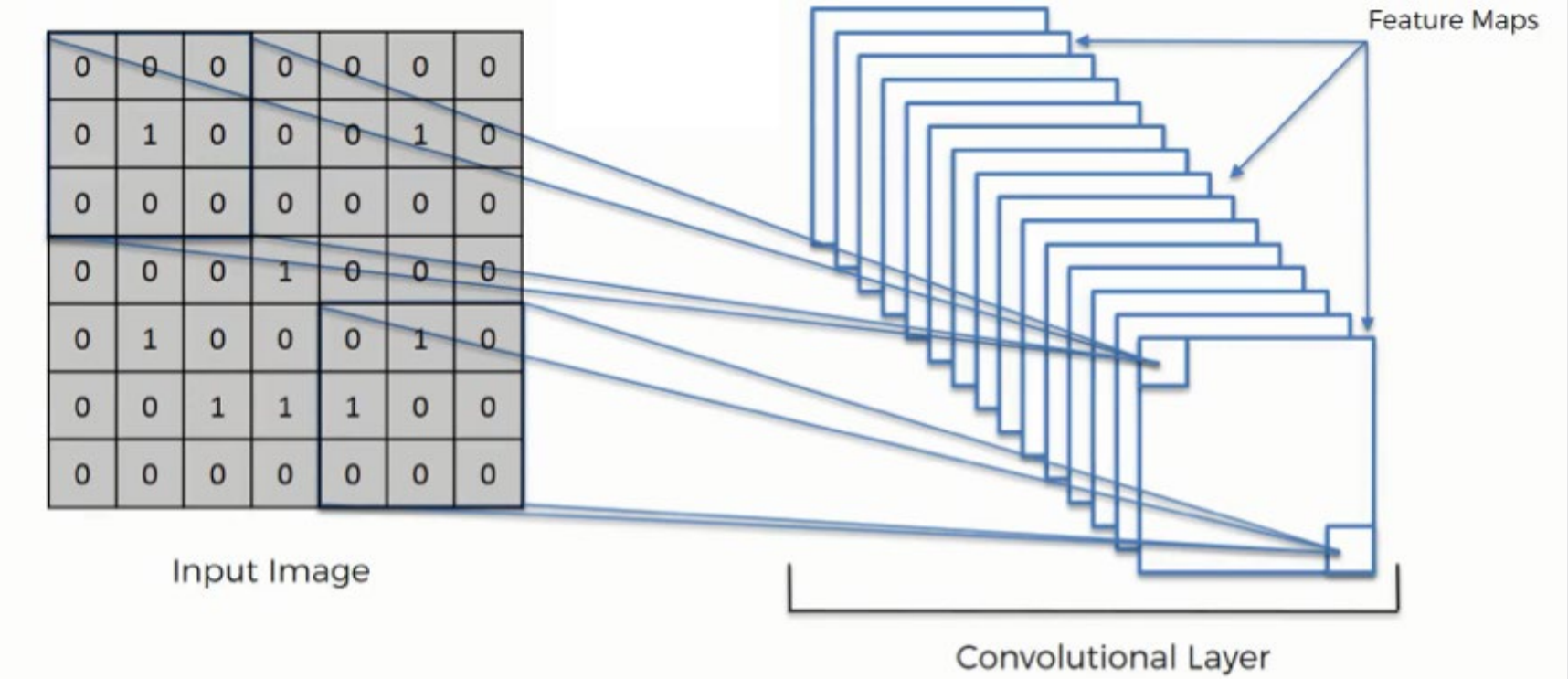
Convolution operator

- Convolutional layers: evaluate affinities based on the principle of locality.
- Receptive field applied to the image with a stride.
- The kernel/filter **K** contains learnable parameters.



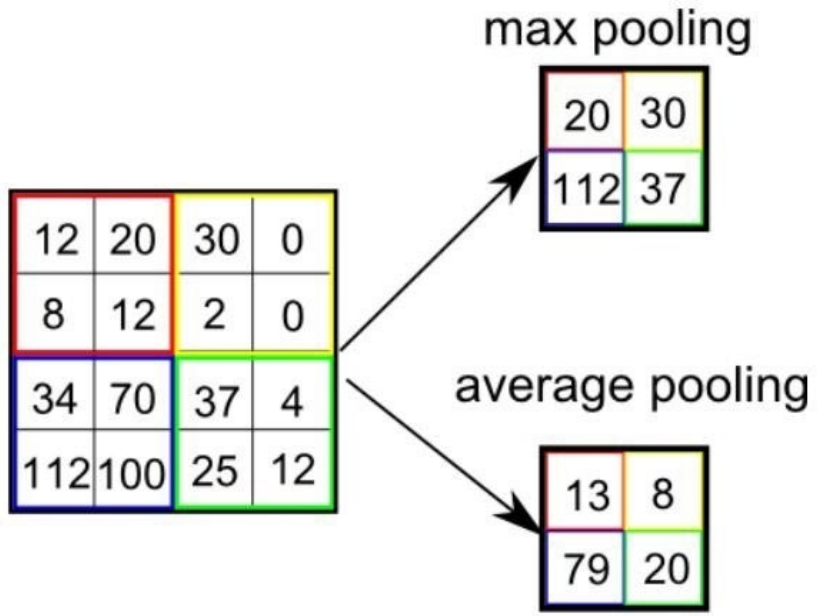
Convolution layer

- Many filters can be applied in parallel.
- As each one is learned, filters **K**s are different; after convolution, each one provides a different **feature map**.



Pooling operator

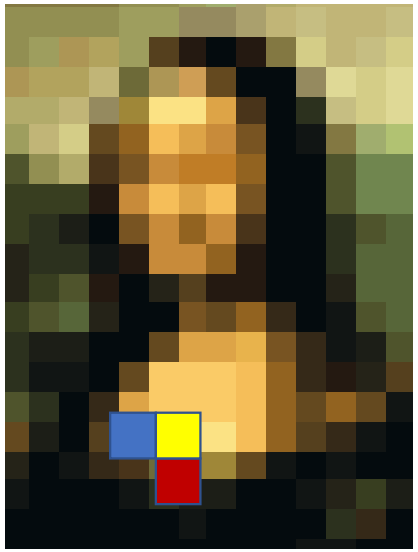
- Pooling layers reduce the image size based on some rules.

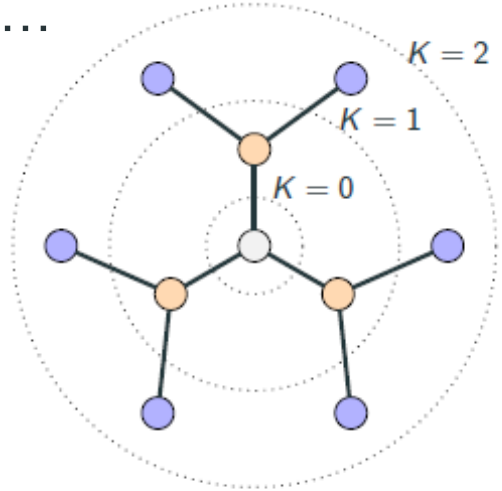


- Different pooling operators can be designed e.g., based on local properties.

Graph processing: Graph Neural Networks

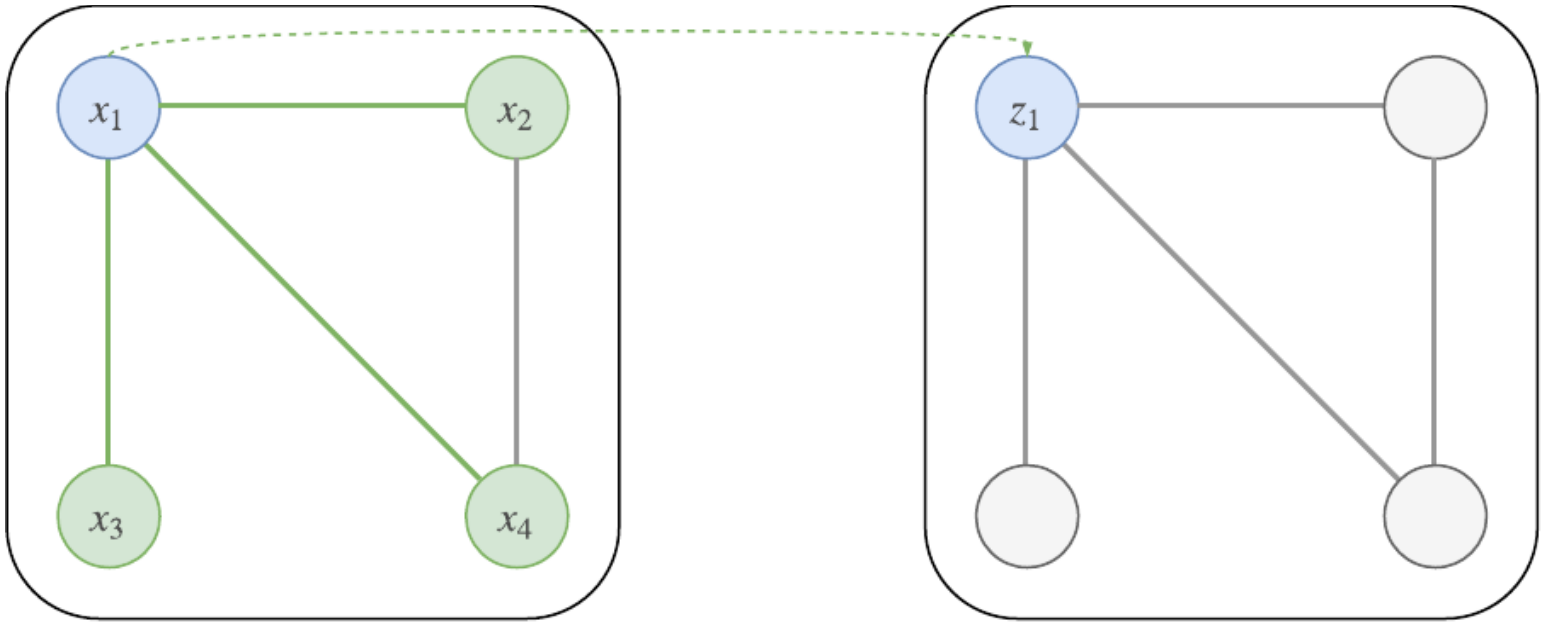
- “Mutatis mutandis” we can naively extend the CNN to a GNN (Graph Neural Network)
- In images, functional proximity mostly coincides with physical proximity
- In a graph functional proximity does not coincide with physical proximity; yet the locality principle is there…





GNN operators: Graph Convolution

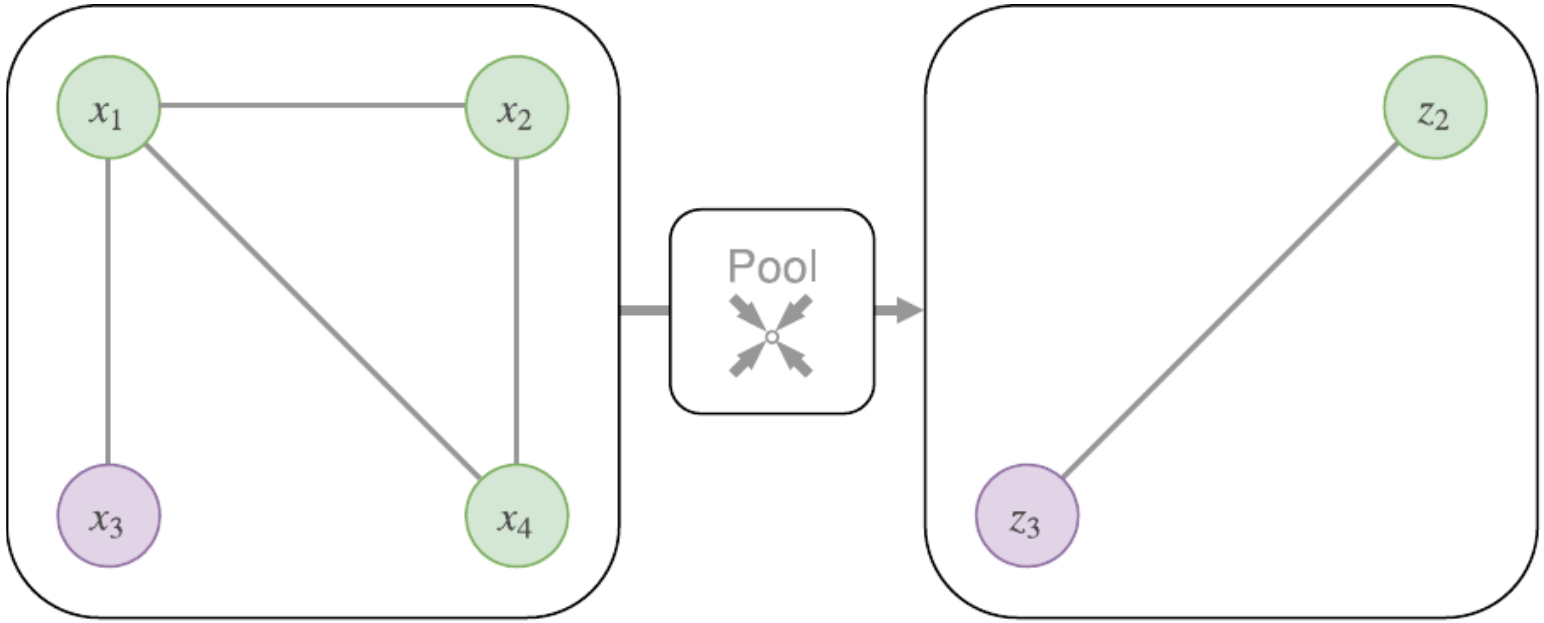
- Graph convolution exploits the local neighborhood of each node to compute a node value embedding



- The above convolution is at node level, but we might have information associated with edges too
- “Message passing” as an extension of convolution in next lecture topic

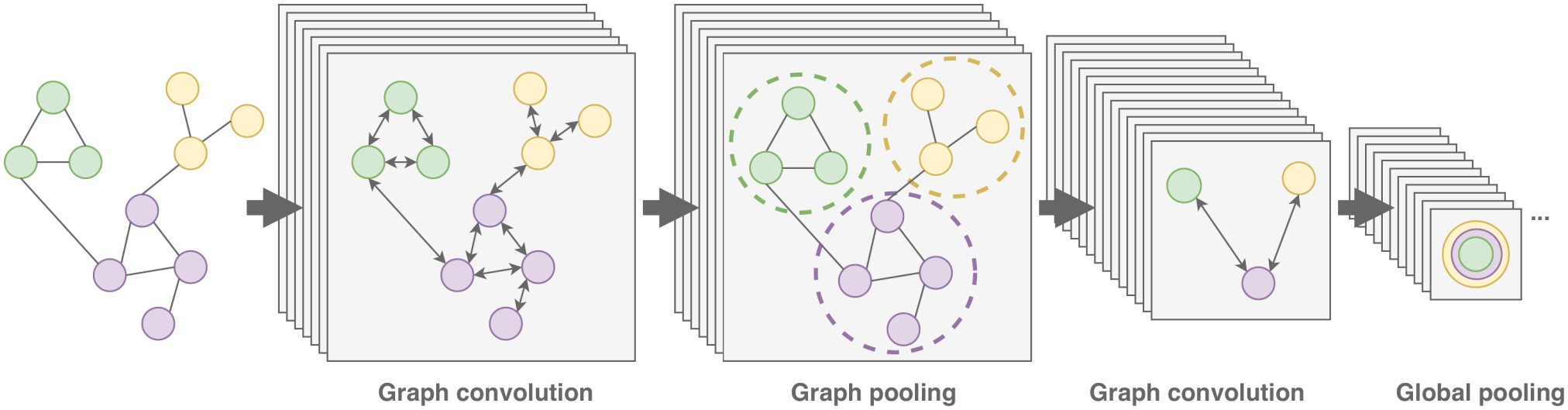
Graph Pooling

- Pooling aggregates nodes (shrinks the graph topology) to
  - obtain a more abstract representation of the graph
  - reduce the graph complexity (then, valuable to manage huge graphs)



GNN: Graph Neural Networks

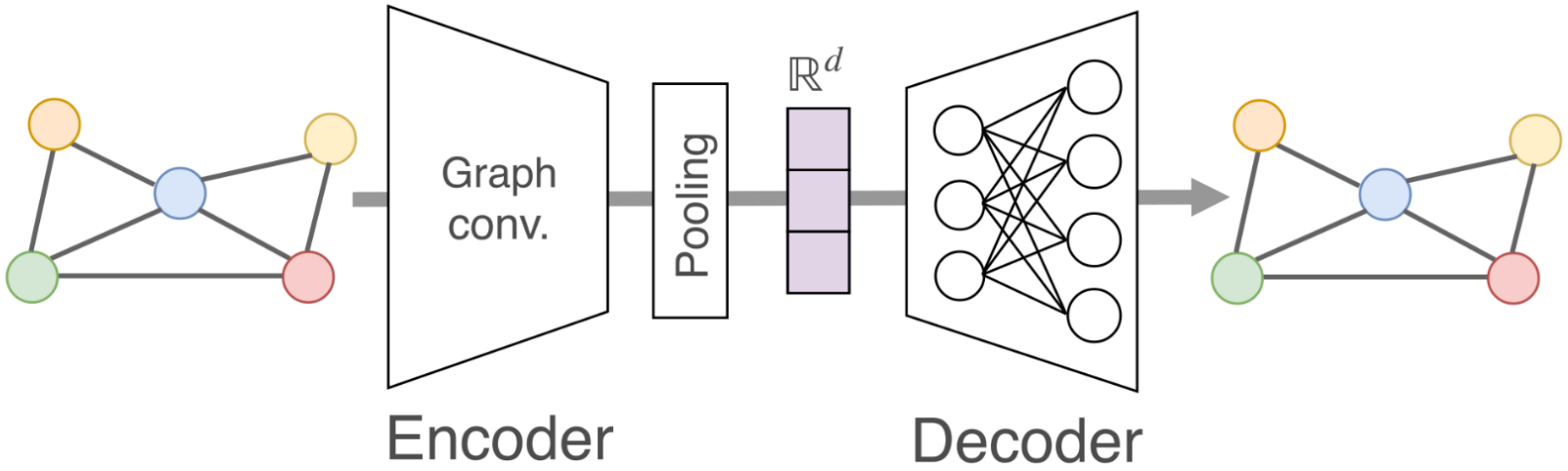
- We get a deep network – GNN – by interleaving operators



- Indeed, you can enjoy “conceptual transfer” of neural processing to other architectures…

Graph autoencoders

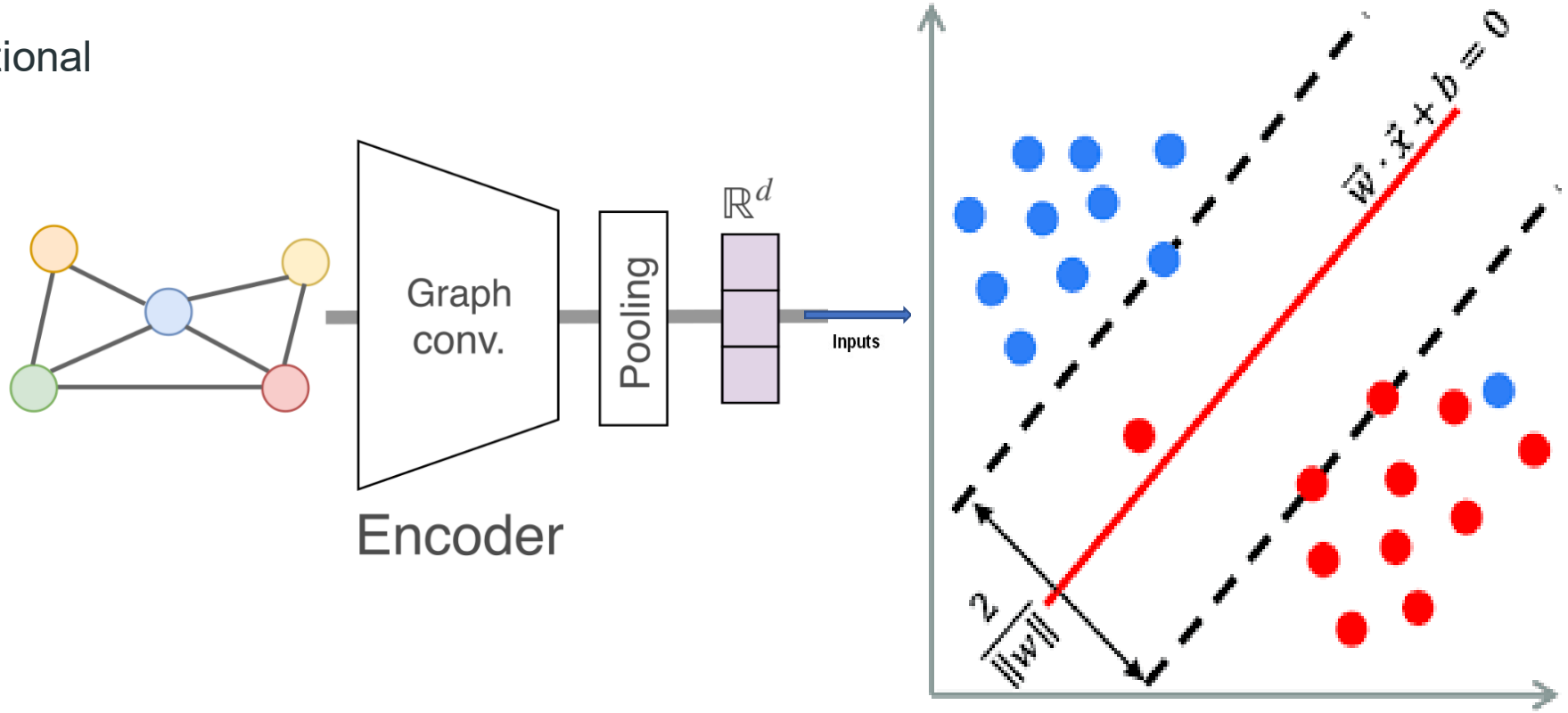
- The encoder is composed of graph convolutional layers with the pooling one
- A dense decoder reconstructs the matrices describing the graph



- The latent space represents a natural embedding

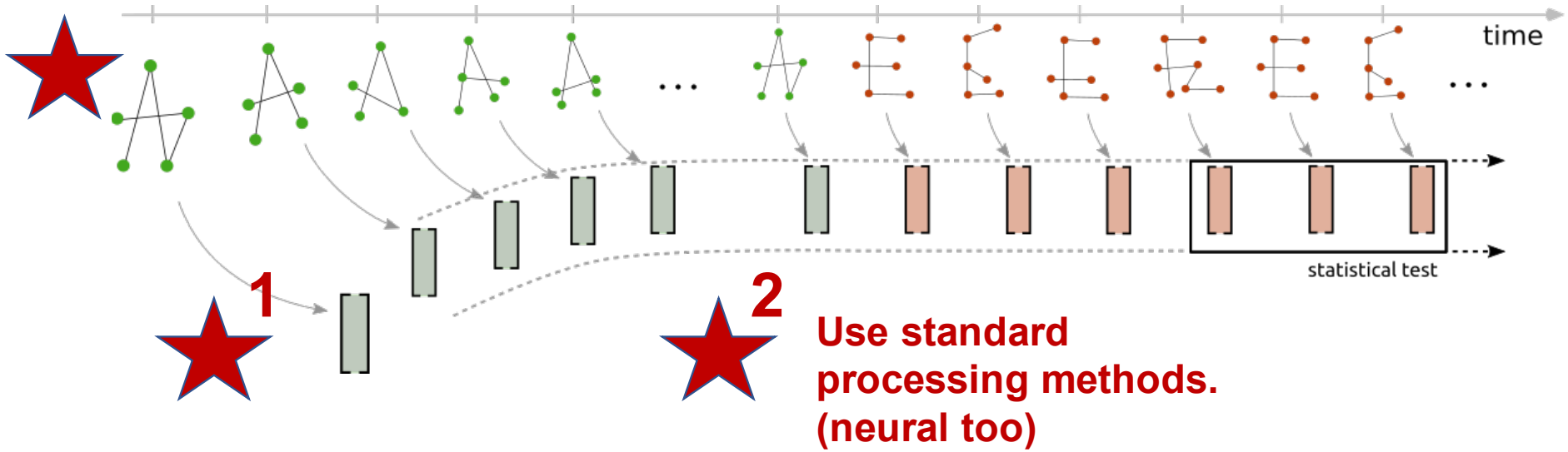
Latent space representation

- Once we have a graph to vector mapping (embedding) we can apply our favorite processing:
  - Neural
  - Traditional



The «vanilla» operational framework

- Map graphs to vectors (embedding)

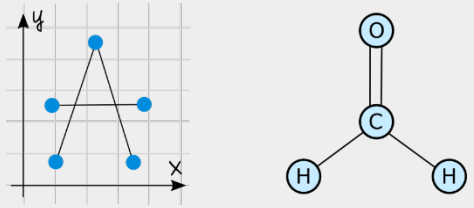


Graphs and embedding spaces

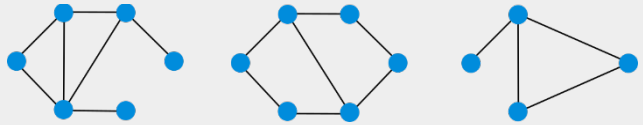
Which types of graphs are we interested in?

Attributed graphs represent a very large family of graphs

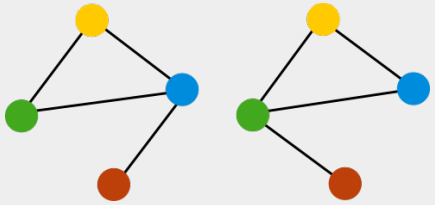
Generic attributes



Variable topology and order



Identified vertices



Non-identified vertices

The graph space $(\mathcal{G}[\mathcal{A}], \mathrm{d})$

- Set $\mathcal{G}[\mathcal{A}]$ of graphs $g = (V, E, a)$
  - V, E sets of vertices and edges (finite)
  - *A:* set of attributes
  - *a*: attribute function

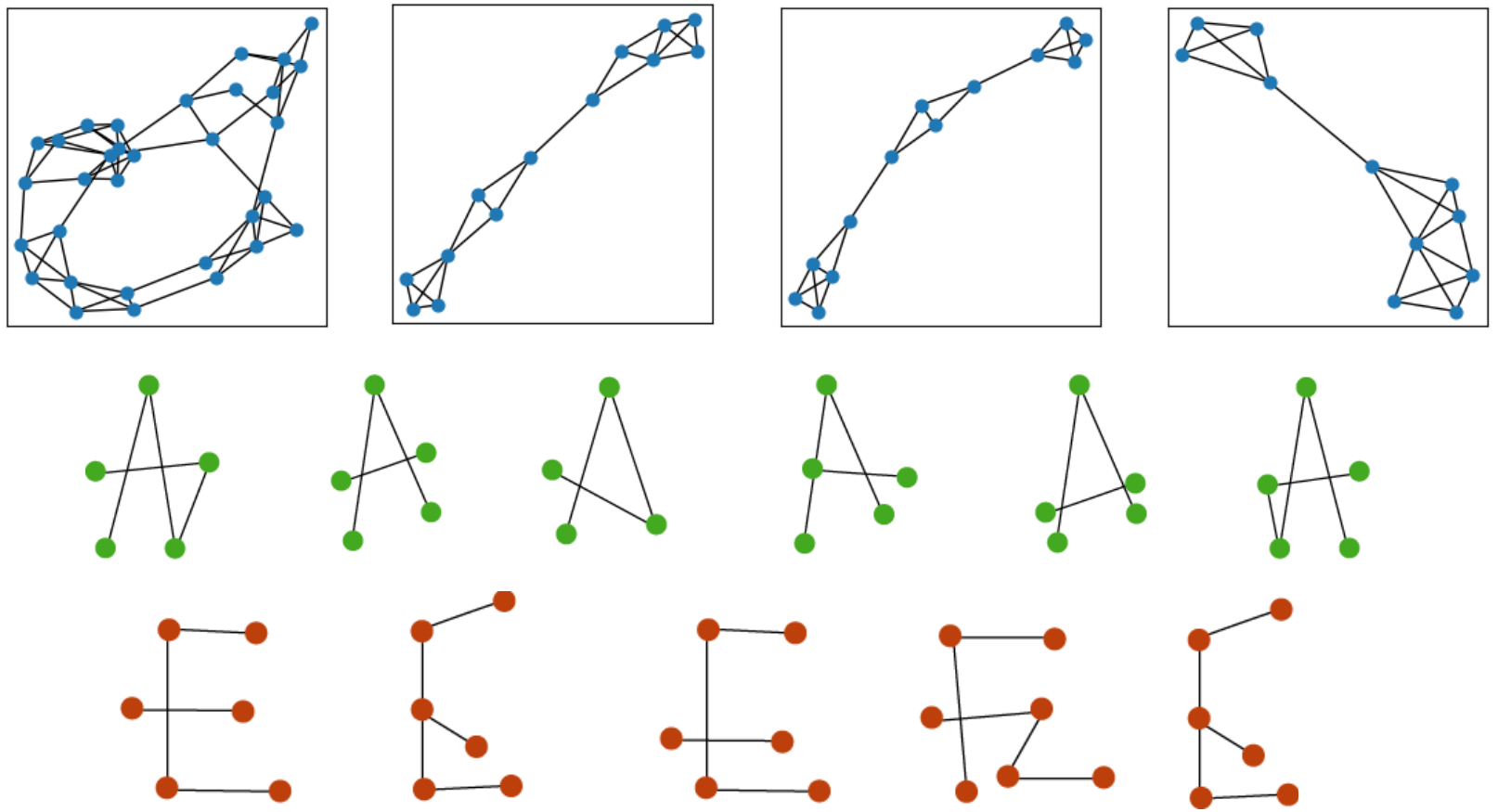
$$a : V \cup E \rightarrow \mathcal{A}$$

- Graph distance $\mathrm{d}(g_i, g_j)$
- On $(\mathcal{G}[\mathcal{A}], \mathrm{d})$ we can define a probability space

B.J. Jain, On the geometry of graph spaces, Discrete Applied Mathematics, December 2016.

Stationarity and graphs

- Mind, both topology and attributes can change under the stationarity hypothesis



Enzymes

Characters

Example of $\mathrm{d}(\cdot,\cdot)$: the graph (edit) distance

Sequence of edit operations that generates graph g_i starting from graph g_j

Operations:

- node insertion/deletion
- edge insertion/deletion

$$cost(o) = k(a,a)$$

- node modification
- edge modification

$$cost(o) = k(a,a) + k(a',a') - 2k(a,a')$$

$$\mathrm{d}(g_1, g_2) = \sqrt{\min_{(o_1,\ldots,o_m)} \sum_{i=1}^{m} cost(o_i)}$$

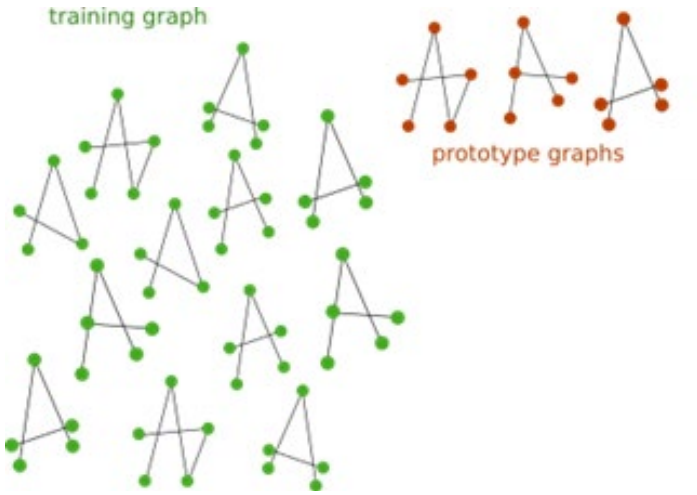
J. Jain, On the geometry of graph spaces, DAM, 2016.

Embedding methods (some)

- **Distance-based methods**:
  - Dissimilarity representation
  - Multi-dimensional scaling
- **Neural approaches**:
  - Autoencoders – already seen
    (latent space embedding)
  - Adversarial learning
    (inducing constraints on the latent space)

Dissimilarity representation

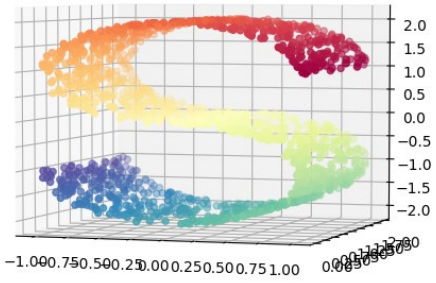
- Training phase:
  - Identify a set of prototypes R

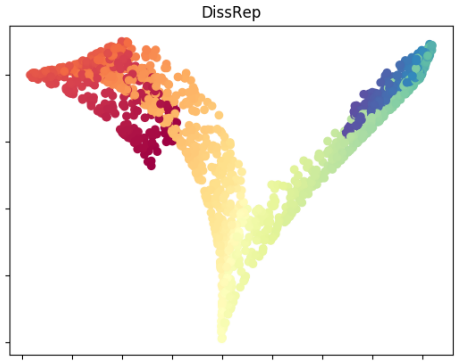


- Out-of-sample technique:

$$g_i \mapsto x_i = \begin{bmatrix} d(g_i, r_1) \\ \vdots \\ d(g_i, r_M) \end{bmatrix}$$

$$R = \{r_1, r_2, \ldots, r_M\}$$

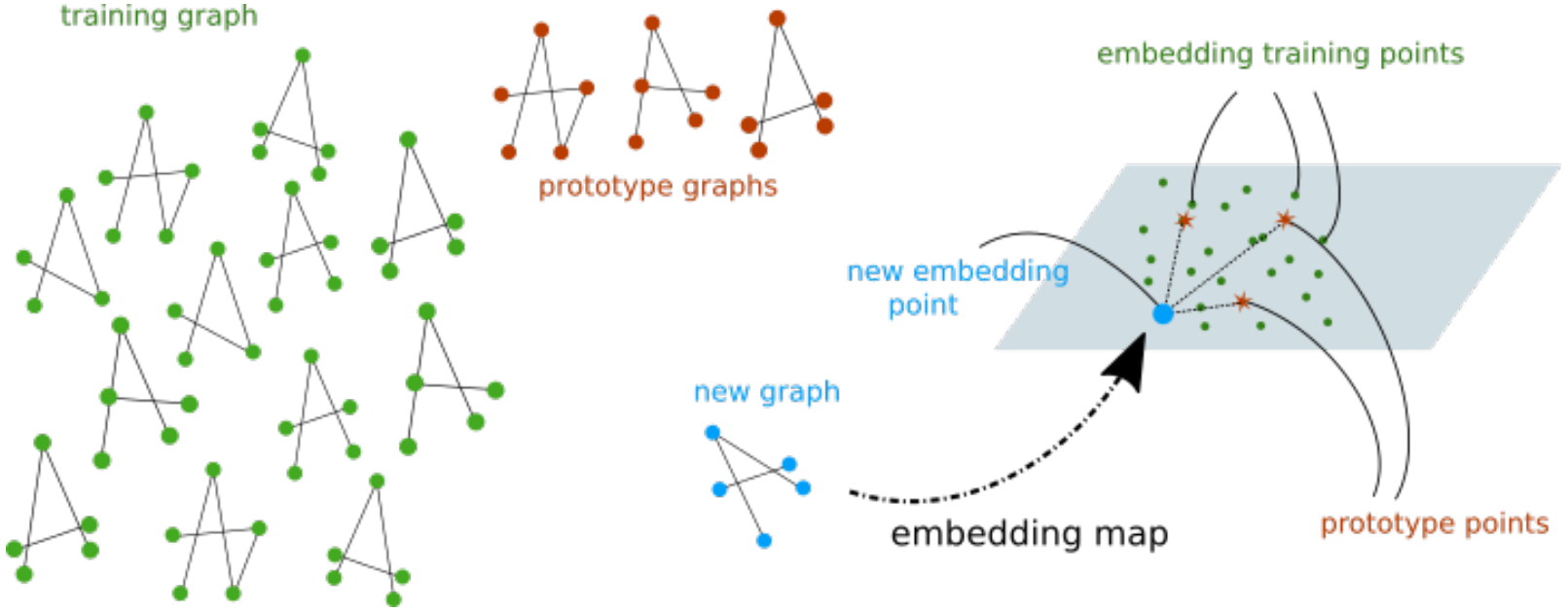




D. ZE. Pękalska and R. P. W. Duin, The Dissimilarity Representation for Pattern Recognition: Foundations and Applications. WS, 2005.

Multi-dimensional scaling

- Training phase:
  - Identify a set of prototypes
- Out-of-sample technique:
  - Use the dissimilarity representation by attempting to preserve the distance from prototypes



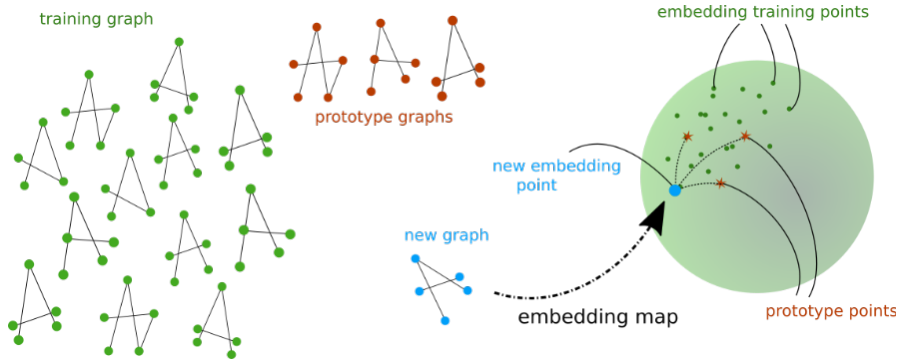
D. ZE. Pękalska and R. P. W. Duin, The Dissimilarity Representation for Pattern Recognition: Foundations and Applications. WS, 2005.

Embedding on constant curvature manifolds

Similar to MDS, but the optimization is on the manifold.

Training phase

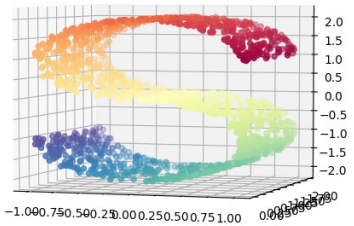
- Select a set of prototypes

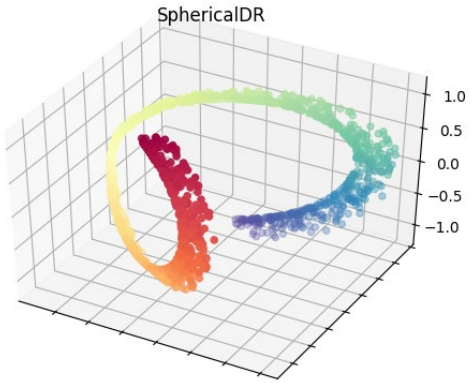


Out-of-sample technique

- Builds on the dissimilarity representation by attempting to preserve the distance from the prototypes

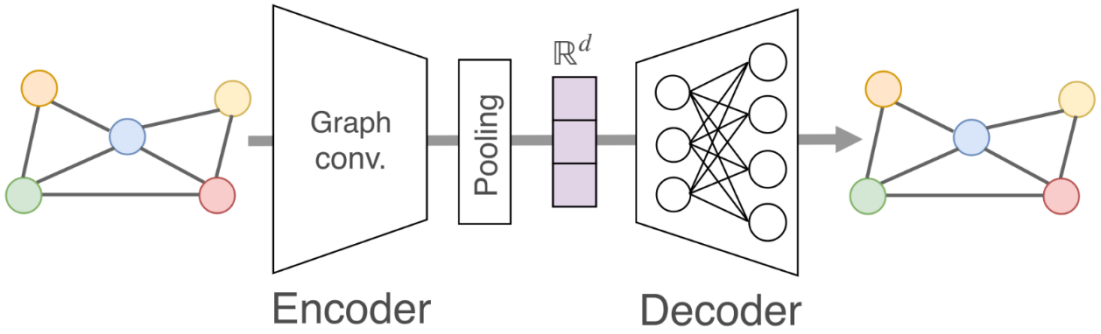
$$\rho(x, y) = \frac{1}{\sqrt{\kappa}} \arccos(\kappa \langle x, y \rangle_\kappa),$$



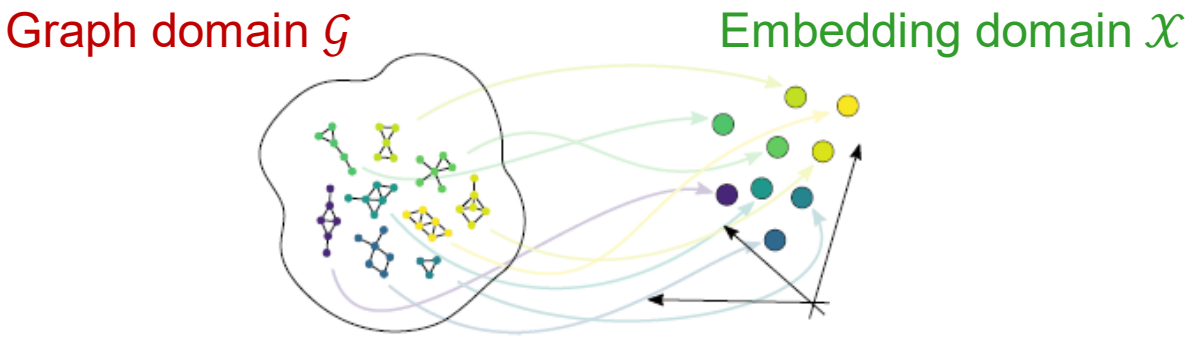


Graph autoencoders

- Autoencoders provide a nice way to automatically build the embedding

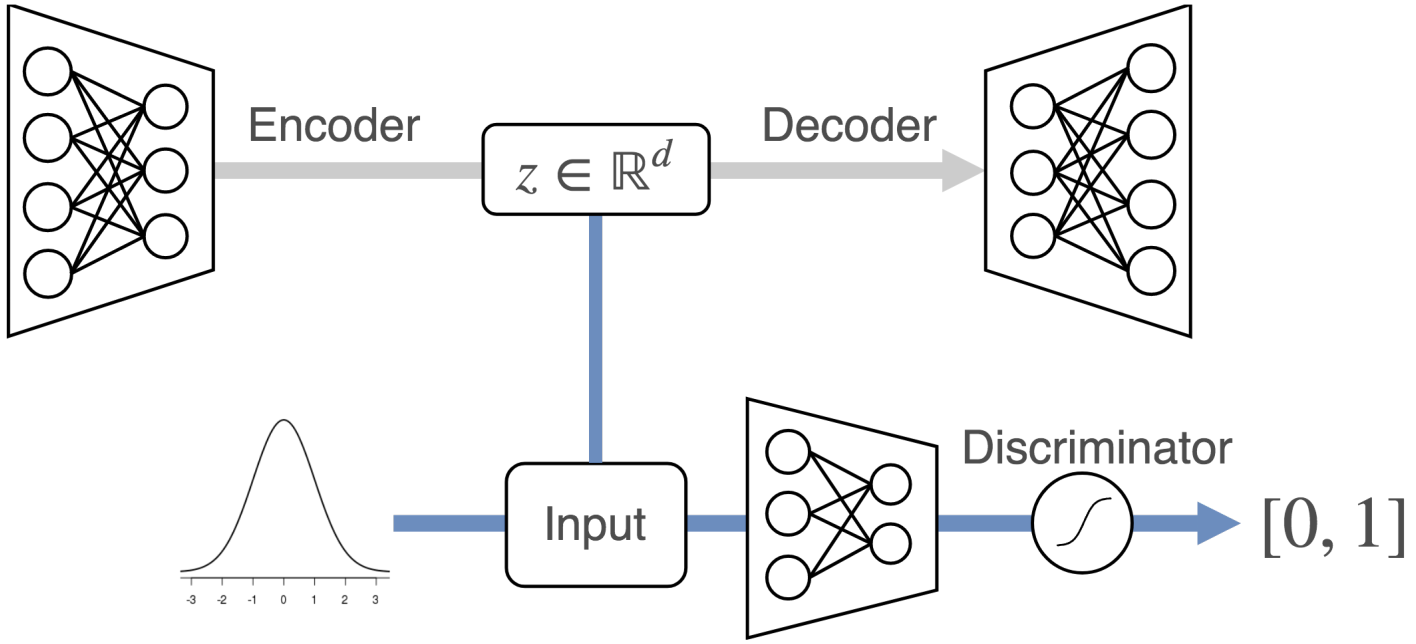


- But we can neither grant that the distance nor that the concept of distribution is preserved in the graph/embedding spaces



Adversarial autoencoders

- Match the distribution of embedded information in the latent space with an arbitrary (given) prior
- In this way we impose the concept of distance and a wished distribution in the embedded space



Processing Operators

Outline

Part 1. Neural Message Passing

- Towards graph convolutions
- Message passing

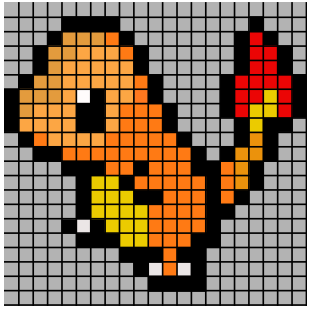
Part 2. Pooling on Graphs

- Select, Reduce, Connect
- Pooling methods
- Global pooling

Towards graph convolutions

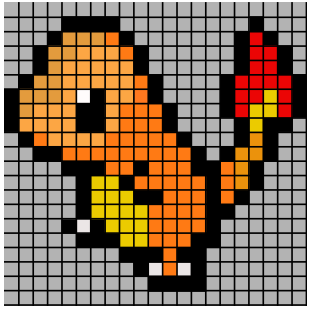
Convolution on images

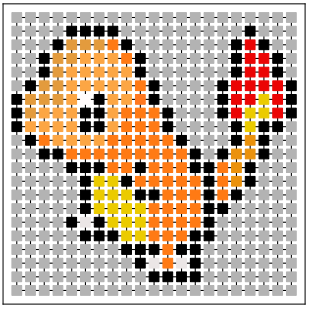
Consider the convolution operation in Convolutional Neural Networks (CNNs).



Convolutions on images

Consider the convolution operation in Convolutional Neural Networks (CNNs).





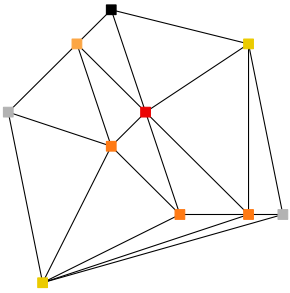
- The receptive field of a CNN reflects the **underlying grid structure**.
- The CNN exploits an **inductive bias** on how to process the individual pixels/timesteps/nodes.

Going beyond grids

- Not everything can be effectively cast into a grid...

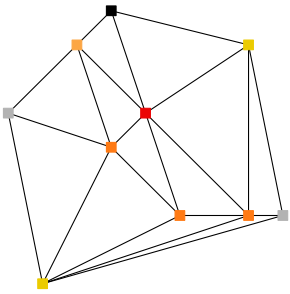
Going beyond grids

- Not everything can be **effectively** cast into a grid...
- ...**graphs** are a nice representations for **irregular structures**.



Going beyond grids

- Not everything can be effectively cast into a grid...
- ...graphs are a nice representations for irregular structures.
- Can we generalize the concept of convolution on graphs?

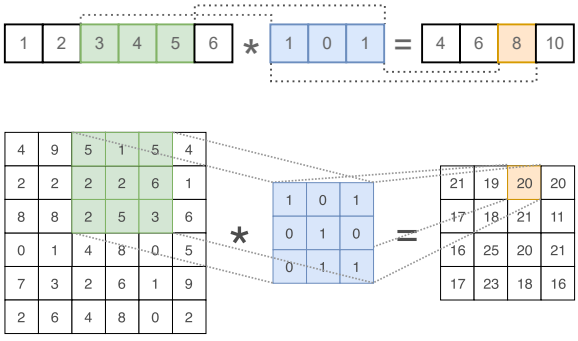


Convolution on Euclidean spaces

The discrete convolution of CNNs:

$$(f \star g)[n] = \sum_{m=-M}^{M} f[n-m]g[m]$$

operates on Euclidean spaces.

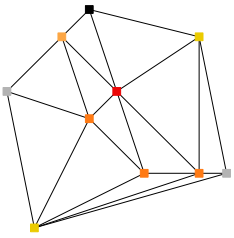


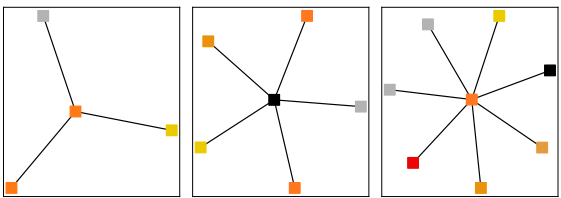
Convolution on non-Euclidean spaces

Moving to non-Euclidean spaces is not that trivial.

Challenges:

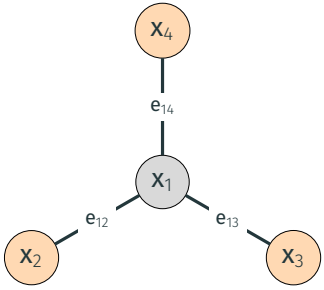
- Variable number of neighbors
- Loss of orientation





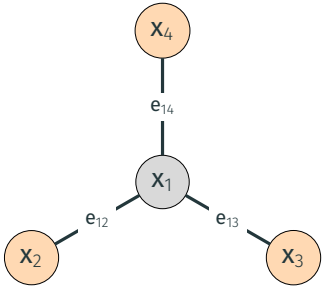
Notation

- Graph $\mathcal{G}\langle\mathcal{V},\mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$



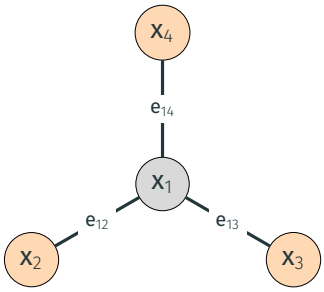
Notation

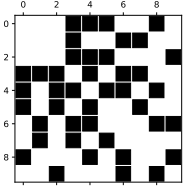
- Graph $\mathcal{G}\langle\mathcal{V}, \mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$
- $\mathbf{X} \in \mathbb{R}^{N \times d_x}$ node-attribute matrix or graph signal
  - $\mathbf{x}_i \in \mathbb{R}^{d_x}$, i-th node attribute vector



Notation

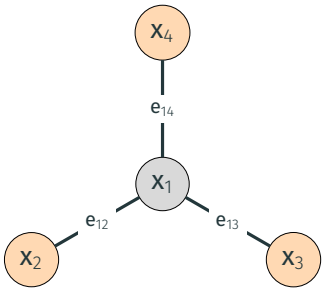
- Graph $\mathcal{G}\langle\mathcal{V}, \mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$
- $\mathbf{X} \in \mathbb{R}^{N \times d_x}$ node-attribute matrix or graph signal
  - $\mathbf{x}_i \in \mathbb{R}^{d_x}$, i-th node attribute vector
- $\mathbf{A} \in \mathbb{R}^{N \times N}$, (weighted) adjacency matrix
  - $a_{ij} \in \mathbb{R}$, edge weight for edge $(i, j) \in \mathcal{E}$

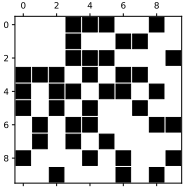




Notation

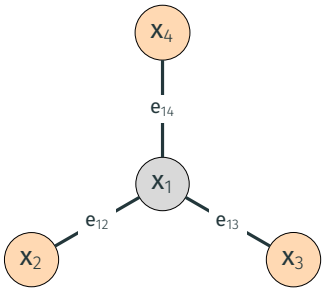
- Graph $\mathcal{G}\langle\mathcal{V},\mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$
- $\mathbf{X} \in \mathbb{R}^{N\times d_x}$ node-attribute matrix or graph signal
  - $\mathbf{x}_i \in \mathbb{R}^{d_x}$, i-th node attribute vector
- $\mathbf{A} \in \mathbb{R}^{N\times N}$, (weighted) adjacency matrix
  - $a_{ij} \in \mathbb{R}$, edge weight for edge $(i,j) \in \mathcal{E}$
- $\mathbf{D} = diag\,(\mathbf{A}\mathbf{1}_N) \in \mathbb{R}^{N\times N}$, degree matrix

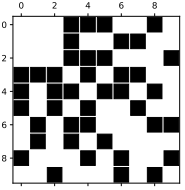




Notation

- Graph $\mathcal{G}\langle\mathcal{V}, \mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$
- $\mathbf{X} \in \mathbb{R}^{N \times d_x}$ node-attribute matrix or graph signal
  - $\mathbf{x}_i \in \mathbb{R}^{d_x}$, i-th node attribute vector
- $\mathbf{A} \in \mathbb{R}^{N \times N}$, (weighted) adjacency matrix
  - $a_{ij} \in \mathbb{R}$, edge weight for edge $(i, j) \in \mathcal{E}$
- $\mathbf{D} = diag\,(\mathbf{A}\mathbf{1}_N) \in \mathbb{R}^{N \times N}$, degree matrix
- $\mathbf{e}_{ij} \in \mathbb{R}^{d_e}$, edge attribute for edge $(i, j) \in \mathcal{E}$

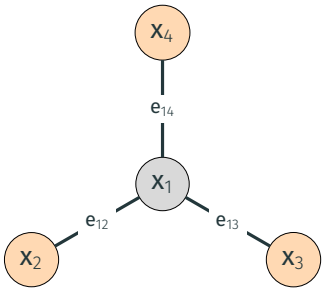


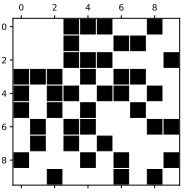


Notation

- Graph $\mathcal{G}\langle\mathcal{V},\mathcal{E}\rangle$: nodes in $\mathcal{V}$ connected by edges in $\mathcal{E}$
- $\mathbf{X} \in \mathbb{R}^{N\times d_x}$ node-attribute matrix or graph signal
  - $\mathbf{x}_i \in \mathbb{R}^{d_x}$, i-th node attribute vector
- $\mathbf{A} \in \mathbb{R}^{N\times N}$, (weighted) adjacency matrix
  - $a_{ij} \in \mathbb{R}$, edge weight for edge $(i,j) \in \mathcal{E}$
- $\mathbf{D} = diag\,(\mathbf{A}\mathbf{1}_N) \in \mathbb{R}^{N\times N}$, degree matrix
- $\mathbf{e}_{ij} \in \mathbb{R}^{d_e}$, edge attribute for edge $(i,j) \in \mathcal{E}$

In the following, we focus on undirected graphs: $\mathbf{A} = \mathbf{A}^\top$



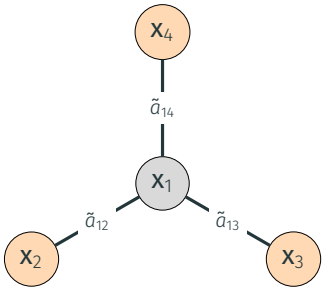


Graph Shift Operator

Graph Shift Operator [1]

A matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N}$ is called a **Graph Shift Operator (GSO)** if it satisfies:

$$\tilde{a}_{ij} = 0 \text{ for } (i,j) \notin \mathcal{E} \text{ and } i \neq j.$$



[1] A. Sandryhaila *et al.*, "Discrete signal processing on graphs," 2013.

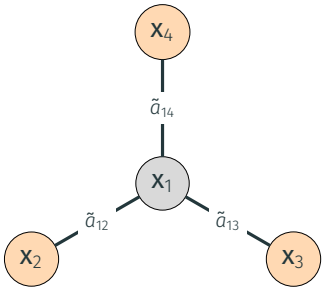
Graph Shift Operator

Graph Shift Operator [1]

A matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N}$ is called a **Graph Shift Operator (GSO)** if it satisfies:

$$\tilde{a}_{ij} = 0 \text{ for } (i,j) \notin \mathcal{E} \text{ and } i \neq j.$$

A GSO $\tilde{\mathbf{A}}$ can be viewed as some function of the adjacency matrix $\mathbf{A}$.



[1] A. Sandryhaila *et al.*, "Discrete signal processing on graphs," 2013.

Graph Shift Operator

Graph Shift Operator [1]

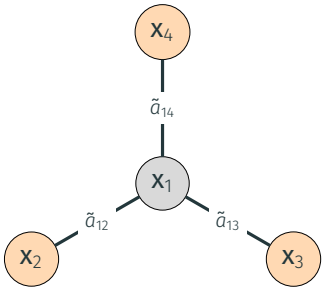
A matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N}$ is called a **Graph Shift Operator (GSO)** if it satisfies:

$$\tilde{a}_{ij} = 0 \text{ for } (i,j) \notin \mathcal{E} \text{ and } i \neq j.$$

A GSO $\tilde{\mathbf{A}}$ can be viewed as some function of the adjacency matrix $\mathbf{A}$.

Examples of GSOs are:

- Laplacian: $\tilde{\mathbf{A}} = \mathbf{L} = \mathbf{D} - \mathbf{A}$
- Random-walk matrix: $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$



[1] A. Sandryhaila *et al.*, "Discrete signal processing on graphs," 2013.

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X}$$

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X} \qquad \mathbf{x}'_i = (\tilde{\mathbf{A}}\mathbf{X})_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j$$

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

- the i-th node attributes are affected only by its neighbors $\mathcal{N}(i)$.

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X} \qquad \mathbf{x}'_i = (\tilde{\mathbf{A}}\mathbf{X})_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j \qquad \mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j$$

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

- the i-th node attributes are affected only by its neighbors $\mathcal{N}(i)$.

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X} \qquad \mathbf{x}'_i = (\tilde{\mathbf{A}}\mathbf{X})_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j \qquad \mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j$$

Using parameter matrix $\Theta \in \mathbb{R}^{d_x \times d_h}$ we can apply the filter on a different space

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta$$

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

- the i-th node attributes are affected only by its neighbors $\mathcal{N}(i)$.

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X} \qquad \mathbf{x}'_i = (\tilde{\mathbf{A}}\mathbf{X})_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j \qquad \mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j$$

Using parameter matrix $\Theta \in \mathbb{R}^{d_x \times d_h}$ we can apply the filter on a different space

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta \qquad \mathbf{h}_i = (\tilde{\mathbf{A}}\mathbf{X}\Theta)_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j\Theta \qquad \mathbf{h}_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j\Theta$$

GSOs for local (learnable) filters

Applying $\tilde{\mathbf{A}}$ to node attributes $\mathbf{X}$ has a **local** action:

- the i-th node attributes are affected only by its neighbors $\mathcal{N}(i)$.

$$\mathbf{X}' = \tilde{\mathbf{A}}\mathbf{X} \qquad \mathbf{x}'_i = (\tilde{\mathbf{A}}\mathbf{X})_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j \qquad \mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j$$

Using parameter matrix $\Theta \in \mathbb{R}^{d_x \times d_h}$ we can apply the filter on a different space

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta \qquad \mathbf{h}_i = (\tilde{\mathbf{A}}\mathbf{X}\Theta)_i = \sum_{j=1}^{N} \tilde{a}_{ji} \cdot \mathbf{x}_j\Theta \qquad \mathbf{h}_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} \cdot \mathbf{x}_j\Theta$$

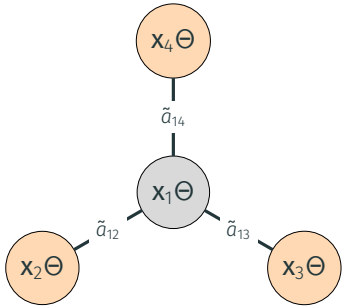
NOTE: We have **local** filters with parameters Θ **shared** among all nodes. Looks familiar?

Graph Convolution

Adding a **nonlinear activation** σ to the operation we have just seen

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta \quad \rightarrow \quad \mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$$

we obtain a nonlinear graph convolutional filter.



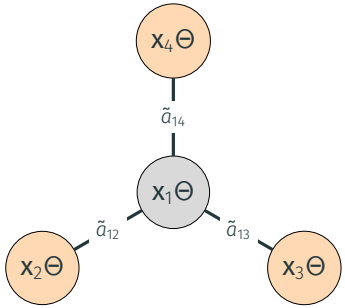
Graph Convolution

Adding a **nonlinear activation** σ to the operation we have just seen

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta \quad \rightarrow \quad \mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$$

we obtain a nonlinear graph convolutional filter.

Since this operation is **differentiable**, we can learn Θ with gradient-based optimization methods.



Graph Convolution

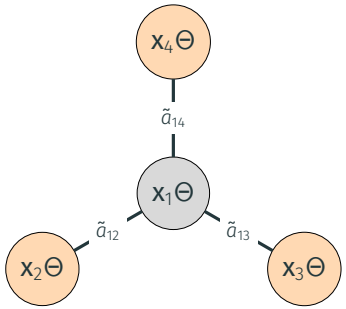
Adding a **nonlinear activation** σ to the operation we have just seen

$$\mathbf{H} = \tilde{\mathbf{A}}\mathbf{X}\Theta \quad \rightarrow \quad \mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$$

we obtain a nonlinear graph convolutional filter.

Since this operation is **differentiable**, we can learn Θ with gradient-based optimization methods.

This enables us to build *neural networks* with *graph*-like inputs, i.e., Graph Neural Networks (GNNs).



Sequence of graph convolutions

What if we apply two graph convolutions in sequence?

$$\mathbf{H}^{(1)} = \tilde{\mathbf{A}}\mathbf{X}\Theta^{(1)}$$
$$\mathbf{H}^{(2)} = \tilde{\mathbf{A}}\mathbf{H}^{(1)}\Theta^{(2)} = \tilde{\mathbf{A}}^2\mathbf{X}\Theta^{(1)}\Theta^{(2)}$$

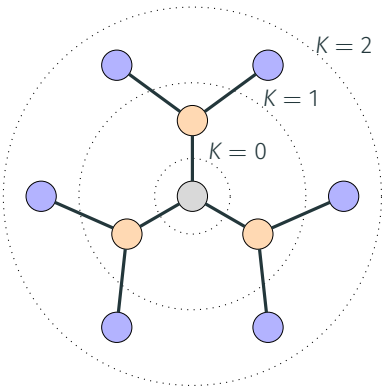
Sequence of graph convolutions

What if we apply two graph convolutions in sequence?

$$\mathbf{H}^{(1)} = \tilde{\mathbf{A}}\mathbf{X}\Theta^{(1)}$$
$$\mathbf{H}^{(2)} = \tilde{\mathbf{A}}\mathbf{H}^{(1)}\Theta^{(2)} = \tilde{\mathbf{A}}^2\mathbf{X}\Theta^{(1)}\Theta^{(2)}$$

Let's focus on the effect of $\tilde{\mathbf{A}}^2\mathbf{X}$:

$$(\tilde{\mathbf{A}}^2\mathbf{X})_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji} (\tilde{\mathbf{A}}\mathbf{X})_j = \sum_{j \in \mathcal{N}(i)} \sum_{k \in \mathcal{N}(j)} \tilde{a}_{ji} \cdot \tilde{a}_{kj} \cdot \mathbf{x}_k$$



Sequence of graph convolutions

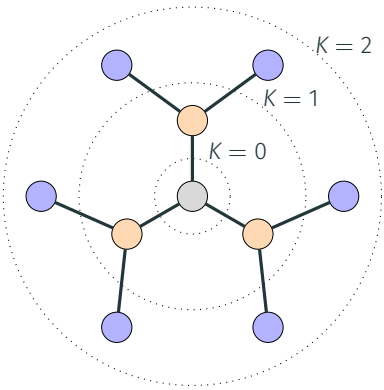
What if we apply two graph convolutions in sequence?

$$\mathbf{H}^{(1)} = \tilde{\mathbf{A}}\mathbf{X}\Theta^{(1)}$$
$$\mathbf{H}^{(2)} = \tilde{\mathbf{A}}\mathbf{H}^{(1)}\Theta^{(2)} = \tilde{\mathbf{A}}^2\mathbf{X}\Theta^{(1)}\Theta^{(2)}$$

Let's focus on the effect of $\tilde{\mathbf{A}}^2\mathbf{X}$:

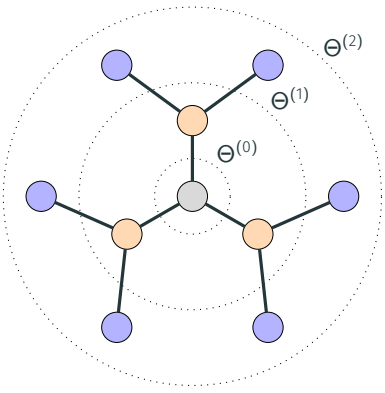
$$(\tilde{\mathbf{A}}^2\mathbf{X})_i = \sum_{j \in \mathcal{N}(i)} \tilde{a}_{ji}(\tilde{\mathbf{A}}\mathbf{X})_j = \sum_{j \in \mathcal{N}(i)} \sum_{k \in \mathcal{N}(j)} \tilde{a}_{ji} \cdot \tilde{a}_{kj} \cdot \mathbf{x}_k$$

The second convolution aggregates information from the 2-hop neighbors, i.e., the neighbors' neighbors.



K-th-order filters

To aggregate information **up to the K-th-order neighborhood**, we can either use

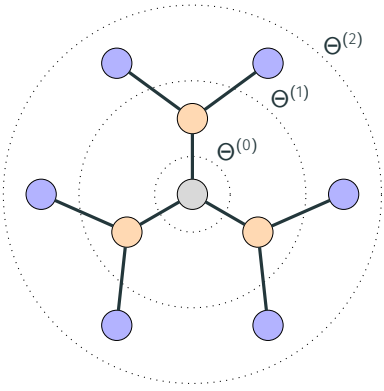


K-th-order filters

To aggregate information **up to the K-th-order neighborhood**, we can either use

- polynomial filters

$$\mathbf{H}^{(K)} = \sum_{k=0}^{K} \tilde{\mathbf{A}}^k \mathbf{X} \Theta^{(k)}$$



K-th-order filters

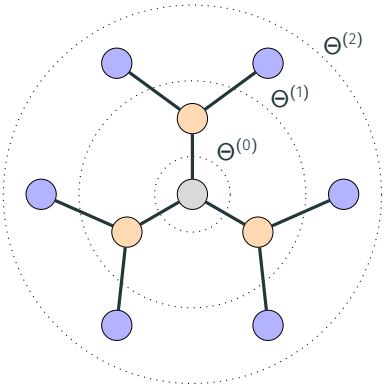
To aggregate information **up to the K-th-order neighborhood**, we can either use

- polynomial filters

$$\mathbf{H}^{(K)} = \sum_{k=0}^{K} \tilde{\mathbf{A}}^k \mathbf{X} \Theta^{(k)}$$

- a sequence of first-order-neighborhood filters

$$\mathbf{H}^{(0)} = \mathbf{X} \qquad \mathbf{H}^{(k)} = \tilde{\mathbf{A}} \mathbf{H}^{(k-1)} \Theta^{(k)}$$



K-th-order filters

To aggregate information **up to the K-th-order neighborhood**, we can either use

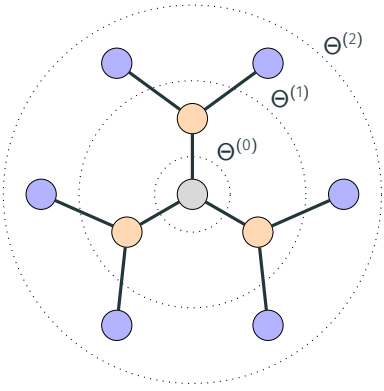
- polynomial filters

$$\mathbf{H}^{(K)} = \sum_{k=0}^{K} \tilde{\mathbf{A}}^k \mathbf{X} \Theta^{(k)}$$

- a sequence of first-order-neighborhood filters

$$\mathbf{H}^{(0)} = \mathbf{X} \qquad \mathbf{H}^{(k)} = \tilde{\mathbf{A}} \mathbf{H}^{(k-1)} \Theta^{(k)}$$

...eventually with nonlinearities.



Graph convolutional layers

Examples of graph convolutional layers from the literature:

- GCN [2]: $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}(\mathbf{I}_N + \mathbf{A})\mathbf{D}^{-1/2}$
- Diffusion Convolution [3]: $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$
- GIN [4]: $\tilde{\mathbf{A}} = \mathbf{A} + (1 + \epsilon) \cdot \mathbf{I}_N$

[2] T. N. Kipf *et al.*, "Semi-supervised classification with graph convolutional networks," 2016.

[3] Y. Li *et al.*, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," 2017.

[4] K. Xu *et al.*, "How powerful are graph neural networks?" 2019.

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

- dependent only on the graph **topology**;

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

- dependent only on the graph **topology**;
- **localized** in the root node's neighborhood;

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

- dependent only on the graph **topology**;
- **localized** in the root node's neighborhood;
- **shared** among all nodes in the graph (i.e., applied equally everywhere).

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

- dependent only on the graph **topology**;
- **localized** in the root node's neighborhood;
- **shared** among all nodes in the graph (i.e., applied equally everywhere).

What if we want to:

- take into account **edge attributes**?

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

- dependent only on the graph **topology**;
- **localized** in the root node's neighborhood;
- **shared** among all nodes in the graph (i.e., applied equally everywhere).

What if we want to:

- take into account **edge attributes**?
- make the filter dependent also on the **nodes' features**, not only on the topology?

A more expressive framework

Graph convolutions based on GSOs are a powerful tool to learn graph filters:

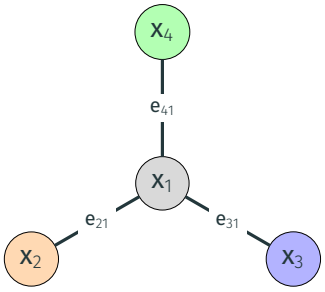
- dependent only on the graph **topology**;
- **localized** in the root node's neighborhood;
- **shared** among all nodes in the graph (i.e., applied equally everywhere).

What if we want to:

- take into account **edge attributes**?
- make the filter dependent also on the **nodes' features**, not only on the topology?
- e.g., **weigh** the contribution of a neighbor based on the **root node features**?

Message passing

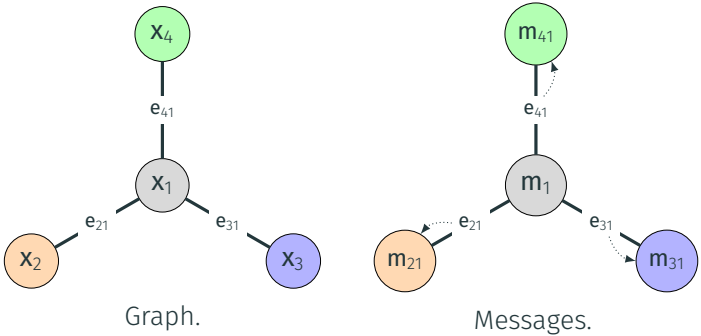
Message-passing neural networks [5]



Graph.

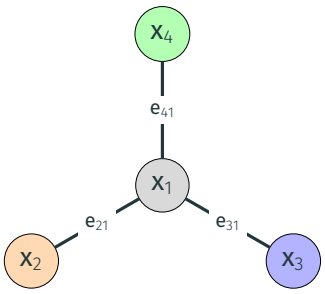
[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message-passing neural networks [5]

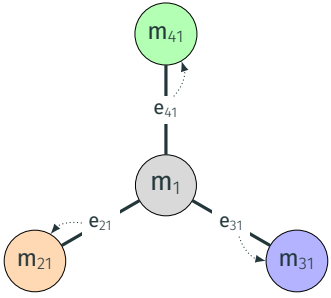


[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

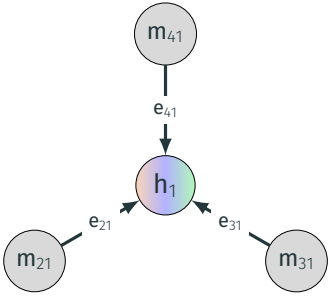
Message-passing neural networks [5]



Graph.



Messages.



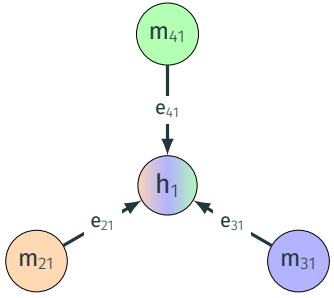
Propagation.

[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message-passing neural networks

A general scheme for message-passing (MP) networks [5]:

$$\mathbf{h}_i = \gamma\left(\mathbf{x}_i, \underset{j\in\mathcal{N}(i)}{\mathit{Aggr}}\left\{\phi\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ji}\right)\right\}\right)$$



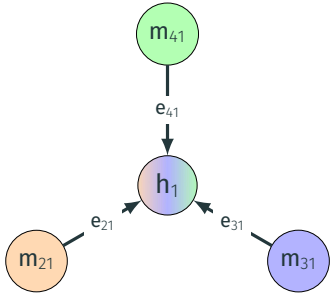
[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message-passing neural networks

A general scheme for message-passing (MP) networks [5]:

$$\mathbf{h}_i = \gamma\left(\mathbf{x}_i, \underset{j\in\mathcal{N}(i)}{\mathit{Aggr}}\left\{\phi\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ji}\right)\right\}\right)$$

- ϕ **message function**, depends on $\mathbf{x}_i$, $\mathbf{x}_j$ and possibly the edge attribute $\mathbf{e}_{ji}$;



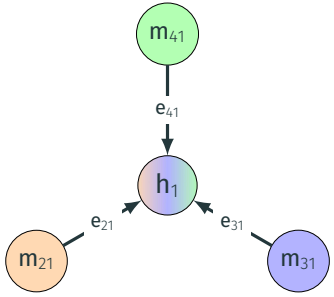
[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message-passing neural networks

A general scheme for message-passing (MP) networks [5]:

$$\mathbf{h}_i = \gamma\left(\mathbf{x}_i, \underset{j\in\mathcal{N}(i)}{\boldsymbol{Aggr}}\left\{\phi\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ji}\right)\right\}\right)$$

- ϕ **message function**, depends on $\mathbf{x}_i$, $\mathbf{x}_j$ and possibly the edge attribute $\mathbf{e}_{ji}$;
- $\boldsymbol{Aggr}$: permutation-invariant **aggregation function** (e.g., sum, mean, max);



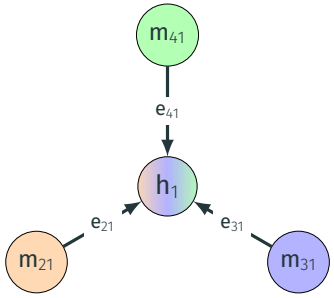
[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Message-passing neural networks

A general scheme for message-passing (MP) networks [5]:

$$\mathbf{h}_i = \gamma\left(\mathbf{x}_i, \underset{j\in\mathcal{N}(i)}{\mathit{Aggr}}\left\{\phi\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ji}\right)\right\}\right)$$

- ϕ **message function**, depends on $\mathbf{x}_i$, $\mathbf{x}_j$ and possibly the edge attribute $\mathbf{e}_{ji}$;
- Aggr: permutation-invariant **aggregation function** (e.g., sum, mean, max);
- γ **update function**, to obtain new attributes from aggregated messages and previous attributes.



[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

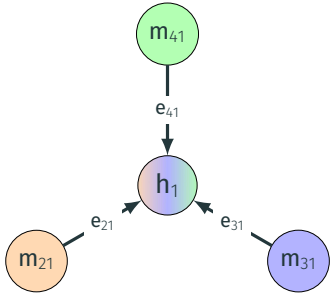
Message-passing neural networks

A general scheme for message-passing (MP) networks [5]:

$$\mathbf{h}_i = \gamma\left(\mathbf{x}_i, \underset{j\in\mathcal{N}(i)}{\boldsymbol{Aggr}}\left\{\phi\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ji}\right)\right\}\right)$$

- ϕ **message function**, depends on $\mathbf{x}_i$, $\mathbf{x}_j$ and possibly the edge attribute $\mathbf{e}_{ji}$;
- $\boldsymbol{Aggr}$: permutation-invariant **aggregation function** (e.g., sum, mean, max);
- γ **update function**, to obtain new attributes from aggregated messages and previous attributes.

Note: ϕ and γ are usually **parametric** (e.g., MLPs).



[5] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," 2017.

Isotropic vs. Anisotropic MP

The MP equation is the most general (and expressive) form of GNN, encompassing also the convolutional graph filters discussed before.

Isotropic vs. Anisotropic MP

The MP equation is the most general (and expressive) form of GNN, encompassing also the convolutional graph filters discussed before.

E.g., $\mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$ can be rewritten as:

$$\mathbf{h}_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} a_{ji}\mathbf{x}_j\Theta\right)$$

where:

- $\phi\left(\mathbf{x}_j\right) = a_{ji}\mathbf{x}_j\Theta$
- $\boldsymbol{Aggr}$ is the sum
- $\gamma(\,\cdot\,) = \sigma(\,\cdot\,)$

Isotropic vs. Anisotropic MP

The MP equation is the most general (and expressive) form of GNN, encompassing also the convolutional graph filters discussed before.

E.g., $\mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$ can be rewritten as:

$$\mathbf{h}_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} a_{ji}\mathbf{x}_j\Theta\right)$$

where:

- $\phi\left(\mathbf{x}_j\right) = a_{ji}\mathbf{x}_j\Theta$
- $\boldsymbol{Aggr}$ is the sum
- $\gamma(\,\cdot\,) = \sigma(\,\cdot\,)$

MP operations whose message function depends **only on the sender node's features** are called isotropic.

Isotropic vs. Anisotropic MP

The MP equation is the most general (and expressive) form of GNN, encompassing also the convolutional graph filters discussed before.

E.g., $\mathbf{H} = \sigma\left(\tilde{\mathbf{A}}\mathbf{X}\Theta\right)$ can be rewritten as:

$$\mathbf{h}_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} a_{ji}\mathbf{x}_j\Theta\right)$$

where:

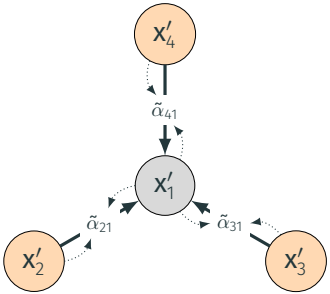
- $\phi\left(\mathbf{x}_j\right) = a_{ji}\mathbf{x}_j\Theta$
- $\boldsymbol{Aggr}$ is the sum
- $\gamma(\,\cdot\,) = \sigma(\,\cdot\,)$

MP operations whose message function depends **only on the sender node's features** are called isotropic.

We call them anisotropic when also **edge's or receiver node's features** are exploited.

Graph attention network (GAT)

Graph attention networks [6] are an example of anisotropic MP.



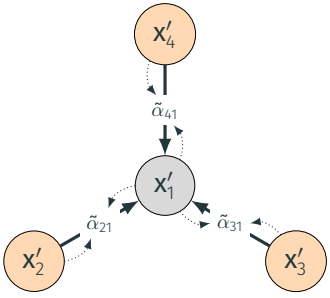
$\|$ indicates concatenation

[6] P. Velickovic *et al.*, "Graph attention networks," 2017.

Graph attention network (GAT)

Graph attention networks [6] are an example of anisotropic MP.

1. Transform node features: $\mathbf{x}'_i = \mathbf{x}_i \Theta_1$, with $\Theta_1 \in \mathbb{R}^{d_x \times d_h}$.



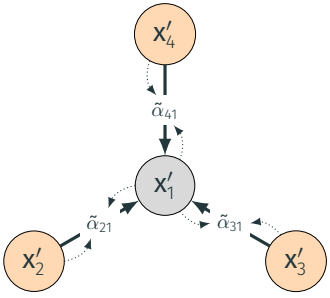
$\|$ indicates concatenation

[6] P. Velickovic *et al.*, "Graph attention networks," 2017.

Graph attention network (GAT)

Graph attention networks [6] are an example of anisotropic MP.

1. Transform node features: $\mathbf{x}'_i = \mathbf{x}_i \Theta_1$, with $\Theta_1 \in \mathbb{R}^{d_x \times d_h}$.
2. Compute **attention scores** between neighbors:
   2.1 Score: $\alpha_{ji} = \sigma\left([\mathbf{x}'_i \parallel \mathbf{x}'_j]\theta_2\right)$, with $\theta_2 \in \mathbb{R}^{2d_h \times 1}$.



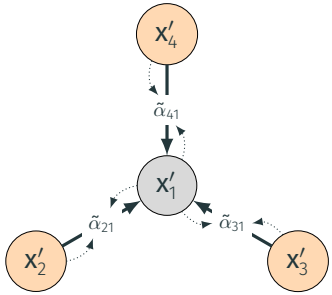
$\parallel$ indicates concatenation

[6] P. Velickovic *et al.*, "Graph attention networks," 2017.

Graph attention network (GAT)

Graph attention networks [6] are an example of anisotropic MP.

1. Transform node features: $\mathbf{x}'_i = \mathbf{x}_i \Theta_1$, with $\Theta_1 \in \mathbb{R}^{d_x \times d_h}$.
2. Compute **attention scores** between neighbors:
   - 2.1 Score: $\alpha_{ji} = \sigma\left([\mathbf{x}'_i \parallel \mathbf{x}'_j]\theta_2\right)$, with $\theta_2 \in \mathbb{R}^{2d_h \times 1}$.
   - 2.2 Normalize with Softmax: $\tilde{\alpha}_{ji} = \dfrac{\exp\left(\alpha_{ji}\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\alpha_{ki}\right)}$



$\parallel$ indicates concatenation

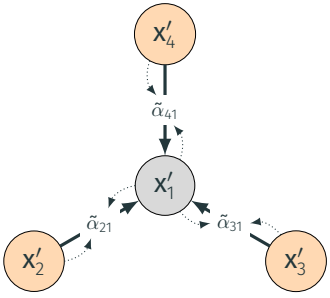
[6] P. Velickovic *et al.*, "Graph attention networks," 2017.

Graph attention network (GAT)

Graph attention networks [6] are an example of anisotropic MP.

1. Transform node features: $\mathbf{x}'_i = \mathbf{x}_i \Theta_1$, with $\Theta_1 \in \mathbb{R}^{d_x \times d_h}$.
2. Compute **attention scores** between neighbors:
   2.1 Score: $\alpha_{ji} = \sigma\left([\mathbf{x}'_i \parallel \mathbf{x}'_j]\theta_2\right)$, with $\theta_2 \in \mathbb{R}^{2d_h \times 1}$.
   2.2 Normalize with Softmax: $\tilde{\alpha}_{ji} = \dfrac{\exp\left(\alpha_{ji}\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\alpha_{ki}\right)}$
3. Aggregate using attention coefficients as weights:

$$\mathbf{h}_i = \sum_{j \in \mathcal{N}(i)} \tilde{\alpha}_{ji} \mathbf{x}'_j$$

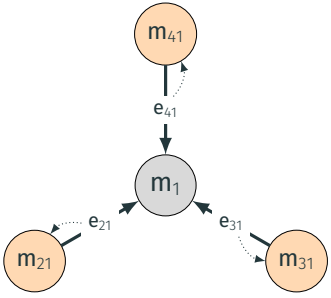


$\parallel$ indicates concatenation

[6] P. Velickovic *et al.*, "Graph attention networks," 2017.

Edge-conditioned convolution [7]

Key idea: incorporate edge attributes into the messages.



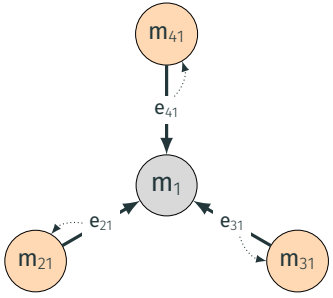
[7] M. Simonovsky *et al.*, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," 2017.

Edge-conditioned convolution [7]

Key idea: incorporate edge attributes into the messages.

Use a MLP $\rho : \mathbb{R}^{d_e} \to \mathbb{R}^{d_x \times d_h}$ to **generate weights**:

$$\Theta_{ji} = \rho(\mathbf{e}_{ji})$$



[7] M. Simonovsky *et al.*, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," 2017.

Edge-conditioned convolution [7]

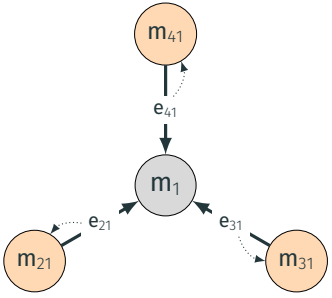
Key idea: incorporate edge attributes into the messages.

Use a MLP $\rho : \mathbb{R}^{d_e} \to \mathbb{R}^{d_x \times d_h}$ to **generate weights**:

$$\Theta_{ji} = \rho(\mathbf{e}_{ji})$$

Use the edge-dependent weights to compute messages:

$$\mathbf{h}_i = \mathbf{x}_i \Theta_i + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j \Theta_{ji}$$



[7] M. Simonovsky *et al.*, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," 2017.

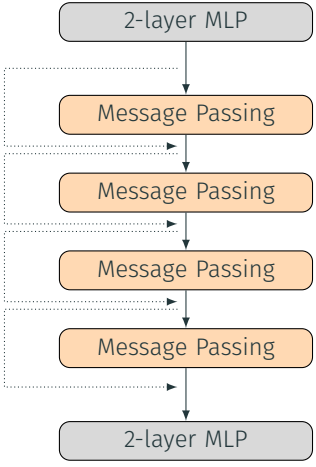
The zoo of GNNs

GCNConv *Kipf & Welling*	**ChebConv** *Defferrard et al.*	**GraphSageConv** *Hamilton et al.*	**ARMAConv** *Bianchi et al.*
ECCConv *Simonovsky & Komodakis*	**GATConv** *Velickovic et al.*	**GCSConv** *Bianchi et al.*	**APPNPConv** *Klicpera et al.*
GINConv *Xu et al.*	**DiffusionConv** *Li et al.*	**GatedGraphConv** *Li et al.*	**AGNNConv** *Thekumparampil et al.*
TAGConv *Du et al.*	**CrystalConv** *Xie & Grossman*	**EdgeConv** *Wang et al.*	**MessagePassing** *Gilmer et al.*

A good recipe [8]

Architecture:

- Pre- and post-process node features using 2-layer MLPs;
- 4-6 message-passing steps.



[8] J. You *et al.*, "Design space for graph neural networks," 2020.

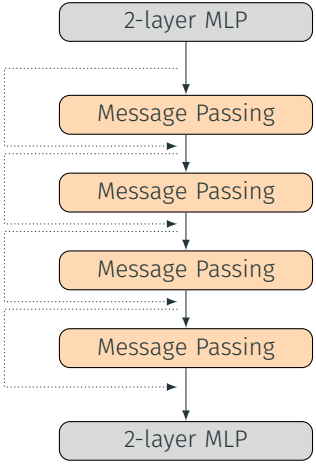
A good recipe [8]

Architecture:

- Pre- and post-process node features using 2-layer MLPs;
- 4-6 message-passing steps.

Message passing at l-th layer:

- Message: $\mathbf{m}_{ji}^{l} = \text{PReLU}\left(\text{BatchNorm}\left(\mathbf{h}_{j}^{l}\Theta^{l} + \mathbf{b}^{l}\right)\right)$
- Aggregation: sum, i.e., $\mathbf{m}_{i}^{l} = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ji}^{l}$
- Update: $\mathbf{h}_{i}^{l+1} = \mathbf{h}_{i}^{l} \,||\, \mathbf{m}_{i}^{l}$;



[8] J. You *et al.*, "Design space for graph neural networks," 2020.

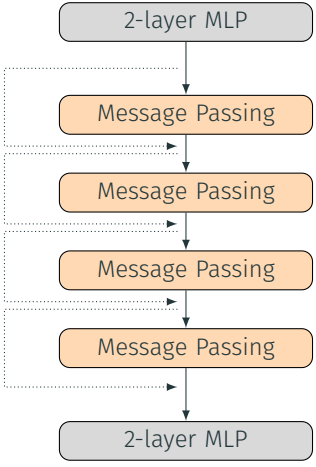
A good recipe [8]

Architecture:

- Pre- and post-process node features using 2-layer MLPs;
- 4-6 message-passing steps.

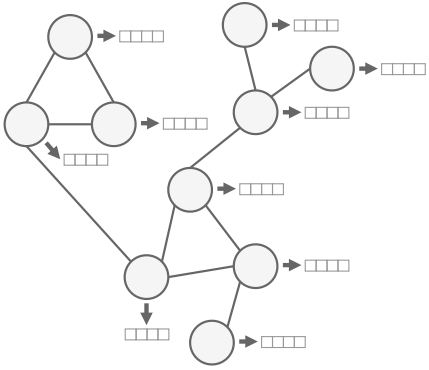
Message passing at l-th layer:

- Message: $\mathbf{m}_{ji}^{l} = \text{PReLU}\left(\text{BatchNorm}\left(\mathbf{h}_{j}^{l}\Theta^{l} + \mathbf{b}^{l}\right)\right)$
- Aggregation: sum, i.e., $\mathbf{m}_{i}^{l} = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ji}^{l}$
- Update: $\mathbf{h}_{i}^{l+1} = \mathbf{h}_{i}^{l} \,||\, \mathbf{m}_{i}^{l}$;

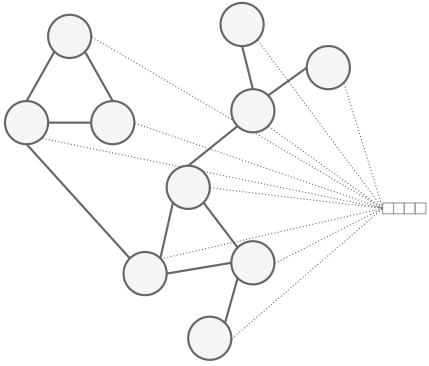


[8] J. You *et al.*, "Design space for graph neural networks," 2020.

How do we use this?



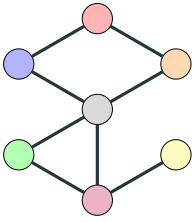
Node-level learning.
(e.g., social networks)



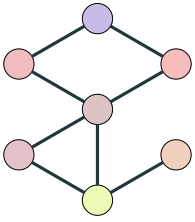
Graph-level learning.
(e.g., molecules)

A little warning

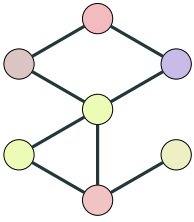
Graph convolutions act as **low-pass** filters, reducing the dissimilarity of neighbors' features at every application.



$l = 0$

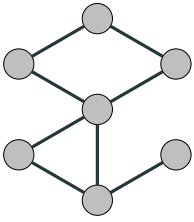


$l = 1$



$l = 2$

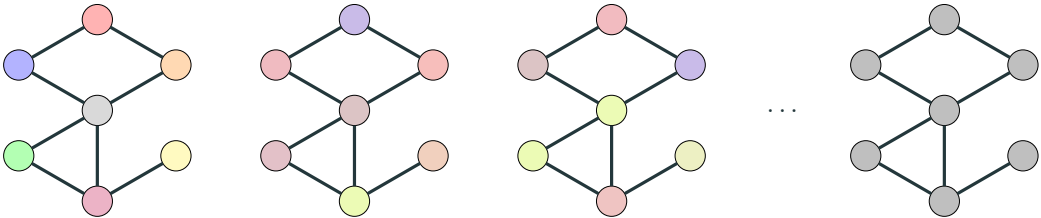
$\ldots$



$l = K$

A little warning

Graph convolutions act as **low-pass** filters, reducing the dissimilarity of neighbors' features at every application.

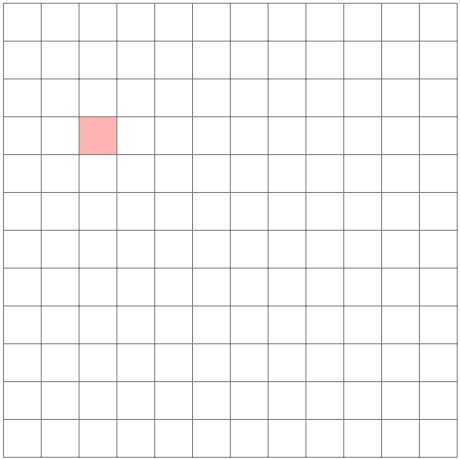


$l = 0$ $l = 1$ $l = 2$ $l = K$

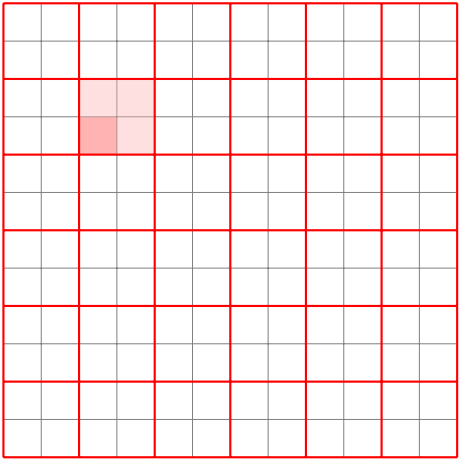
This phenomenon is reffered to as over-smoothing. Can you guess why it can be harmful?

Pooling on Graphs

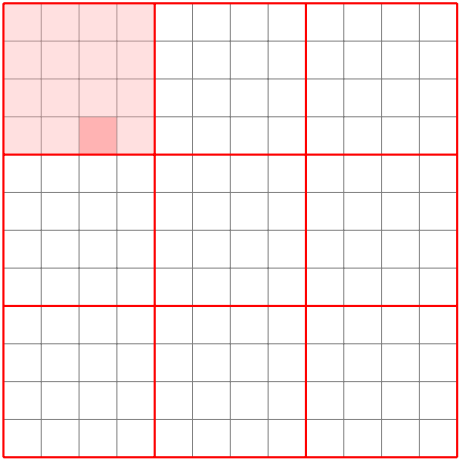
Pooling in CNNs



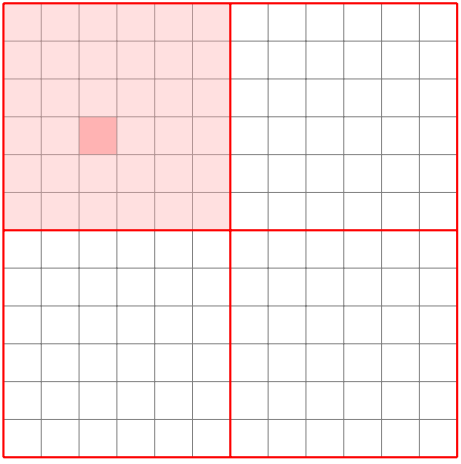
Pooling in CNNs



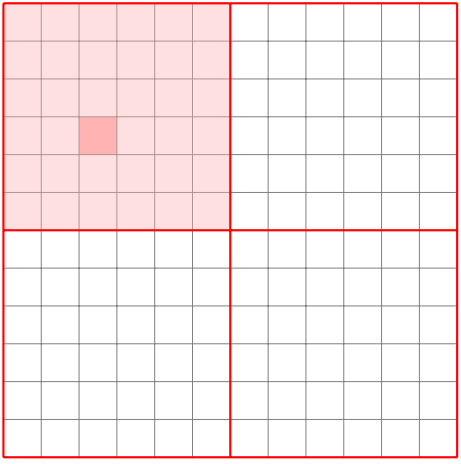
Pooling in CNNs

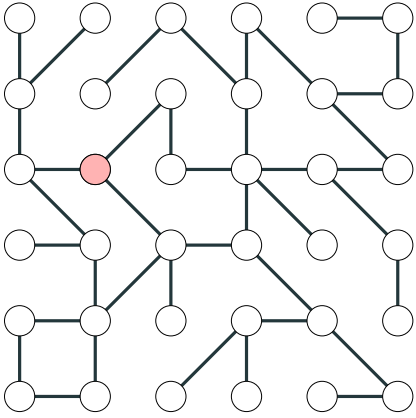


Pooling in CNNs

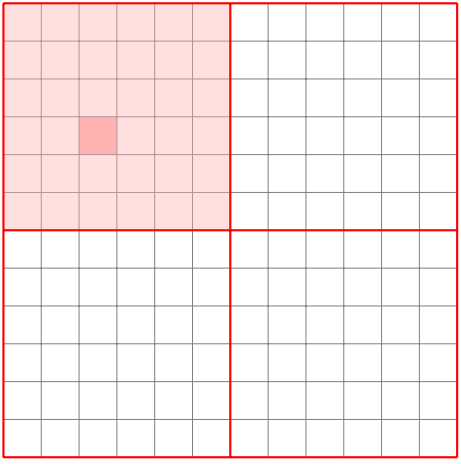


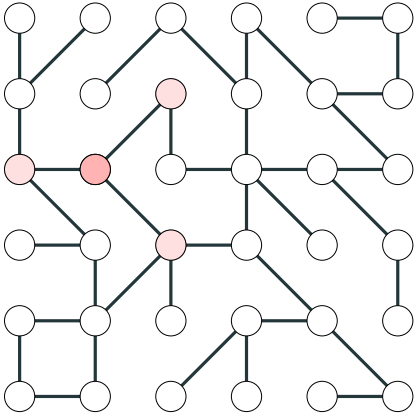
Pooling in CNNs



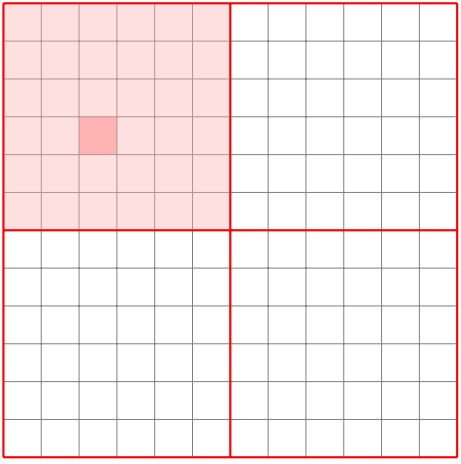


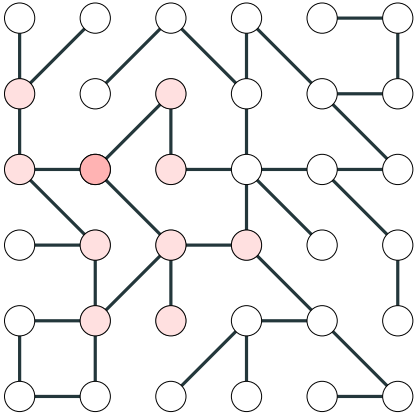
Pooling in CNNs



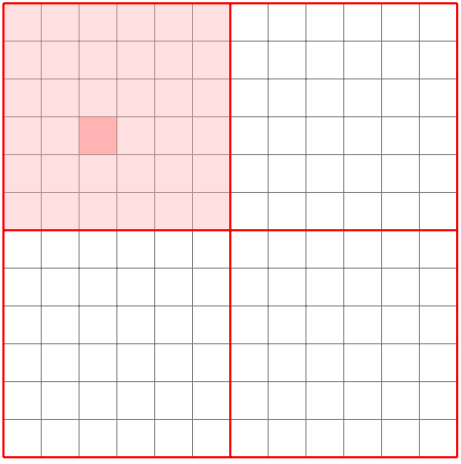


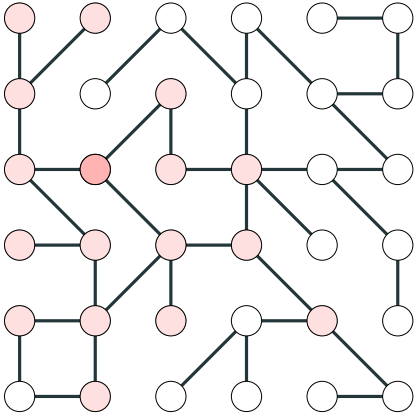
Pooling in CNNs





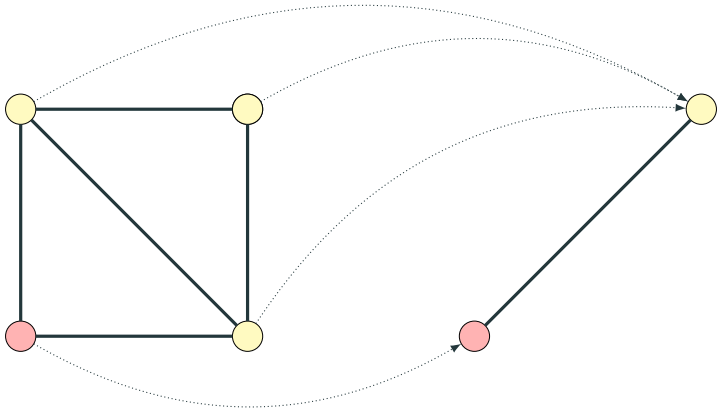
Pooling in CNNs





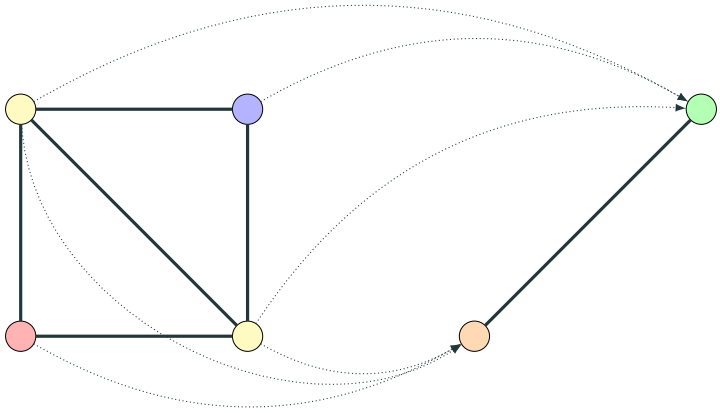
Graph pooling by example

Strategy 1: aggregate same attributes (Candy Crush pooling).



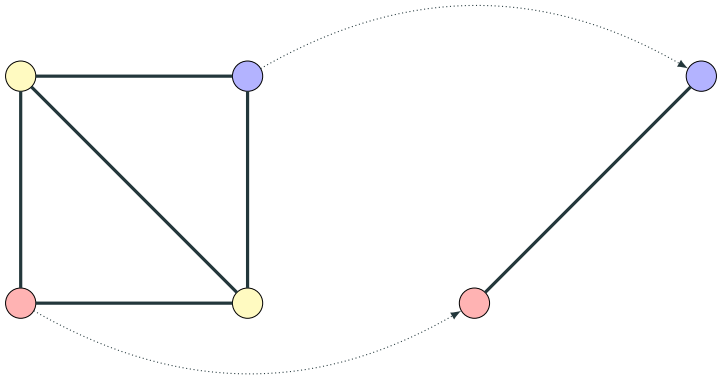
Graph pooling by example

Strategy 2: aggregate cliques.



Graph pooling by example

Strategy 3: keep only some types/colors.



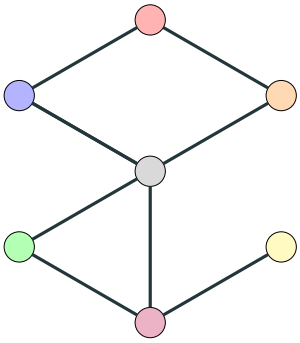
Three main questions [9]

1. How to identify **groups of related nodes**?
2. How to get **new node attributes** from the groups?
3. How to **connect** the new nodes?

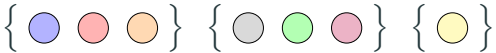
[9] D. Grattarola *et al.*, "Understanding pooling in graph neural networks," 2022.

Step 1: Select

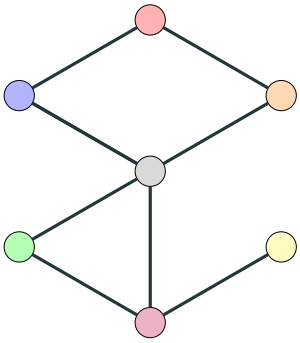
Selecting nodes



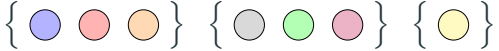
Example 1: partition.



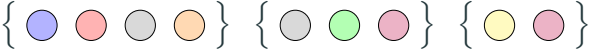
Selecting nodes



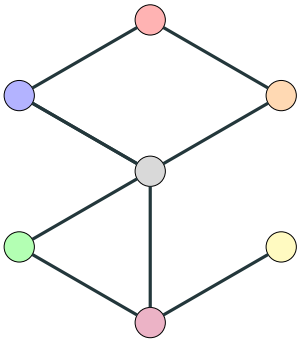
Example 1: partition.



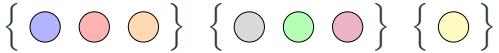
Example 2: cover (possible overlaps).



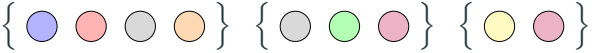
Selecting nodes



Example 1: partition.



Example 2: cover (possible overlaps).



Example 3: sparse.

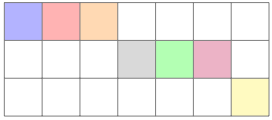
Selecting nodes

The **selection** stage computes K **supernodes**:

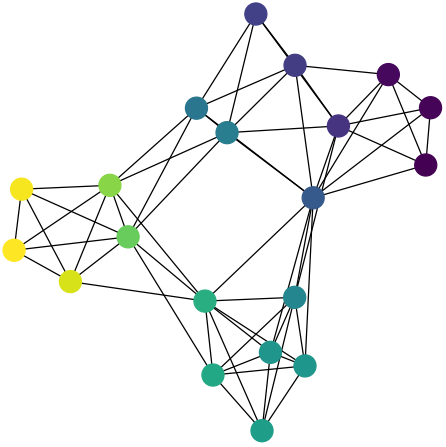
$$\textsc{Sel} : \mathcal{G} \mapsto \mathcal{S} = \{\mathcal{S}_1, \dots, \mathcal{S}_K\}.$$

Each supernode is a set of nodes (with relative features) associated with a score:

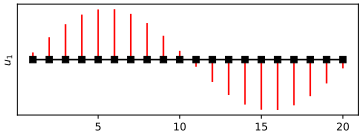
$$\mathcal{S}_k = \{(\mathbf{x}_i, s_{ki}) \mid s_{ki} > 0\}$$

$\mathbf{S} =$  $\in \mathbb{R}^{K \times N}$

Spectral clustering [11]



The low-frequency eigenvectors of the Laplacian naturally cluster the nodes.

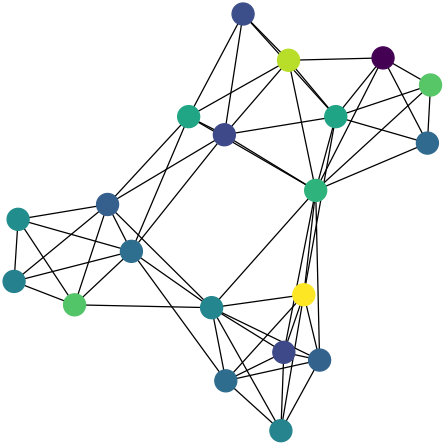


Idea: run k-means clustering (or similar) using the first few eigenvectors.

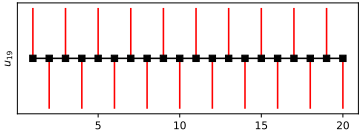
[10] J. Shi *et al.*, "Normalized cuts and image segmentation," 2000.

[11] U. Von Luxburg, "A tutorial on spectral clustering," 2007.

Node decimation [13]



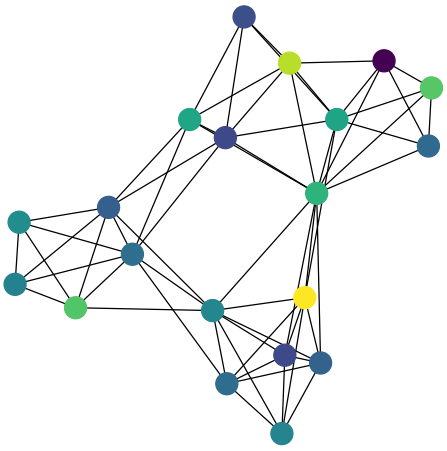
Alternative: use the highest-frequency eigenvector to do something similar to a regular subsampling.



[12] L. Palagi *et al.*, "Computational approaches to max-cut," 2012.

[13] F. M. Bianchi *et al.*, *Hierarchical Representation Learning in Graph Neural Networks with Node Decimation Pooling*, 2019.

Some problems



Problems with spectral methods:

- Computing eigenvectors is **expensive** ($O(N^3)$);
- They do not consider **attributes**.

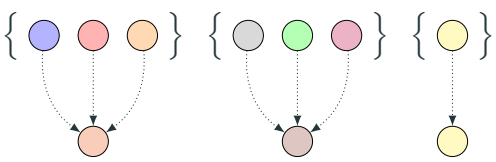
But we get the general idea...

Step 2: Reduce

Reducing supernodes

The **reduction** stage aggregates the supernodes in a **permutation-invariant** way:

$$\textsc{Red} : \mathcal{G}, \mathcal{S}_k \mapsto \mathbf{x}'_k$$



Typical approach is to take a **weighted sum** (weights given by the scores in the supernodes):

$$\mathbf{X}' = \mathbf{S}\mathbf{X} \quad (\in \mathbb{R}^{K \times d_x})$$

Step 3: Connect

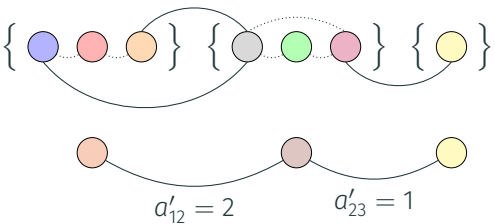
Connecting supernodes

The **connection** function decides whether two supernodes are connected (and, in case, computes the associated attributes):

$$\textsc{Con} : \mathcal{G}, \mathcal{S}_k, \mathcal{S}_l \mapsto (a'_{kl}, \mathbf{e}'_{kl})$$

Typical approach is again to take a **weighted sum** of edges between two supernodes:

$$\mathbf{A}' = \mathbf{S}\mathbf{A}\mathbf{S}^\top \quad (\in \mathbb{R}^{K \times K})$$



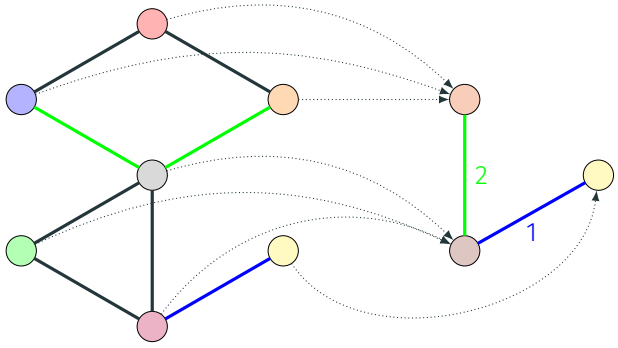
Select, Reduce, Connect [9]

Putting everything together:

$$\underbrace{\mathcal{S} = \{\mathcal{S}_k\}_{k=1:K} = \textsc{Sel}(\mathcal{G})}_{\text{Selection}};$$

$$\underbrace{\mathcal{X}' = \{\textsc{Red}(\mathcal{G}, \mathcal{S}_k)\}_{k=1:K}}_{\text{Reduction}};$$

$$\underbrace{\mathcal{E}' = \{\textsc{Con}(\mathcal{G}, \mathcal{S}_k, \mathcal{S}_l)\}_{k,l=1:K}}_{\text{Connection}};$$



[9] D. Grattarola *et al.*, "Understanding pooling in graph neural networks," 2022.

Pooling methods

Pooling methods

A few ideas:

1. **Graclus** [14], approximately halves nodes:
   1.1 select a (not merged) node i randomly;
   1.2 merge i with (not merged) neighbor j such that $\underset{j}{\text{argmax}} \, a_{ji} \left(\frac{1}{deg_i} + \frac{1}{deg_j} \right)$

[14] I. S. Dhillon *et al.*, "Weighted graph cuts without eigenvectors a multilevel approach," 2007.

[15] E. Luzhnica *et al.*, "Clique pooling for graph classification," 2019.

[16] E. Noutahi *et al.*, "Towards Interpretable Sparse Graph Representation Learning with Laplacian Pooling," 2019.

Pooling methods

A few ideas:

1. **Graclus** [14], approximately halve nodes:
   1.1 select a (not merged) node i randomly;
   1.2 merge i with (not merged) neighbor j such that $\operatorname*{argmax}_j a_{ji} \left(\frac{1}{deg_i} + \frac{1}{deg_j} \right)$
2. **Clique Pooling** [15]: merge together cliques, i.e., fully-connected subgraphs.

[14] I. S. Dhillon *et al.*, "Weighted graph cuts without eigenvectors a multilevel approach," 2007.

[15] E. Luzhnica *et al.*, "Clique pooling for graph classification," 2019.

[16] E. Noutahi *et al.*, "Towards Interpretable Sparse Graph Representation Learning with Laplacian Pooling," 2019.

Pooling methods

A few ideas:

1. **Graclus** [14], approximately halve nodes:
   1.1 select a (not merged) node i randomly;
   1.2 merge i with (not merged) neighbor j such that $\operatorname*{argmax}_j a_{ji} \left(\frac{1}{deg_i} + \frac{1}{deg_j} \right)$
2. **Clique Pooling** [15]: merge together cliques, i.e., fully-connected subgraphs.
3. **LaPool** [16]: select "leaders" that have highest local variation $\|\mathbf{L}\mathbf{X}\|$ w.r.t. all their neighbors. Create clusters by assigning nodes to nearest leader.

[14] I. S. Dhillon *et al.*, "Weighted graph cuts without eigenvectors a multilevel approach," 2007.

[15] E. Luzhnica *et al.*, "Clique pooling for graph classification," 2019.

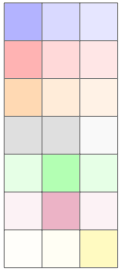
[16] E. Noutahi *et al.*, "Towards Interpretable Sparse Graph Representation Learning with Laplacian Pooling," 2019.

Learning to pool

Key idea: learn to output $\mathbf{S}^\top$ by giving node features $\mathbf{X}$ as input to a neural network.

- **DiffPool** [17]: GNN for $\mathbf{S}^\top$, regularize with "link prediction" loss;

$$\phi(\mathbf{X}) = \mathbf{S}^\top = \quad \in \mathbb{R}^{N \times K}$$



[17] R. Ying *et al.*, "Hierarchical Graph Representation Learning withDifferentiable Pooling," 2018.

[18] F. M. Bianchi *et al.*, "Spectral Clustering with Graph Neural Networks for Graph Pooling," 2020.

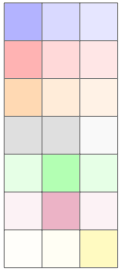
[19] C. Bodnar *et al.*, "Deep Graph Mapper: Seeing Graphs through the Neural Lens," 2020.

Learning to pool

Key idea: learn to output $\mathbf{S}^\top$ by giving node features $\mathbf{X}$ as input to a neural network.

- **DiffPool** [17]: GNN for $\mathbf{S}^\top$, regularize with "link prediction" loss;
- **MinCutPool** [18]: MLP for $\mathbf{S}^\top$, regularize with "minimum cut" loss (same objective as spectral clustering);

$$\phi(\mathbf{X}) = \mathbf{S}^\top = \quad \in \mathbb{R}^{N \times K}$$



[17] R. Ying *et al.*, "Hierarchical Graph Representation Learning withDifferentiable Pooling," 2018.

[18] F. M. Bianchi *et al.*, "Spectral Clustering with Graph Neural Networks for Graph Pooling," 2020.

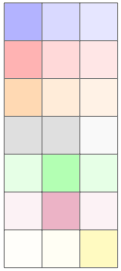
[19] C. Bodnar *et al.*, "Deep Graph Mapper: Seeing Graphs through the Neural Lens," 2020.

Learning to pool

Key idea: learn to output $\mathbf{S}^\top$ by giving node features $\mathbf{X}$ as input to a neural network.

- **DiffPool** [17]: GNN for $\mathbf{S}^\top$, regularize with "link prediction" loss;
- **MinCutPool** [18]: MLP for $\mathbf{S}^\top$, regularize with "minimum cut" loss (same objective as spectral clustering);
- **Deep Graph Mapper** [19]: combine Mapper [20] and GCN [2] to compute clusters.

$$\phi(\mathbf{X}) = \mathbf{S}^\top = \quad \in \mathbb{R}^{N \times K}$$



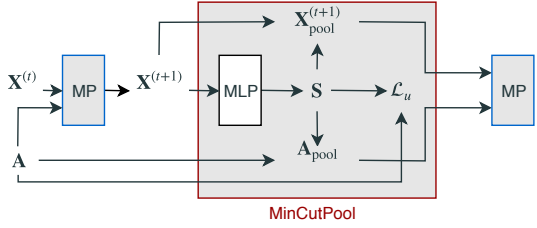
[17] R. Ying *et al.*, "Hierarchical Graph Representation Learning withDifferentiable Pooling," 2018.

[18] F. M. Bianchi *et al.*, "Spectral Clustering with Graph Neural Networks for Graph Pooling," 2020.

[19] C. Bodnar *et al.*, "Deep Graph Mapper: Seeing Graphs through the Neural Lens," 2020.

MinCut Pooling [18]

- Select: $\mathbf{S}^\top = \text{MLP}(\mathbf{X})$
- Reduce: $\mathbf{X}' = \mathbf{SX}$
- Connect: $\mathbf{A}' = \mathbf{SAS}^\top$
- MinCut loss: $\mathcal{L}_c = -\frac{Tr(\mathbf{SAS}^\top)}{Tr(\mathbf{SDS}^\top)}$
- Orthogonality loss:
  $\mathcal{L}_o = \left\| \frac{\mathbf{SS}^\top}{\|\mathbf{SS}^\top\|_F} - \frac{\mathbf{I}_K}{\sqrt{K}} \right\|_F$

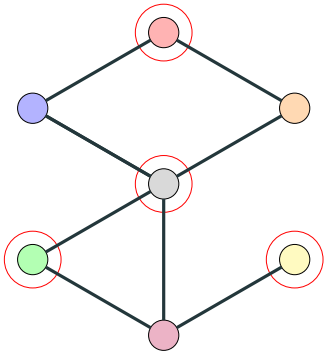


[18] F. M. Bianchi *et al.*, "Spectral Clustering with Graph Neural Networks for Graph Pooling," 2020.

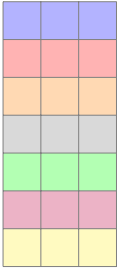
Top-K methods

Problem: computing **S** with neural network is likely to yield a very **dense** matrix.

Can we learn a sparse selection?

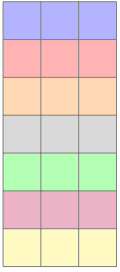


Top-K methods

$X =$ 

Features

Top-K methods

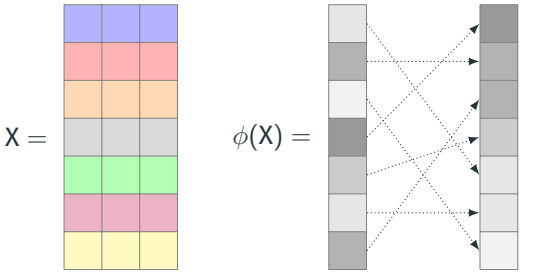
$\mathbf{X} =$ 

Features

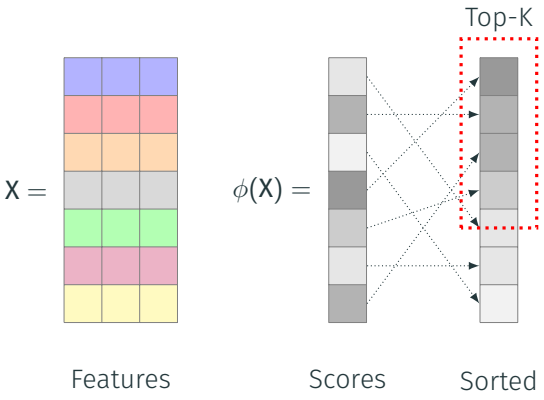
$\phi(\mathbf{X}) =$ 

Scores

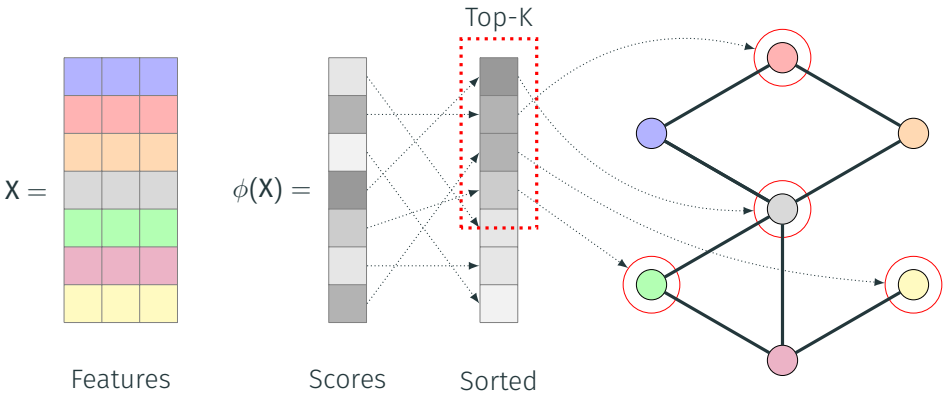
Top-K methods



Top-K methods



Top-K methods



Top-K methods

Different ways of computing the selection indices :

- Select with a simple linear projection $\theta \in \mathbb{R}^{d_x}$ [21];
- Select with a GNN [22];
- Train the selection with a supervised objective (needs ground truth for which nodes to keep) [23].

[21] H. Gao *et al.*, "Graph U-Nets," 2019.

[22] J. Lee *et al.*, "Self-Attention Graph Pooling," 2019.

[23] B. Knyazev *et al.*, "Understanding attention in graph neural networks," 2019.

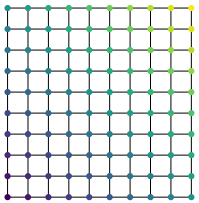
Top-K methods

Reduce: $\mathbf{X}' = \mathbf{X}_i$ Connect: $\mathbf{A}' = \mathbf{A}_{i,i}$

Problems:

- Top-k selection is **non-differentiable**.
  Solved by **gating** (multiplying) the node attributes with the scores.
- Graph is likely to be **disconnected** or simply **cut off** (like in the image on the right).
  Not really solvable...

Original



Top-K

Main properties of pooling operators

- **Dense vs. Sparse**: how many nodes are selected for the supernodes;

Main properties of pooling operators

- **Dense vs. Sparse**: how many nodes are selected for the supernodes;
- **Fixed vs. Adaptive**: how many supernodes does the selection compute;

Main properties of pooling operators

- **Dense vs. Sparse**: how many nodes are selected for the supernodes;
- **Fixed vs. Adaptive**: how many supernodes does the selection compute;
- **Trainable vs. Non-trainable**: learn to pool from data or not;

Global pooling

Global Pooling

In CNNs, after convolutions, we usually **flatten** out the matrix representation to give a vector as input to an MLP:

1	2	3
4	5	6
7	8	9

1	2	3	4	5	6	7	8	9

Global Pooling

In CNNs, after convolutions, we usually **flatten** out the matrix representation to give a vector as input to an MLP:

We may want to do the same operation on graphs, e.g., for *graph classification* tasks.

This operation is called global pooling.

Global Pooling

In CNNs, after convolutions, we usually **flatten** out the matrix representation to give a vector as input to an MLP:

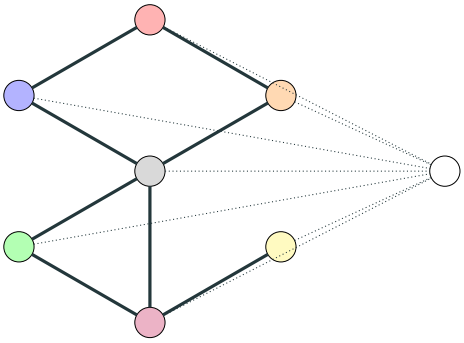
1	2	3
4	5	6
7	8	9

1	2	3	4	5	6	7	8	9

We may want to do the same operation on graphs, e.g., for *graph classification* tasks.

This operation is called global pooling.

Global pooling must be **invariant to permutations** of the nodes:



Global Pooling

Once again, there are many ways to do this:

- Sum, average, product, max;

[24] Y. Li *et al.*, "Gated graph sequence neural networks," 2015.

[25] N. Navarin *et al.*, "Universal readout for graph convolutional neural networks," 2019.

Global Pooling

Once again, there are many ways to do this:

- Sum, average, product, max;
- Weighted sum with attention [24];

[24] Y. Li *et al.*, "Gated graph sequence neural networks," 2015.

[25] N. Navarin *et al.*, "Universal readout for graph convolutional neural networks," 2019.

Global Pooling

Once again, there are many ways to do this:

- Sum, average, product, max;
- Weighted sum with attention [24];
- Sum and then apply a neural network [25];

[24] Y. Li *et al.*, "Gated graph sequence neural networks," 2015.

[25] N. Navarin *et al.*, "Universal readout for graph convolutional neural networks," 2019.

Coding GNNs

GNNs libraries

Spektral is a Python library based on Keras providing a simple but flexible framework for creating graph neural networks (GNNs).

GitHub: danielegrattarola/spektral
Website: graphneural.network

PyG (PyTorch Geometric) is a library built upon PyTorch to easily write and train Graph Neural Networks (GNNs).

GitHub: pyg-team/pytorch_geometric
Website: pyg.org

Demo

In this demo, we will use **PyG** to address the **node classification** task with GNNs.

Introduction to Graph Neural Networks

References i

[1] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs," *IEEE transactions on signal processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2016.

[3] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[4] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" In *International Conference on Learning Representations (ICLR)*, 2019.

[5] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

References ii

[6] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[7] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[8] J. You, R. Ying, and J. Leskovec, "Design space for graph neural networks," *arXiv preprint arXiv:2011.08843*, 2020.

[9] D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi, "Understanding pooling in graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[10] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

References iv

[15] E. Luzhnica, B. Day, and P. Lio, "Clique pooling for graph classification," *International Conference of Learning Representations (ICLR) – Representation Learning on Graphs and Manifolds workshop*, 2019.

[16] E. Noutahi, D. Beani, J. Horwood, and P. Tossou, "Towards interpretable sparse graph representation learning with laplacian pooling," *arXiv preprint arXiv:1905.11577*, 2019.

[17] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning withdifferentiable pooling," *arXiv preprint arXiv:1806.08804*, 2018.

[18] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *Proceedings of the 37th international conference on Machine learning*, ACM, 2020.

[19] C. Bodnar, C. Cangea, and P. Liò, “Deep graph mapper: Seeing graphs through the neural lens,” *arXiv preprint arXiv:2002.03864*, 2020.

[20] G. Singh, F. Mémoli, and G. E. Carlsson, “Topological methods for the analysis of high dimensional data sets and 3d object recognition.,” in *SPBG*, 2007, pp. 91–100.

[21] H. Gao and S. Ji, “Graph u-nets,” *CoRR*, vol. abs/1905.05178, 2019. arXiv: *1905.05178*. [Online]. Available: *http://arxiv.org/abs/1905.05178*.

[22] J. Lee, I. Lee, and J. Kang, “Self-attention graph pooling,” *CoRR*, vol. abs/1904.08082, 2019. arXiv: *1904.08082*. [Online]. Available: *http://arxiv.org/abs/1904.08082*.

[23] B. Knyazev, G. W. Taylor, and M. R. Amer, “Understanding attention in graph neural networks,” *CoRR*, vol. abs/1905.02850, 2019. arXiv: *1905.02850*. [Online]. Available: *http://arxiv.org/abs/1905.02850*.

[24] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[25] N. Navarin, D. Van Tran, and A. Sperduti, "Universal readout for graph convolutional neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–7.

[26] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[27] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

Spatiotemporal Graph Neural Networks

What we are going to see in this lecture

1. **Graphs and time dimension**
   - Examples
   - Taxonomy of temporal graph signals
2. **Spatiotemporal graph signals**
   - Sensor networks
   - Spatiotemporal forecasting
3. **Spatiotemporal GNNs**
   - Spatiotemporal message passing
   - Design paradigms
4. **Challenges**
   - Missing data imputation
   - Virtual sensing
   - Latent graph inference
5. **Coding Spatiotemporal GNNs**
   - `tsl`: a PyTorch library for spatiotemporal data processing

Graphs and the time dimension

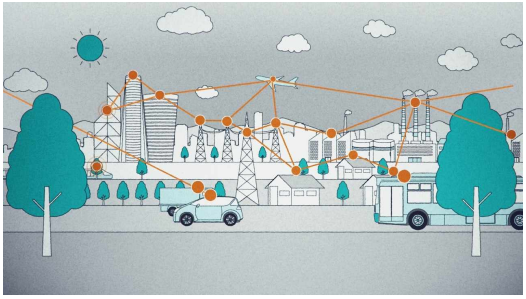
Graphs and time

We saw how attributed graphs are effective in modeling relational information and in accounting for the structure of many physical systems.

However:

- Interactions might happen over time.
- (Cyber-)physical systems are often made of sensor networks that acquire spatiotemporal data streams.

Examples

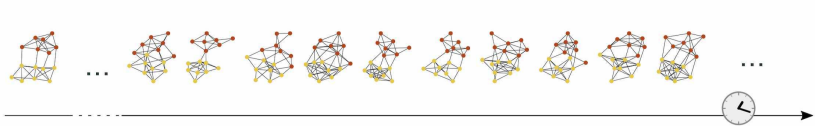


- Interaction networks
  - Social networks
  - Recommender systems
- Sensor networks
  - Traffic networks
  - Smart grids

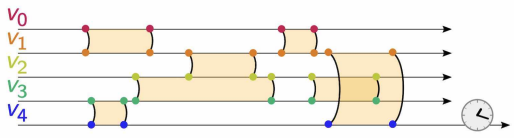
How to model such systems?

Taxonomy of temporal graph signals

There are several settings in which time comes into play when considering graph data.



Graph streams



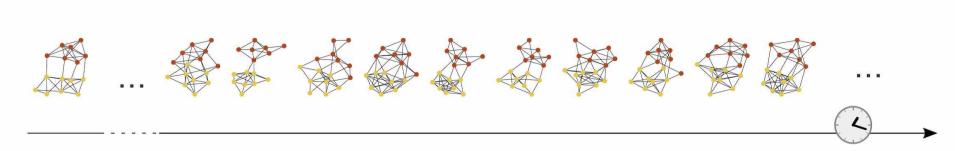
Temporal networks

Spatiotemporal graph signals

Image credits: Daniele Zambon.

Graph streams

We start from the most general setting.

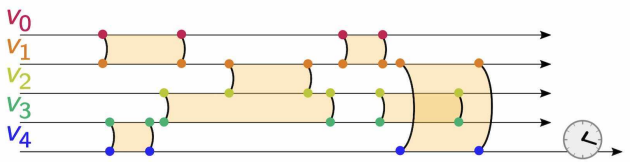


- Graphs are sampled from a stochastic process $\mathcal{G}_t \sim P$.
- Nodes are not identified. $\rightarrow$ No correspondence between nodes at different time steps.
- Arbitrary changes in topology.
- Difficult to track changes, defining statistics is not trivial.

[1] D. Zambon, "Anomaly and Change Detection in Sequences of Graphs", 2022.

Temporal networks

We can look at interactions that happen over time as sequences of **relational events**.



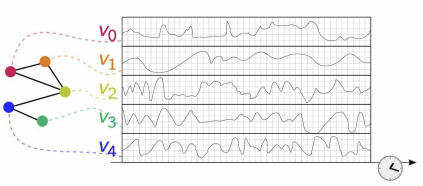
- Temporal networks are used to model systems where relationships and node attributes evolve over time.
- Event-based paradigm: target data are sequences of interactions among nodes.
- Powerful paradigm to model social/interaction networks and build recommender systems.

[2] S. M. Kazemi *et al.*, "Representation learning for dynamic graphs: A survey", 2020.

Spatiotemporal graph signals

Spatiotemporal graphs capture the setting typical of sensor networks.

- An approach to model multivariate time series coming from multiple sources.
- Each node (sensor) is associated with a time series (possibly with multiple channels).
- Edges describe functional dependencies among sensors.
  - E.g.: causality, physical constraints, etc.
- The underlying graph, i.e., the sensors and their relations, can change over time.



We will focus on this setting

Spatiotemporal graph signals

Sensor networks

We refer to **Sensor Networks** as systems where

- A set of nodes (sensors) collects observations with regular frequency.
- Each node is identified (allowing us to talk about time series).
- Nodes constitute what we refer to as the spatial dimension.
- Sensors are related according to some measure of similarity.

Examples: traffic networks, air quality monitoring systems, smart grids, etc.

We model a set of time series coming from a sensor network as a sequence of graph signals.

[3] A. Cini *et al.*, "Taming Local Effects in Graph-based Spatiotemporal Forecasting", 2023.

Graph signal

We consider a **graph signal** at time step t as a tuple $\mathcal{G}_t = \langle \boldsymbol{A}_t, \boldsymbol{X}_t, \boldsymbol{U}_t, \boldsymbol{E}_t \rangle$ where

- $\boldsymbol{A}_t \in \mathbb{R}^{N_t \times N_t}$ is a weighted adjacency matrix, with N_t being the number of nodes;
- $\boldsymbol{X}_t \in \mathbb{R}^{N_t \times d_x}$ is the node-attribute matrix;
  - $\boldsymbol{x}_t^i$ (the i-th row of $\boldsymbol{X}_t$) is the d_x-dimensional attribute vector associated with the i-th node;
- $\boldsymbol{U}_t \in \mathbb{R}^{N_t \times d_u}$ are exogenous variables (e.g., weather forecasts, datetime information);
- $\boldsymbol{E}_t \in \mathbb{R}^{E_t \times d_e}$ is an edge-attribute matrix, with E_t being the number of edges;

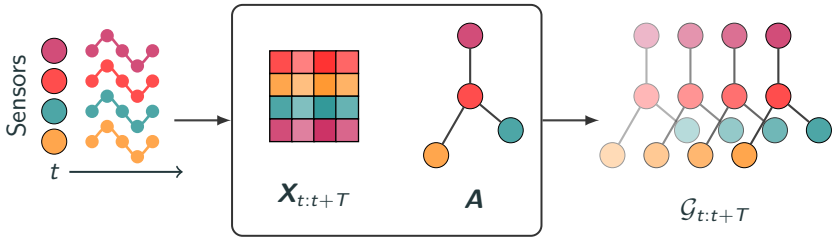
We use the notation $\boldsymbol{X}_{t:t+T}$ to indicate the sequence of matrices $\{\boldsymbol{X}_t, \dots, \boldsymbol{X}_{t+T}\}$.

[3] A. Cini *et al.*, "Taming Local Effects in Graph-based Spatiotemporal Forecasting", 2023.

Spatiotemporal graph signal

We call **spatiotemporal graph signal** the sequence of graph signals $\mathcal{G}_{t:t+T} = \{\mathcal{G}_t, \ldots, \mathcal{G}_{t+T}\}$ modeling a multivariate time series $\boldsymbol{X}_{t:t+T}$ with covariates $\boldsymbol{U}_{t:t+T}$ and additional relational information $\boldsymbol{A}_{t:t+T}$ and $\boldsymbol{E}_{t:t+T}$.

We focus on settings where the graph topology is constant over time, i.e., $\boldsymbol{A}_t = \boldsymbol{A}$ and $N_t = N$.



The role of A

Let us stop and think about the meaning of $\boldsymbol{A}$.

- In spatiotemporal graphs, we can interpret the adjacency matrix as representing mutual constraints on the time evolution of connected time series.
- We might consider edges as functional dependencies among observed values at different time steps.

From this perspective, one could also look at spatiotemporal graph processing as a regularization of standard neural time series processing methods.

Processing these data with a fully-connected net would require much more parameters.

– Think of convolutional neural networks processing subsequent frames in a video.
  $\rightarrow$ **Making use of spatial inductive biases is critical!**

Spatiotemporal forecasting

Viewing spatiotemporal graphs as time series with relational information enables their adoption in sequence-processing tasks (e.g., **forecasting** and imputation).

Node-level spatiotemporal forecasting

Given **a window** W of past observations, the **node-level spatiotemporal forecasting** problem consists in predicting **the next** H observations at each sensor:

$$\boldsymbol{x}^i_{t:t+H} \sim p(\boldsymbol{x}^i_{t:t+H} | \mathcal{G}_{t-W:t}) \qquad \forall i = 1, \dots, N_t$$

For simplicity, we consider only **point forecasts**, e.g., we estimate $\hat{\boldsymbol{x}}^i_{t:t+H}$ s.t.

$$\hat{\boldsymbol{x}}^i_{t:t+H} \approx \mathbb{E}\left[p(\boldsymbol{x}^i_{t:t+H} | \mathcal{G}_{t-W:t})\right]$$

Assumptions

Since we are dealing with time series we have to make **some assumptions** on the underlying data-generating process.

- We assume that the underlying process is **stationary**.
  - $\rightarrow$ Model parameters are time independent.
- We also assume the graph to be **homogeneous**.
  - $\rightarrow$ All sensors are of the same type.
  - This hypothesis can be easily relaxed.
- We assume to have a **known graph**.
  - We will see how to obtain a graph when it is not given in the second part.

Spatiotemporal Graph Neural Networks

Learning from Time and Space

We consider families of parametric models f_θ for node-level forecasting:

$$\hat{\boldsymbol{x}}^i_{t:t+H} = f_\theta(\mathcal{G}_{t-W:t}).$$

More precisely, we focus on those families where f_θ is a neural network.

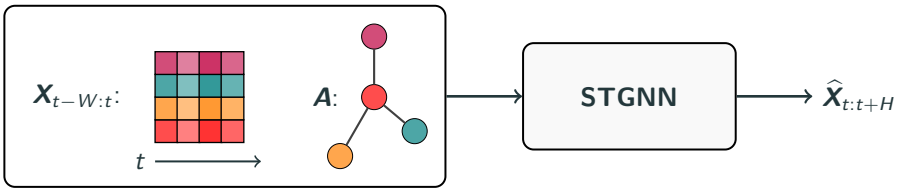
We now know how to use neural networks for processing:

- the **temporal** dimension, with RNNs, TCNs, and Transformers;
- the **spatial** dimension, with CNNs and **GNNs**.

What about processing the **temporal and spatial dimensions jointly**?

Spatiotemporal Graph Neural Networks

We call Spatiotemporal Graph Neural Network (STGNN) a neural network exploiting both temporal and spatial relations of the input spatiotemporal graph signals.



We consider families of models that exploit message passing to process the spatial dimension, by leveraging on some graph shift operator $\widetilde{\boldsymbol{A}} = f(\boldsymbol{A})$.

Spatiotemporal message passing

A general scheme for spatiotemporal message-passing networks:

$$\boldsymbol{z}^i_{t-W:t} = \gamma\left(\boldsymbol{x}^i_{t-W:t}, \underset{j\in\mathcal{N}(i)}{\texttt{Aggr}}\left\{\phi\left(\boldsymbol{x}^i_{t-W:t}, \boldsymbol{x}^j_{t-W:t}, \boldsymbol{e}^{ij}_{t-W:t}\right)\right\}\right)$$

We already saw this:

- ϕ is the **message function**.
- $\texttt{Aggr}$ is the **aggregation function**.
- γ is the **update function**.

The difference here is that instead of vectors we have **sequences** associated with node features.
$\rightarrow$ We must use operators that work on sequences!

[3] A. Cini *et al.*, "Taming Local Effects in Graph-based Spatiotemporal Forecasting", 2023.

Design paradigms for STGNNs

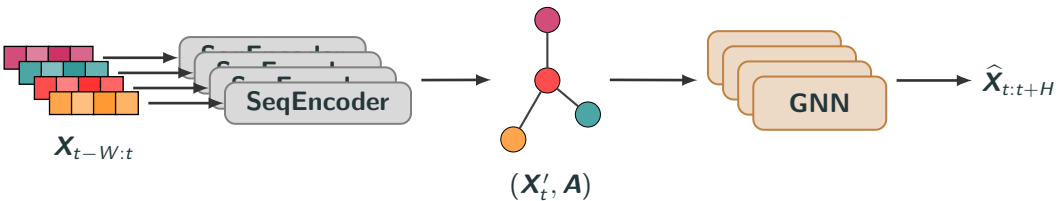
There exist different design paradigms on how to integrate temporal and spatial processing in a single architecture:

- **Time-then-Space**
  Embed each time series in a vector, which is then propagated over the graph.
- **Space-then-Time**
  Propagate nodes features at first and then process the resulting time series.
- **Time-and-Space**
  Temporal and spatial processing are integrated inside the same architecture's module.
- **Product graph**
  The sequence of graphs is transformed into a single graph, then processed with a GNN.

Time-then-Space

A straightforward approach to process spatiotemporal graphs is simply to:

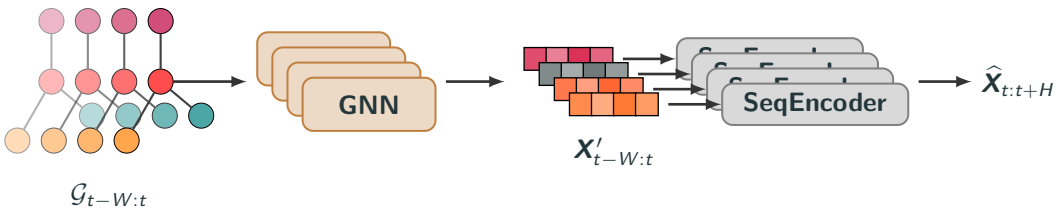
1. **Embed** each node-level time series in a vector.
2. Use (a stack of) any of the **graph convolutional layers** we have seen so far.



Space-then-Time

Or conversely, we can switch the two processing steps:

1. Propagate nodes features using (a stack of) any **graph convolutional layers**.
2. Process updated node-level time series with any **sequence-processing architecture**.



Pros & Cons

Pros:
+ Very simple paradigm, easy and efficient to implement.
+ We can reuse operators we already know.

Cons:
– Do not exploit space-time dependencies (if needed).
– Time-then-Space models struggle to handle changes in topology within the input window.

Time-and-Space

The idea is to use graph convolutional layers to implement (part of) neural operators for sequential data...

...or, conversely, implement message-passing networks by using temporal operators.

We look at 2 different strategies:

- Interleaved **spatial and temporal convolutional filters**.
- **Graph-based recurrent neural networks**.

Spatiotemporal Graph Convolutional Networks (i)

We can build deep spatiotemporal convolutional neural networks by **alternating spatial and temporal convolutional filters**.

The main idea:

- Compute intermediate representations by using a node-wise temporal convolutional layer:

$$\boldsymbol{x}'^{i}_{t-W:t} = \xi\left(\boldsymbol{\Theta}_1 \star_{\mathcal{T}} \boldsymbol{x}^{i}_{t-w:t}\right)$$

  where $\star_{\mathcal{T}}$ indicates the temporal convolution operator and ξ is an activation function.
- Then, compute the updated representation by using a time-wise graph convolution:

$$\boldsymbol{Z}_t = \sigma\left(\widetilde{\boldsymbol{A}}_t \boldsymbol{X}'_t \boldsymbol{\Theta}_2\right)$$

[4] B. Yu *et al.*, "Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting", 2018.

Spatiotemporal Graph Convolutional Networks (ii)

Putting it all together (with a slight abuse of notation) we get

$$\boldsymbol{Z}_{t-W:t} = \sigma\left(\widetilde{\boldsymbol{A}}_t \xi\left(\boldsymbol{\Theta}_1 \star_\tau \boldsymbol{X}_{t-W:t}\right)\boldsymbol{\Theta}_2\right)$$

Stacking a sequence of layers, we **increase** both temporal and spatial **receptive fields**.

Clearly one can combine **any flavor of graph and temporal convolutions** to incorporate exogenous variables, edge attributes, and so on.

It is also possible to substitute convolutional filters with self-attention [5].

[5] C. Zheng *et al.*, "Gman: A graph multi-attention network for traffic prediction", 2020.

Example: Temporal Graph Convolution

Using the spatiotemporal message-passing framework, we can write a temporal graph convolution as

$$\boldsymbol{z}^i_{t-W:t} = \sigma\left(\boldsymbol{\Theta}_1 \star_\tau \left[\boldsymbol{x}^i_{t-W:t} \,\middle\|\, \underset{j\in\mathcal{N}(i)}{\texttt{Aggr}}\left(\boldsymbol{\Theta}_2 \star_\tau \left[\boldsymbol{x}^i_{t-W:t} \| \boldsymbol{x}^j_{t-W:t} \| \boldsymbol{e}^{ij}_{t-W:t}\right]\right)\right]\right),$$

or, more simply,

$$\boldsymbol{z}^i_{t-W:t} = \text{TCN}_1\left(\boldsymbol{x}^i_{t-W:t}, \underset{j\in\mathcal{N}(i)}{\texttt{Aggr}}\ TCN_2\left(\boldsymbol{x}^i_{t-W:t}, \boldsymbol{x}^j_{t-W:t}, \boldsymbol{e}^{ij}_{t-W:t}\right)\right).$$

Other examples

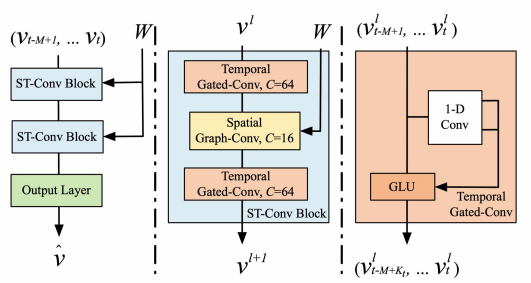


Figure 1: STGCN [4]

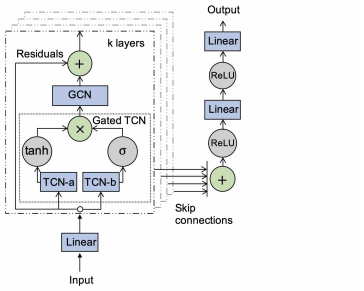


Figure 2: Graph Wavenet [6]

[4] B. Yu *et al.*, "Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting", 2018.

[6] Z. Wu *et al.*, "Graph wavenet for deep spatial-temporal graph modeling", 2019.

Graph Convolutional Recurrent Neural Networks (i)

Let us now consider a standard GRU [7] cell.

$$\boldsymbol{r}_t^i = \sigma\left(\boldsymbol{\Theta}_r\left[\boldsymbol{x}_t^i || \boldsymbol{h}_{t-1}^i\right] + \boldsymbol{b}_r\right) \tag{1}$$

$$\boldsymbol{u}_t^i = \sigma\left(\boldsymbol{\Theta}_u\left[\boldsymbol{x}_t^i || \boldsymbol{h}_{t-1}^i\right] + \boldsymbol{b}_u\right) \tag{2}$$

$$\boldsymbol{c}_t^i = \tanh\left(\boldsymbol{\Theta}_c\left[\boldsymbol{x}_t^i || \boldsymbol{r}_t^i \odot \boldsymbol{h}_{t-1}^i\right] + \boldsymbol{b}_c\right) \tag{3}$$

$$\boldsymbol{h}_t^i = \left(1 - \boldsymbol{u}_t^i\right) \odot \boldsymbol{c}_t^i + \boldsymbol{u}_t^i \odot \boldsymbol{h}_{t-1}^i \tag{4}$$

Note that here time series would be processed **independently** for each node (or alternatively as a single multivariate TS).

Any idea on how to integrate graph convolutions?

[7] J. Chung *et al.*, "Empirical evaluation of gated recurrent neural networks on sequence modeling", 2014.

Graph Convolutional Recurrent Neural Networks (ii)

We simply implement the **gates** by using **graph convolutions**:

$$\boldsymbol{R}_t = \sigma\left(\widetilde{\boldsymbol{A}}_t\left[\boldsymbol{X}_t||\boldsymbol{H}_{t-1}\right]\boldsymbol{\Theta}_r + \boldsymbol{b}_r\right) \tag{5}$$

$$\boldsymbol{U}_t = \sigma\left(\widetilde{\boldsymbol{A}}_t\left[\boldsymbol{X}_t||\boldsymbol{H}_{t-1}\right]\boldsymbol{\Theta}_u + \boldsymbol{b}_u\right) \tag{6}$$

$$\boldsymbol{C}_t = \tanh\left(\widetilde{\boldsymbol{A}}_t\left[\boldsymbol{X}_t||\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1}\right]\boldsymbol{\Theta}_c + \boldsymbol{b}_c\right) \tag{7}$$

$$\boldsymbol{H}_t = (1 - \boldsymbol{U}_t) \odot \boldsymbol{C}_t + \boldsymbol{U}_t \odot \boldsymbol{H}_{t-1} \tag{8}$$

Introduced in [8] and later popularized in the traffic forecasting context [9].

[8] Y. Seo *et al.*, "Structured sequence modeling with graph convolutional recurrent networks", 2018.
[9] Y. Li *et al.*, "Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting", 2018.

Graph Convolutional Recurrent Neural Networks (iii)

Or, if you prefer a different – more general – notation:

$$\boldsymbol{R}_t = \sigma\left(\text{GNN}_r\left(\mathcal{G}_t, \boldsymbol{H}_{t-1}\right)\right) \tag{9}$$

$$\boldsymbol{U}_t = \sigma\left(\text{GNN}_u\left(\mathcal{G}_t, \boldsymbol{H}_{t-1}\right)\right) \tag{10}$$

$$\boldsymbol{C}_t = \tanh\left(\text{GNN}_c\left(\mathcal{G}_t, \boldsymbol{R}_t \odot \boldsymbol{H}_{t-1}\right)\right) \tag{11}$$

$$\boldsymbol{H}_t = (1 - \boldsymbol{U}_t) \odot \boldsymbol{C}_t + \boldsymbol{U}_t \odot \boldsymbol{H}_{t-1} \tag{12}$$

Again, the gates can be implemented by using any GNN, a popular choice in the literature is **diffusion convolution** [9], where $\widetilde{\boldsymbol{A}}_t = \boldsymbol{D}^{-1}\boldsymbol{A}_t$ (random-walk matrix). For **directed** graphs:

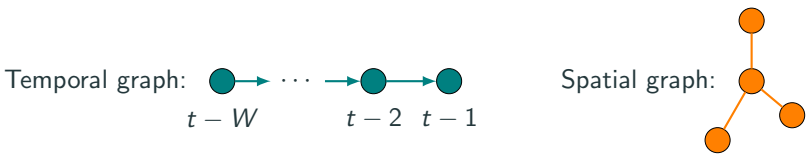
$$\boldsymbol{Z}_t = \sum_{k=0}^{K} \left(\boldsymbol{D}_{t,out}^{-1}\boldsymbol{A}_t\right)^k \boldsymbol{X}_t\boldsymbol{\Theta}_1^k + \left(\boldsymbol{D}_{t,in}^{-1}\boldsymbol{A}_t^\top\right)^k \boldsymbol{X}_t\boldsymbol{\Theta}_2^k \tag{13}$$

[9] Y. Li *et al.*, "Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting", 2018.

Product graph

Another possibility is to consider the sequence of spatiotemporal graph signals $\mathcal{G}_{t-W:t}$ as a **single graph signal** $\mathcal{S}_t$ over a new spatiotemporal graph.

This graph, called **product graph**, is a combination of the temporal and spatial graphs.

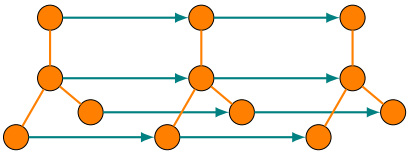


We can process the resulting product graph $\mathcal{S}_t$ with a graph neural network.

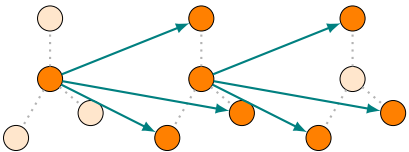
How can we build such a graph?

Product graph: combining rules

- **Cartesian product graph**
  Spatial graphs are kept and each node is connected to itself in the previous time instant.



- **Kronecker product graph**
  Each node is connected **only** to its neighbors in the previous time instant.

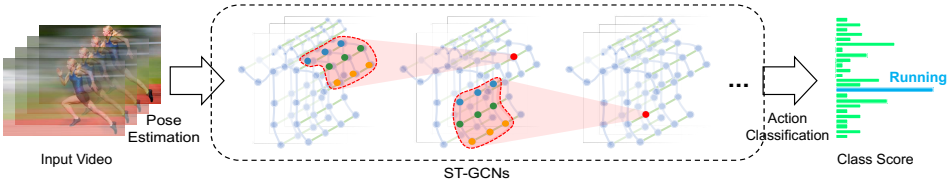


Product graph models

Of course, spatial and temporal edges can (and should) be **treated differently** in the processing.

A possibility is to represent the product graph as a **heterogeneous graph**, assigning a different class to spatial and temporal edges.

Some approaches in the literature process spatiotemporal data in this fashion, e.g. [10]:



[10] S. Yan *et al.*, "Spatial temporal graph convolutional networks for skeleton-based action recognition", 2018.

Challenges

What if ...?

STGNNs are a powerful tool to process a set of time series with graph-side information.

However, we are implicitly assuming that:

- input data are **tabular**, i.e., we have a (valid) value for each node and time step;
- the underlying graph is **given**.

What if one of these assumptions **does not hold**?

The problem of missing data

So far, we assumed to deal with **complete sequences**, i.e., to have valid observations associated with each node (sensor) and time step.

However, in real-world sensor networks this is often **not the case**.

Collected time series are affected by missing data due to faults of different nature (e.g., readout failures or communication flaws).

If we wish to use any of the methods presented before, we need a way to impute, i.e., reconstruct, missing observations.

Multivariate time series imputation

The problem of filling missing values in a (multivariate) sequence of data is often referred to as **multivariate time series imputation** (MTSI).

Let $\boldsymbol{X}_{t:t+T} = \{\boldsymbol{X}_t, \ldots, \boldsymbol{X}_{t+T}\}$ be a multivariate time series with missing values. We group all valid observations into set $\mathcal{X}_{t:t+T} = \{\boldsymbol{x}_t^i \mid \boldsymbol{x}_t^i \in \boldsymbol{X}_{t:t+T}, \boldsymbol{x}_t^i \text{ is valid}\}$. Then, we want to estimate **the missing observations**, i.e.,

$$\boldsymbol{x}_t^i \sim p(\boldsymbol{x}_t^i \mid \mathcal{X}_{t:t+T}) \qquad \forall i, t \text{ such that } \boldsymbol{x}_t^i \notin \mathcal{X}_{t:t+T}$$

MTSI methods

In principle, any forecasting method can be used for imputation, but we would not expolit **future observations** we have.

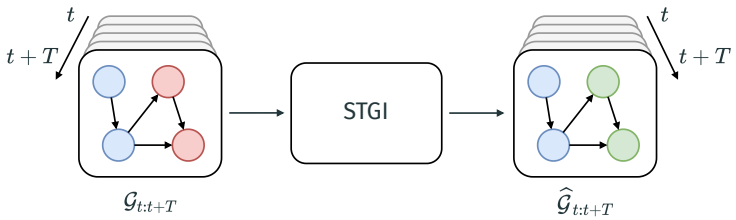
The common deep learning approach consists in using **autoregressive models** for sequential data (e.g., RNNs, TCNs).

Drawbacks:

- relational information (often strong in sensor networks) not taken into account;
- hard to capture **nonlinear space-time dependencies**.

Spatiotemporal graph imputation

Representing the input multivariate time series as a spatiotemporal graph signal, we can treat the MTSI problem as a **spatiotemporal graph imputation** (STGI) problem.



We embed relational constraints – besides temporal ones – **explicitly** into the data processing:

$$\underbrace{\boldsymbol{x}_t^i \sim p(\boldsymbol{x}_t^i \,|\, \mathcal{X}_{t:t+T})}_{\text{MTSI}} \quad \mapsto \quad \underbrace{\boldsymbol{x}_t^i \sim p(\boldsymbol{x}_t^i \,|\, \mathcal{X}_{t:t+T}, \boldsymbol{A}_{t:t+T})}_{\text{STGI}}$$

Spatiotemporal imputation methods

There are several methods that consider spatial information during processing (e.g., [11]).

However:

- Most of them account only for linear dependencies...
- ...or require prior physical knowledge on the processes involved.
- Often flexibility is limited.

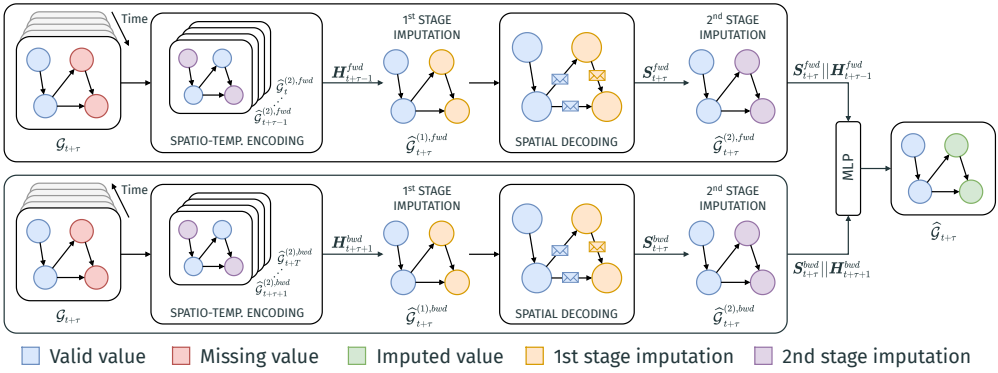
Graph-based methods – in particular, Graph Deep Learning – are appealing in this context:

+ High flexibility, given by working with graphs.
- Designing models that exploit all the available – useful – information is not trivial.

[11] X. Yi *et al.*, "ST-MVL: Filling Missing Values in Geo-sensory Time Series Data", 2016.

Graph Recurrent Imputation Network (GRIN)

GRIN is a graph-based, bidirectional, recurrent neural network which aims to reconstruct the input sequence by leveraging on both the temporal and spatial dimensions, jointly.

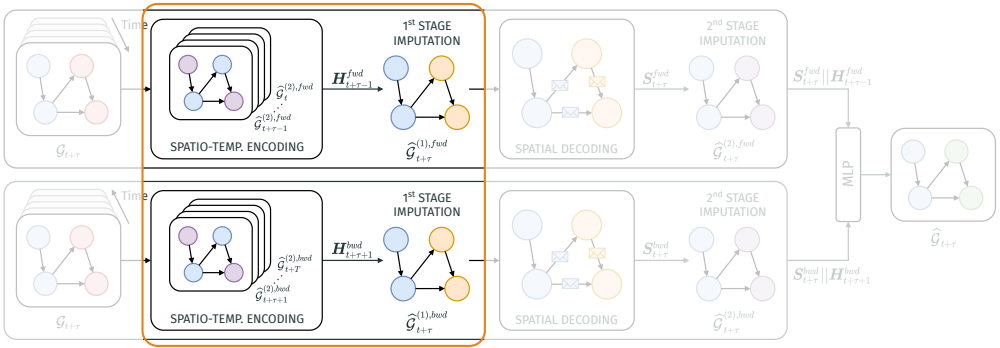


[12] A. Cini *et al.*, "Filling the G_ap_s: Multivariate Time Series Imputation by Graph Neural Networks", 2022.

Graph Recurrent Imputation Network (GRIN)

1 Feed a recurrent GNN with $\widehat{\mathcal{G}}^{(2)}_{t+\tau-1}$ and obtain representation $\boldsymbol{H}_{t+\tau-1}$.

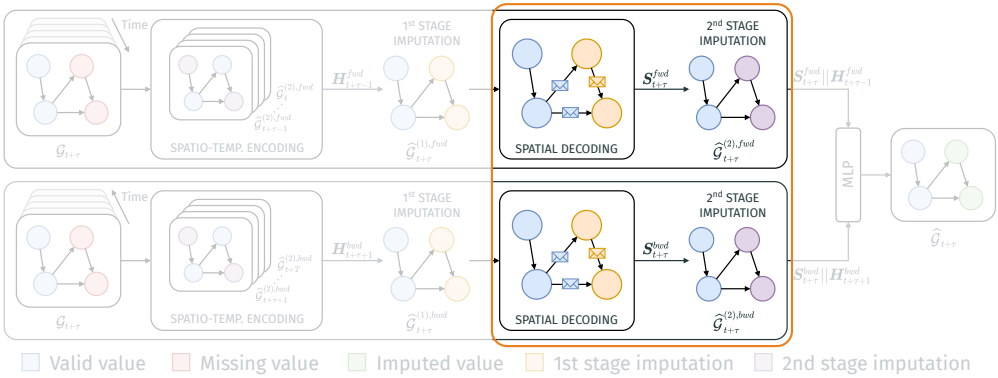
2 Impute missing features as one-step-ahead predictions from $\boldsymbol{H}_{t+\tau-1}$. $\mapsto$ $\widehat{\mathcal{G}}^{(1)}_{t+\tau}$



Graph Recurrent Imputation Network (GRIN)

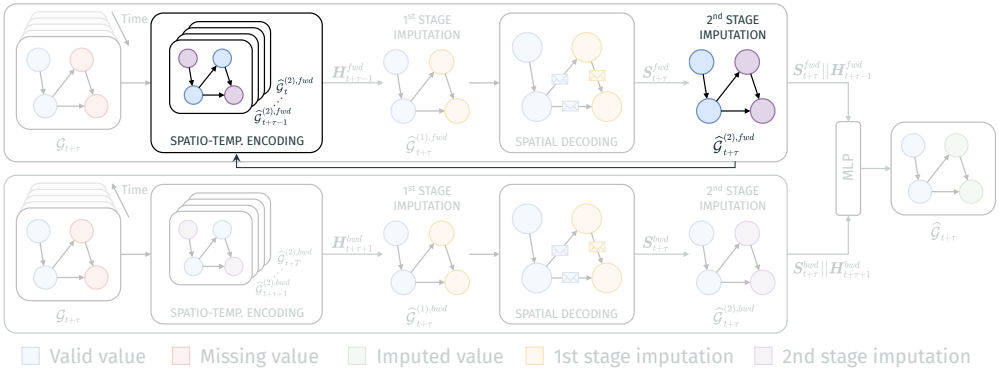
3 Exploit relationships between nodes at time $t + \tau$ through a GNN and obtain $\boldsymbol{S}_{t+\tau}$.

4 Refine imputations using $\boldsymbol{S}_{t+\tau}$. $\mapsto$ $\widehat{\mathcal{G}}^{(2)}_{t+\tau}$



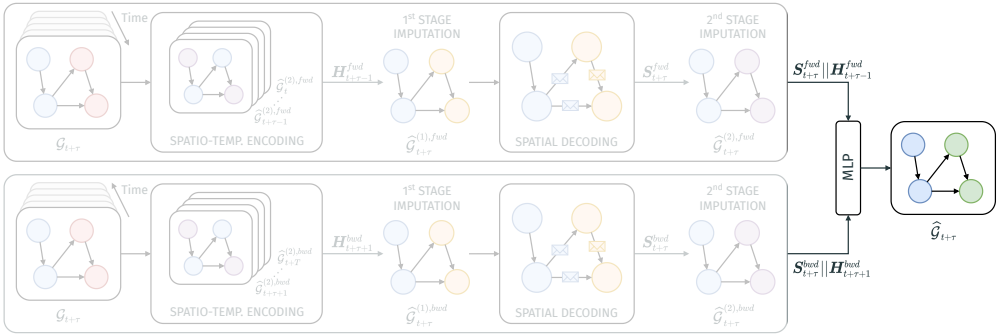
Graph Recurrent Imputation Network (GRIN)

The 2nd stage imputation $\widehat{\mathcal{G}}^{(2)}_{t+\tau}$ is then fed back to the recurrent GNN to update the state, obtaining representation $\boldsymbol{H}_{t+\tau}$.



Graph Recurrent Imputation Network (GRIN)

Obtain final imputations by combining (with an MLP) the representations extracted by processing the sequence in both forward and backward directions.



Beyond imputation

Graph-based imputation methods estimates missing values at an **existing** node by using available information at **neighboring** nodes.

Thus, we are interested in estimating missing values in sequences for which we have **some** observations (at least).

Question
Can we use the same approach to **infer** observations of virtual sensors, i.e., fictitious nodes **not** associated with an existing sensor?

This problem is also referred to as **kriging**.

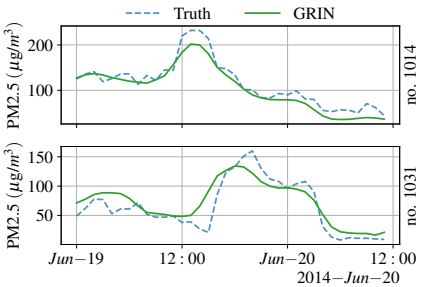
Virtual sensing

Idea

Simulate the presence of a sensor by adding a node with **no data**, then let the model infer the corresponding time series.

Clearly, several assumptions are needed

- high-degree of homogeneity of sensors,
- capability to reconstruct from observations at neighboring sensors,
- and many more.

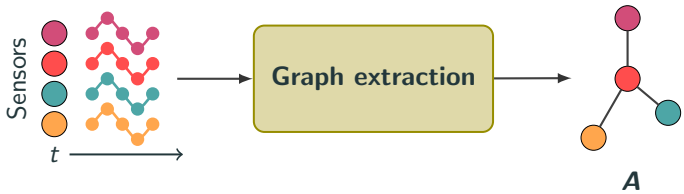


How to obtain A

At the beginning, we made the assumption that the underlying graph $\boldsymbol{A}$ is given.

However, in most cases we **do not know it** or it is not optimal for spatial processing.

In these cases, we want to obtain a **new graph** somehow, e.g., by using the data we have.



Time-series similarity measures

A simple approach consists in computing **pairwise similarity scores** for each node pairs.

In principle, any **time-series similarity measure** can be used, e.g.:

- **Pearson's correlation**

$$r_{ji} = \frac{\sum_{k=0}^{T} \left(\boldsymbol{x}_{t+k}^{j} - \overline{\boldsymbol{x}}^{j} \right) \left(\boldsymbol{x}_{t+k}^{i} - \overline{\boldsymbol{x}}^{i} \right)}{\sqrt{\sum_{k=0}^{T} \left(\boldsymbol{x}_{t+k}^{j} - \overline{\boldsymbol{x}}^{j} \right)^2 \sum_{k=0}^{T} \left(\boldsymbol{x}_{t+k}^{i} - \overline{\boldsymbol{x}}^{i} \right)^2}}$$

- **Granger causality**
  Test the hypothesis that adding node j as regressor into model $\hat{\boldsymbol{x}}_{t:t+H}^{i} = f(\boldsymbol{x}_{t-W:t}^{i})$, i.e., $\hat{\boldsymbol{x}}_{t:t+H}^{i} = g(\boldsymbol{x}_{t-W:t}^{i}, \boldsymbol{x}_{t-W:t}^{j})$, increases forecasting accuracy.

Latent graph inference

More advanced methods propose instead to learn the graph used to propagate information **end-to-end** with the model's parameters.

This problem is referred to as **graph learning** or **latent graph inference**.

We consider two different approaches:

- learning an adjacency matrix $\widehat{\boldsymbol{A}}_\theta \in \mathbb{R}^{N \times N}$;
- learning the probability distribution p_θ generating $\widehat{\boldsymbol{A}}$.

An orthogonal classification can be made on whether the obtained $\widehat{\boldsymbol{A}}$ is **dense** or **sparse**.

Factorization methods

Several approaches propose to factorize the target adjacency matrix $\widehat{\boldsymbol{A}}$ into two matrices:

- the **sender nodes** embeddings $\boldsymbol{S} \in \mathbb{R}^{N \times d_a}$;
- the **receiver nodes** embeddings $\boldsymbol{R} \in \mathbb{R}^{N \times d_a}$.

These matrices can be learned as free parameters or be the outcome of a complex model.

The graph is then obtained as

$$\widehat{\boldsymbol{A}} = \sigma\left(\boldsymbol{R}\boldsymbol{S}^\top\right).$$

Drawbacks:
A score is computed for every pair of nodes ($\mathcal{O}(N^2)$), leading also to **very dense graphs**!

[6] Z. Wu *et al.*, "Graph wavenet for deep spatial-temporal graph modeling", 2019.

Probabilistic methods

In this context, probabilistic methods aim at learning a parametric distribution p_θ such that

$$\operatorname{argmin} \mathbb{E}_{\widehat{\boldsymbol{A}} \sim p_\theta} \left[\text{Loss} \left(\widehat{\boldsymbol{X}}_{t:t+H}, \boldsymbol{X}_{t:t+H} \right) \right].$$

We can use the method we've just seen to model the distribution parameters instead, i.e.,

$$\widehat{\boldsymbol{A}} \sim p_\theta = \text{Bernoulli} \left(\sigma \left(\boldsymbol{R}\boldsymbol{S}^\top \right) \right).$$

This enables **sparsification** of otherwise dense learned adjacency matrices.

Drawbacks:
Estimating the gradient w.r.t the distributional parameters is challenging.

Gradient estimation

A possible solution is to reparametrize $\widehat{\boldsymbol{A}} \sim p_\theta$ as $\widehat{\boldsymbol{A}} = g(\varepsilon, \theta)$, decoupling parameters θ from the random component ε.

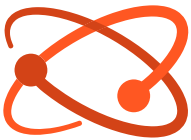
While being effective and easy to implement, **reparametrization tricks** of this kind usually lead to $\mathcal{O}(N^2)$ complexity during back-propagation, even for sparse $\widehat{\boldsymbol{A}}$.

Leveraging on score-function (SF) gradient estimators, instead, allows us to maintain the advantages of **sparse sampled graphs** while leading to accuracy improvements [13].

[13] A. Cini *et al.*, "Sparse Graph Learning for Spatiotemporal Time Series", 2022.
[14] T. Kipf *et al.*, "Neural relational inference for interacting systems", 2018.

Coding Spatiotemporal GNNs

tsl: PyTorch Spatiotemporal Library



tsl (Torch Spatiotemporal) is a python library built upon PyTorch and PyG to accelerate research on neural spatiotemporal data processing methods, with a focus on **Graph Neural Networks**.

Spatiotemporal Graph Neural Networks with tsl

Open in Colab

Summary

In this lecture, we saw several methods to deal with inference problems on sets of time series by exploiting relational inductive biases.

Some takeaway points:

- In processing spatiotemporal data you have to deal with several subtleties to build a good model.
- Graph deep learning models are very flexible and can be extended to work in several different setting.
  - Even if it is not always trivial.
- When spatial dependencies exist use them, they will help a lot!

Questions?

References i

[1] D. Zambon, "Anomaly and change detection in sequences of graphs," Ph.D. dissertation, Università della Svizzera italiana, 2022.

[2] S. M. Kazemi, R. Goel, K. Jain, *et al.*, "Representation learning for dynamic graphs: A survey," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 2648–2720, 2020.

[3] A. Cini, I. Marisca, D. Zambon, and C. Alippi, "Taming local effects in graph-based spatiotemporal forecasting," *arXiv preprint arXiv:2302.04071*, 2023.

[4] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 3634–3640.

[5] C. Zheng, X. Fan, C. Wang, and J. Qi, "Gman: A graph multi-attention network for traffic prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 1234–1241.

References ii

[6] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019, pp. 1907–1913.

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[8] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *International Conference on Neural Information Processing*, Springer, 2018, pp. 362–373.

[9] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *International Conference on Learning Representations*, 2018.

References iii

[10] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[11] X. Yi, Y. Zheng, J. Zhang, and T. Li, "St-mvl: Filling missing values in geo-sensory time series data," in *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 2016.

[12] A. Cini, I. Marisca, and C. Alippi, "Filling the g_ap_s: Multivariate time series imputation by graph neural networks," in *International Conference on Learning Representations*, 2022. [Online]. Available: `https://openreview.net/forum?id=kOu3-S3wJ7`.

[13] A. Cini, D. Zambon, and C. Alippi, "Sparse graph learning for spatiotemporal time series," *arXiv preprint arXiv:2205.13492*, 2022.

References iv

[14] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural relational inference for interacting systems," in *International conference on machine learning*, PMLR, 2018, pp. 2688–2697.