Graph Deep Learning

SP 2024

Prof. Cesare Alippi

Università della Svizzera italiana

The people you will see

Cesare **Alippi**
(me)

Tommaso **Marzi**
(TA)

Gabriele **Dominici**
(TA)

Teaching material

- **Slides** provided by the lecturers and TAs (iCorsi3 platform)
- **Selected papers** given to the students
- **Prerequisites**:
  - Machine Learning
  - Deep learning lab

Course evaluation

- One assignment (demo+report) and its presentation to the class (90% of the score)
- A quiz exam at the end of the course (10%)

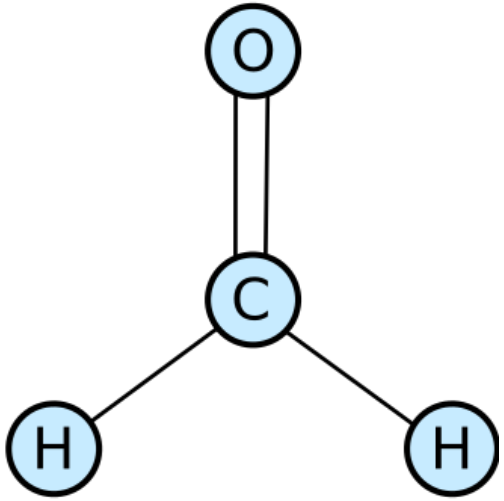
Program and time schedule

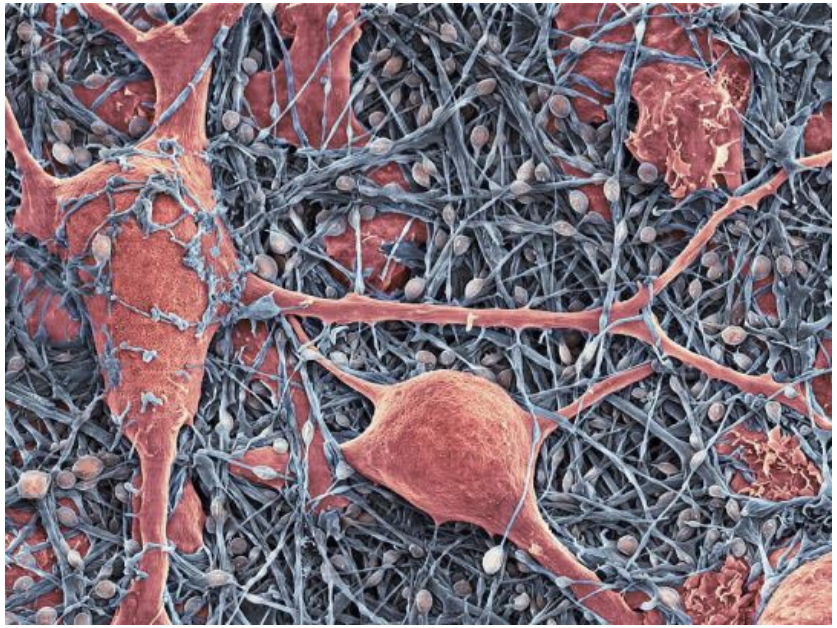
Date	Speaker	Topic
23 February	We	Introduction to graphs
1 March	We	Graph convolution and graph pooling
8 March	We	Spatiotemporal graphs
15 March	We+	Learning NSE
22 March	Michael Bronstein (U.Oxford)	Erlangen program of ML; Continuous models for GNNs
17 May	Students	Students presentations
24 May	Students	Students presentations

- On Fridays, Room D1.14
  - First slot: 8:50- 10:20(+)
  - Second slot: 10:40- 12:10(+)

Why graphs

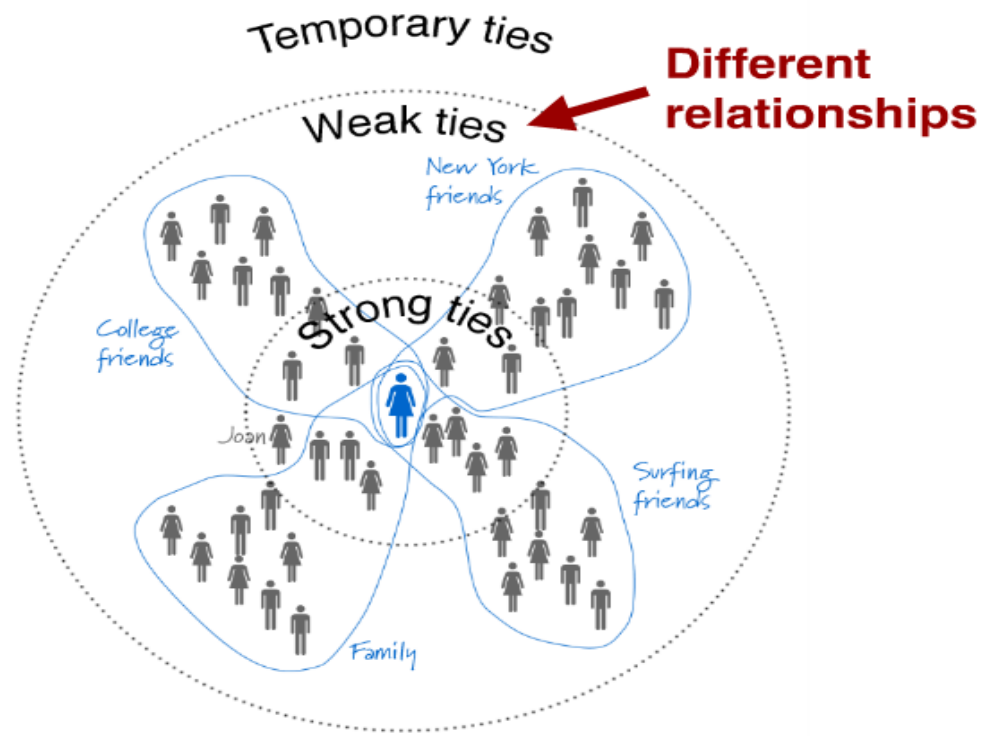
In many applications graphs come naturally





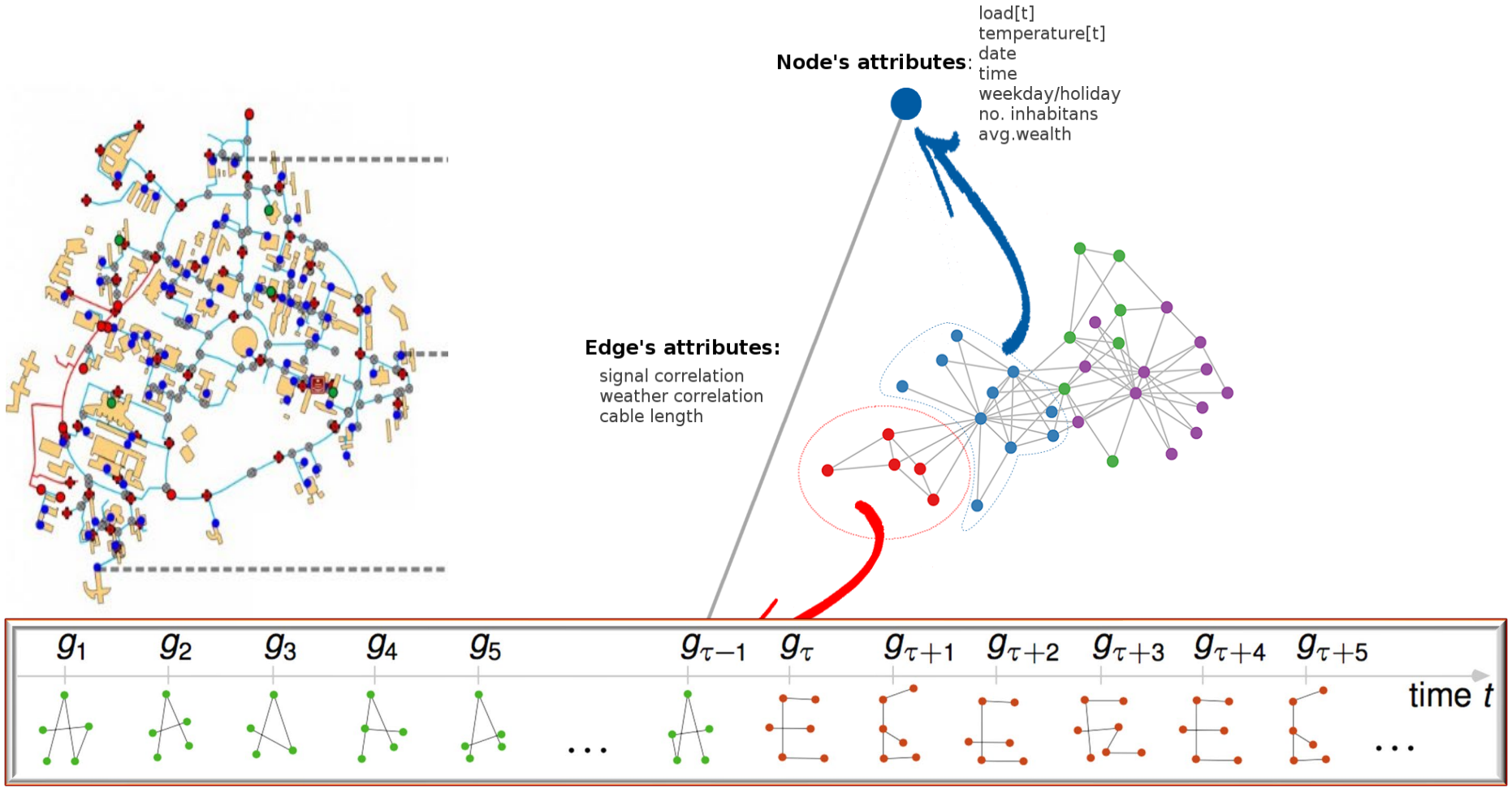
Why graphs

In others are latent

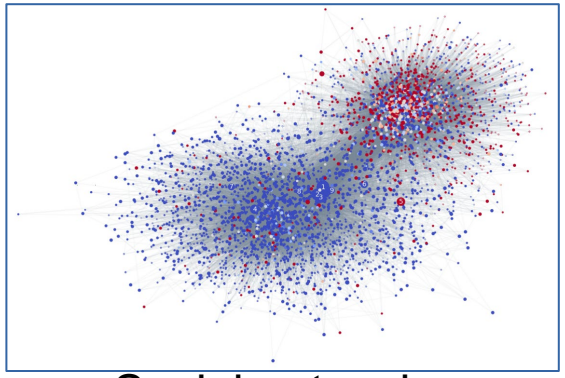


Data streams and graph streams

In others, we derive graphs from timeseries (signals)

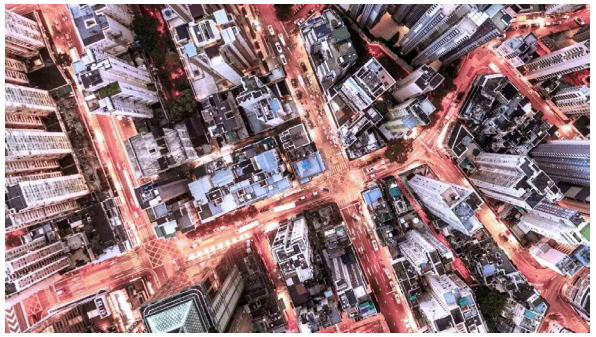


A plethora of applications



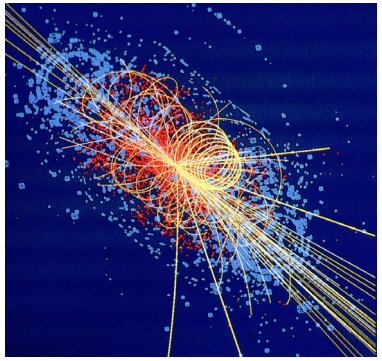
Social networks

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



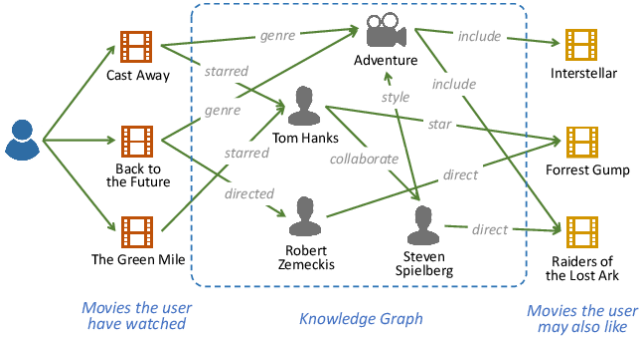
Traffic prediction

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



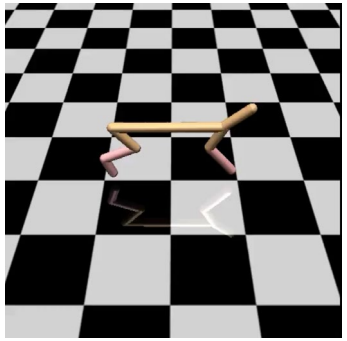
Physics

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."

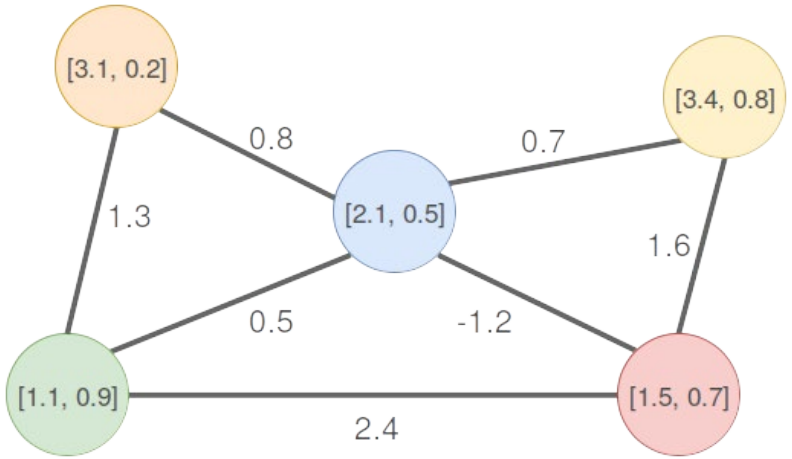


Reinforcement learning

Zambaldi et al. "Relational deep reinforcement learning."

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

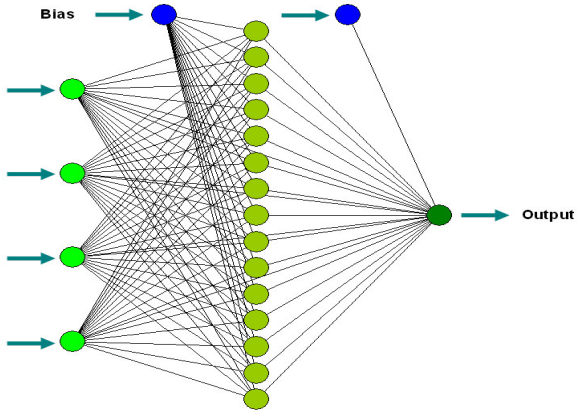
We might argue that having node features is enough provided that you have a strong inference engine (say) able to extract functional interdependencies.



e.g., Pearson's correlation coefficient

X =

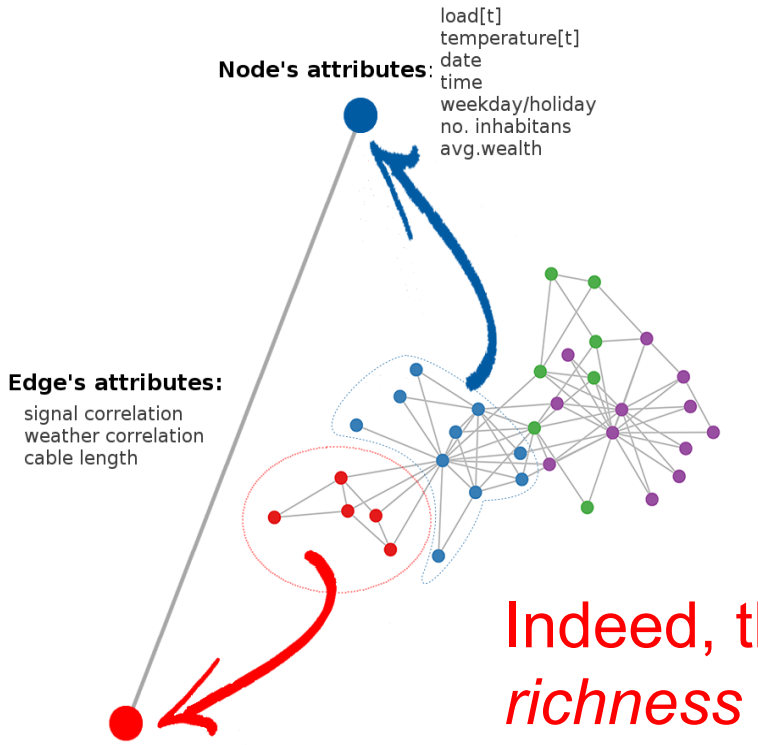
3.1	0.2
3.4	0.8
2.1	0.5
1.1	0.9
1.5	0.7



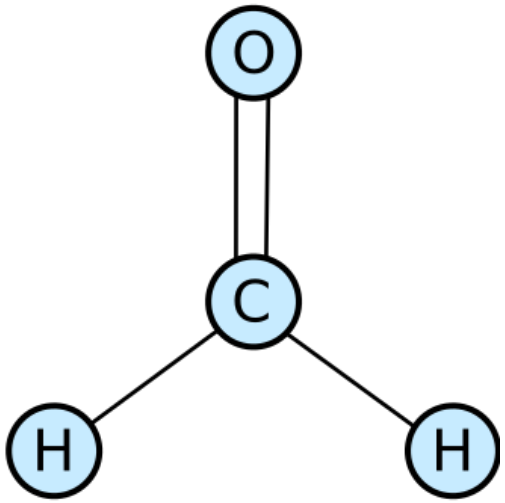
Do we really need to represent and process graphs in a different way?

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

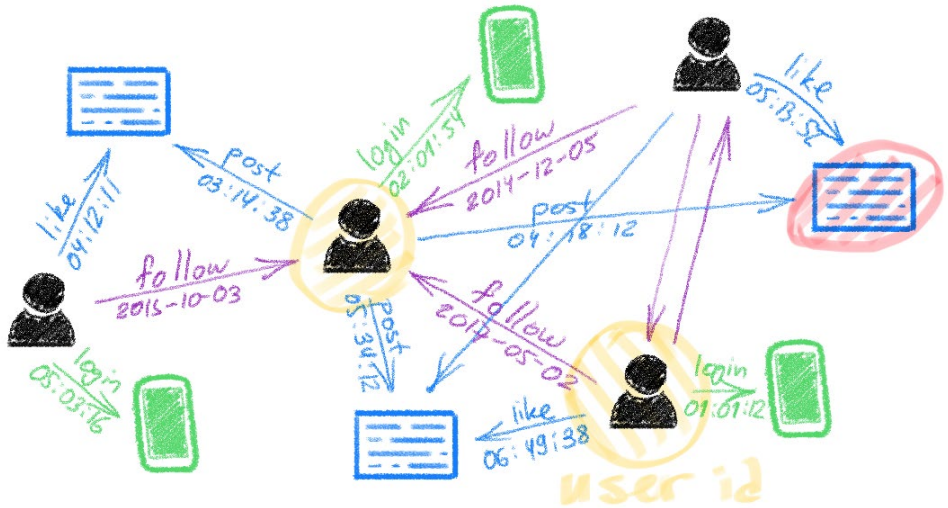
- A school believes that we should place ourselves in-between by taking advantage of both
  - Rich representations
  - Inductive bias

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.

Relational inductive bias

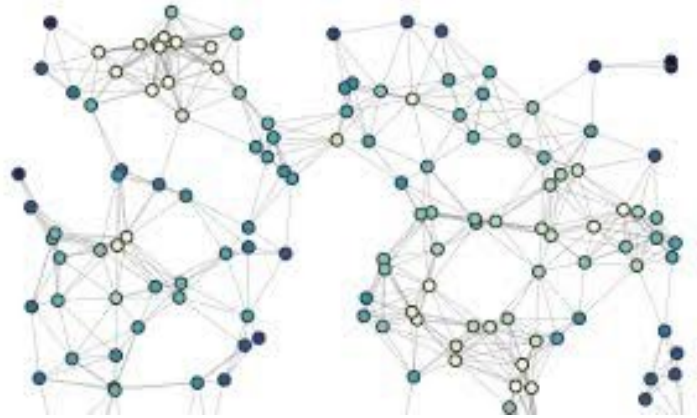
- Complex systems can be seen as a composition of interacting entities



- Not rarely, the world is compositional, or at least, we understand it in compositional terms.

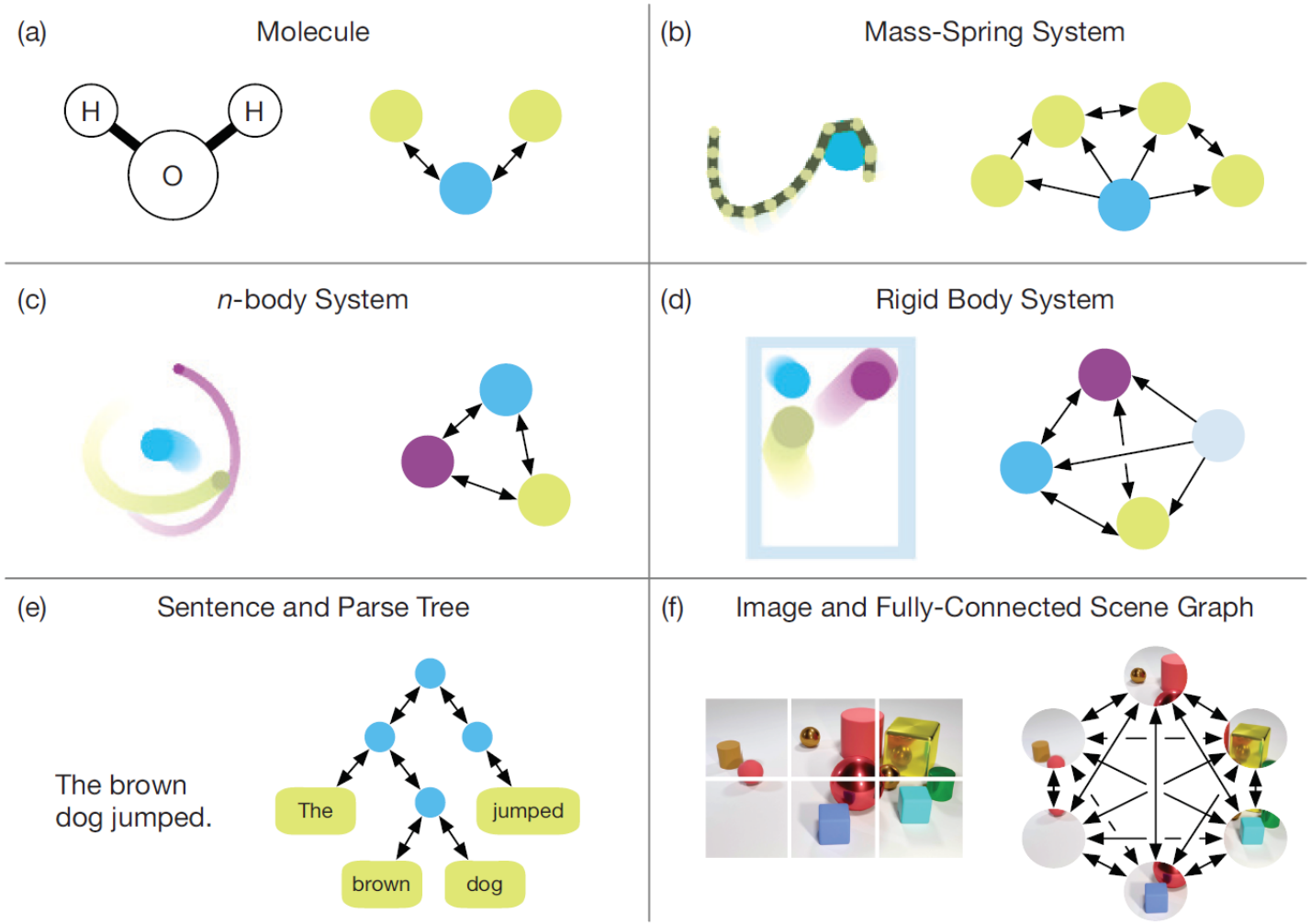
Relational inductive bias

- Recent research has proposed a class of models at the intersection of deep learning and structured approaches, which focuses on reasoning about *explicitly structured data*, in particular **graphs**



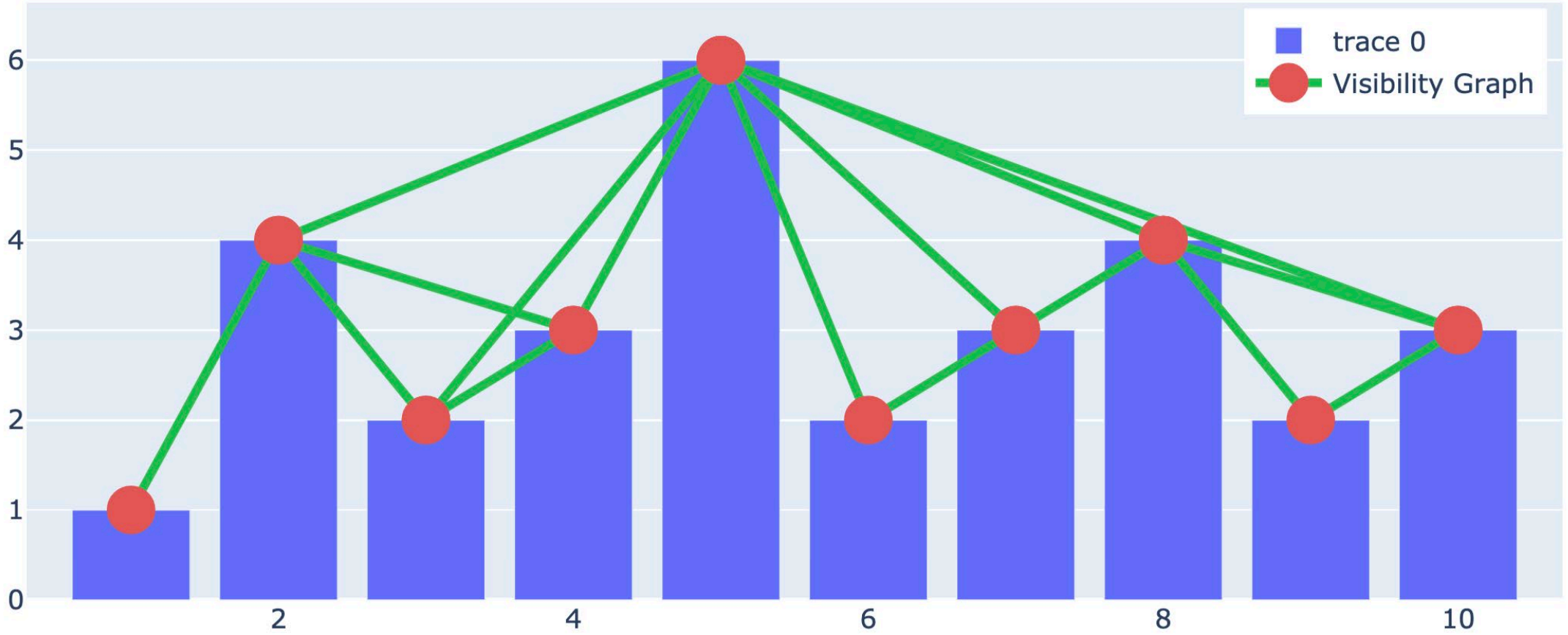
- We have entities (graph nodes)
- We have relations (graph edges)
- How to design a graph, represent information at node level and learn that assigned to edges?
- In other terms, how to compose and populate entities and relations and compute their implications to solve a task?

Graphs and graph representations

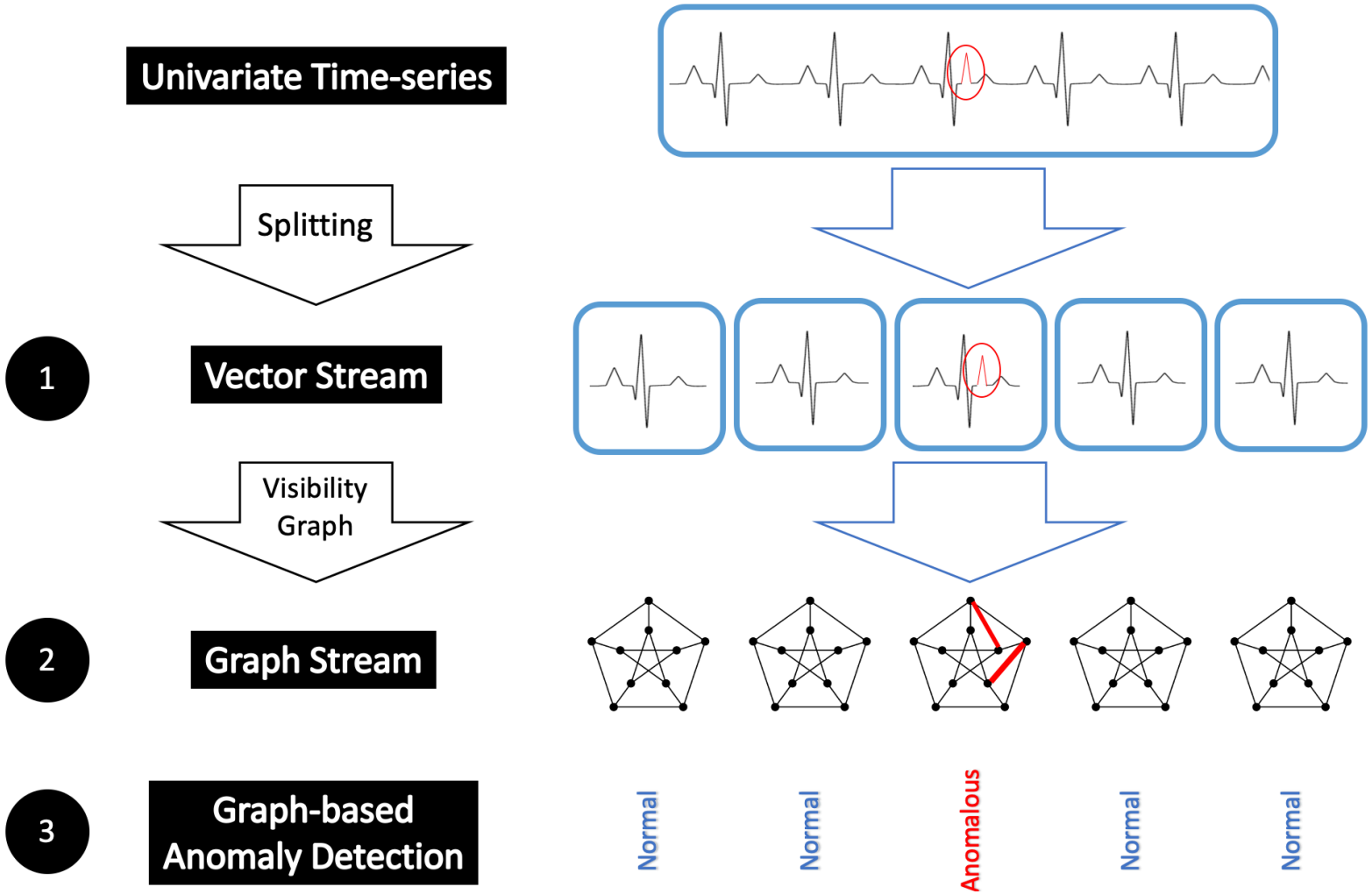


Graphs and graph representations

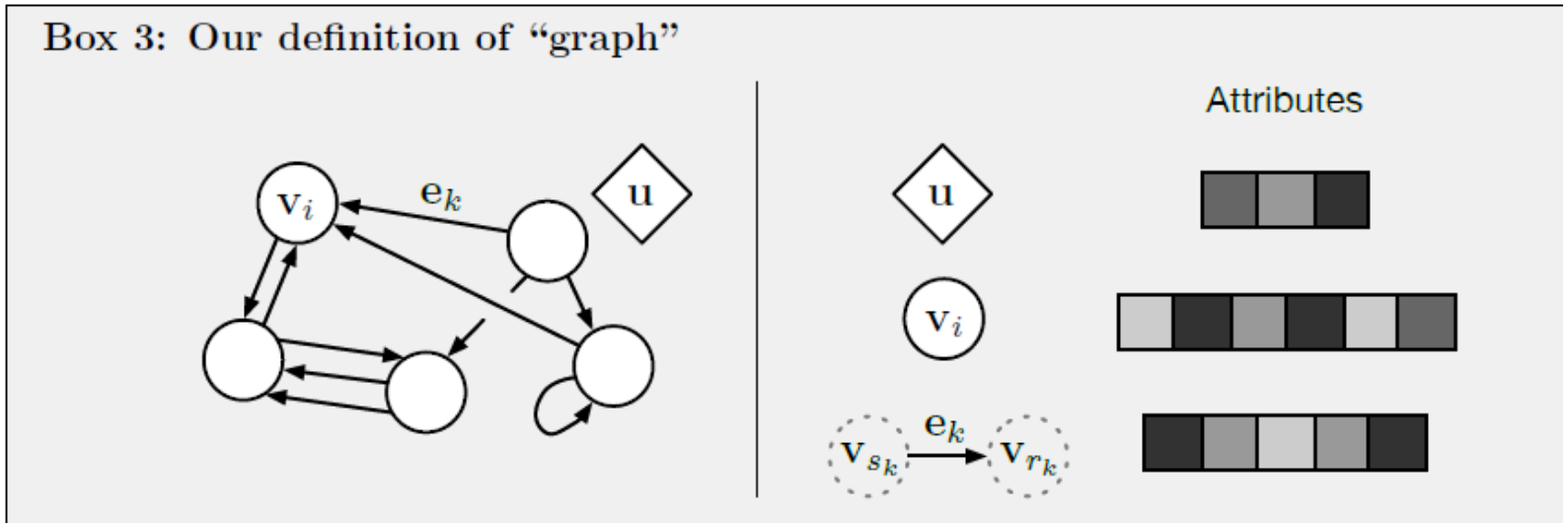
From signals to graphs: e.g., *horizontal visibility graph*



Graph extraction: example

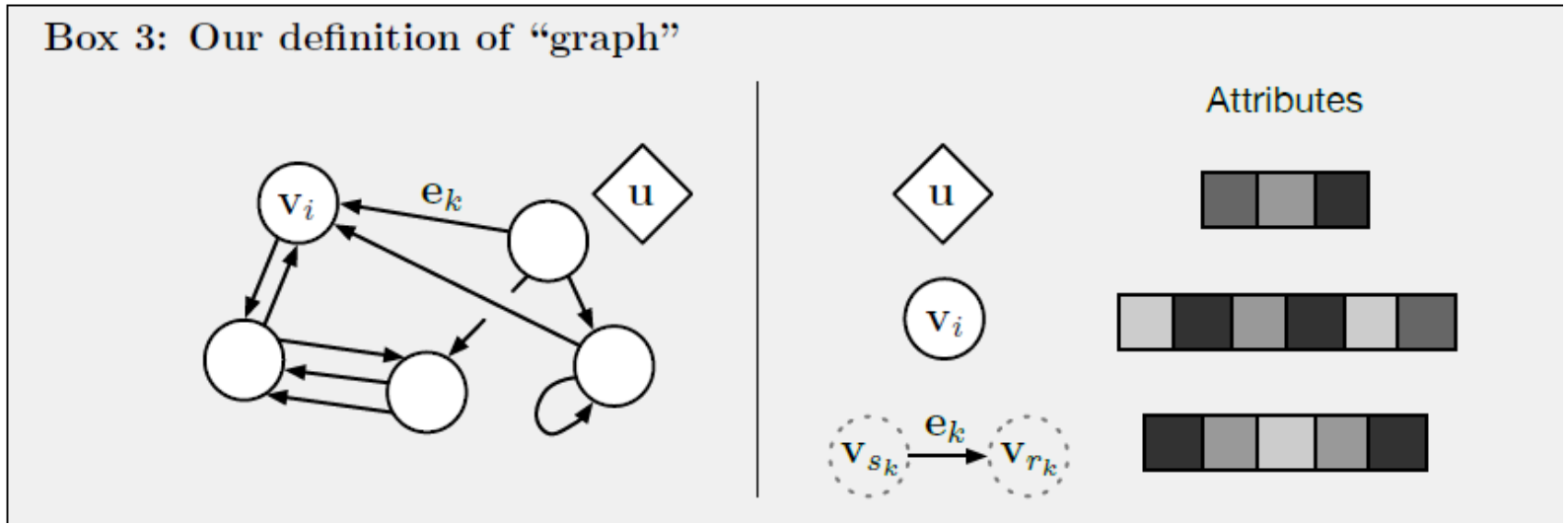


Graphs and graphs…



- 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?

A

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

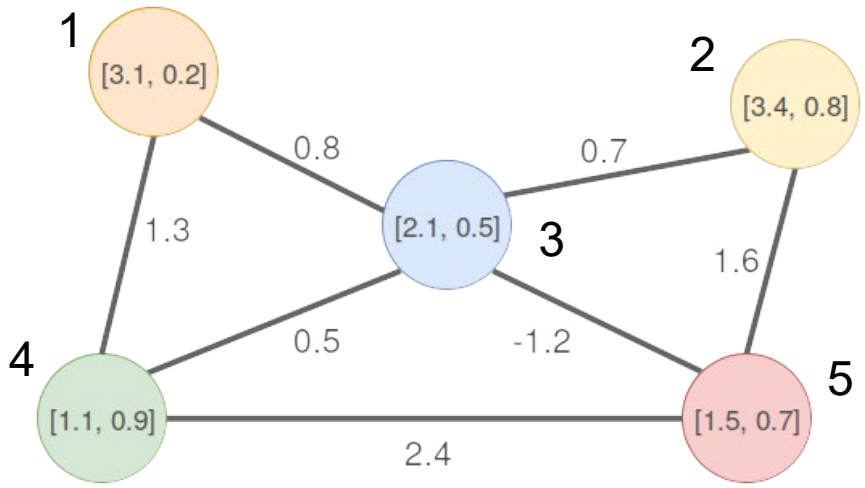
X

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.

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



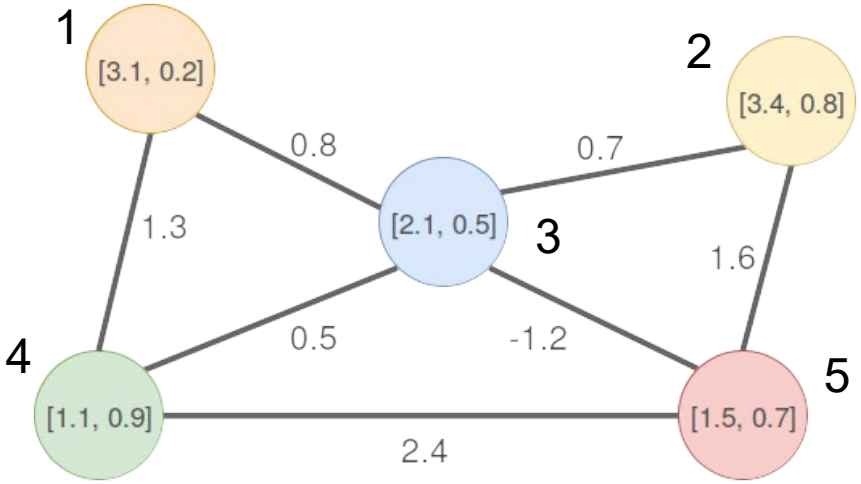
How to represent a graph?

X

3.1	0.2
3.4	0.8
2.1	0.5
1.1	0.9
1.5	0.7

A *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

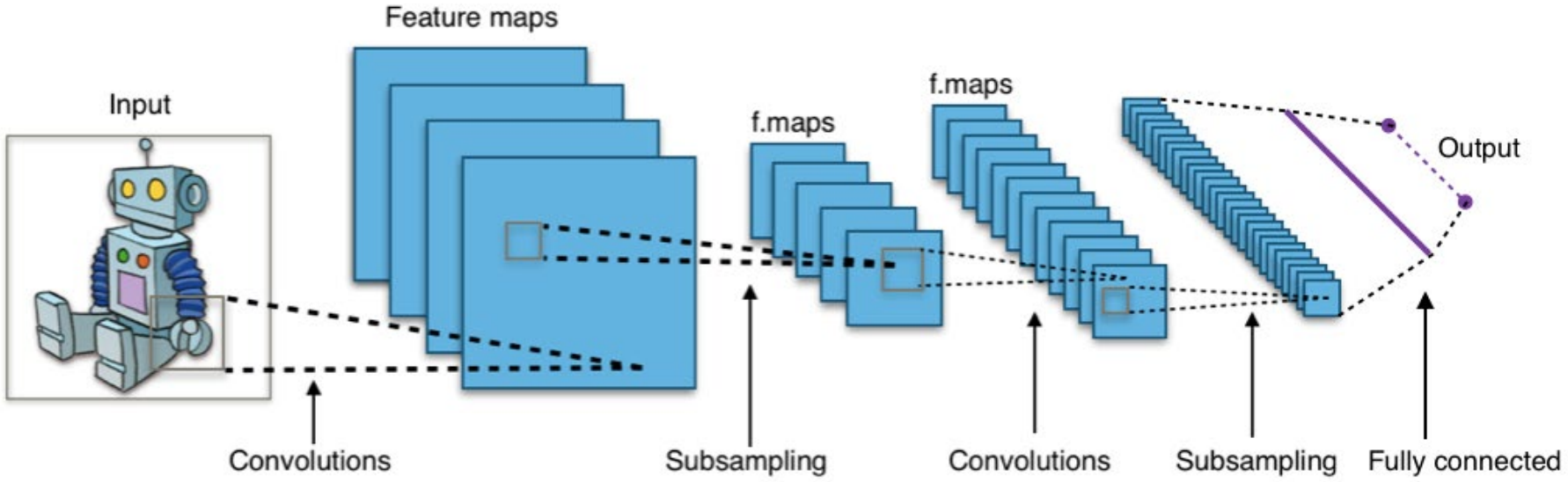
Yes, finally, we have matrices (or tensors)

Processing blocks

“a smooth transition to graph processing operators and architectures”

CNN: Deep Convolutional Networks

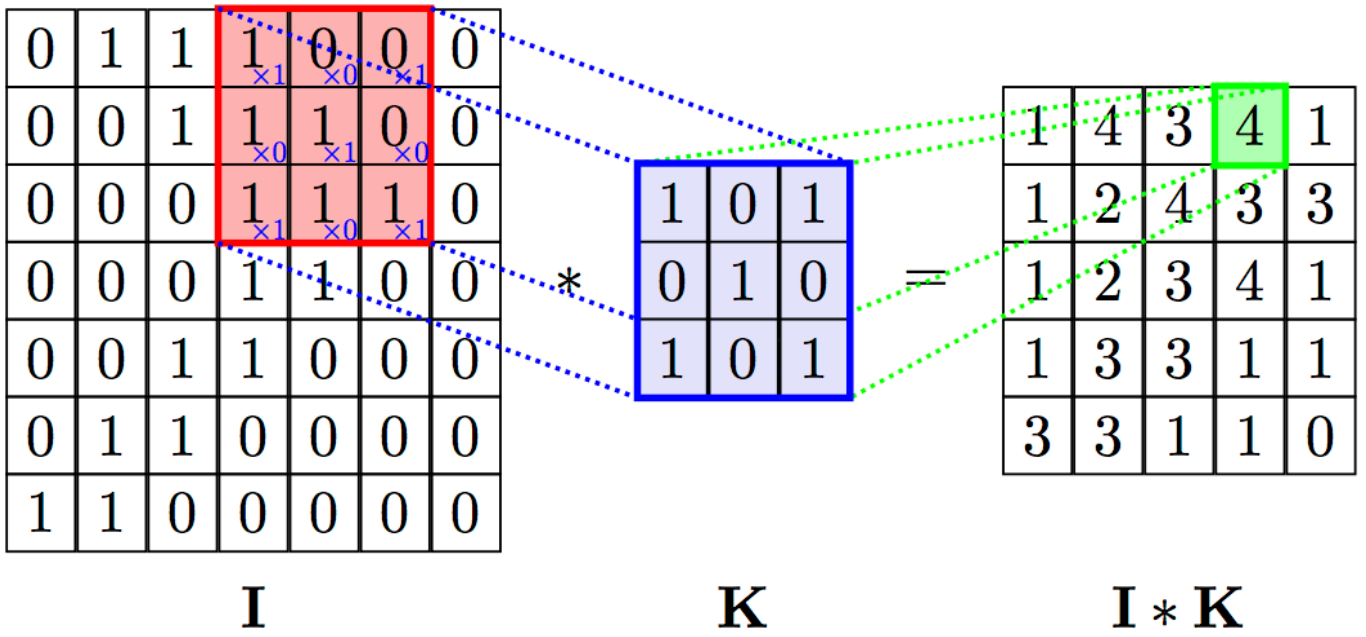
- A CNN takes advantage of the space locality principle (inductive bias)



- 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.

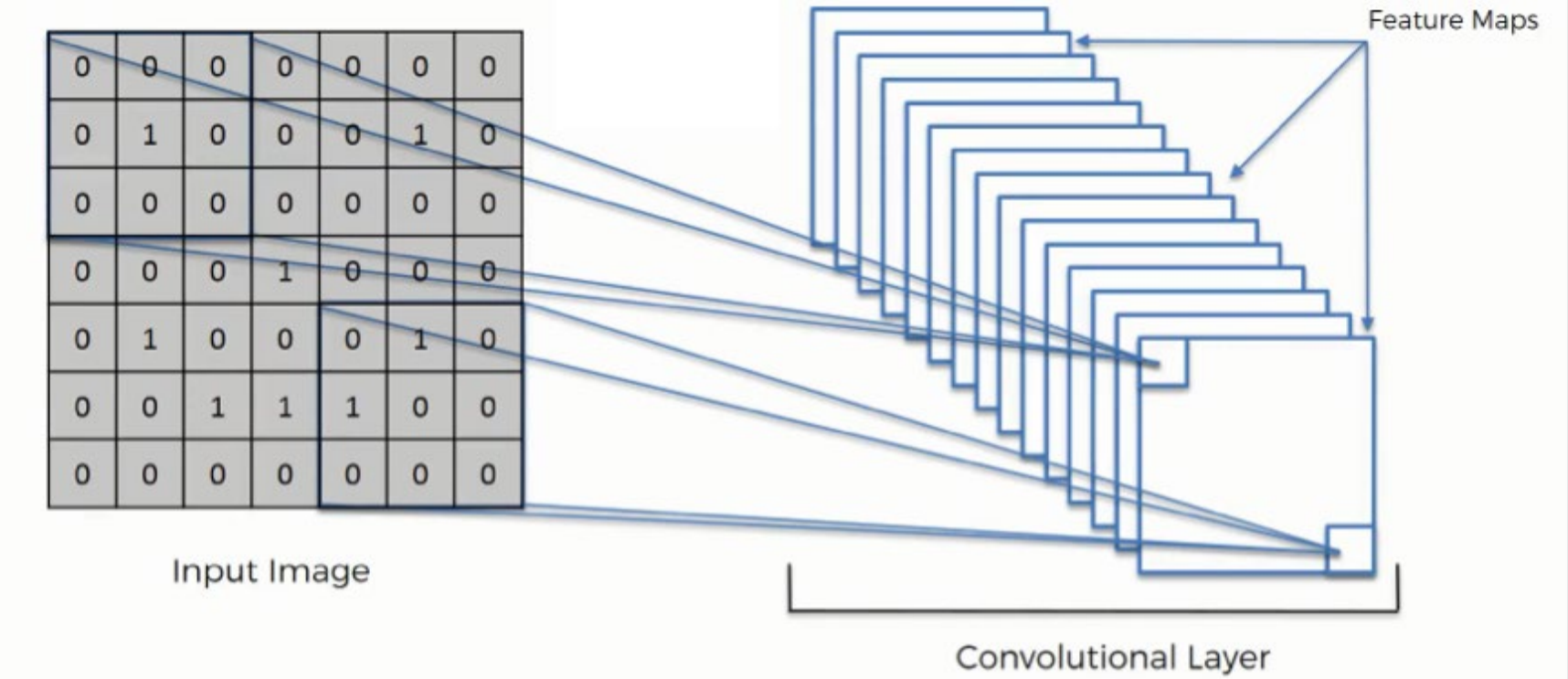
Convolution operator

- 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.



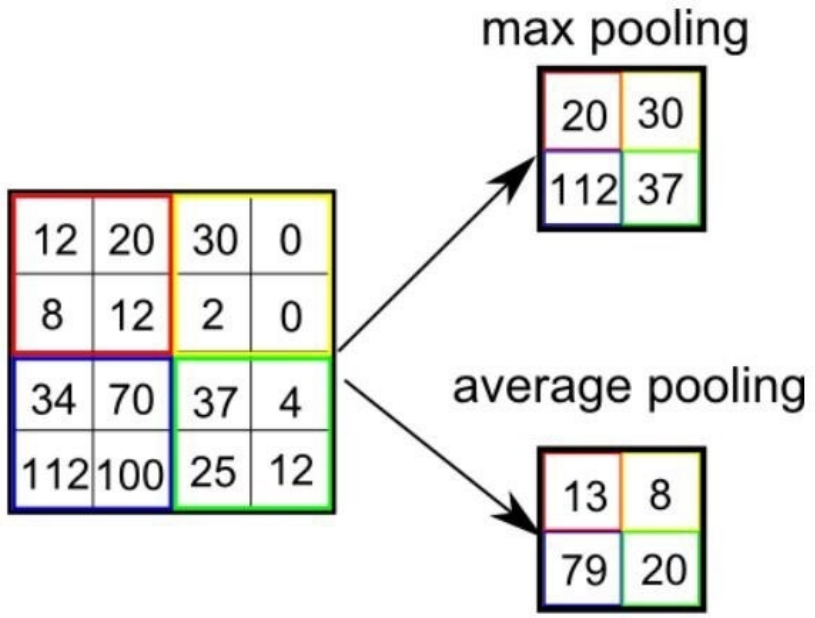
Convolution layer

- 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

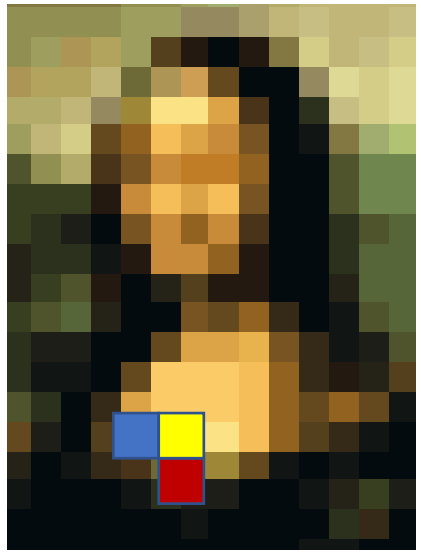
- Pooling layers reduce the image size based on some rules.



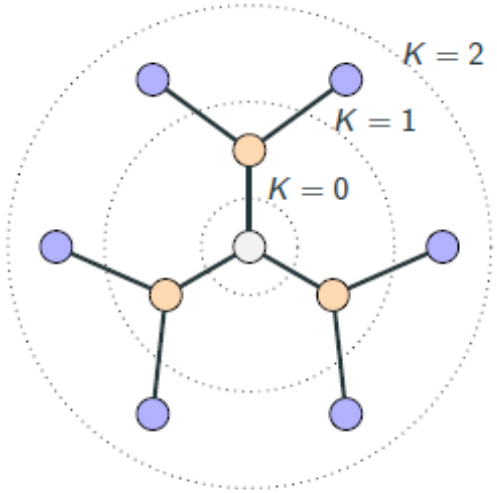
- Different pooling operators can be designed e.g., based on local properties.

Graph processing: Graph Neural Networks

- “Mutatis mutandis” we can naively extend the CNN to a GNN (Graph Neural Network)
- In images, functional proximity coincides with physical proximity

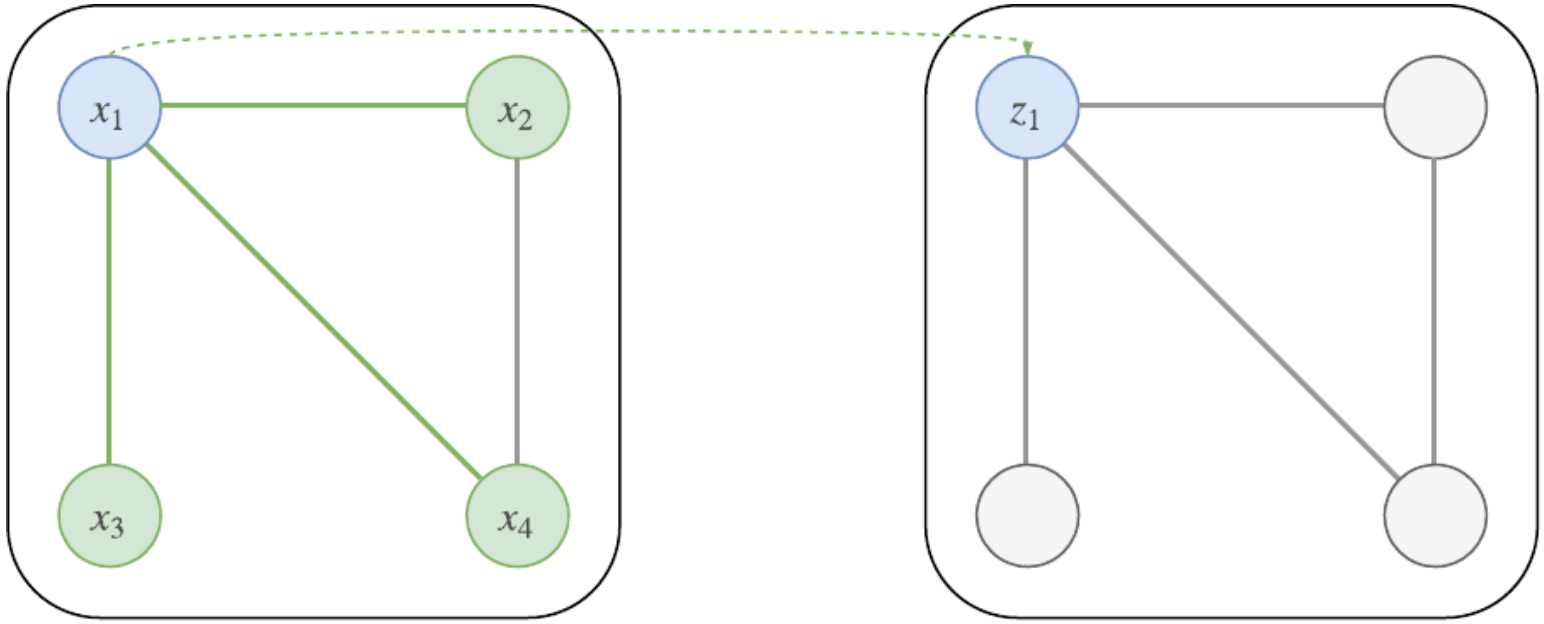


- In a graph, functional proximity does not necessarily coincide with physical proximity; yet the locality principle is there…



GNN operators: Graph Convolution

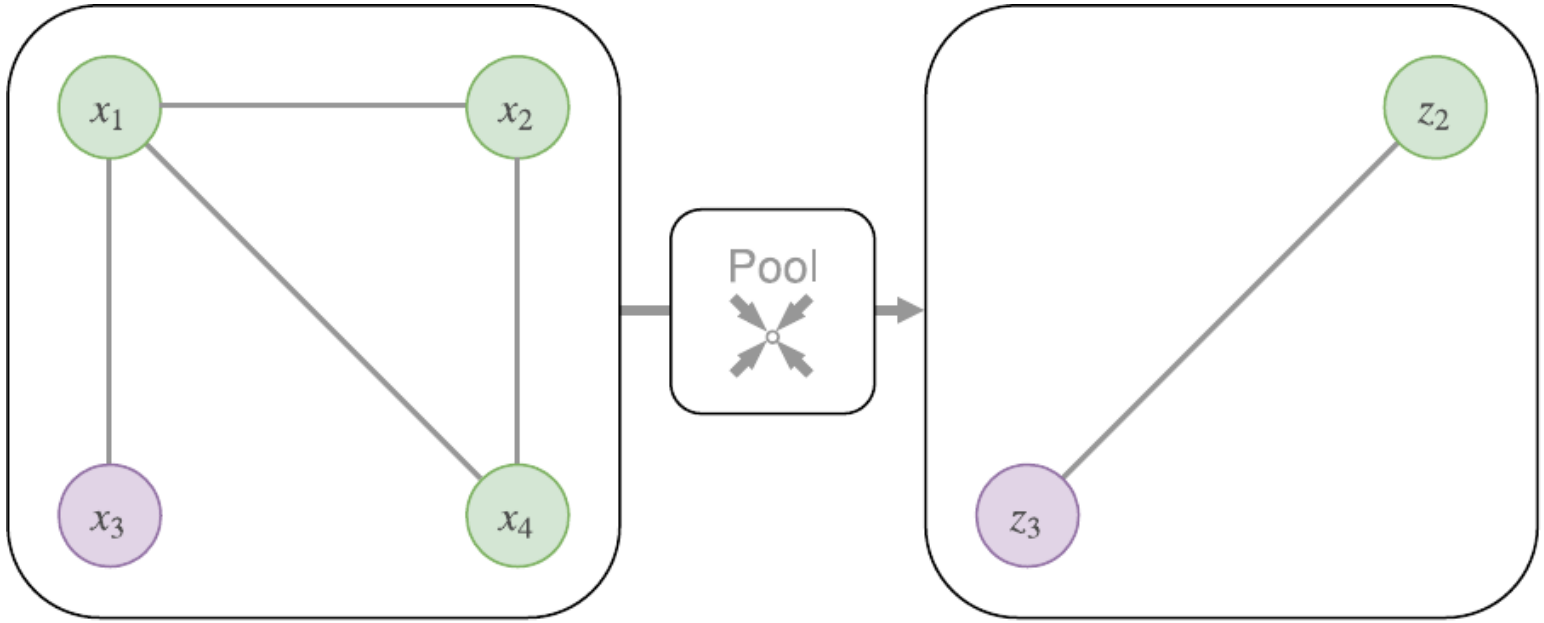
- Graph convolution exploits the local neighborhood of each node to compute a node value embedding



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

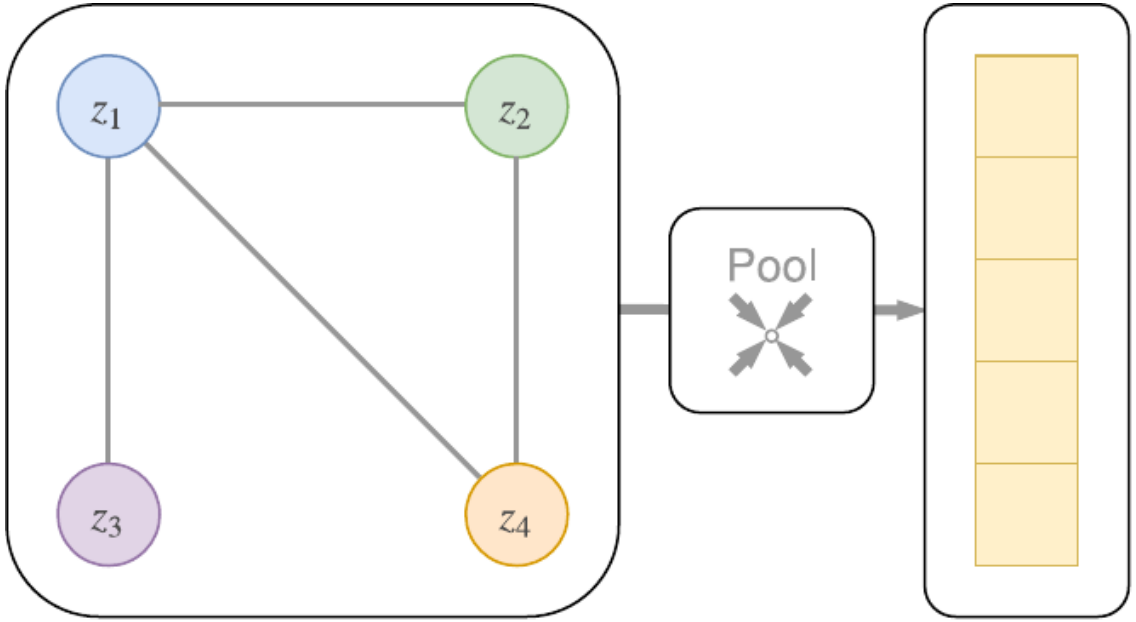
Graph Pooling

- Pooling aggregates nodes (shrinks the graph topology) to
  - provide a more abstract representation of the graph
  - reduce the graph complexity



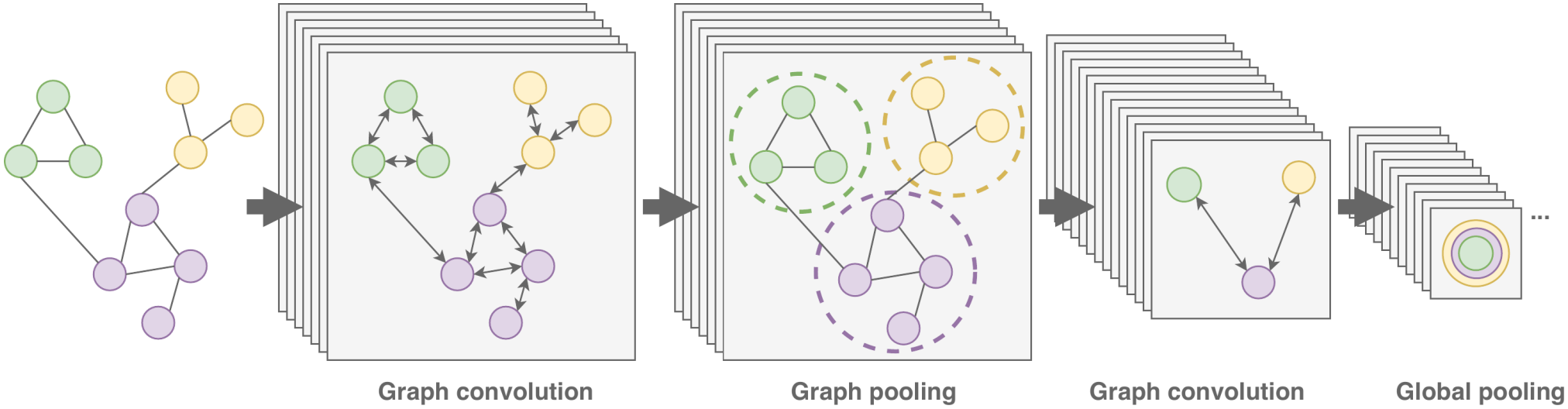
Global Graph Pooling

- Global pooling embeds the graph to a vector (more to come later)



GNN: Graph Neural Networks

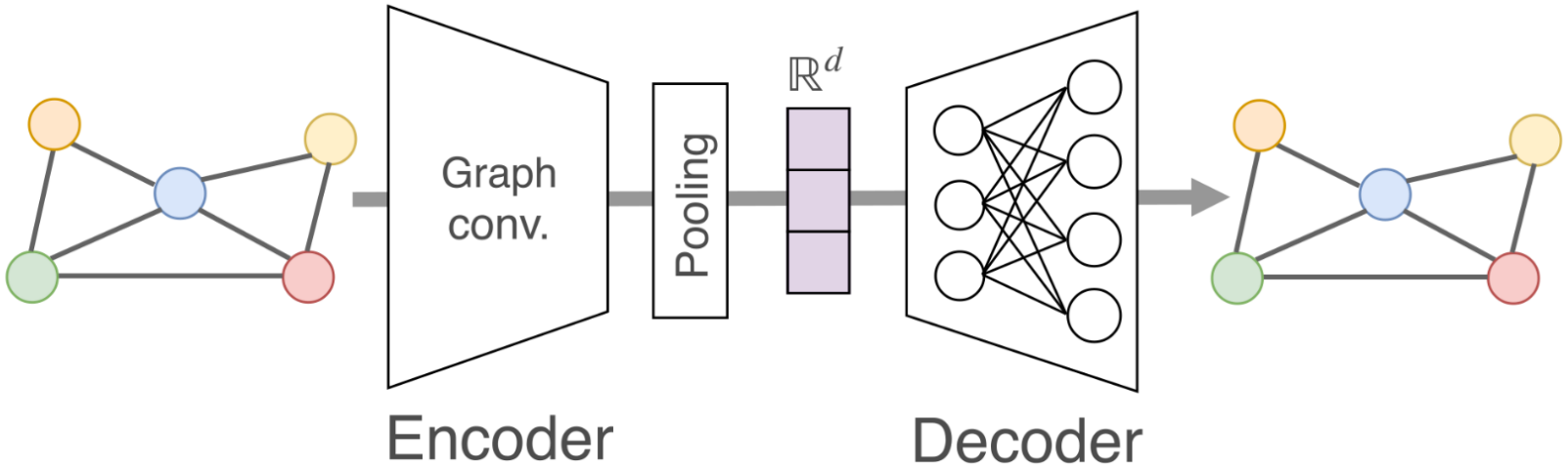
- We get a deep network – GNN – by interleaving operators



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

Graph autoencoders

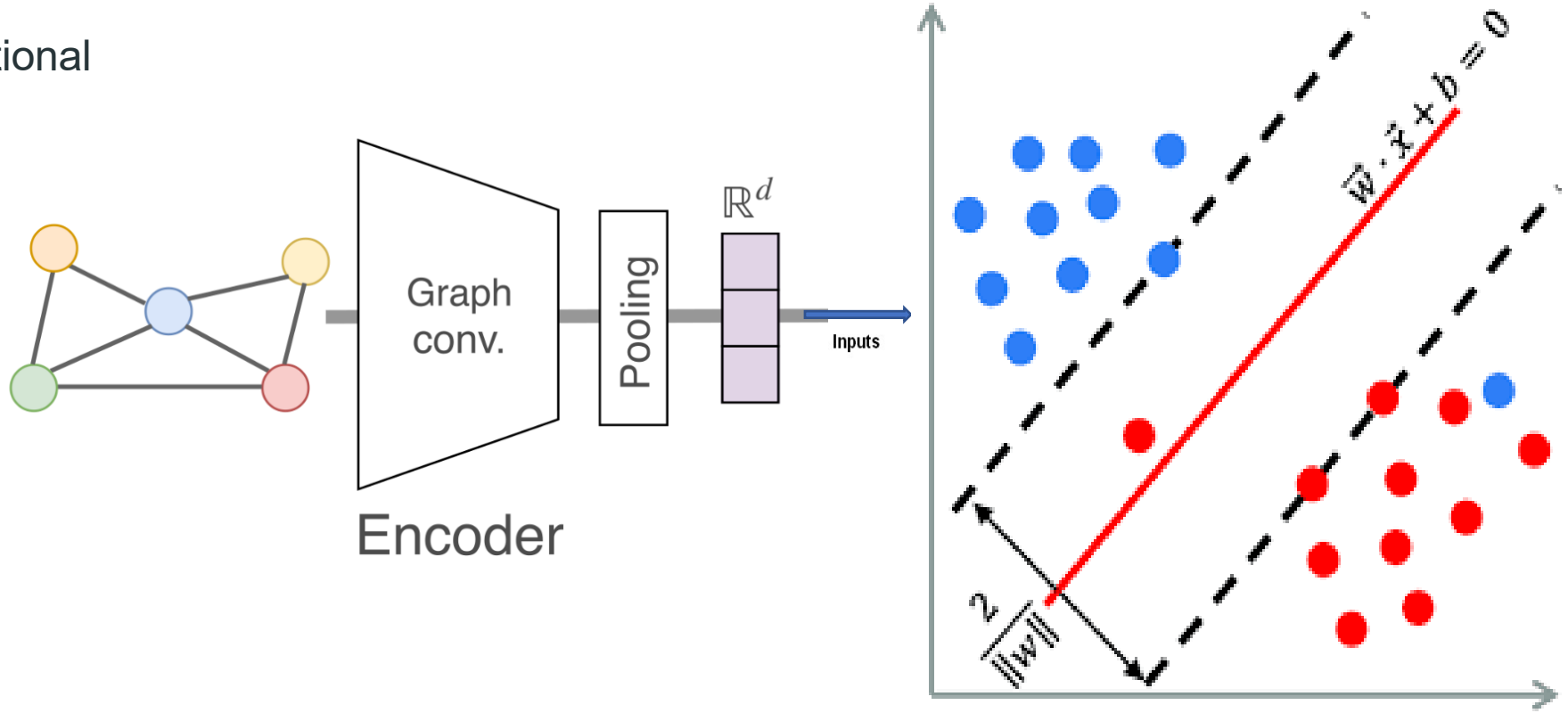
- The encoder is composed of graph convolutional layers with the pooling one
- A dense decoder reconstructs the matrices describing the graph



- The latent space represents a natural embedding

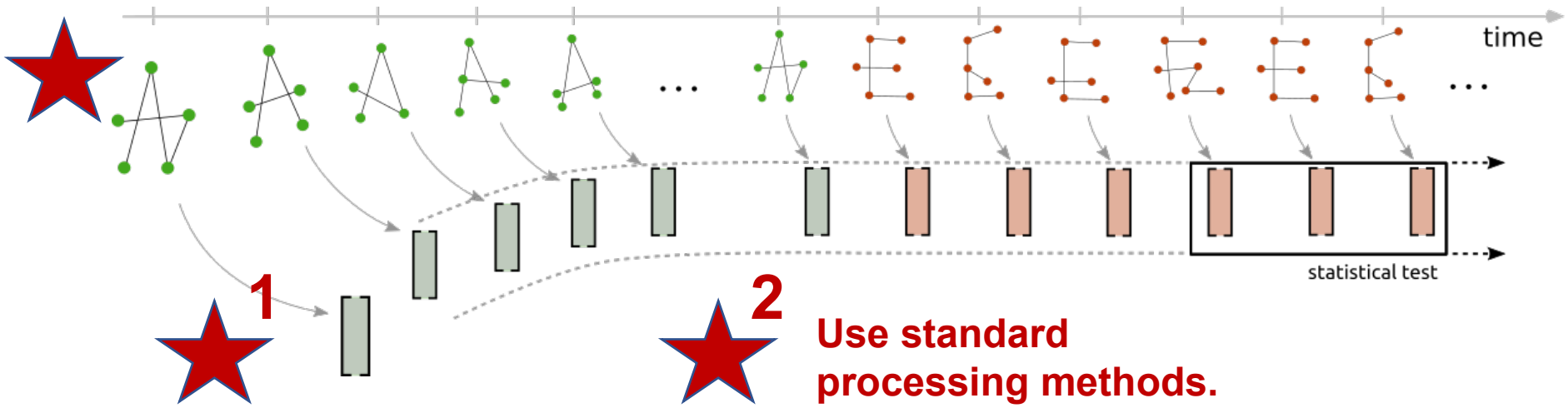
Latent space representation

- 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 (graph embedding)

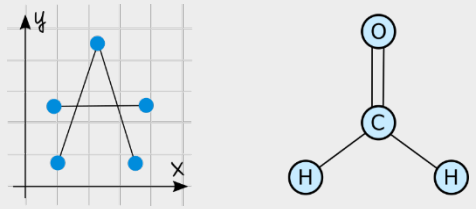


Graphs and embedding spaces

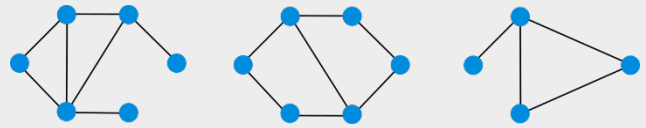
Which types of graphs are we interested in?

Attributed graphs represent a very large family of graphs

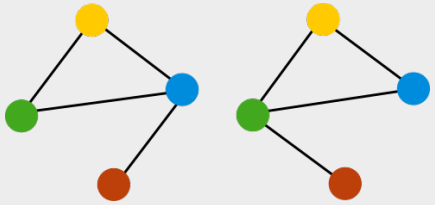
Generic attributes



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 (finite) sets of vertices and edges
  - $\mathcal{A}$ *:* set of attributes
  - *a*: attribute function

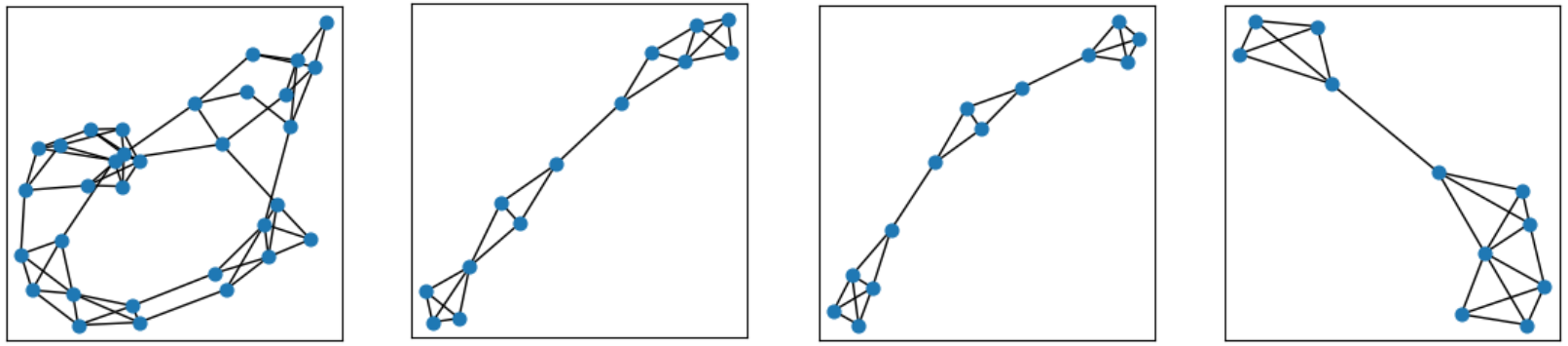
$$a : V \cup E \rightarrow \mathcal{A}$$

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

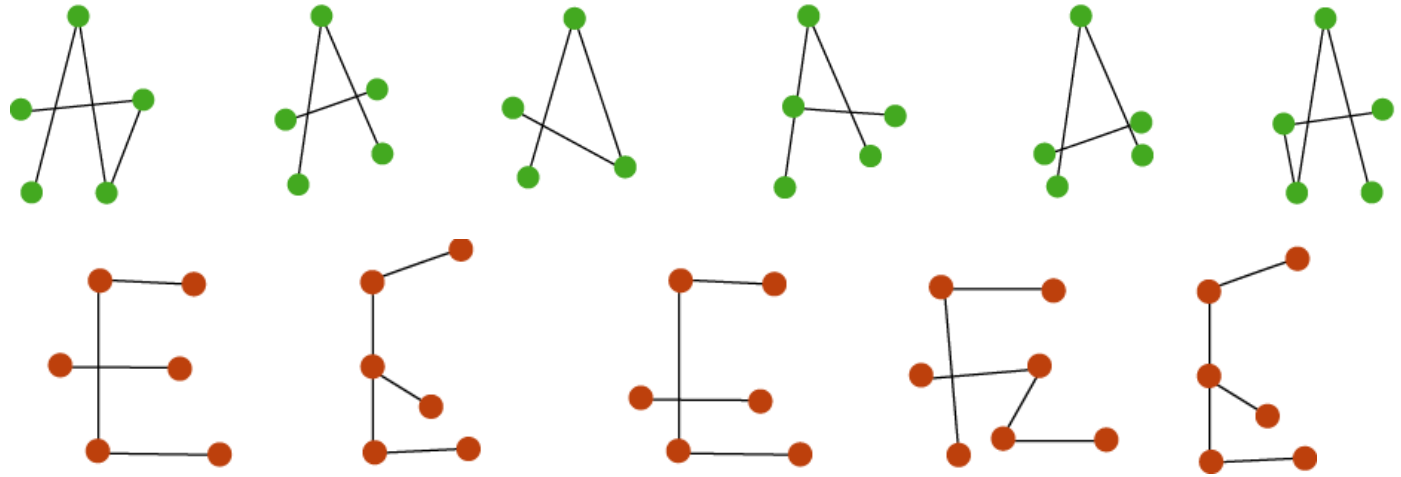
B.J. Jain, On the geometry of graph spaces, Discrete Applied Mathematics, December 2016.

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 to generate graph g_i starting from graph g_j

Operations:

- node insertion/deletion
- edge insertion/deletion

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

- node modification
- edge modification

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

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

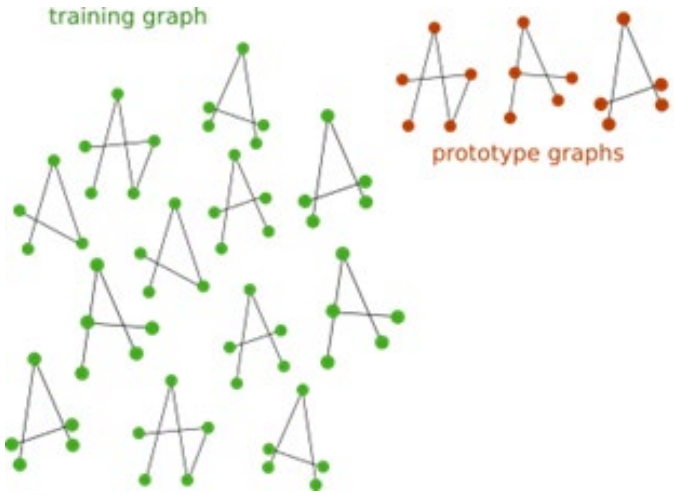
J. Jain, On the geometry of graph spaces, DAM, 2016.

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)
- **Train free random approaches** (later)

Dissimilarity representation

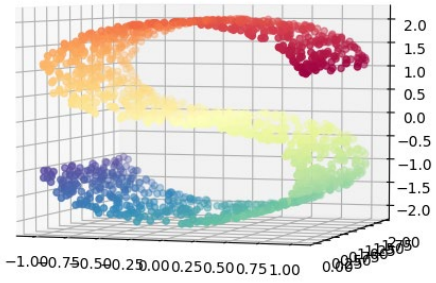
- Training phase:
  - Identify a set of prototypes R

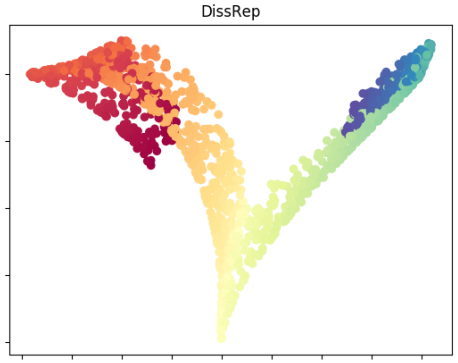


- Out-of-sample technique:

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

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

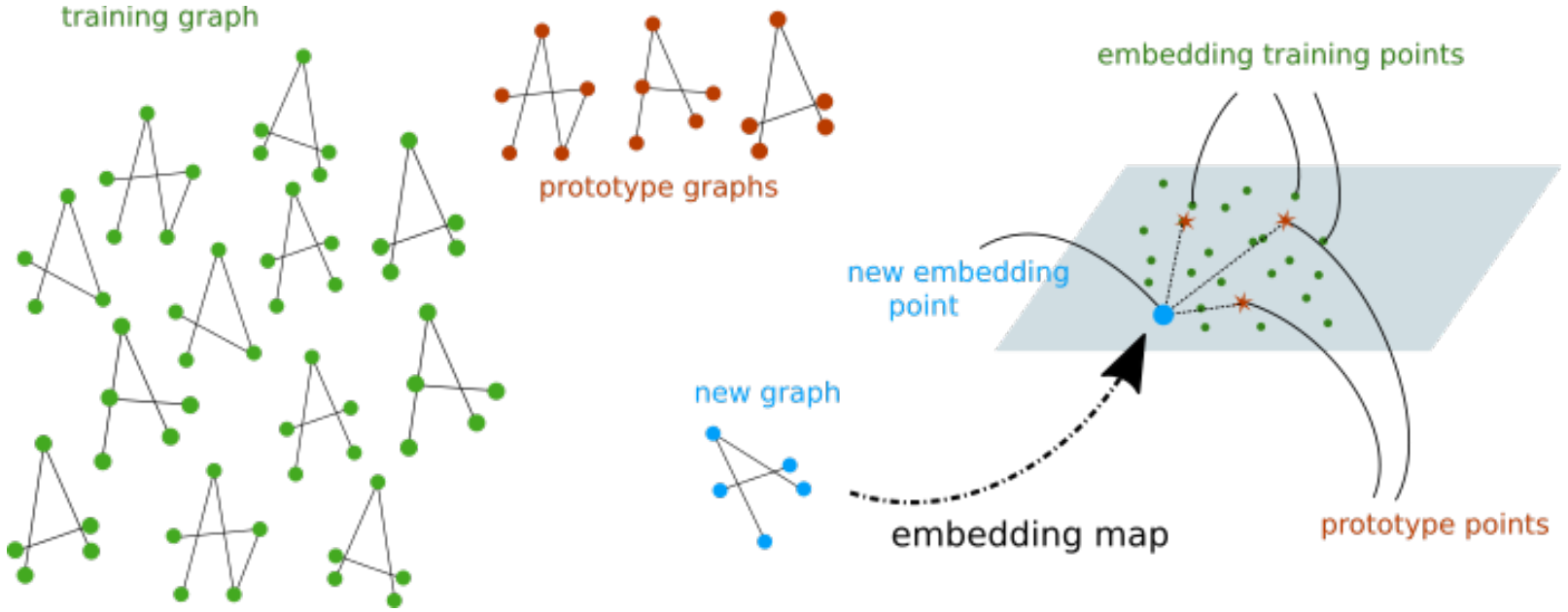




D. ZE. Pękalska and R. P. W. Duin, The Dissimilarity Representation for Pattern Recognition: Foundations and Applications. WS, 2005.

Multi-dimensional scaling

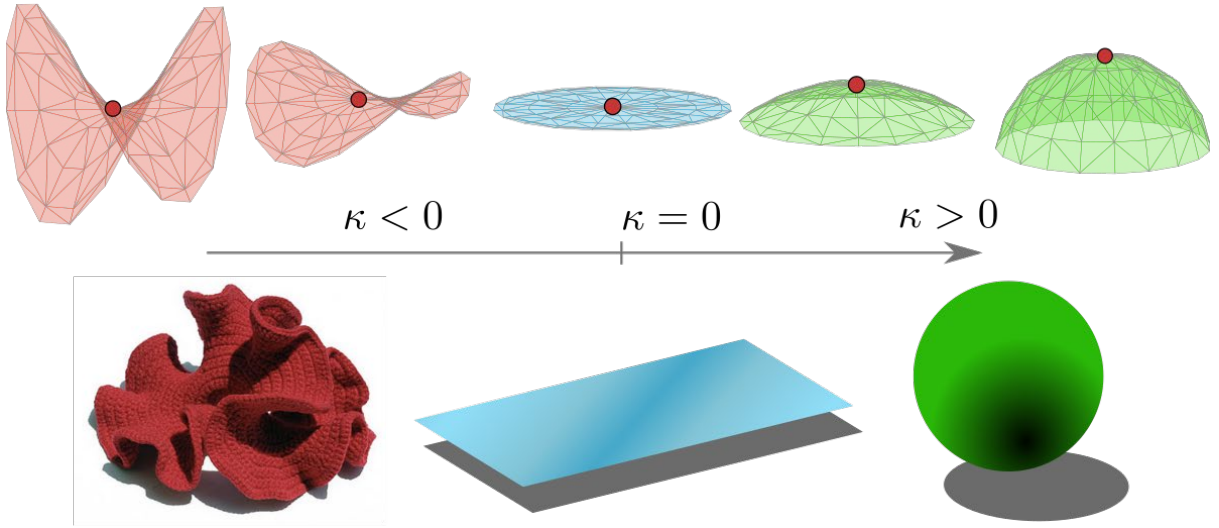
- Training phase:
  - Identify a set of prototypes
- Out-of-sample technique:
  - Use the dissimilarity representation by attempting to preserve the distance from prototypes



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

A step beyond Euclidean embedding

- Real-world data sets are often non-Euclidean
- Constant curvature Riemannian manifolds
  - Riemannian manifolds provide a metric structure
  - Mathematics is relatively easy (geodesic, distribution)



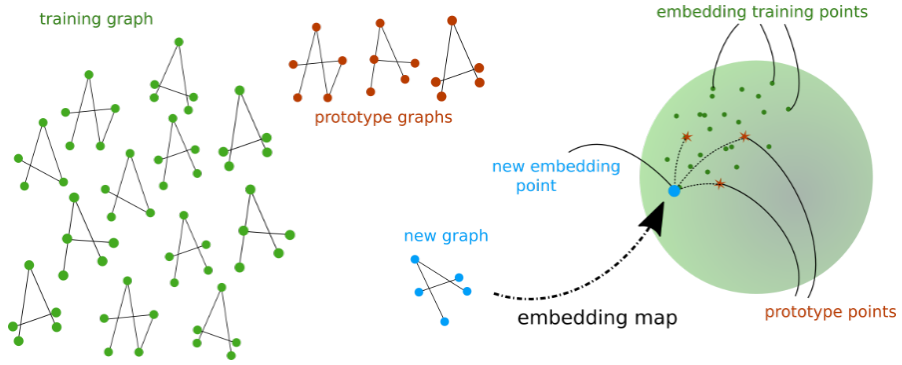
D. Zambon, L. Livi, and C. Alippi, Anomaly and Change Detection in Graph Streams through Constant-Curvature Manifold Embeddings, IEEE IJCNN, 2018

Embedding on constant curvature manifolds

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

Training phase

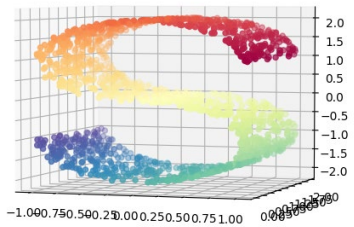
- Select a set of prototypes

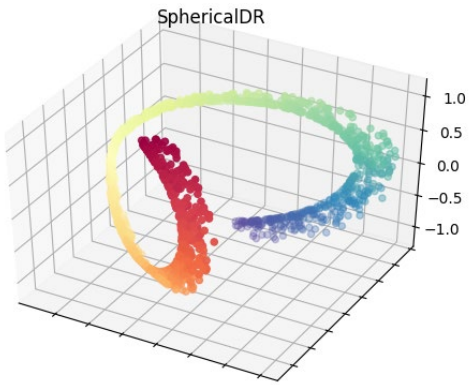


Out-of-sample technique

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

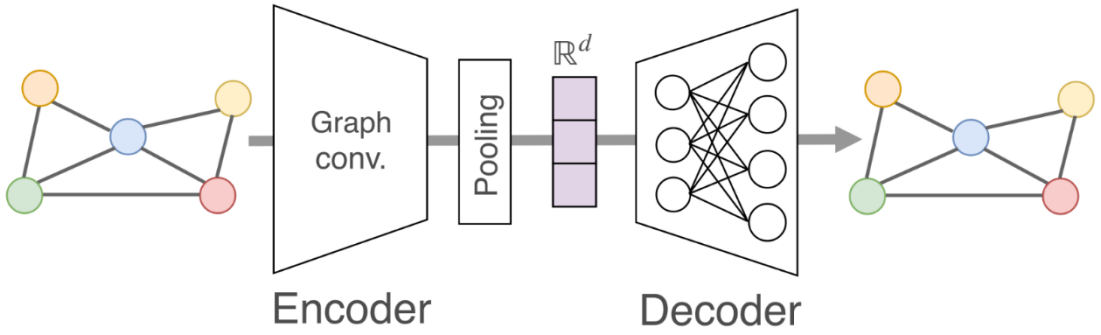
$$\rho(x, y) = \frac{1}{\sqrt{\kappa}} \arccos(\kappa \langle x, y \rangle_\kappa),$$



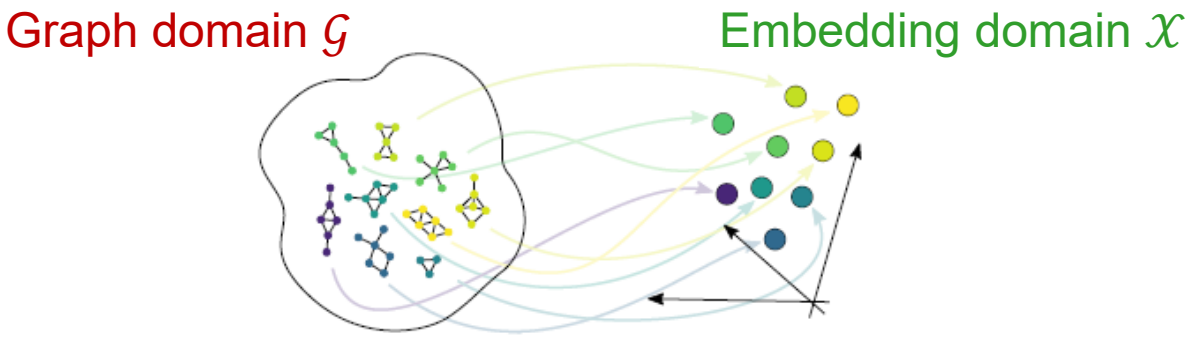


Graph autoencoders

- Autoencoders provide a nice way to automatically build the embedding

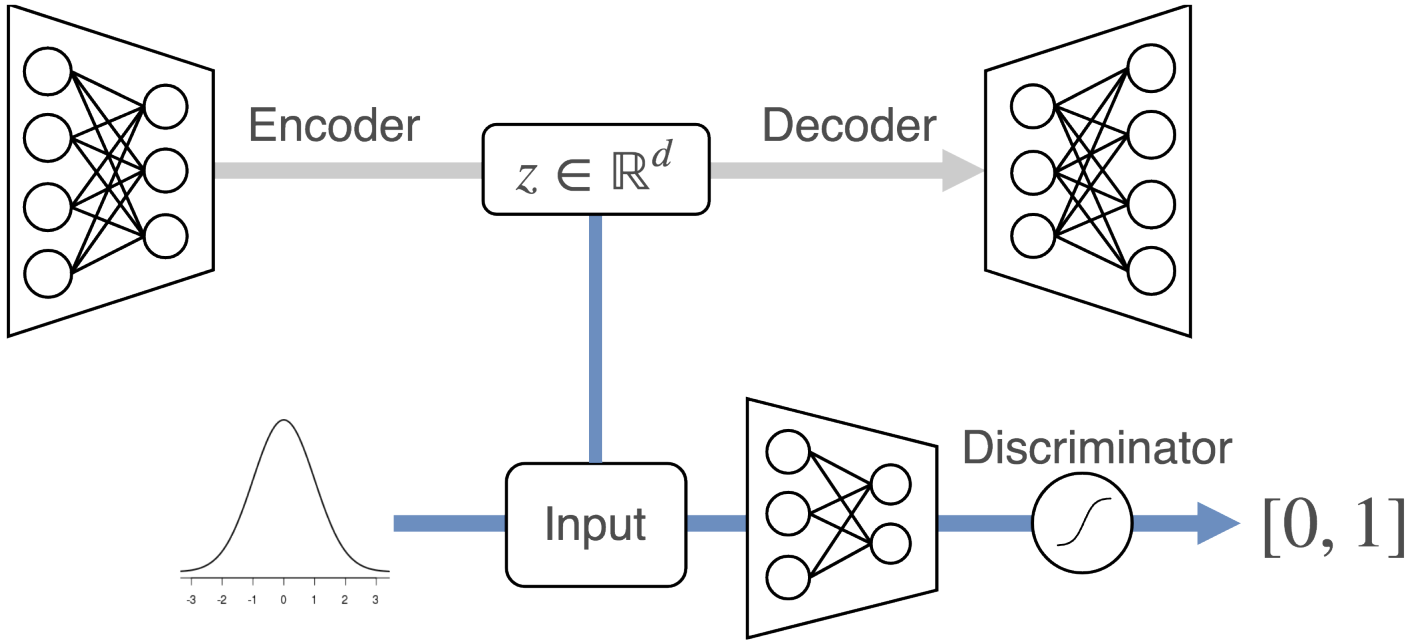


- But neither we can grant that the distance nor that the concept of distribution are preserved in the graph/embedding spaces



Adversarial autoencoders

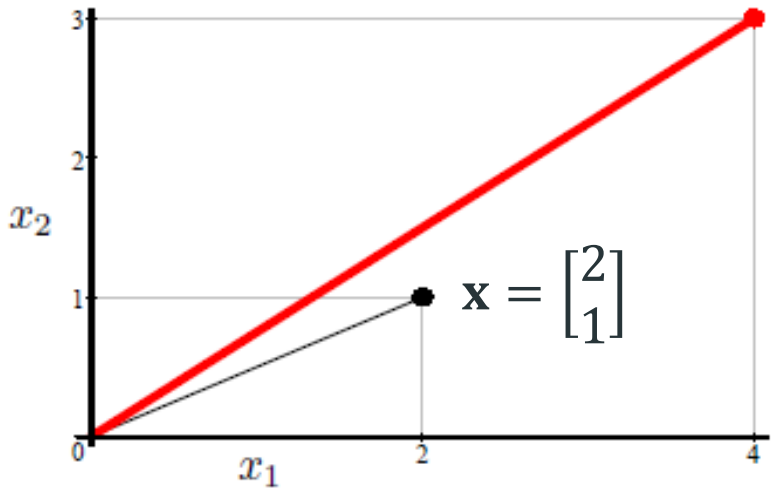
- Match the distribution of embedded information in the latent space with an arbitrary (given) prior
- In this way we impose the concept of distance and a wished distribution in the embedded space



Linear algebra (recall): a matrix as an operator

- Consider an n-dimensional real column vector x and a square matrix $\mathbf{A}$ of order n
- By taking $\mathbf{Ax}$, matrix $\mathbf{A}$ can be seen as a function (operator) mapping $\mathbf{x}$ to vector $\mathbf{x}' = \mathbf{Ax}$

$$\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$



Eigenvalues and eigenvectors

- Is there any “interesting direction” for $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$?

- Well, one might be the one in which $\mathbf{x}$ is transformed into a $\mathbf{x}' = \mathbf{Ax}$ and both vectors share the same direction, the second eventually scaled in magnitude.
  This request can be modelled as

$$\lambda \mathbf{x} = \mathbf{Ax}$$

Eigenvalues and eigenvectors

- Equation $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0$ is called the *eigenvalue equation*
- The λs for which the equation holds are called *eigenvalues*, and associated $\mathbf{x}$s are named *eigenvectors*
- A theorem states that the eigenvalue equation has non null solutions iff $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$
- Therefore:
  - We have n eigenvalues;
  - Given eigenvalue λ, the eigenvector satisfies equation $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0$

Eigenvalues and eigenvectors

- Simple example $\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$
- Eigenvalues satisfy

$$\det(\mathbf{A} - \lambda\mathbf{I}) = \det\left(\begin{bmatrix} 2-\lambda & 0 \\ 0 & 3-\lambda \end{bmatrix}\right) = 0$$

- Yielding to $\lambda_1 = 2,\ \lambda_2 = 3$

$$(\mathbf{A} - \lambda_1\mathbf{I})\mathbf{x} = \begin{bmatrix} 2-\lambda_1 & 0 \\ 0 & 3-\lambda_1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

Eigenvectors $\quad \mathbf{u}_{\lambda_1} = \begin{bmatrix} x_1 \\ 0 \end{bmatrix};\quad \mathbf{u}_{\lambda_2} = \begin{bmatrix} 0 \\ x_2 \end{bmatrix}$

- Privileged directions are $x_2 = 0,\ x_1 = 0$

Spectral Theorem

- Consider a square symmetric matrix $\mathbf{A}$ of order n, then,

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i \mathbf{u}_{\lambda_i} \mathbf{u}_{\lambda_i}^T$$

Each eigenvector is scaled to have magnitude 1 (*orthonormality*).

- In our example:

$$\mathbf{A} = 2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} + 3 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix}$$

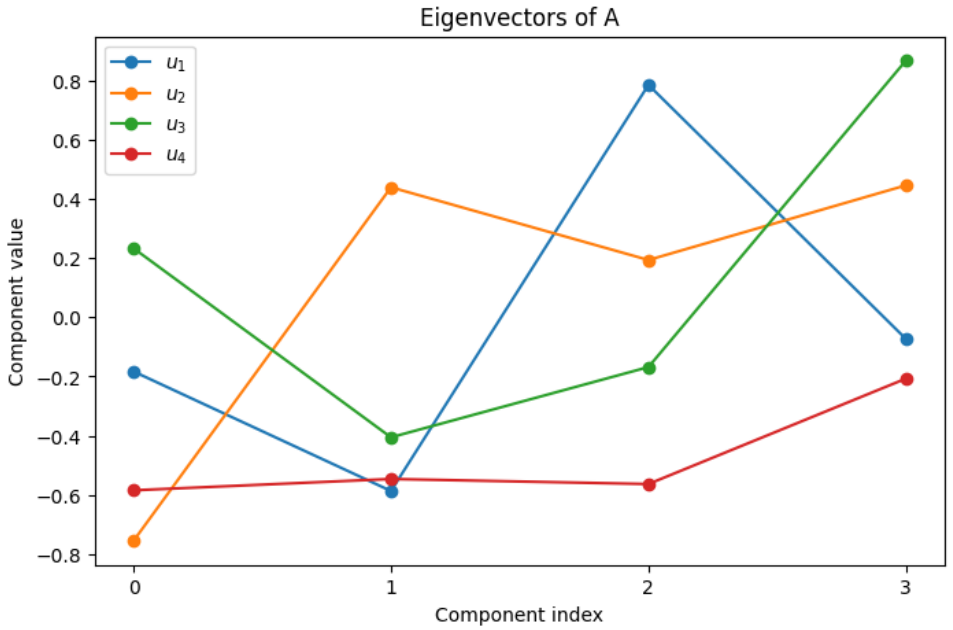
- The spectral theorem is very much used in PCA to reduce the complexity of the input space (it is applied to the covariance matrix of the inputs)

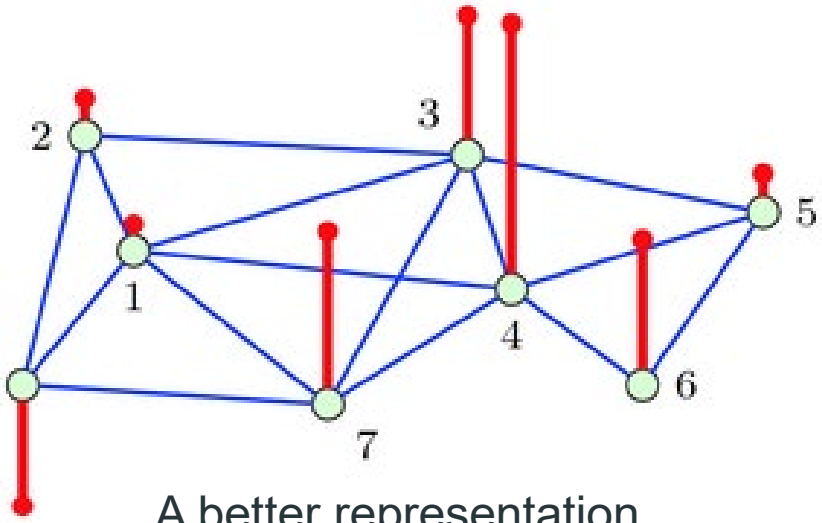
Eigenvectors

- Consider matrix

$$A = \begin{bmatrix} 2 & 3 & 4 & 2 \\ 3 & 1 & 5 & 0 \\ 4 & 5 & 0 & 1 \\ 2 & 0 & 1 & 1 \end{bmatrix}$$

- Plot eigenvectors and intend them as signals (whatever it does mean):

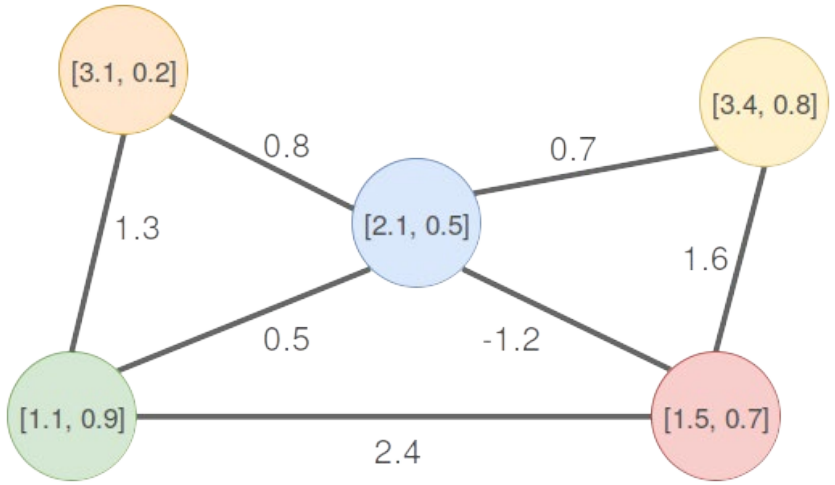




A better representation

Graphs: formalization: recall

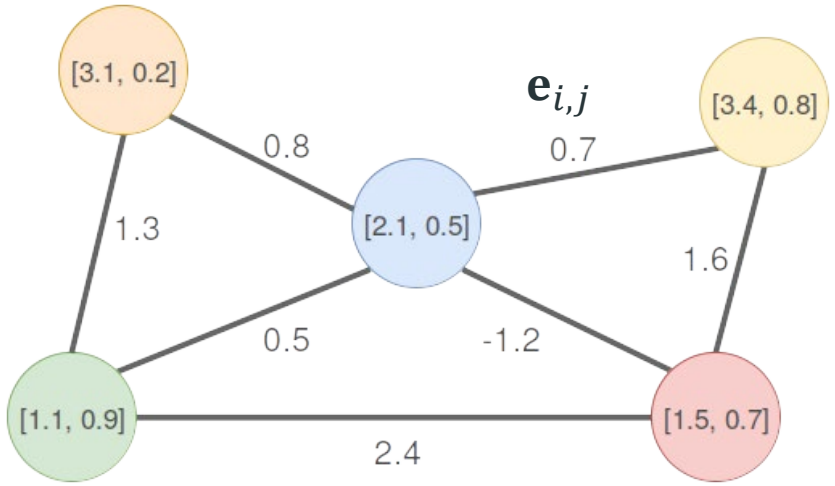
- Let a graph be a tuple $G = (V, E)$ where $V = \{1, \ldots, n\}$ is the *node set* and $E \subseteq V \times V$ is the *edge set*.



- Each vertex can be associated with a real-valued *attribute* (or *feature*) vector. We indicate with $\mathbf{x}_i \in \mathbb{R}^{d_v}$ the d_v-dimensional attribute of the i-th node.

Graphs: formalization

- Edges can also have real-valued features. We indicate with $\mathbf{e}_{i,j} \in \mathbb{R}^{d_e}$ the d_e-dimensional attribute of edge (i, j).



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

- The *adjacency matrix* of a graph is a square matrix $\mathbf{A}$ whose entries are given by

$$\mathbf{A}_{i,j} = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$

Graphs: formalization

- In an *undirected* graph, edges are unordered pairs of vertices. Alternatively, we can say that in an undirected graph $(i, j) \in \Leftrightarrow E(j, i) \in E$.
- The adjacency matrix of an undirected graph is symmetric.
- In this course, we mostly consider undirected graphs.
- The degree of a node in an unweighted graph is the number of nodes attached to it.
- If edges are weighted, then the vector of degrees $\mathbf{d}$ is

$$\mathbf{d} = \mathbf{A1}$$

- Define $\mathbf{D}$ to be a diagonal matrix so that

$$diag(\mathbf{D}) = \mathbf{d}$$

The Laplacian

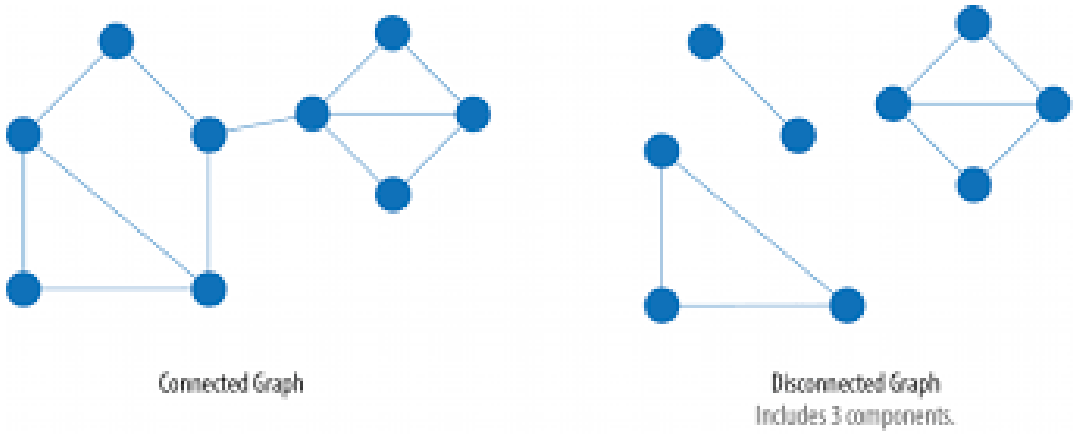
- In addition to the adjacency matrix there are other interesting matrices derived from that.
- The *Laplacian*

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

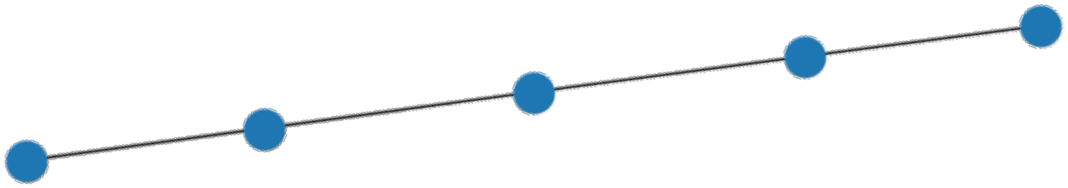
- The Laplacian is
  - Symmetric ($\mathbf{L}$ coincides with its transpose)
  - Semidefinite positive (all its eigenvalues are non-negative)

The Laplacian

- The Laplacian embeds several important properties of the graph:
  - The number of zero eigenvalues of the Laplacian (i.e., the multiplicity of the 0 eigenvalue) equals the number of connected components of the graph.
  - If the graph is fully connected, then there is a single null eigenvalue; its eigenvector is a constant vector for all nodes in the graph.



The case of a *chain graph*

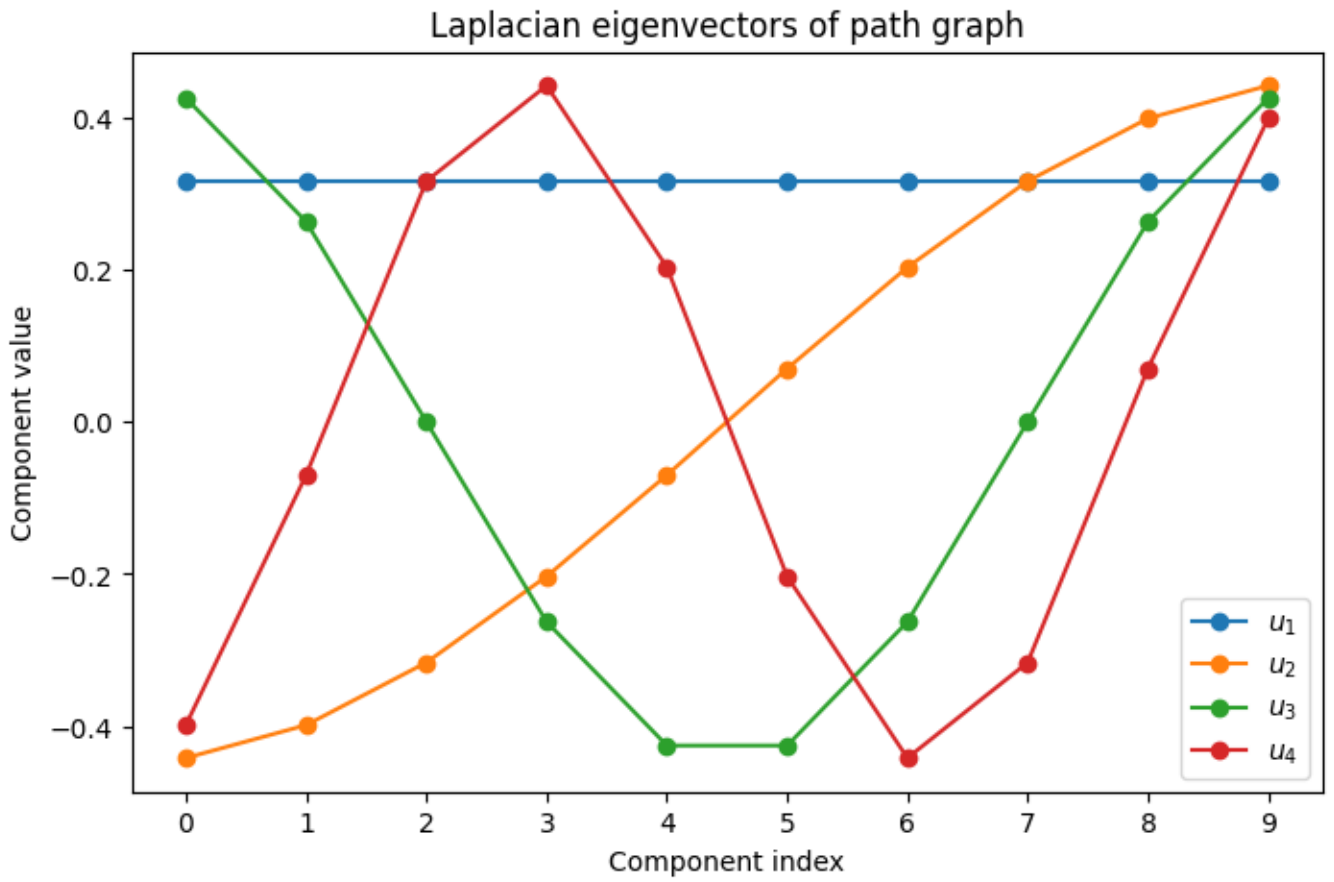


$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

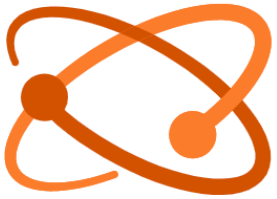
$$L = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Eigenvectors

The plot of eigenvectors resembles that of signal



Our libraries



Torch Spatiotemporal

A library for neural spatiotemporal data processing, with a focus on Graph Neural Networks.

Spektral

A library for building graph neural networks in Keras and Tensorflow.

CDG

A Python library for detecting changes in stationarity in sequences of graphs.

DTS

A Keras library that provides multiple deep architectures for multi-step time-series forecasting.