EE626:
Quick introduction into graph machine learning

Dr Dorina Thanou

18.09.2024

Questions from previous lecture?

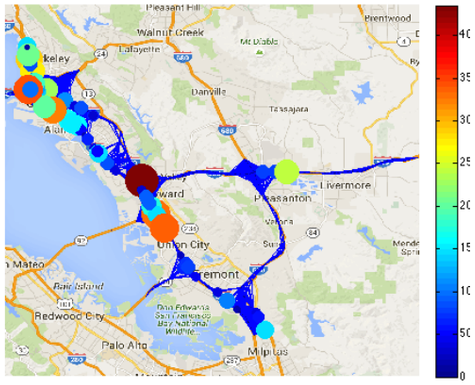
- Link to preferences (please add your name if you haven't done so!):
  - https://docs.google.com/spreadsheets/d/1dFSR_FaHpUtTFW5rA0gnV8qMPXFRbK4y8kYhvaevH6g/edit?usp=sharing

- Volunteers for next week?

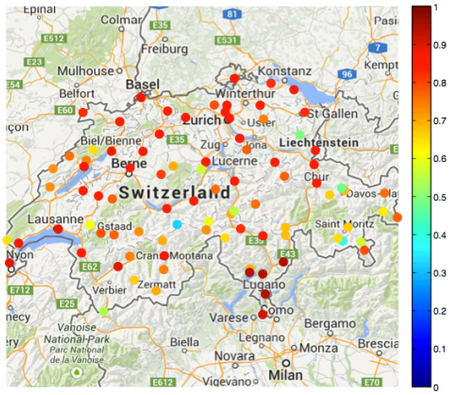
- Any suggestions?

Going beyond graph structure

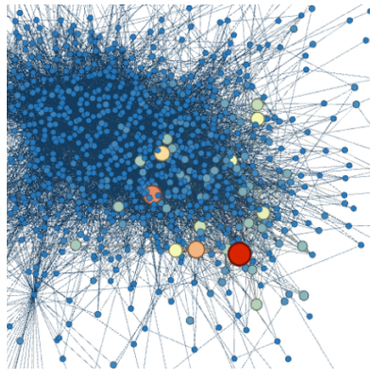
- Very often data comes with additional features
  - Not only graphs, but attributes on the nodes of the graph



Transportation networks

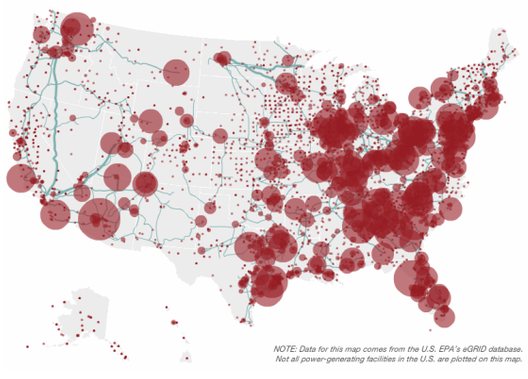


Weather networks

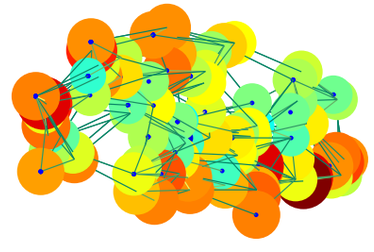


Social networks

Disease spreading networks



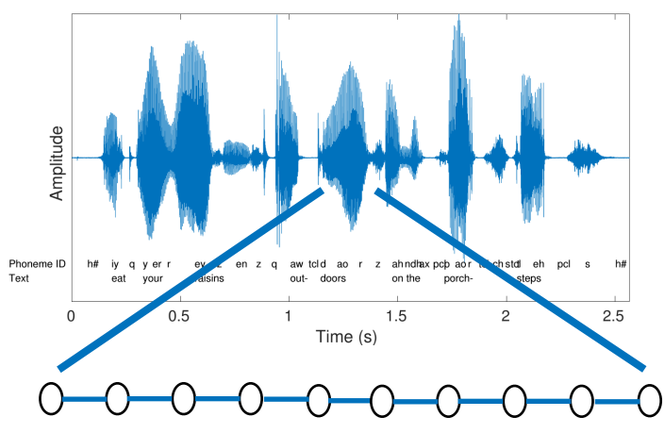
Electric grid networks



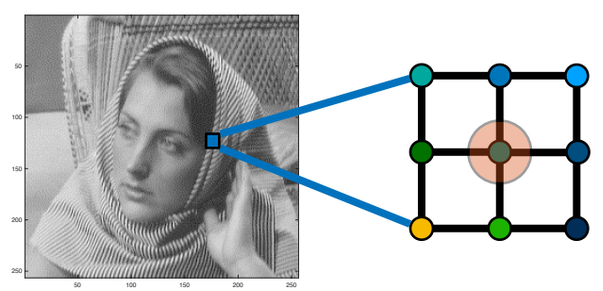
Biological networks

Graph structured data

- Data live on a regular domain

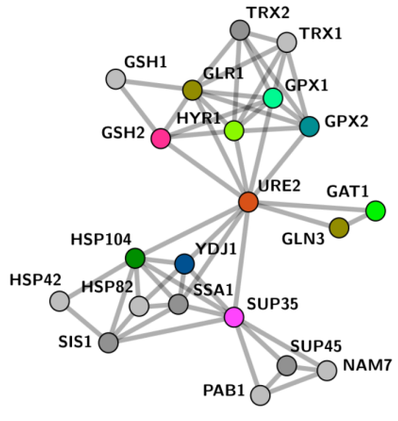


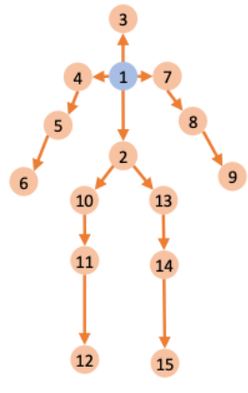
Domain: line



Domain: grid

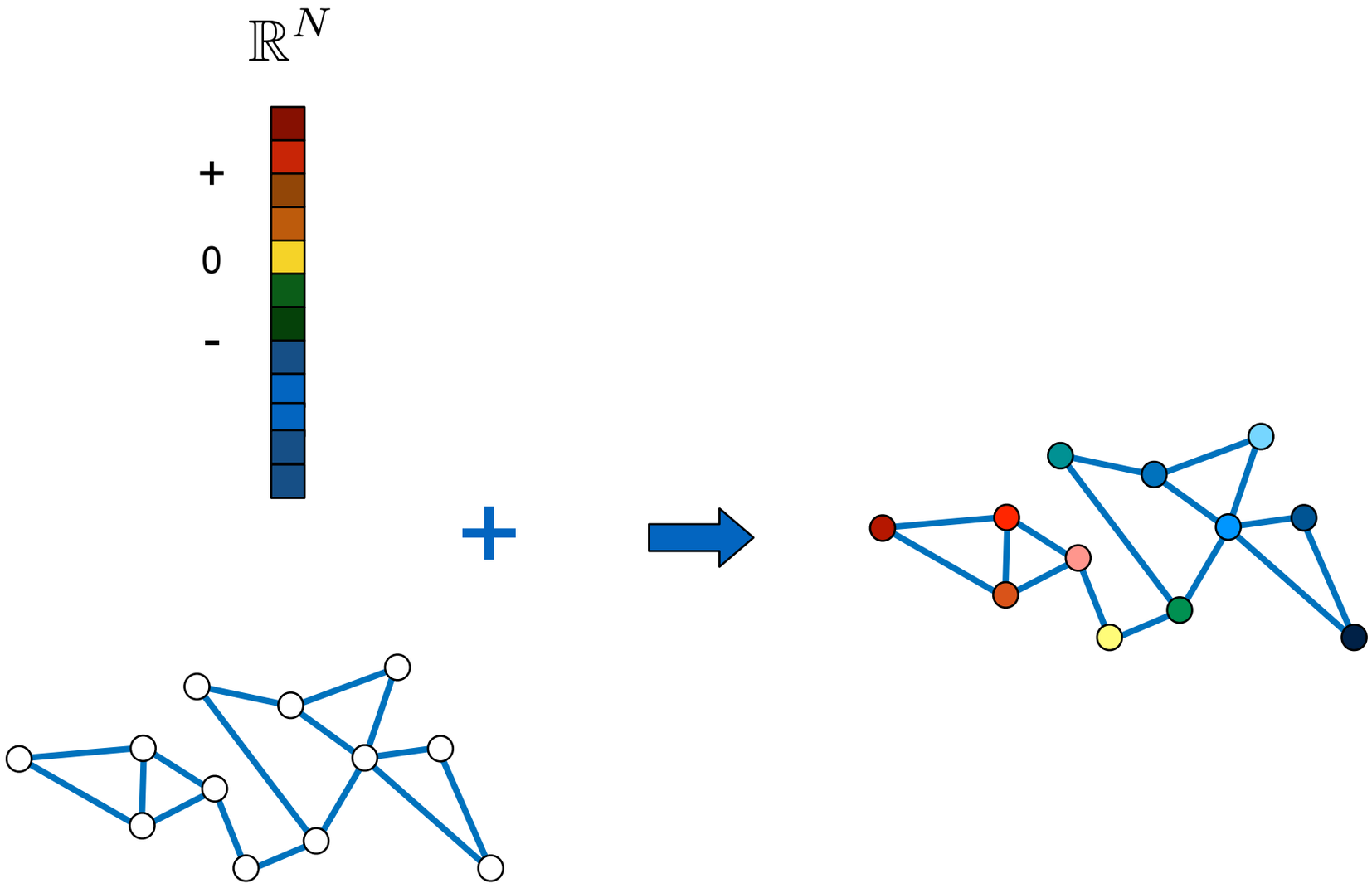
- Weighted graphs capture the geometric structure of complex, i.e., irregular, domains





Domain: irregular graph

Processing graph structured data

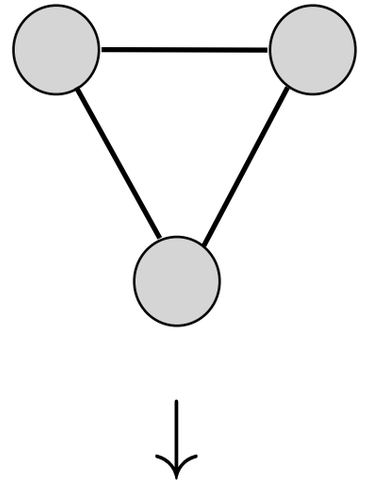


How can we extract useful information by taking into account both **structure (edges)** and **data (values/features on vertices)**?

Recap of classical graph matrices

- Undirected graph of N nodes, i.e., $|\mathcal{V}| = N$:

$$G = (\mathcal{V}, \mathcal{E}, W), \quad \mathcal{E} \subseteq \{(i,j) : i,j \in \mathcal{V}\}, \quad (i,j) = (j,i)$$



- Adjacency matrix or weight matrix :

$$W_{ij} = \begin{cases} w_{ij}, & \text{if } (i,j) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}$$

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

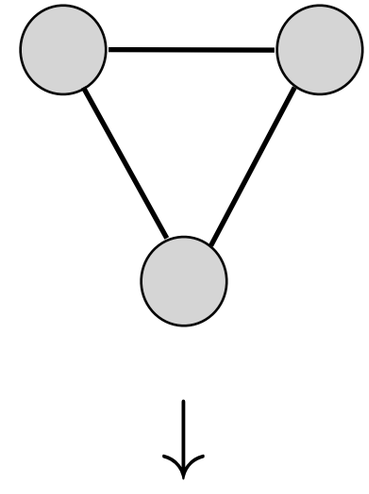
- If the graph is unweighted (often denoted as A) :

$$W_{ij} = \begin{cases} 1, & \text{if } (i,j) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}$$

Recap of classical graph matrices

- Neighborhood of node i : Set of nodes connected to node i by an edge

$$\mathcal{N}_i = \{j : (i,j) \in \mathcal{E}\}$$



- Degree of a node i : It is the sum of the weights of the edges incident to node i

$$D_i = \sum_{j \in \mathcal{N}_i} W_{ij}$$

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

- Degree matrix: A diagonal matrix containing the degree of each node

$$D_{ij} = \begin{cases} \sum_j W_{ij}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

$$D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

The graph Laplacian matrix

- The combinatorial Laplacian is defined as:

$$L = D - W$$

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

$$\downarrow$$

$$L\mathbf{1} = \mathbf{0}$$

  - Symmetric
  - Off-diagonal entries non-positive
  - Rows sum up to zero

- It is a positive semi-definite matrix:
  - For each function $f : \mathcal{V} \to \mathbb{R}$, where f_i is the value on the i^{th} node of the graph:

$$f^T L f = f^T (D - W) f = \sum_{i=1}^{N} D_{ii} f_i^2 - \sum_{i,j=1}^{N} f_i f_j W_{ij}$$

$$= \frac{1}{2} \sum_{i,j=1}^{N} W_{ij} (f_i - f_j)^2 \geq 0, \quad \forall f \in \mathbb{R}^N$$

Connection to continuous

- Graph Laplacian: A discrete differential operator

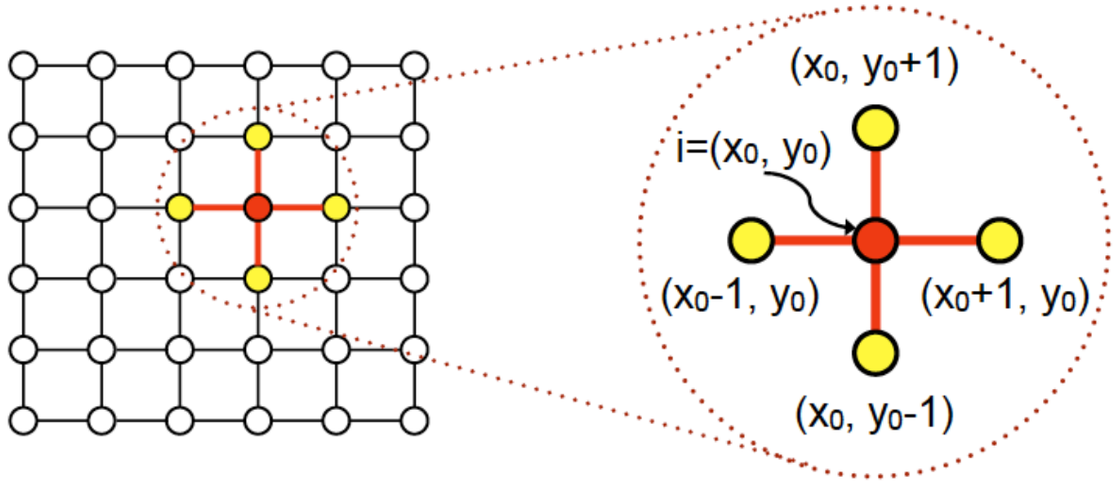
$$\big(Lf\big)(i) = \sum_{j \in \mathcal{N}_i} W_{i,j}(f_i - f_j)$$

- The Laplace operator:
  - A second-order differential operator: divergence of the gradient $\Delta f = \nabla^2 f$
  - The gradient is defined as: $\nabla f = (\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_N})$
  - Finally, the Laplacian is: $\Delta f = \sum_{i=1}^{N} \frac{\partial^2 f}{\partial x_i^2}$

- The Laplacian matrix is the graph analogue to the Laplace operator on continuous functions!

Illustrative example

- Example: Unweighted grid graph



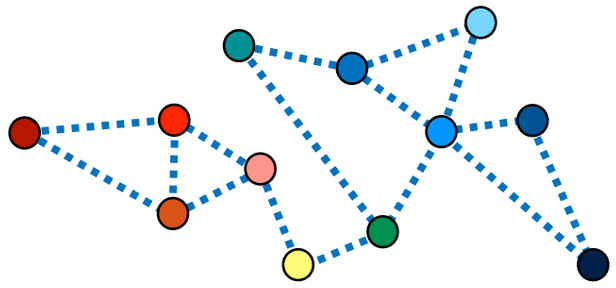
$$L = D - W$$

$$\begin{aligned} -Lf(i) = & \left[f(x_0+1, y_0) - f(x_0, y_0)\right] - \left[f(x_0, y_0) - f(x_0-1, y_0)\right] \\ & + \left[f(x_0, y_0+1) - f(x_0, y_0)\right] - \left[f(x_0, y_0) - f(x_0, y_0-1)\right] \end{aligned}$$

$$\sim \frac{\partial^2 f}{\partial x^2}(x_0, y_0) + \frac{\partial^2 f}{\partial y^2}(x_0, y_0) = (\Delta f)(x_0, y_0)$$

Signal on the graph or graph signal

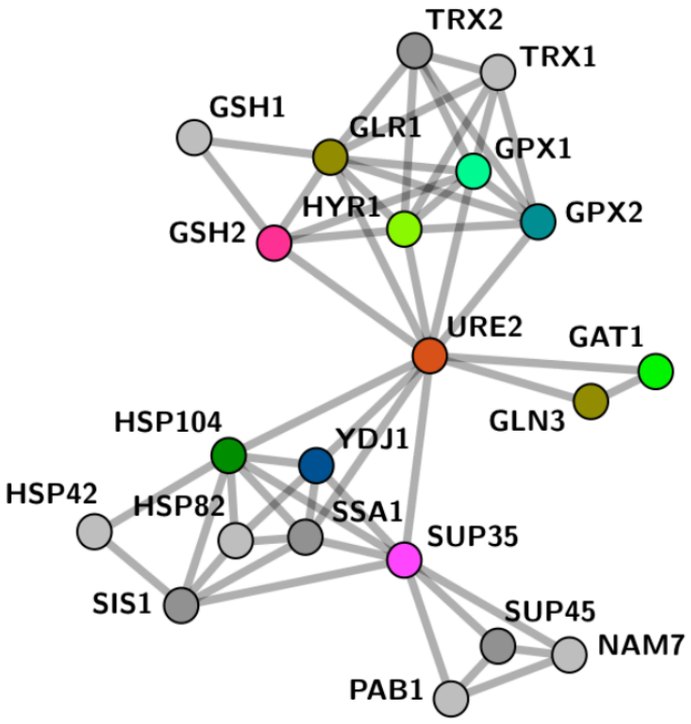
- A function $f : \mathcal{V} \rightarrow \mathbb{R}^N$ that assigns real values to each vertex of the graph
- It is defined on the vertices of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$

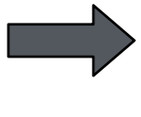


- Often represented as a vector $f \in \mathbb{R}^N$, where $f(i)$ is the signal value at node i
- The ordering of the vector follows the ordering of the adjacency matrix

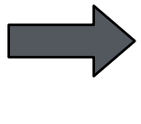
In this lecture…

- How can we infer useful information from graph structured data?





Machine learning (ML)/ Inference on graphs



Actionable knowledge

e.g., node/graph classification,
signal inpainting/denoising, link prediction

$$X, \mathcal{G} \quad \Rightarrow \quad f(X, \mathcal{G}) \quad \Rightarrow \quad Y$$

Outline

- Traditional ML on graphs
  - Graph-based feature engineering

- Recent ML on graphs
  - Feature learning on graphs

Graph-structured features/embeddings: A high level overview

- **Hand-crafted features:** Capture some structural properties of the graph, followed by some statistics (signatures)
- **Graph kernel methods:** Design similarity functions in an embedding space
- **Spectral features:** Capture the graph properties through spectral graph theory

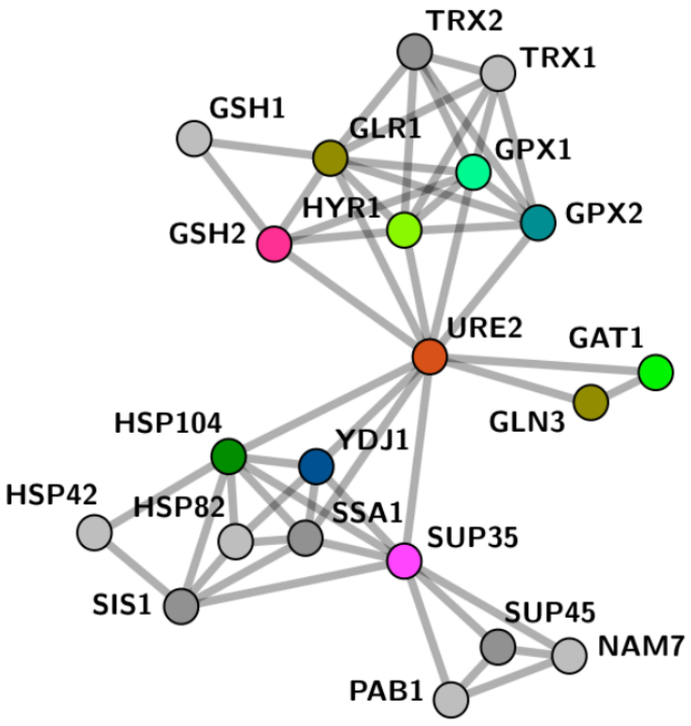
Model-driven

- **Learned features:** Learn graph features directly from data by designing models based on meaningful assumptions
  - **Unsupervised (shallow) embeddings:** Learn features based on different ways of preserving information from the original graph (often without node attributes)
  - **Graph neural network features:** Learn features from the data using a well-designed family of neural networks (often with node attributes)

Data-driven

Traditional ML pipeline on graphs

- How can we learn useful information from graph structured data?

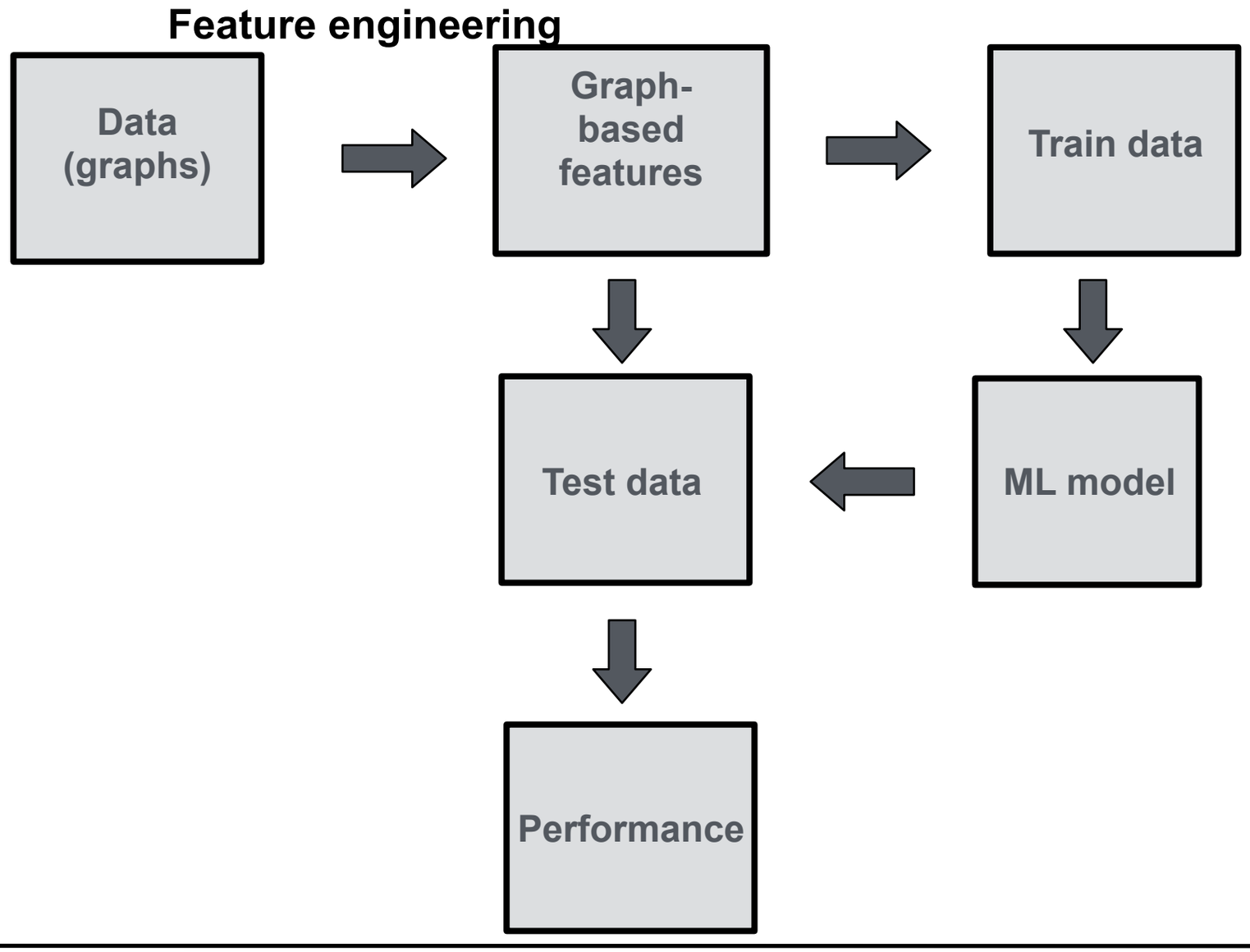


e.g., node/graph classification, signal inpainting/denoising

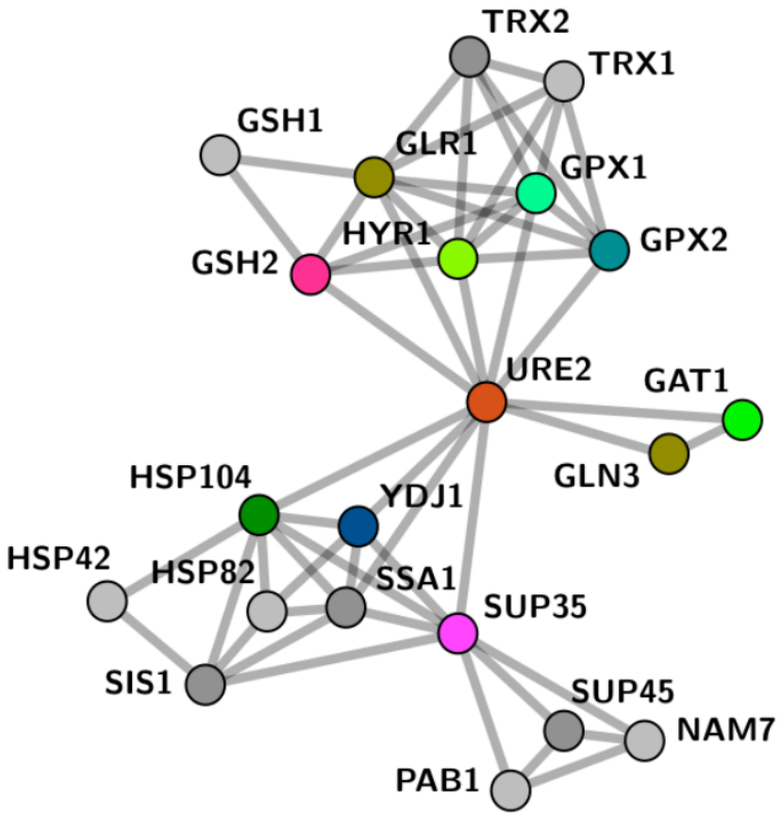
$$X, \mathcal{G} \quad \Rightarrow \quad \phi(X, \mathcal{G}) \quad \Rightarrow \quad f(\phi(X, \mathcal{G})) \quad \Rightarrow \quad Y$$

Traditional ML pipeline on graphs

- Feature engineering is a way of extracting meaningful information from graphs



Traditional ML pipeline: Input



- Input:
  - Graph: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
  - Graph with attributes: $\mathcal{G}, X$

$X, \mathcal{G}$

Traditional ML pipeline: Features

- Should reveal important information regarding the graph structure

- Key to achieving good model performance

Graph-based features

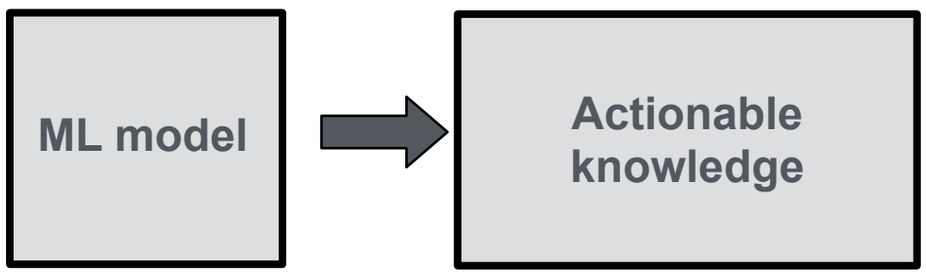
- Features can be defined at different scales
  - At a node, edge, sets of nodes, entire graph level

$\phi(X, \mathcal{G})$

- The choice of the features depends on
  - the end task
  - prior knowledge on the data

Traditional ML pipeline: Learning tasks

- The features are given as input to an ML model
  - Examples: logistic regression, SVM, neural networks, etc.



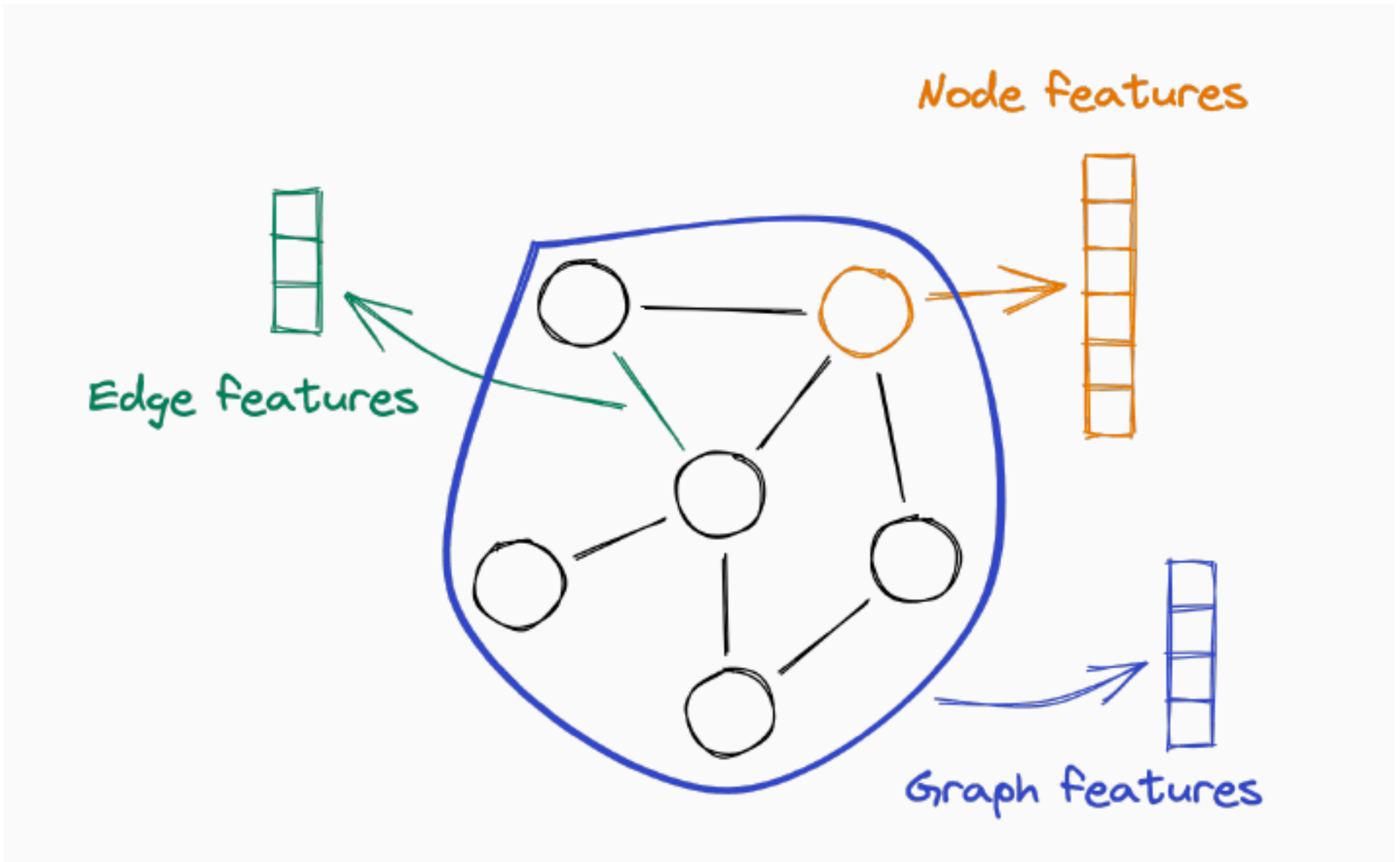
e.g., node/graph classification, signal inpainting/denoising

- Training phase:
  - Given a set of graph-based features, train a model f that predicts the correct Y

$$f(\phi(X, \mathcal{G})) \rightarrow Y$$

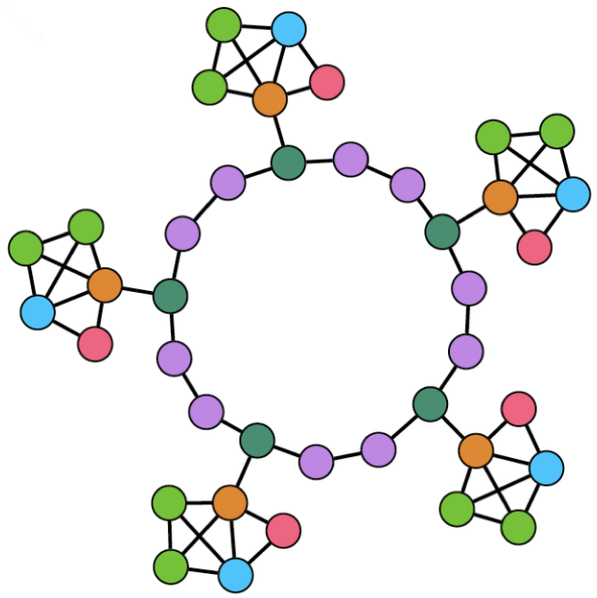
- Testing phase:
  - Given a new node/link/graph, compute its features, and give them as an input to f to make a prediction

Extracting information at different levels



Node level features

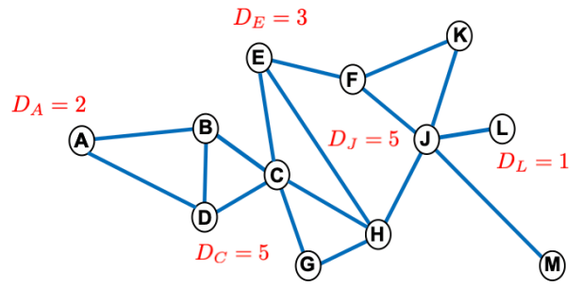
- Typically useful for node classification/clustering tasks



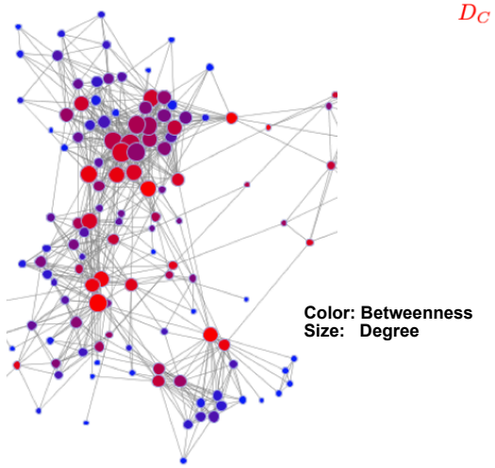
- Aim at characterizing the structure and position of a node in the network

Common node level features

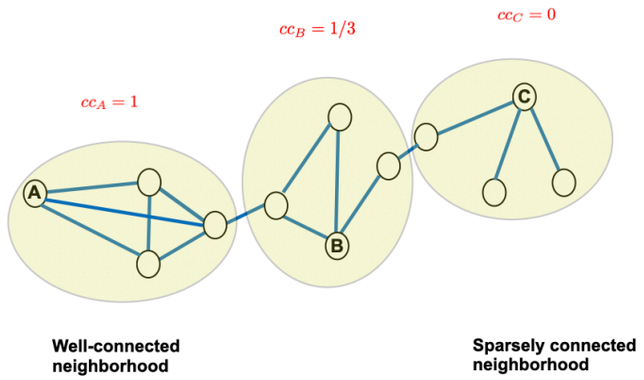
- Node degree



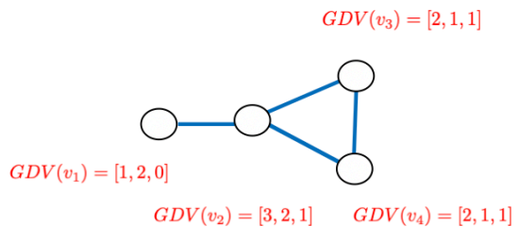
- Node centrality



- Clustering coefficient

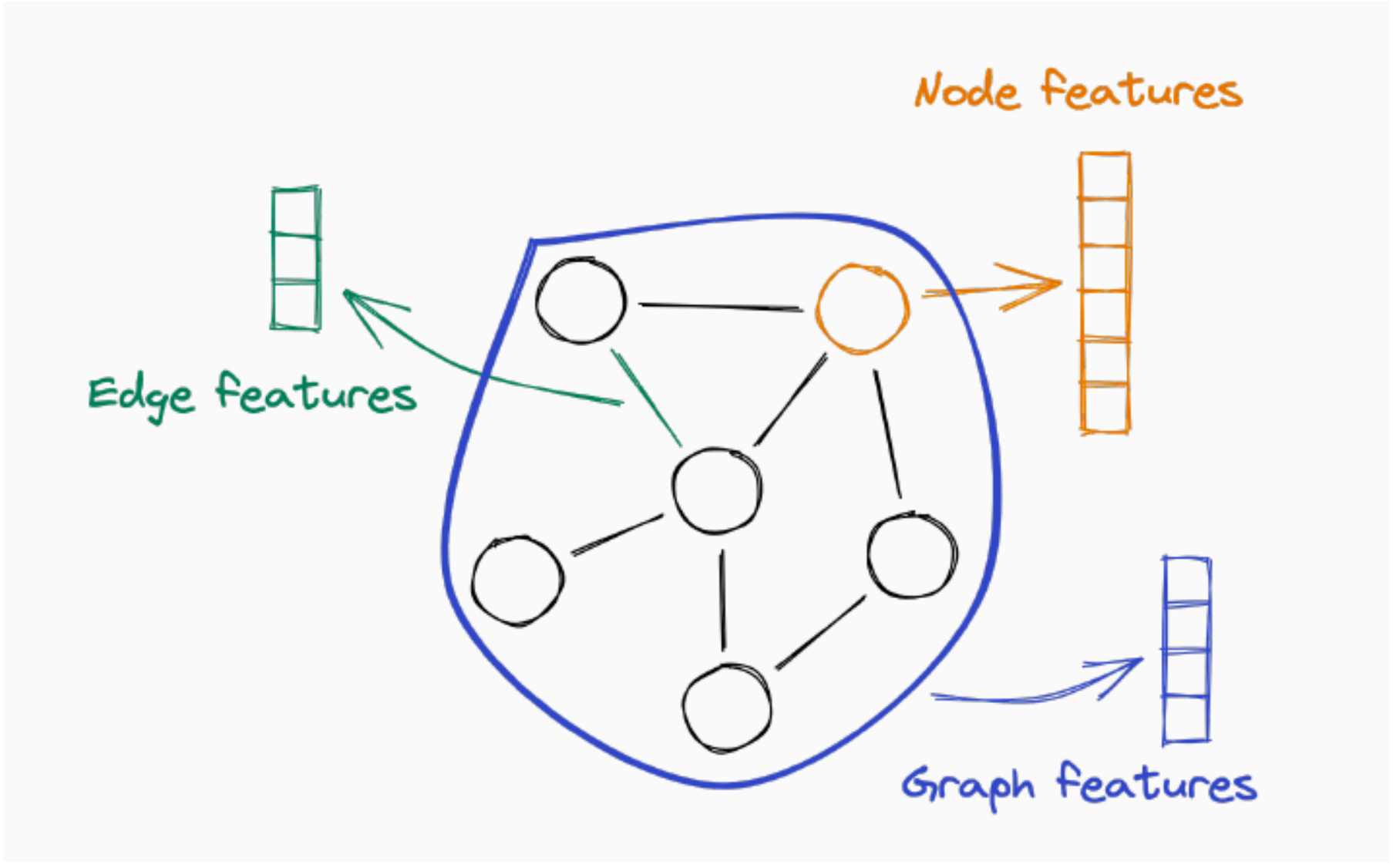


- Graphlets



Possible graphlets:

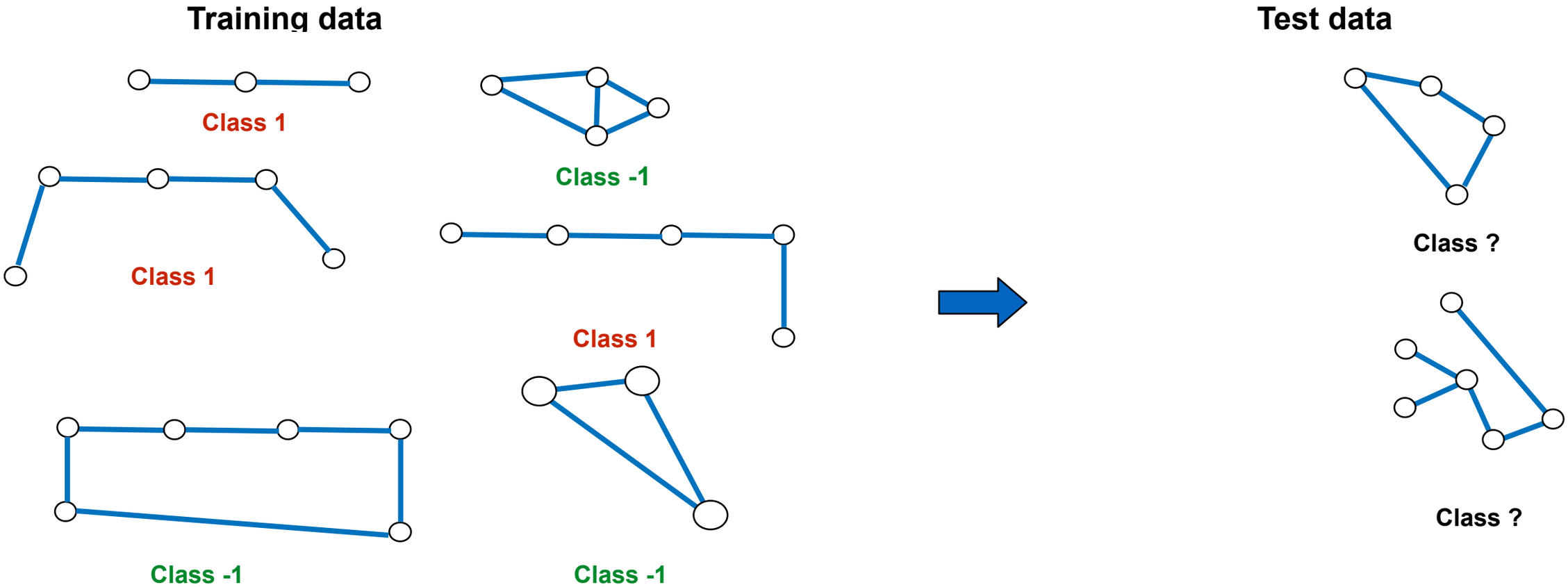
From node level to graph level task



How can we design features that characterize the structure of the entire graph?

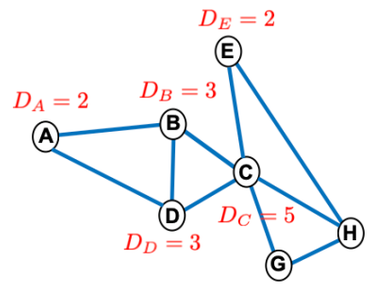
Illustrative example: Graph classification

- Common assumption: Graphs with similar structure have similar label

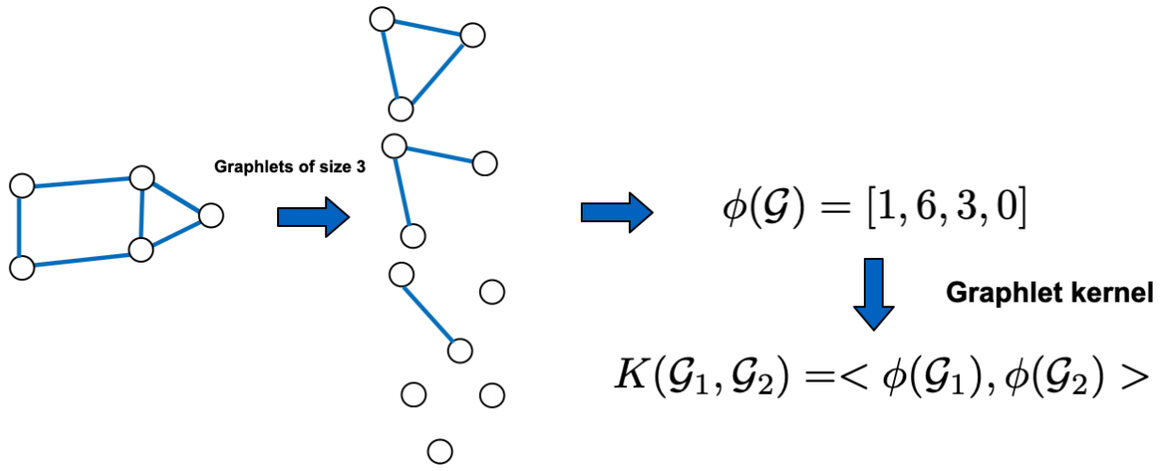


Graph level features

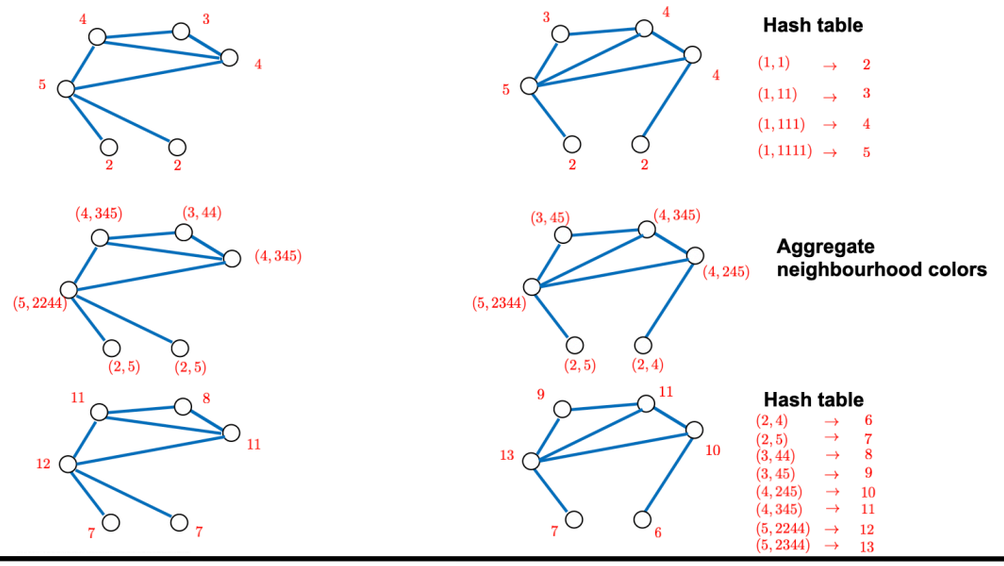
- Bag of nodes



- Graphlet kernel



- The Weisfeiler-Lehman kernel



Limitations of hand-crafted graph features

- Hand-engineered features are defined a priori: no adaptation to the data

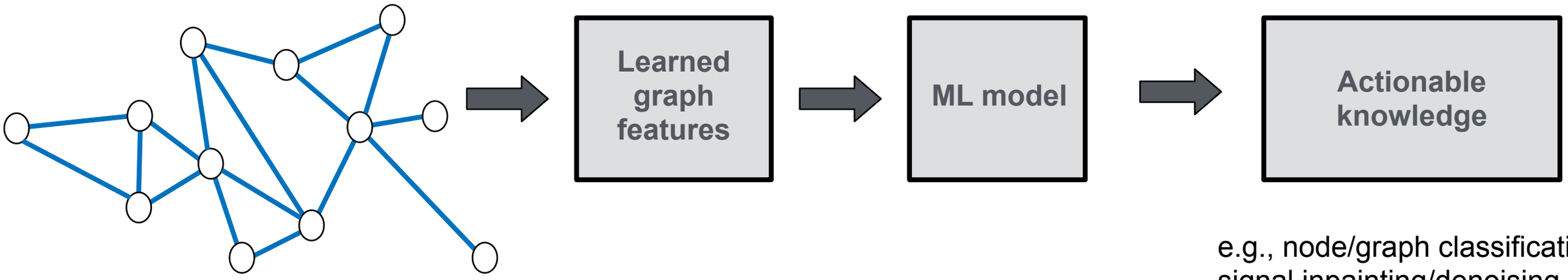
- Designing graph features can very often be a time consuming and expensive process

- Not easy to incorporate additional features on the nodes

- More flexibility can be achieved with an end-to-end learning pipeline

Graph representation learning

- **Intuition:** Optimize the feature extraction part by adapting it to the specific instances of the graphs/data



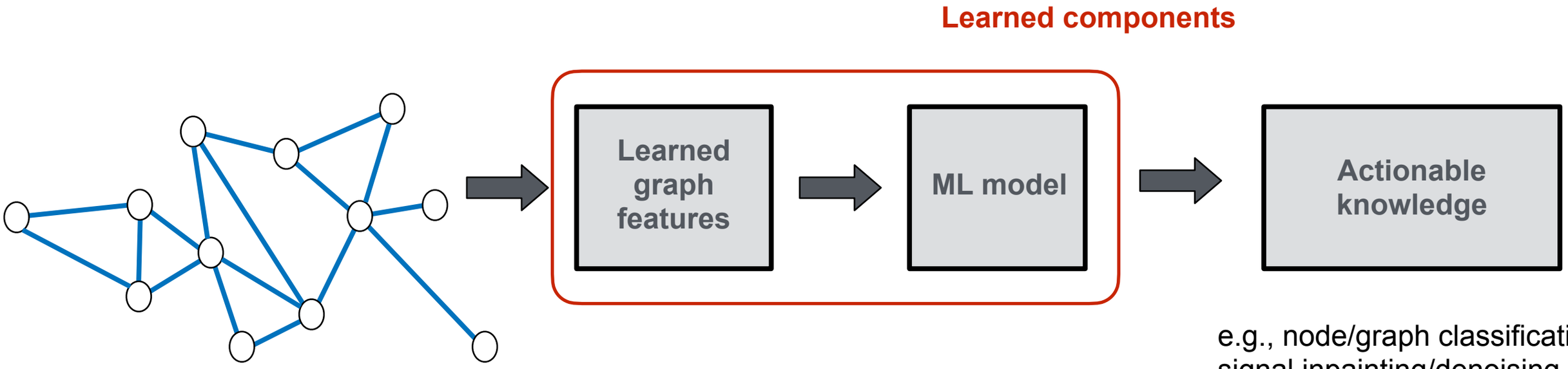
e.g., node/graph classification, signal inpainting/denoising

Feature Learning

$$\mathcal{G} \rightarrow \phi(\mathcal{G}) \rightarrow f(\phi(\mathcal{G})) \rightarrow Y$$

Graph representation learning

- **Intuition:** Optimize the feature extraction part by adapting it to the specific instances of the graphs/data



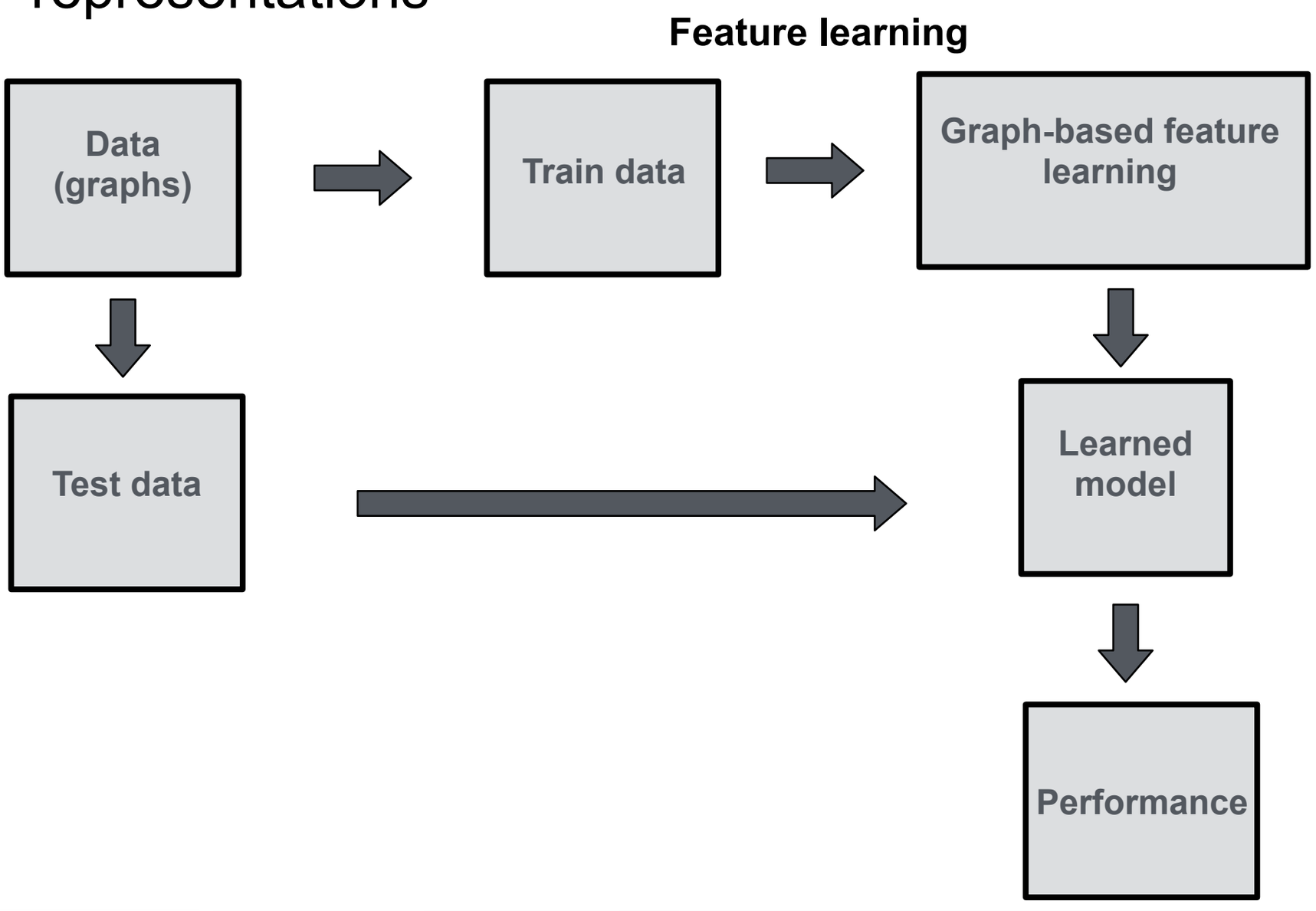
e.g., node/graph classification, signal inpainting/denoising

Feature Learning

$$\mathcal{G} \rightarrow \phi(\mathcal{G}) \rightarrow f(\phi(\mathcal{G})) \rightarrow Y$$

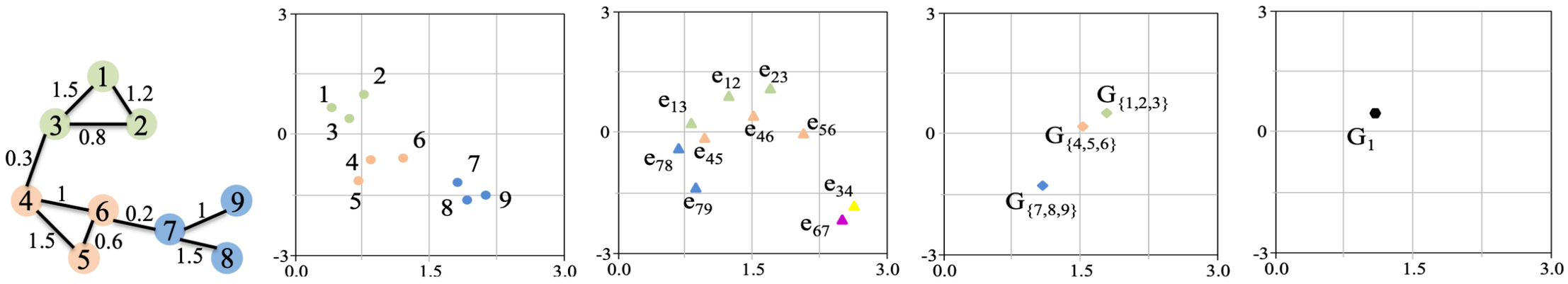
Graph representation learning: basic pipeline

- Feature learning is a way of extracting data-adaptive graph representations



Learning features on graphs

- Learned features convert the graph data in a (low dimensional) latent space (i.e., **embedding space)** where hidden/discriminative information about data is revealed



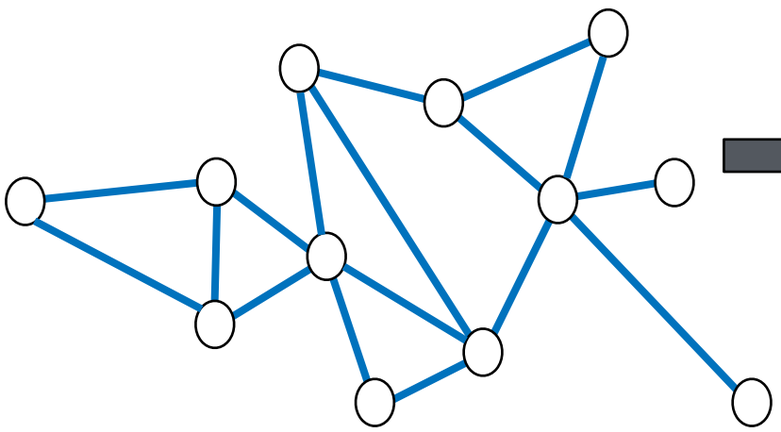
Original space | **Node embedding** | **Edge embedding** | **Subgraph embedding** | **Graph embedding**

Embedding space

How can we learn the embedding space?

Supervised graph representation learning

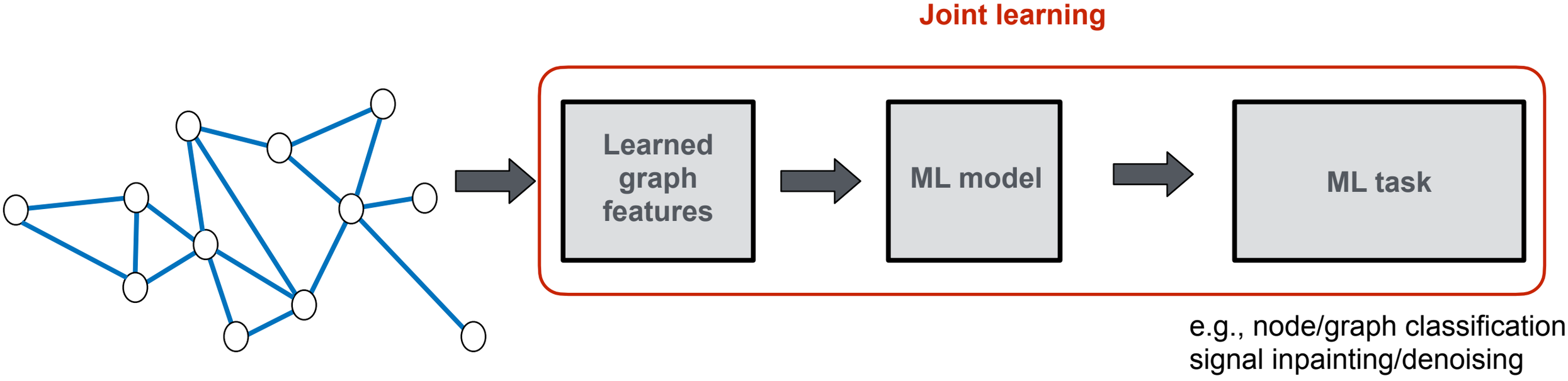
- Learn low-dimensional embeddings for a specific downstream task, e.g., node or graph classification



e.g., node/graph classification, signal inpainting/denoising

Supervised graph representation learning

- Learn low-dimensional embeddings for a specific downstream task, e.g., node or graph classification

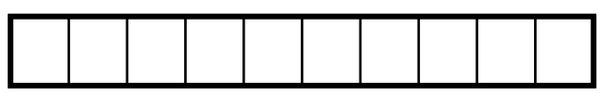


Unsupervised graph representation learning

- Representations are not optimized for a specific downstream task
  - They are optimized with respect to some notion of “closeness” in the graph
  - The notion of “closeness” defines the design of the embedding algorithm

- Potentially used for many downstream inference tasks

Learned embedding vector

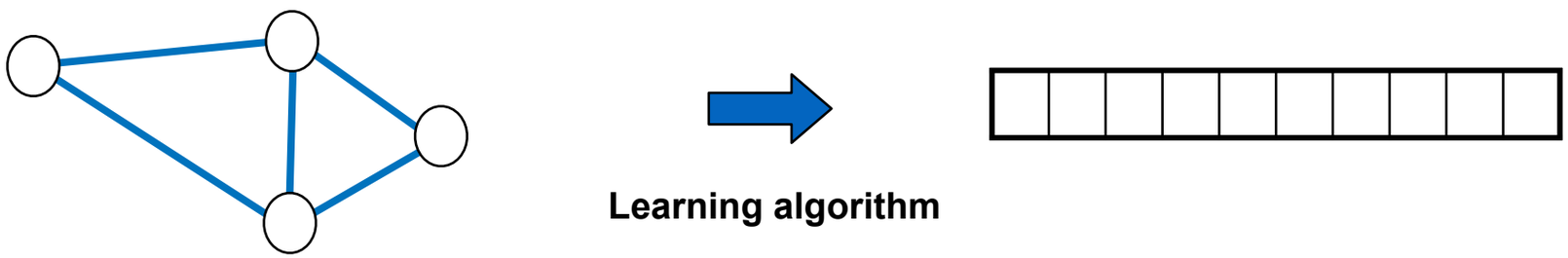


Example of tasks

1. Node/graph classification
2. Node/graph clustering
3. Link prediction
4. Visualization
5. …

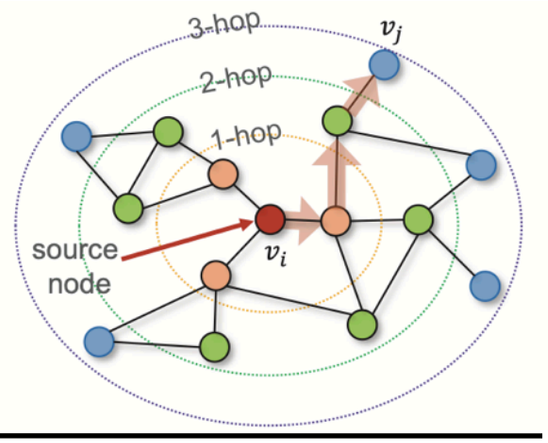
Embeddings on graphs: Definition

- Given an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$, and a predefined dimensionality of the embedding $d << |\mathcal{V}|$, the goal is to convert $\mathcal{G}$ (or a subgraph, or a node) into a $d-$dimensional space in which graph properties are preserved

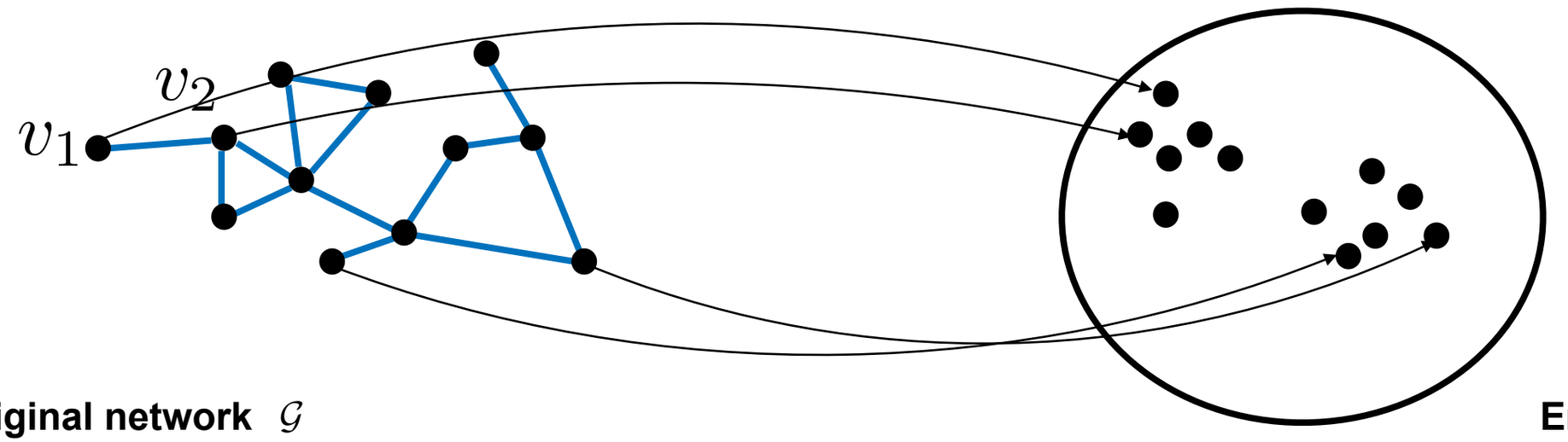


$\mathcal{G}$

- Graph properties can be quantified using proximity measures on the graph (e.g., $K-$hop neighborhood)



Illustrative example: Node embeddings



Original network $\mathcal{G}$

Embedding space Y

What is the similarity in the graph that should be preserved in the embedding space?

$$sim_{\mathcal{G}}(v_1, v_2) \approx sim_{Y}(Y_1, Y_2)$$

Example: Laplacian Eigenmaps

- **Intuition:** Preserve pairwise node similarities derived from the adjacency/weight matrix

$$sim_{\mathcal{G}}(v_i, v_j) = W_{ij}$$

- Measure similarity in the embedding space using the mean square error

$$sim_Y(Y_i, Y_j) = \|Y_i - Y_j\|_2^2$$

- Impose larger penalty if two nodes with larger pairwise similarity are embedded far apart

$$\begin{aligned} l(sim_{\mathcal{G}}(v_i, v_j), sim_Y(Y_i, Y_j)) &= sim_{\mathcal{G}}(v_i, v_j) \cdot sim_Y(Y_i, Y_j) \\ &= W_{ij}\|Y_i - Y_j\|_2^2 \end{aligned}$$

Laplacian Eigenmaps - algorithm

- Compute embeddings that minimize the expected square distance between connected nodes

Centered embeddings

Uncorrelated embedding coordinates

$$\min_{Y \in \mathbb{R}^{N \times K}: Y^T 1 = 0; Y^T Y = I_K} \sum_{(i,j) \in \mathcal{E}} W_{ij} \|Y_i - Y_j\|^2$$

$$\Downarrow \quad L = D - W$$

$$\min_{Y \in \mathbb{R}^{N \times K}: Y^T 1 = 0; Y^T Y = I_K} \operatorname{tr}(Y^T L Y)$$

Graph smoothness

$$\Downarrow$$ **Lagrangian**

$$\min_{Y \in \mathbb{R}^{N \times K}; Y^T 1 = 0} \operatorname{tr}(Y^T L Y - (Y^T Y - I_K)\Gamma)$$

$$\Downarrow$$ **Gradient**

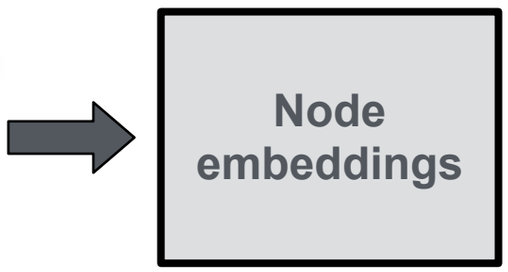
$$LY = Y\Gamma \quad \Rightarrow \quad \boxed{u_i \rightarrow (\chi_2(i), ..., \chi_{K+1}(i))}$$

Laplacian Eigenmaps: K first non-trivial eigenvectors of the Laplacian!

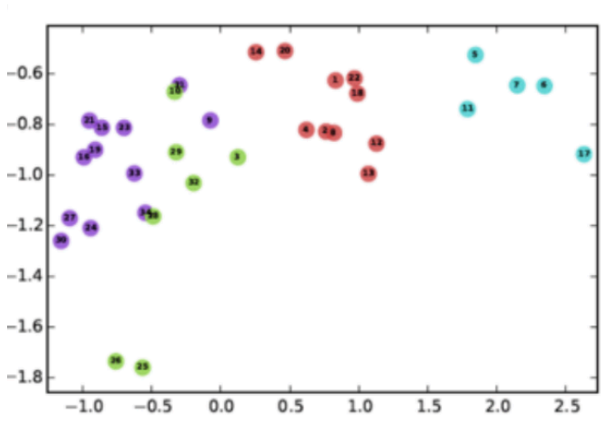
[Belkin et al, 2003, Laplacian Eigenmaps for Dimensionality Reduction and Data Representation, Neural Comp.]

Application of node embeddings: Node clustering/Community

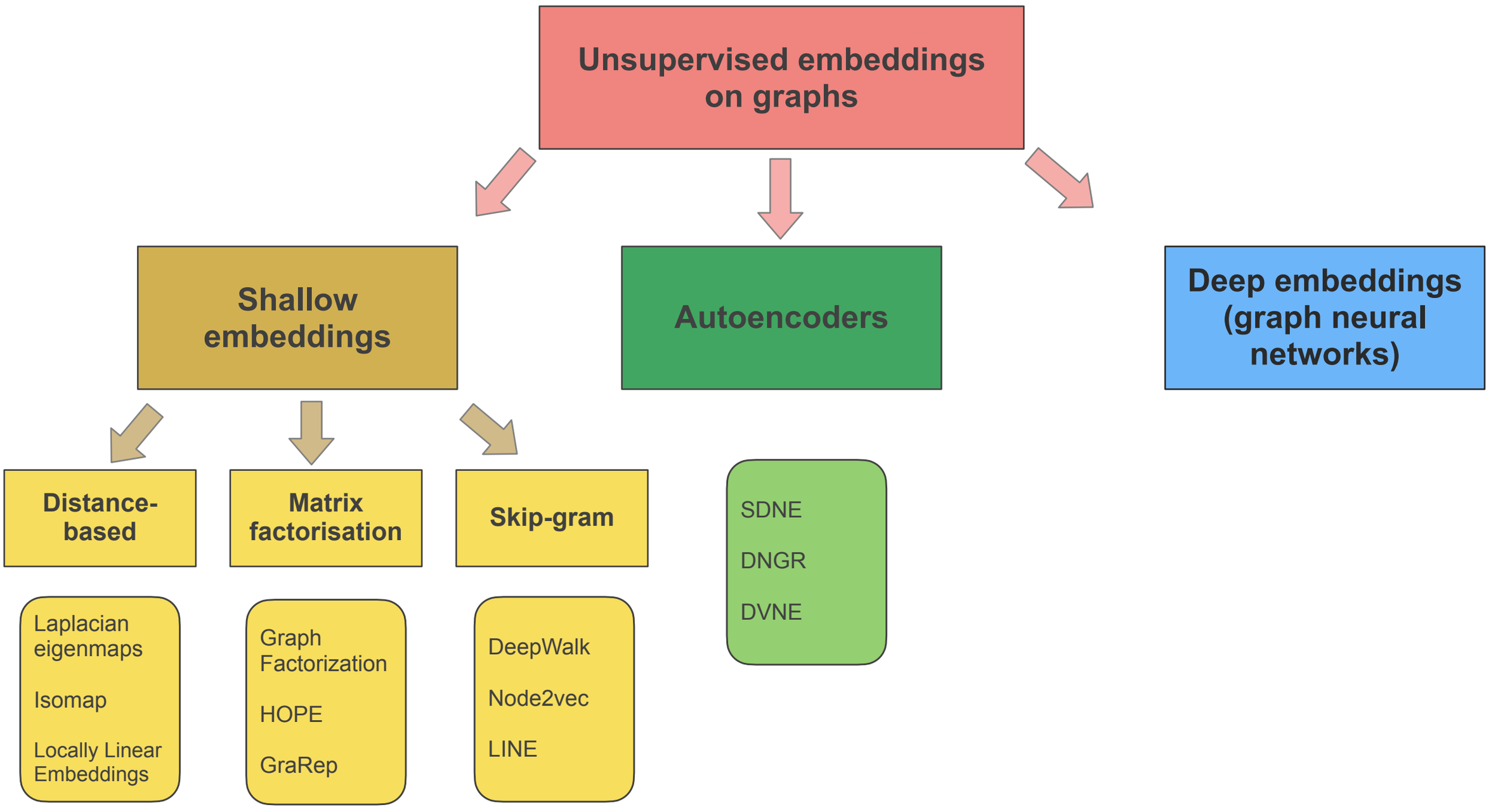
- The karate-club example
  - Compute node embeddings
  - Apply any clustering algorithm (e.g., K-means) on the learned embeddings



e.g., K-means



Learning unsupervised embeddings on graphs: A (partial) taxonomy



Summary so far

- Feature learning on graphs is a data-driven (and ofter more flexible) alternative to designing hand-crafted features

- Unsupervised learning on graphs provides representations i.e., embeddings, that are not adapted to specific tasks

- Different assumptions lead to different ways of preserving information from the original graph in the embedding space (e.g., weight matrix, random walks…)

- The choice of what structure information to preserve depends on the application

Limitations of the (discussed) node embedding algorithms

- Usually transductive not inductive
  - Learned embedding models often do not generalize to new nodes

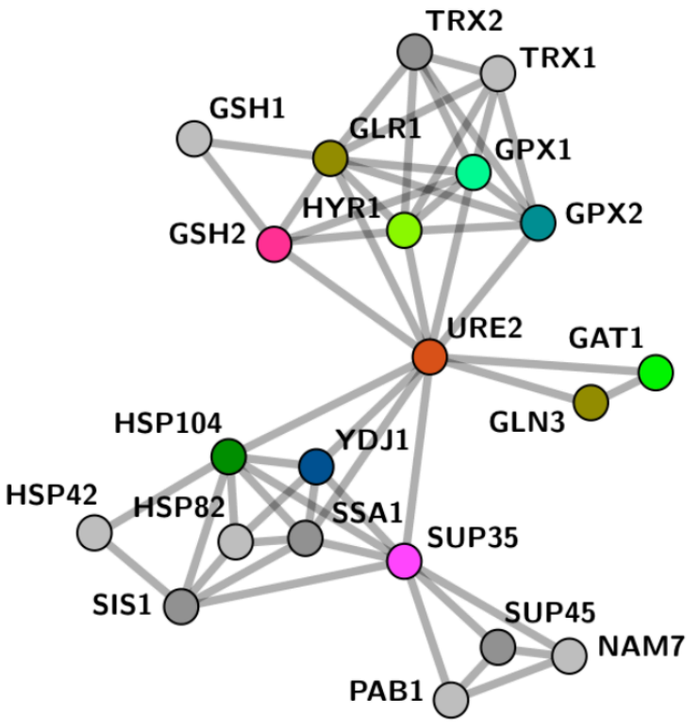
- Do not incorporate node attributes

- Independent of downstream tasks

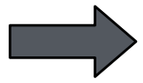
- No parameter sharing:
  - Every node has its own unique embedding

Graph neural networks (GNNs)

- A different way of obtaining ‘deeper’ embeddings inspired by deep learning
- They generalize to graphs with node attributes



Graph neural networks

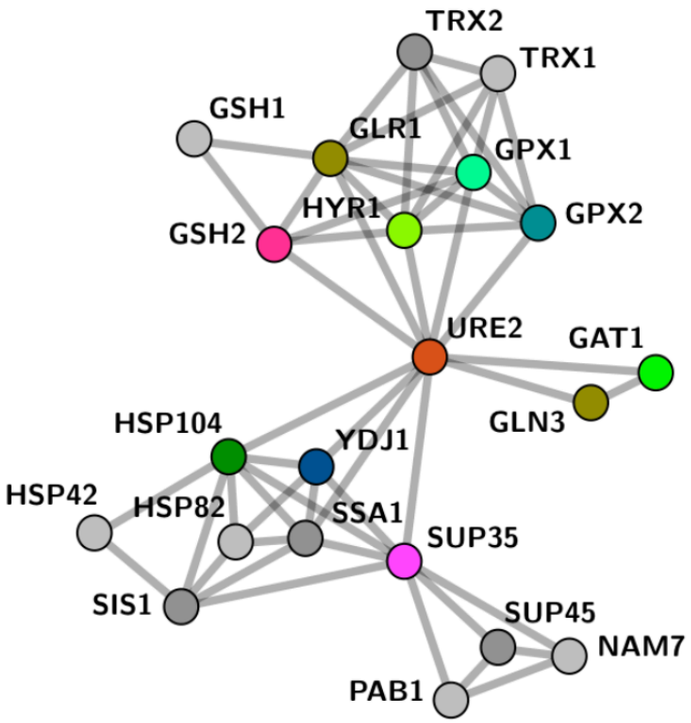


Actionable knowledge

e.g., node/graph classification, signal inpainting/denoising

Graph neural networks (GNNs)

- A different way of obtaining ‘deeper’ embeddings inspired by deep learning
- They generalize to graphs with node attributes



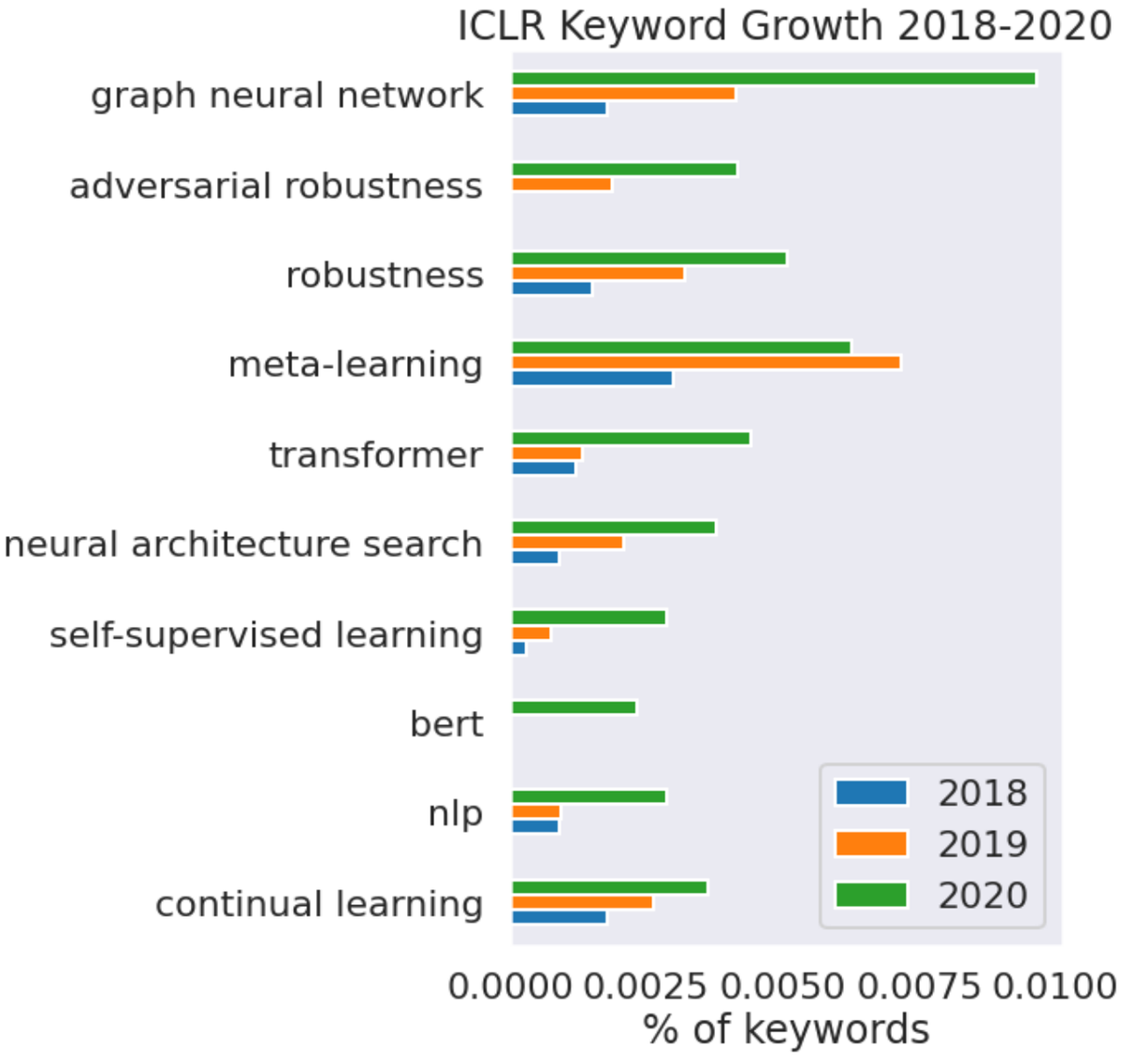
Learned components

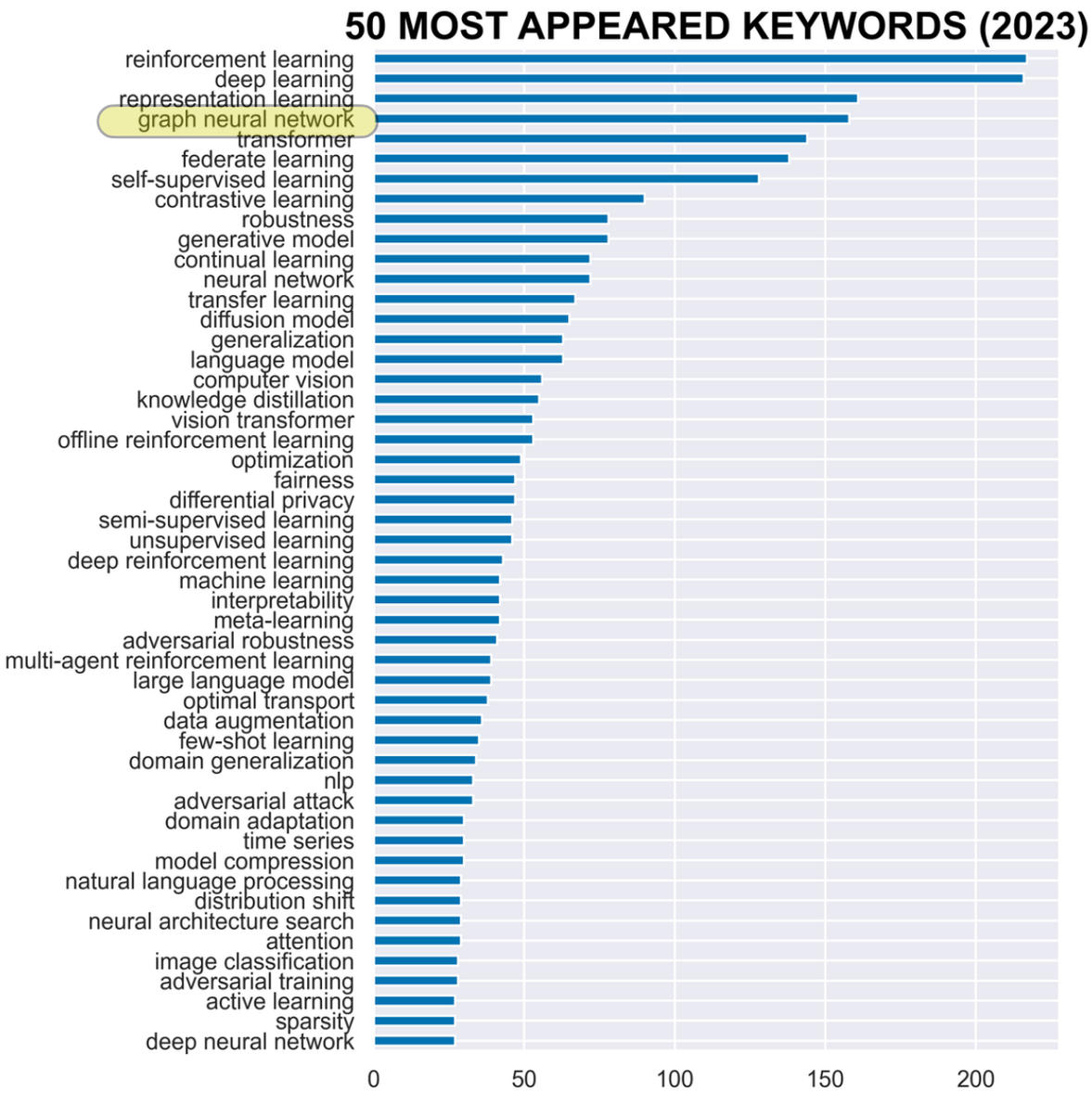
Graph neural networks

Actionable knowledge

e.g., node/graph classification, signal inpainting/denoising

GNNs: A growing trend

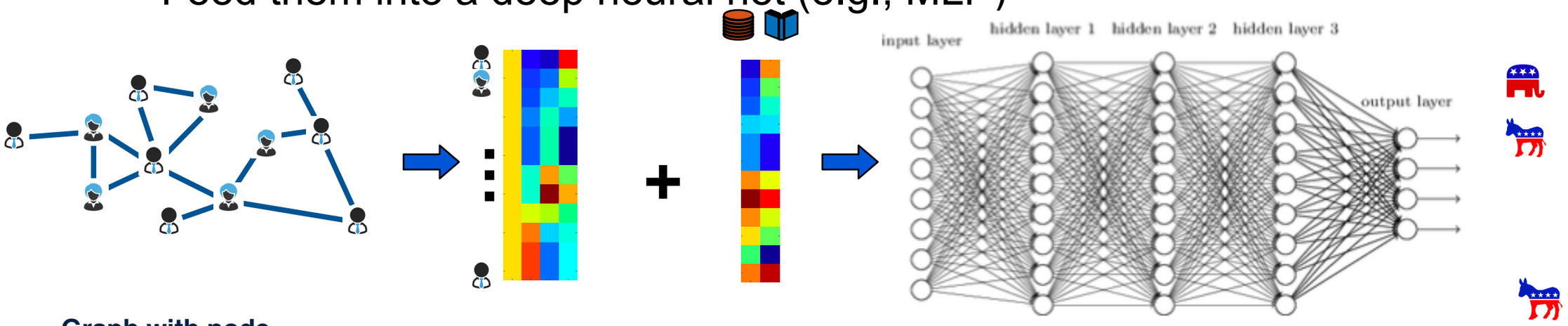




ICLR 2023

Computing embeddings from

- A naive approach:
  - Embed graph and node attributes into a Euclidean space
  - Feed them into a deep neural net (e.g., MLP)



Graph with node attributes | Node embedding | Node attributes | Predicted labels

- Issues with that:
  - Computationally expensive
  - Not applicable to graphs of different sizes
  - Not invariant to node ordering: if we reorder nodes the representations will be different

Can we do better? Yes!

Good priors are key to learning

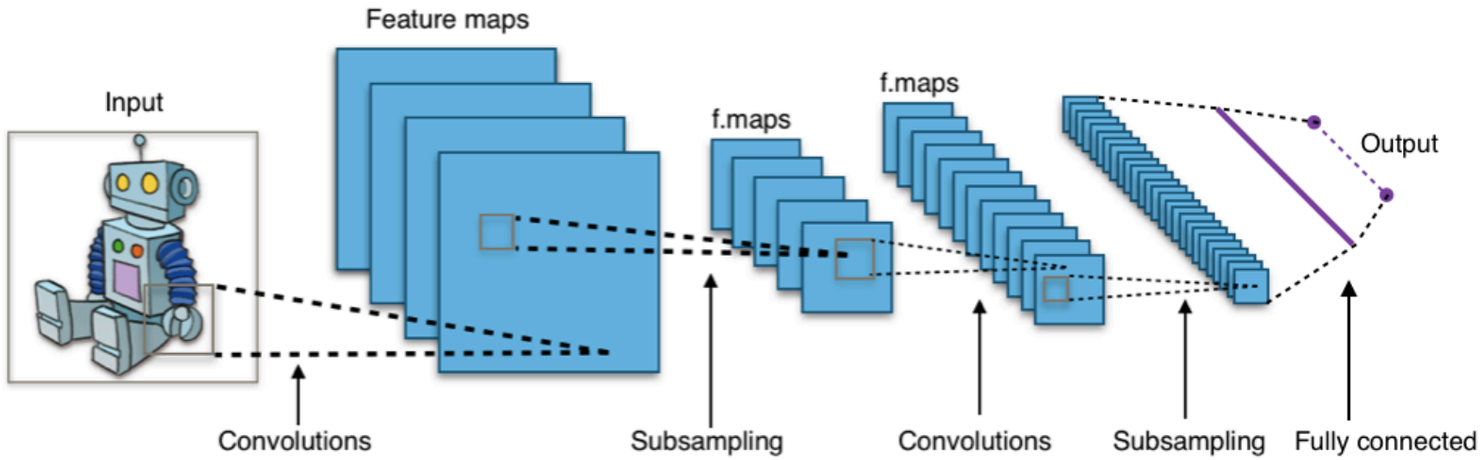
- We build intuition from classical deep learning algorithms
- CNNs exploit structure in the images

Translation invariance

Composability

CNN architecture: Illustrative

- CNNs hierarchically aggregate (through convolution) and pool (i.e., subsample) images along pixel-grid



https://en.wikipedia.org/wiki/File:Typical_cnn.png

How can we extend CNNs on graphs?

- Desirable properties

  - **Convolution:** how to achieve translation invariance

  - **Localization:** what is the notion of locality

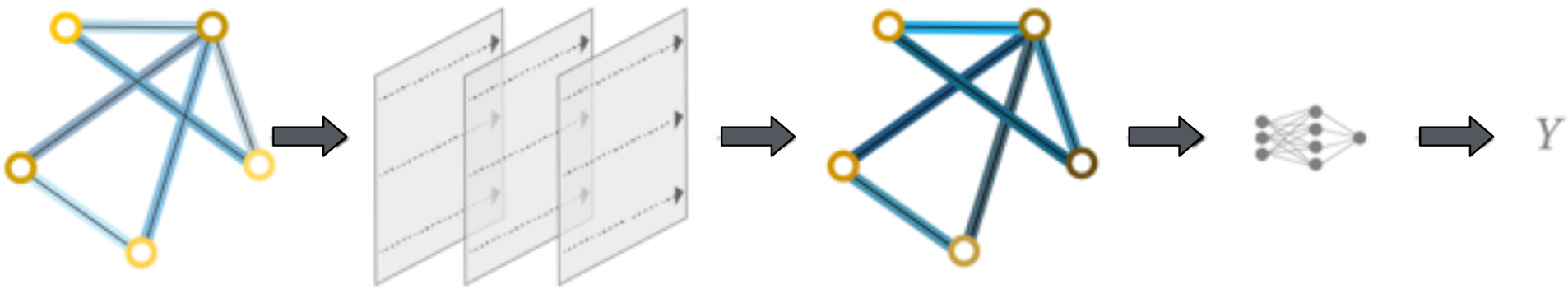
  - **Graph pooling:** how to downsample on graphs

  - **Efficiency:** how to keep the computational complexity low

  - **Generalization:** how to build models that generalize to unseen graphs

GNN model: schematic overview

- Applicable to most state-of-the-art architectures



Graph and node attributes

$\mathcal{G}, X$

GNN blocks
(graph convolution, pooling, nonlinearity)

Transformed graph
(node/edge embeddings)

$\mathcal{G}', H$

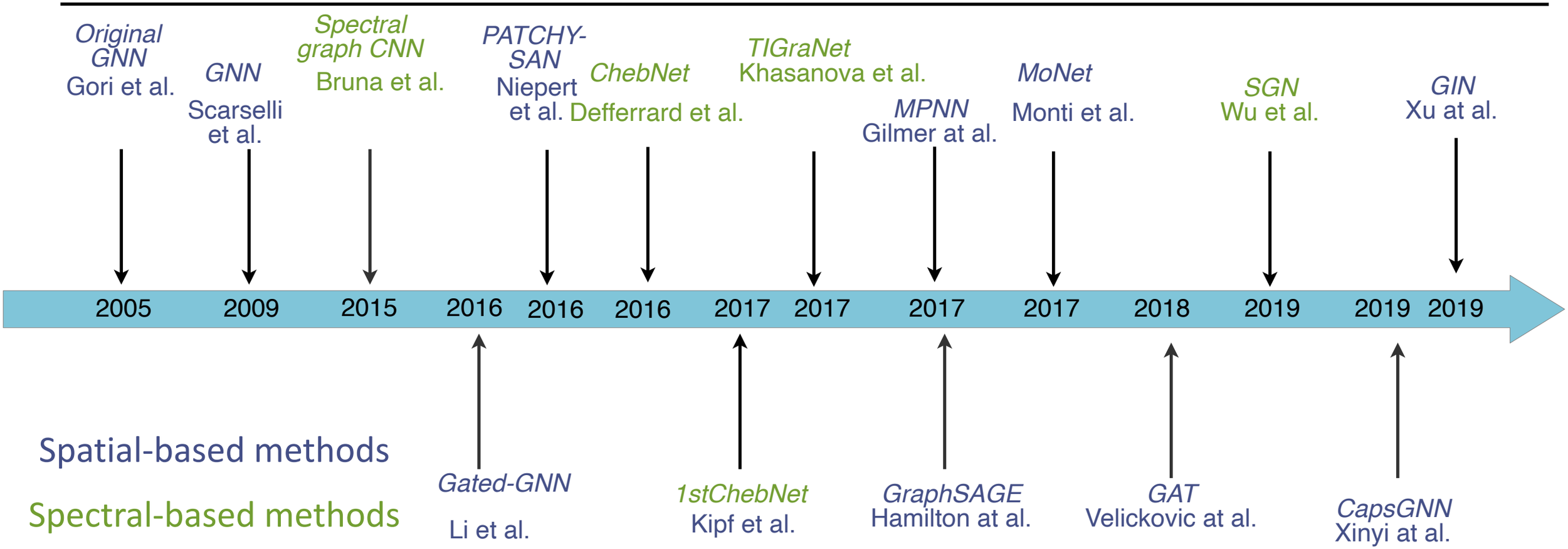
Classification layer

Prediction
(node, edge, graph level)

Y

- We apply permutation invariant functions on local neighbourhoods of the graph

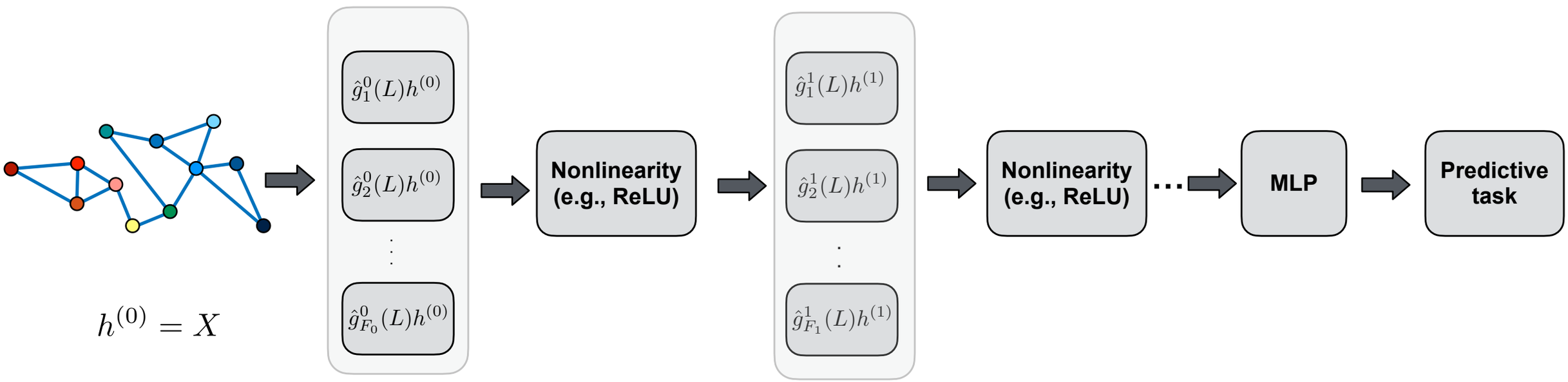
First GNN architectures



- Recent trends
  - Spectrally-inspired architectures: GraphHeat (Xu'19), GWNN (Xu'19), SIGN (Frasca'20), DGN (Beaini'20), Framelets (Zheng'21), FAGCN (Bo'21)
  - More expressive GNNs: higher order WL test (Maron'19, Morris'20), physics-inspired GNNs (Chamberlain'21), and many more!

The basic GNN: a spectral viewpoint

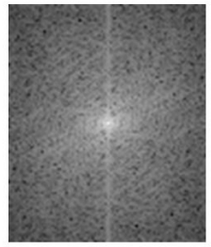
- Typical GNN architectures consist of a set of graph convolutional layers, each of which is followed by elementwise nonlinearity



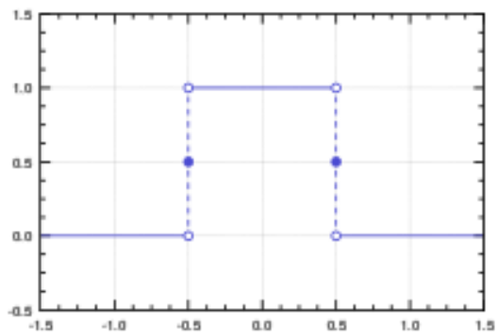
- By learning the parameters of the each convolutional filter, we learn how to propagate information on a graph to compute node embeddings

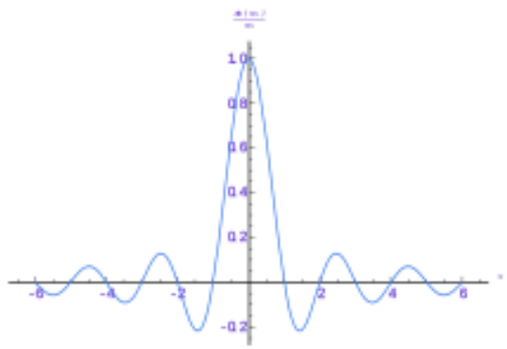
The Fourier transform

- One of the most fundamental notions in signal processing/ analysis



- A mathematical transform that decomposes functions depending on space or time into functions depending on spatial or temporal frequency





How can we define the Graph Fourier transform for graph structured data?

A notion of frequency on the graph

- The Laplacian L admits the following eigendecomposition: $L\chi_\ell = \lambda_\ell \chi_\ell$

one-dimensional Laplace operator: $\frac{d^2}{dx^2}$

eigenfunctions: $e^{j\omega x}$

Classical FT $\hat{f}(\omega) = \int (e^{j\omega x})^* f(x) dx$

$$f(x) = \frac{1}{2\pi} \int \hat{f}(\omega) e^{j\omega x} d\omega$$

graph Laplacian: L

eigenvectors: χ_ℓ

$f : \mathcal{V} \to \mathbb{R}^N$

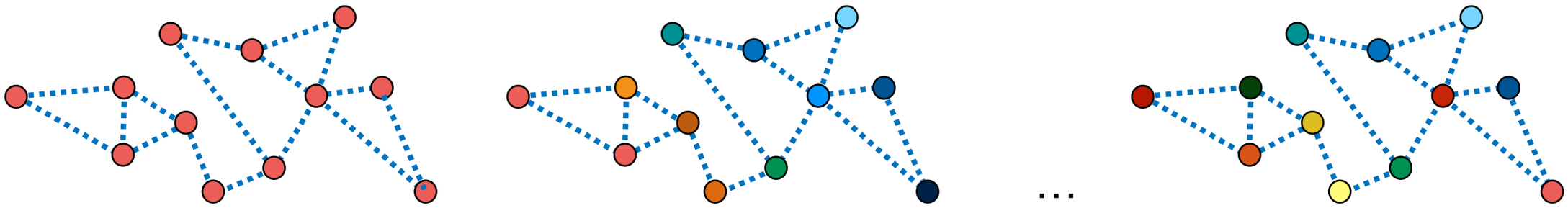
Graph FT: $\hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^{N} \chi_\ell^*(i) f(i)$

$$f(i) = \sum_{\ell=1}^{N} \hat{f}(\ell) \chi_\ell(i)$$

FT: Fourier Transform

Graph Fourier transform

- The eigenvectors of the Laplacian provide a harmonic analysis of graph signals



$\chi_1^T L \chi_1 = \lambda_1$ $\chi_2^T L \chi_2 = \lambda_2$ $\chi_N^T L \chi_N = \lambda_N$

Graph Fourier Transform: $\hat{f}(\lambda_\ell) =< f, \chi_\ell >= \sum_{n=1}^{N} f(n)\chi_\ell^T(n), \quad \ell = 1, 2, ..., N$

- By exploiting the orthonormality of the eigenvectors, we obtain:

Inverse Graph Fourier Transform: $f(n) = \sum_{\ell=1}^{N} \hat{f}(\lambda_\ell)\chi_\ell(n), \quad \forall n \in \mathcal{V}$

Towards a convolution on graphs: A spectral viewpoint

- **Key intuition:** Convolution in the vertex domain is equivalent to multiplication in the spectral domain
- Recall that: The graph Fourier transform of a graph signal x is defined using the eigenvectors and the eigenvalues of the Laplacian matrix ($L = \chi \Lambda \chi^T$)
- We define convolution on graphs starting from the multiplication in the GFT domain

$$(x * g)(t) = \int_{-\infty}^{\infty} x(t-\tau) g(\tau) d\tau$$

FT

$$\widehat{(x * g)}(\omega) = \hat{x}(\omega) \cdot \hat{g}(\omega)$$

Classical convolution

$$x * g = \chi \hat{g}(\Lambda) \chi^T x = \hat{g}(L) x$$

IGFT

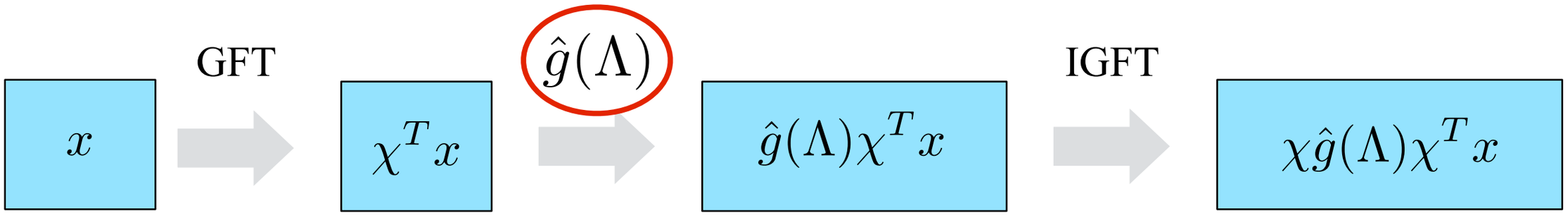
$$\widehat{(x * g)}(\lambda) = \big((\chi^T x) \circ \hat{g}\big)(\lambda)$$

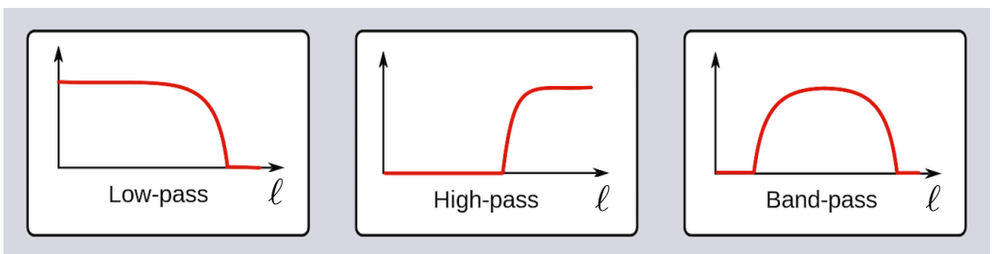
GFT

Convolution on graphs

Graph spectral filtering

- It is defined in direct analogy with classical filtering in the frequency domain
  - Filtering a graph signal x with a spectral filter $\hat{g}(\cdot)$ is performed in the graph Fourier domain





Convolution on graphs is equivalent to filtering!

$$x * g = \chi\hat{g}(\Lambda)\chi^T x = \hat{g}(L)x$$

Shuman et al., “The emerging field of signal processing on graphs”, IEEE Signal Process. Mag., 2013

A spatial interpretation of graph convolution

- If we approximate the filter with polynomials of the Laplacian, we get a spatial interpretation on the graph

$$x * g = \hat{g}(L)x = \sum_{k=0}^{K} \theta_k L^k x = \sum_{k=0}^{K} \theta_k z_k$$

- Note that:

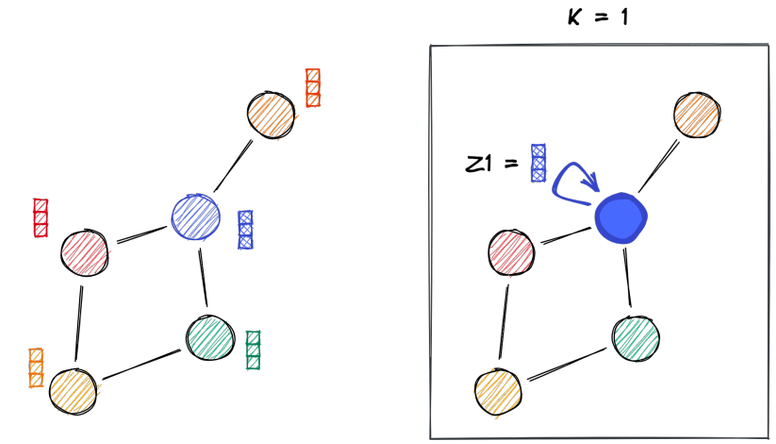
$$z_0 = x$$

$$z_1 = Lz_0$$

$$z_2 = Lz_1 = L^2 z_0$$

$$\vdots$$

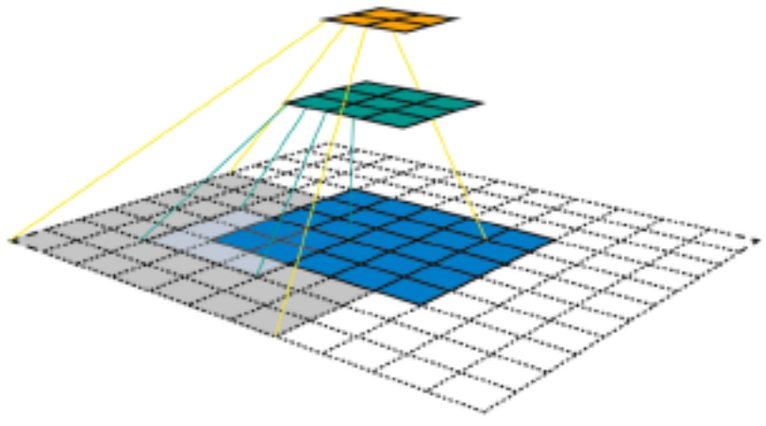
$$z_K = Lz_{K-1} = \cdots = L^K z_0$$



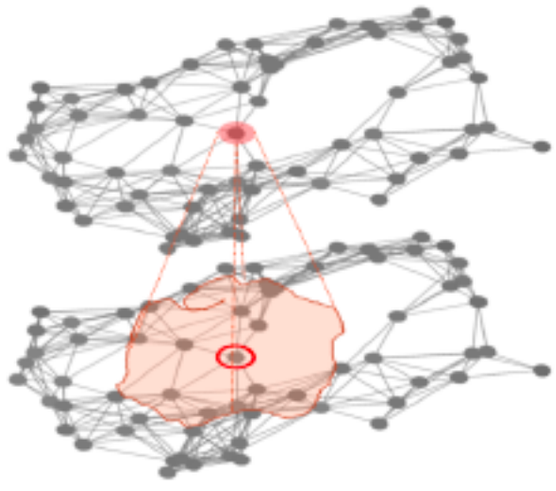
- Graph convolution can be computed recursively by exchanging information in a local neighborhood (i.e., message passing)
- The kernel $\hat{g}(\cdot)$ does not depend on the order of the nodes: permutation invariant!

The receptive field of graph convolution

- Node embeddings are based on local neighborhood propagation
- Due to the irregular nature of the graph, there is no fixed size neighbourhood
- The degree K of the polynomial defines the receptive field of each node



Receptive field on an image



Receptive field on a graph

Spectral approaches in one slide

- Convolution is defined in the graph Fourier domain

$$x *_\theta g = \chi \hat{g}(\theta) \chi^T x$$

- **Spectral GCNN:** $\hat{g}(\lambda) = \theta$ ⇒ $x * g = \chi \theta \chi^T x$

- **ChebNet:** $\hat{g}(\lambda) = \sum_{k=0}^{K} \theta_i T_i(\lambda)$ ⇒ $x * g = \sum_{k=0}^{K} \theta_i T_i(L) x$

- **GCN:** $K = 1$ ⇒ $x * g = (\theta_0 - \theta_1 D^{-1/2} W D^{-1/2}) x$

- Parameters θ are learned through the network

Graph Convolutional Networks

- **Main intuition:** Design a scalable architecture with first-order approximation of spectral graph convolution

$$g * x \approx \theta_0 x + \theta_1 (L - I_N) x = \theta_0 x - \theta_1 D^{-1/2} W D^{-1/2} x$$

$$\theta = \theta_0 = -\theta_1$$

$$g * x \approx \theta (I + D^{-1/2} W D^{-1/2}) x$$

$$\tilde{W} = W + I_N$$

$$g * x \approx \theta \tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2} x$$

Adjacency/Weight matrix

Degree matrix

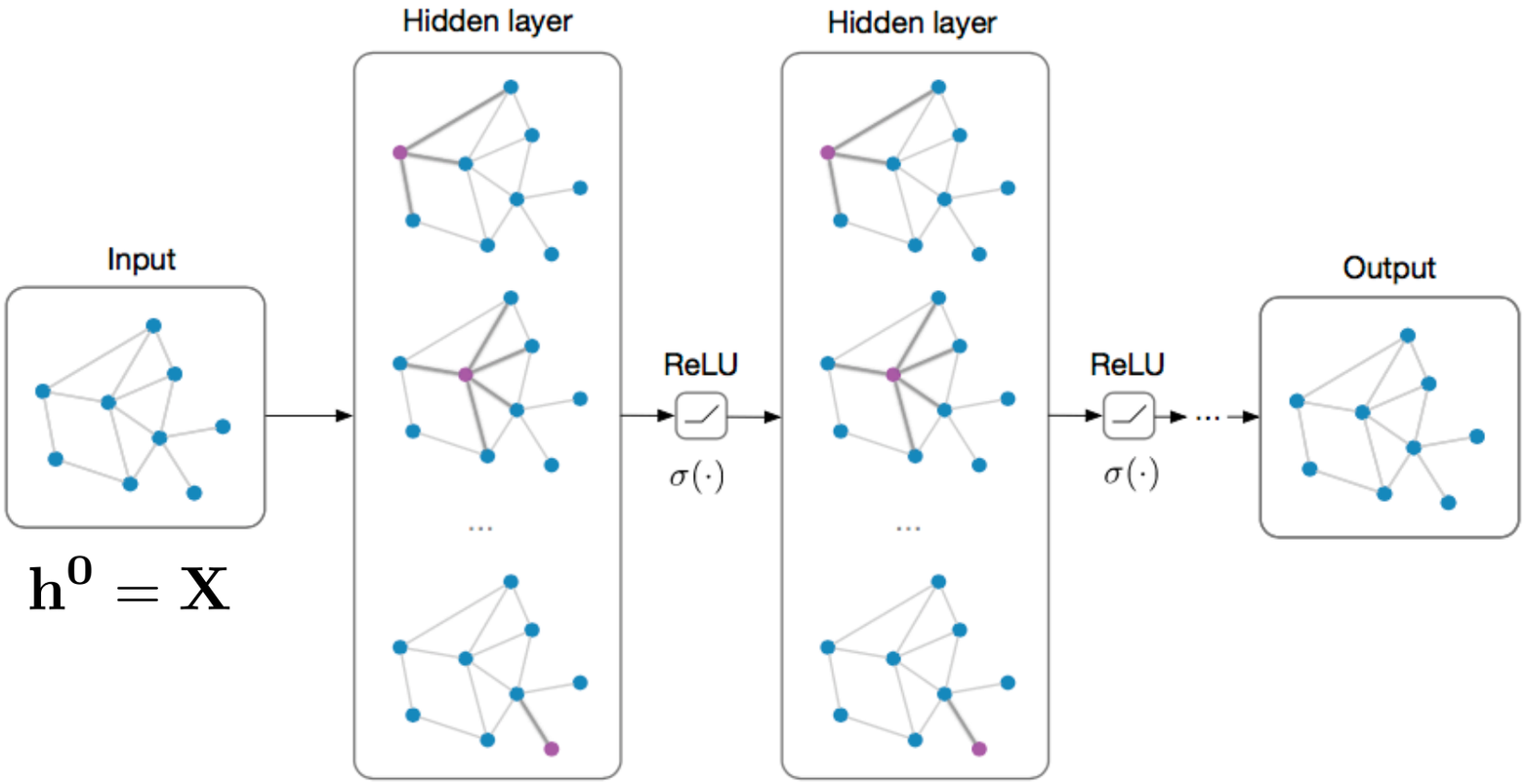
- A convolutional layer is defined as:

$$h^{l+1} = \sigma(\tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2} h^l \theta^{l+1})$$

Learned parameters

GCN architecture

- Very often, it consists of two GCN layers

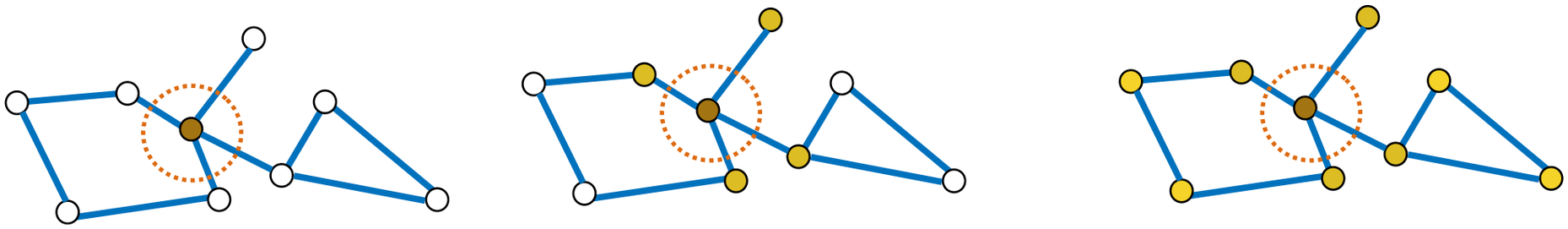


$$\mathbf{h^{l+1}} = \sigma(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{W}}\tilde{\mathbf{D}}^{-1/2}\mathbf{h^l}\theta^{l+1})$$

[Kipf et al., Semi-Supervised Classification with Graph Convolutional Networks, ICLR, 2017]

Each layer increases the receptive

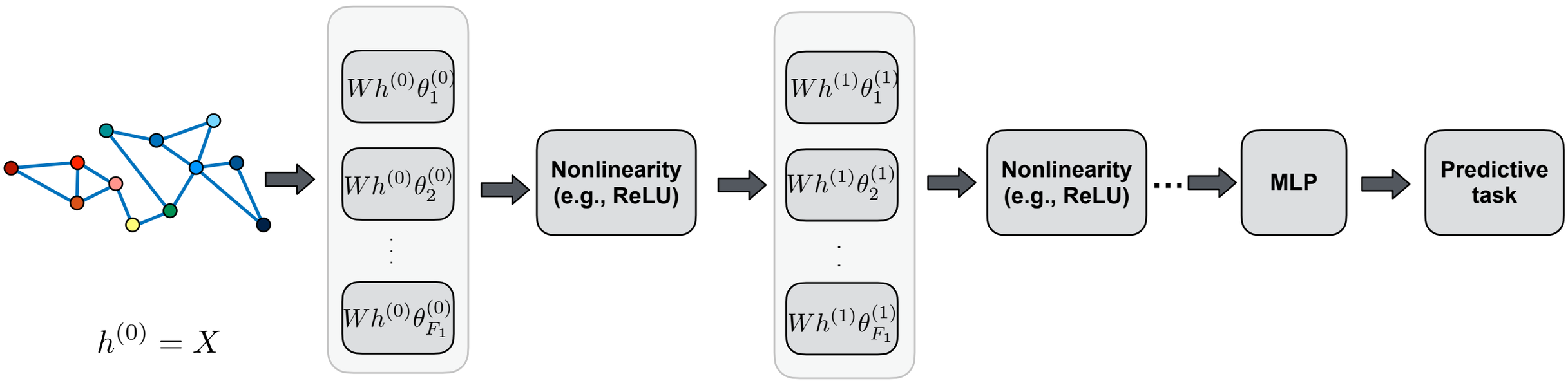
- Each layer increases the receptive field by K hops
- Example: $K = 1$



Layer 0 **Layer 1** **Layer 2**

The basic GNN: a spatial viewpoint

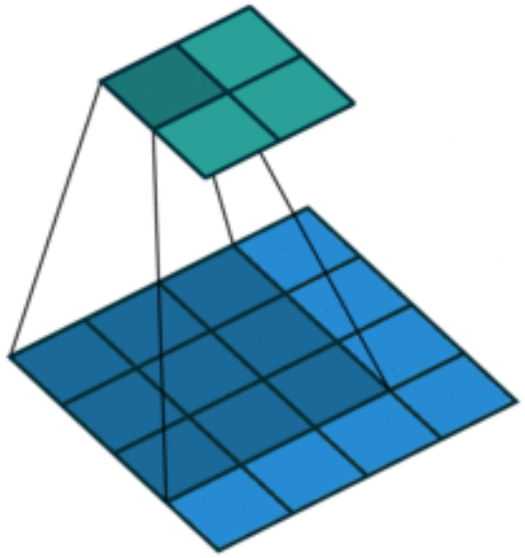
- Consists of a set of graph convolutional layers, each of which is followed by elementwise nonlinearity, i.e., $h^{(l+1)} = \sigma(z^{(l)})$



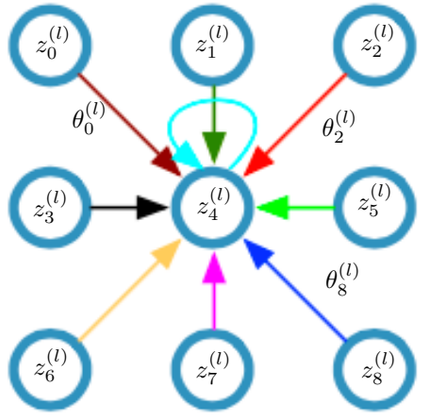
- Each layer increases the receptive field by 1-hop neighbors

Towards a graph convolution: A spatial viewpoint

- **Key intuition:** Generalize the notion of convolution from images (grid graph) to networks (irregular graph)
- Example of a single CNN layer with 3x3 filter
  - Fixed neighbourhood
  - Canonical order across neighbors



Animation from V. Dumoulin

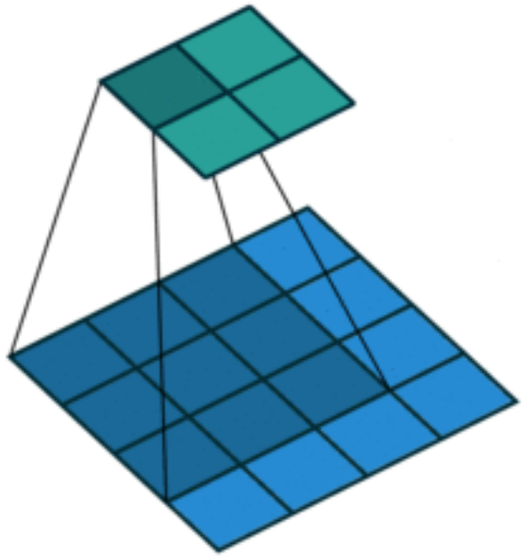


$$z_i^{(l+1)} = \sum_{i=0}^{8} \theta_i^{(l)} z_i^{(l)}$$

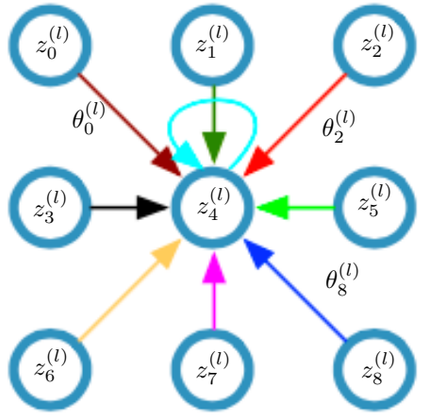
Can we exploit similar structure for graph data?

Towards a graph convolution: A spatial viewpoint

- **Key intuition:** Generalize the notion of convolution from images (grid graph) to networks (irregular graph)
- Example of a single CNN layer with 3x3 filter
  - Fixed neighbourhood
  - Canonical order across neighbors



Animation from V. Dumoulin



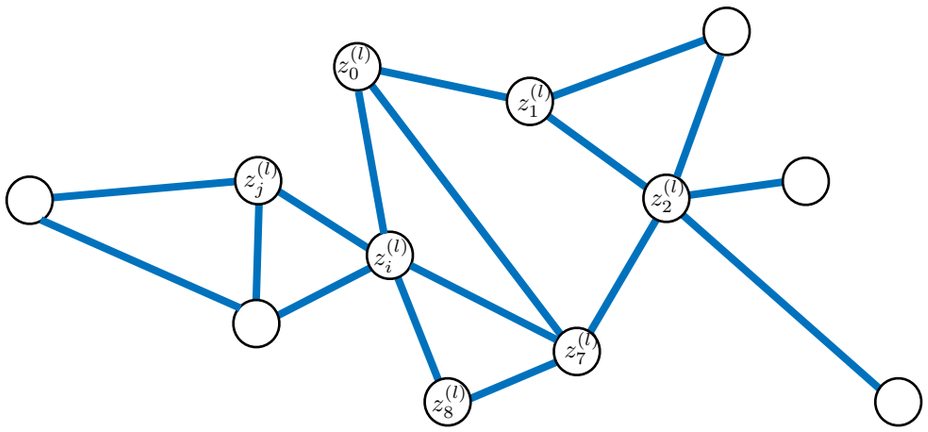
$$z_i^{(l+1)} = \sum_{i=0}^{8} \theta_i^{(l)} z_i^{(l)}$$

Can we exploit similar structure for graph data?

Spatial graph convolution

- Main issue: We cannot have variable number of weights; it requires assuming an order on the nodes

- Solution: Impose same filter weights for all nodes



$$z_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} \theta_j^{(l)} z_j^{(l)}$$

$$z_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} \theta^{(l)} z_j^{(l)}$$

$$z_i^{(l+1)} = \theta^{(l)} z_i^{(l)} + \sum_{j \in \mathcal{N}_i} \theta^{(l)} z_j^{(l)}$$

Update embeddings by exchanging information with 1-hop neighbors

Message Passing Neural Network

- **Main intuition:** Each node exchange messages with its neighbors and update its representations based on these messages

- The message passing scheme runs for T time steps and updates the representation of each vertex based on its previous representation and the representation of its neighbors

$$m_i^{l+1} = \sum_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l, e_{ij})$$

Message function

$$h_i^{l+1} = U_l(h_i^l, m_i^{l+1})$$

Vertex update function

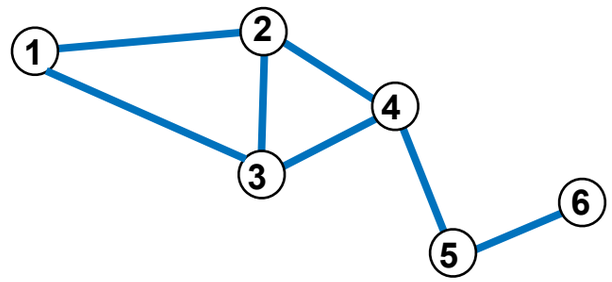
Learned differentiable functions!

[Gilmer et al, Neural Message Passing for Quantum Chemistry, ICML, 2017]

MPNN - Example

- At each iteration, the embeddings are updated as follows:

$$
\begin{aligned}
h_1^{l+1} &= \theta_0^l h_1^l + \theta_1^l h_2^l + \theta_1^l h_3^l \\
h_2^{l+1} &= \theta_0^l h_2^l + \theta_1^l h_1^l + \theta_1^l h_3^l + \theta_1^l h_4^l \\
h_3^{l+1} &= \theta_0^l h_3^l + \theta_1^l h_1^l + \theta_1^l h_2^l + \theta_1^l h_4^l \\
h_4^{l+1} &= \theta_0^l h_4^l + \theta_1^l h_2^l + \theta_1^l h_3^l + \theta_1^l h_5^l \\
h_5^{l+1} &= \theta_0^l h_5^l + \theta_1^l h_4^l + \theta_1^l h_6^l \\
h_6^{l+1} &= \theta_0^l h_6^l + \theta_1^l h_5^l
\end{aligned}
$$



- The output of the message passing is:

$$\{h_1^{l_{max}}, h_2^{l_{max}}, h_3^{l_{max}}, h_4^{l_{max}}, h_5^{l_{max}}, h_6^{l_{max}}\}$$

Comparison between spatial and spectral design

- Spectral convolution: Generalizes the notion of convolution by following a frequency viewpoint

- Spatial convolution: Generalizes the notion of convolution by following a spatial viewpoint

- Strong links exist between both; The practical difference usually relies on the receptive field
  - Spectral approaches: Every layer can ‘reach’ K-hops neighbors
  - Spatial approaches: Each layer can ‘reach’ 1-hops neighbors

A summary of the GNN landscape

- **Convolutional GNNs**:

$$h_i = \phi\big(X_i, \bigoplus_{j \in \mathcal{N}_i} \psi(X_j)\big)$$

- **Message passing GNNs**:

$$h_i = \phi\big(X_i, \bigoplus_{j \in \mathcal{N}_i} \psi(X_i, X_j)\big)$$

- **Attentional GNNs**:

$$h_i = \phi\big(X_i, \bigoplus_{j \in \mathcal{N}_i} \alpha(X_i, X_j)\psi(X_j)\big)$$

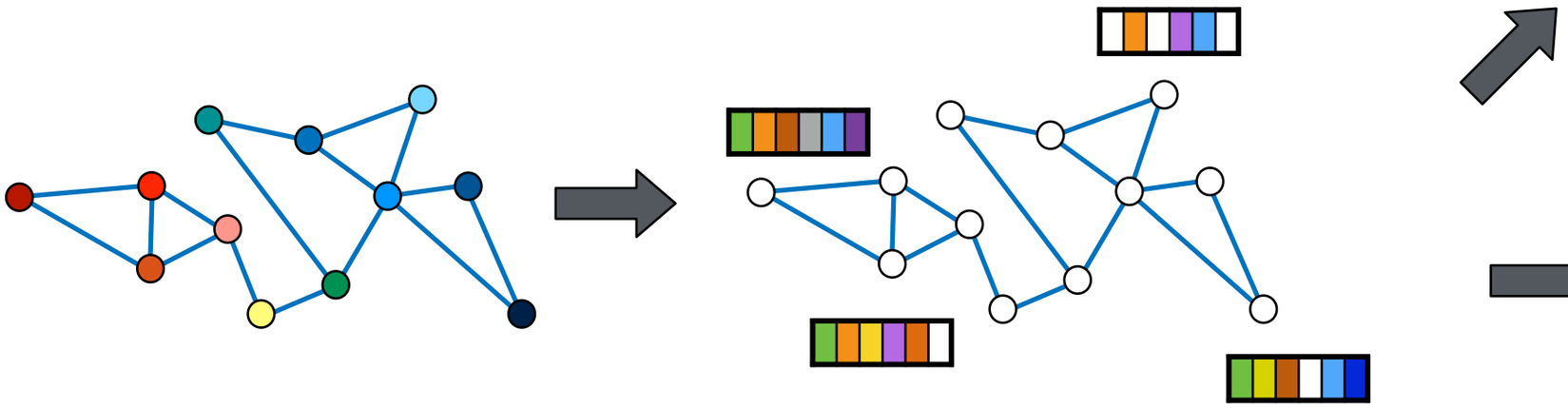
Functions to be learned!

- Depending on how these functions are instantiated, different architectures are obtained

[Slide inspired from P. Veličković]

How to use GNNs?

- GNNs typically provide embeddings at a node level
- These embeddings can be used for learning a downstream task



$\mathcal{G}, X$

$\mathcal{G}, H$

GNN-based embeddings

Node classification

$$\tilde{y}_i = f(h_i)$$

Graph classification

$$\tilde{y}_{\mathcal{G}} = f(\bigoplus_{i \in \mathcal{V}} h_i)$$

Link prediction

$$\tilde{y}_{i,j} = f(h_i, h_j, W_{ij})$$

Useful resources

- **Toolboxes**
  - https://github.com/rusty1s/pytorch_geometric
  - https://github.com/dmlc/dgl
  - https://github.com/deepmind/jraph
  - https://github.com/tensorflow/gnn
- **Datasets**
  - DGL datasets: https://docs.dgl.ai/api/python/dgl.data.html
  - PyG datasets: https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html
  - OGB datasets: https://ogb.stanford.edu
  - https://chrsmrrs.github.io/datasets/
  - https://chrsmrrs.github.io/datasets/

Summary

- Machine learning on graphs/networks requires developing new tools that extract information (i.e., features) from complex structures

- Graph-based features (i.e., embeddings) can be designed based on some prior, or learned from data

- Graph neural networks: A very active area of research
  - Different architecture designs, most of them can be categorized as convolutional, message passing, attentional

- A variety of applications, especially in science and biomedicine!

References

1. Graph representation learning (chap 5), William Hamilton

2. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, Bronstein et al, 2022
   - https://arxiv.org/abs/2104.13478

3. A Comprehensive Survey on Graph Neural Networks, Wu et al, 2021
   - https://arxiv.org/abs/1901.00596

Thank you!