EE626:
Quick introduction into graph machine learning

Dr Dorina Thanou

17.09.2025

Questions from previous lecture?

- Link to preferences (please add your name if you haven't done so!):
  - https://docs.google.com/spreadsheets/d/1WxXG2pXAkG4j6YjpYyscz7lfQ41MO59aoI5mipZoW_4/edit?usp=sharing

- Any suggestions?

Agenda

- Intro into graph machine learning
- Traditional machine learning on graphs

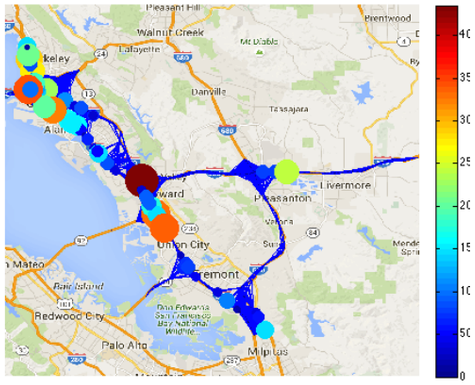
Today

- Graph neural networks
- Applications

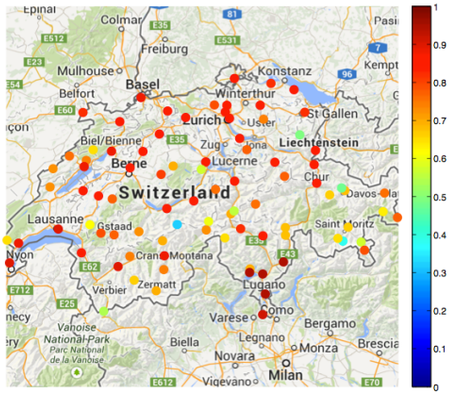
Next week

Going beyond graph structure

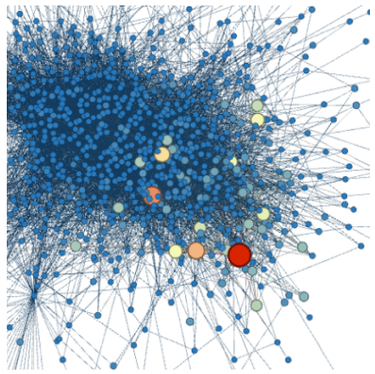
- Very often data comes with additional features
  - Not only graphs, but attributes on the nodes of the graph (e.g., congestion in rode junctions, preferences of individuals, activities in brain regions…)



Transportation networks

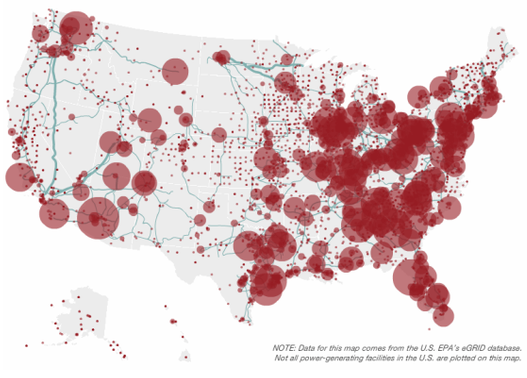


Weather networks

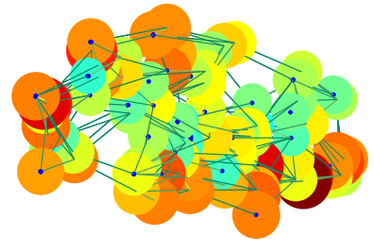


Social networks

Disease spreading networks



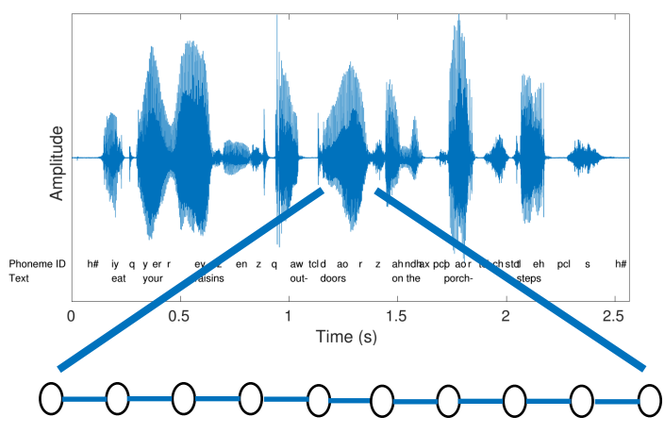
Electric grid networks



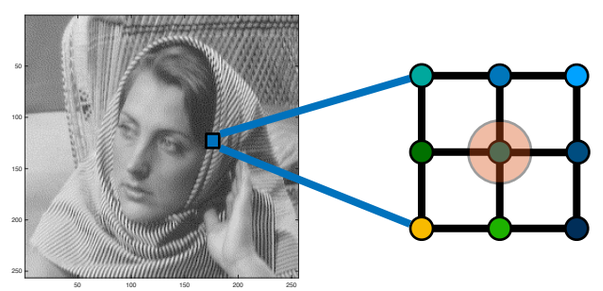
Biological networks

Graph structured data

- Data live on a regular domain

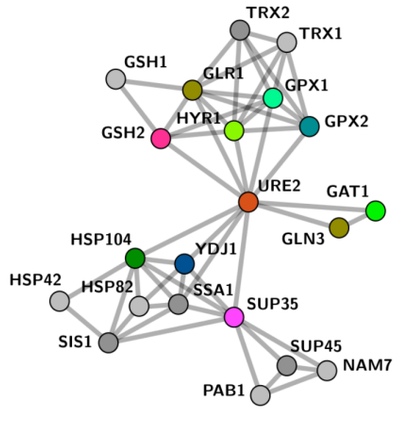


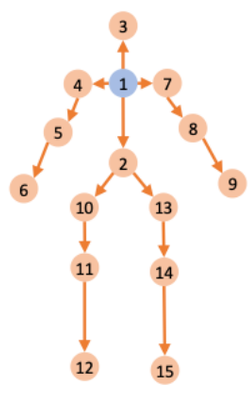
Domain: line



Domain: grid

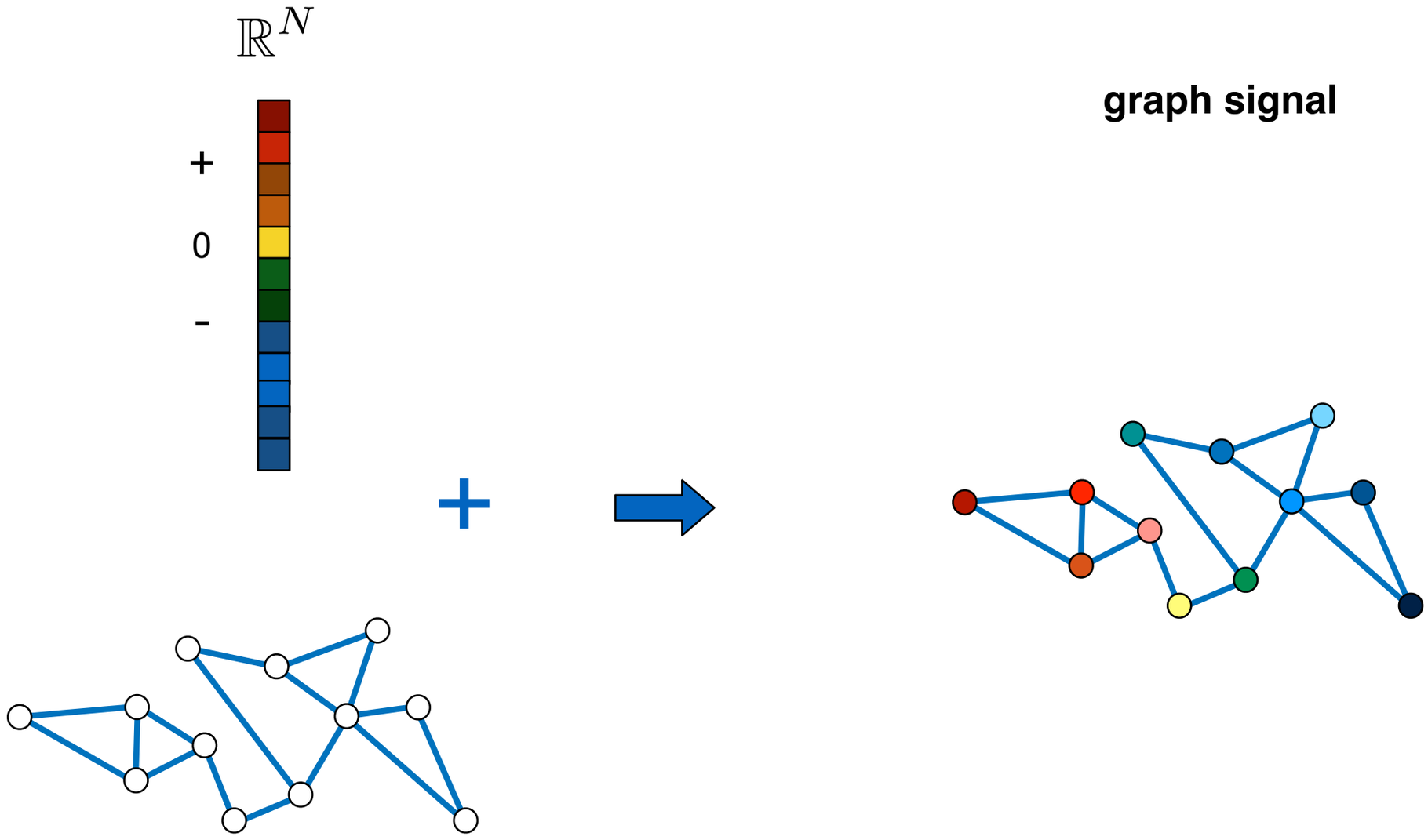
- Weighted graphs capture the geometric structure of complex, i.e., irregular, domains





Domain: irregular graph

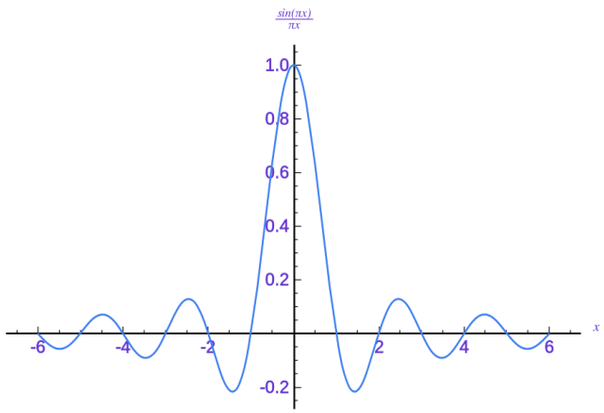
Processing graph structured data



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

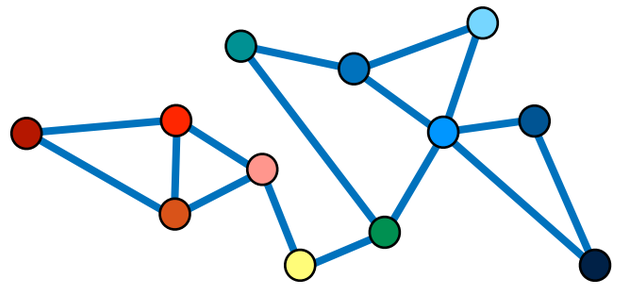
Processing graph structured data

1D signal



2D signal

graph signal

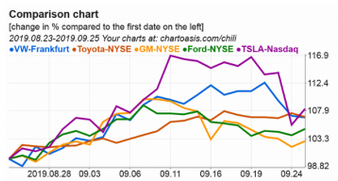


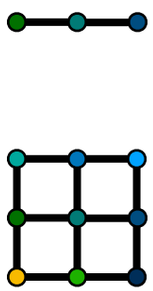
How can we extract useful information by taking into account both **structure (edges)** and **data (values/features on vertices)**?

Classical ML vs Graph ML

Classical ML

regular domain
(real line, 2D grid)





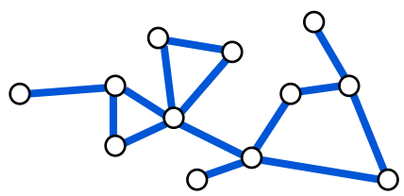
$f(X)$

time series forecasting

image classification

Graph ML

irregular domain
(graphs)



$\mathcal{G}$

X

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

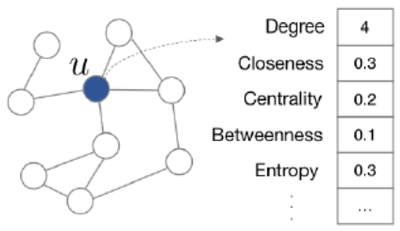
node classification

link prediction

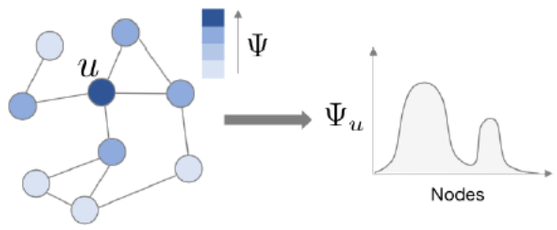
graph classification

graph clustering

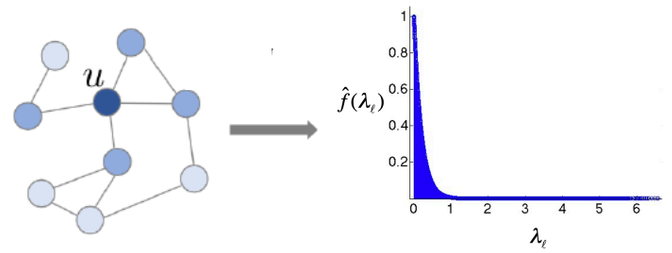
Predominant graph representation learning paradigms



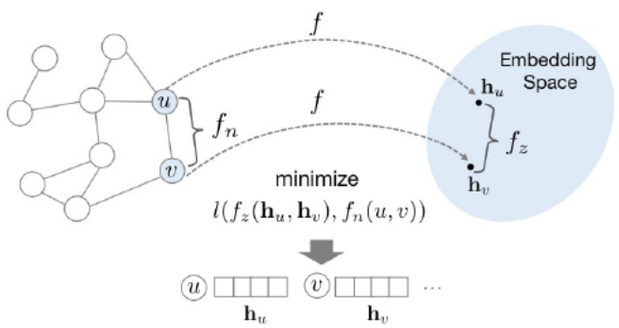
Hand-crafted graph theoretic features



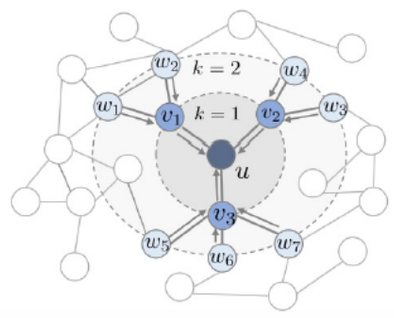
Kernel-based features



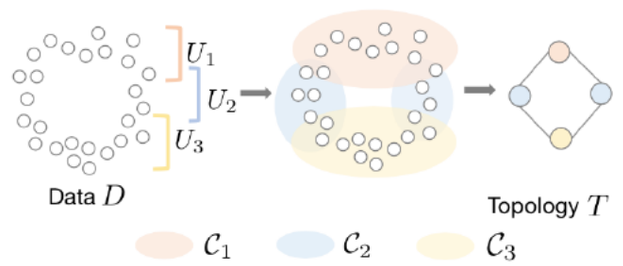
Graph signal processing based features



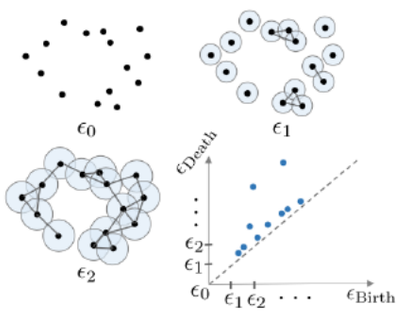
Shallow embeddings



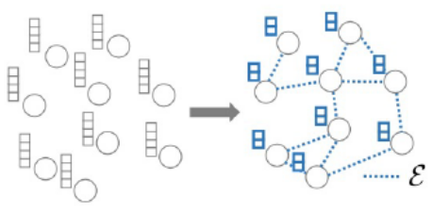
Deep embeddings: Graph neural networks



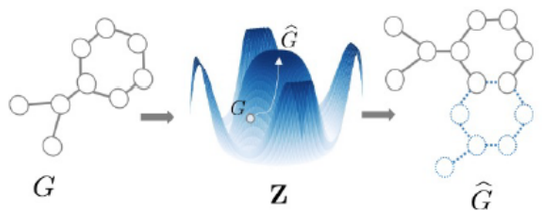
Topological features



Persistent homology



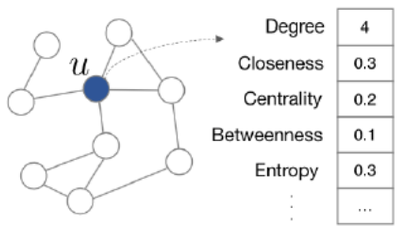
Manifold learning & Topology inference



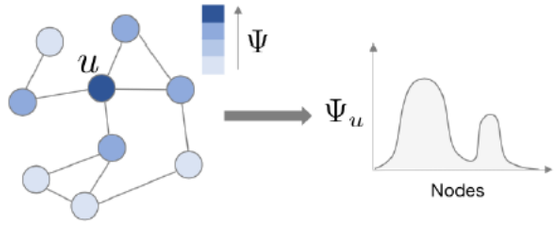
Graph generative models

[Fig modified from M. Li, K. Hunag, and M. Zitinik.,Graph Representation Learning in Biomedicine and Healthcare, Nature Biomedical Engineering, 2022]

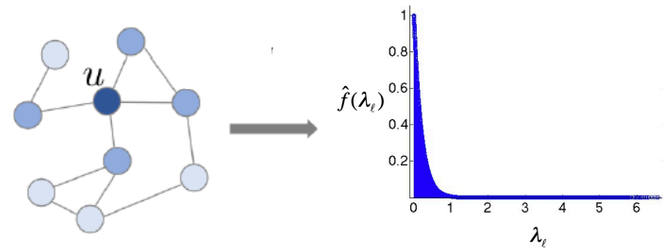
Predominant graph representation learning paradigms



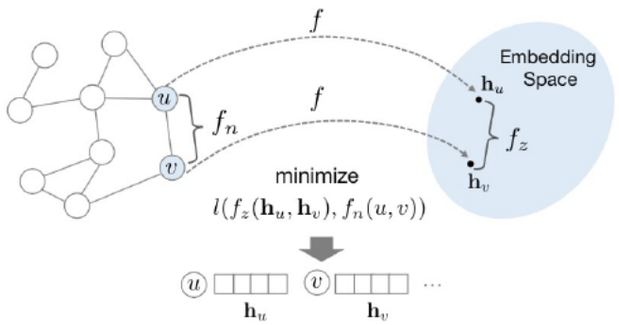
Hand-crafted graph theoretic features



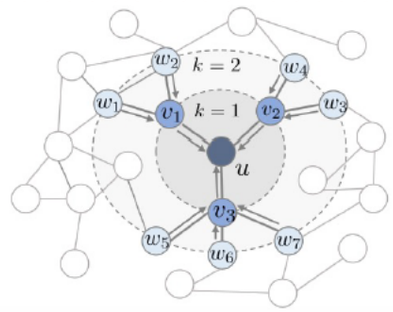
Kernel-based features



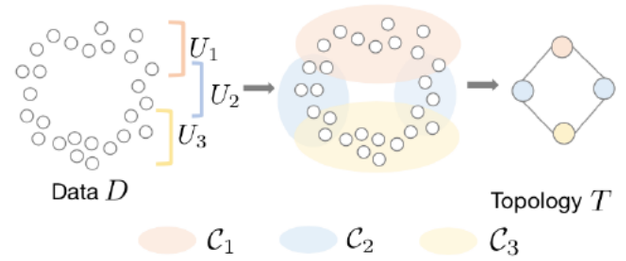
Graph signal processing based features



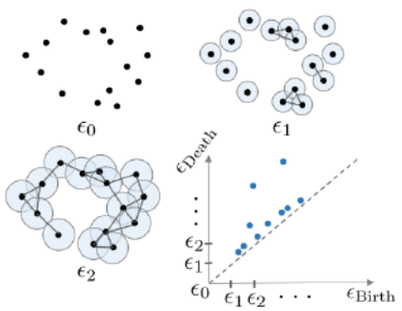
Shallow embeddings



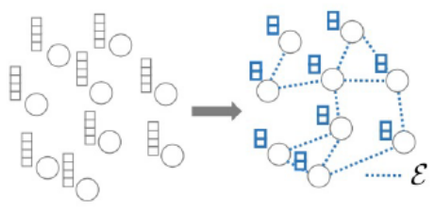
Deep embeddings: Graph neural networks



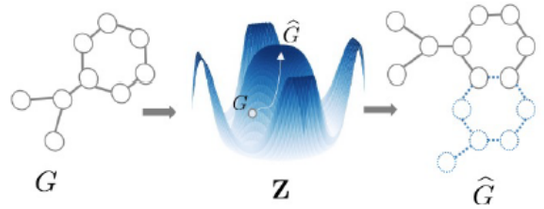
Topological features



Persistent homology



Manifold learning & Topology inference



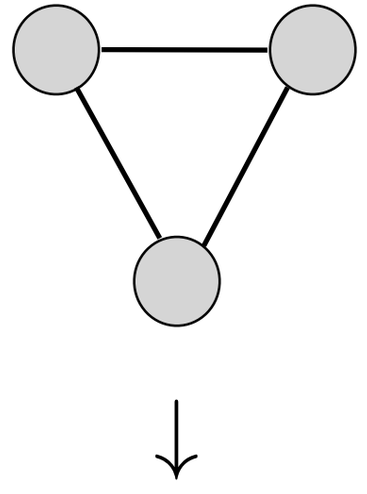
Graph generative models

[Fig modified from M. Li, K. Hunag, and M. Zitinik.,Graph Representation Learning in Biomedicine and Healthcare, Nature Biomedical Engineering, 2022]

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 :

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

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

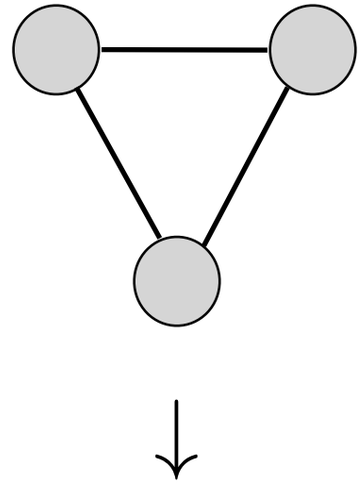
- If the graph is unweighted (often denoted as A) :

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

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

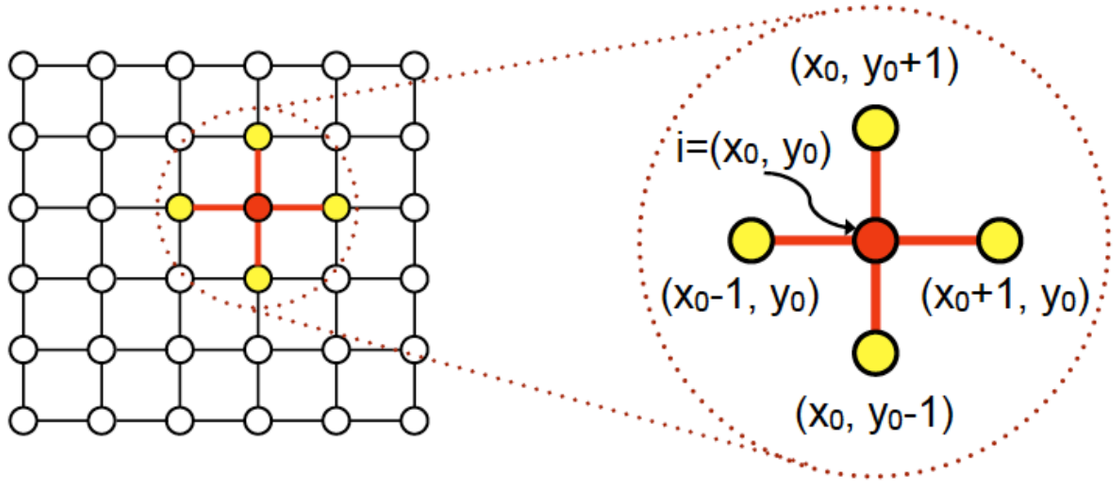
$$\big(Lf\big)(i) = \sum_{j \in \mathcal{N}_i} W_{i,j}(f_i - f_j)$$

- 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



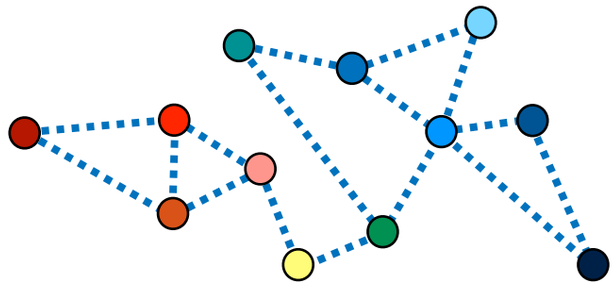
$$L = D - W$$

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

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

Signal on the graph or graph signal

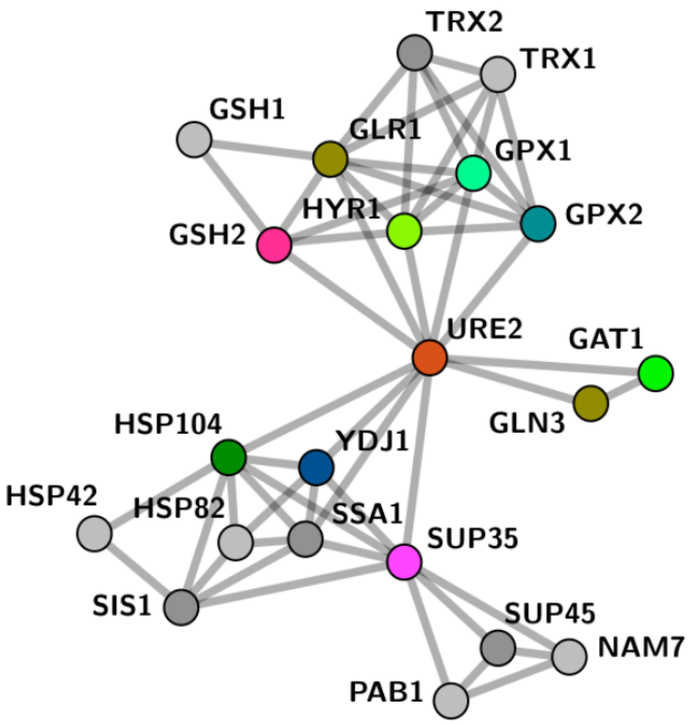
- A function $f : \mathcal{V} \rightarrow \mathbb{R}^N$ that assigns real values to each vertex of the graph
- It is defined on the vertices of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$

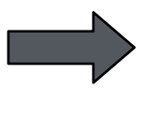


- Often represented as a vector $f \in \mathbb{R}^N$, where $f(i)$ is the signal value at node i
- The ordering of the vector follows the ordering of the adjacency matrix

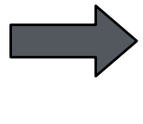
In this lecture…

- 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

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

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

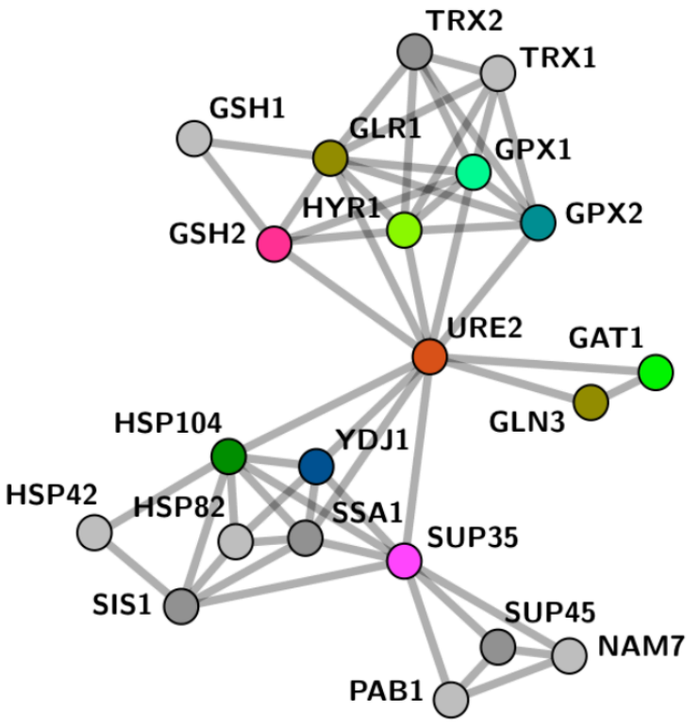
Model-driven

- **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?

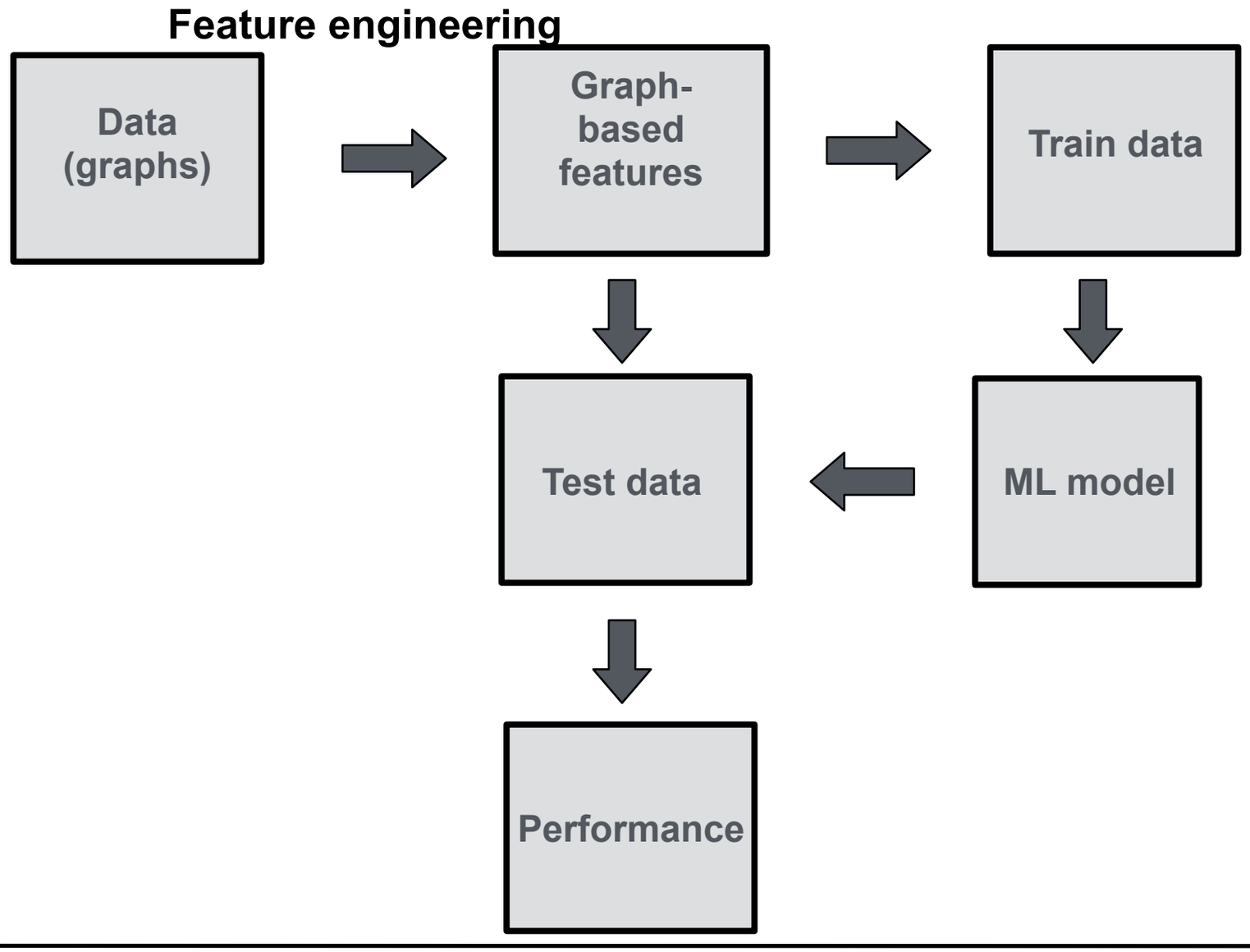


e.g., node/graph classification, signal inpainting/denoising

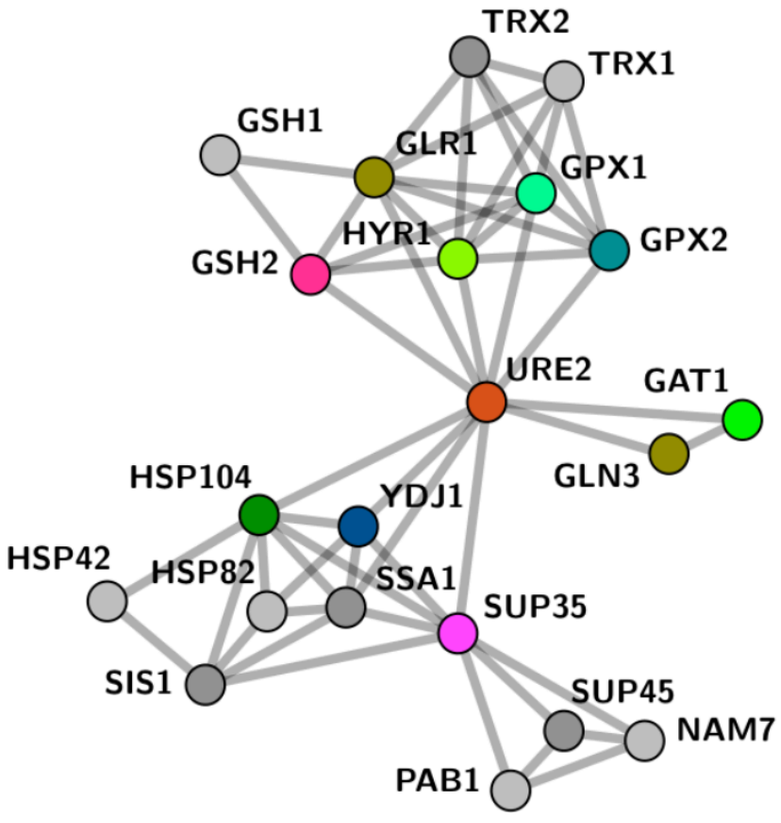
$$X, \mathcal{G} \quad \Rightarrow \quad \phi(X, \mathcal{G}) \quad \Rightarrow \quad f(\phi(X, \mathcal{G})) \quad \Rightarrow \quad Y$$

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

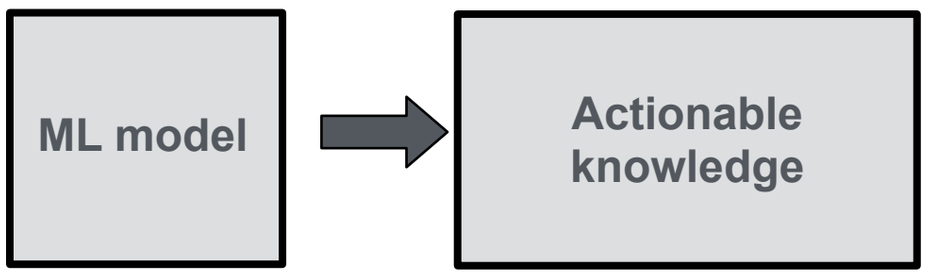
- Features can be defined at different scales
  - At a node, edge, sets of nodes, entire graph level

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

- 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.



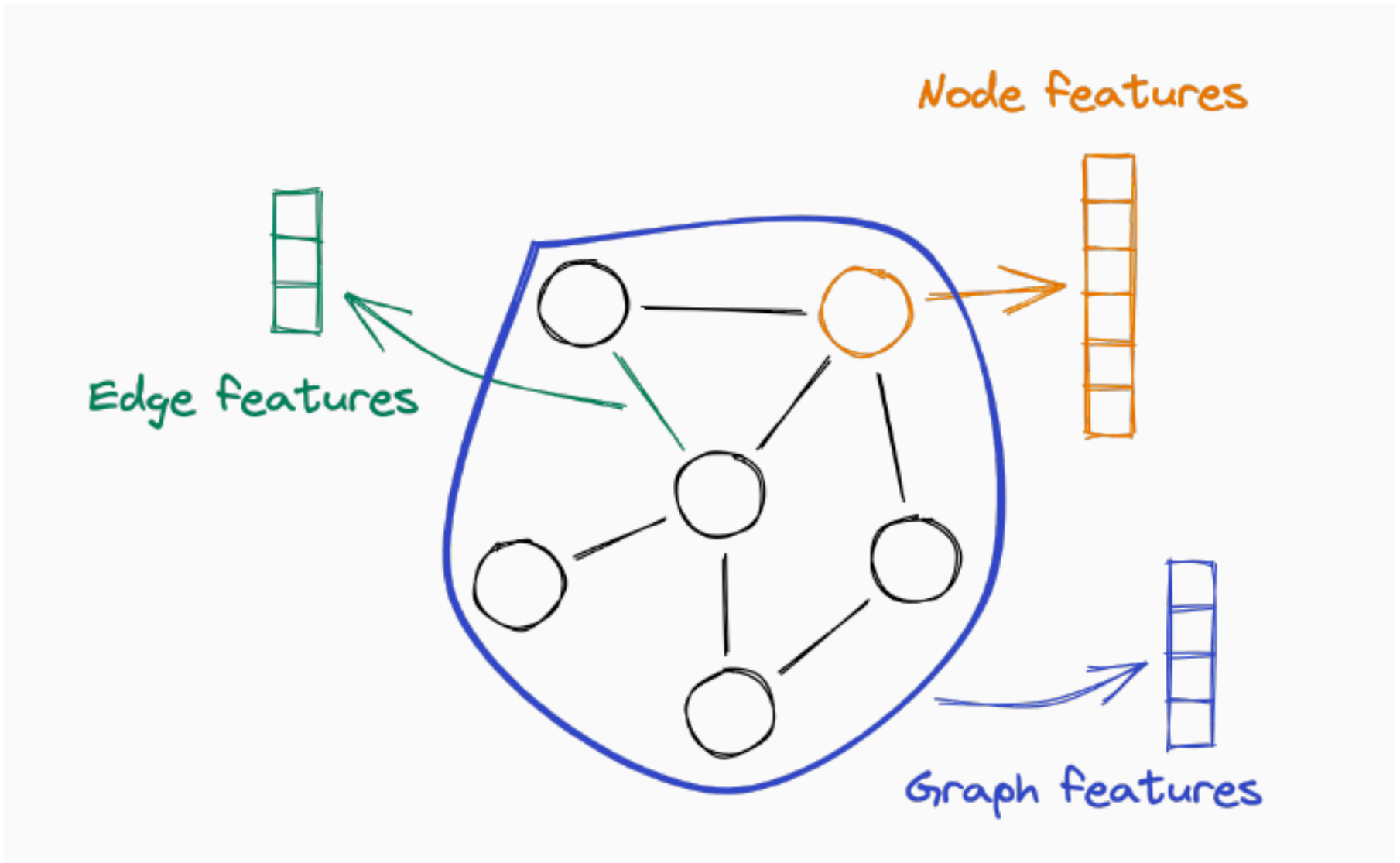
e.g., node/graph classification, signal inpainting/denoising

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

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

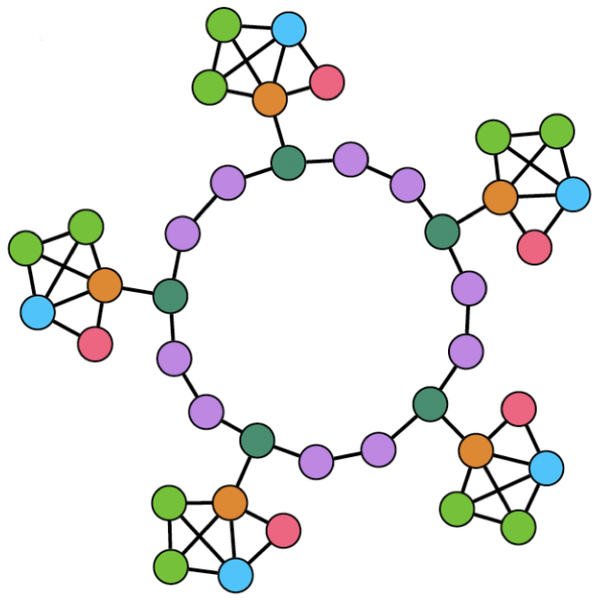
- Testing phase:
  - Given a new node/link/graph, compute its features, and give them as an input to f to make a prediction

Extracting information at different levels



Node level features

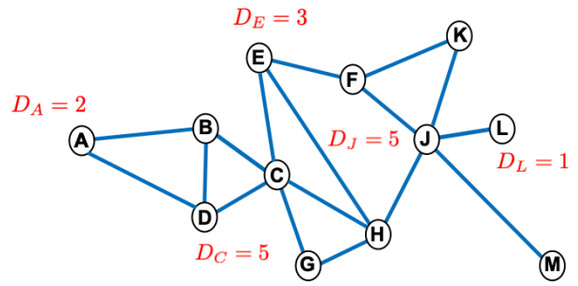
- Typically useful for node classification/clustering tasks



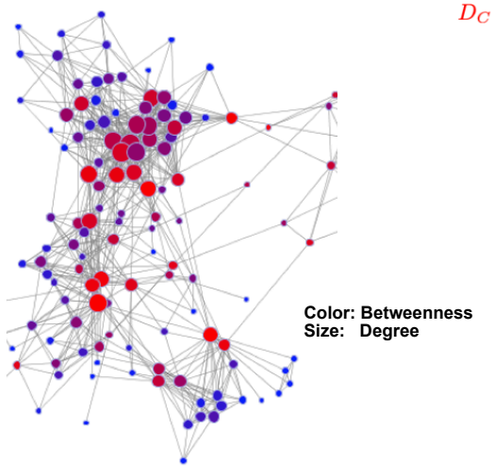
- Aim at characterizing the structure and position of a node in the network

Common node level features

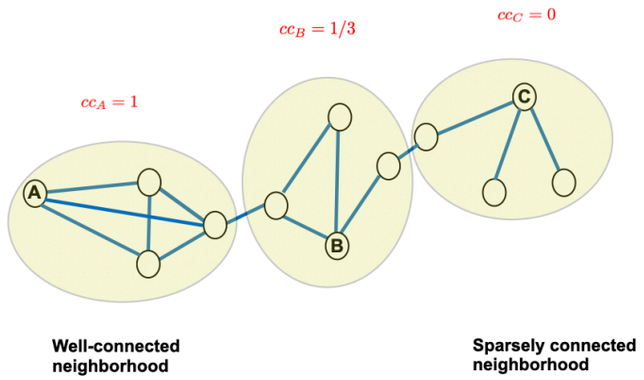
- Node degree



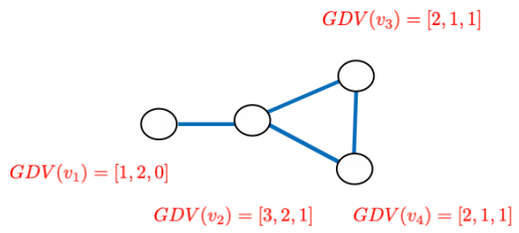
- Node centrality



- Clustering coefficient

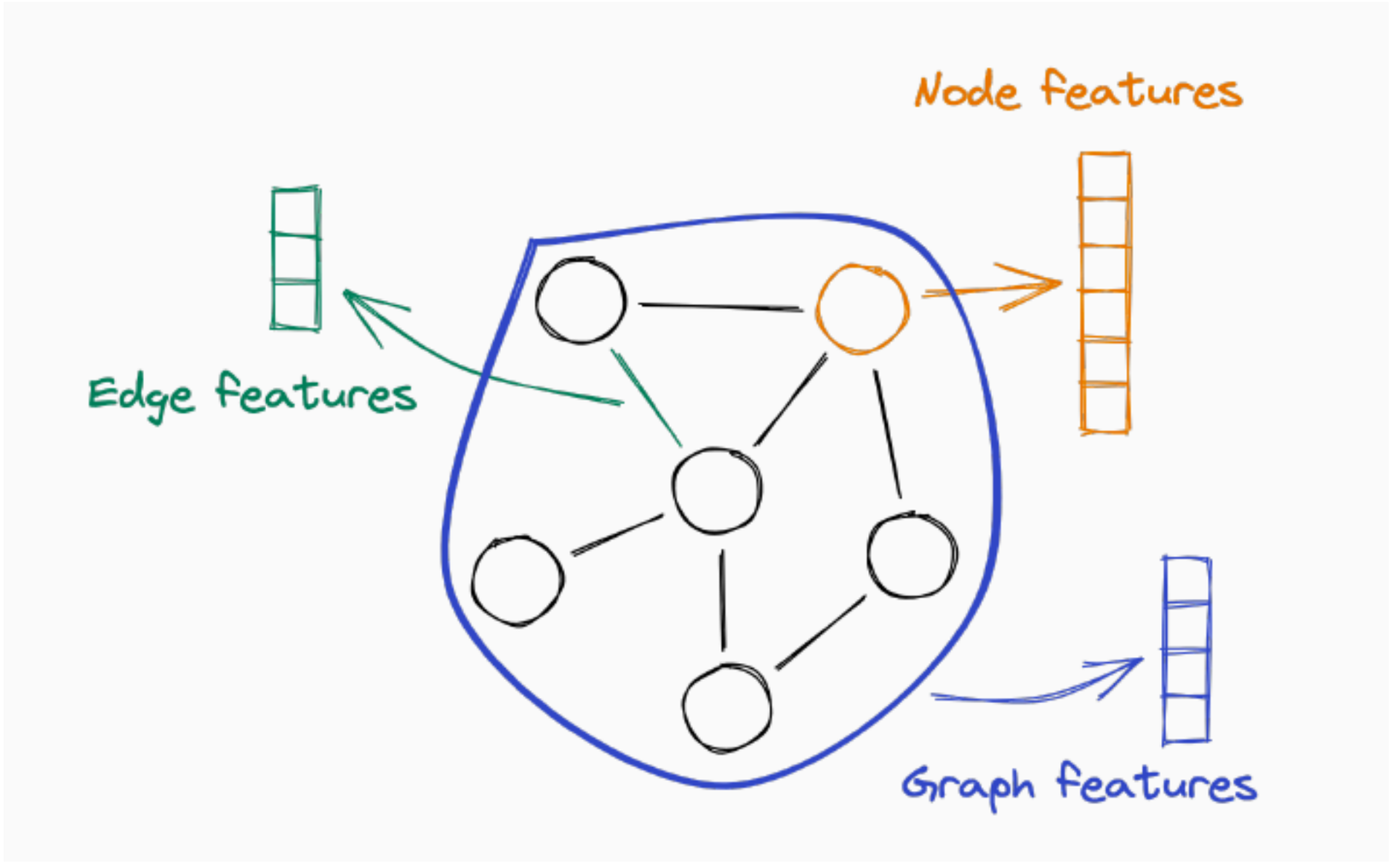


- Graphlets



Possible graphlets:

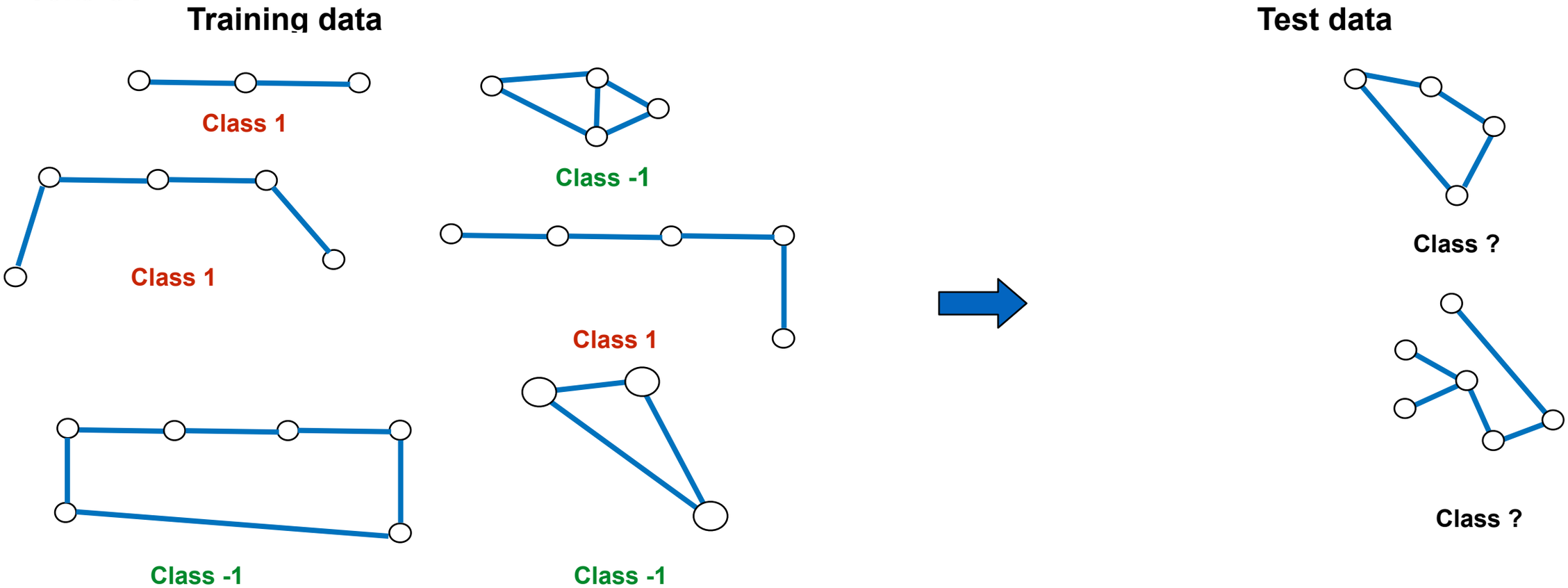
From node level to graph level task



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

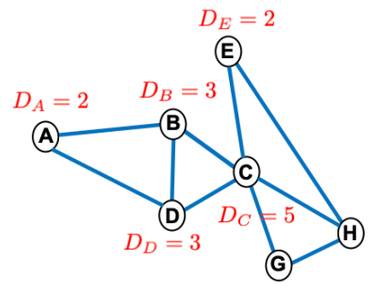
Illustrative example: Graph classification

- Common assumption: Graphs with similar structure have similar label

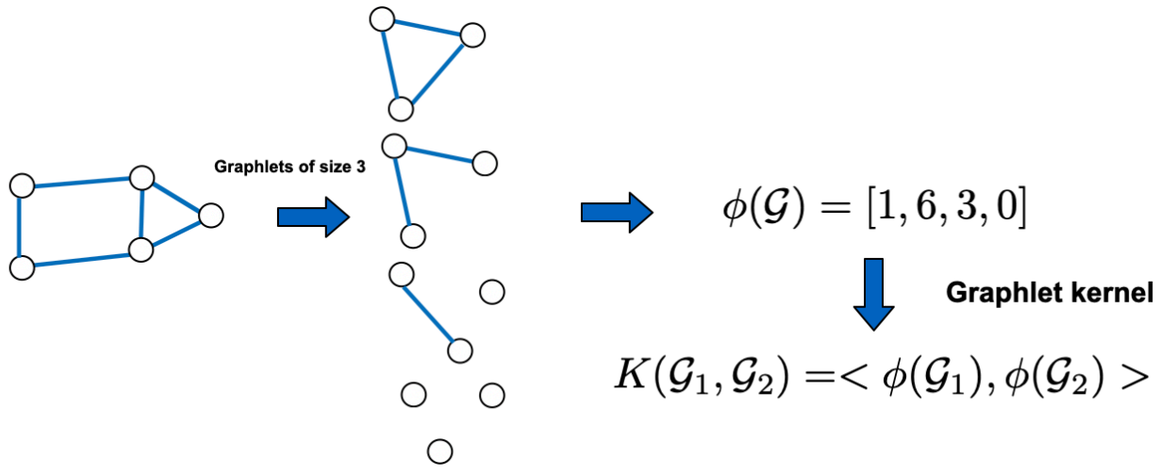


Graph level features

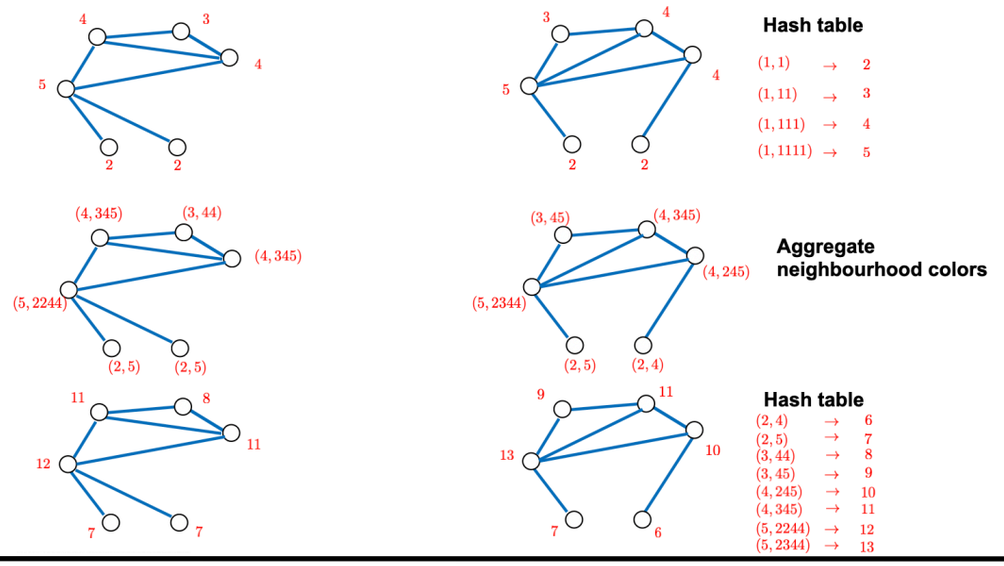
- Bag of nodes



- Graphlet kernel



- The Weisfeiler-Lehman kernel



Weisfeiler-Lehman kernel

- Iteratively aggregate information from node's neighbourhoods wider than the 1-hop neighborhood

- **Color refinement algorithm:**
  - **Input:** a graph $\mathcal{G}$
  - Assign an initial color $c^{(0)}(u)$ (e.g., node degree) to each node u of $\mathcal{G}$
  - For each iteration $k+1$ refine node colors as

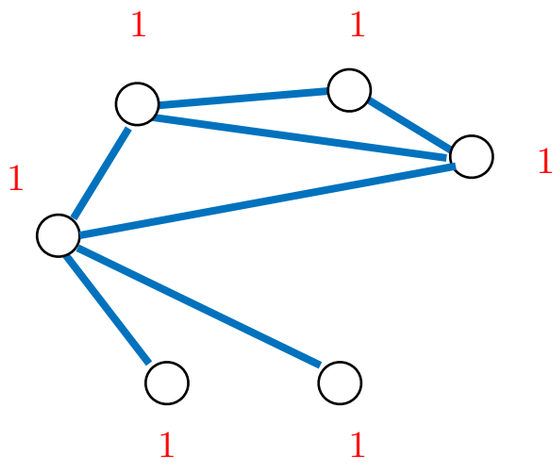
$$c^{(k+1)}(u) = \text{HASH}\left(\left\{c^{(k)}(u), \{c^{(k)}(v)\}_{v \in \mathcal{N}_v}\right\}\right)$$

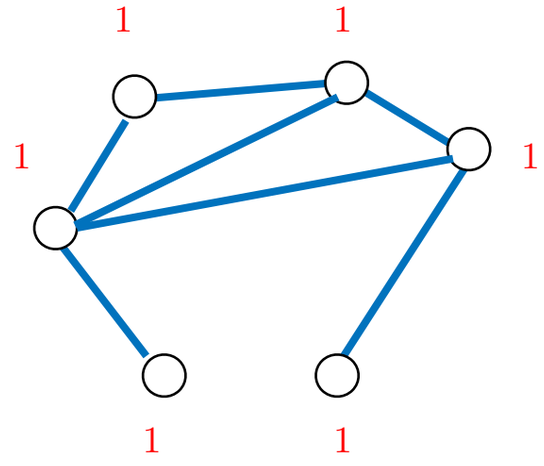
  - Output: The node color $c^{(K)}(u)$ after K iterations

- It provides a description of the K-hop neighborhood with an efficient algorithm

[Shervashidze et al., Weisfeiler-Lehman graph kernels, JMLR, 2011]

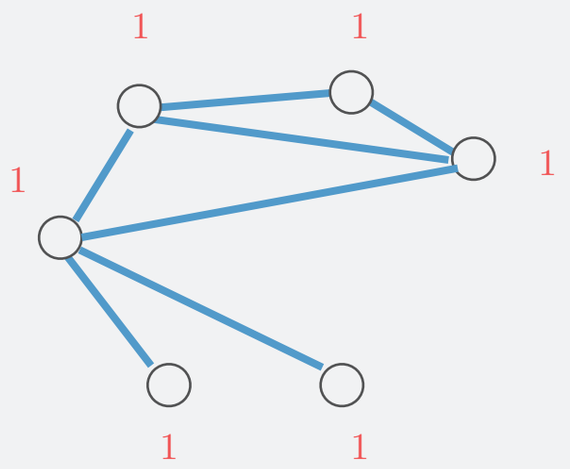
Example of WL kernel

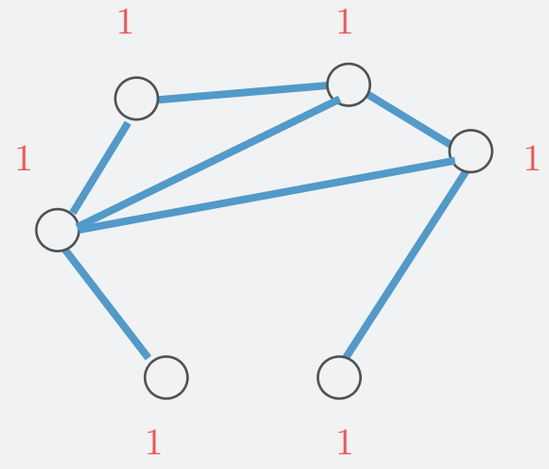




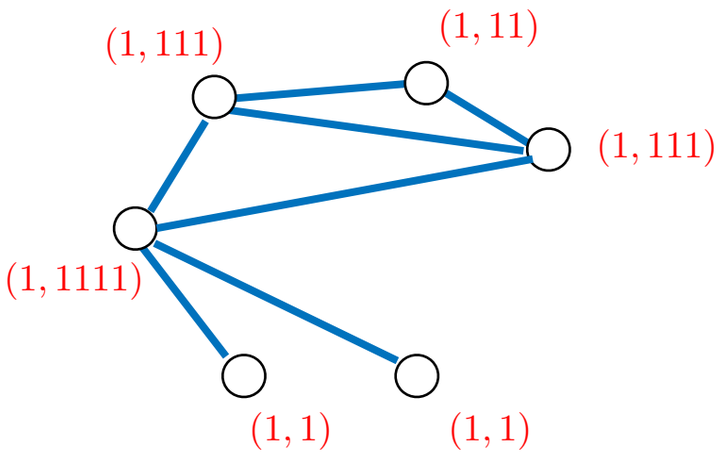
Initial colors

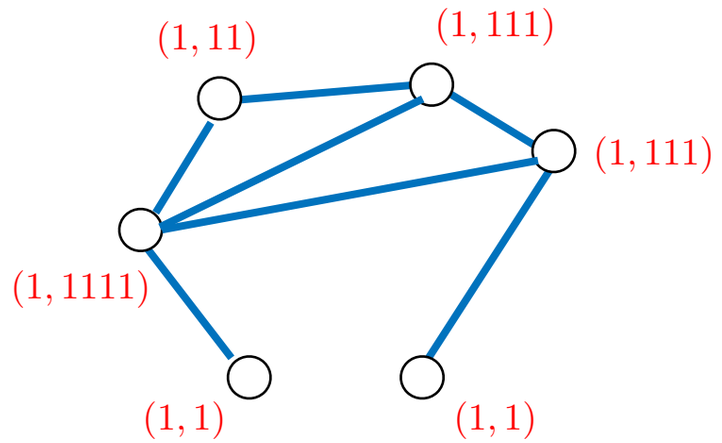
Example of WL kernel





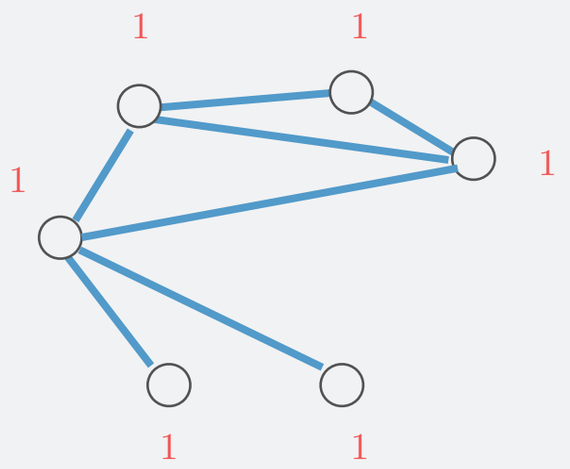
Initial colors

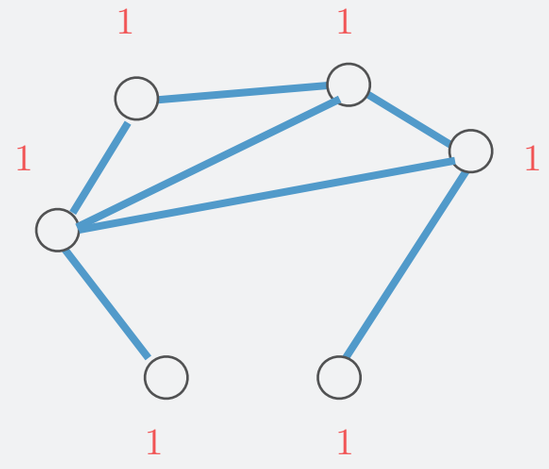




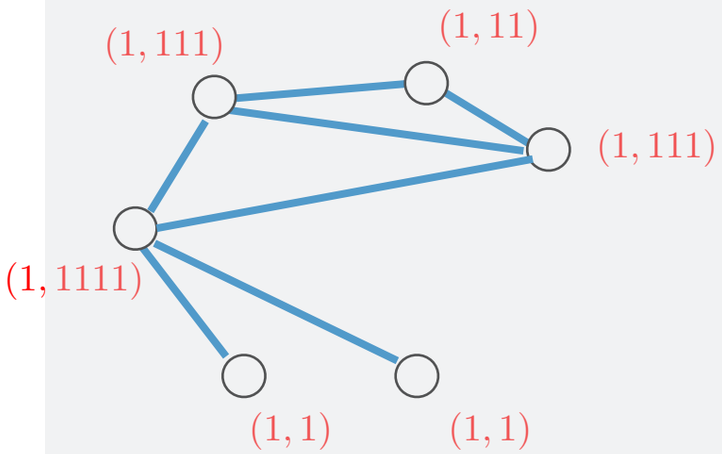
Aggregate neighbourhood colors

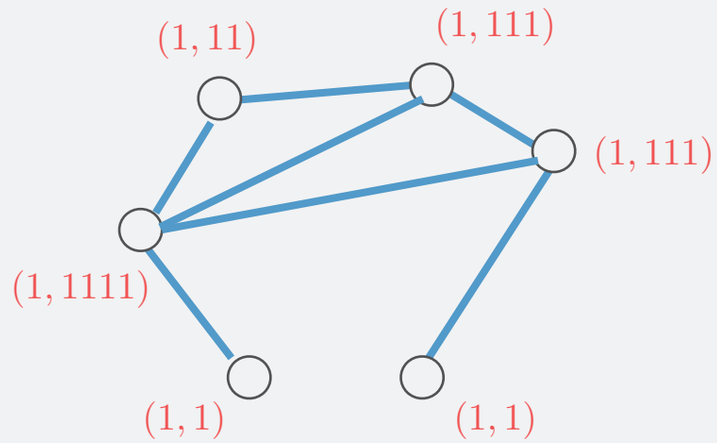
Example of WL kernel



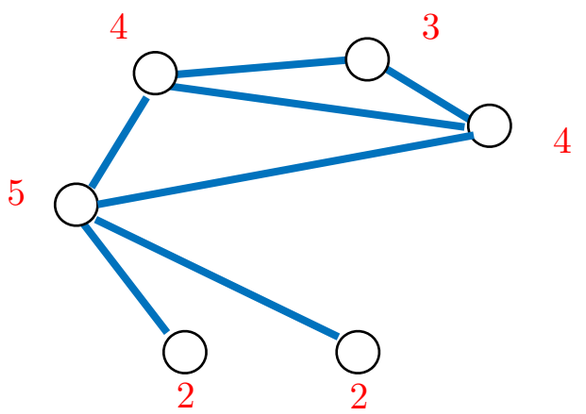


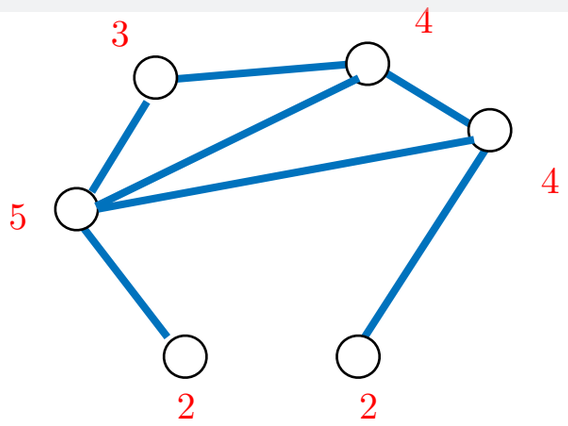
Initial colors





Aggregate neighbourhood colors

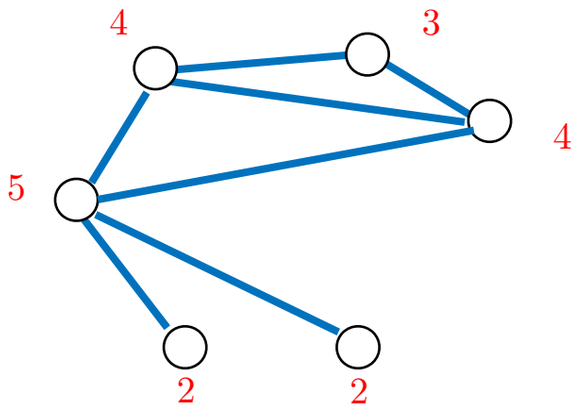


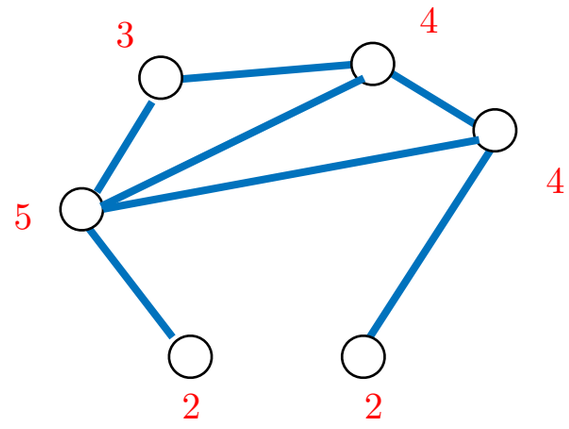


Hash table

$(1,1)$	$\rightarrow$	2
$(1,11)$	$\rightarrow$	3
$(1,111)$	$\rightarrow$	4
$(1,1111)$	$\rightarrow$	5

Example of WL kernel

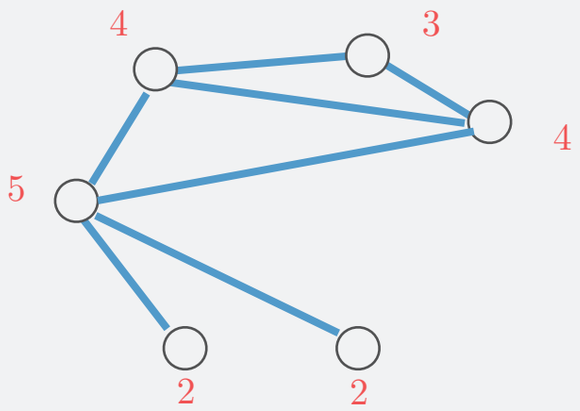


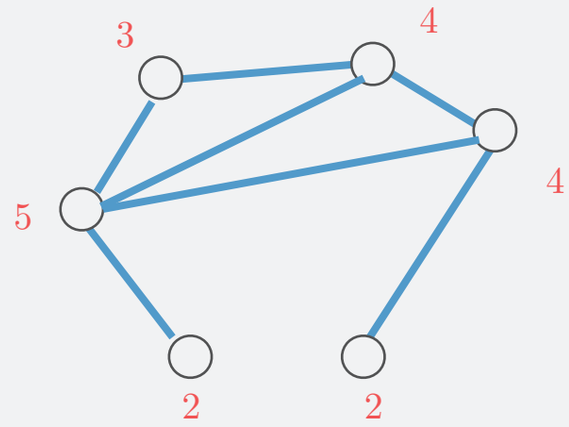


Hash table

$(1,1)$	$\rightarrow$	2
$(1,11)$	$\rightarrow$	3
$(1,111)$	$\rightarrow$	4
$(1,1111)$	$\rightarrow$	5

Example of WL kernel





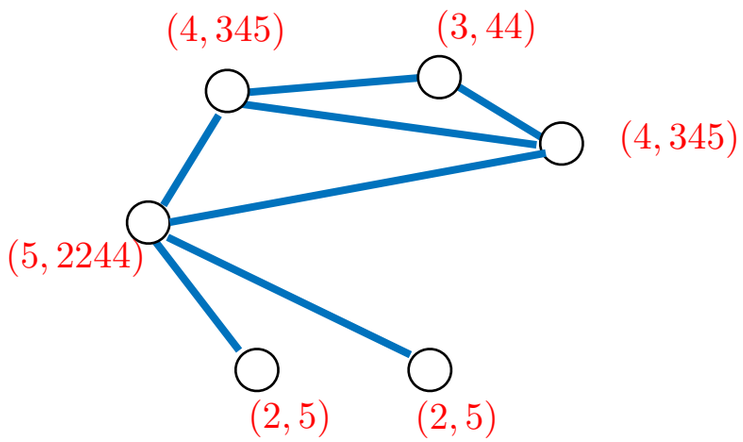
Hash table

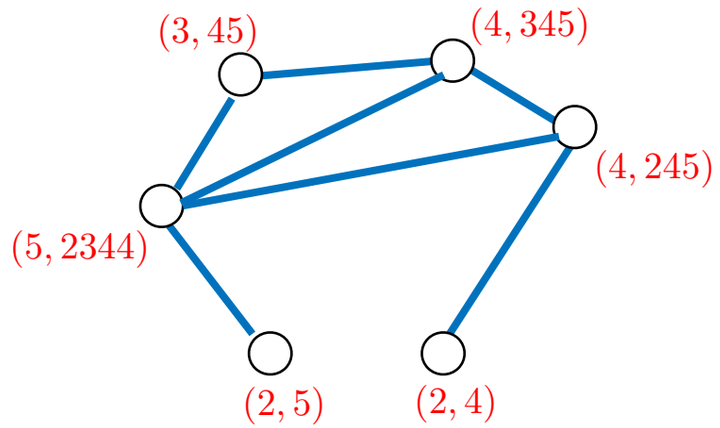
$(1,1) \rightarrow 2$

$(1,11) \rightarrow 3$

$(1,111) \rightarrow 4$

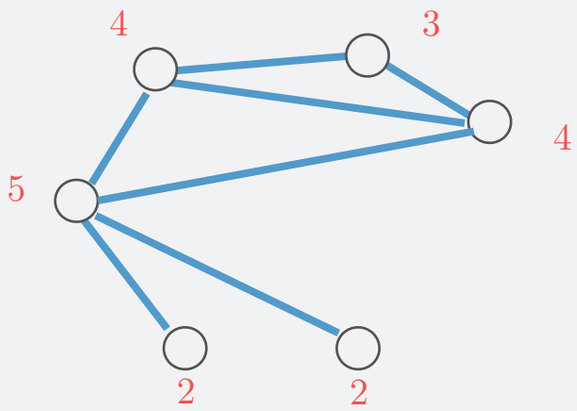
$(1,1111) \rightarrow 5$

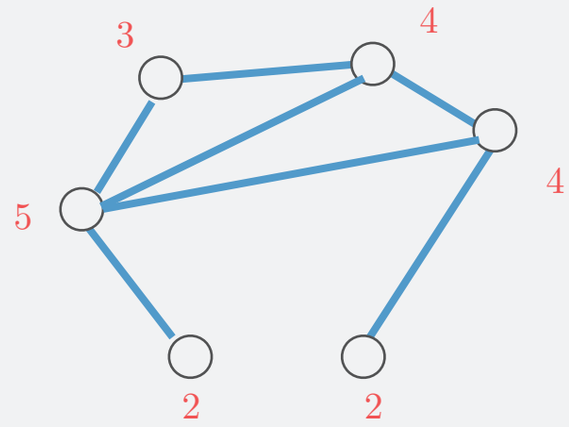




Aggregate neighbourhood colors

Example of WL kernel





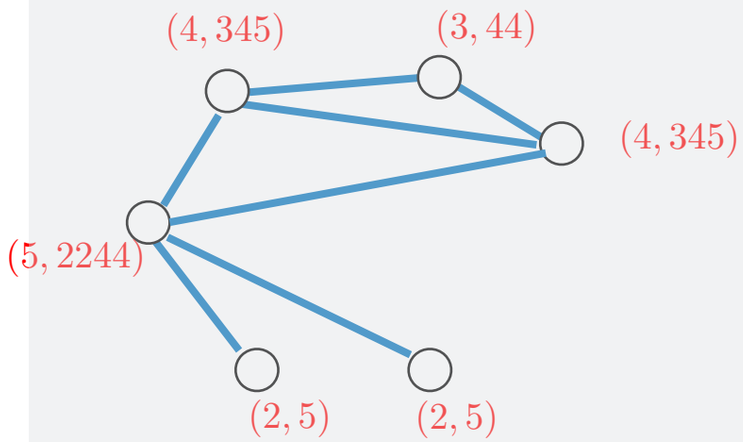
Hash table

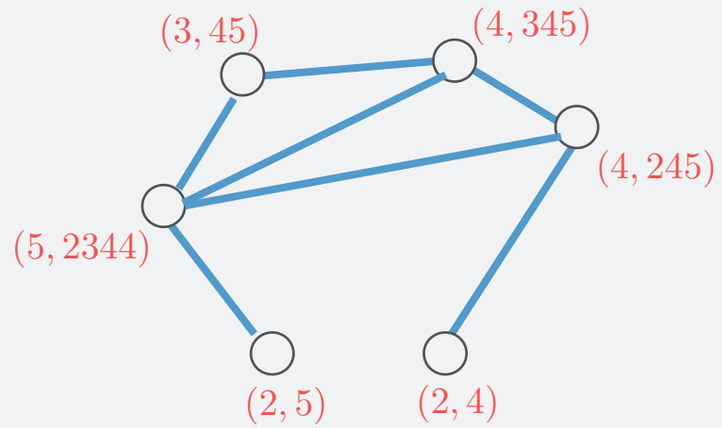
$(1,1) \rightarrow 2$

$(1,11) \rightarrow 3$

$(1,111) \rightarrow 4$

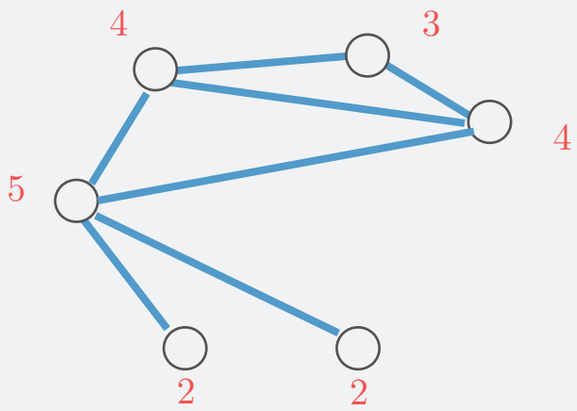
$(1,1111) \rightarrow 5$

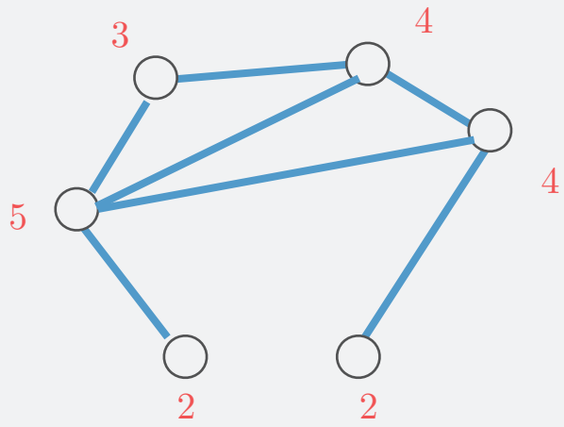




Aggregate neighbourhood colors

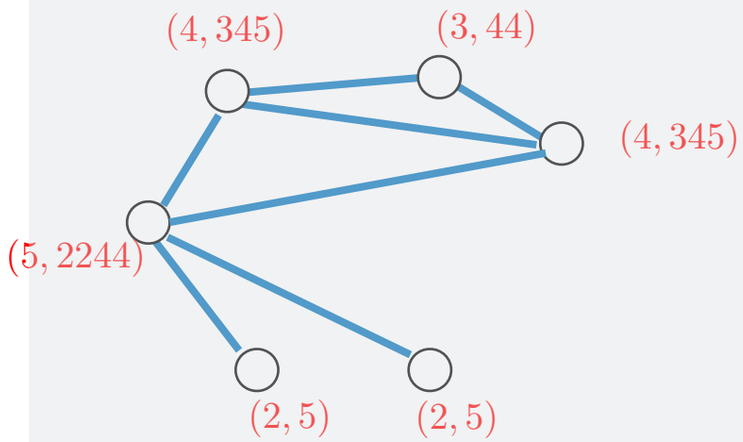
Example of WL kernel

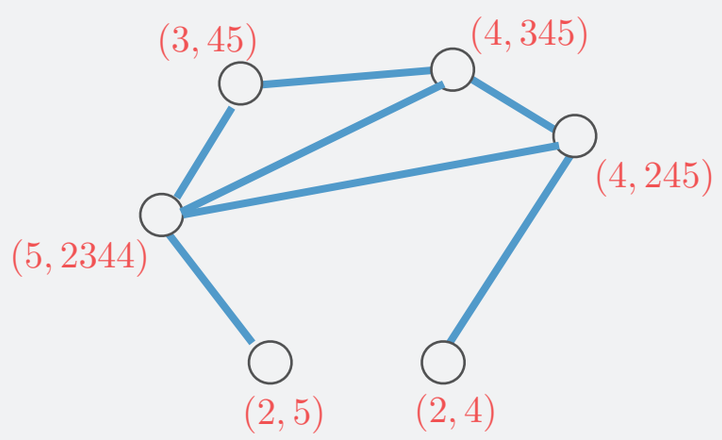




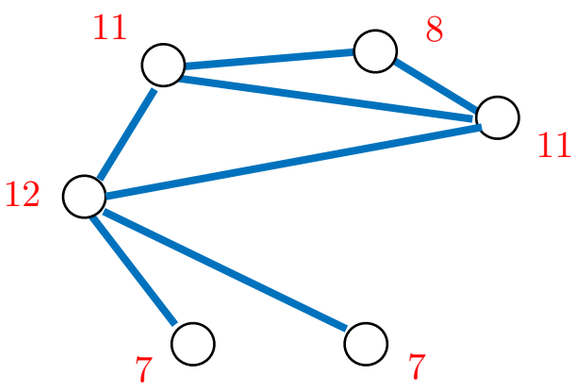
Hash table

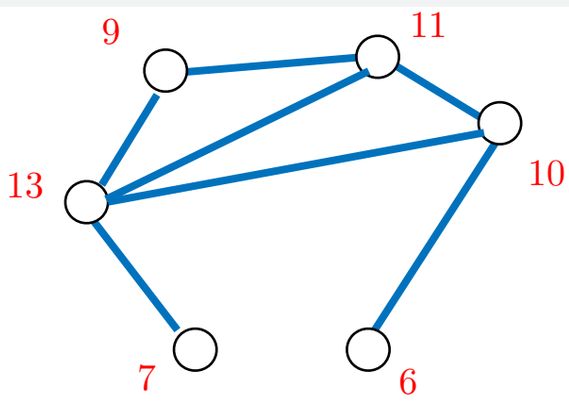
$(1,1)$	$\rightarrow$	2
$(1,11)$	$\rightarrow$	3
$(1,111)$	$\rightarrow$	4
$(1,1111)$	$\rightarrow$	5





Aggregate neighbourhood colors



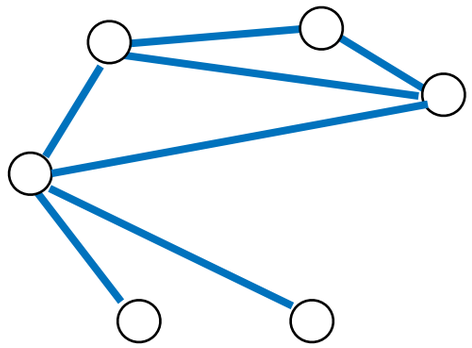


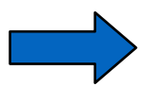
Hash table

$(2,4)$	$\rightarrow$	6
$(2,5)$	$\rightarrow$	7
$(3,44)$	$\rightarrow$	8
$(3,45)$	$\rightarrow$	9
$(4,245)$	$\rightarrow$	10
$(4,345)$	$\rightarrow$	11
$(5,2244)$	$\rightarrow$	12
$(5,2344)$	$\rightarrow$	13

Example of WL kernel

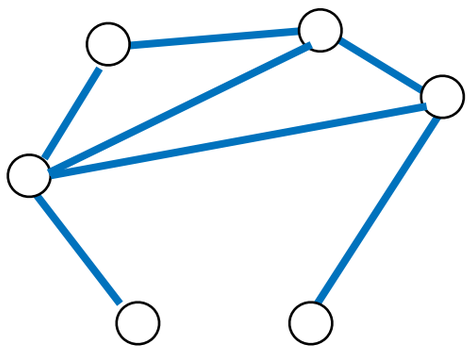
- After K iterations, the WL kernel computes the histogram of colors

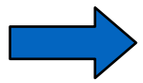




$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13$$

$$\phi(\mathcal{G}_1) = [6, 2, 1, 2, 1, 0, 2, 1, 0, 0, 2, 1, 0]$$





$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13$$

$$\phi(\mathcal{G}_2) = [6, 2, 1, 2, 1, 1, 1, 0, 1, 1, 1, 0, 1]$$

WL kernel

$$K(\mathcal{G}_1, \mathcal{G}_2) = < \phi(\mathcal{G}_1), \phi(\mathcal{G}_2) >$$

Limitations of hand-crafted graph features

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

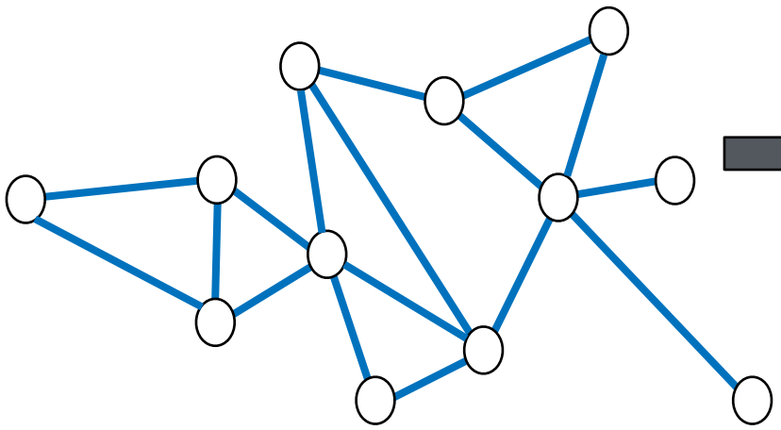
- Designing graph features can very often be a time consuming and expensive process

- 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



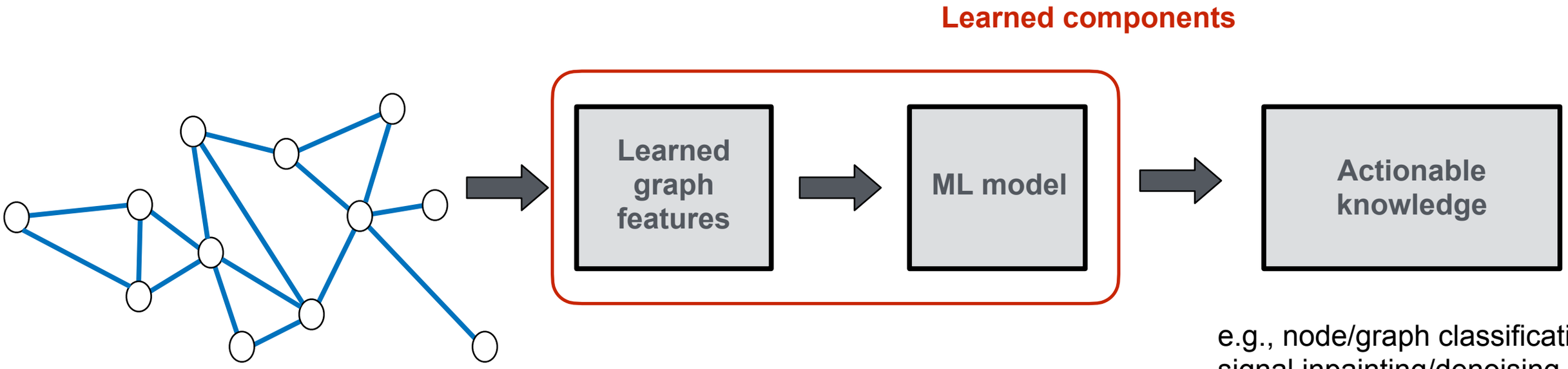
e.g., node/graph classification, signal inpainting/denoising

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



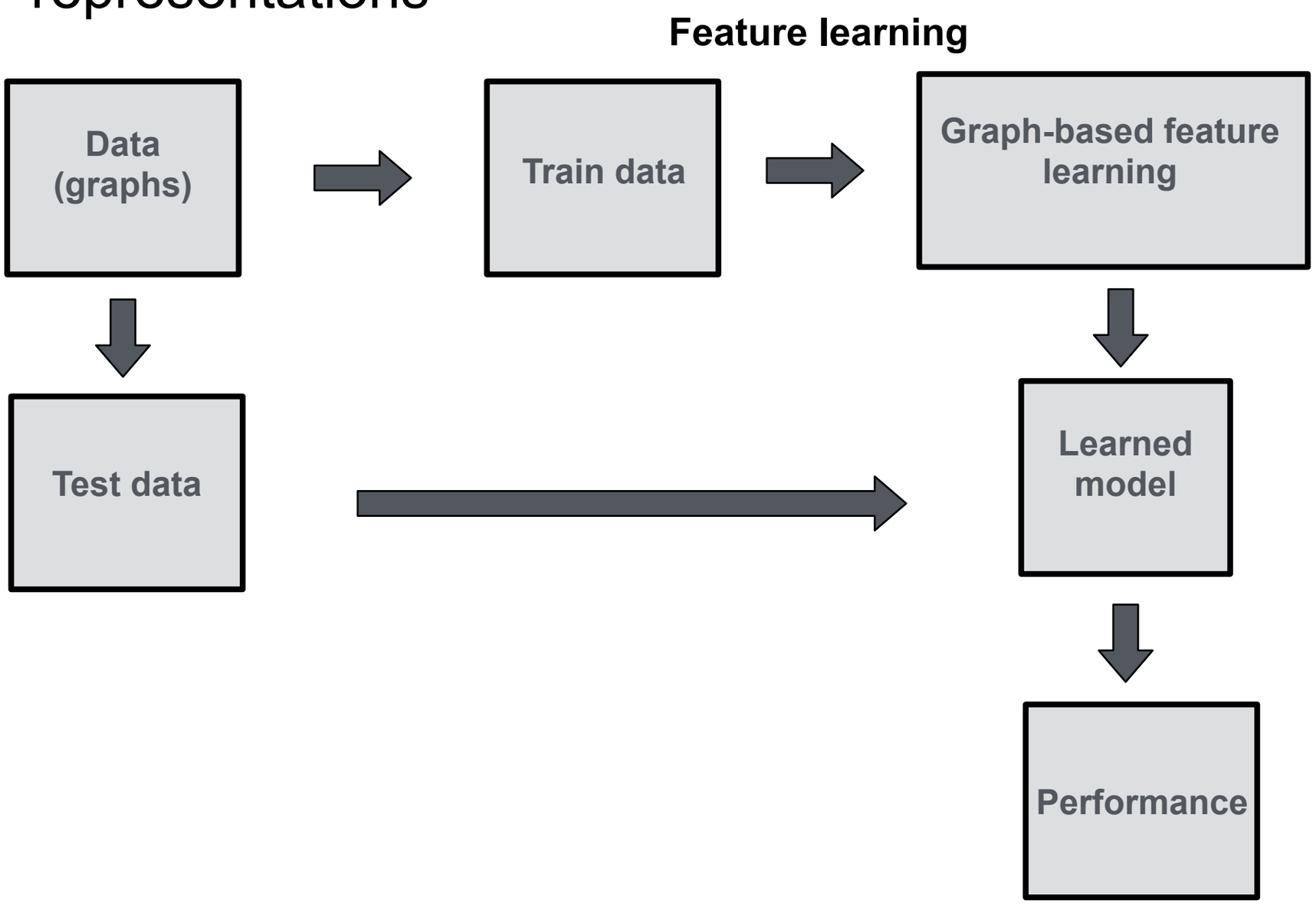
e.g., node/graph classification, signal inpainting/denoising

Feature Learning

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

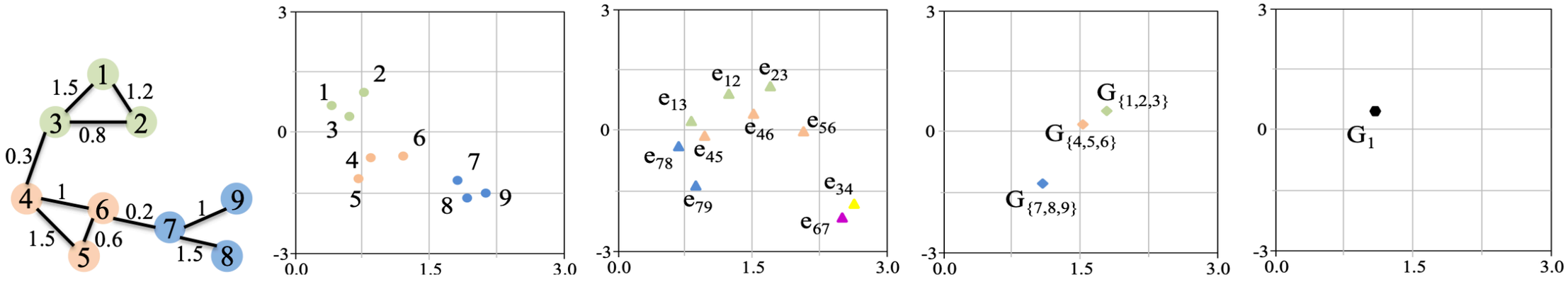
Graph representation learning: basic pipeline

- 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



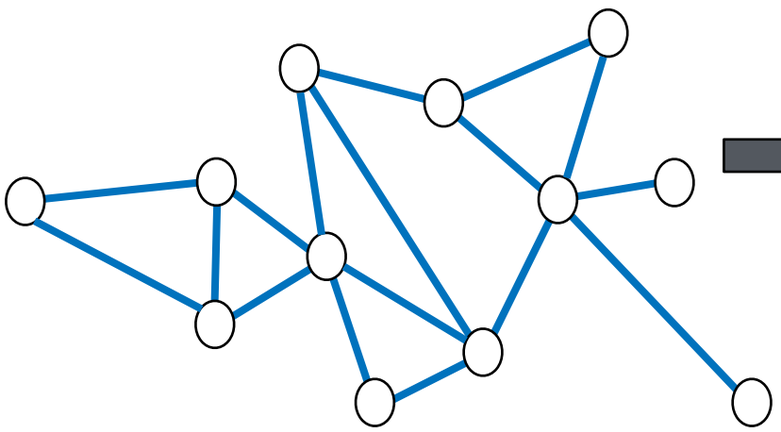
Original space | **Node embedding** | **Edge embedding** | **Subgraph embedding** | **Graph embedding**

Embedding space

How can we learn the embedding space?

Supervised graph representation learning

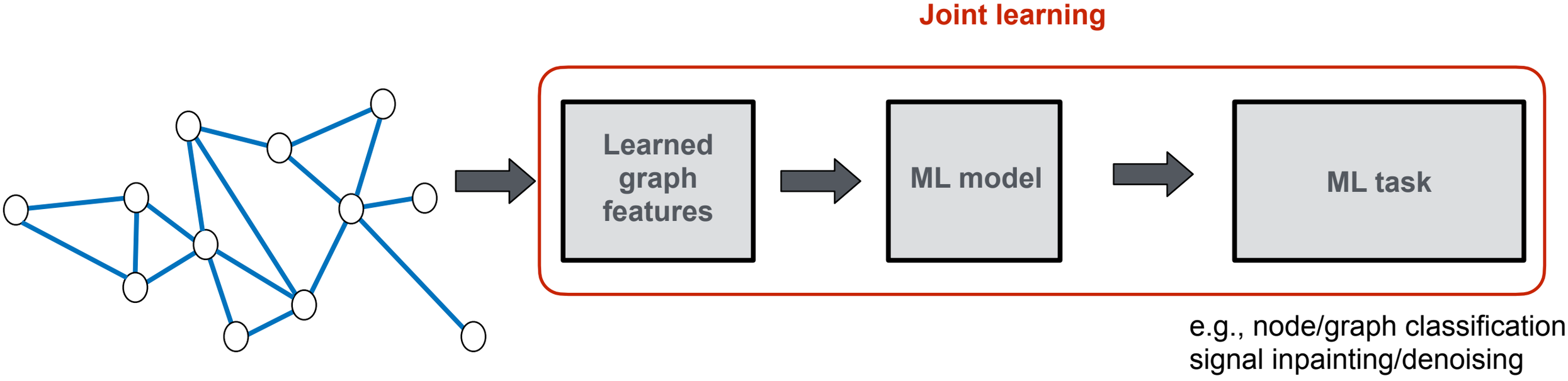
- Learn low-dimensional embeddings for a specific downstream task, e.g., node or graph classification



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

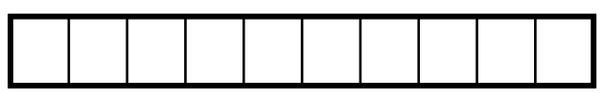


Unsupervised graph representation learning

- 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

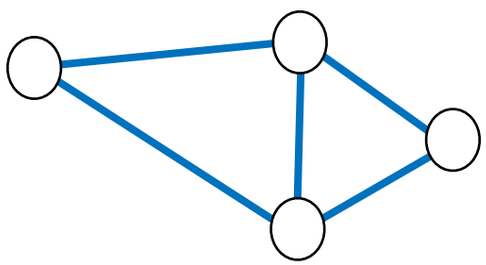


Example of tasks

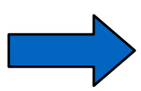
1. Node/graph classification
2. Node/graph clustering
3. Link prediction
4. Visualization
5. …

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



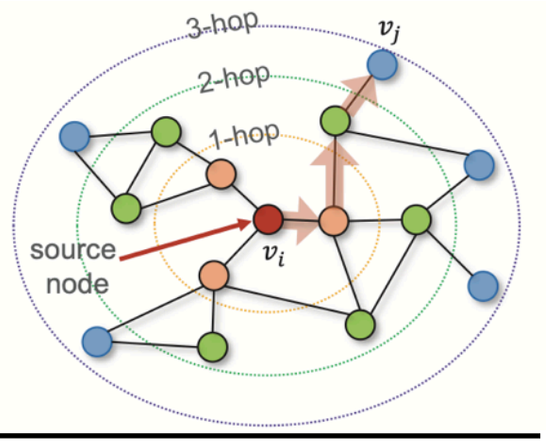
$\mathcal{G}$



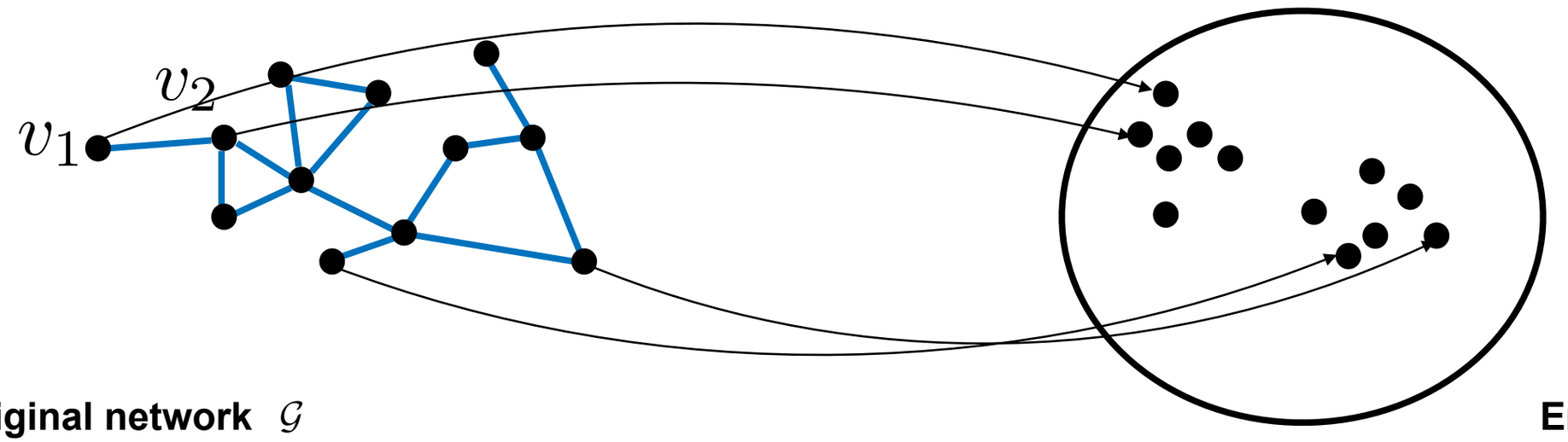
Learning algorithm

Embedding vector/Representations

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



Illustrative example: Node embeddings



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**

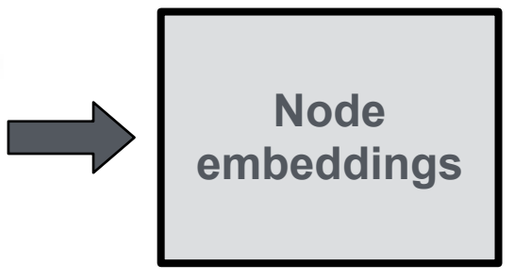
$$LY = Y\Gamma \quad \Rightarrow \quad \boxed{u_i \rightarrow (\chi_2(i), ..., \chi_{K+1}(i))}$$

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

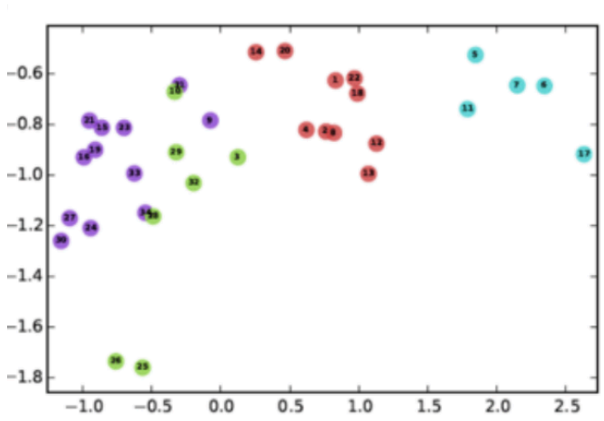
[Belkin et al, 2003, Laplacian Eigenmaps for Dimensionality Reduction and Data Representation, Neural Comp.]

Application of node embeddings: Node clustering/Community

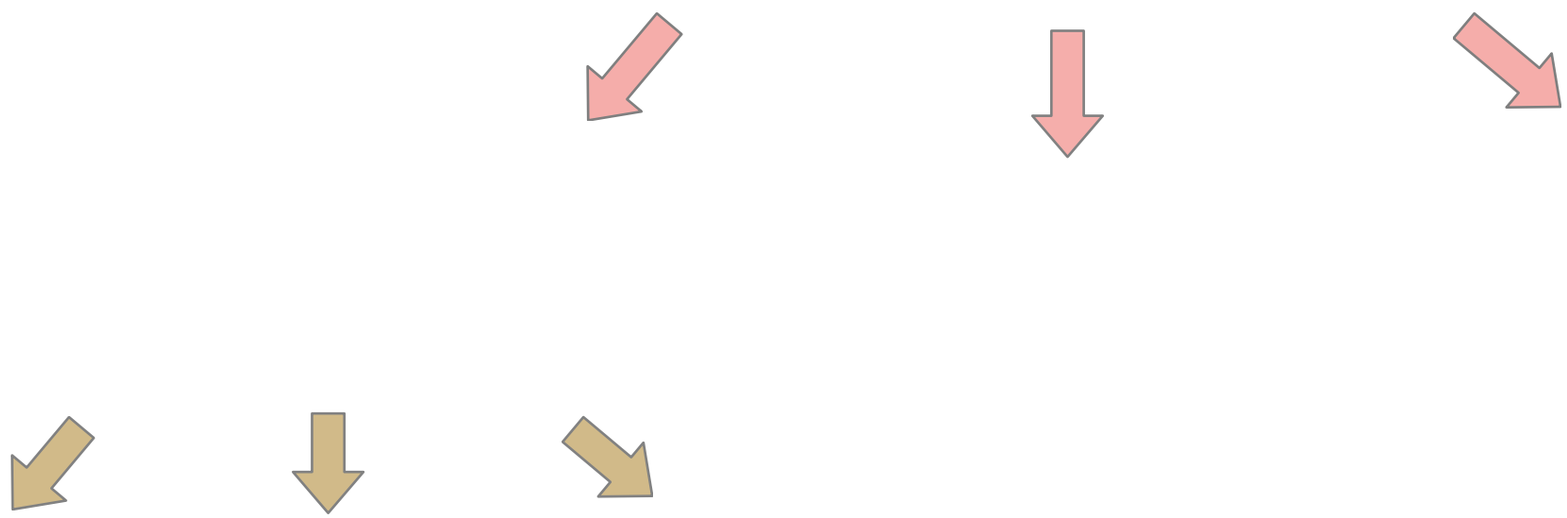
- The karate-club example
  - Compute node embeddings
  - Apply any clustering algorithm (e.g., K-means) on the learned embeddings



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

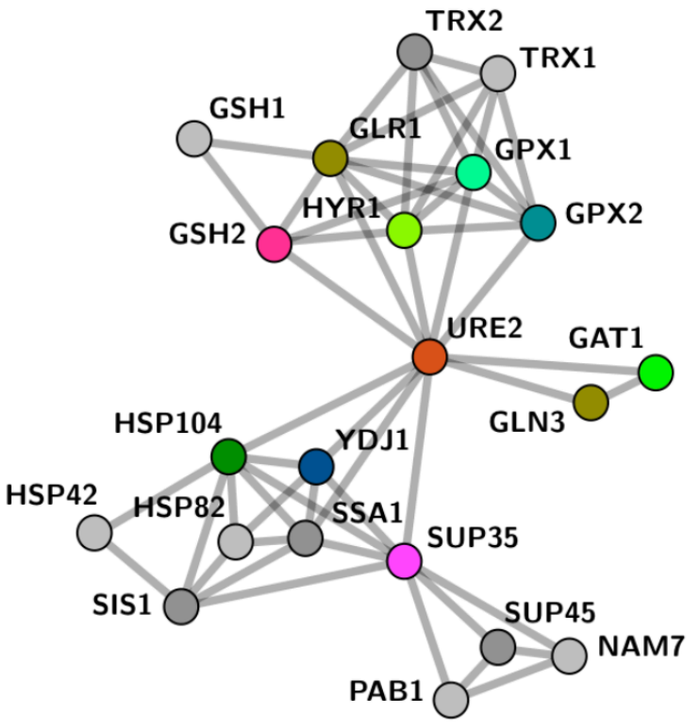
- Do not incorporate node attributes

- Independent of downstream tasks

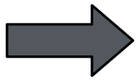
- No parameter sharing:
  - Every node has its own unique embedding

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

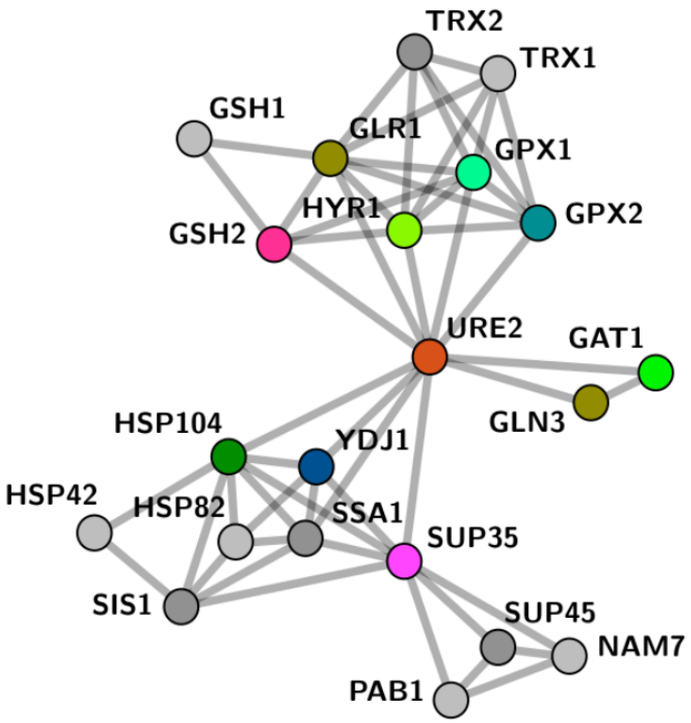


Actionable knowledge

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

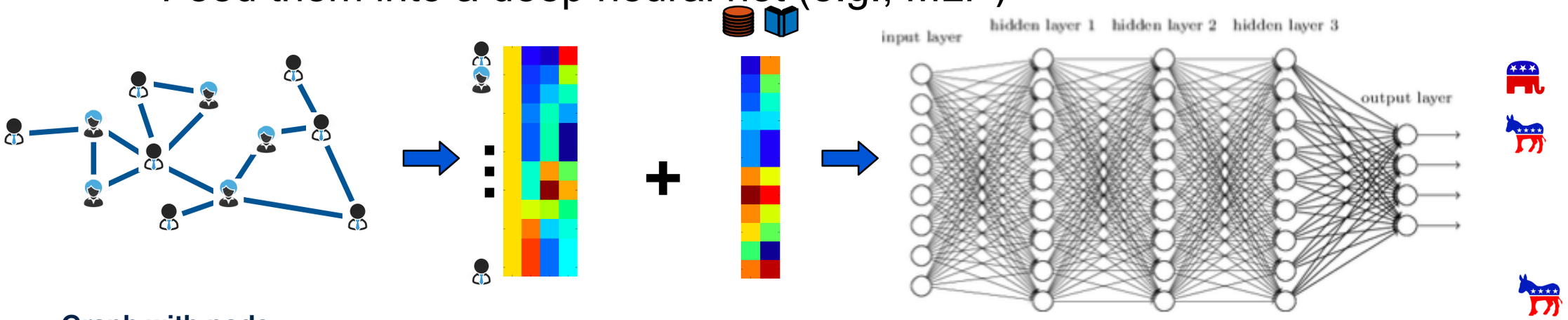
Graph neural networks

Actionable knowledge

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

Computing embeddings from graphs with node attributes

- 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

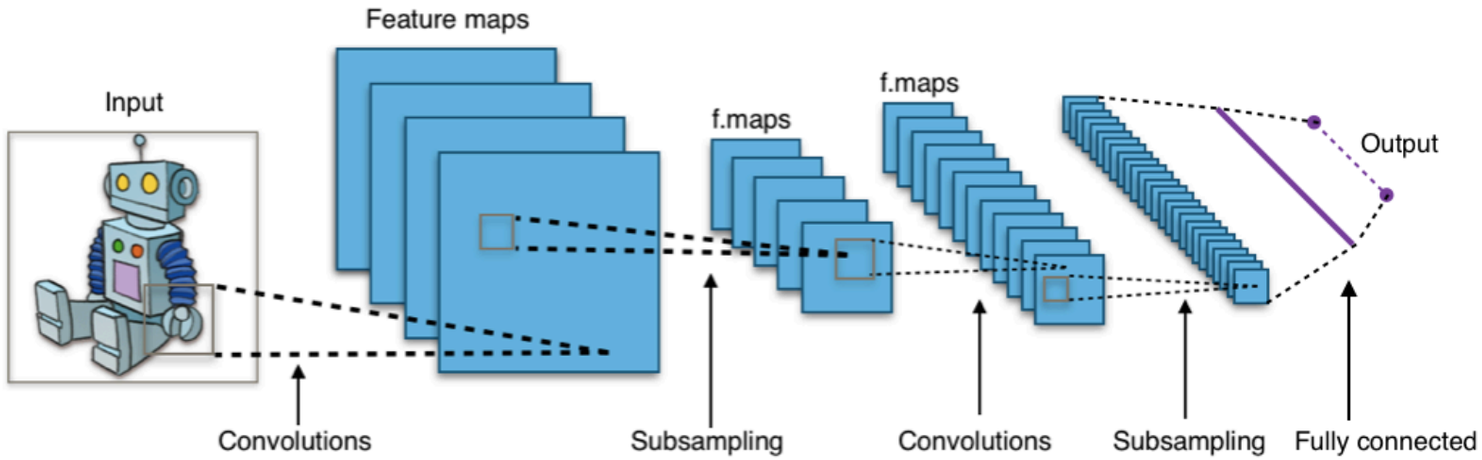
- We build intuition from classical deep learning algorithms
- CNNs exploit structure in the images

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

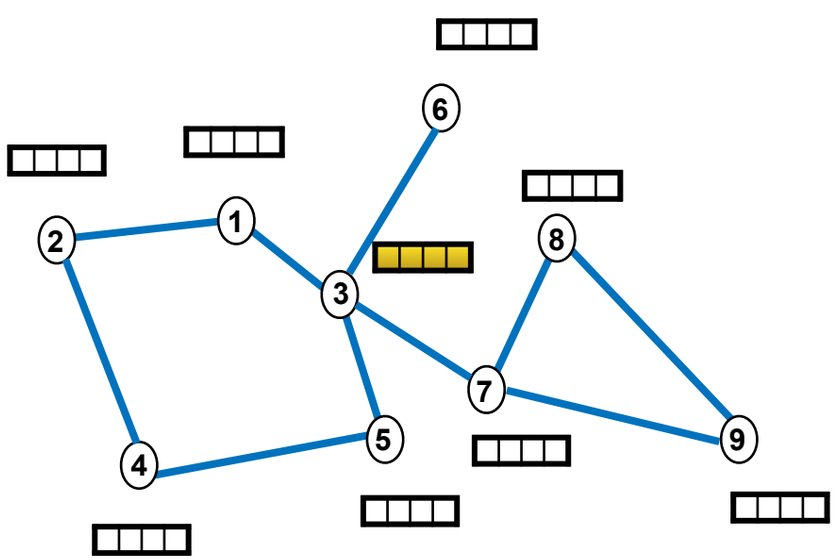
  - **Efficiency:** how to keep the computational complexity low

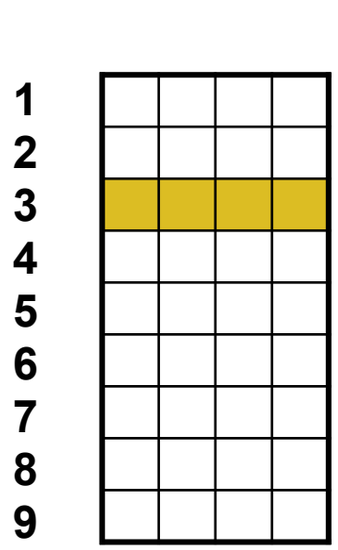
  - **Generalization:** how to build models that generalize to unseen graphs

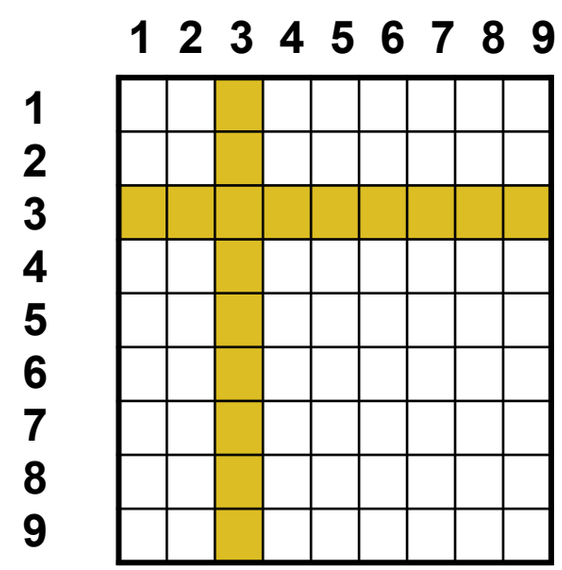
Permutation Invariance

- Graph structure is independent of the labelling of the nodes or from how we choose to draw them

- Graph and node representations should be permutation invariant
  - Graph representations should be invariant to the order of the nodes
  - Similar intuition to translation invariance on images



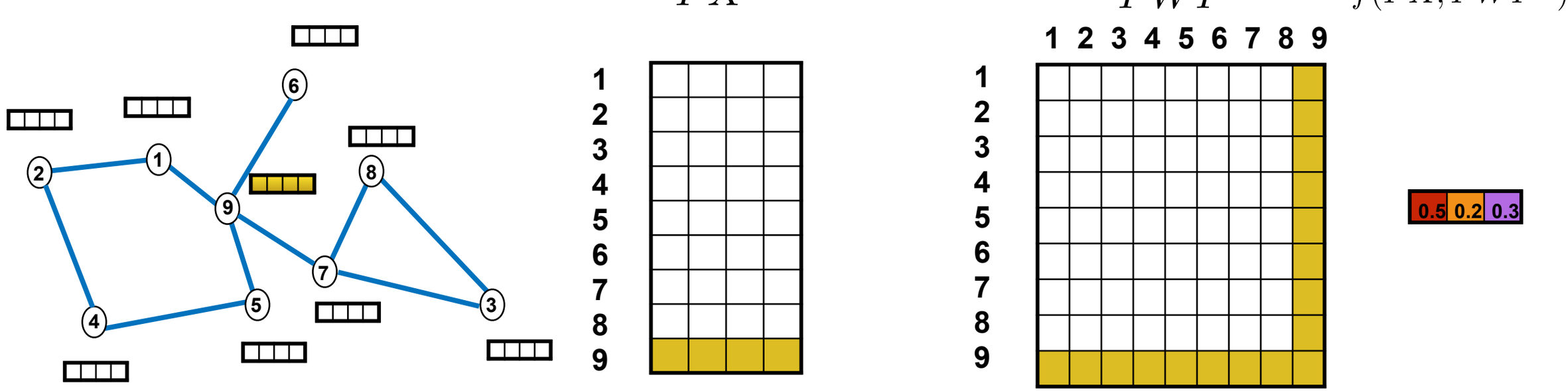




Permutation Invariance

- Graph structure is independent of the labelling of the nodes or from how we choose to draw them

- Graph and node representations should be permutation invariant
  - Graph representations should be invariant to the order of the nodes
  - Similar intuition to translation invariance on images

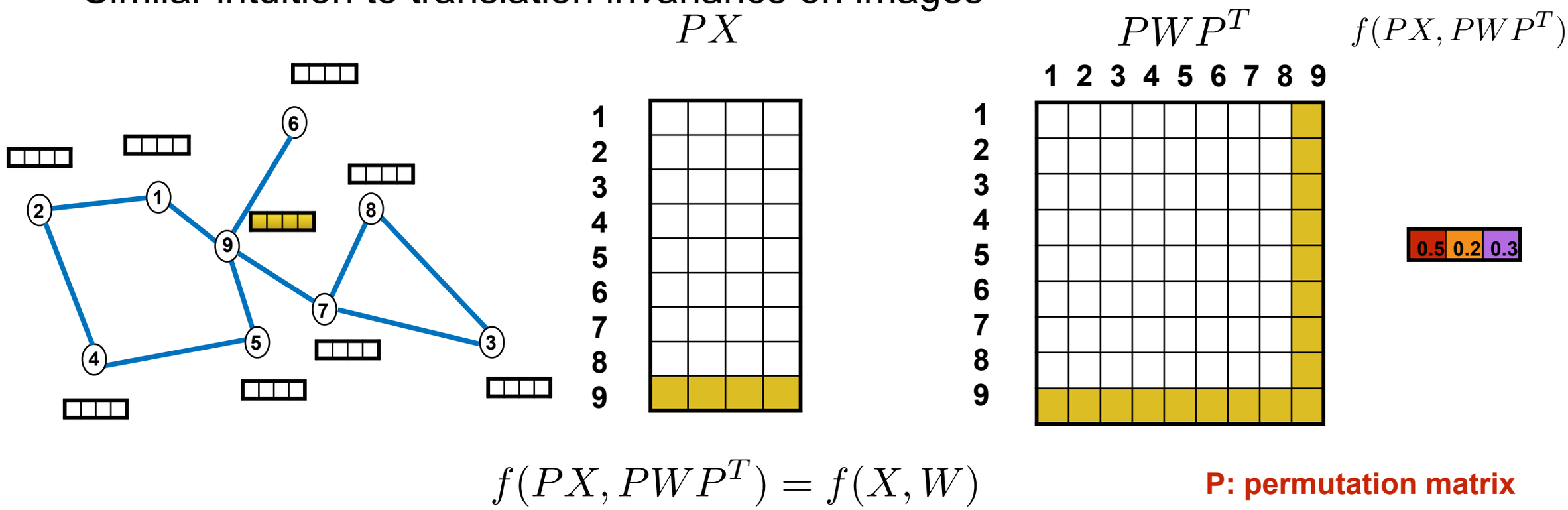


P: permutation matrix

Permutation Invariance

- Graph structure is independent of the labelling of the nodes or from how we choose to draw them

- Graph and node representations should be permutation invariant
  - Graph representations should be invariant to the order of the nodes
  - Similar intuition to translation invariance on images



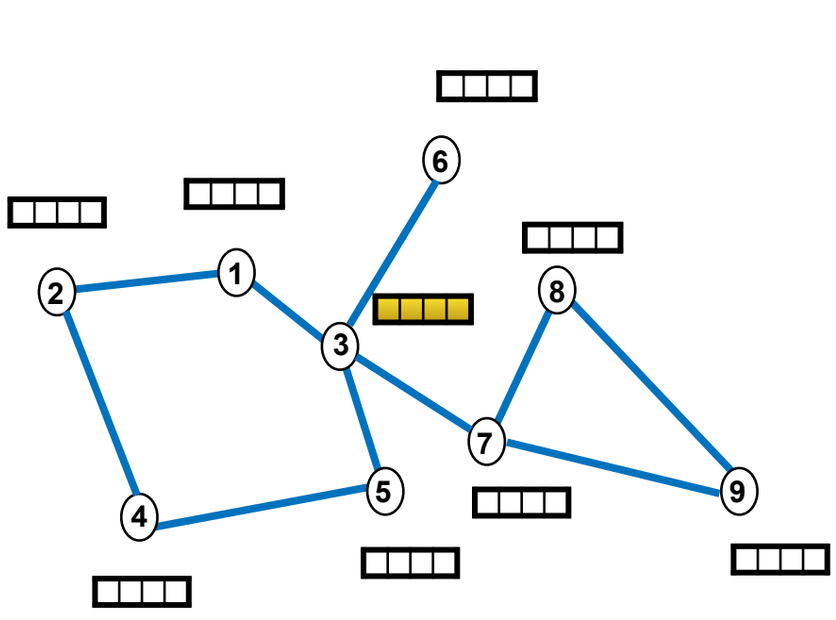
$$f(PX, PWP^T) = f(X, W)$$

P: permutation matrix

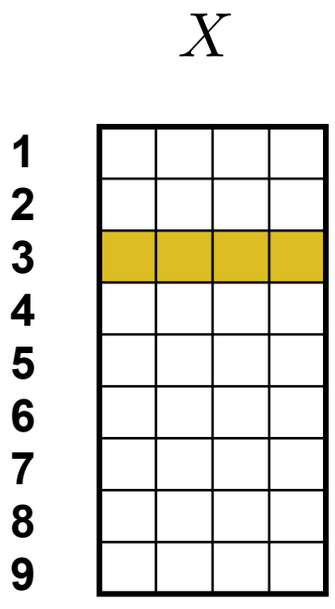
Permutation Equivariance

- Graph structure is independent of the labelling of the nodes or from how we choose to draw them

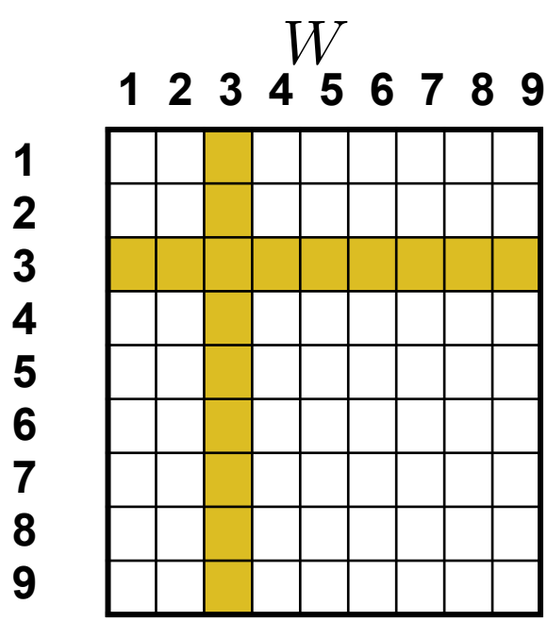
- Node representations should be permutation equivariant
  - If we permute the nodes of the graph, the nodes' output should be permuted in the same way



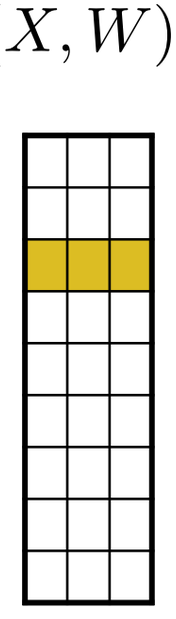
X



W



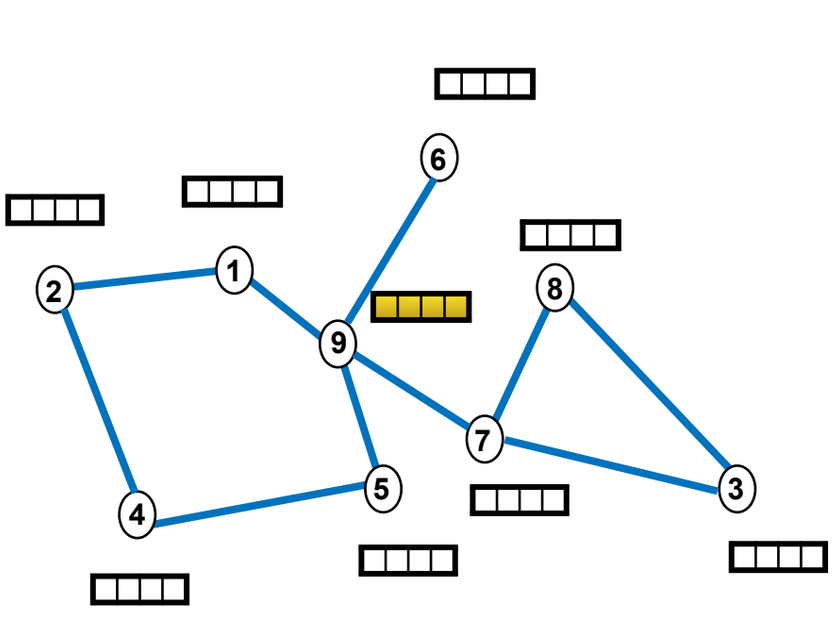
$f(X, W)$



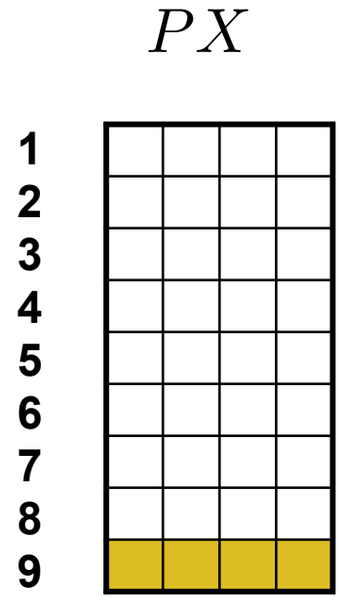
Permutation Equivariance

- Graph structure is independent of the labelling of the nodes or from how we choose to draw them

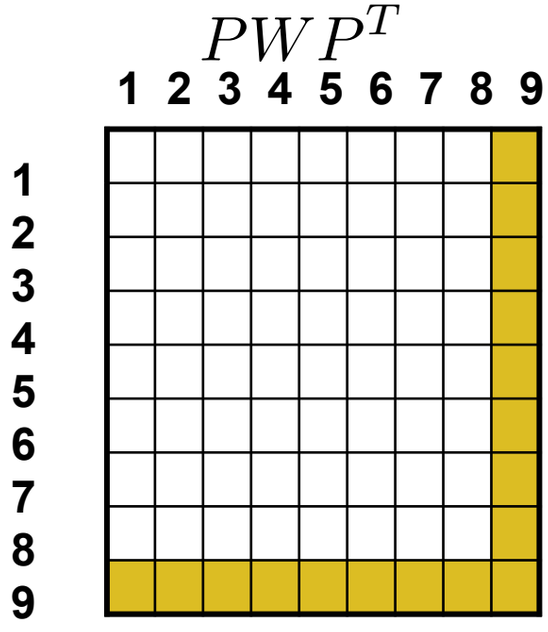
- Node representations should be permutation equivariant
  - If we permute the nodes of the graph, the nodes' output should be permuted in the same way



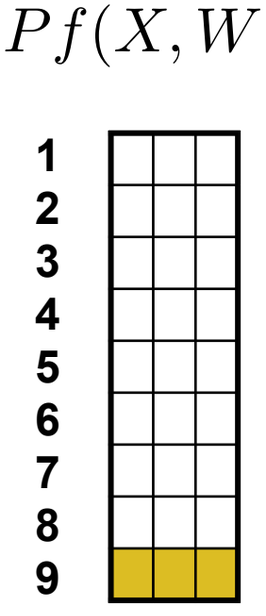
PX



PWP^T



$Pf(X,W)$

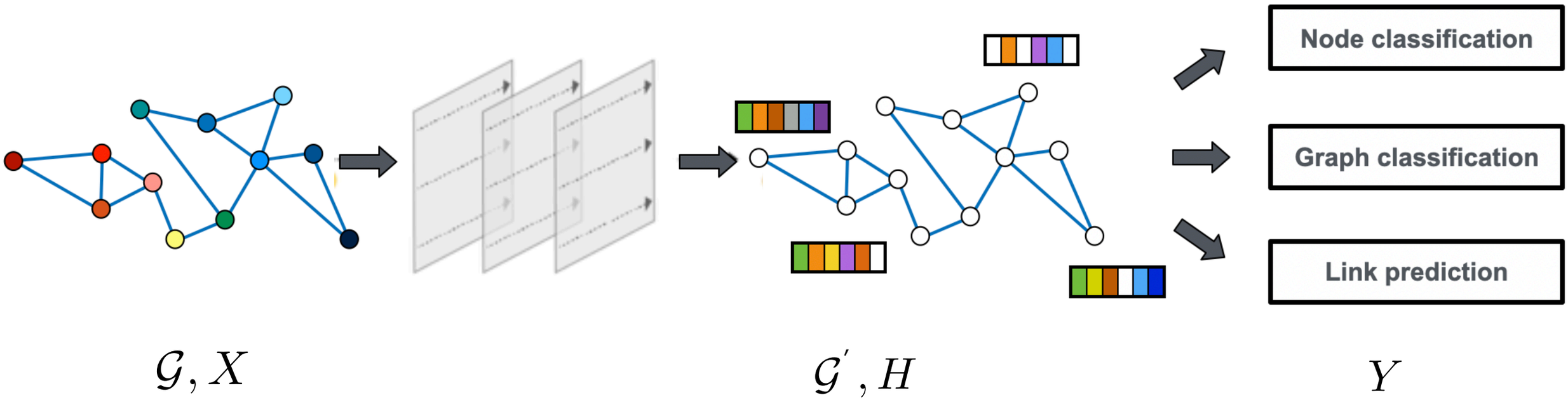


$$f(PX, PWP^T) = Pf(X,W)$$

P: permutation matrix

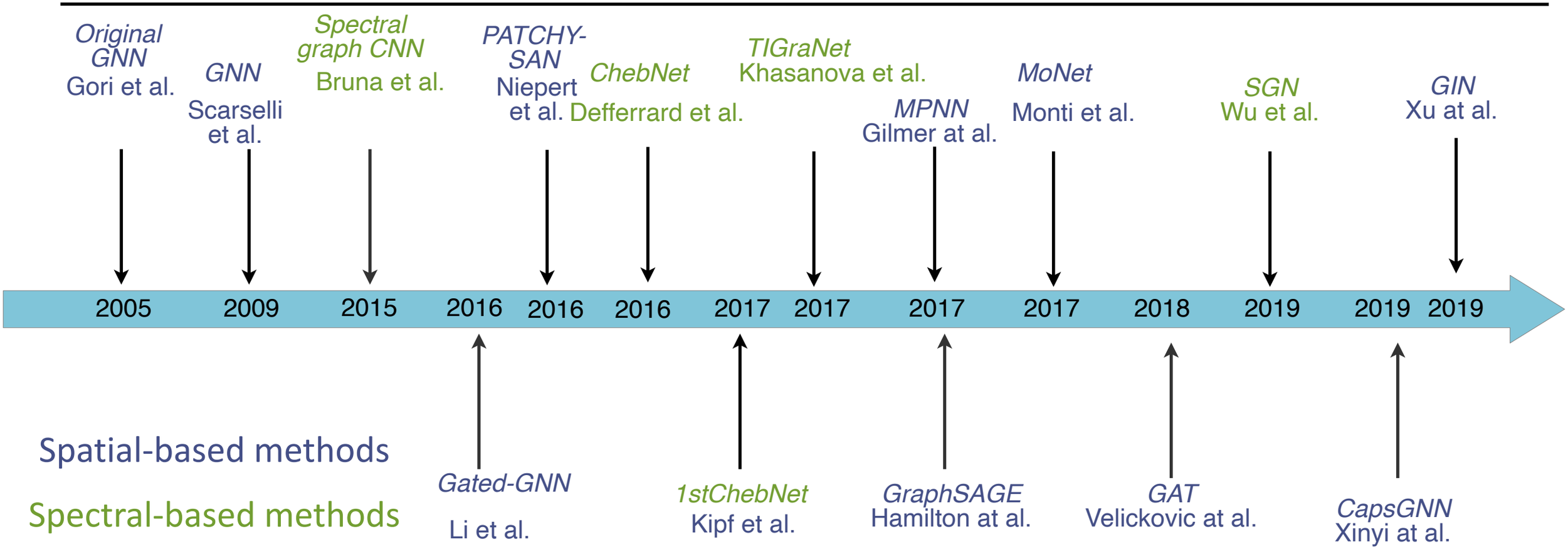
GNN model: schematic overview

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



- We can construct permutation equivariant functions by applying permutation invariant functions on local neighbourhoods

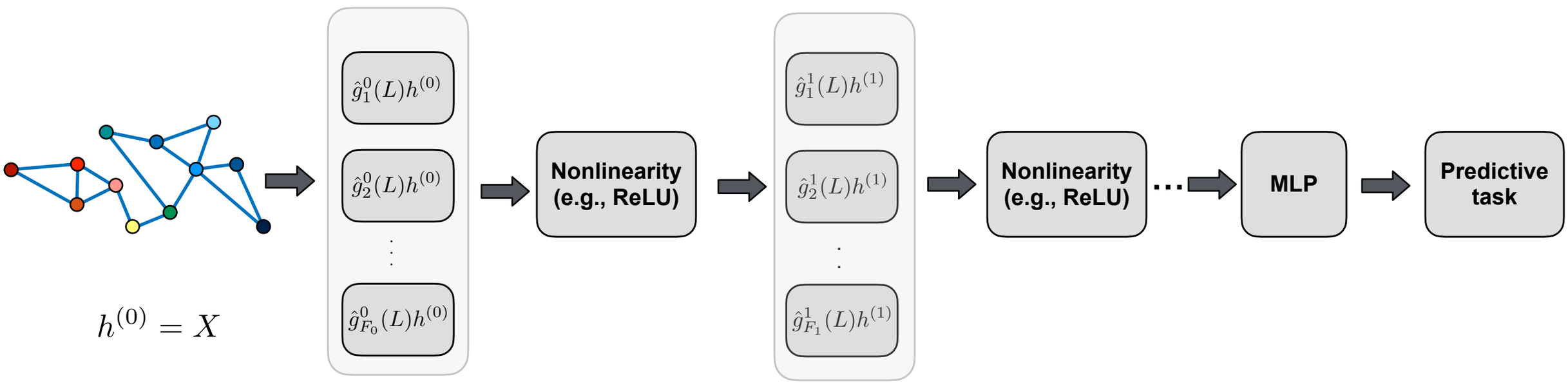
First GNN architectures



- 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

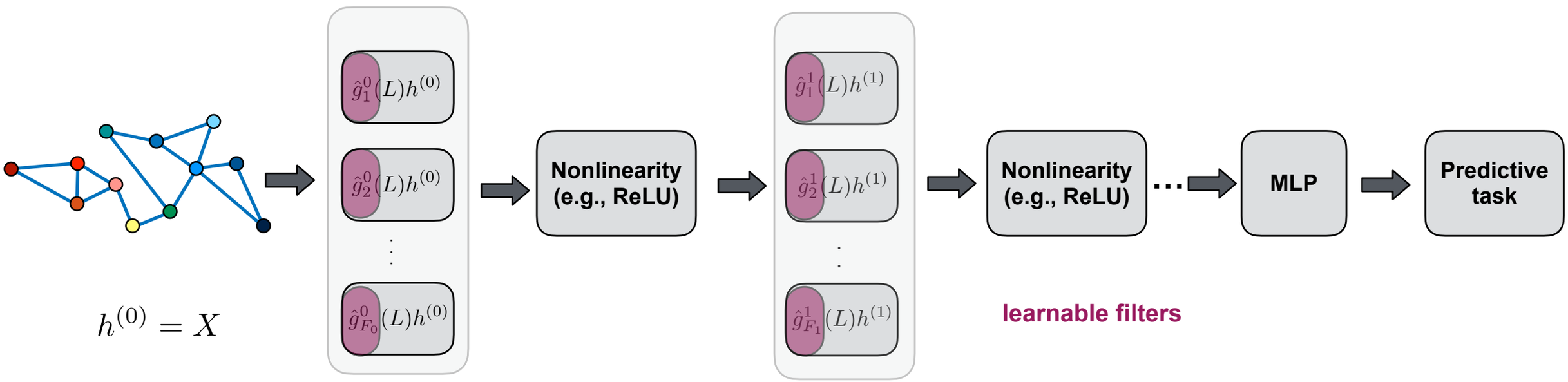
- Typical GNN architectures consist of a set of graph convolutional layers, each of which is followed by elementwise nonlinearity



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

The basic GNN: a spectral viewpoint

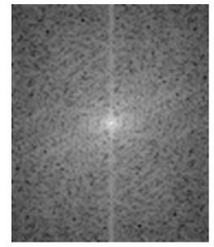
- Typical GNN architectures consist of a set of graph convolutional layers, each of which is followed by element-wise nonlinearity



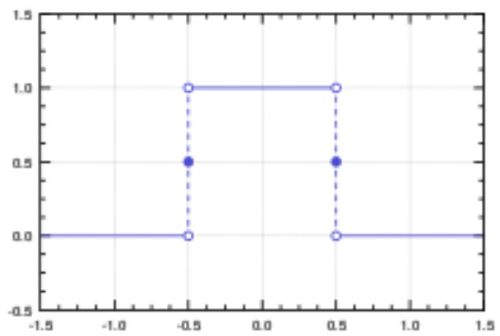
- By learning the parameters of the each filter, we learn how to propagate information on a graph to compute node embeddings

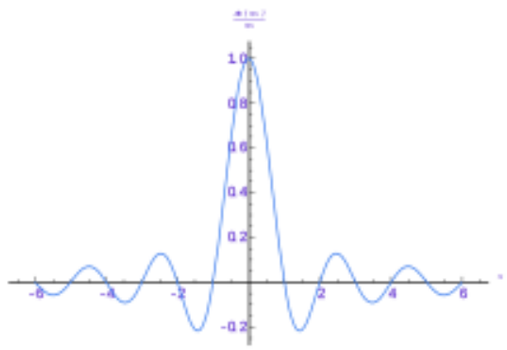
The Fourier transform

- 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

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

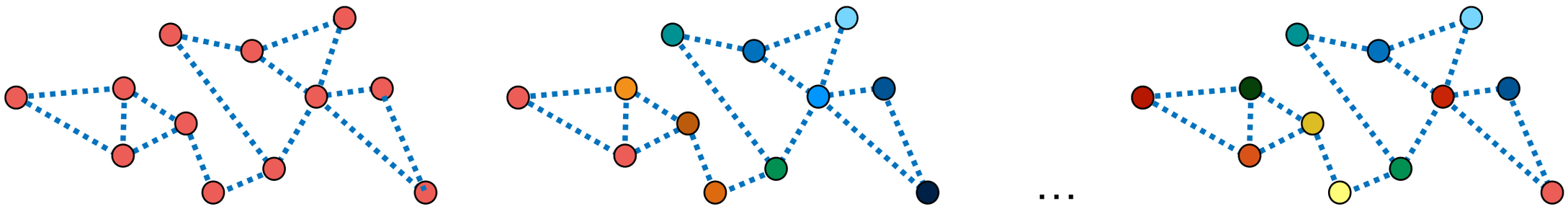
Graph FT: $\hat{f}(\ell) = \langle \chi_\ell, f \rangle = \sum_{i=1}^{N} \chi_\ell^*(i) f(i)$

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

FT: Fourier Transform

Graph Fourier transform

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



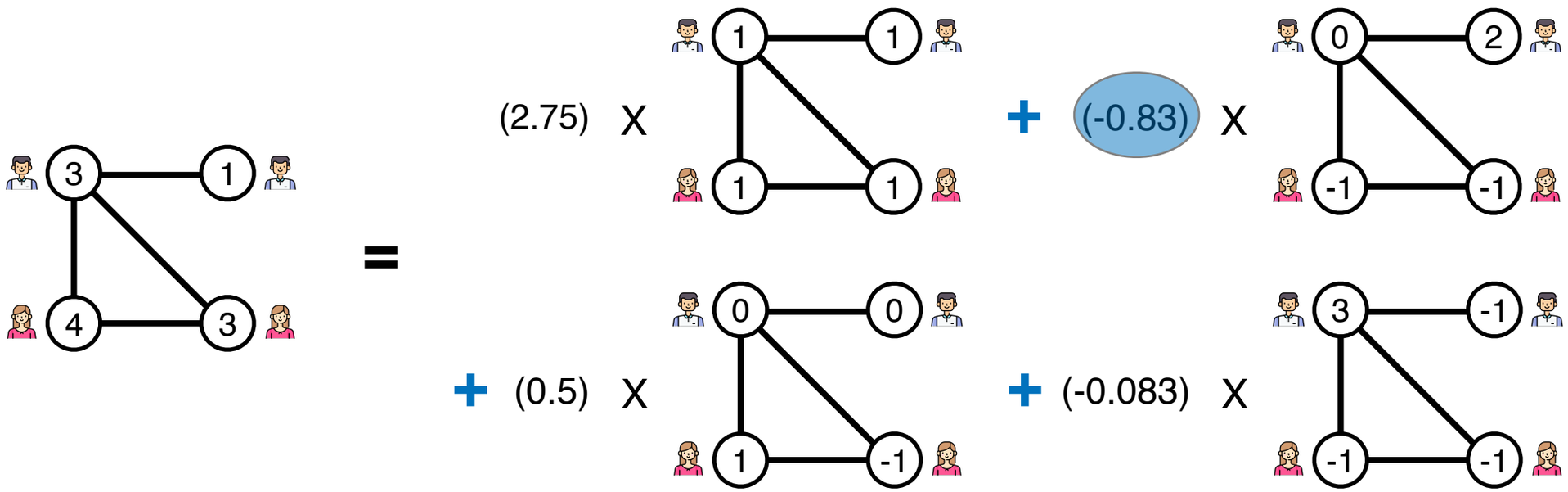
$\chi_1^T L \chi_1 = \lambda_1$ $\chi_2^T L \chi_2 = \lambda_2$ $\chi_N^T L \chi_N = \lambda_N$

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

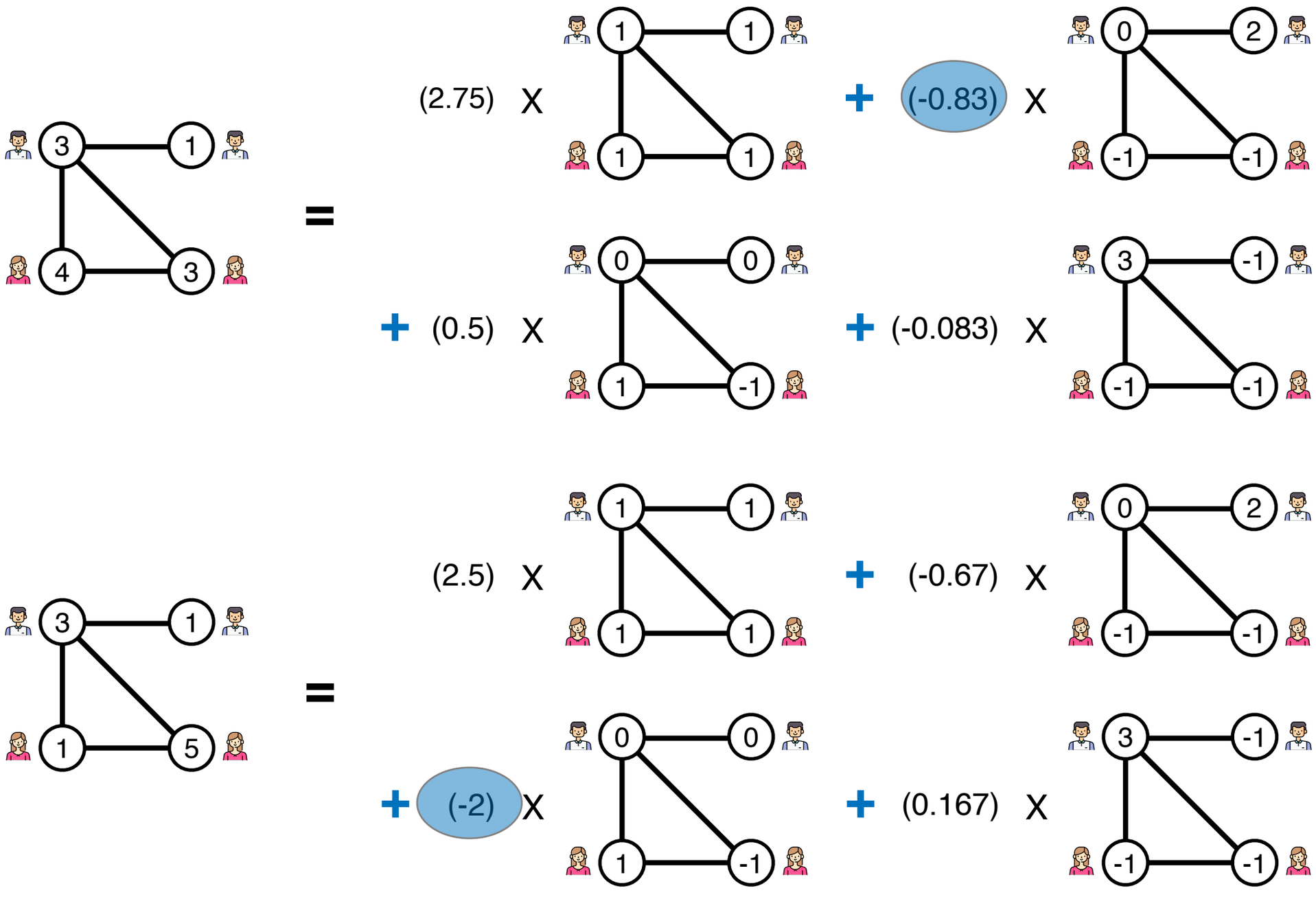
- By exploiting the orthonormality of the eigenvectors, we obtain:

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

Example on movie rating



Example on movie rating



Towards a convolution on graphs: A spectral viewpoint

- **Key intuition:** Convolution in the vertex domain is equivalent to multiplication in the spectral domain
- We define convolution on graphs starting from the multiplication in the GFT domain

$$L = \chi \Lambda \chi^T$$

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

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

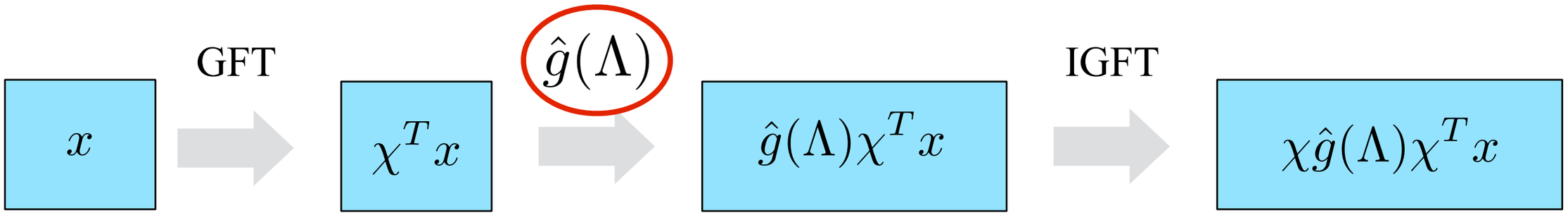
$$\widehat{(x * g)}(\lambda) = \big((\chi^T x) \circ \hat{g}\big)(\lambda)$$

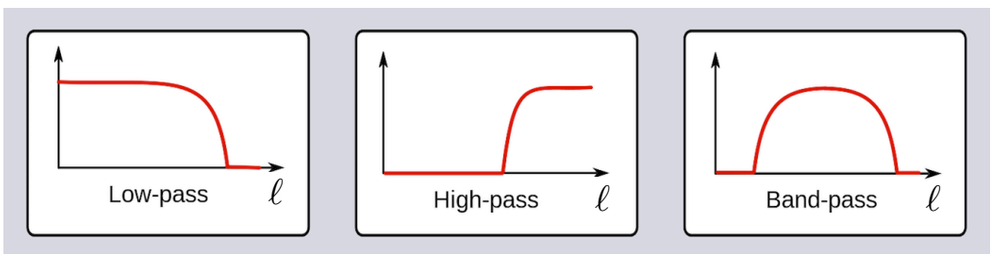
GFT

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

Is the graph convolution localized?

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

- In general the answer is no!
- However, if we consider polynomial filters, the answer is yes

Is the graph convolution localized?

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

- In general the answer is no!
- However, if we consider polynomial filters, the answer is yes

Example: $\hat{g}_\theta(\lambda) = \sum_{j=0}^{K} \theta_j \lambda^j, \ \theta \in \mathbb{R}^{K+1}$ ⇒ $\hat{g}_\theta(L) = \sum_{j=0}^{K} \theta_j L^j$

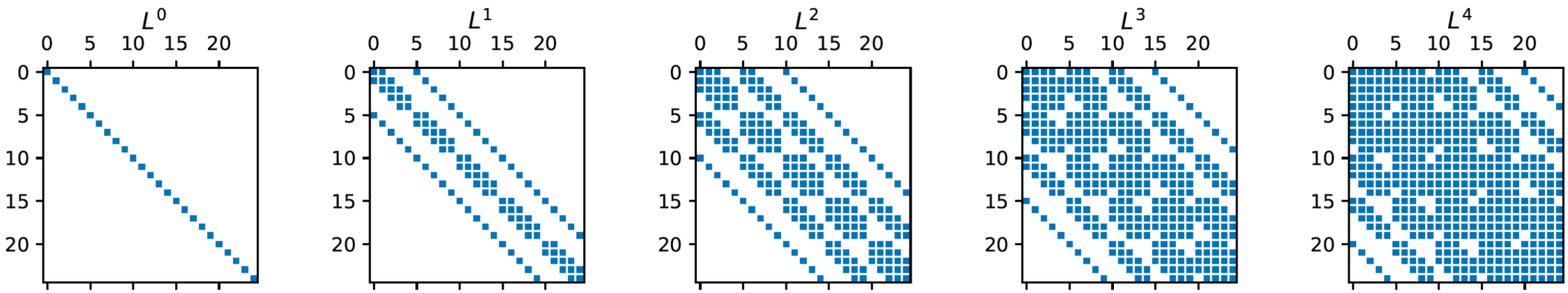
Is the graph convolution localized?

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

- In general the answer is no!
- However, if we consider polynomial filters, the answer is yes

Example: $\hat{g}_\theta(\lambda) = \sum_{j=0}^{K} \theta_j \lambda^j, \ \theta \in \mathbb{R}^{K+1}$ ⇒ $\hat{g}_\theta(L) = \sum_{j=0}^{K} \theta_j L^j$

- L^K defines the K-hop neighborhood: $d_G(v_i, v_j) > K \rightarrow (L^K)_{ij} = 0$



Is the graph convolution localized?

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

- In general the answer is no!
- However, if we consider polynomial filters, the answer is yes

Example: $\hat{g}_\theta(\lambda) = \sum_{j=0}^{K} \theta_j \lambda^j, \ \theta \in \mathbb{R}^{K+1}$ ⇒ $\hat{g}_\theta(L) = \sum_{j=0}^{K} \theta_j L^j$

- L^K defines the K-hop neighborhood: $d_G(v_i, v_j) > K \rightarrow (L^K)_{ij} = 0$

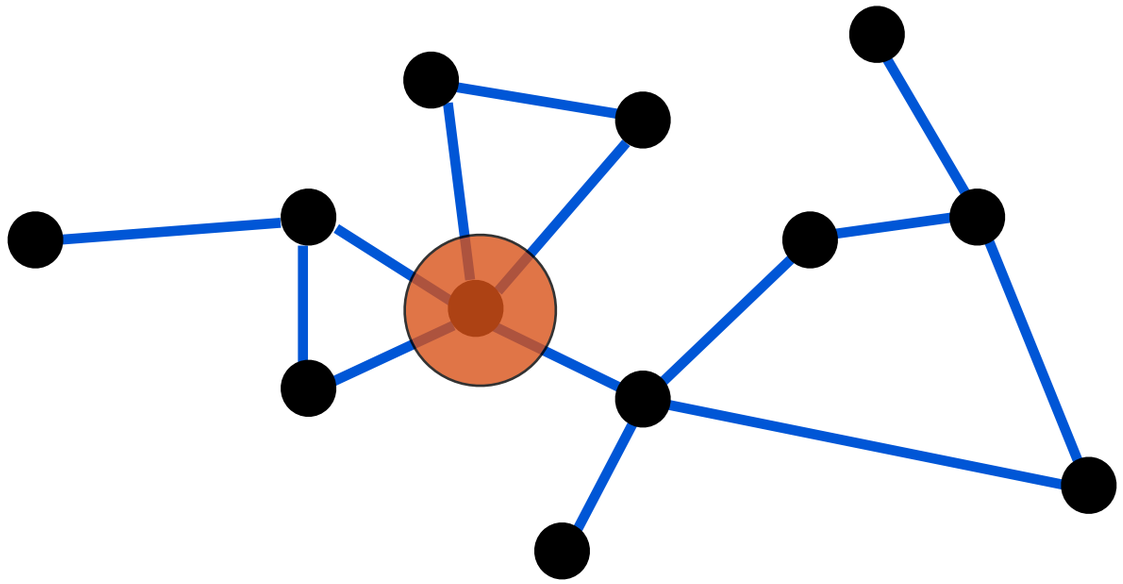
Is the graph convolution localized?

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

- In general the answer is no!
- However, if we consider polynomial filters, the answer is yes

Example: $\hat{g}_\theta(\lambda) = \sum_{j=0}^{K} \theta_j \lambda^j, \ \theta \in \mathbb{R}^{K+1}$ ⇒ $\hat{g}_\theta(L) = \sum_{j=0}^{K} \theta_j L^j$

- L^K defines the K-hop neighborhood: $d_G(v_i, v_j) > K \rightarrow (L^K)_{ij} = 0$



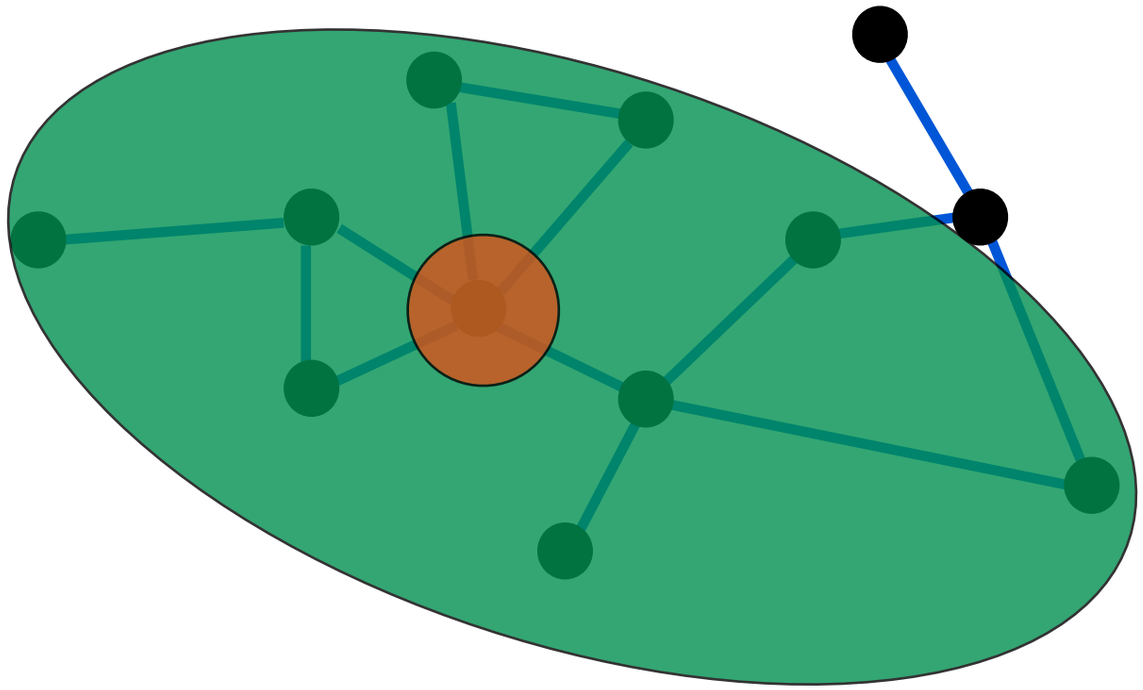
Is the graph convolution localized?

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

- In general the answer is no!
- However, if we consider polynomial filters, the answer is yes

Example: $\hat{g}_\theta(\lambda) = \sum_{j=0}^{K} \theta_j \lambda^j, \ \theta \in \mathbb{R}^{K+1}$ ⇒ $\hat{g}_\theta(L) = \sum_{j=0}^{K} \theta_j L^j$

- L^K defines the K-hop neighborhood: $d_G(v_i, v_j) > K \rightarrow (L^K)_{ij} = 0$



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:

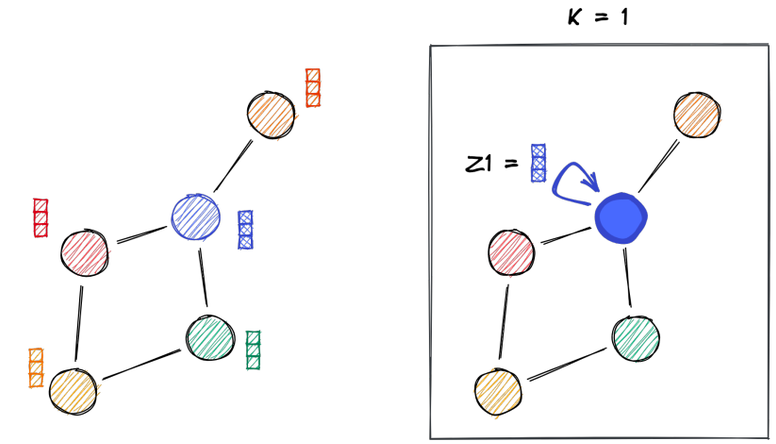
$$z_0 = x$$

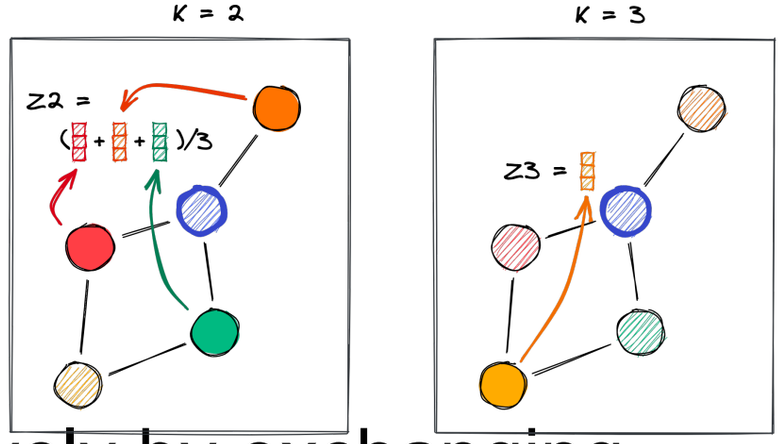
$$z_1 = Lz_0$$

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

$$\vdots$$

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

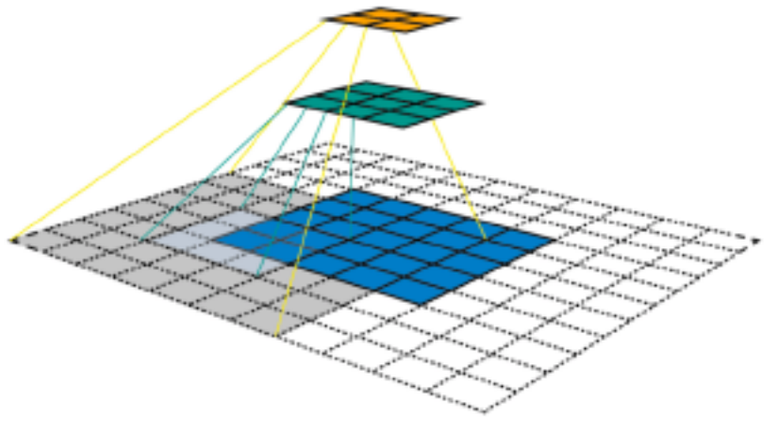




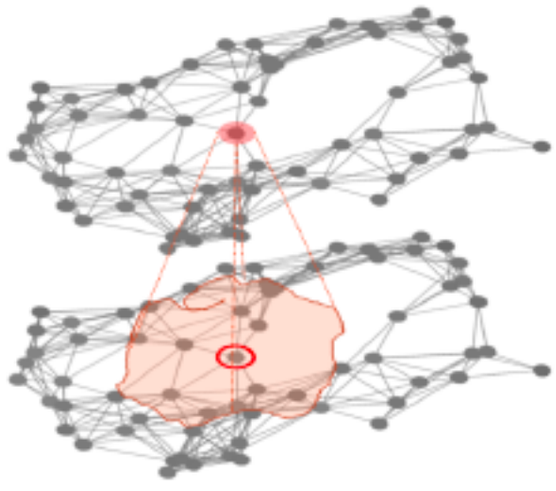
- Graph convolution can be computed recursively by exchanging information in a local neighborhood (i.e., message passing)
- The kernel $\hat{g}(\cdot)$ does not depend on the order of the nodes: permutation invariant!

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

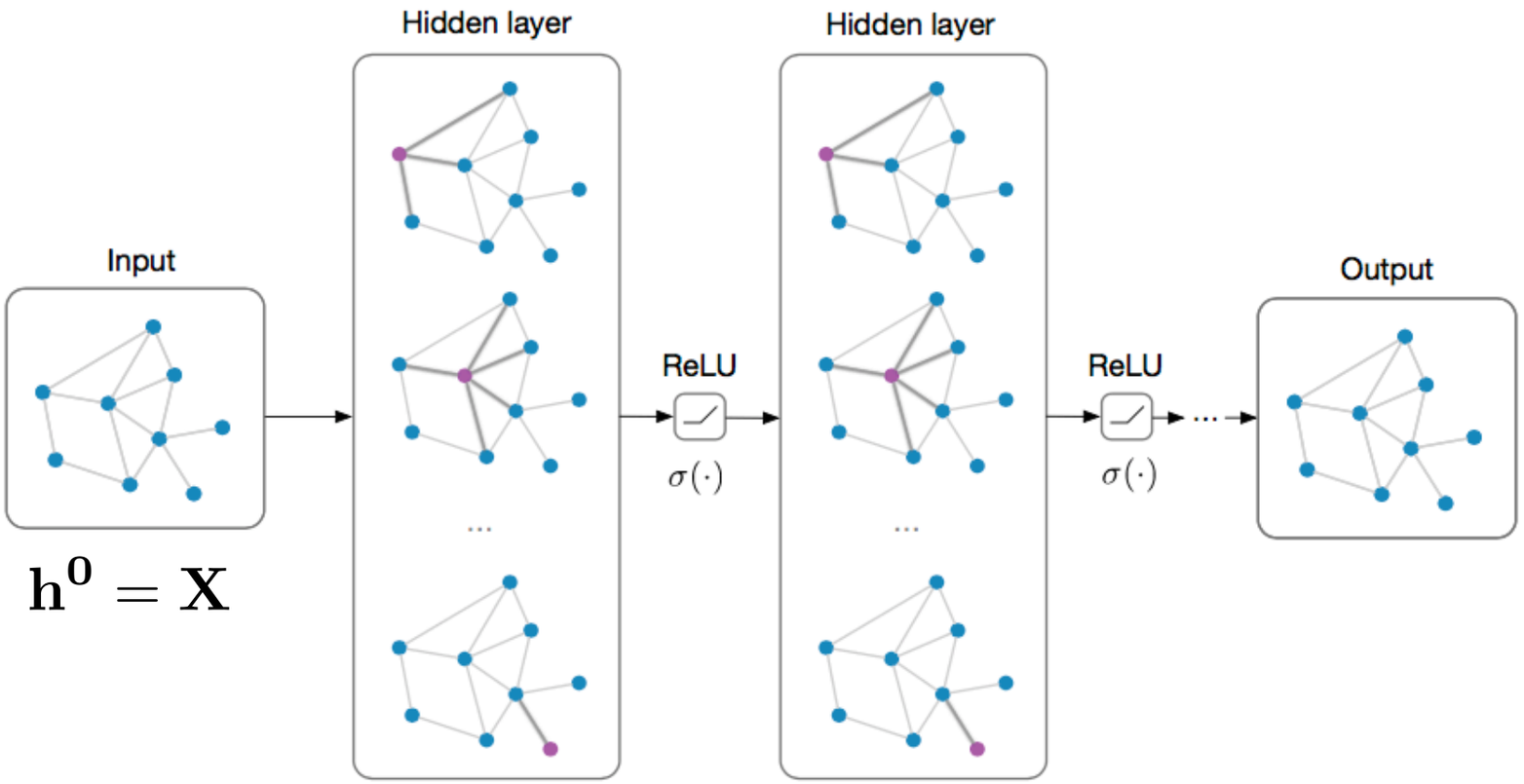
- A convolutional layer is defined as:

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

Learned parameters

GCN architecture

- Very often, it consists of two GCN layers



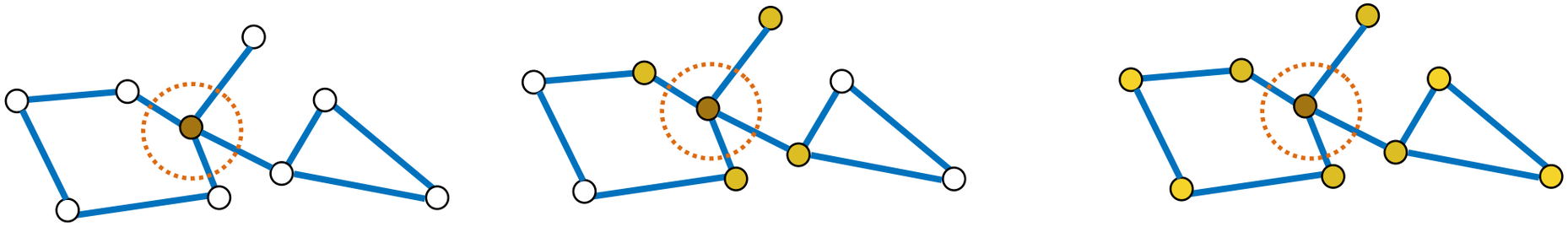
$$\mathbf{h^{l+1}} = \sigma(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{W}}\tilde{\mathbf{D}}^{-1/2}\mathbf{h^l}\theta^{l+1})$$

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

Each layer increases the receptive field of each node

- Each layer increases the receptive field by K hops

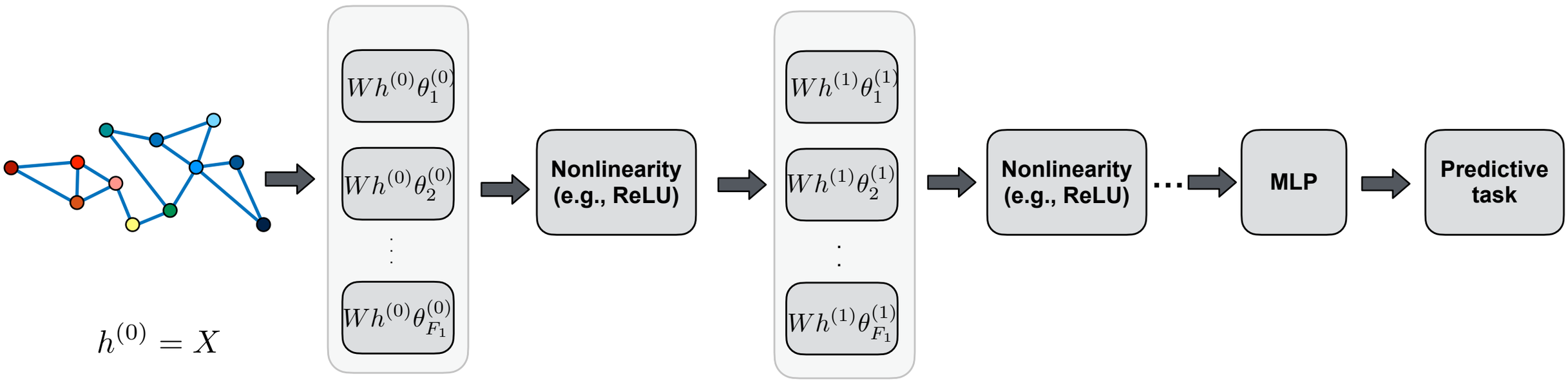
- Example: $K = 1$



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

The basic GNN: a spatial viewpoint

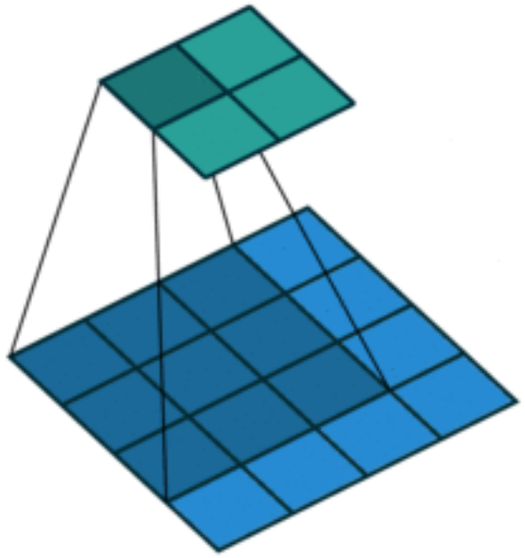
- Consists of a set of graph convolutional layers, each of which is followed by elementwise nonlinearity, i.e., $h^{(l+1)} = \sigma(z^{(l)})$



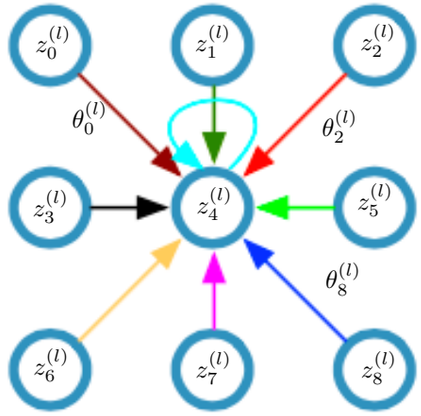
- Each layer increases the receptive field by 1-hop neighbors

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

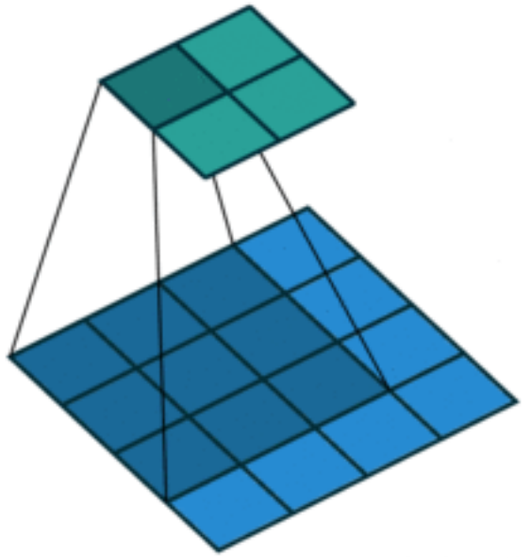


$$z_i^{(l+1)} = \sum_{i=0}^{8} \theta_i^{(l)} z_i^{(l)}$$

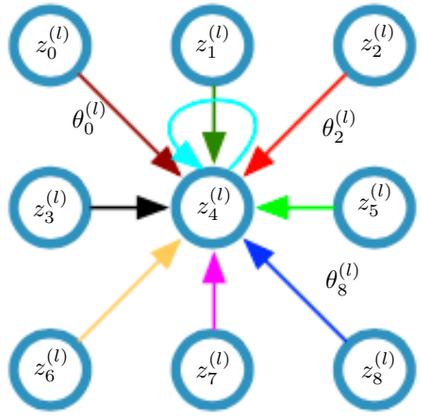
Can we exploit similar structure for graph data?

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



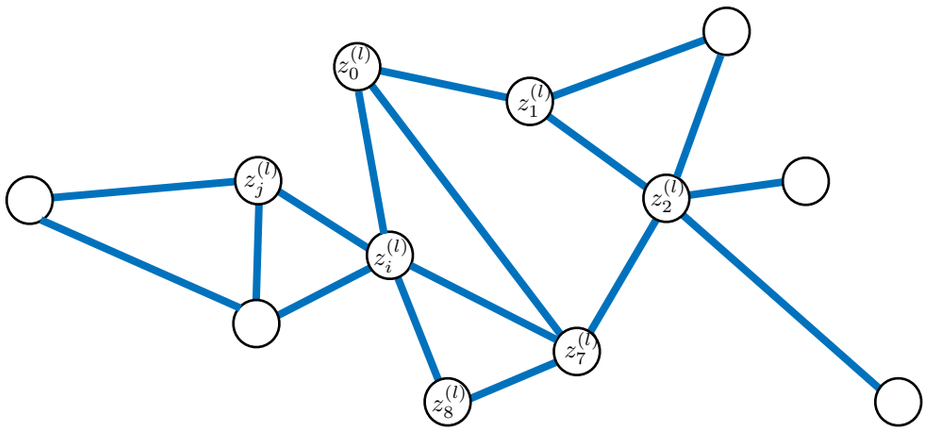
$$z_i^{(l+1)} = \sum_{i=0}^{8} \theta_i^{(l)} z_i^{(l)}$$

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

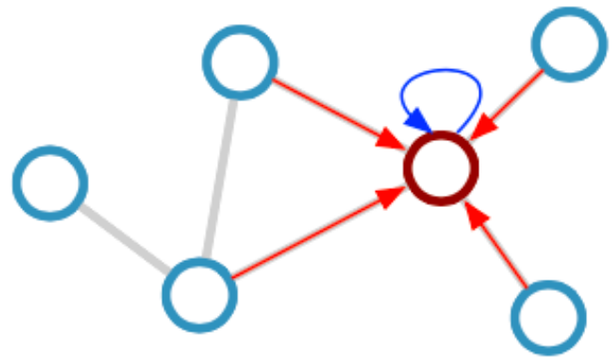
Spatial approaches in one slide

- Generate node embeddings based on local neighborhoods
  - Nodes aggregate information from their neighbors using a permutation-invariant function
  - The feature vector of each node is updated based on its current values and the aggregated neighbourhood representation

- Feed the embeddings into a loss function (usually task specific)

- **Key difference between architectures:** How nodes aggregate information across layers

$$h_i^{l+1} = \sigma(U^l h_i^l + M^l \sum_{j \in \mathcal{N}_i} h_j^l)$$



Spatial approaches in one slide

- Generate node embeddings based on local neighborhoods
  - Nodes aggregate information from their neighbors using a permutation-invariant function
  - The feature vector of each node is updated based on its current values and the aggregated neighbourhood representation

- Feed the embeddings into a loss function (usually task specific)

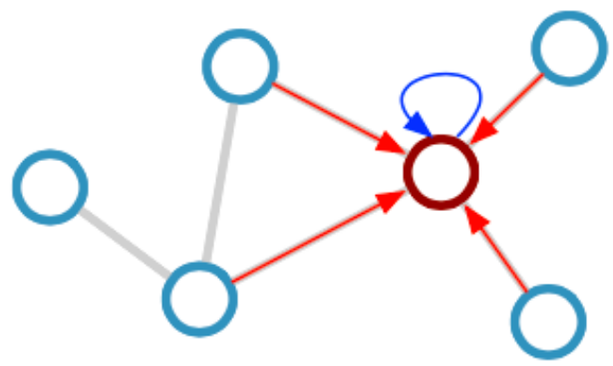
- **Key difference between architectures:** How nodes aggregate information across layers

self information

neighbour information

$$h_i^{l+1} = \sigma(U^l h_i^l + M^l \sum_{j \in \mathcal{N}_i} h_j^l)$$

learnable parameters



Spatial approaches in one slide

- Generate node embeddings based on local neighborhoods
  - Nodes aggregate information from their neighbors using a permutation-invariant function
  - The feature vector of each node is updated based on its current values and the aggregated neighbourhood representation

- Feed the embeddings into a loss function (usually task specific)

- **Key difference between architectures:** How nodes aggregate information across layers

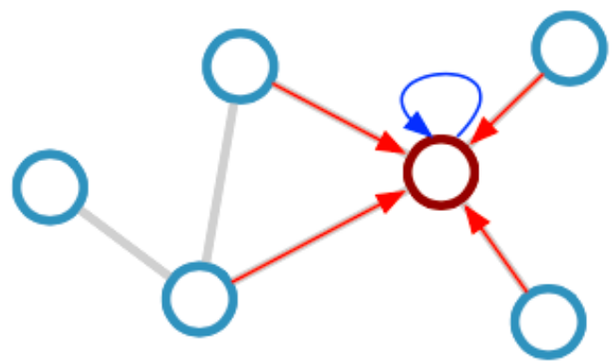
self information

neighbour information

$$h_i^{l+1} = \sigma(U^l h_i^l + M^l \sum_{j \in \mathcal{N}_i} h_j^l)$$

learnable parameters

permutation invariant local neighbourhood



MPNN - Example

update function message function

$$h_i^{l+1} = U_l\Big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\Big)$$

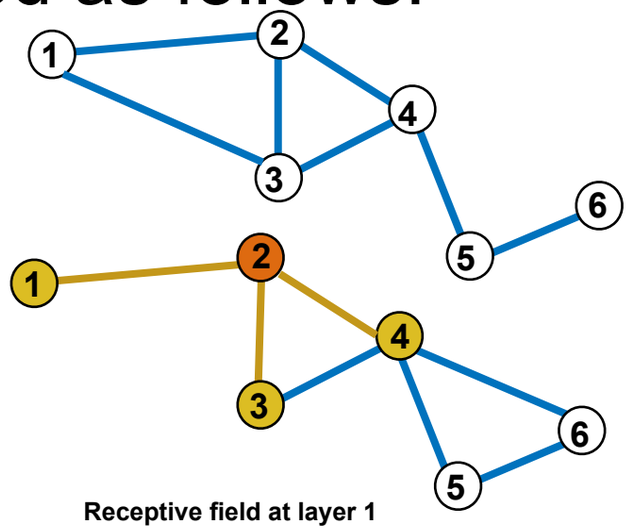
aggregator function

$$\Rightarrow \quad h_i^{l+1} = \theta_0^l h_i^l + \theta_1^l \sum_{j \in \mathcal{N}_i} h_j^l$$

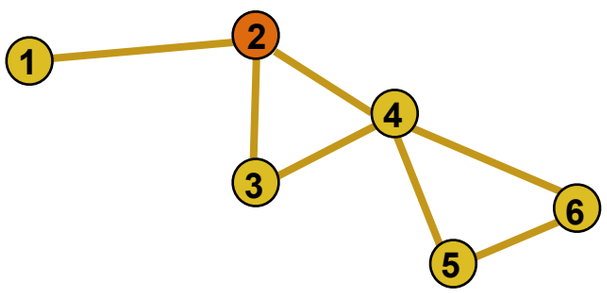
- At each iteration, the embeddings are updated as follows:

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

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



Receptive field at layer 1



Receptive field at layer 2

Graph isomorphism network

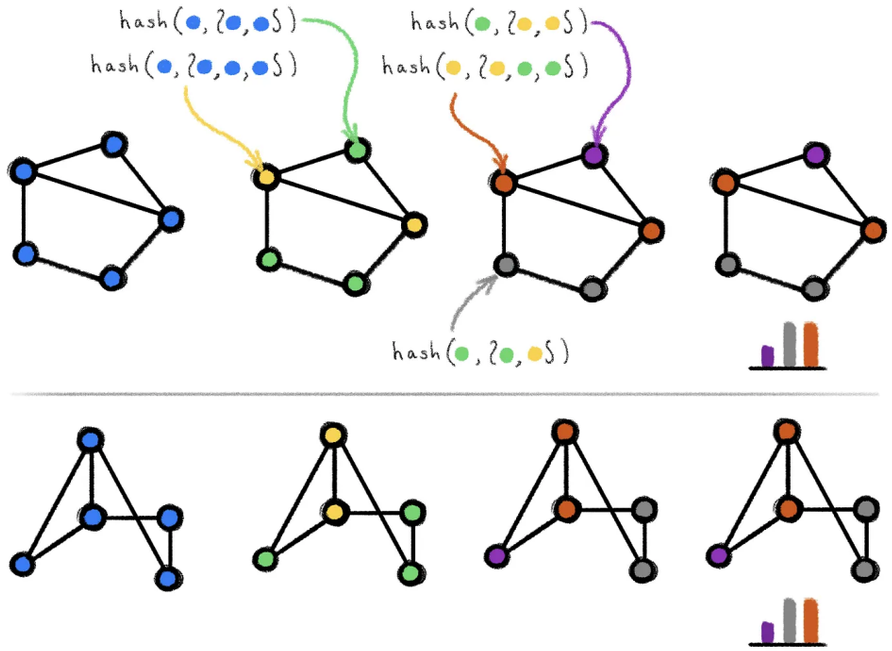
- Motivation: introduce a way of defining **expressive power** of MPNNs
  - for example, for graph classification, what kinds of graphs can they distinguish?
- Inspiration from the Weisfeiler-Lehman test for graph isomorphism (whether two graphs are isomorphic)

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as

$$c_i^{l+1} = \text{HASH}\left(\left\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\right\}\right)$$

- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$



potentially isomorphic graphs (necessary but not sufficient)

[source: M. Bronstein]

Xu et al., "How powerful are graph neural networks?," ICLR, 2019.

Graph isomorphism network

- Striking similarity between MPNNs and 1-WL test
  - both follow three steps: 1) message passing, 2) neighbourhood aggregation, 3) node update

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as

$$c_i^{l+1} = \text{HASH}\left(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\}\right)$$

- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

MPNN

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial embedding h_i^0 (e.g., node attribute) to each node i of $\mathcal{V}$
- For each layer $l+1$ refine node embeddings as

$$h_i^{l+1} = U_l\left(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_j^l, h_i^l)\right)$$

- Until maximum number of layers is reached
- **Output:** The node embeddings $\{h_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

Graph isomorphism network

- Striking similarity between MPNNs and 1-WL test
  - both follow three steps: 1) message passing, 2) neighbourhood aggregation, 3) node update

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as

$$c_i^{l+1} = \text{HASH}\big(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\}\big)$$

- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

MPNN

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial embedding h_i^0 (e.g., node attribute) to each node i of $\mathcal{V}$
- For each layer $l+1$ refine node embeddings as

$$h_i^{l+1} = U_l\big(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_j^l, h_i^l)\big)$$

- Until maximum number of layers is reached
- **Output:** The node embeddings $\{h_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

Graph isomorphism network

- Striking similarity between MPNNs and 1-WL test
  - both follow three steps: 1) message passing, 2) neighbourhood aggregation, 3) node update

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as

$$c_i^{l+1} = \text{HASH}(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\})$$

- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

MPNN

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial embedding h_i^0 (e.g., node attribute) to each node i of $\mathcal{V}$
- For each layer $l+1$ refine node embeddings as

$$h_i^{l+1} = U_l\left(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_j^l, h_i^l)\right)$$

- Until maximum number of layers is reached
- **Output:** The node embeddings $\{h_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

Graph isomorphism network

- Striking similarity between MPNNs and 1-WL test
  - both follow three steps: 1) message passing, 2) neighbourhood aggregation, 3) node update

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as

$$c_i^{l+1} = \text{HASH}(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\})$$

- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

MPNN

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial embedding h_i^0 (e.g., node attribute) to each node i of $\mathcal{V}$
- For each layer $l+1$ refine node embeddings as

$$h_i^{l+1} = U_l\left(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_j^l, h_i^l)\right)$$

- Until maximum number of layers is reached
- **Output:** The node embeddings $\{h_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

Graph isomorphism network

- Striking similarity between MPNNs and 1-WL test
  - both follow three steps: 1) message passing, 2) neighbourhood aggregation, 3) node update
- Can we design an MPNN that is as powerful as the WL test?
  - more than graph isomorphism we can use it for graph classification or regression
  - hash func in WL is injective $\Rightarrow$ injective aggregator/update func in MPNNs?

1-WL test

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial color c_i^0 (e.g., node degree) to each node i of $\mathcal{V}$
- For each iteration $l+1$ refine node colors as

$$c_i^{l+1} = \mathrm{HASH}(\{c_i^l, \{c_j^l\}_{j \in \mathcal{N}_i}\})$$

- Until stable node coloring is reached
- **Output:** The node colors $\{c_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

MPNN

- **Input:** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$
- Assign an initial embedding h_i^0 (e.g., node attribute) to each node i of $\mathcal{V}$
- For each layer $l+1$ refine node embeddings as

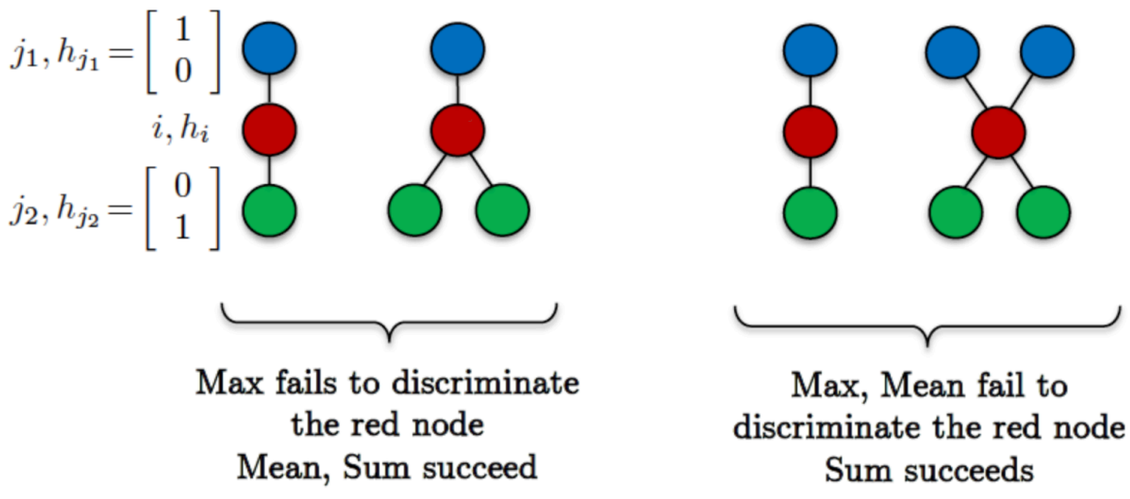
$$h_i^{l+1} = U_l\left(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_j^l, h_i^l)\right)$$

- Until maximum number of layers is reached
- **Output:** The node embeddings $\{h_i^{l_{max}}\}_{i=\{1,2,\ldots,N\}}$

Graph isomorphism network

- What aggregator function is injective?
  - expressive power on a multiset: sum > mean > max-pooling
- GIN is provably as powerful as 1-WL test (under certain conditions)
  - aggregator & update: sum + MLP
  - global readout: sum + MLP

$$h_i^{l+1} = U_l\left(h_i^l, \bigoplus_{j \in \mathcal{N}_i} M_l(h_i^l, h_j^l)\right)$$



[source: X. Bresson]

sum aggregator

$$h_i^{l+1} = \mathrm{MLP}^l\left((1+\epsilon^l)h_i^l + \sum_{j \in \mathcal{N}_i} h_j^l\right)$$

learnable parameters

$$h_{\mathcal{G}} = \mathrm{MLP}\left(\sum_{i \in \mathcal{V}} h_i^L\right)$$

sum aggregator

Comparison between spatial and spectral design

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

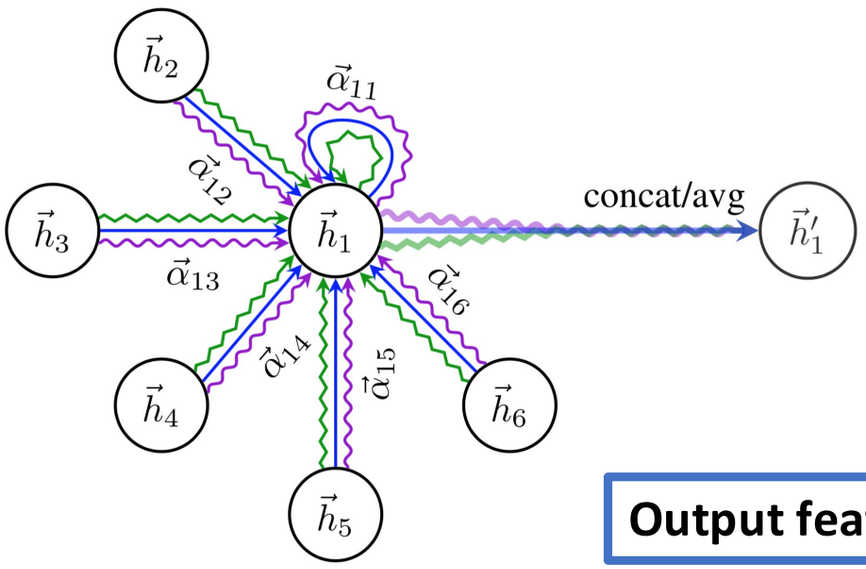
- Spatial convolution: Generalizes the notion of convolution by following a spatial viewpoint

- 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

Graph attention networks (GAT)

- Intuition: learn relative importance of neighbors in aggregation

- In attentional GNNs, weights depend on the neighborhood's features
  - Different weights to different nodes in a neighbourhood
  - Remove dependence on the global graph structure



Output features

Updated embedding **relative importance**

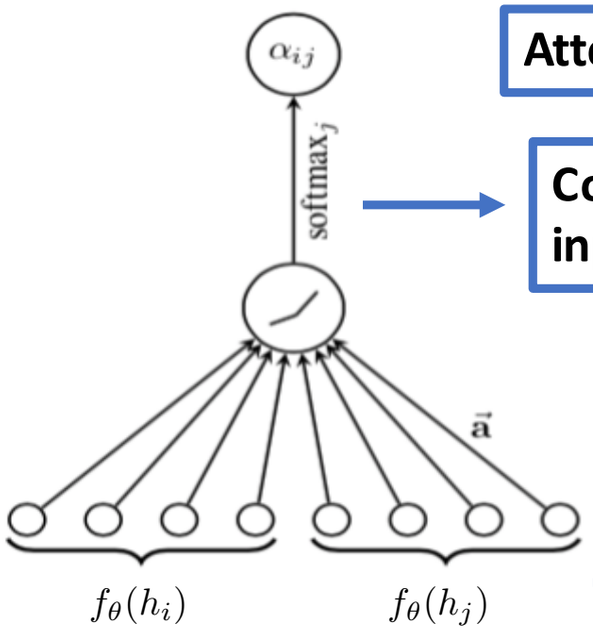
$$h_i^{l+1} = \sigma(\alpha_{ii}^l \theta_0^l h_i^l + \sum_{j \in \mathcal{N}_i} \alpha_{ij}^l \theta_1^l h_j^l)$$

[Velickovic et al., Graph Attention Networks, ICLR 2018]

Graph attention networks (GAT)

- Intuition: learn relative importance of neighbors in aggregation

- In attentional GNNs, weights depend on the neighborhood's features
  - Different weights to different nodes in a neighbourhood
  - Remove dependence on the global graph structure



Attention weights

Comparison of input features

Input features of nodes i, j

Updated embedding — relative importance

$$h_i^{l+1} = \sigma(\alpha_{ii}^l \theta_0^l h_i^l + \sum_{j \in \mathcal{N}_i} \alpha_{ij}^l \theta_1^l h_j^l)$$

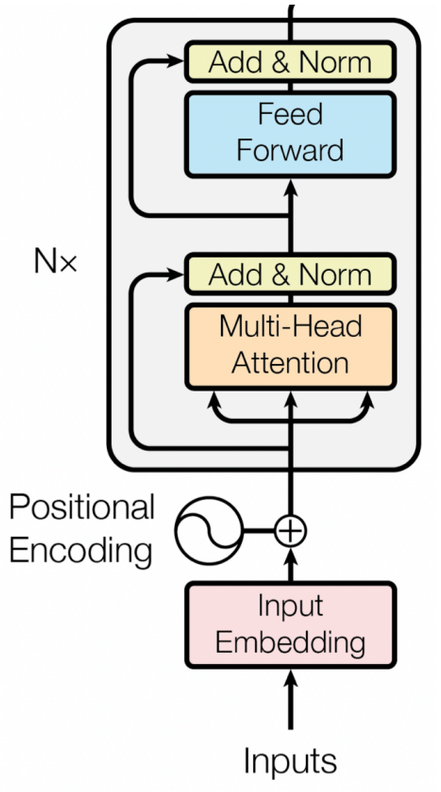
$$\alpha_{ij}^{l+1} = \alpha\big(f_\theta(h_i^{l+1}), f_\theta(h_j^{l+1})\big)$$

Attention function

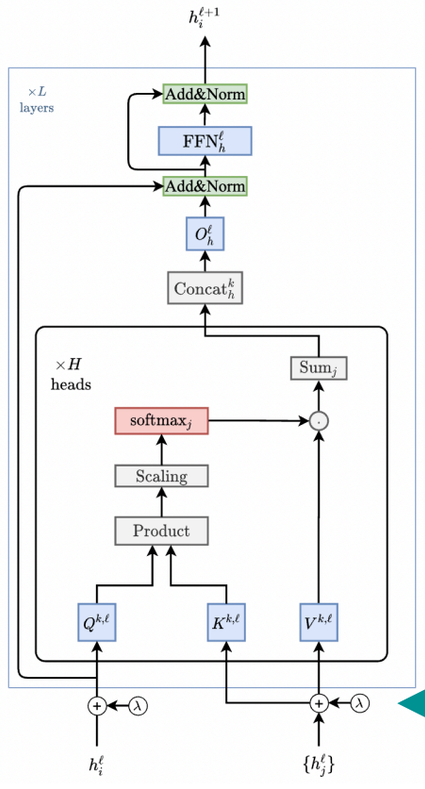
[Velickovic et al., Graph Attention Networks, ICLR 2018]

Graph transformers

- Generalise GAT to global attention
- Capture long-range dependencies (but computationally expensive)



transformers



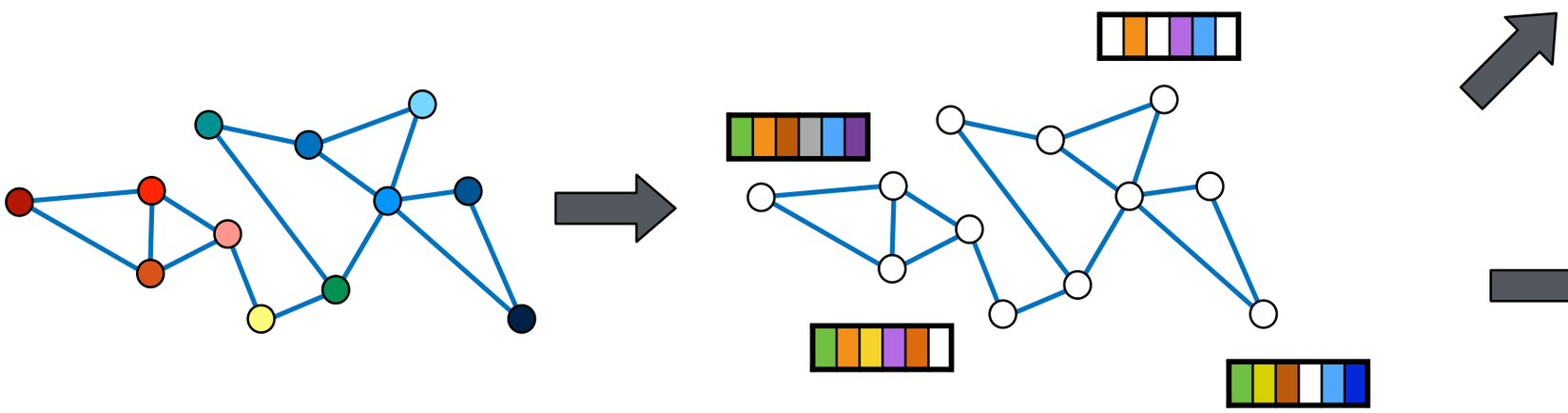
PEs retaining graph info

graph transformers

[Dwivedi and Bresson, “A generalization of transformer networks to graphs,” AAAI Workshop, 2021]

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

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

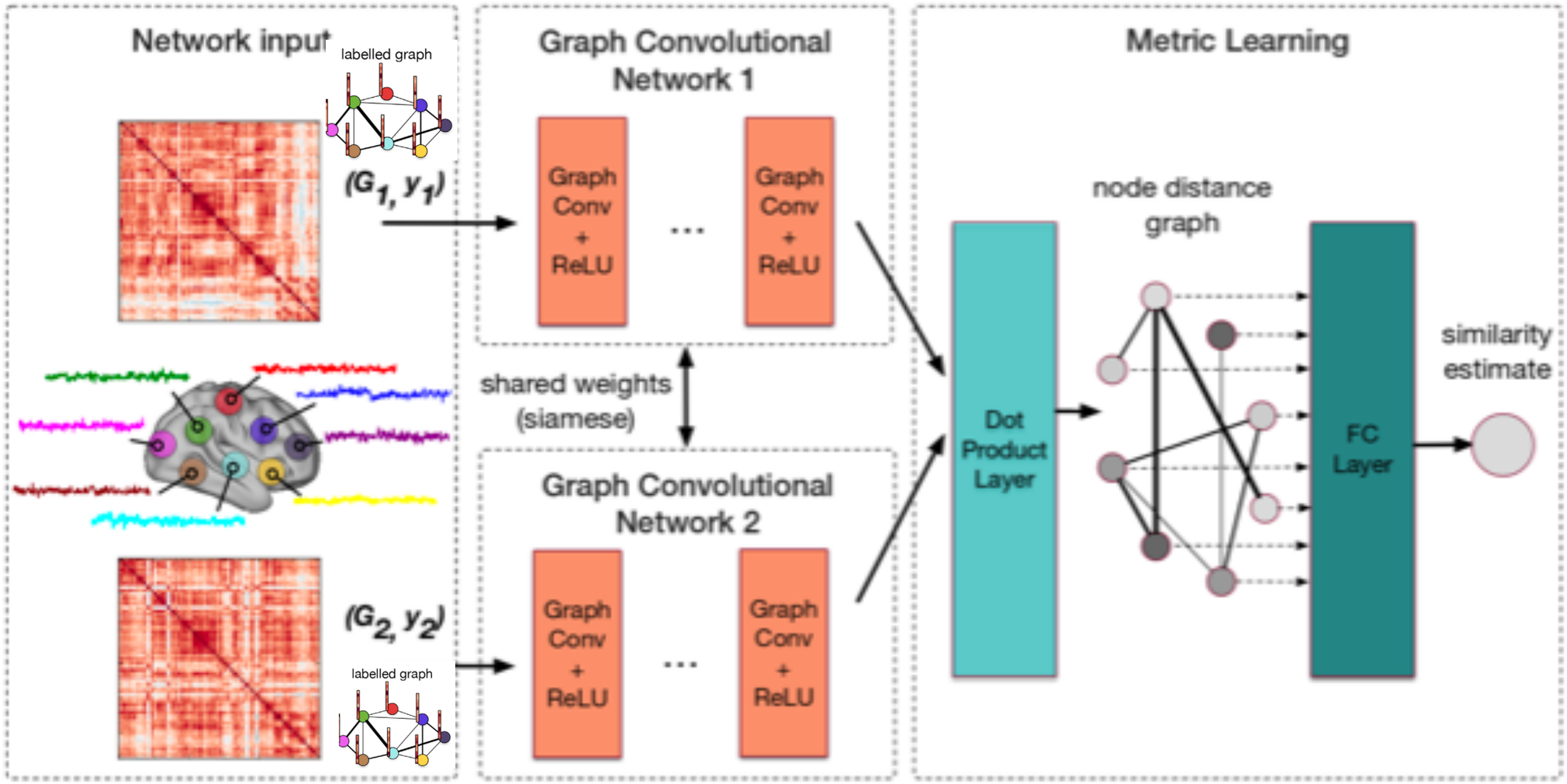
Graph classification

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

Link prediction

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

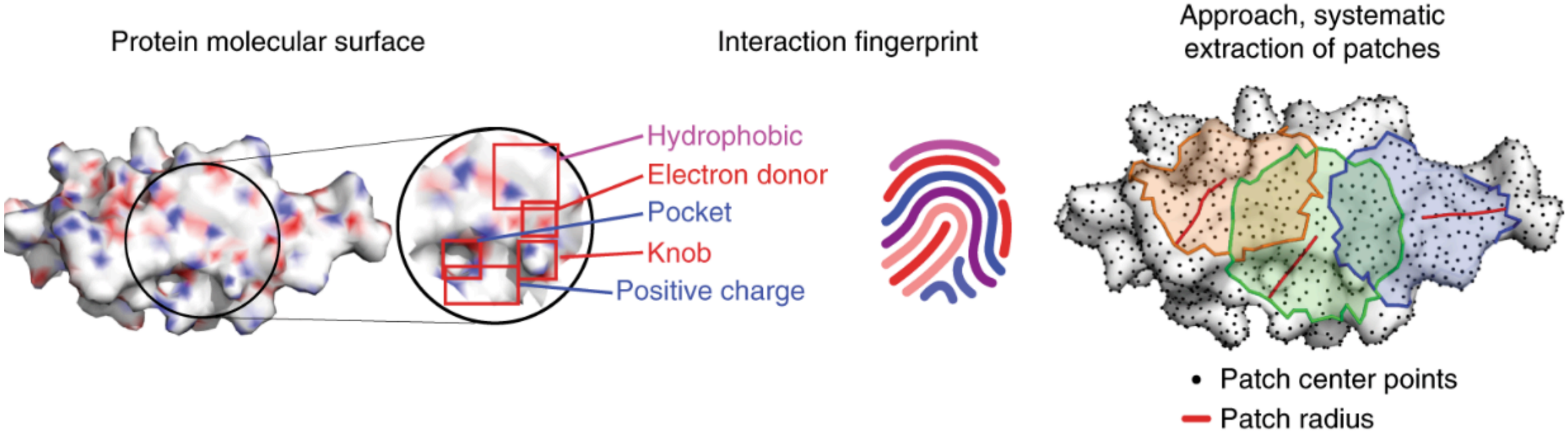
Neuroscience: learn to compare brain networks



[Ktena et al., Metric learning with spectral graph convolutions on brain connectivity networks, NeuroImage, 2018]

Protein-protein interactions

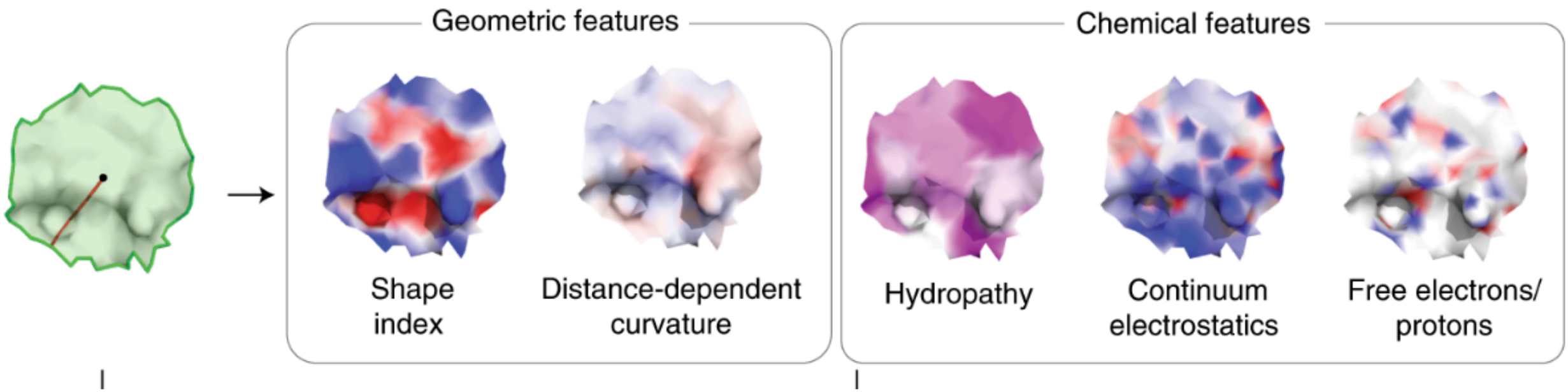
- Predicting interactions between proteins and other biomolecules solely based on structure remains a challenge in biology
- Exploit GNNs to learn interaction fingerprints in protein molecular surfaces that determine protein interactions



[Gainza et al, Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning, Nature methods, 2019]

Protein-protein interactions

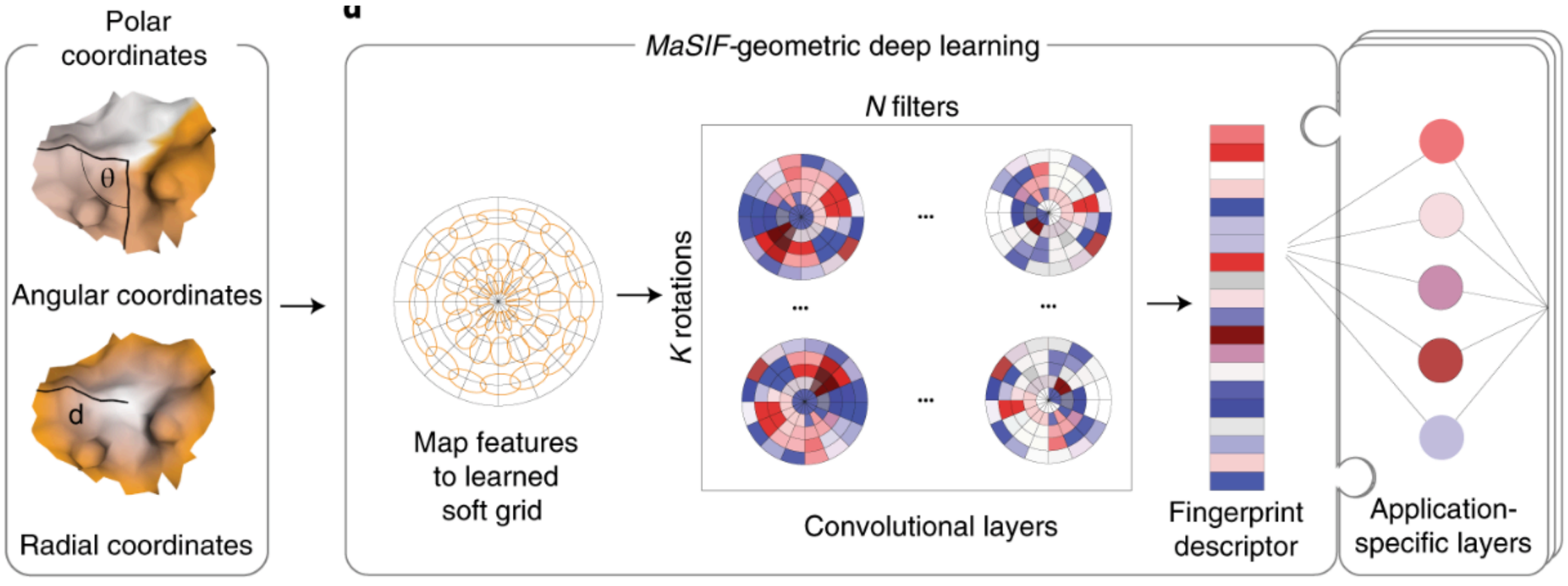
- Predicting interactions between proteins and other biomolecules solely based on structure remains a challenge in biology
- Exploit GNNs to learn interaction fingerprints in protein molecular surfaces that determine protein interactions



[Gainza et al, Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning, Nature methods, 2019]

Protein-protein interactions

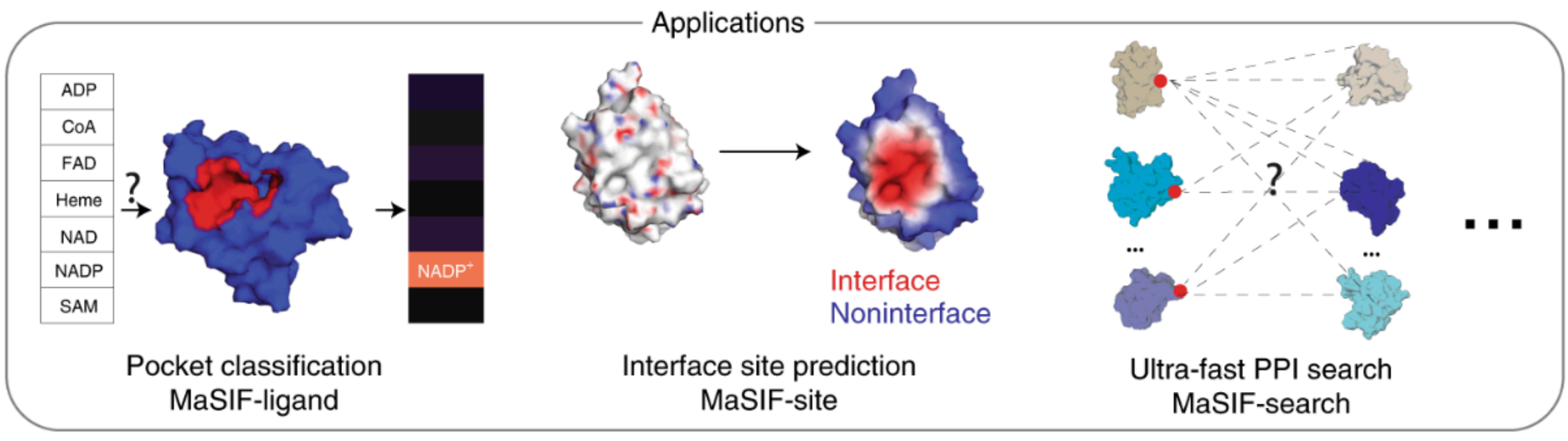
- Predicting interactions between proteins and other biomolecules solely based on structure remains a challenge in biology
- Exploit GNNs to learn interaction fingerprints in protein molecular surfaces that determine protein interactions



[Gainza et al, Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning, Nature methods, 2019]

Protein-protein interactions

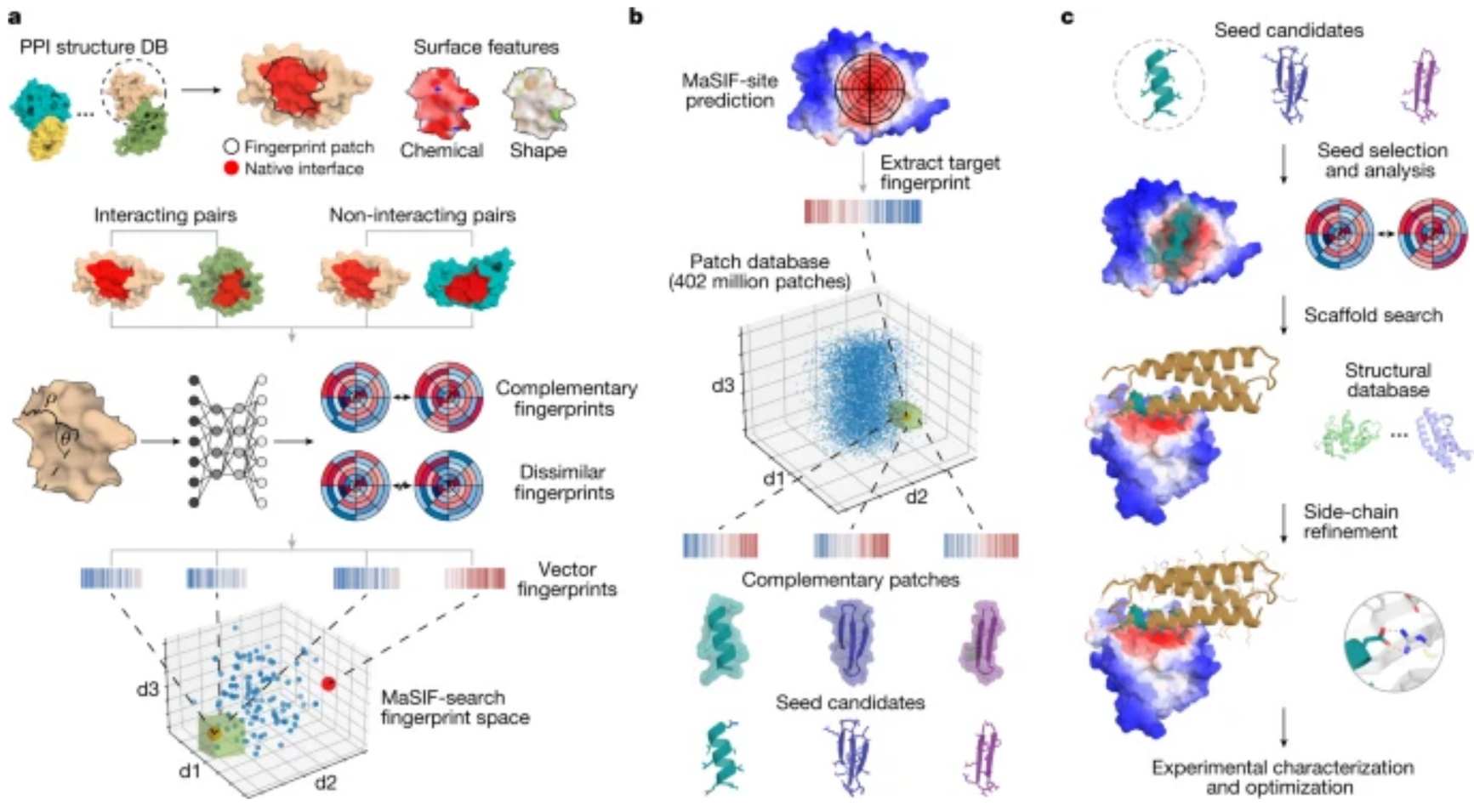
- Predicting interactions between proteins and other biomolecules solely based on structure remains a challenge in biology
- Exploit GNNs to learn interaction fingerprints in protein molecular surfaces that determine protein interactions



[Gainza et al, Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning, Nature methods, 2019]

De novo design of protein-protein interactions

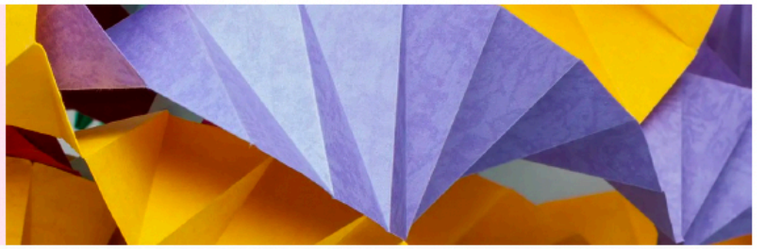
- Design PPIs by targeting sites using only structural information from the target protein



[Gainza et al, De novo design of protein interactions with learned surface fingerprints, Nature, 2023]

Protein folding

AlphaFold: a solution to a 50-year-old grand challenge in biology

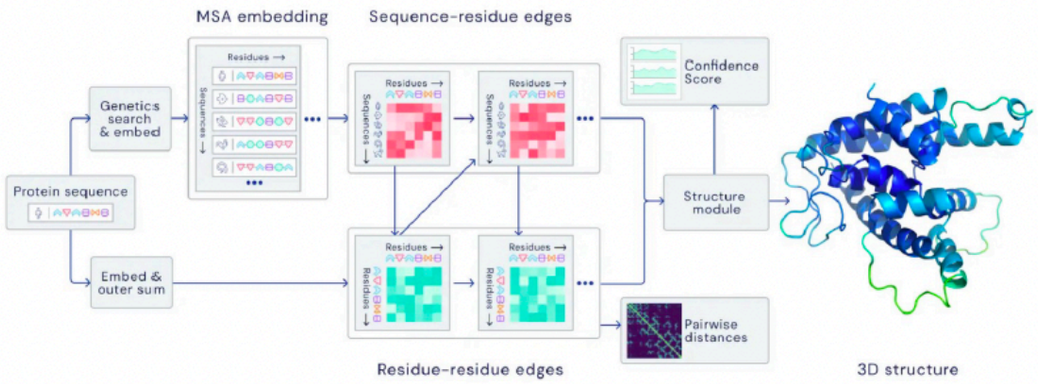


SHARE

AUTHORS

TAt The AlphaFold team

Proteins are essential to life, supporting practically all its functions. They are large complex molecules, made up of chains of amino acids, and what a protein does largely depends on its unique 3D structure. Figuring out what shapes proteins fold into is known as the "protein folding problem", and has stood as a grand challenge in biology for the past 50 years. In a major scientific advance, the latest version of our AI system AlphaFold has been recognised as a solution to this grand challenge by the organisers of the biennial Critical Assessment of protein Structure Prediction (CASP). This breakthrough demonstrates the impact AI can have on scientific discovery and its potential to dramatically accelerate progress in some of the most fundamental fields that explain and shape our world.

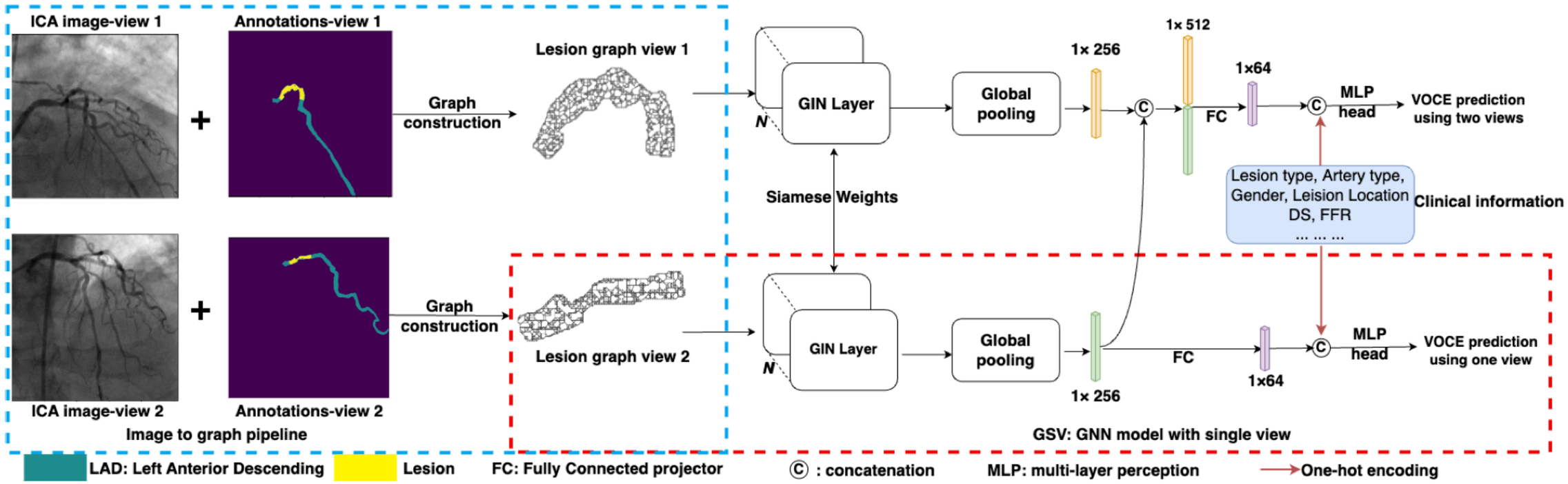


Graph Transformers

[Jumper et al., High protein structure prediction with AlphaFold, Nature, 2021]

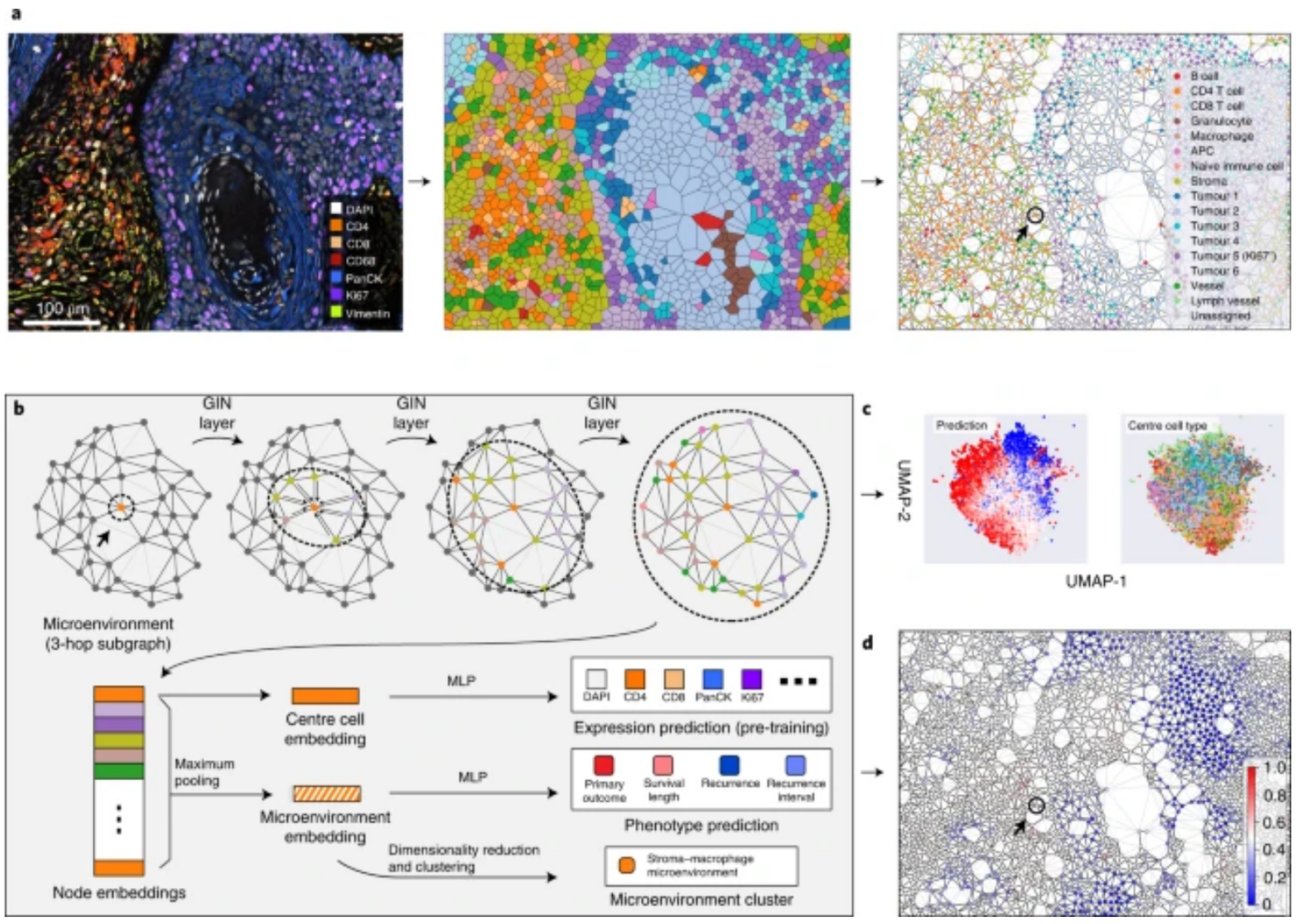
Future cardiovascular events prediction

- Graphs are used to model the geometry of the arteries
- Graph neural networks extract predictive features from invasive coronary artery images



[Sun et al., Graph neural network based future clinical events prediction from invasive coronary angiography, ISBI, 2024]

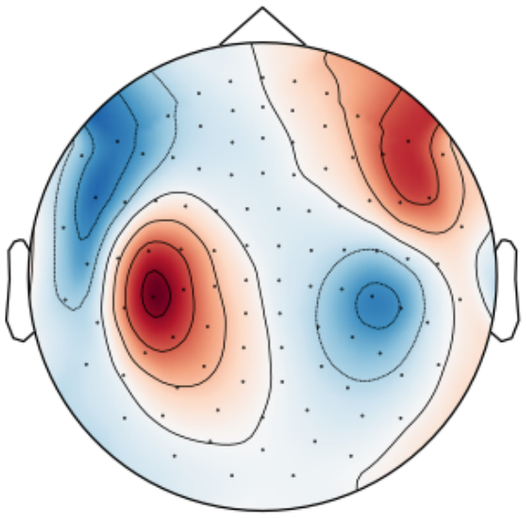
Geometric deep learning models cellular microenvironment



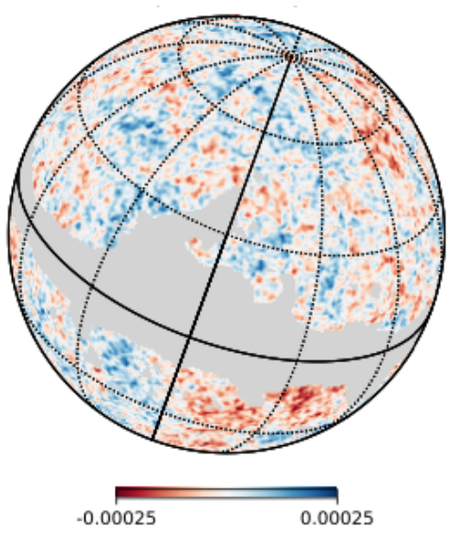
[Wu et al, Graph deep learning for the characterization of tumour microenvironments from spatial protein profiles in tissue specimens, Nature Biomedical Engineering, 2022]

Spherical imaging

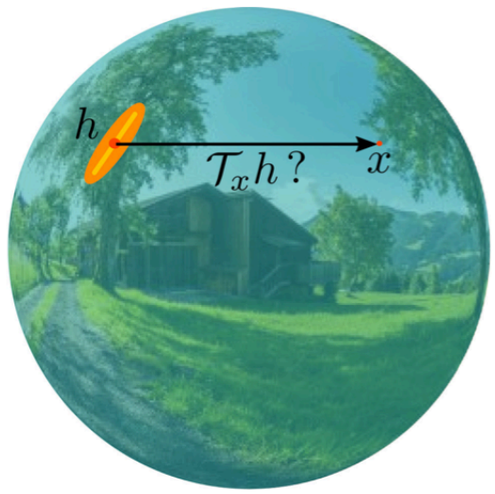
- Spherical data has specific spatial and statistical properties that cannot be captured by regular CNN models



Brain activity (MEG)



Cosmic microwave background temperature



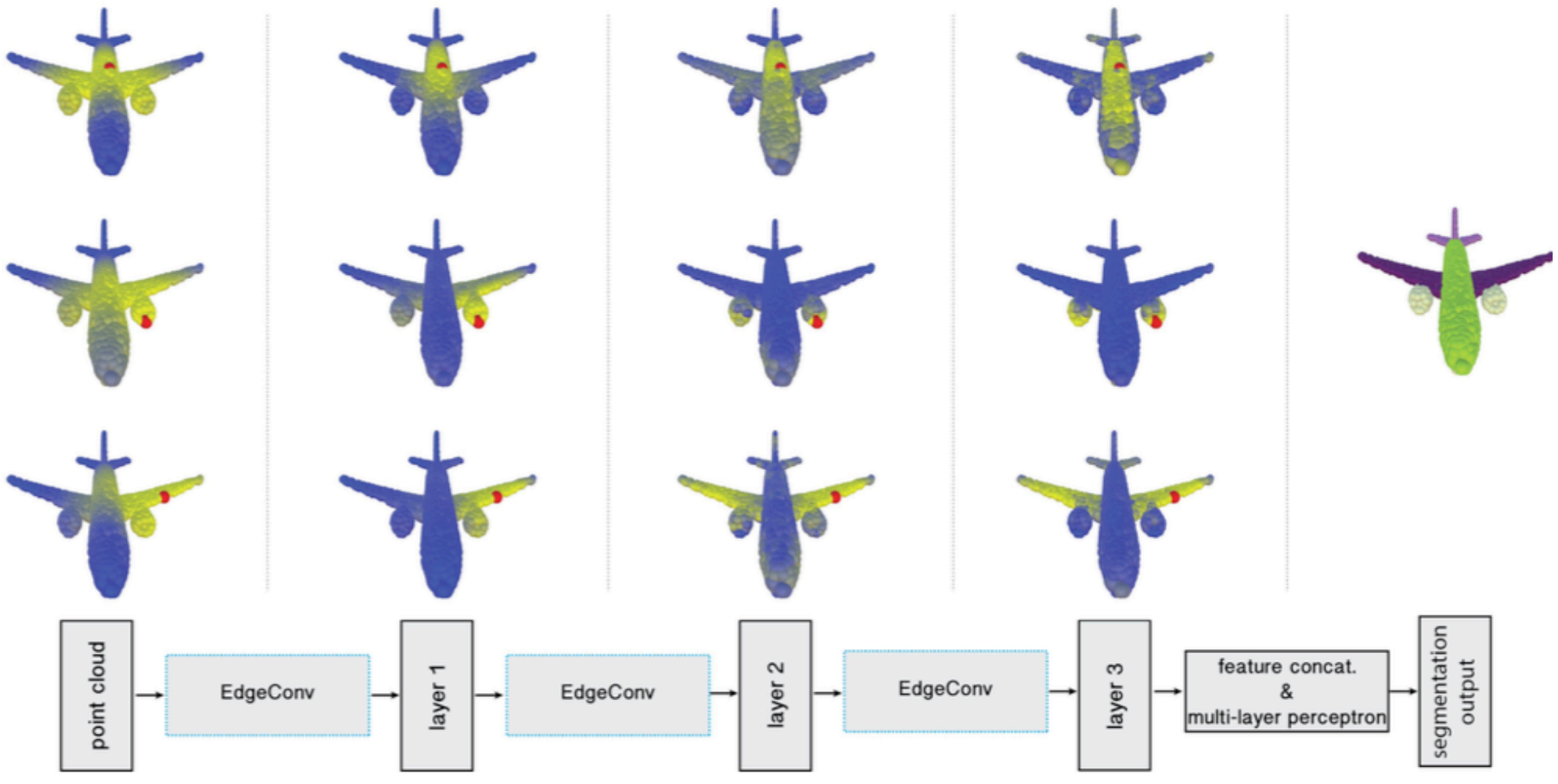
Omnidirectional images

- Sphere is modelled as a graph and classical operation (convolution,

[Perraudin et al., “DeepSphere”, Astronomy and Computing, 2019]
[Bidgoli et al, OSLO: On-the-Sphere Learning for Omnidirectional images and its application to 360-degree image compression, arXiv, 2021]

Point cloud semantic segmentation

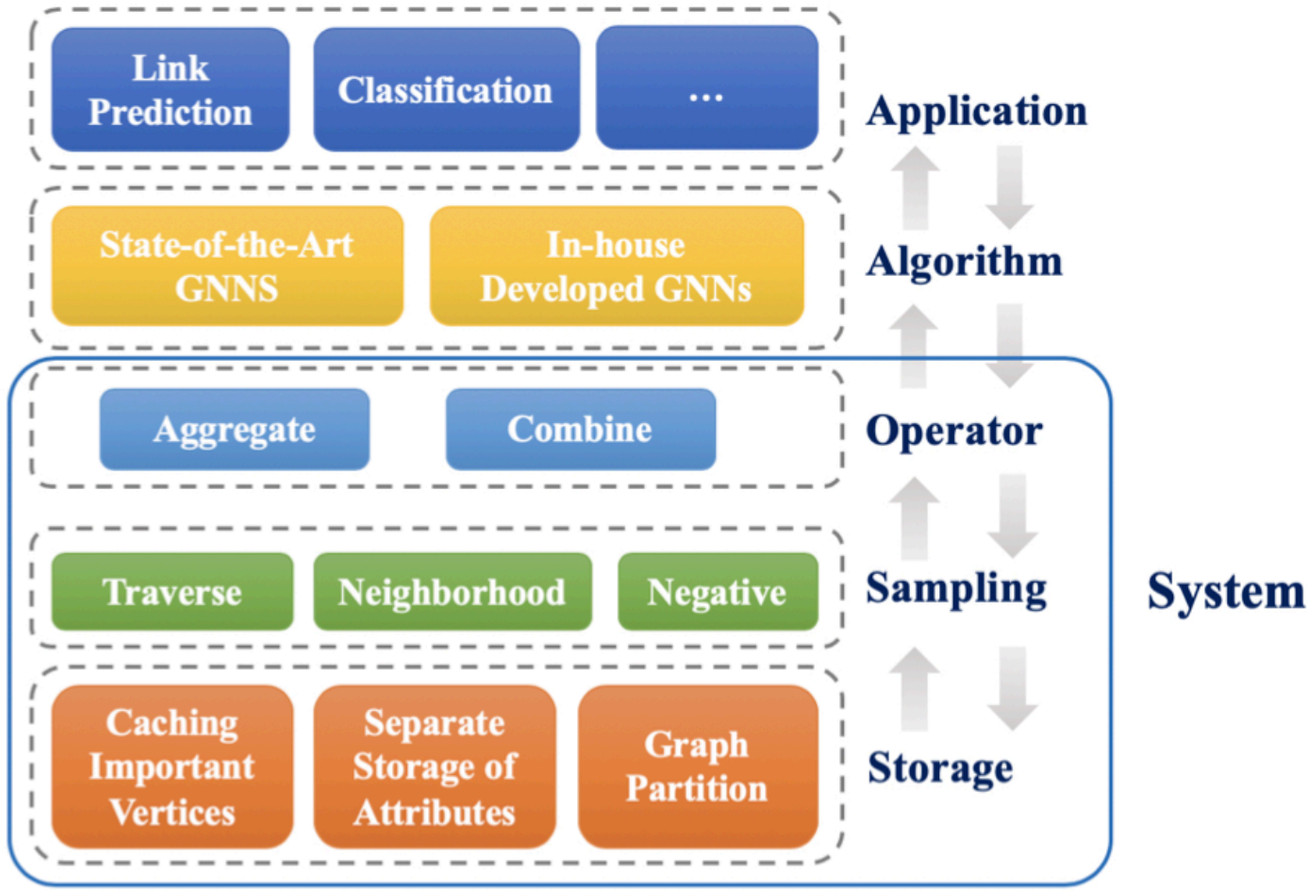
- Graph attention convolution are successful in capturing specific shapes that adapt to the structure of an object



[Wang et al., Graph Attention Convolution for Point Cloud Semantic Segmentation, CVPR, 2019]

Recommender systems: Aligraph

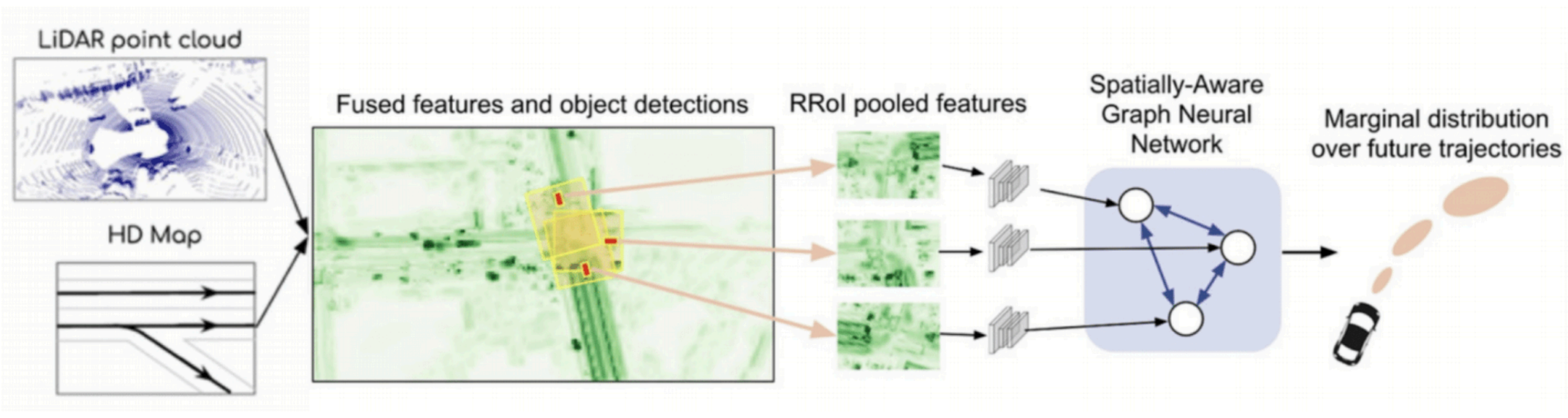
- The system is currently deployed at Alibaba to support product recommendation and personalized search at Alibaba's E-Commerce platform



[Zhu et al., *AliGraph*: A Comprehensive Graph Neural Network Platform, 2019]

Self driving cars

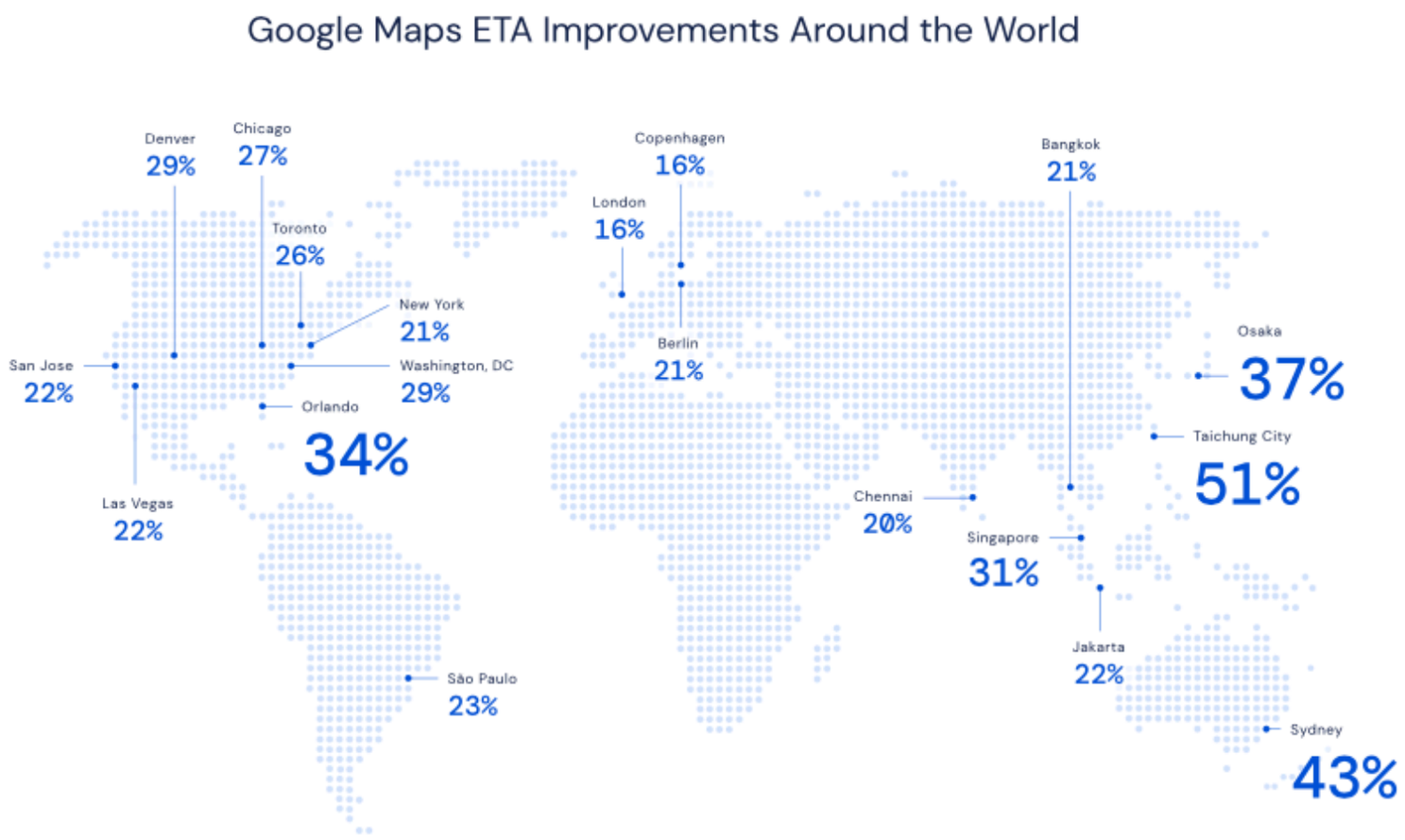
- GNNs provide probabilistic estimates of future trajectories
  - A CNN detects objects
  - A GNN captures interactions between objects and predicts behaviors



[Casas et al, Spatially aware graph neural networks for relational behaviour forecasting for sensor data, ICRA, 2020]

Traffic prediction

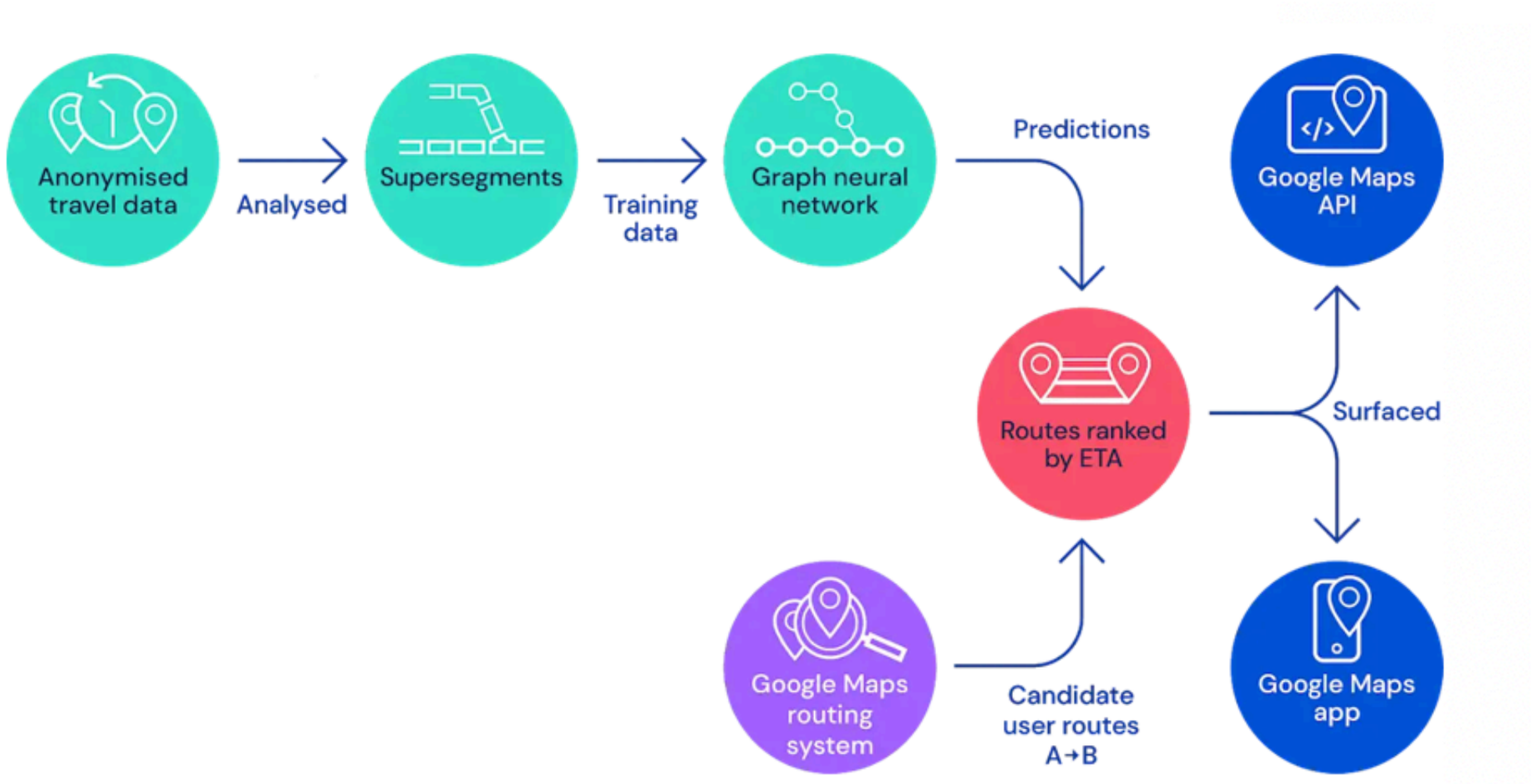
- As the road network is naturally modelled by a graph of road segments and intersections, ETA prediction can be improved with graph representation learning



[Derrow-Pinion et al., ETA Prediction with Graph Neural Networks in Google Maps, 2021]

Traffic prediction

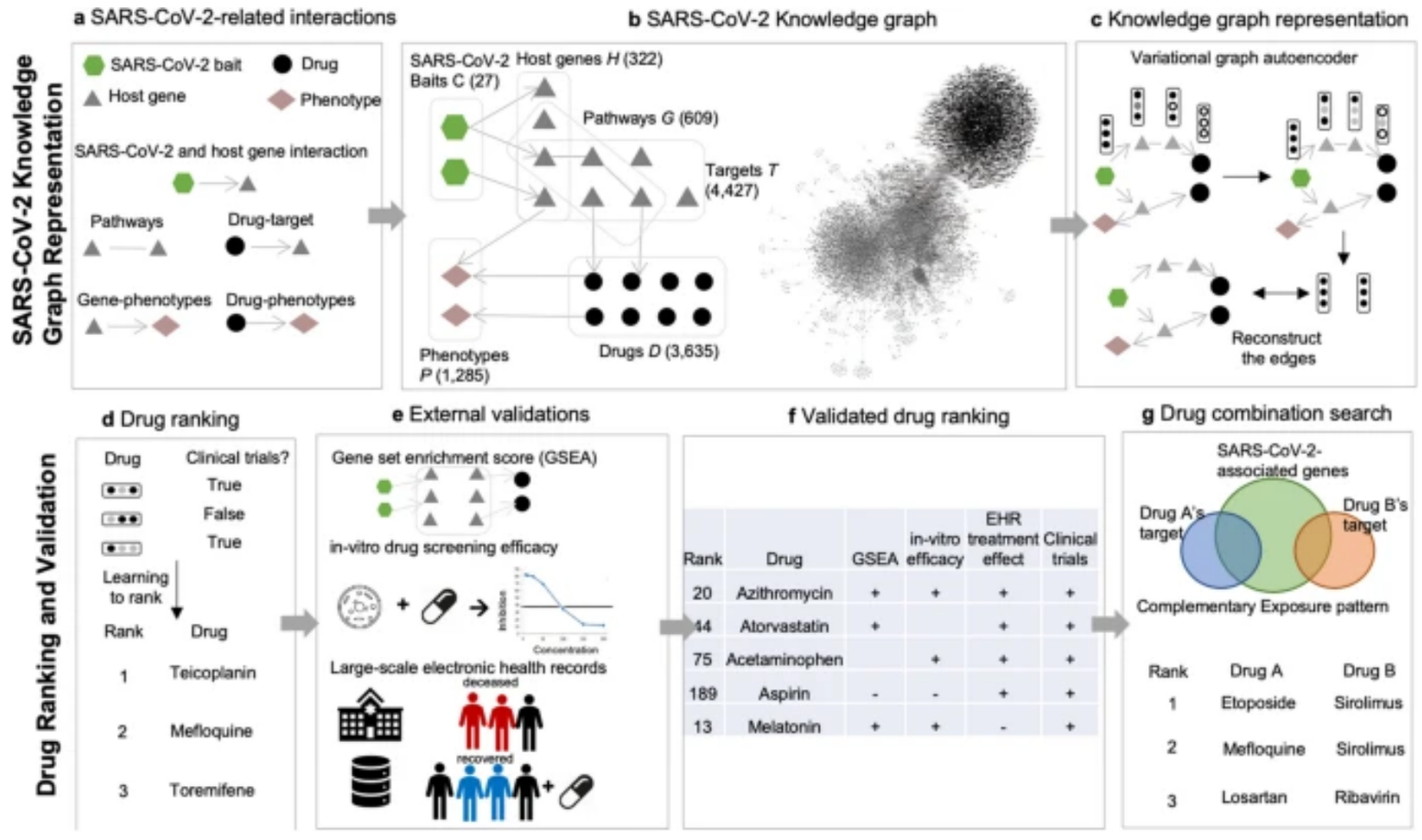
- As the road network is naturally modelled by a graph of road segments and intersections, ETA prediction can be improved with graph representation learning



[Derrow-Pinion et al., ETA Prediction with Graph Neural Networks in Google Maps, 2021]

Drug repurposing for COVID-19

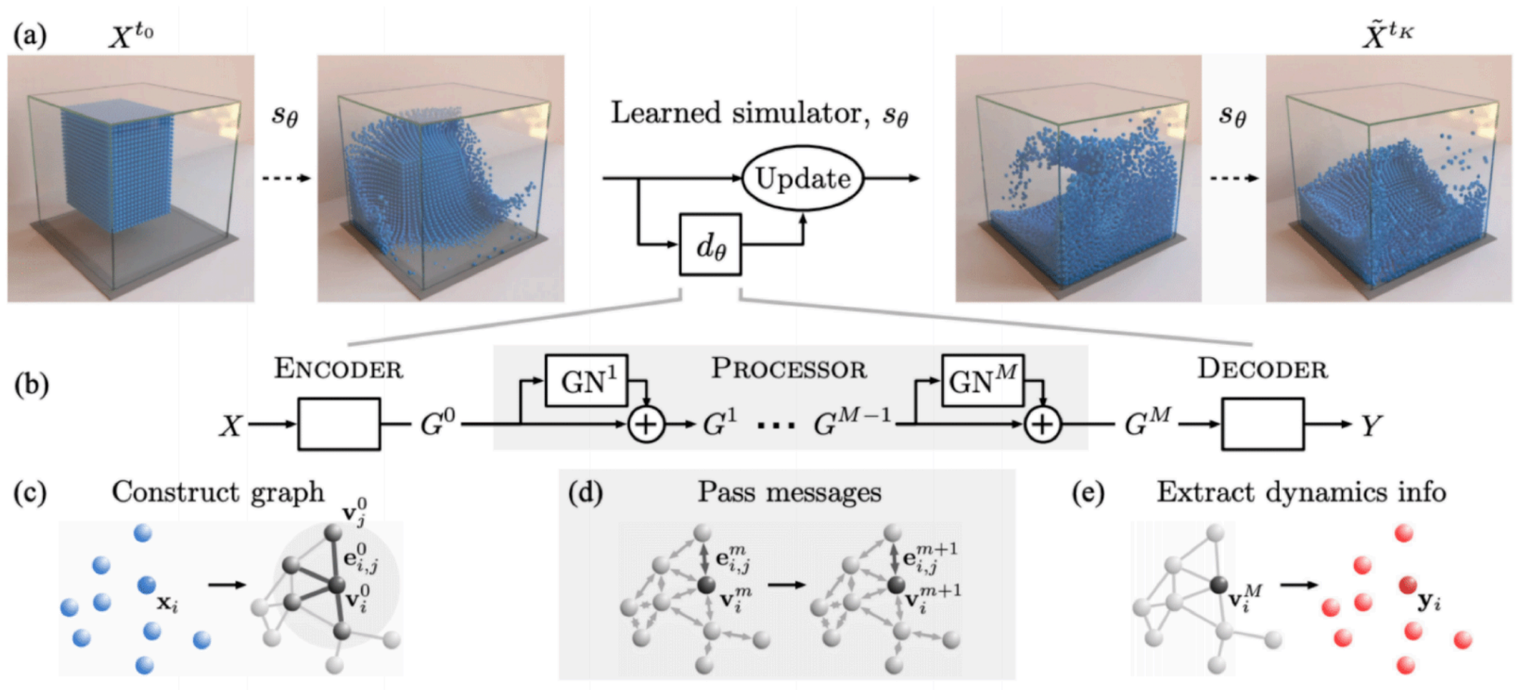
- Deep GNN approaches have been used to derive the candidate drug's representation based on the biological interactions



[Hsieh et al, Drug repurposing for COVID-19 using graph neural network and harmonizing multiple evidence, Nature Sc. Rep., 2021]

Learning physical simulations

- Mesh-based simulations are central to modeling complex physical systems
- High-dimensional scientific simulations are very expensive
- GNNs have been used to learn mesh-based simulations and predict the dynamics of physical systems

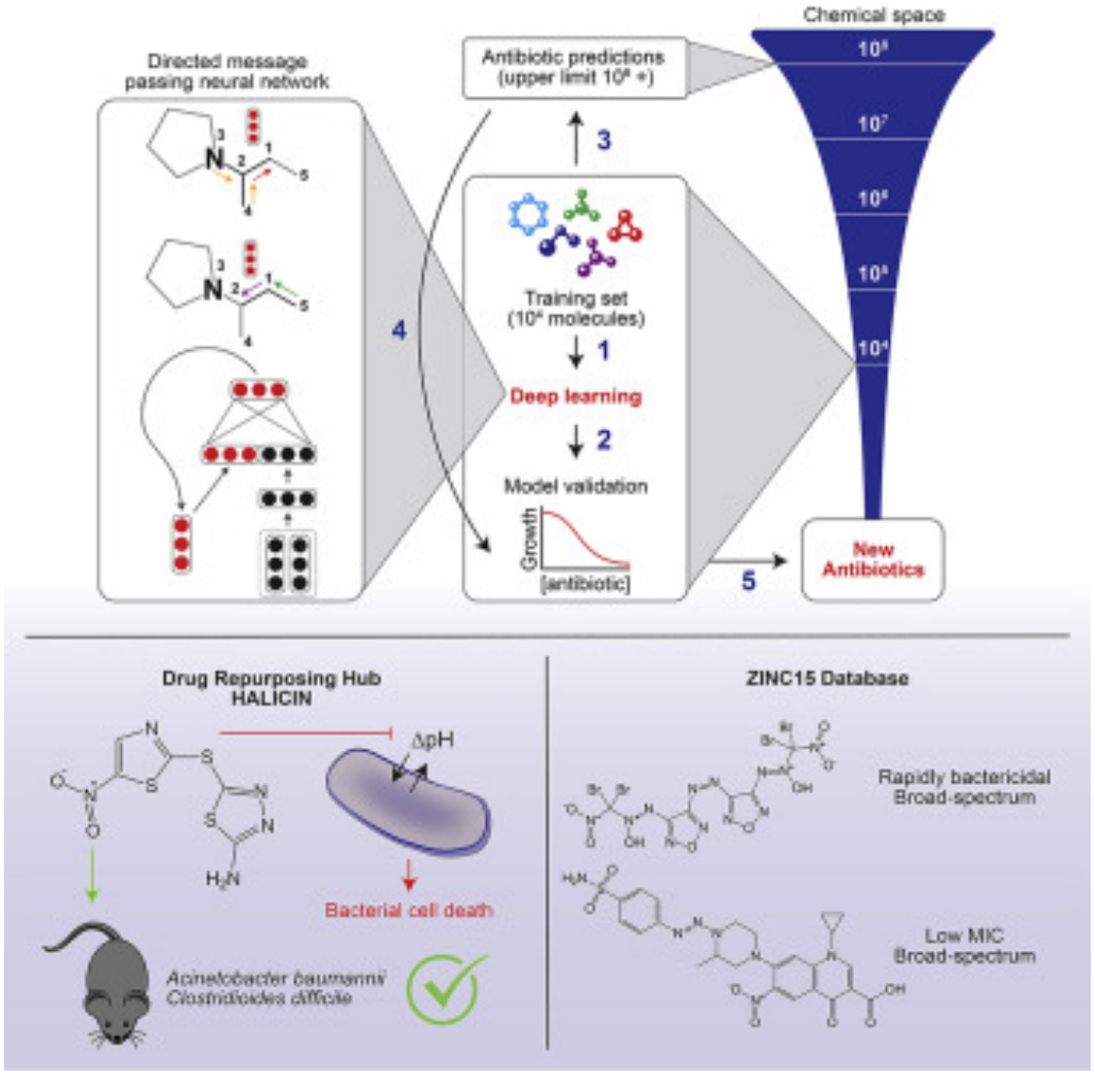


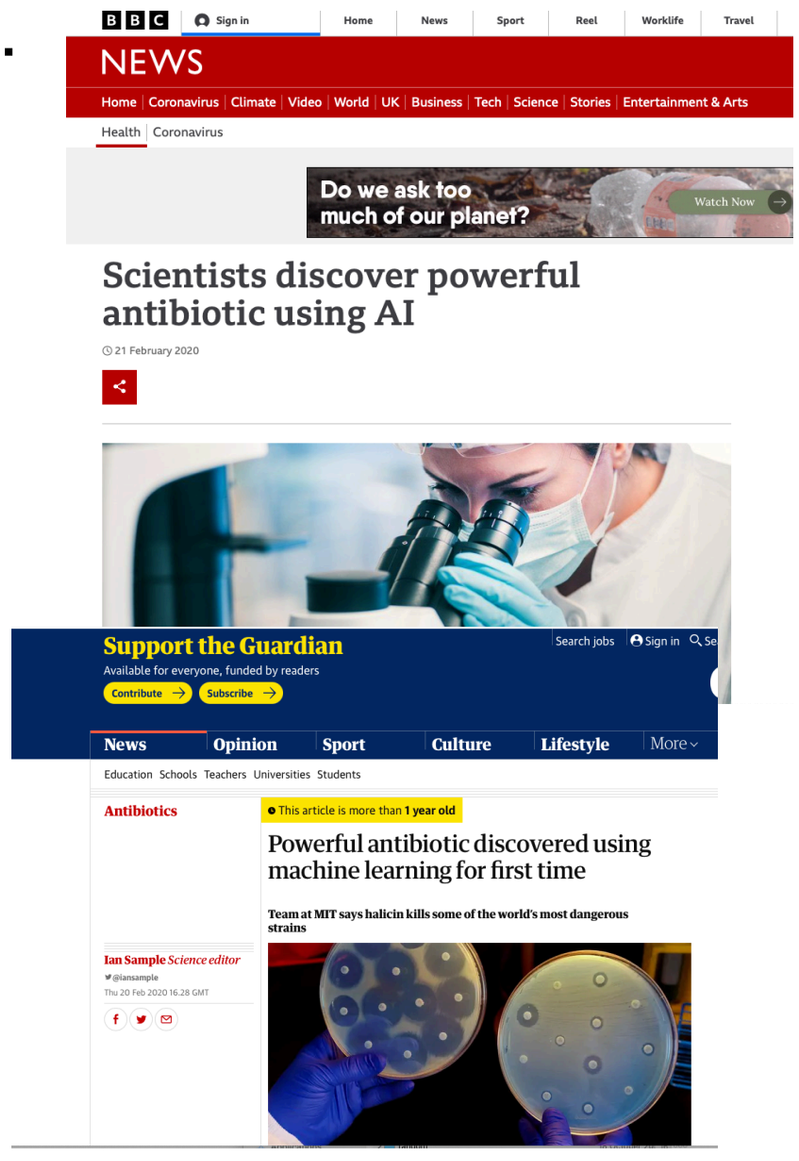
[Sanchez-Gonzales et al, Learning to Simulate Complex Physics with Graph Networks, ICML 2020]

[Pfaff et al, Learning mesh-based simulation with Graph Networks, ICML 2021]

Molecular graph generation

- Recent advances in antibiotic discovery …





[Simonovsky et al, 2017, De Cao et al 2018, Stokes et al 2020]

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

4. Introduction to Graph Signal Processing, Antonio Ortega, Cambridge University Press, 2022

Thank you!