Graph neural networks: Building blocks

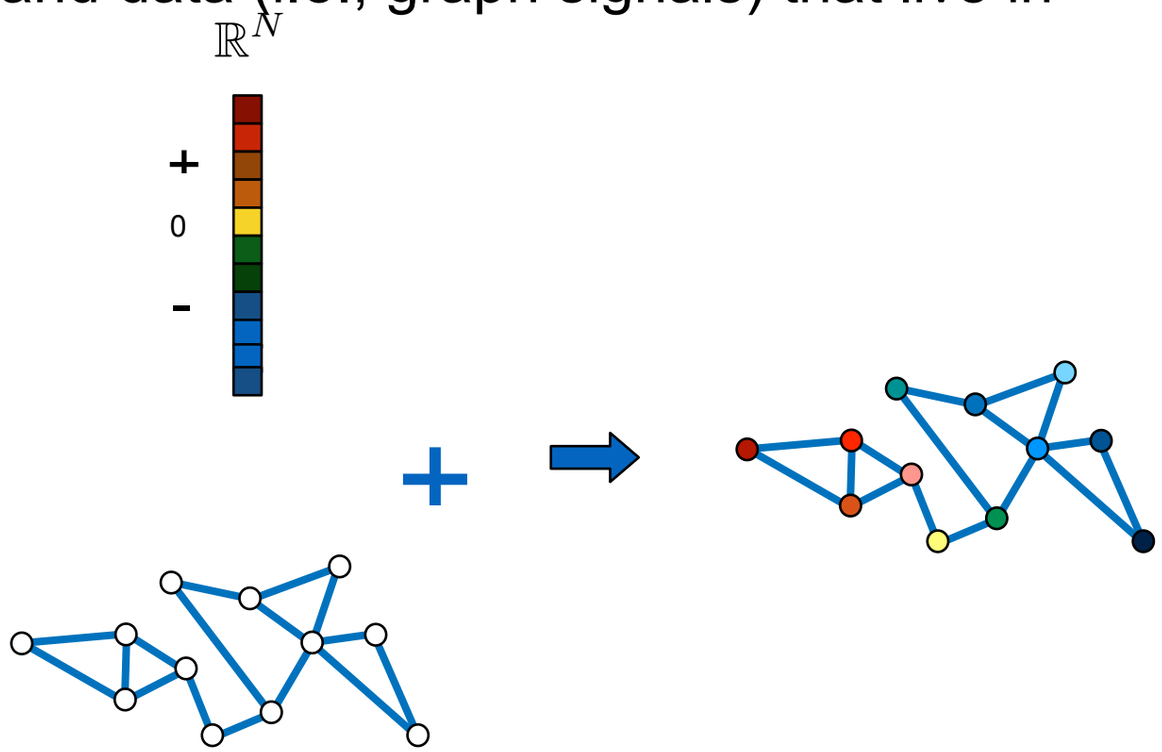
Dr Dorina Thanou

May 1, 2023

Recap from previous class

- Graphs are flexible tools to model the data domain

- Going beyond graph structure:
  - Jointly consider domain (i.e., graph) and data (i.e., graph signals) that live in that domain



- Useful information can be extracted by generalizing classical signal processing tools to the graph domain

How to extract information from graph data: A summary so far

✓ **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, graph signal processing

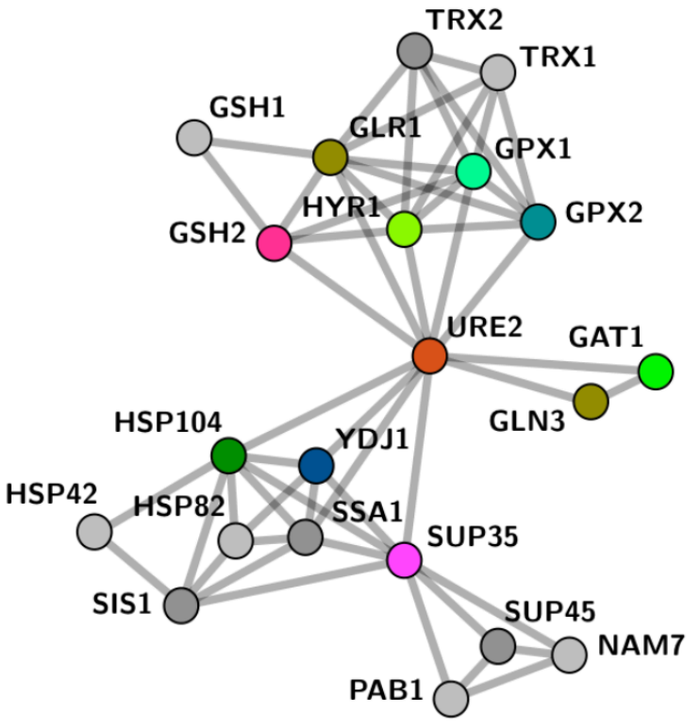
Model-driven

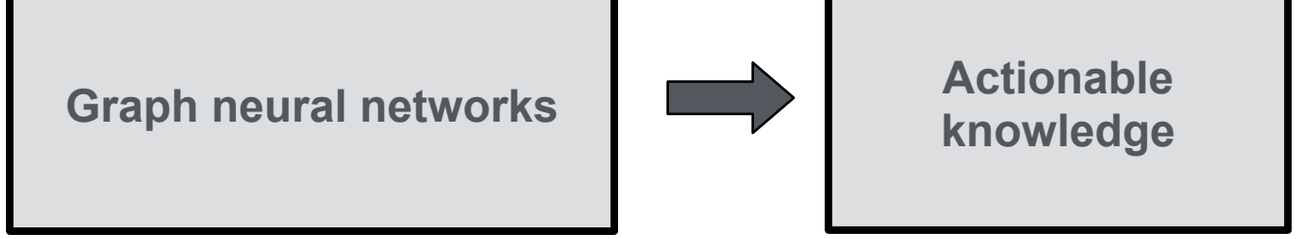
- **Learned features:** Learn graph features directly from data by designing models based on meaningful assumptions
  - ✓ **Unsupervised embeddings:** Learn features based on different ways of preserving information from the original graph (without node attributes)
  - ➡ **Graph neural network features:**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

Data-driven

Today: Graph neural networks (GNNs)

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

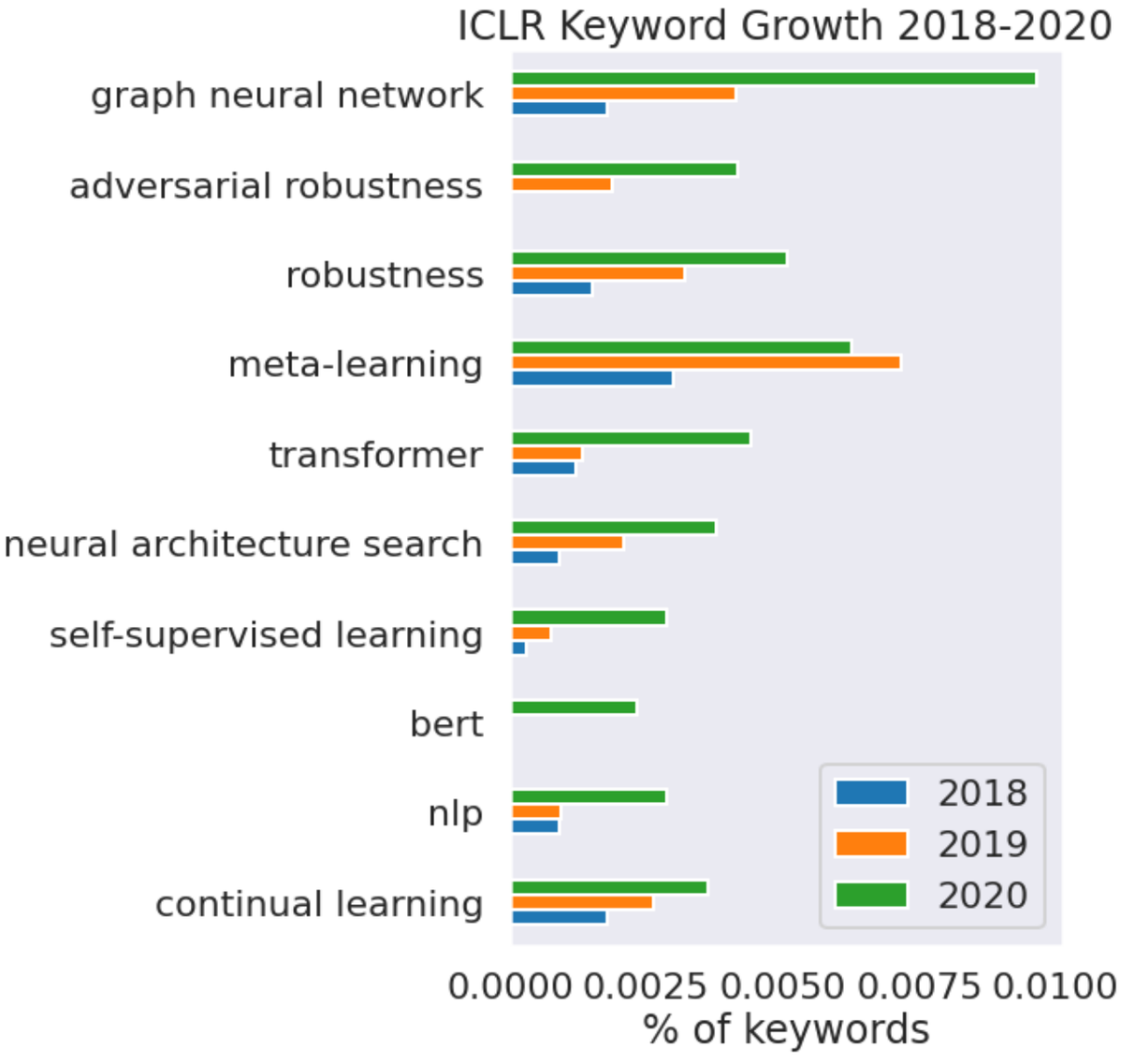


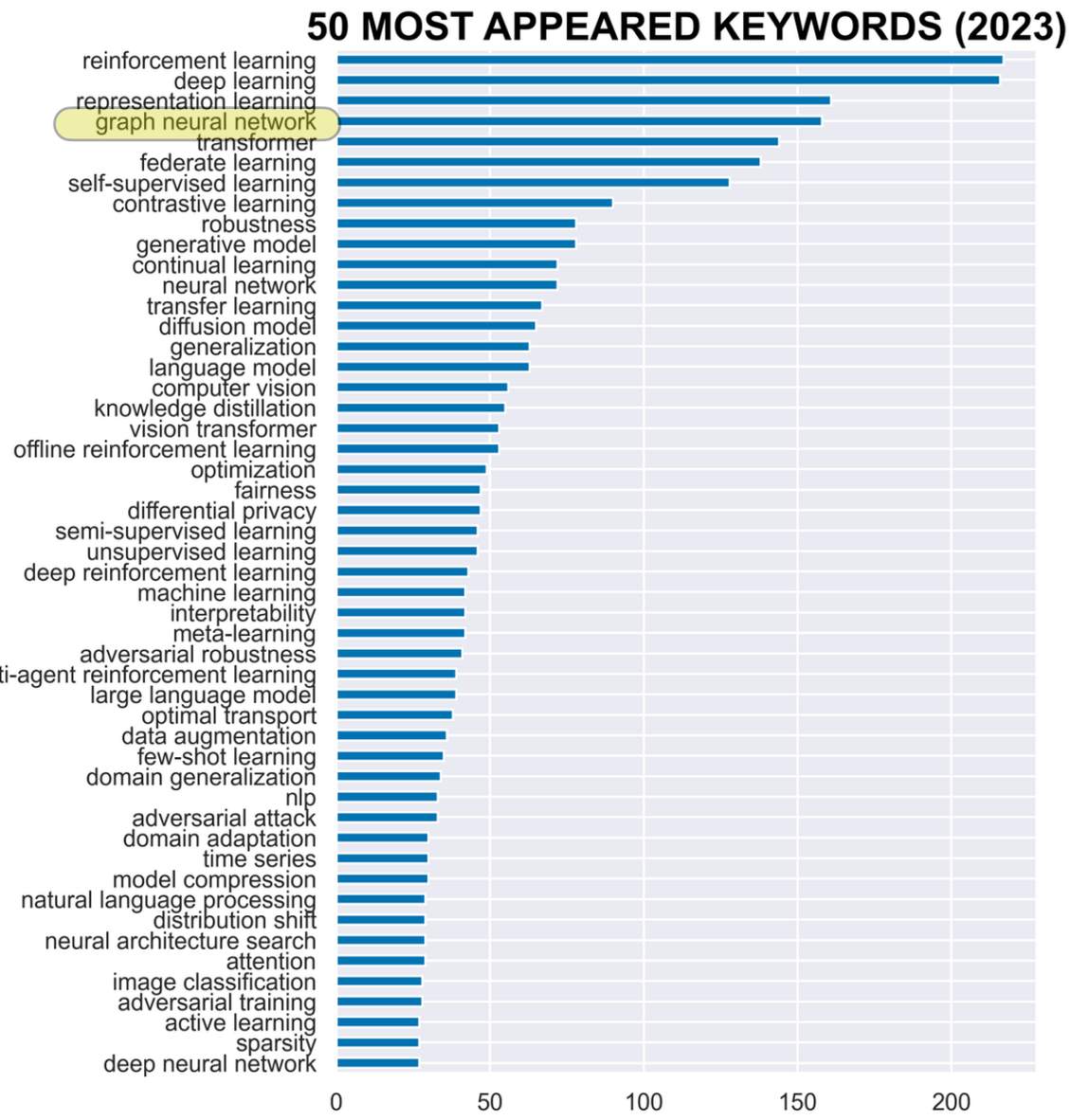


e.g., node/graph classification, signal inpainting/denoising

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

GNNs: A growing trend





ICLR 2023

Outline

- From CNNs to GNNs
- Key building blocks of GNNs
  - Graph convolution
    - A spectral approach
    - A spatial approach
  - Local and global pooling
  - Loss functions

Outline

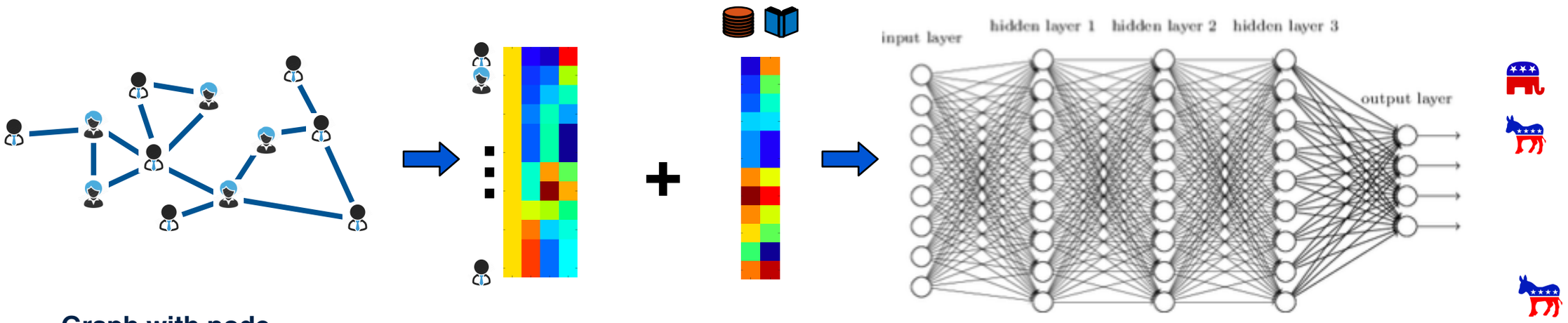
- **From CNNs to GNNs**

- Key building blocks of GNNs

  - Graph convolution
    - A spectral approach
    - A spatial approach
  - Local and global pooling
  - Loss functions

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

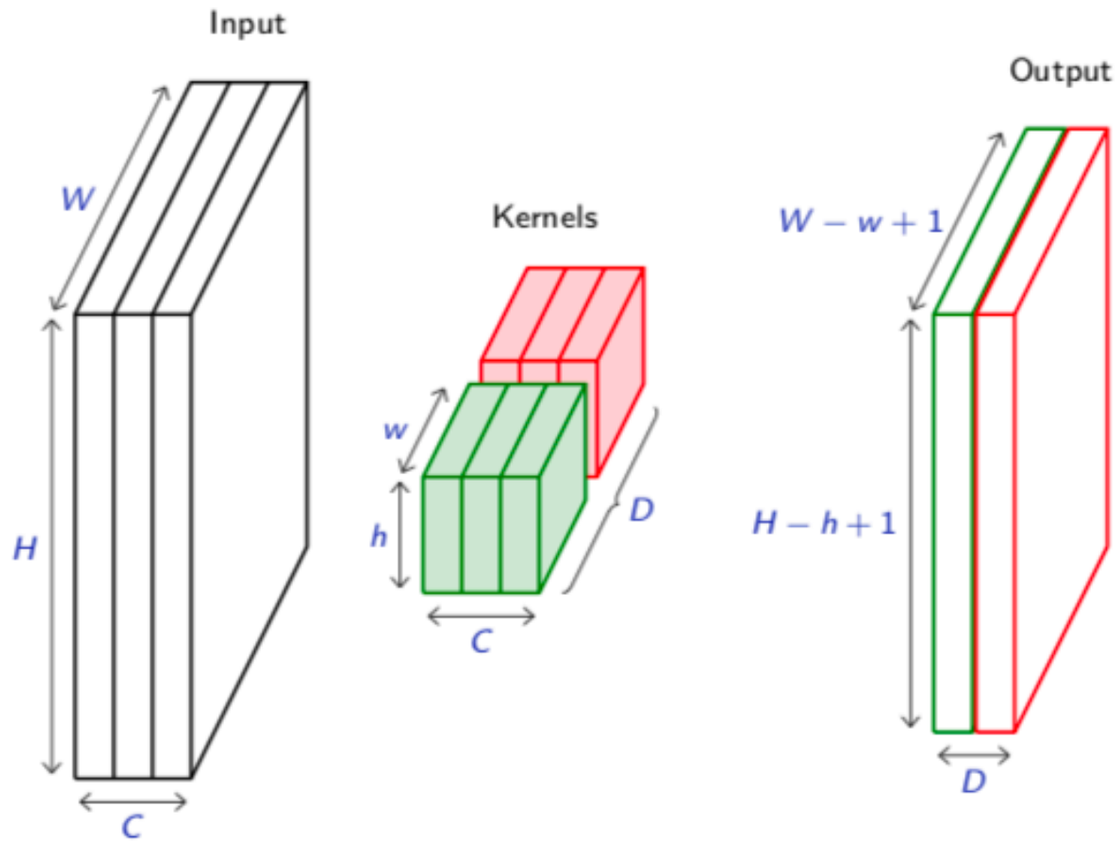
- We build intuition from classical deep learning algorithms
- CNNs exploit structure in the images

Translation invariance

Composability

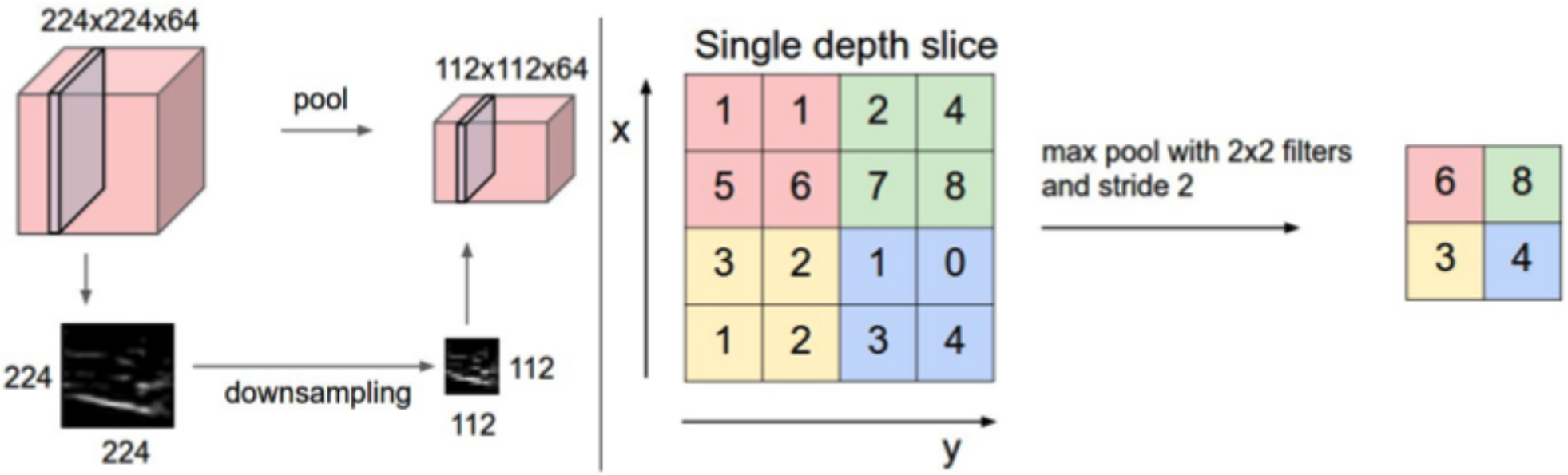
CNNs: Translation invariance

- We leverage spatial information in an image with convolution
- It is achieved by learning a set of convolution filters/kernels, which are applied to an image to identify similar patterns



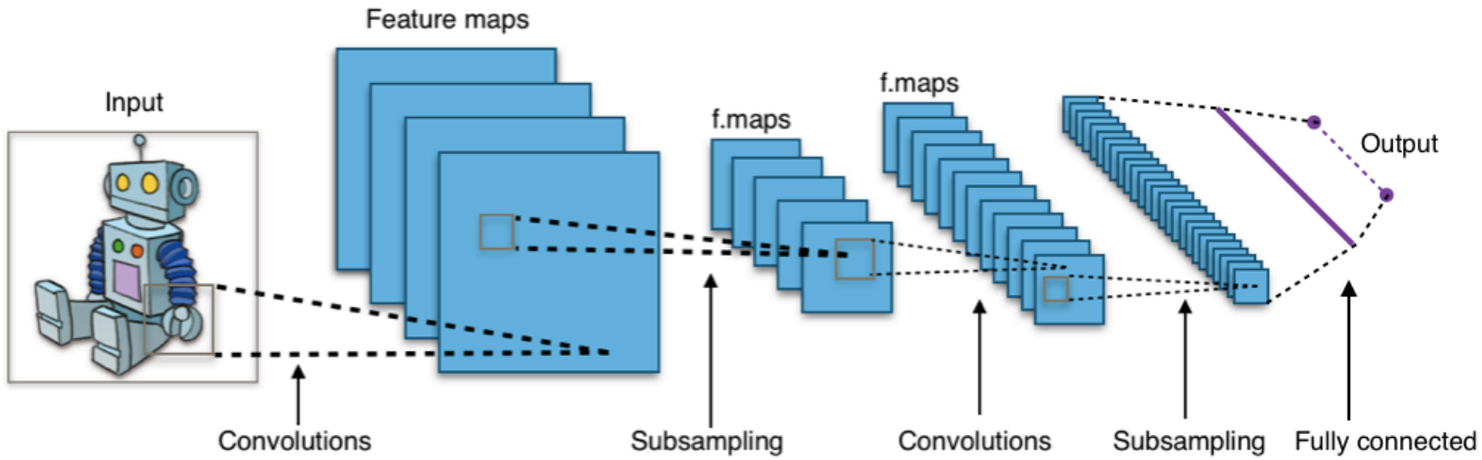
CNNs: Composability

- Aggregate local filters by pooling meaningful pixels to identify bigger patterns



CNN architecture: Illustrative example

- 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

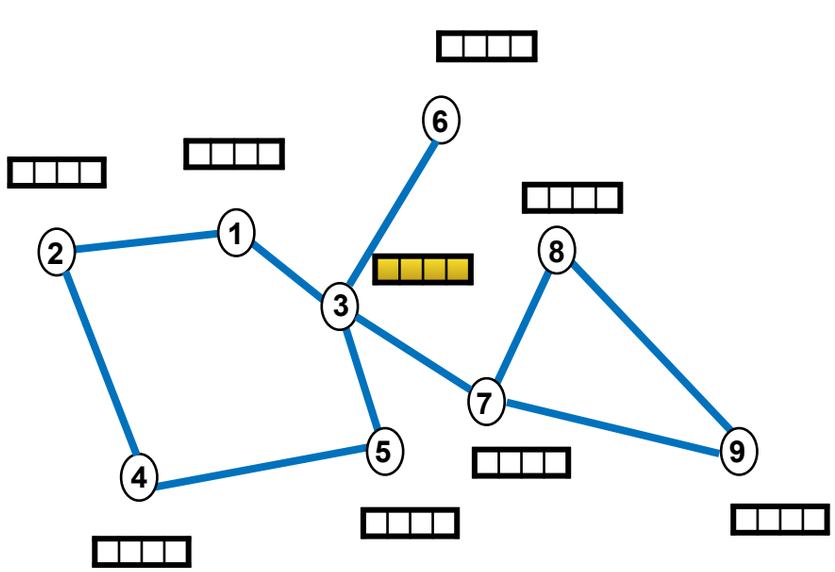
  - **Efficiency:** how to keep the computational complexity low

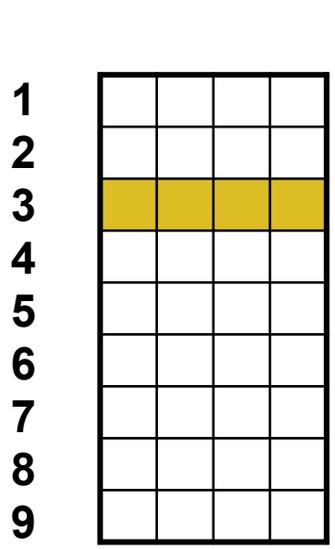
  - **Generalization:** how to build models that generalize to unseen graphs

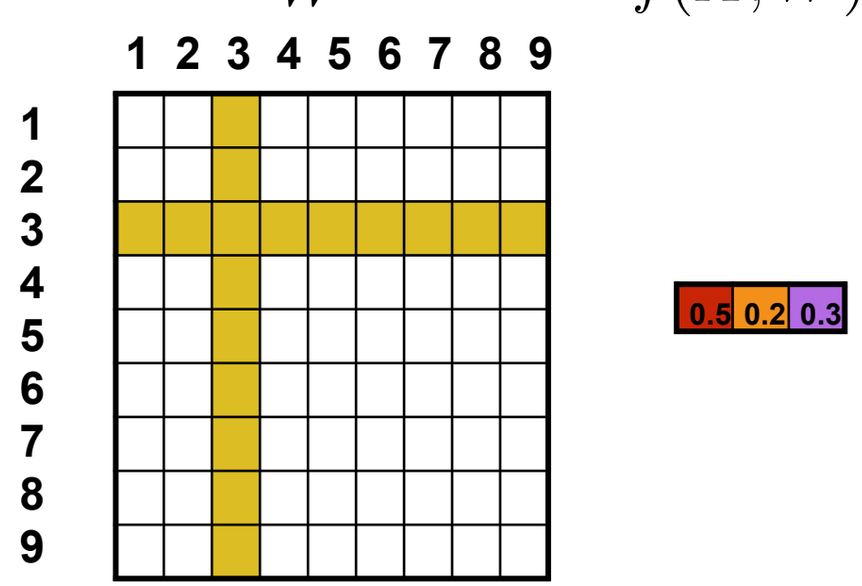
Permutation Invariance

- 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



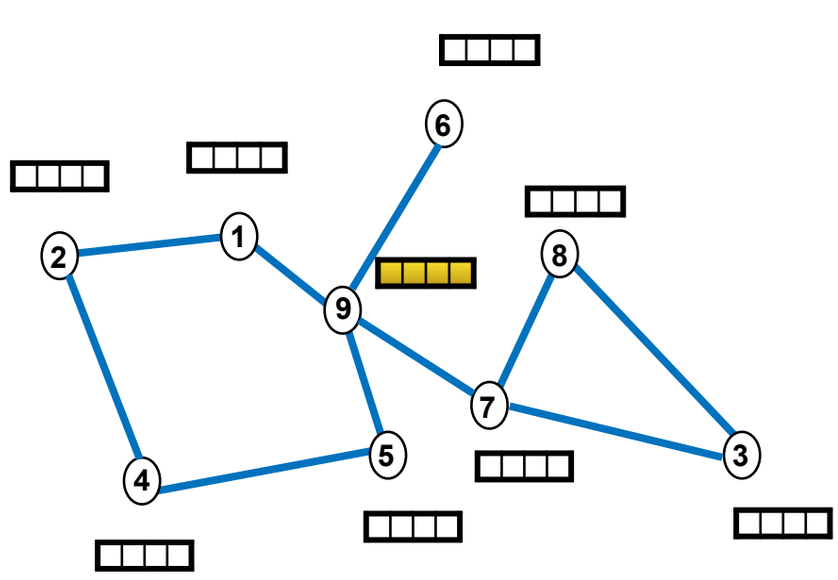




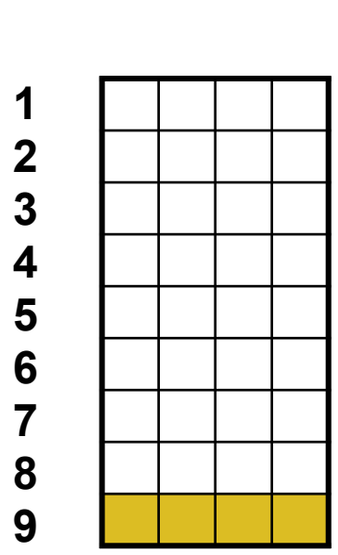
Permutation Invariance

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

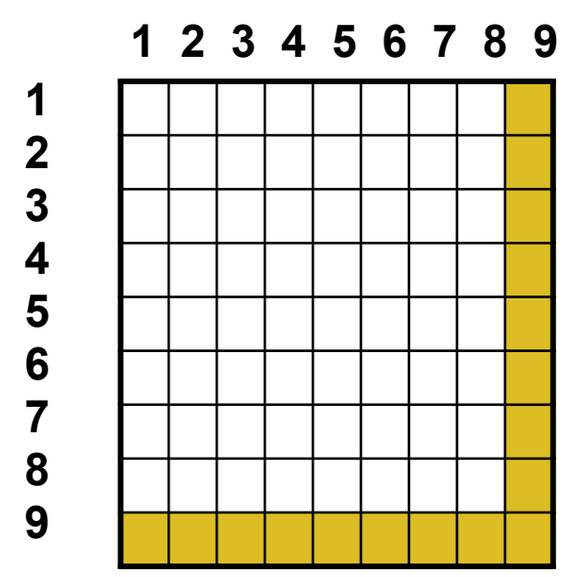
- Graph and node representations should be permutation invariant
  - Graph representations should be invariant to the order of the nodes
  - Similar intuition to translation invariance on images



PX



PWP^T



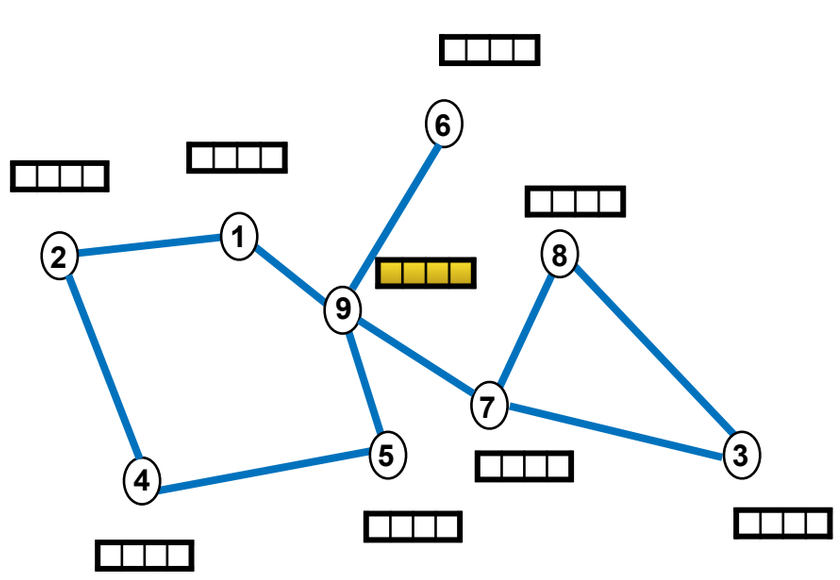
$f(PX, PWP^T)$

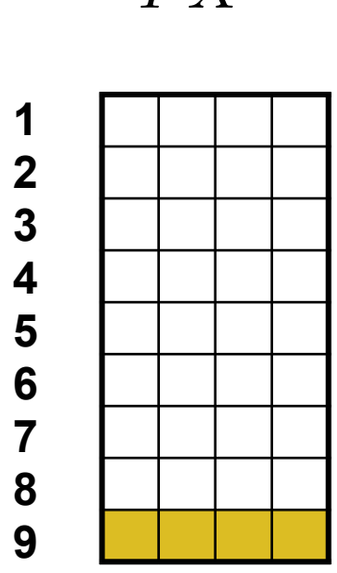
P: permutation matrix

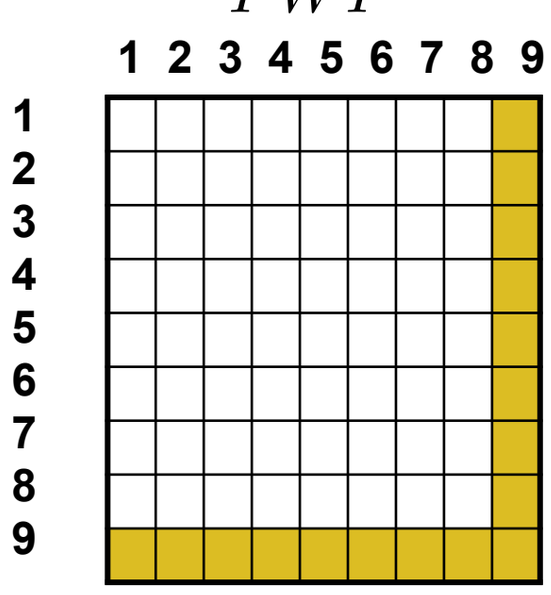
Permutation Invariance

- 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







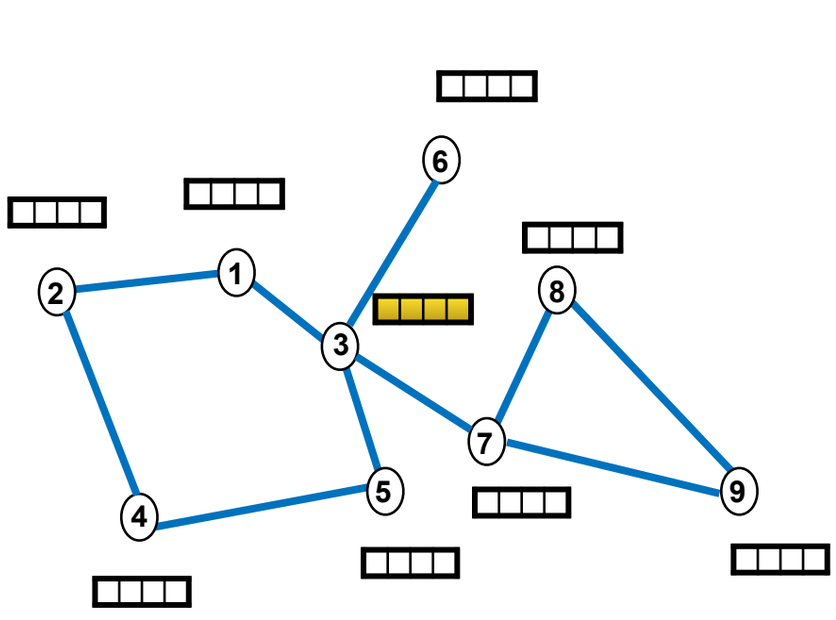
$$f(PX, PWP^T) = f(X, W)$$

P: permutation matrix

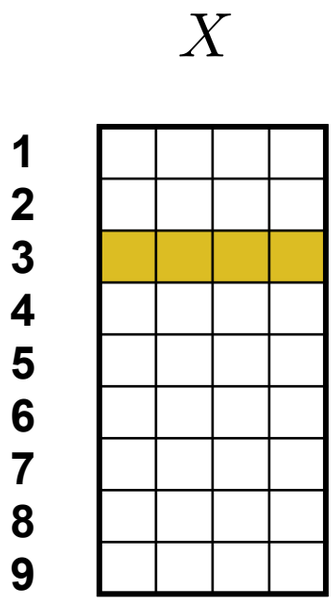
Permutation Equivariance

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

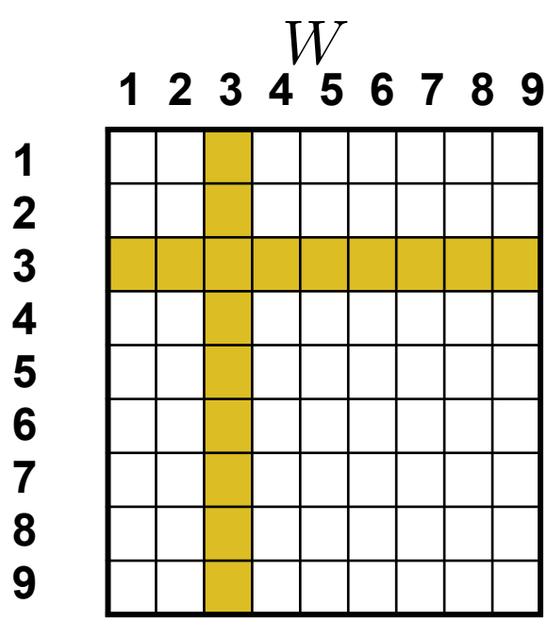
- Node representations should be permutation equivariant
  - If we permute the nodes of the graph, the nodes' output should be permuted in the same way



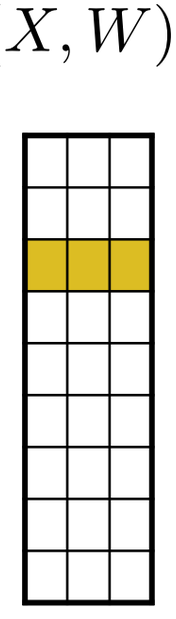
X



W



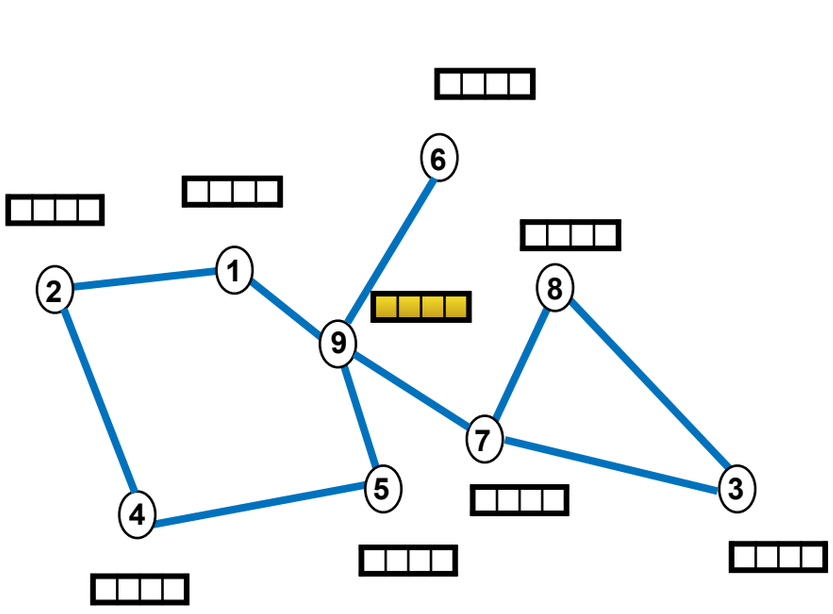
$f(X, W)$



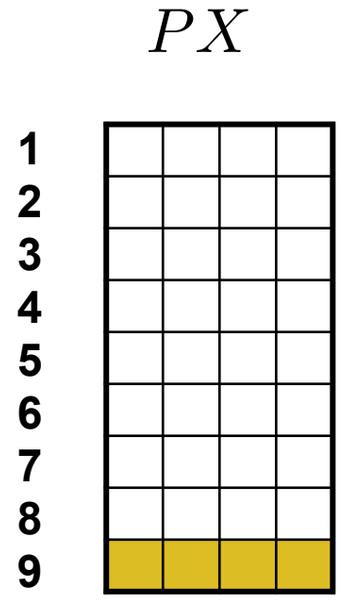
Permutation Equivariance

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

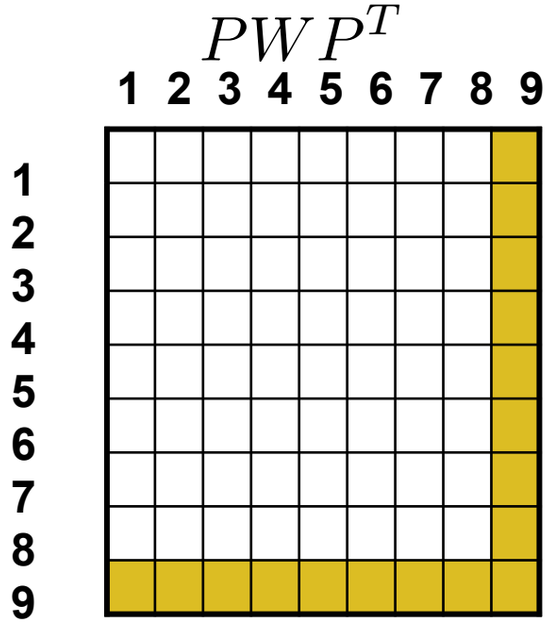
- Node representations should be permutation equivariant
  - If we permute the nodes of the graph, the nodes' output should be permuted in the same way



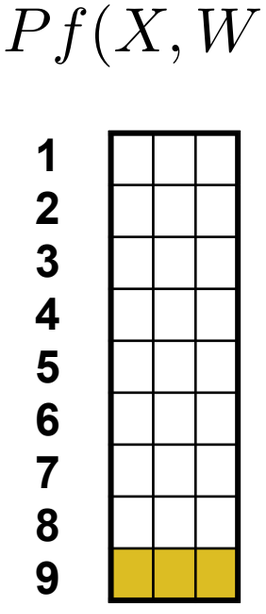
PX



PWP^T



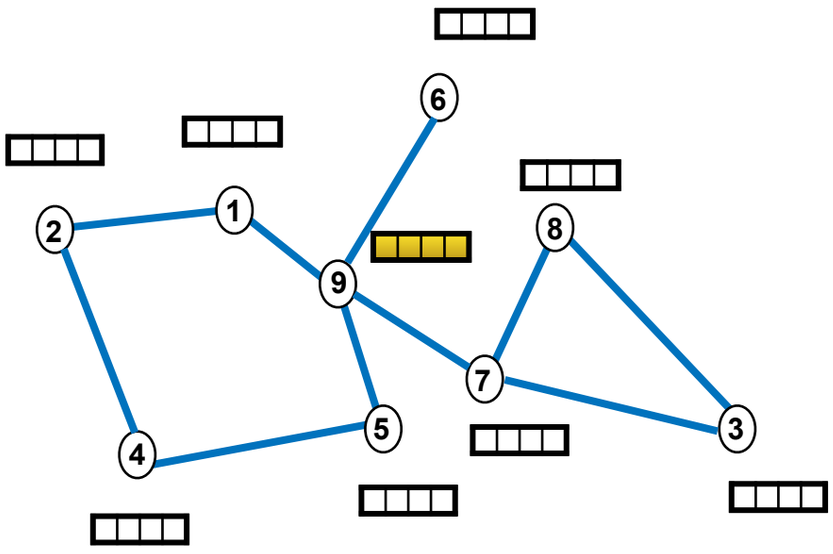
$Pf(X,W)$



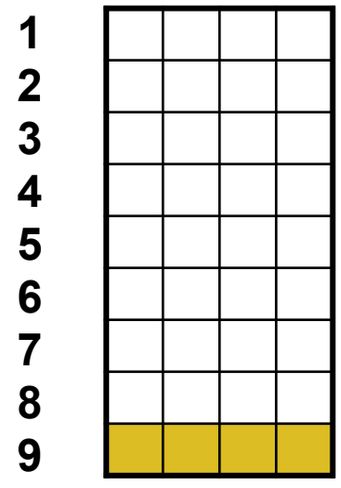
Permutation Equivariance

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

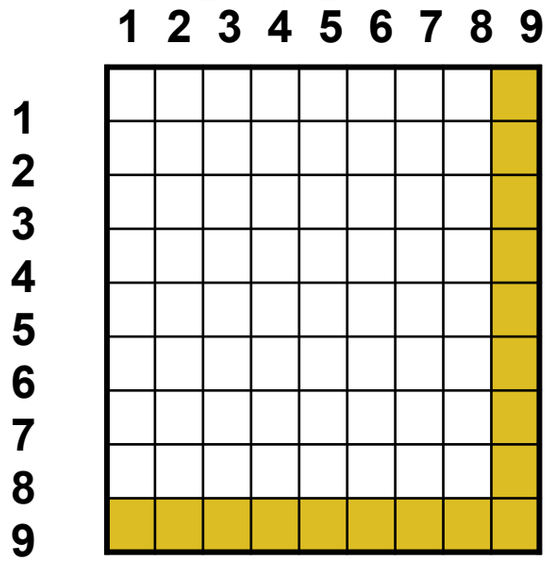
- Node representations should be permutation equivariant
  - If we permute the nodes of the graph, the nodes' output should be permuted in the same way



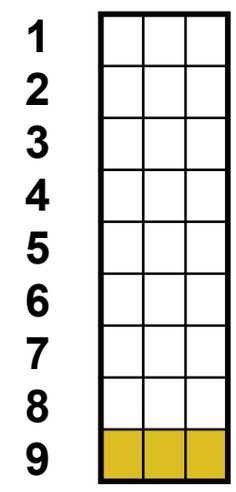
PX



PWP^T



$Pf(X, W)$

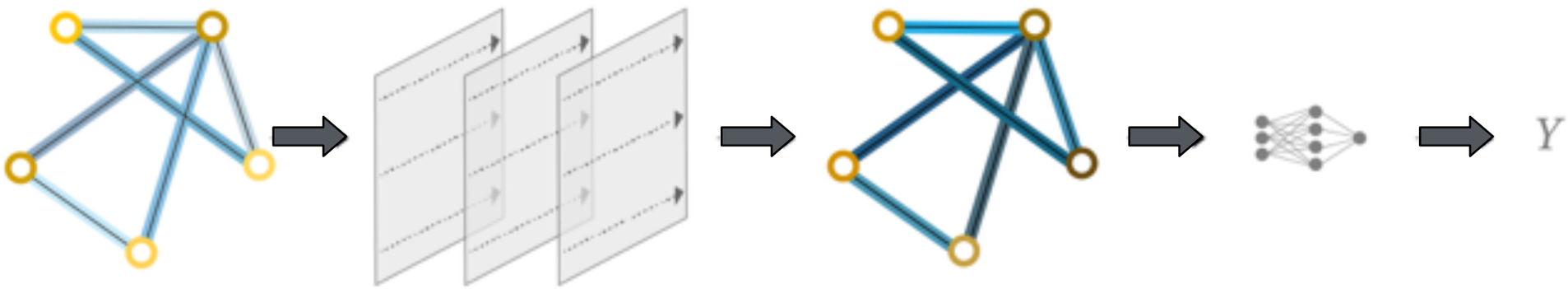


$$f(PX, PWP^T) = Pf(X, A)$$

P: permutation matrix

GNN model: schematic overview

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



Graph and node attributes
$\mathcal{G}, X$

GNN blocks
(graph convolution, pooling, nonlinearity)

Transformed graph
(node/edge embeddings)
$\mathcal{G}', H$

Classification layer

Prediction
(node, edge, graph level)
Y

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

Outline

- From CNNs to GNNs

- Key building blocks of GNNs
  - **Graph convolution**
    - A spectral approach
    - A spatial approach
  - Local and global pooling
  - Loss functions

Towards a convolution on graphs: A spectral viewpoint

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

Classical convolution

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

FT

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

Convolution on graphs

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

IGFT

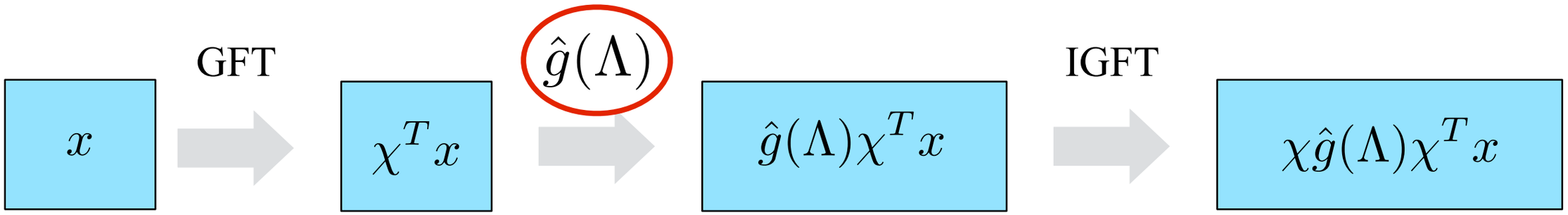
GFT

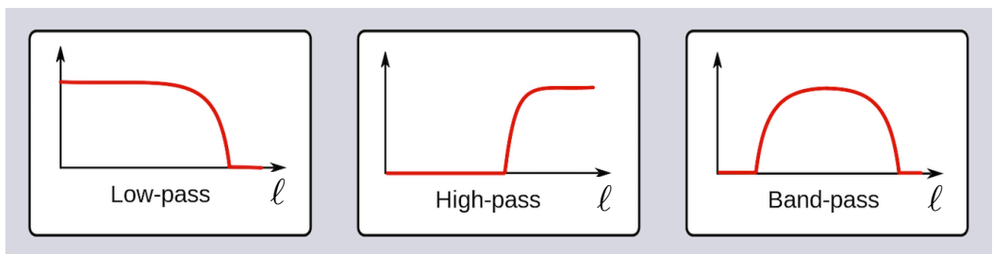
$$\widehat{(x * g)}(\lambda) = \big((\chi^T x) \circ \hat{g}\big)(\lambda)$$

How can we interpret graph convolution?

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

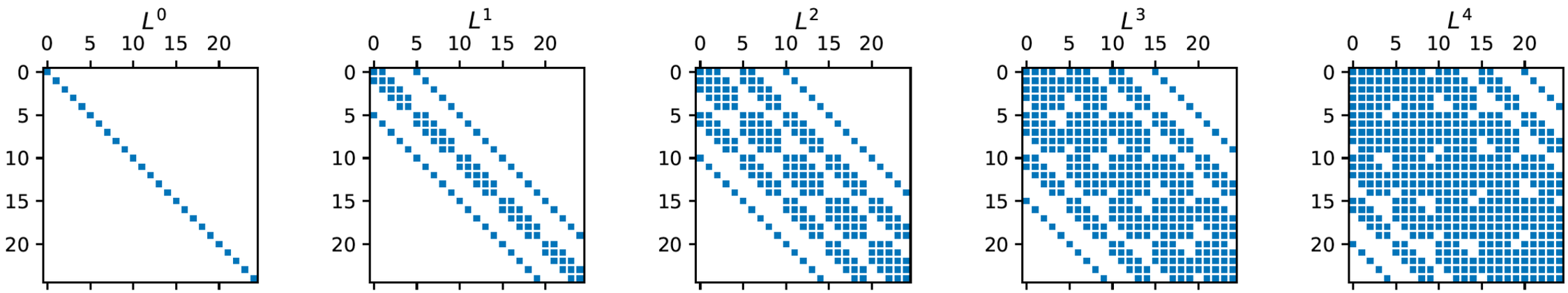
Is the graph convolution localized?

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

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

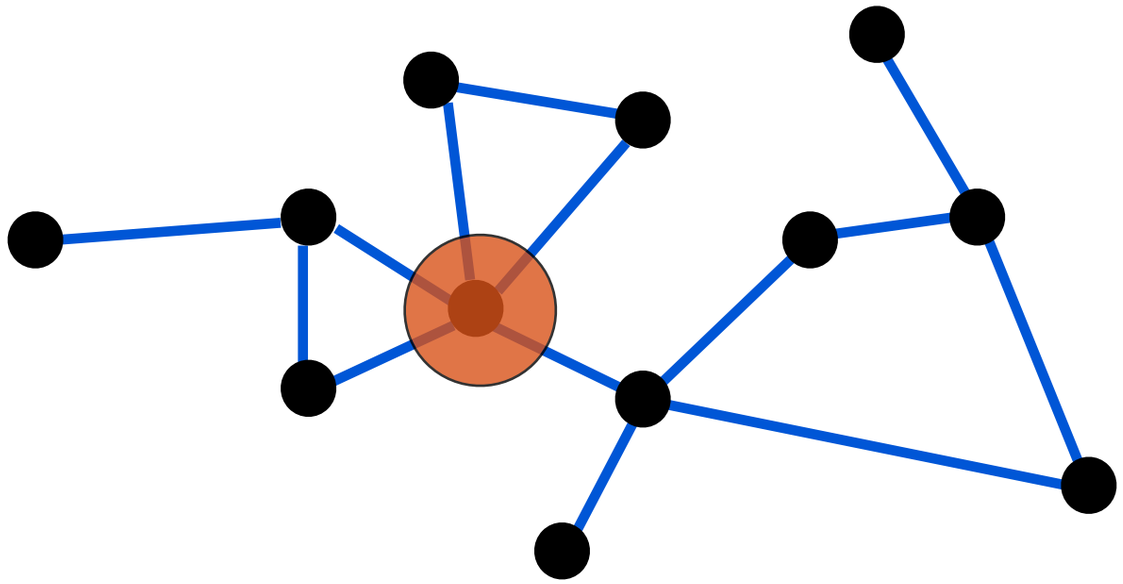
Is the graph convolution localized?

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

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



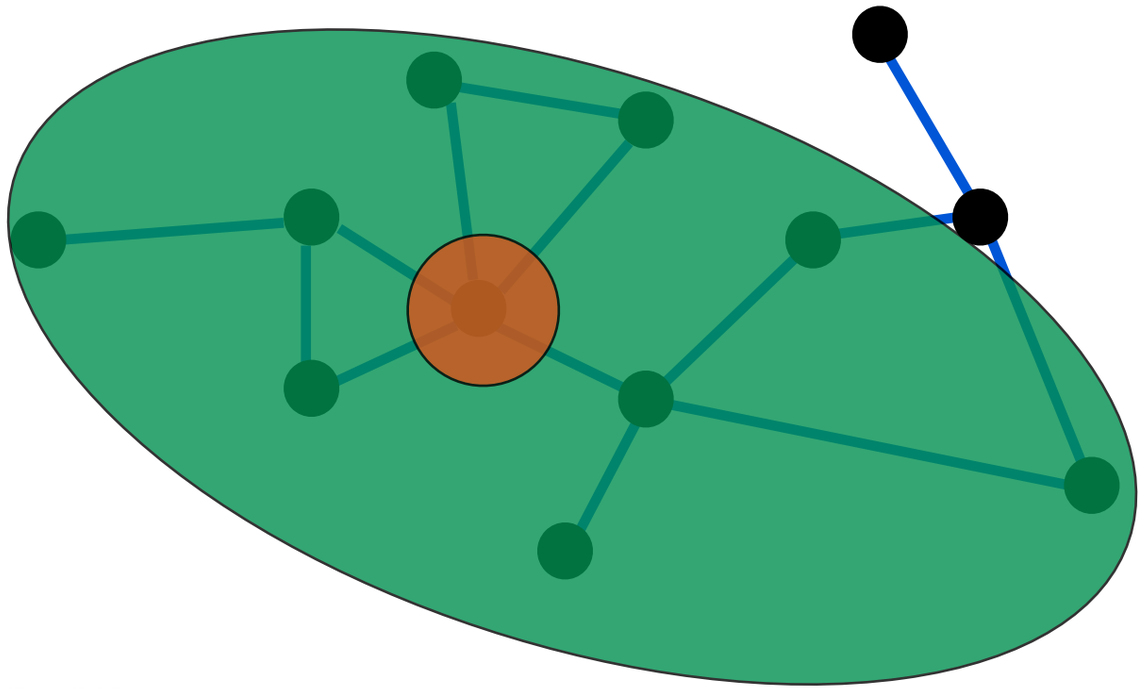
Is the graph convolution localized?

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

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



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

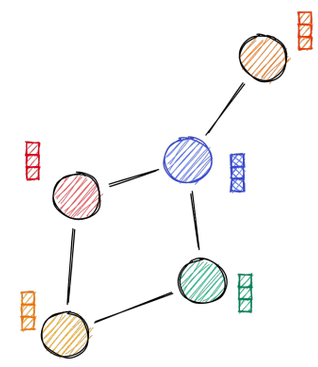
Example: $\hat{g}_\theta(\lambda) = \sum_{j=0}^{K} \theta_j \lambda^j, \ \theta \in \mathbb{R}^{K+1}$ ⇒ $\hat{g}_\theta(L) = \sum_{j=0}^{K} \theta_j L^j$

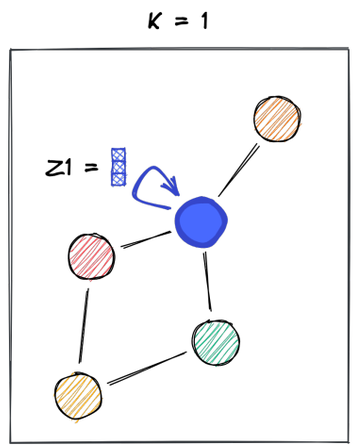
- L^K defines the K-hop neighborhood: $d_G(v_i, v_j) > K \rightarrow (L^K)_{ij} = 0$

A spatial interpretation of graph convolution

- Localization of the Laplacian polynomials leads to 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:

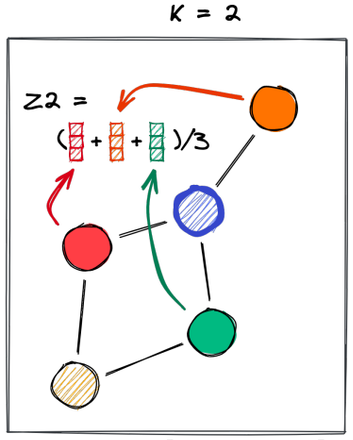
$$z_0 = x$$

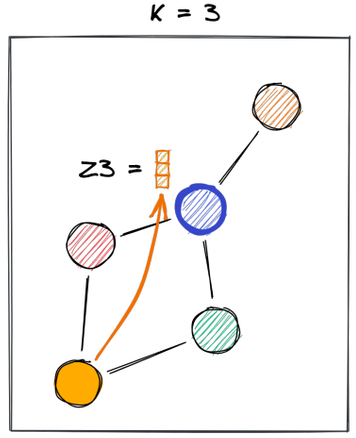
$$z_1 = Lz_0$$

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

$$\vdots$$

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

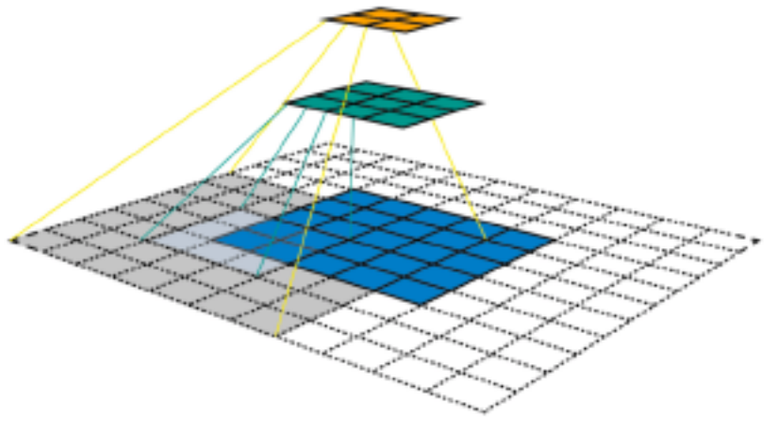




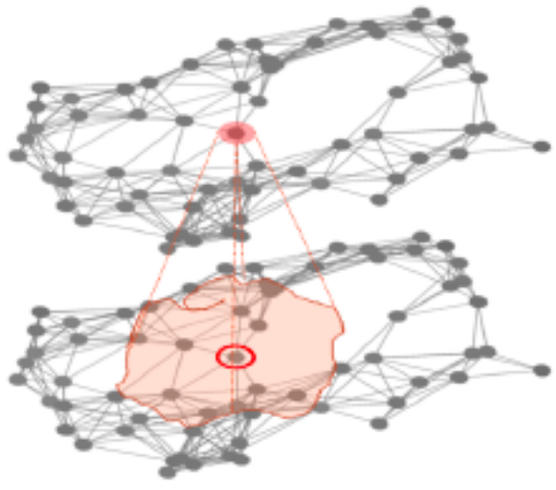
- Graph convolution can be computed recursively by exchanging information in a local neighborhood (i.e., message passing)
- The kernel $\hat{g}(\cdot)$ does not depend on the order of the nodes: permutation invariant!

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



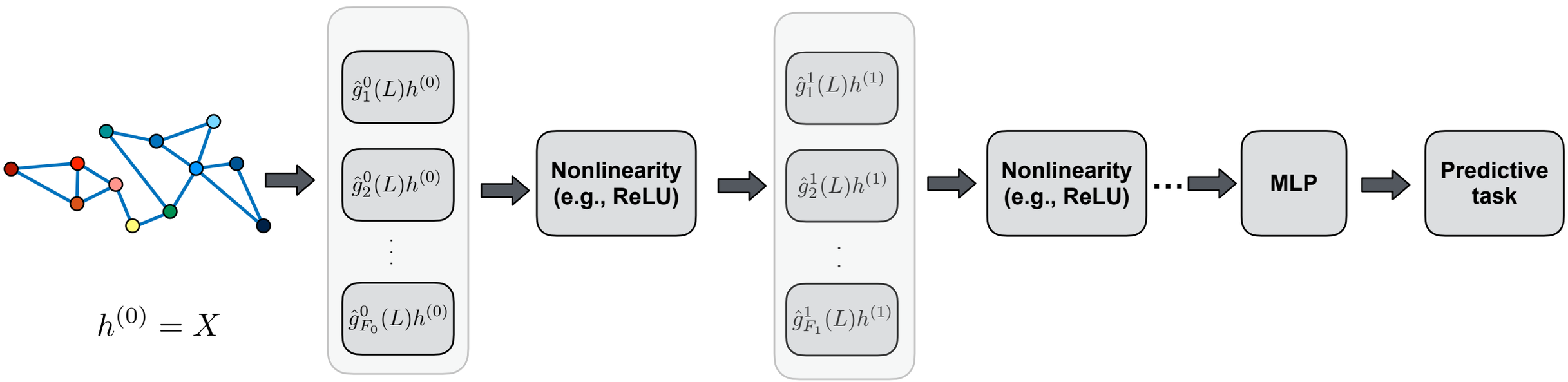
Receptive field on an image



Receptive field on a graph

The basic GNN: a spectral viewpoint

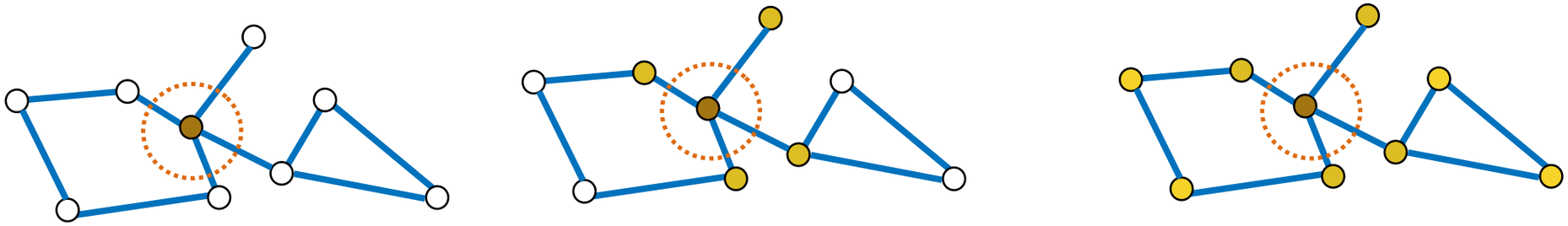
- Typical GNN architectures consist of a set of graph convolutional layers, each of which is followed by element-wise nonlinearity



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

Each layer increases the receptive field of each node

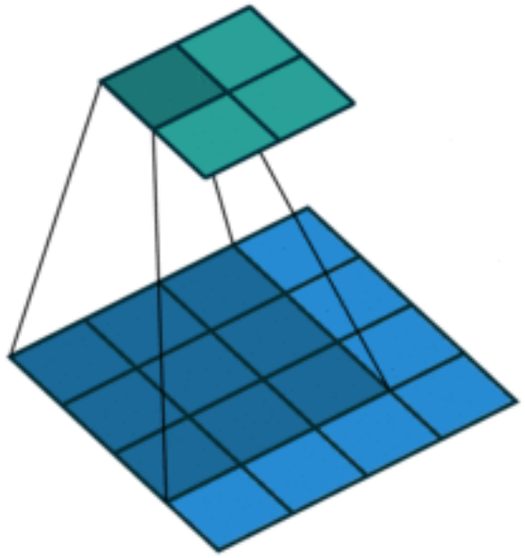
- Each layer increases the receptive field by K hops
- Example: $K = 1$



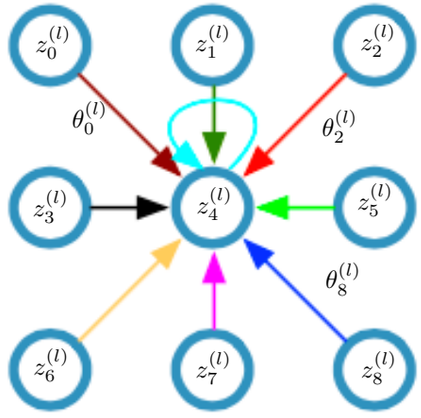
Layer 0 **Layer 1** **Layer 2**

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

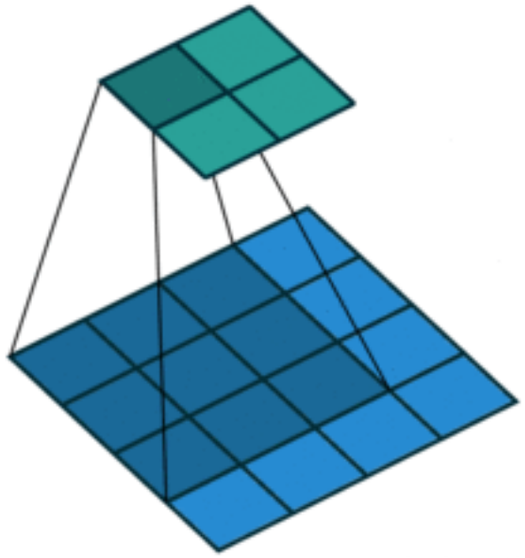


$$z_i^{(l+1)} = \sum_{i=0}^{8} \theta_i^{(l)} z_i^{(l)}$$

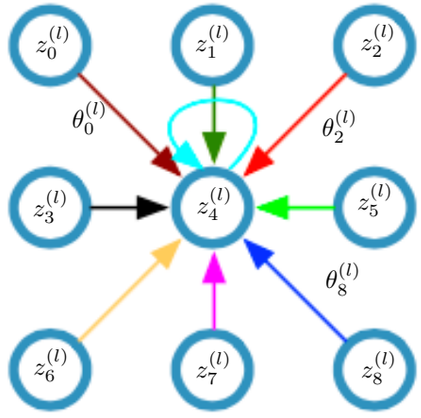
Can we exploit similar structure for graph data?

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



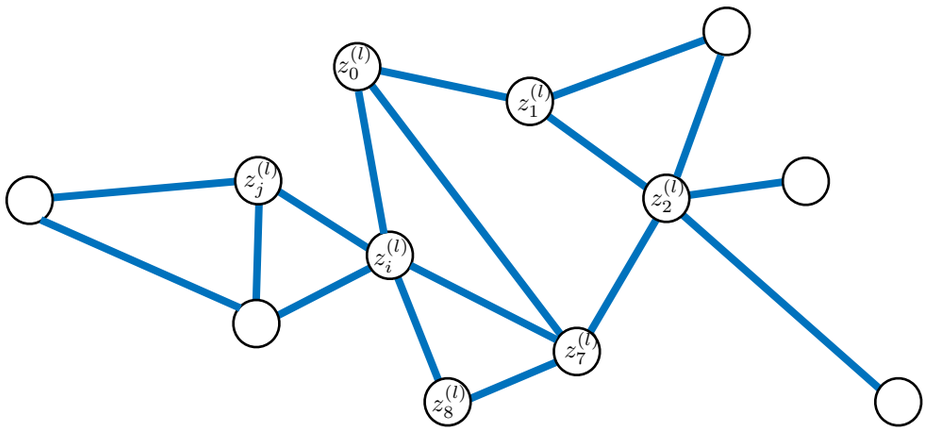
$$z_i^{(l+1)} = \sum_{i=0}^{8} \theta_i^{(l)} z_i^{(l)}$$

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



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

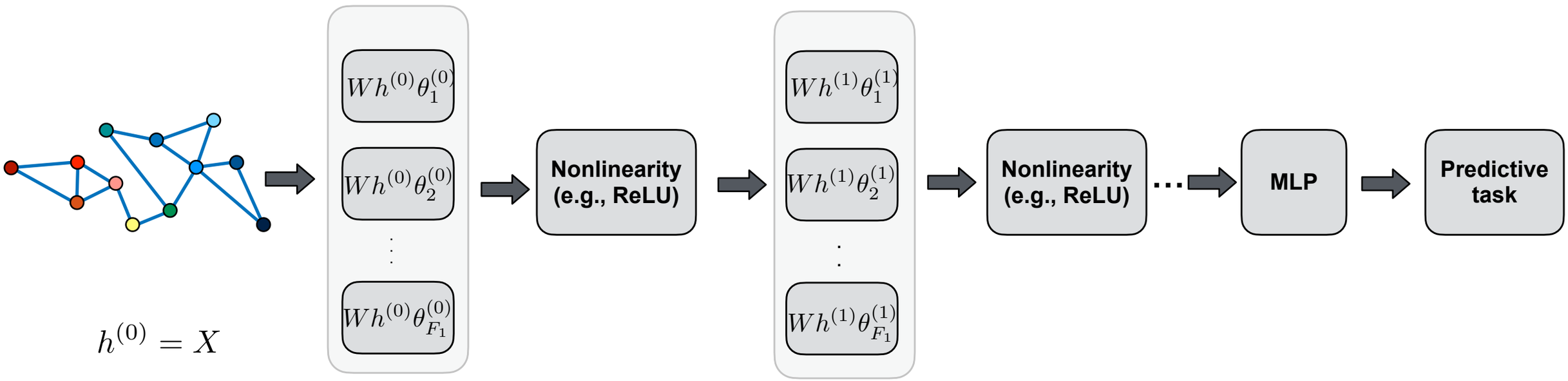
$$z_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} \theta^{(l)} z_j^{(l)}$$

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

Update embeddings by exchanging information with 1-hop neighbors

The basic GNN: a spatial viewpoint

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



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

Comparison between spatial and spectral design

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

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

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

A summary of the GNN landscape

- **Convolutional GNNs**:

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

- **Message passing GNNs**:

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

- **Attentional GNNs**:

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

Functions to be learned!

- Depending on how these functions are instantiated, different architectures are obtained; More in the following lecture!

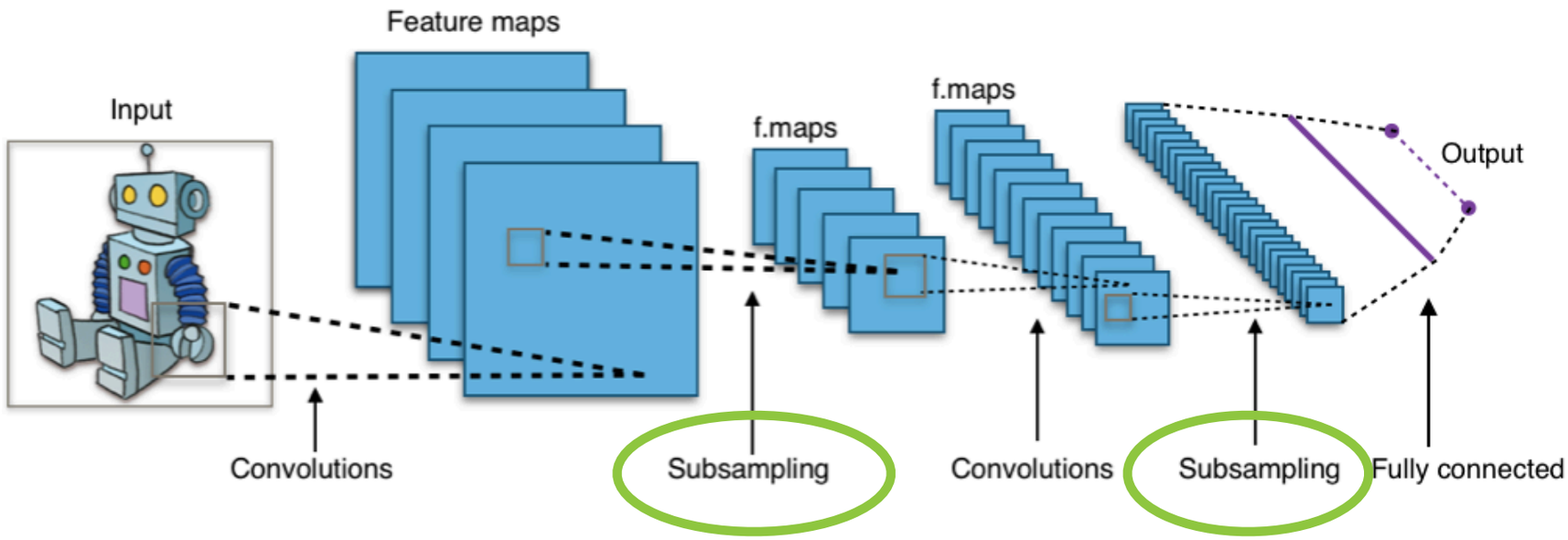
[Slide inspired from P. Veličković]

Outline

- From CNNs to GNNs

- Key building blocks of GNNs
  - Graph convolution
    - A spectral approach
    - A spatial approach
  - **Local and global pooling**
  - Loss functions

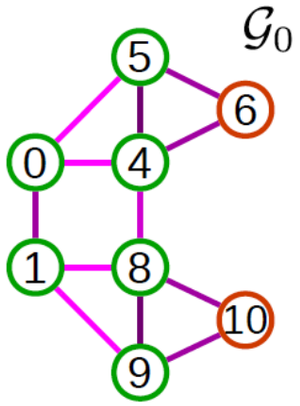
How to define pooling on graphs?



- A relatively open question, with ongoing research
- No single way to do coarsening/pooling!
- Methods can be grouped in three main categories:
  - topology based pooling
  - global pooling
  - hierarchical pooling

Topology based pooling

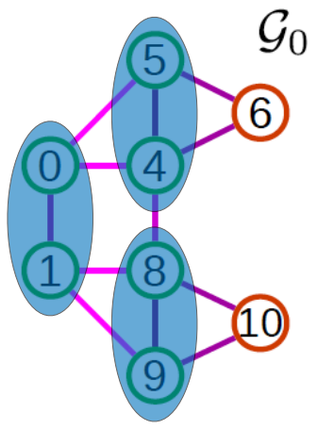
- Multi-scale graph coarsening: no features involved



- Graclus algorithm (Dhillon et al. 2007)

Topology based pooling

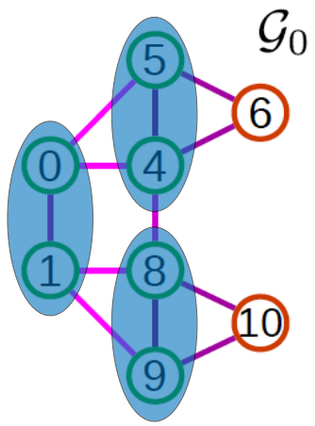
- Multi-scale graph coarsening: no features involved



- Graclus algorithm (Dhillon et al. 2007)

Topology based pooling

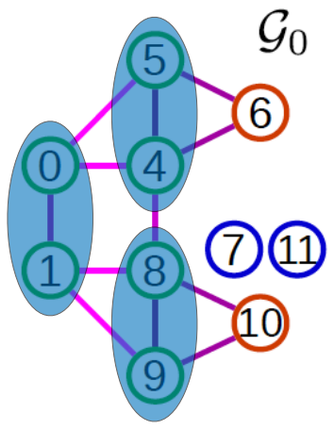
- Multi-scale graph coarsening: no features involved



- Graclus algorithm (Dhillon et al. 2007)
  - Local greedy way of merging vertices that minimises the normalised cut

Topology based pooling

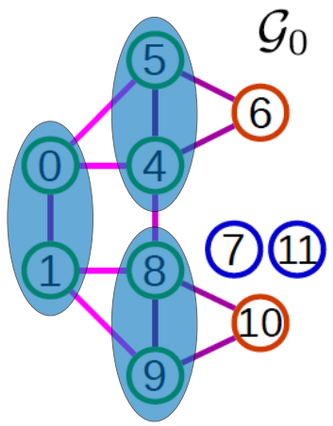
- Multi-scale graph coarsening: no features involved



- Graclus algorithm (Dhillon et al. 2007)
  - Local greedy way of merging vertices that minimises the normalised cut

Topology based pooling

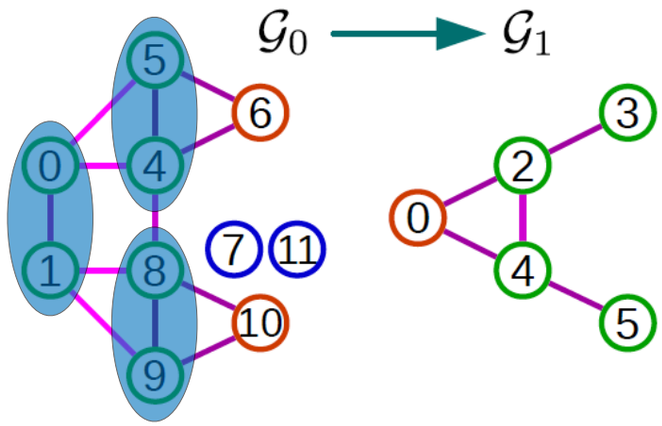
- Multi-scale graph coarsening: no features involved



- Graclus algorithm (Dhillon et al. 2007)
  - Local greedy way of merging vertices that minimises the normalised cut
  - Add artificial nodes to ensure two children for each vertex

Topology based pooling

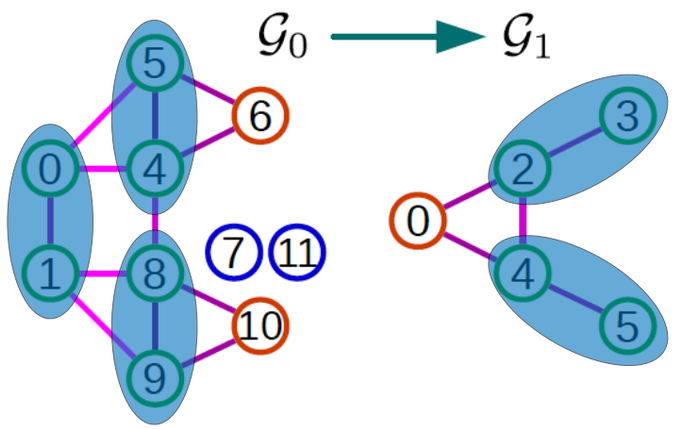
- Multi-scale graph coarsening: no features involved



- Graclus algorithm (Dhillon et al. 2007)
  - Local greedy way of merging vertices that minimises the normalised cut
  - Add artificial nodes to ensure two children for each vertex

Topology based pooling

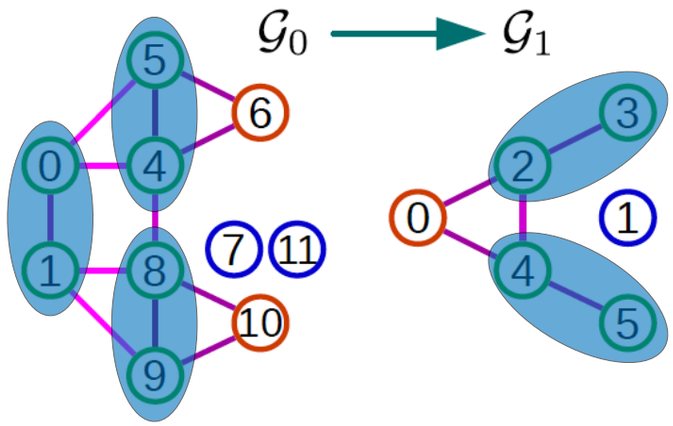
- Multi-scale graph coarsening: no features involved



- Graclus algorithm (Dhillon et al. 2007)
  - Local greedy way of merging vertices that minimises the normalised cut
  - Add artificial nodes to ensure two children for each vertex

Topology based pooling

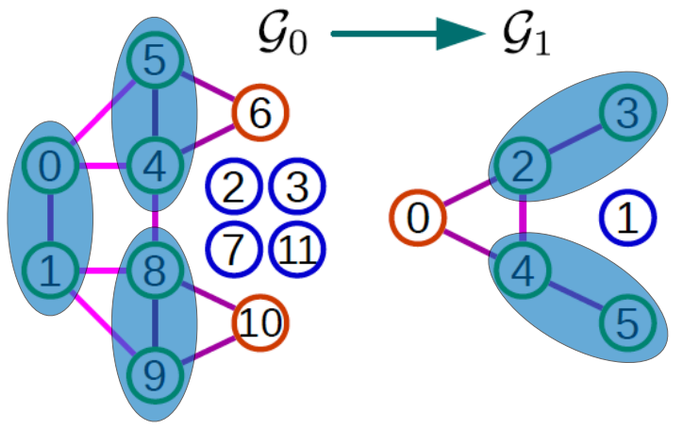
- Multi-scale graph coarsening: no features involved



- Graclus algorithm (Dhillon et al. 2007)
  - Local greedy way of merging vertices that minimises the normalised cut
  - Add artificial nodes to ensure two children for each vertex

Topology based pooling

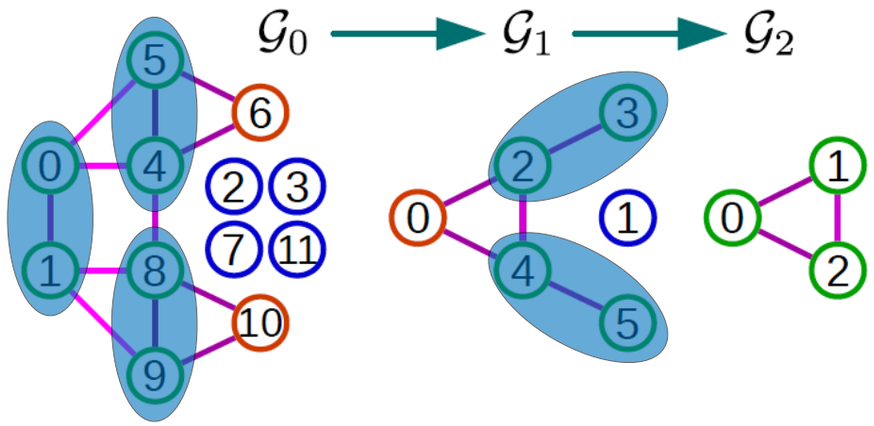
- Multi-scale graph coarsening: no features involved



- Graclus algorithm (Dhillon et al. 2007)
  - Local greedy way of merging vertices that minimises the normalised cut
  - Add artificial nodes to ensure two children for each vertex

Topology based pooling

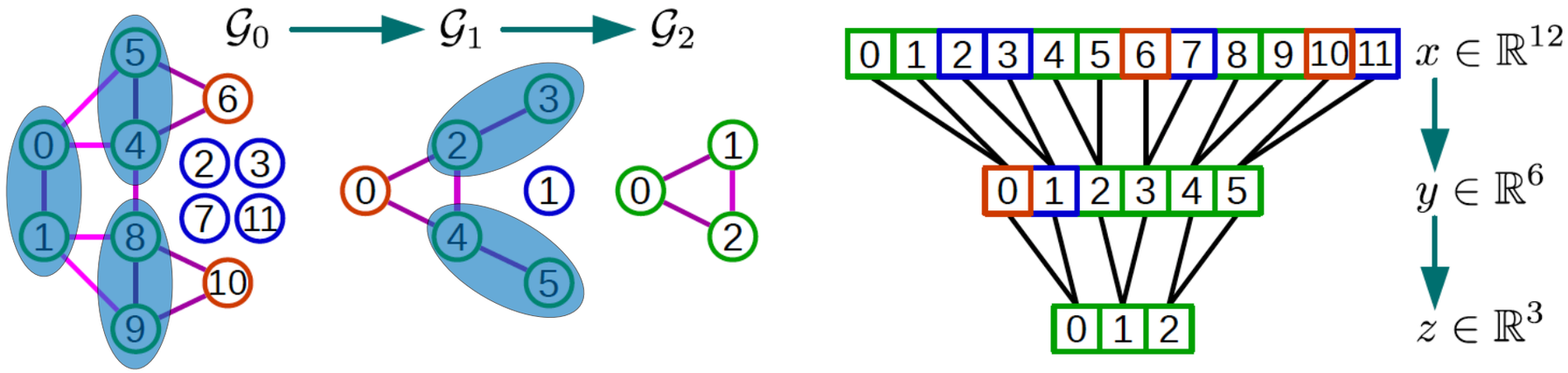
- Multi-scale graph coarsening: no features involved



- Graclus algorithm (Dhillon et al. 2007)
  - Local greedy way of merging vertices that minimises the normalised cut
  - Add artificial nodes to ensure two children for each vertex

Topology based pooling

- Multi-scale graph coarsening: no features involved



Defferrard et al. 2016

- Graclus algorithm (Dhillon et al. 2007)
  - Local greedy way of merging vertices that minimises the normalised cut
  - Add artificial nodes to ensure two children for each vertex
  - 1D grid pooling: [max(0,1), max(4,5,6), max(8,9,10)]

Global pooling

- Involves node features
- Uses sum/max or neural networks to pool all representation of nodes

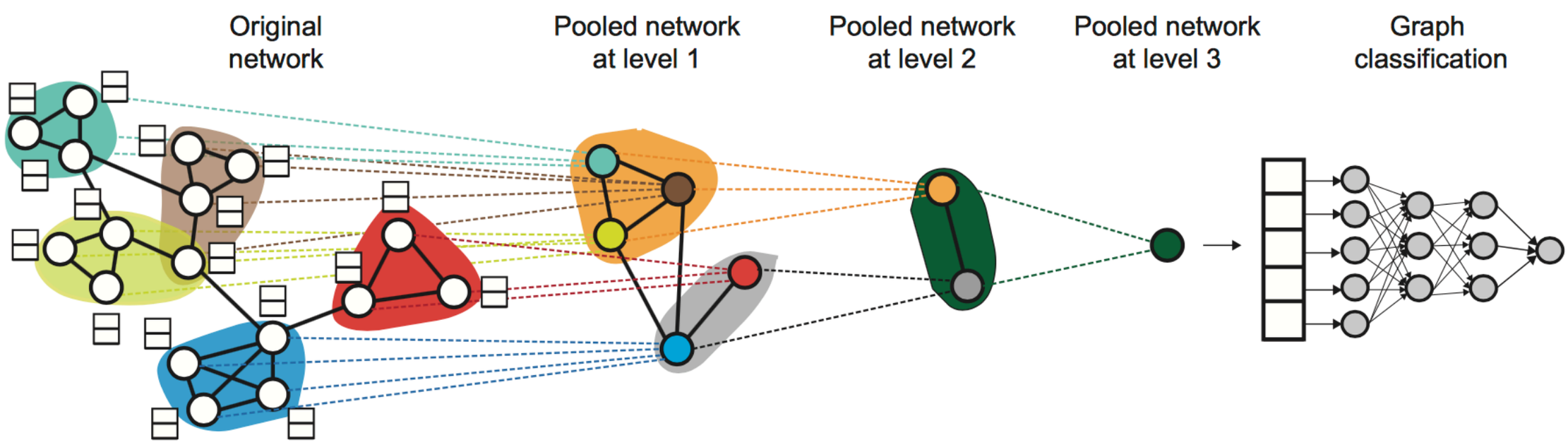
$$h_G = mean/max/sum(h_1^{(K)}, h_2^{(K)}, ..., h_N^{(K)})$$

- Also known as READOUT

- Example: SortPool (Zhang et al. 2018)
  - sorts embeddings for nodes according to the structural roles of a graph

Hierarchical pooling

- Involves nodes features
- Aggregates information in a hierarchical way that respects the graph structure
- Results in learning clusters
- Example: DiffPool [Ying et al. 2019]



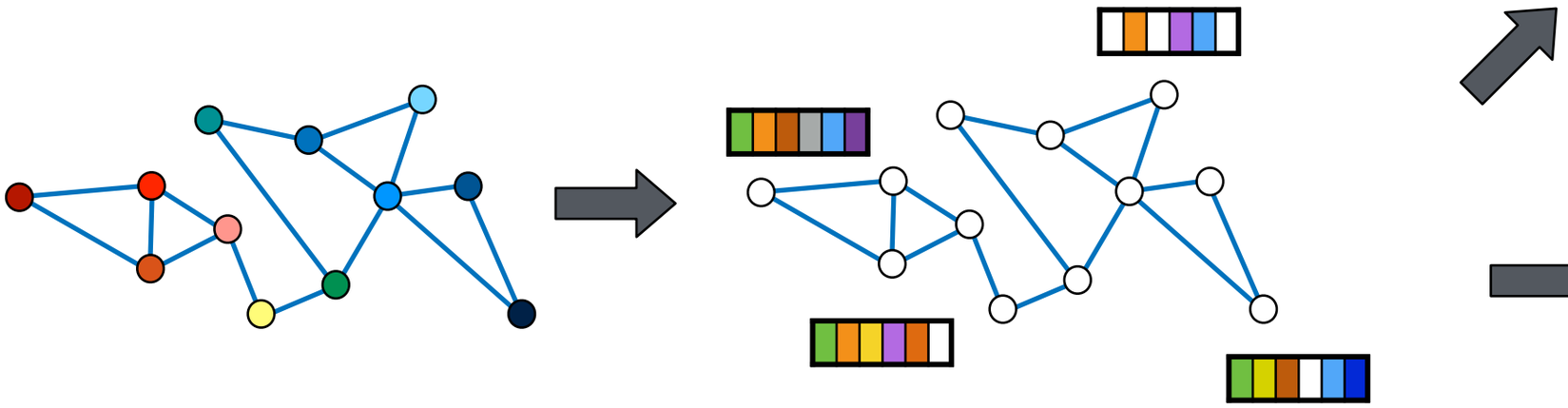
Outline

- From CNNs to GNNs

- Key building blocks of GNNs
  - Graph convolution
    - A spectral approach
    - A spatial approach
  - Local and global pooling
  - **Loss functions**

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

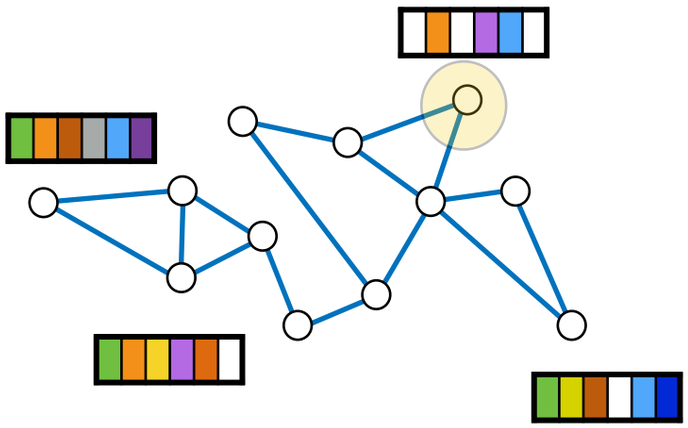
$$\tilde{y}_{\mathcal{G}} = f(\bigoplus_{i \in \mathcal{V}} h_i)$$

Link prediction

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

Loss function: Node level task

- Given labels $Y = [y_1, y_2, ..., y_N]$ on each node, predict labels $\tilde{Y} = [\tilde{y}_1, \tilde{y}_2, ..., \tilde{y}_N]$ by minimizing the following loss function:

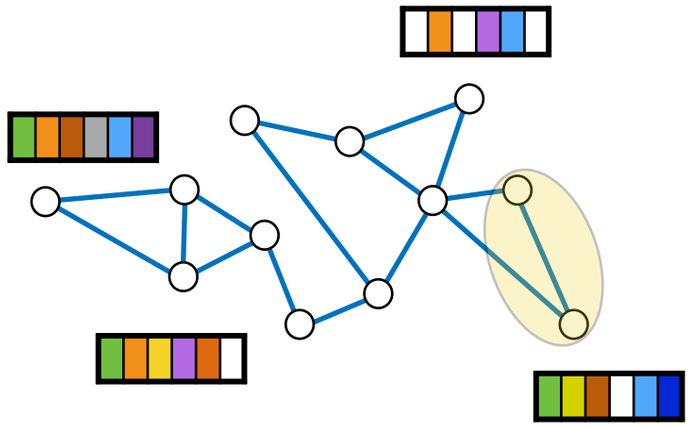


$$loss = \sum_{i \in \mathcal{V}} loss_i(y_i - \tilde{y}_i)$$

- Example of a loss function per node: Cross entropy loss

Loss function: Edge level task

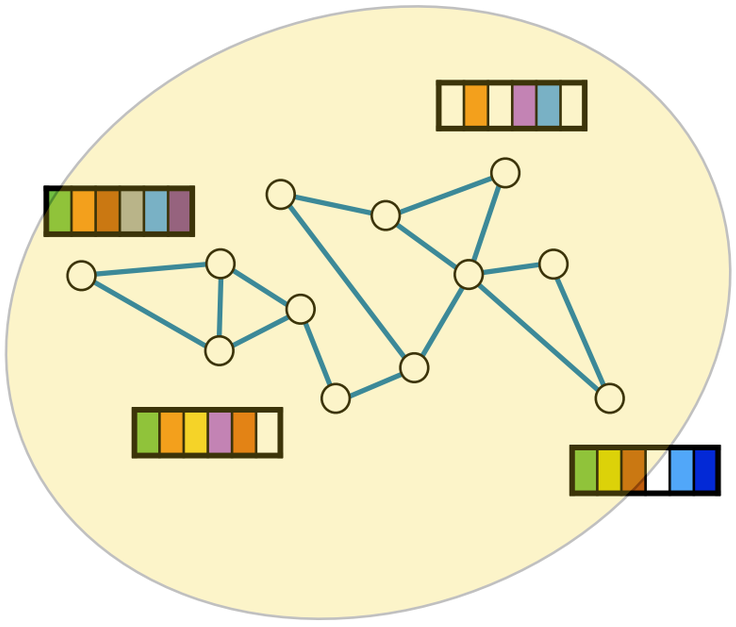
- Given labels $Y = [y_{1,2}, y_{2,3}, ..., y_{N,N-1}]$ on each node, predict labels $\tilde{Y} = [\tilde{y}_{1,2}, \tilde{y}_{2,3}, ..., \tilde{y}_{N,N-1}]$ by minimizing the following loss function:



$$loss = \sum_{i,j \in \mathcal{E}} loss_i(y_{i,j} - \tilde{y}_{i,j})$$

Loss function: Graph level task

- Given a set of M graphs with labels $Y = [y_1, y_2, ..., y_M]$ on each graph, predict labels $\tilde{Y} = [\tilde{y}_1, \tilde{y}_2, ..., \tilde{y}_M]$ by minimizing the following loss function:



$$loss = \sum_{m=1}^{M} loss_m(y_m - \tilde{y}_m)$$

$$\tilde{y}_m = f_{readout}(\{h_{m,i}\}_i, \forall i \in \mathcal{V})$$

Some practical aspects

- Data pre-processing: Normalization of the attributes is important
- Optimizer: ADAM is often a good starting point
- Learning rate: Probably the most important hyperparameters
- Activation function: ReLU often works well (other options are possible e.g., LeakyReLU)
- Batch size: Should be a good compromise between performance and size. Smaller batches are usually better, and 32 is a good starting point
- Embedding dimensions: 32, 64, 128 are often good starting points
- Start always with shallow models: deep models may be harder to optimize, and do not necessarily improve the performance

Summary

- Graph neural networks are designed based on a generalization of deep learning techniques on graphs

- Their key ingredients are:
  - Parameter sharing across nodes (permutation invariance)
  - Local exchange of information and neighbourhood aggregation on the graph (e.g., graph convolution)
  - Graph subsampling/pooling (mainly used for graph level tasks)

- GNNs provide ways of computing node/edge/graph embeddings by exploiting structure (i.e., graphs) and node attributes

- We discuss state-of-the-art architectures in the next lecture!

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