CSDS503 / COMP552 – Advanced Machine Learning

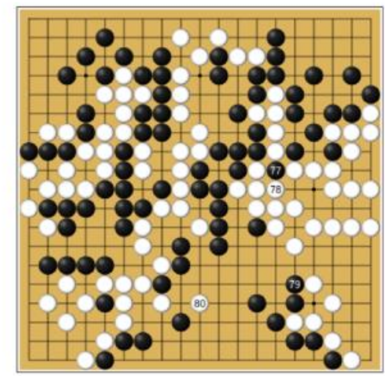
Faizad Ullah

Traditional Neural Networks

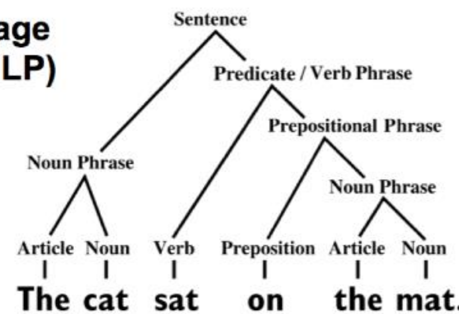
IMAGENET

Speech data

Grid games



Natural language processing (NLP)



...

Deep neural nets that exploit:

- translation equivariance (weight sharing)
- hierarchical compositionality

* slide from Thomas Kipf, **University of Amsterdam**

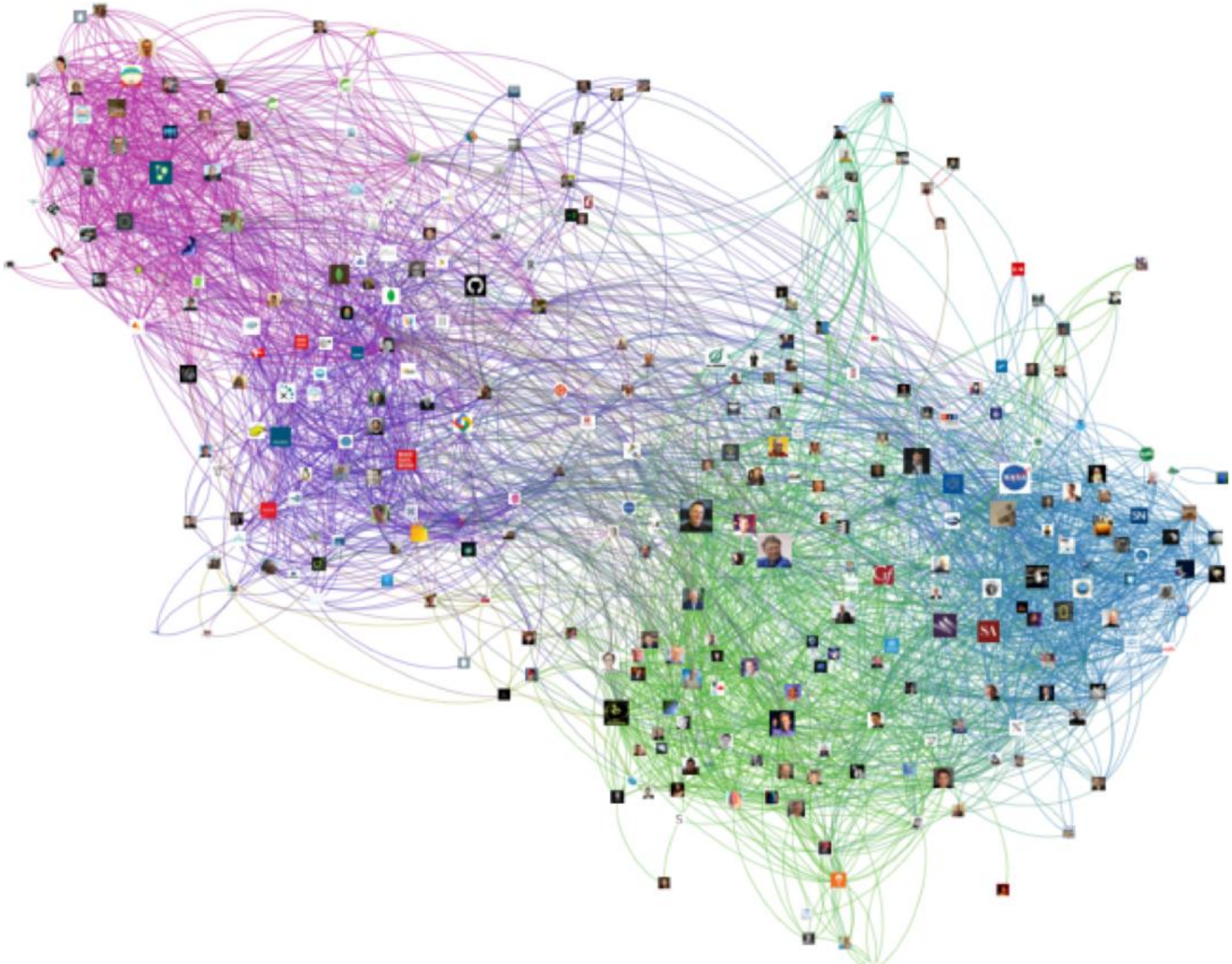
Graph-structured Data

- A lot of real-world data does not “live” on grids

Social Networks

Standard **CNN** and **RNN** architectures don’t work on this data

Twitter followers graph

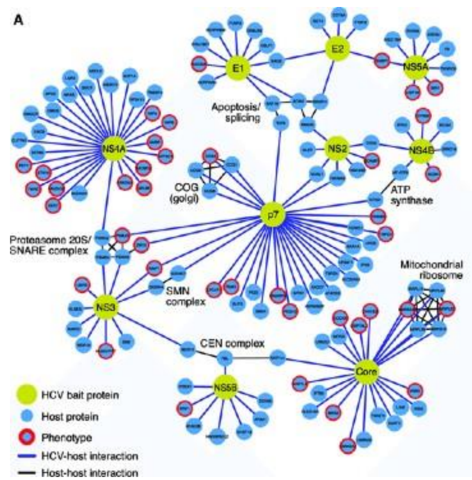


http://allthingsgraphed.com/2014/11/02/twitter-friends-network/

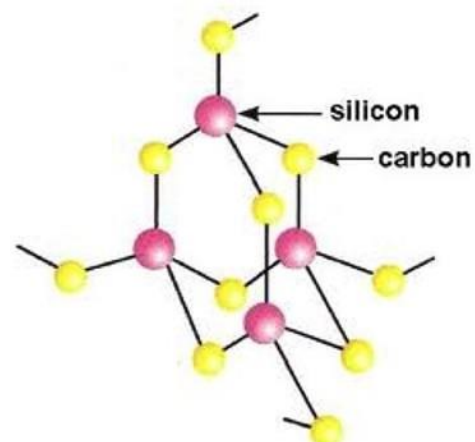
Graph-structured Data

- A lot of real-world data does not “live” on grids

Social Networks



Protein Networks

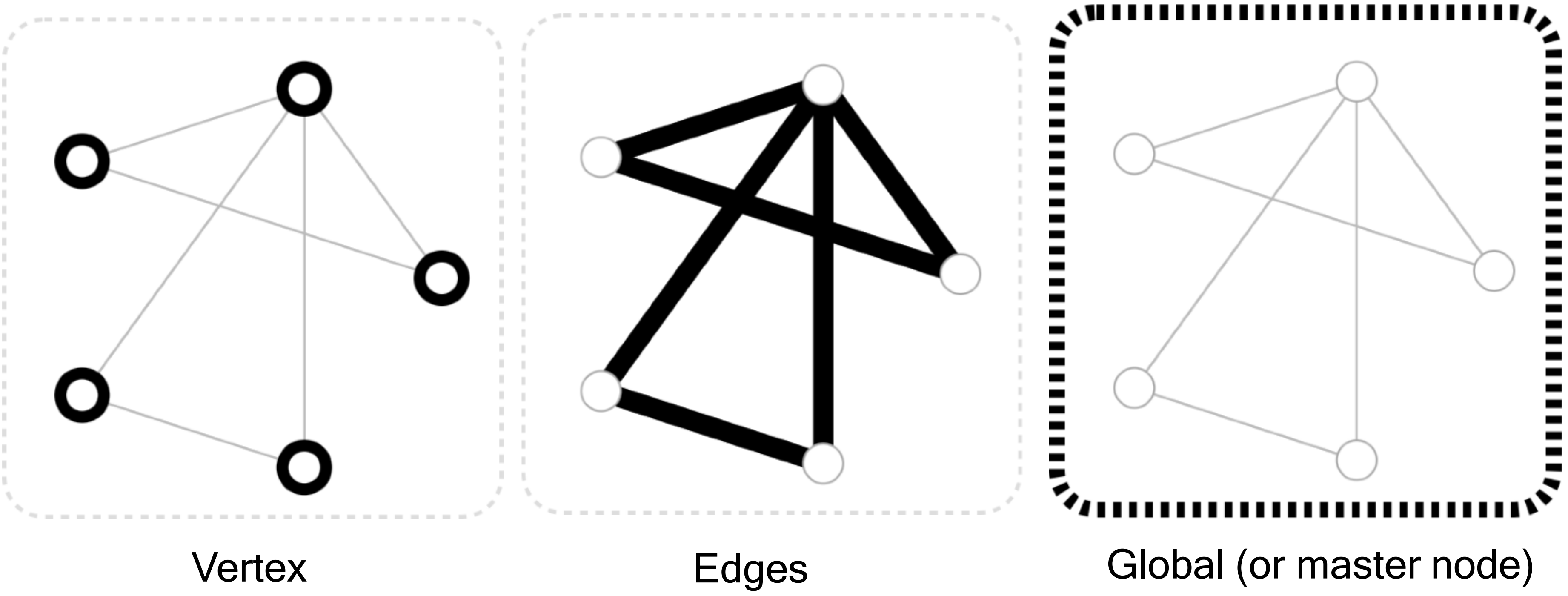


Molecules

Standard **CNN** and **RNN** architectures don’t work on this data

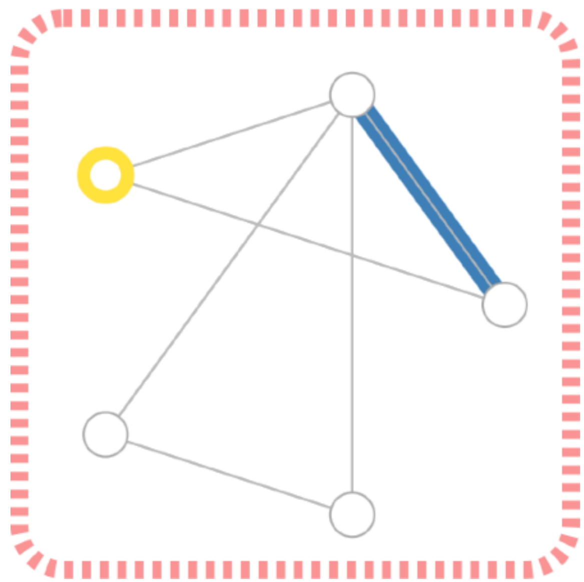
Graph

- A set of objects, and the connections between them, are naturally expressed as a *graph*.
- A graph represents the relations (*edges*) between a collection of entities (*nodes*).



Graph

A graph is a pair $G = (V, E)$, where $V = \{1, \cdots, n\}$is a set of *n* vertices (nodes), and $E = \{(i, j) \,|\, i, j \in N\}$ is a set of edges between them.



Graph

- A graph is a pair $G = (V, E)$, where $V = \{1, \cdots, n\}$is a set of *n* vertices (nodes), and $E = \{(i, j) \,|\, i, j \in N\}$ is a set of edges between them.

- Alternatively, we can define an adjacency matrix A where:

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

- If $A^T = A$, we say the graph is undirected.

Images as graphs

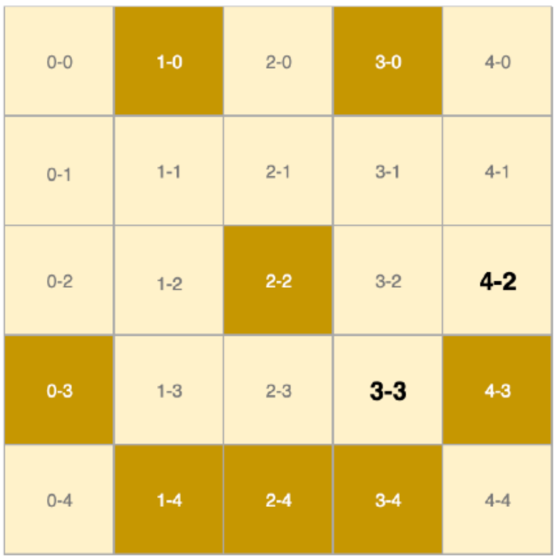
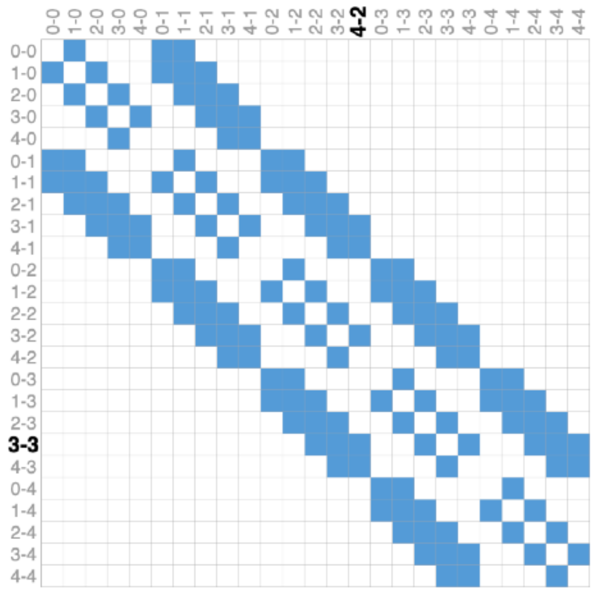
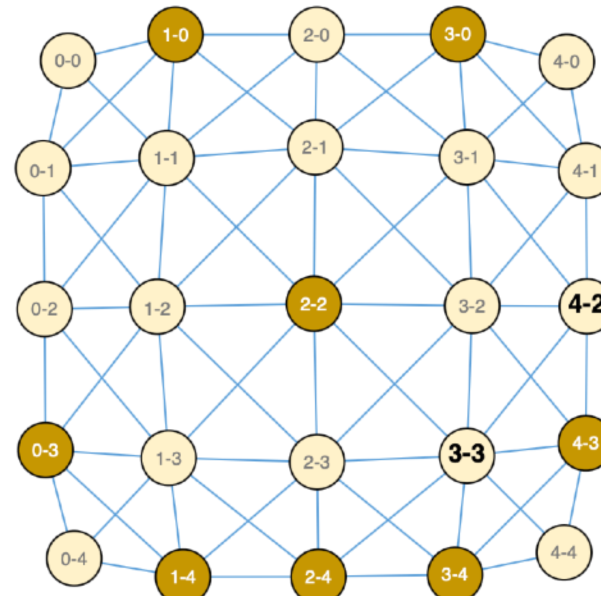


Image Pixels



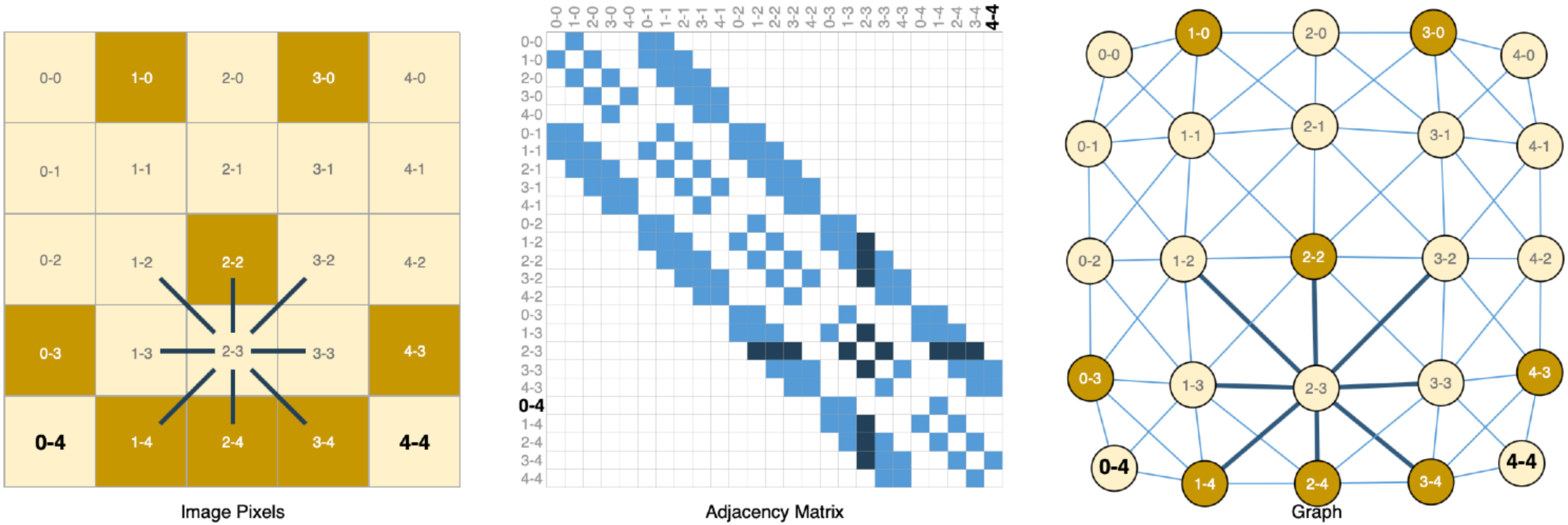
Adjacency Matrix



Graph

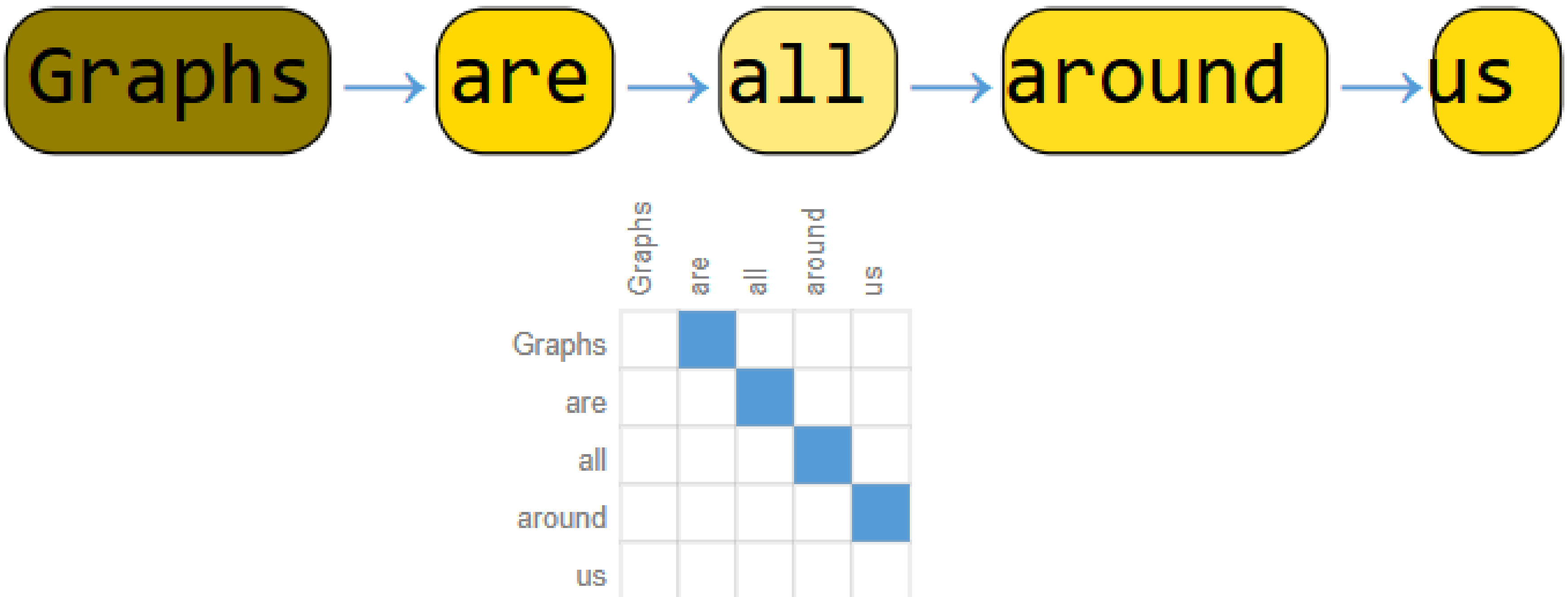
https://distill.pub/2021/gnn-intro/

Images as graphs

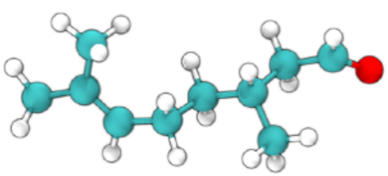


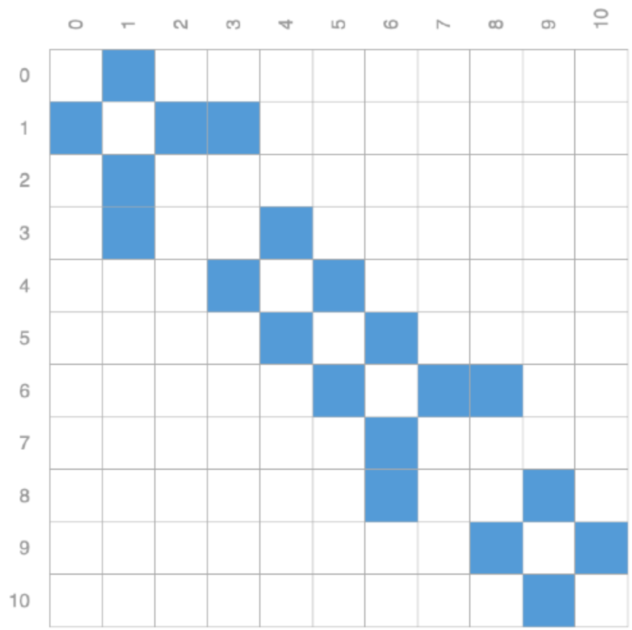
https://distill.pub/2021/gnn-intro/

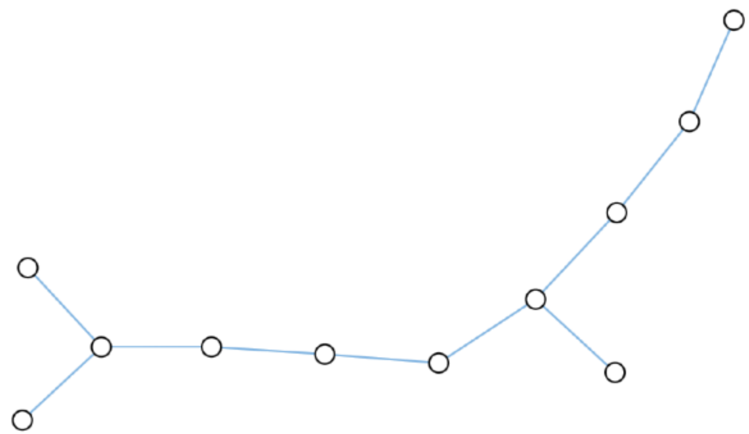
Text as graphs



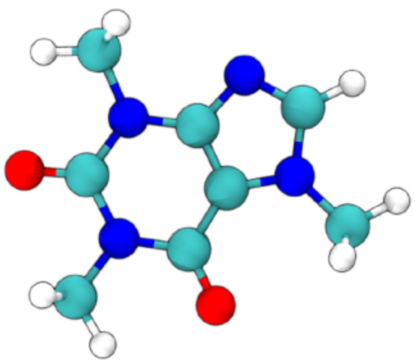
Molecules as graphs

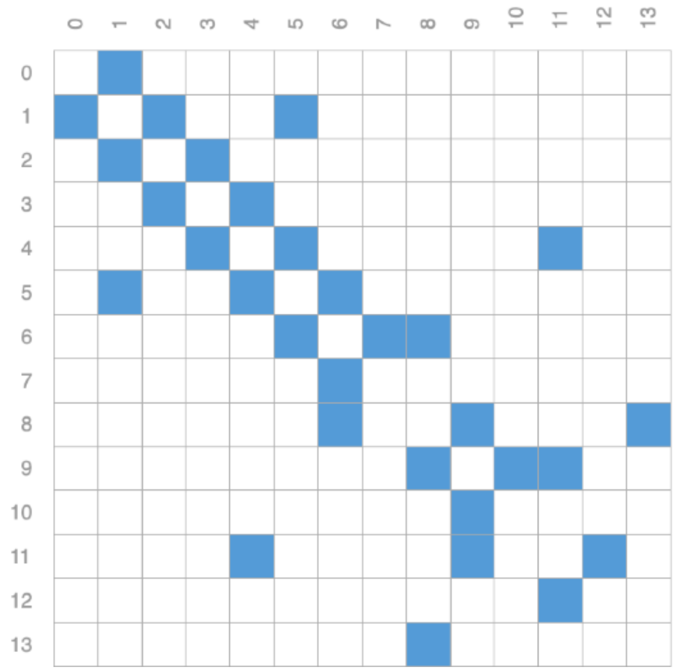


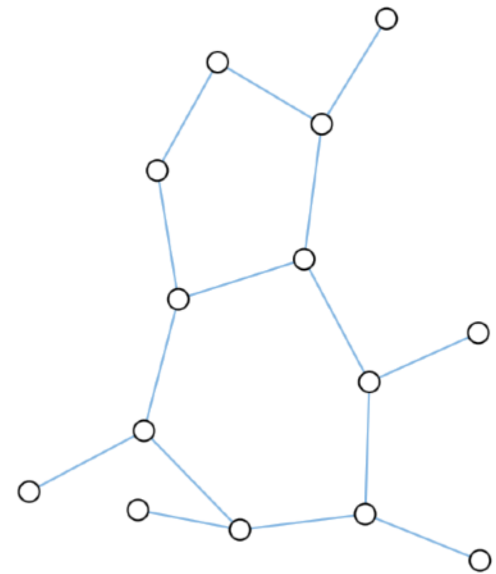




Molecules as graphs

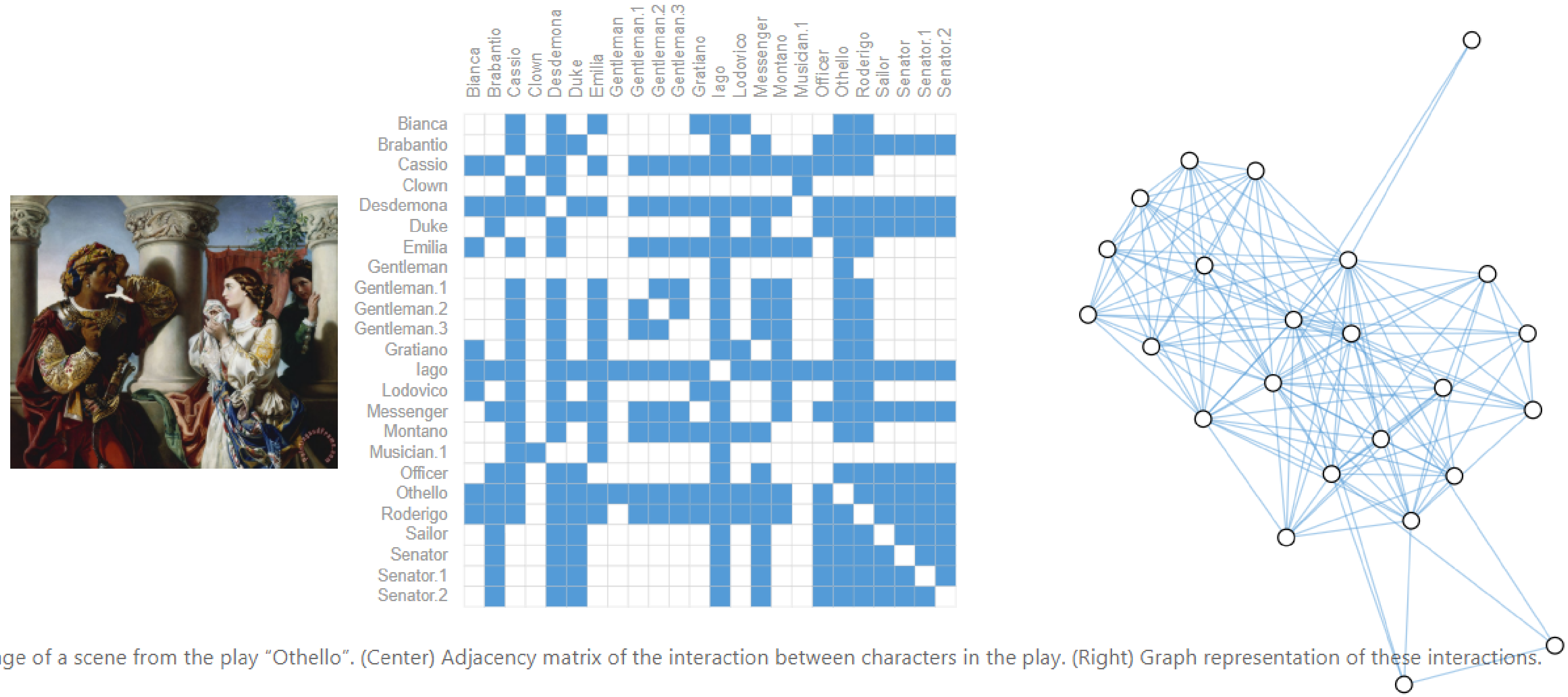






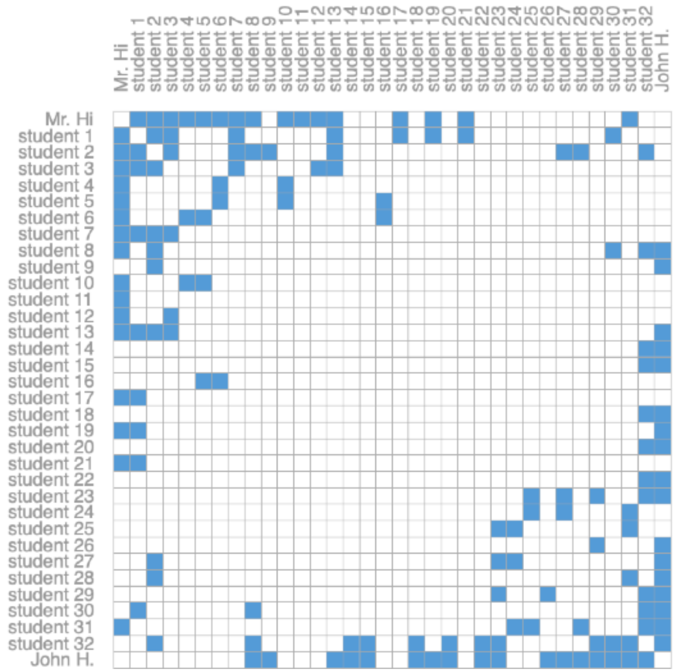
https://distill.pub/2021/gnn-intro/

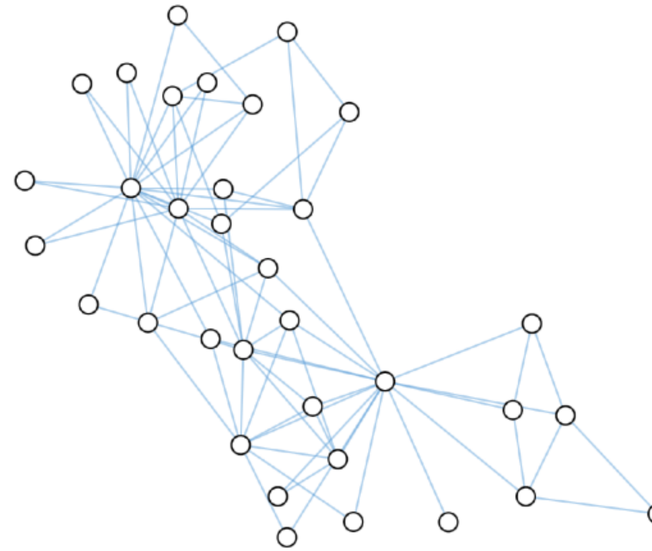
Social networks as graphs



(Left) Image of a scene from the play “Othello”. (Center) Adjacency matrix of the interaction between characters in the play. (Right) Graph representation of these interactions.

Social Network as Graph

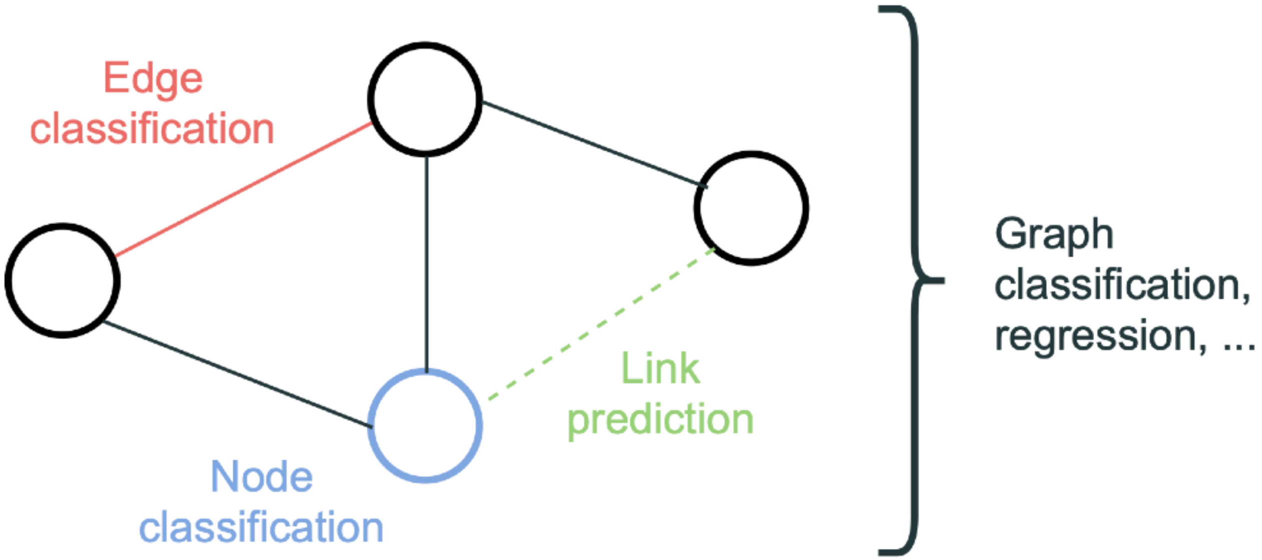




https://distill.pub/2021/gnn-intro/

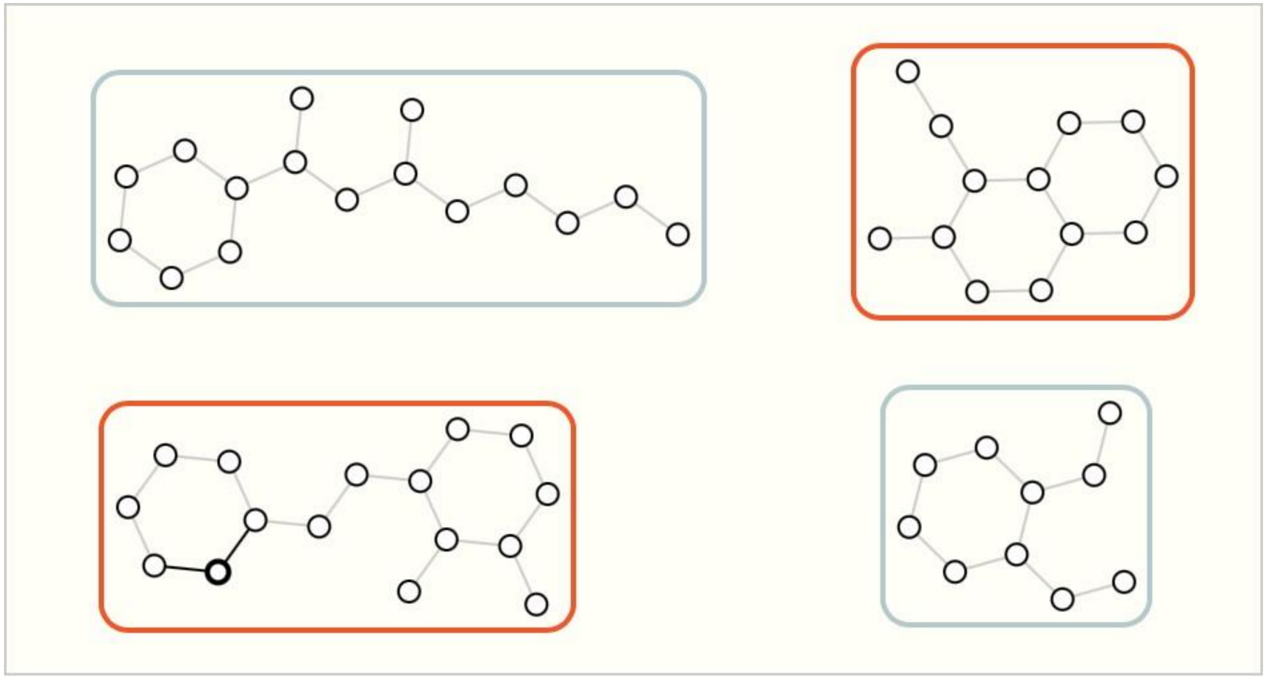
Learning Problems on Graphs

Learning Problems on Graphs



https://www.sscardapane.it/assets/files/nnds2021/Lecture_6_graphs.pdf

Graph-level task

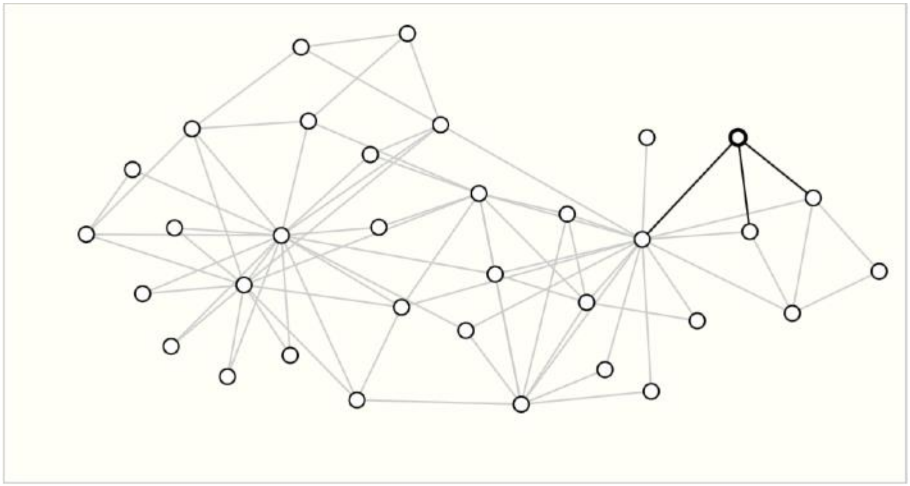


Output: labels for each graph, (e.g., "does the graph contain two rings?")

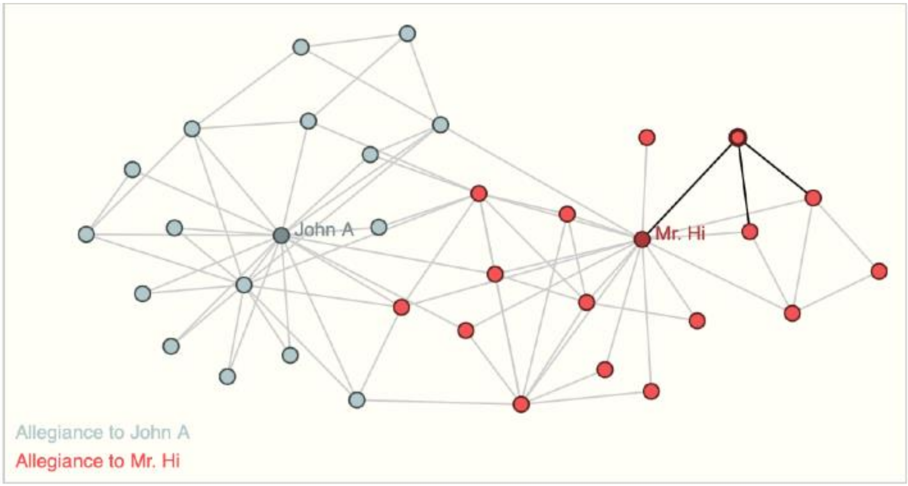
This is analogous to image classification problems with MNIST and CIFAR, where we want to associate a label to an entire image. With text, a similar problem is sentiment analysis where we want to identify the mood or emotion of an entire sentence at once.

Node-level task

- Node-level tasks are concerned with predicting the identity or role of each node within a graph.
- The prediction problem is to classify whether a given member becomes loyal to either Mr. Hi or John H.

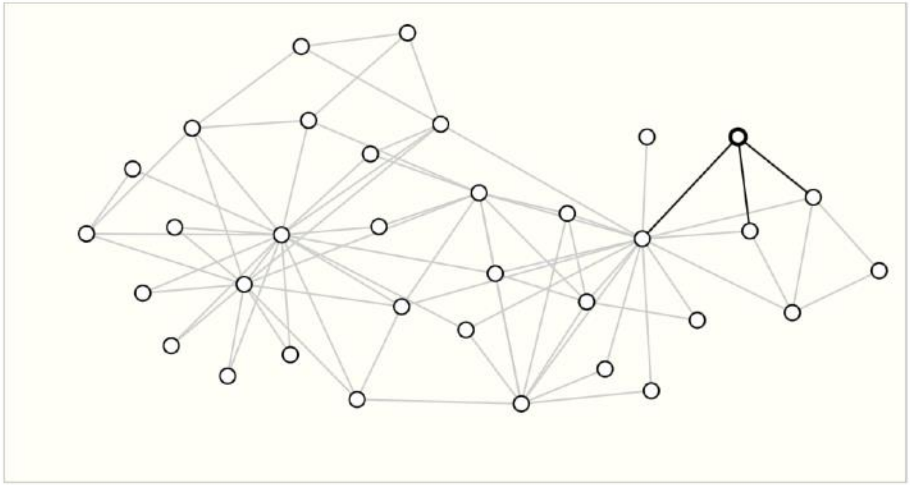


→

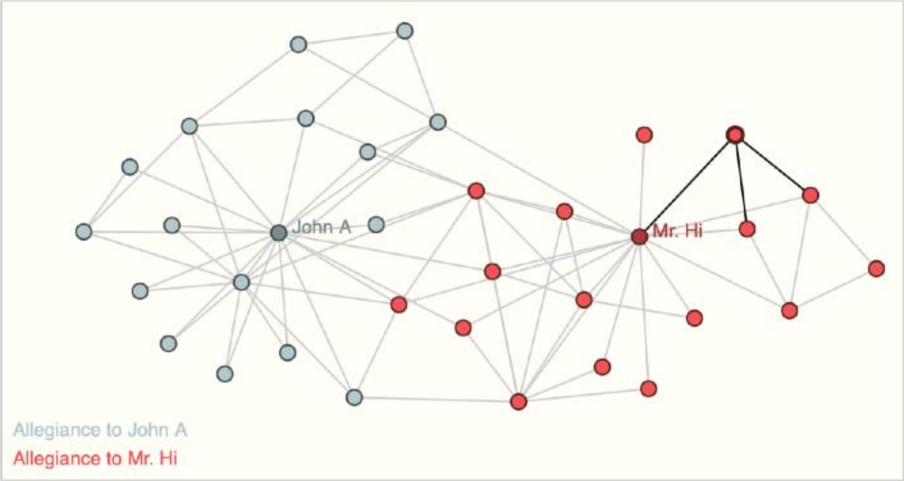


Input: graph with unlabled nodes

Output: graph node labels



→

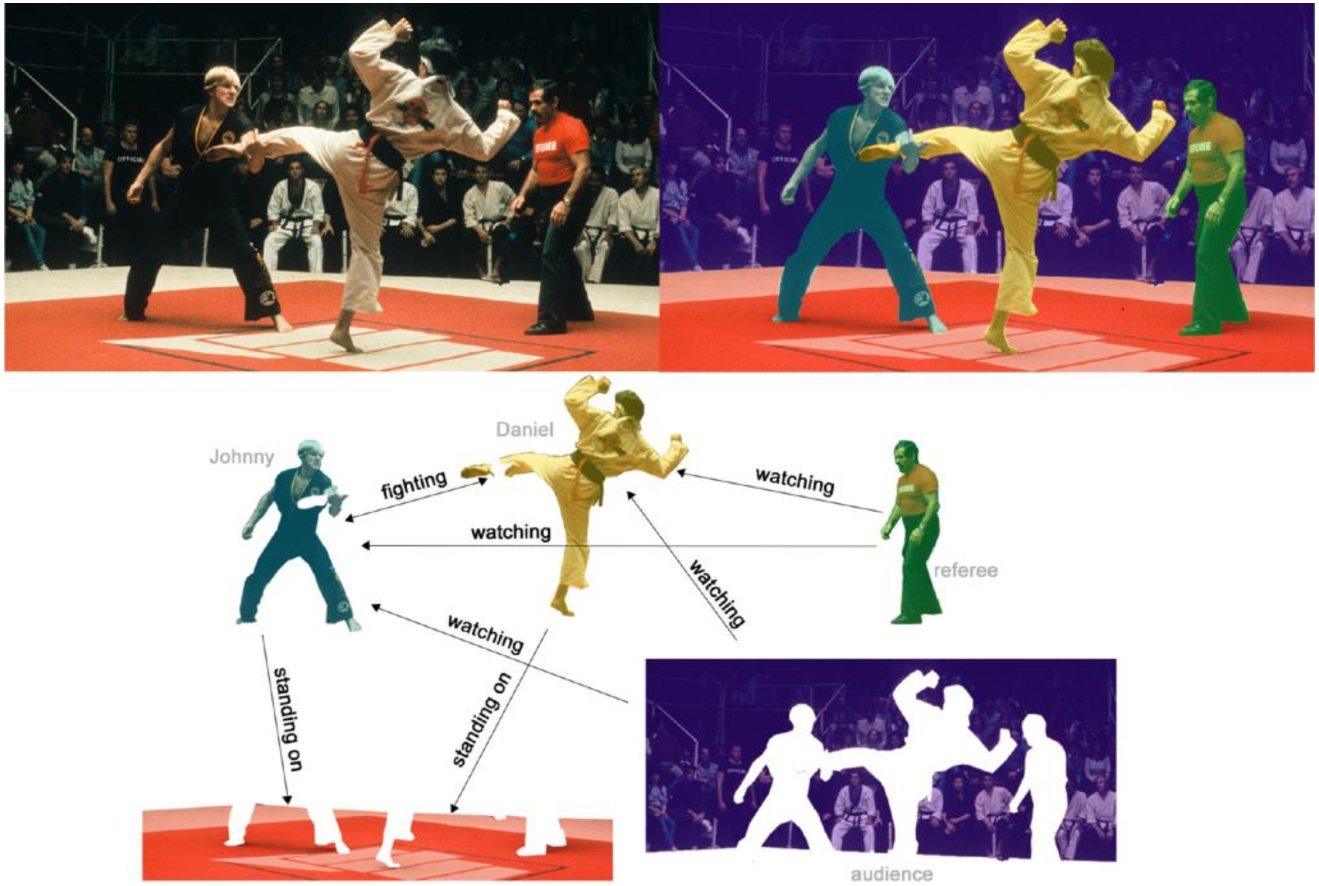


Input: graph with unlabled nodes

Output: graph node labels

Edge-level task

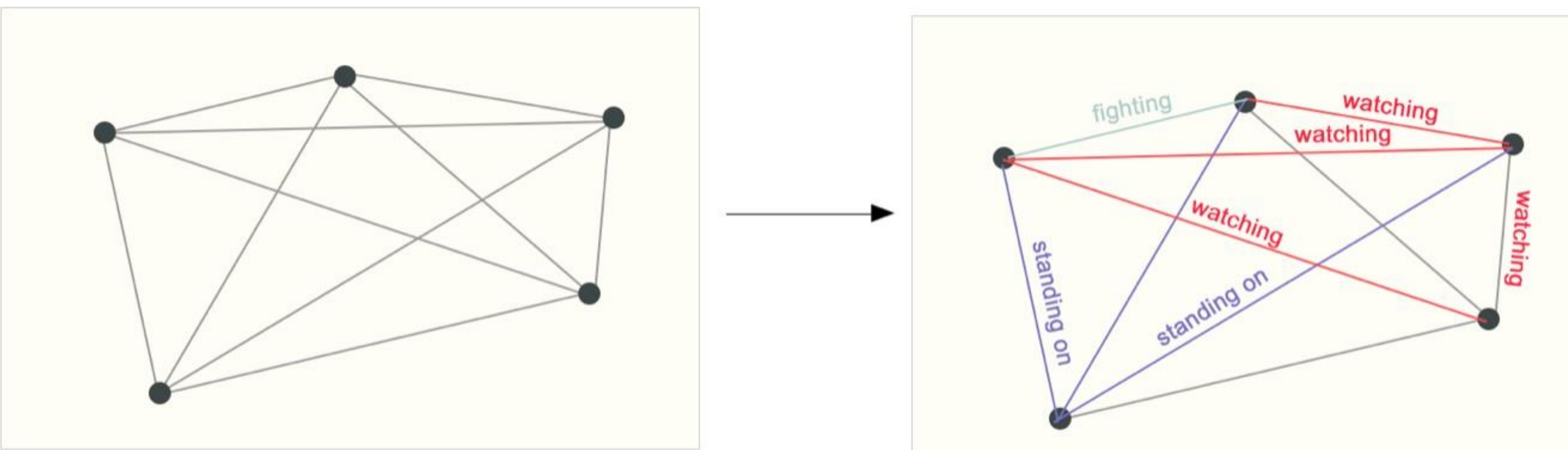
- Beyond identifying objects in an image, deep learning models can be used to predict the relationship between them.



https://distill.pub/2021/gnn-intro/

Edge-level task

- Beyond identifying objects in an image, deep learning models can be used to predict the relationship between them.



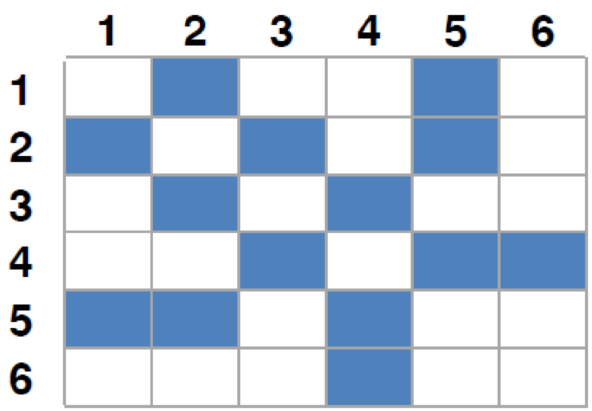
Input: fully connected graph, unlabeled edges **Output:** labels for edges

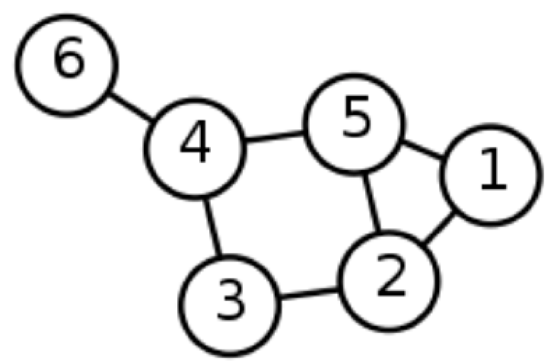
https://distill.pub/2021/gnn-intro/

Graph: Neighbours and degrees

- The neighborhood $\mathcal{N}_i$ of a node i is defined as the set of nodes sharing an edge with the node:

$$\mathcal{N}_i = \{j \,|\, A_{ij} = 1\}\,.$$



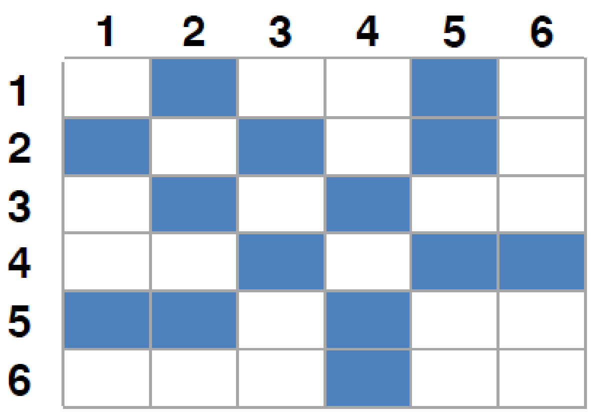


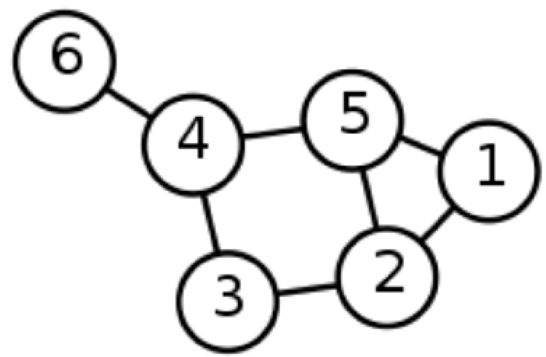
$$\mathcal{N}_4 = \{3,\, 5,\, 6\}\,.$$

Graph: Neighbours and degrees

- The size $|\mathcal{N}_i|$ of the neighborhood is called the degree of the node. The degree matrix D is a diagonal matrix containing the degrees:

$$[D]_{i,i} = \sum_j A_{ij}\,.$$

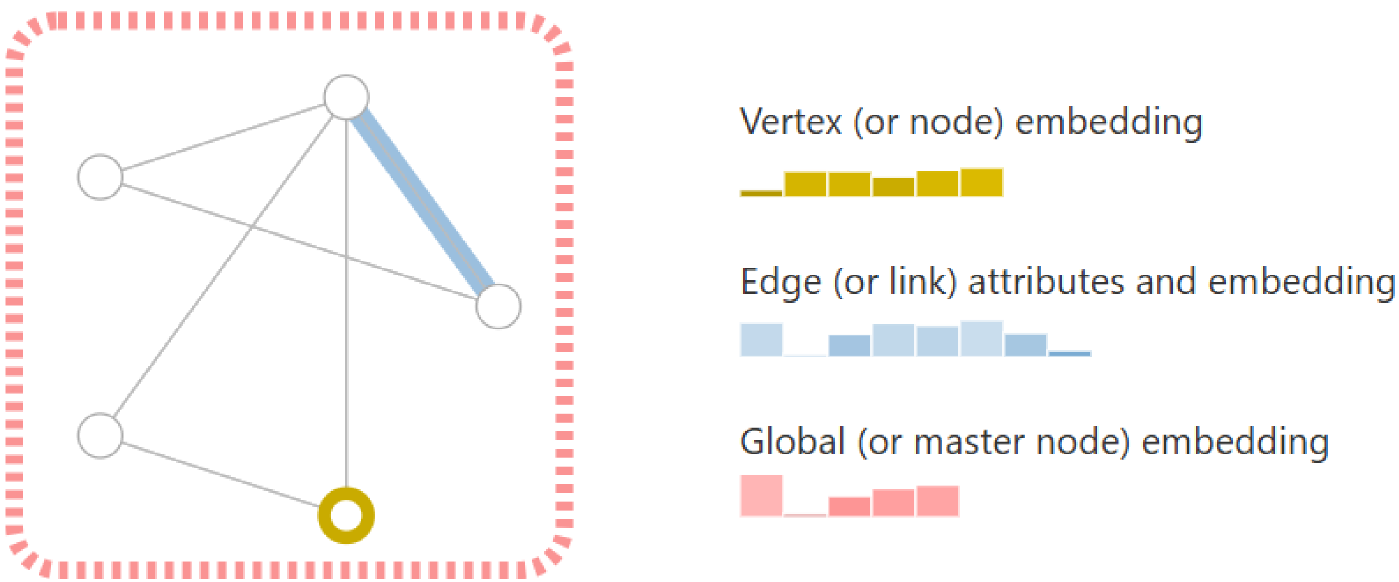




$$[D]_{4,4} = \sum_j A_{4j} = 3$$

Features on a graph

- Depending on the application, we can have node features x_i, edge features $\mathbf{e}_{ij}$, or graph features.



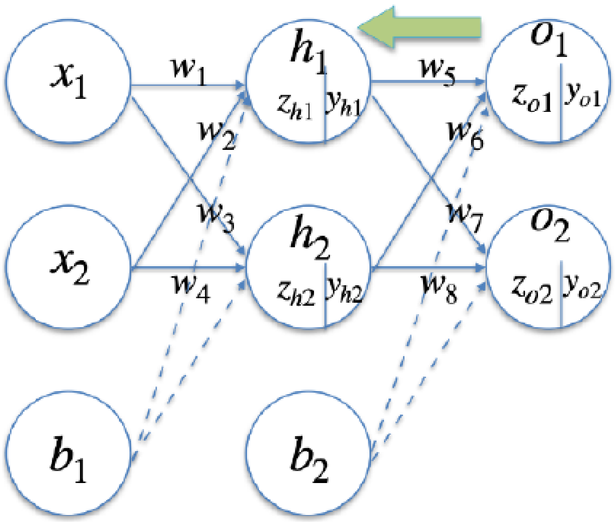
Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right).

- In the simplest case, we associate to each node a vector$_i$ of features, from which we can build a matrix

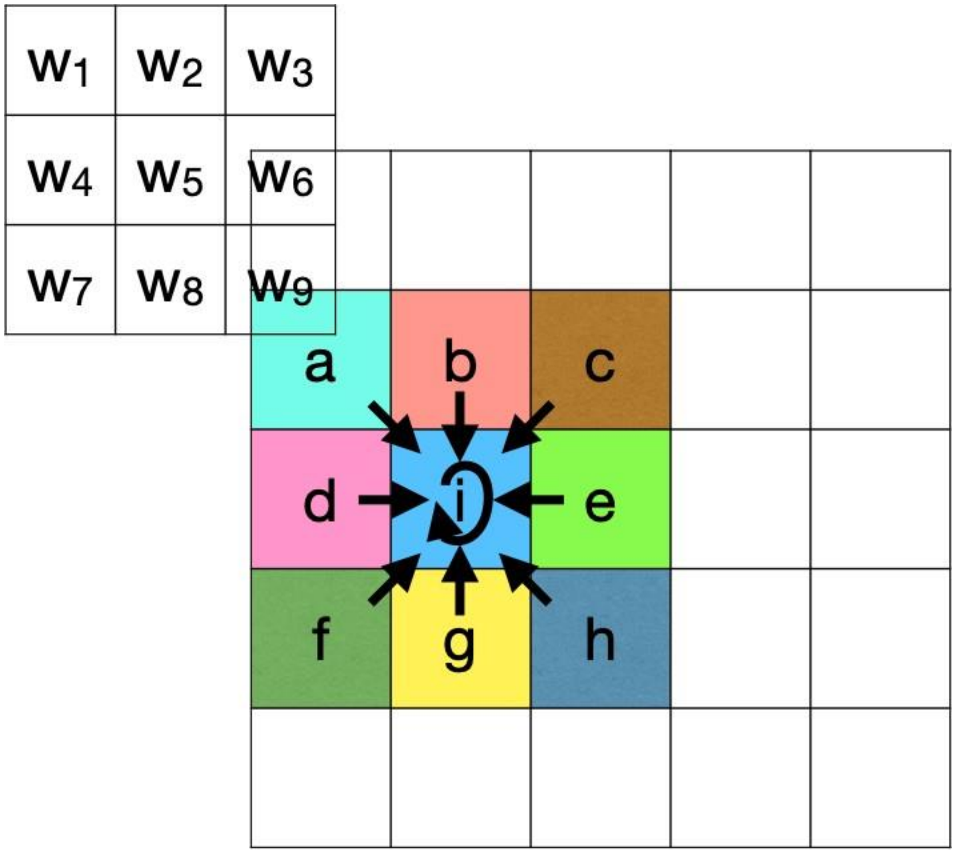
Extending Convolutions to Graphs

Recap: MLP

- $\mathbf{x} = \{1, x_1, x_2, \cdots, \}^T$
- $\mathbf{w} = \{w_0, w_1, w_2, \cdots, \}^T$
- $y_{h1} = \mathbf{w}^T\mathbf{x} = \sum_j w_j x_j = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$
- $h_1^l = z = \sigma(y_{h1})$
- $h_1^{l+1} = \sigma(w_0^l + w_1^l h_1^l + w_2^l h_2^l + \cdots + w_n^l h_n^l)$

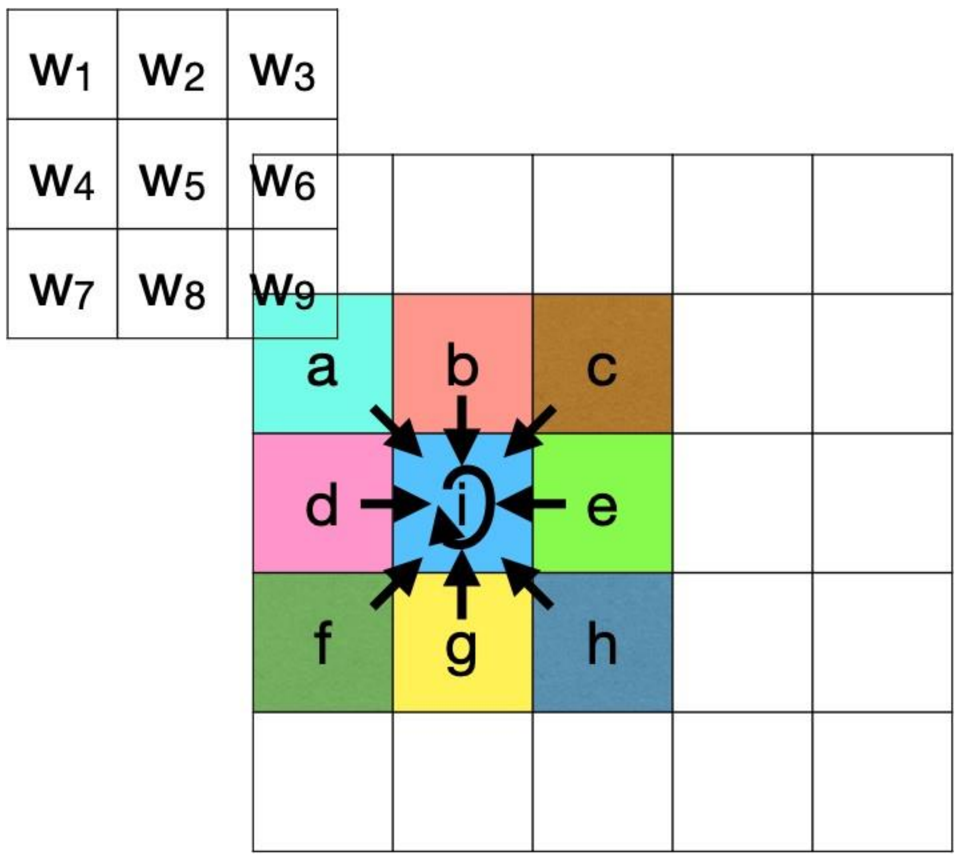


Convolution as Message Passing

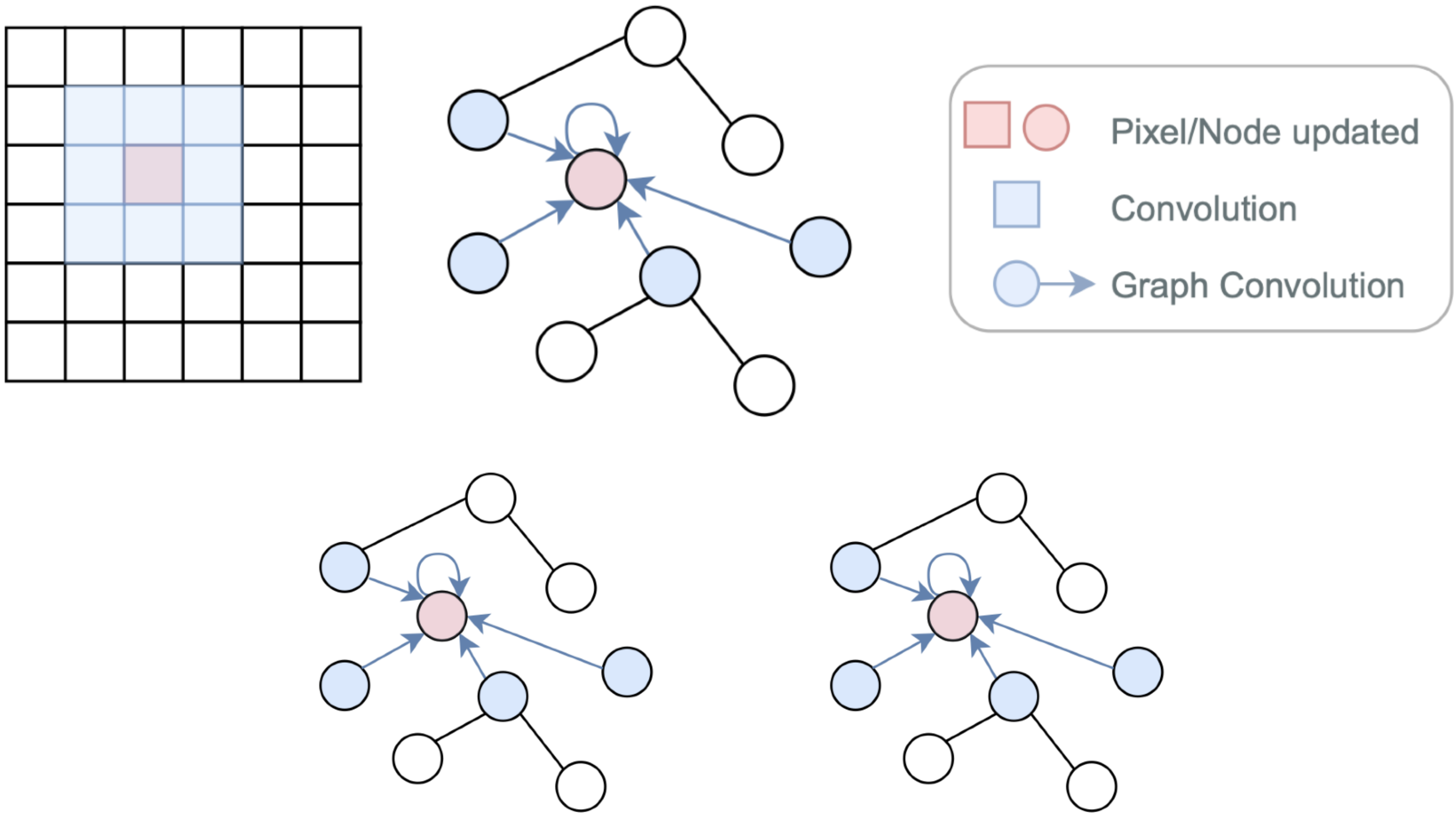


Convolution as Message Passing

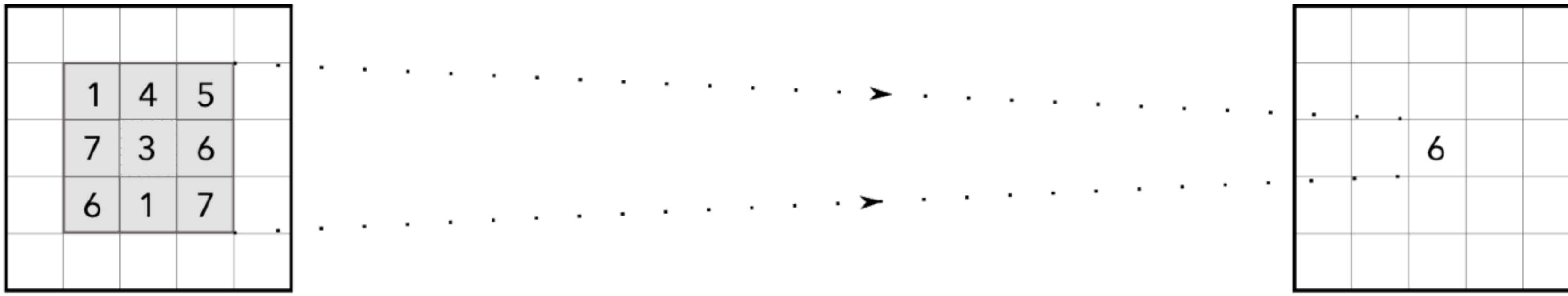
$$\mathbf{h}_i^{l+1} = w_1^l \mathbf{h}_a^l + w_2^l \mathbf{h}_b^l + \cdots + w_5^l \mathbf{h}_i^l + \cdots + w_9^l \mathbf{h}_h^l$$



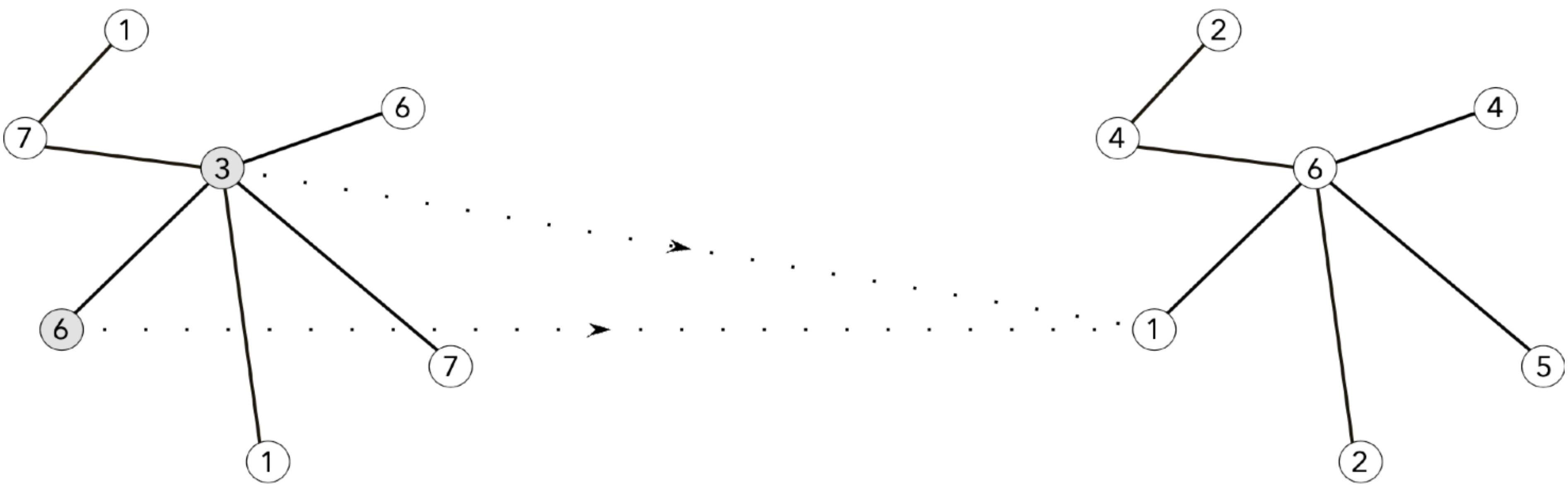
Extending Convolutions to Graphs



Extending Convolutions to Graphs

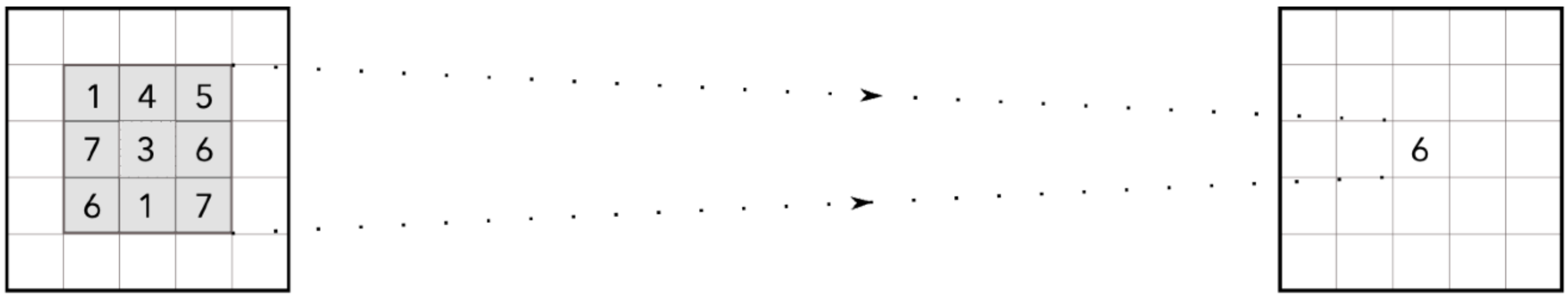


Convolution in CNNs

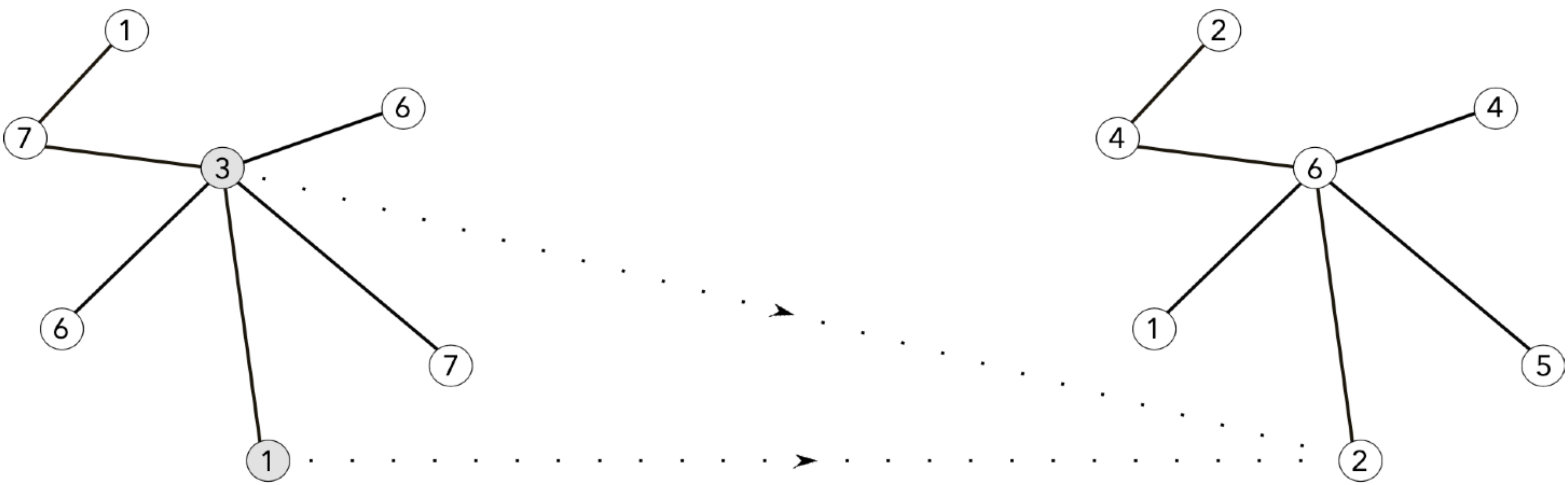


Localized Convolution in GNNs

Extending Convolutions to Graphs

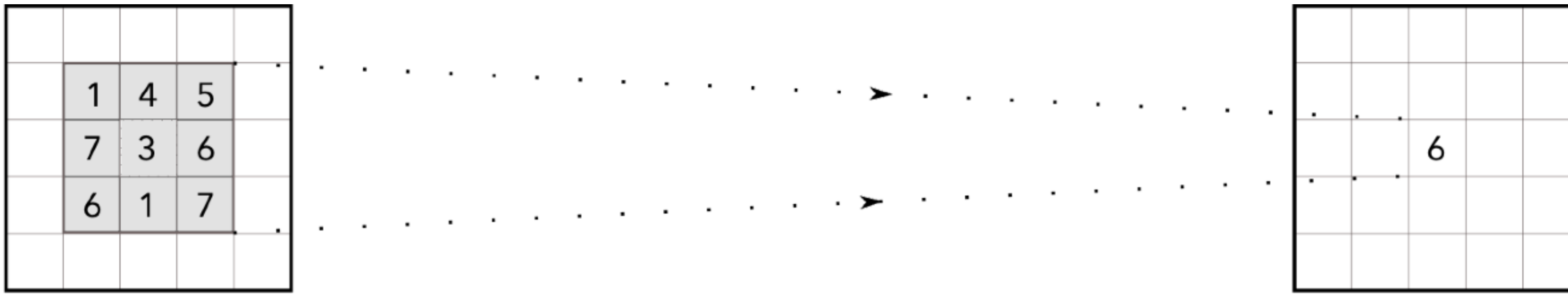


Convolution in CNNs

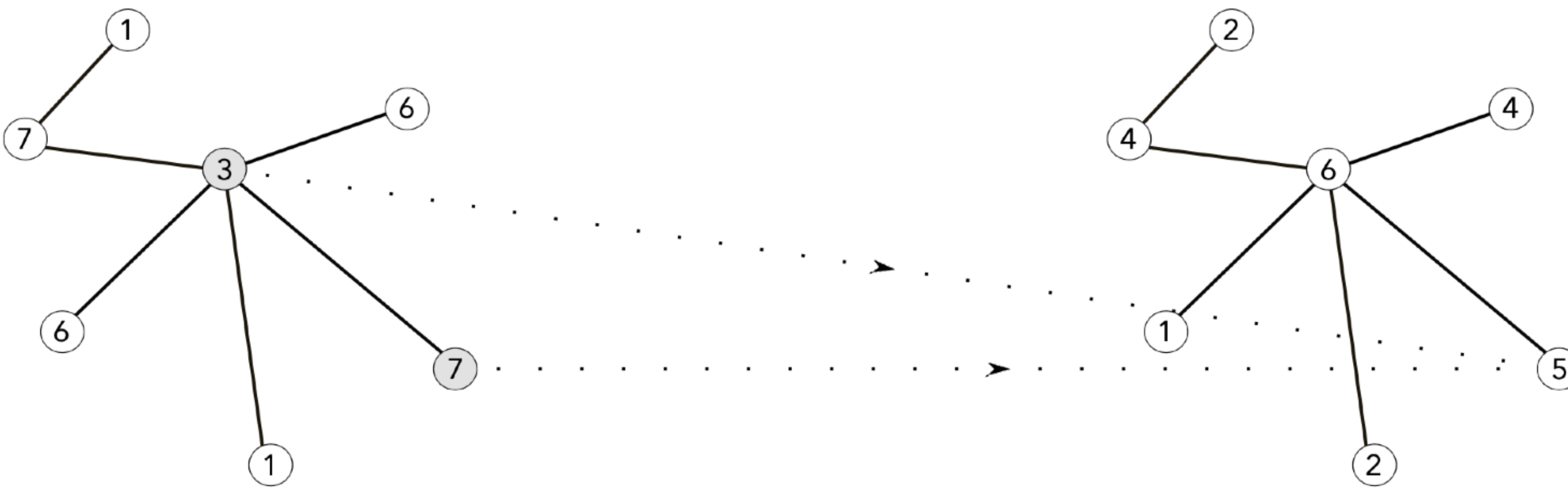


Localized Convolution in GNNs

Extending Convolutions to Graphs

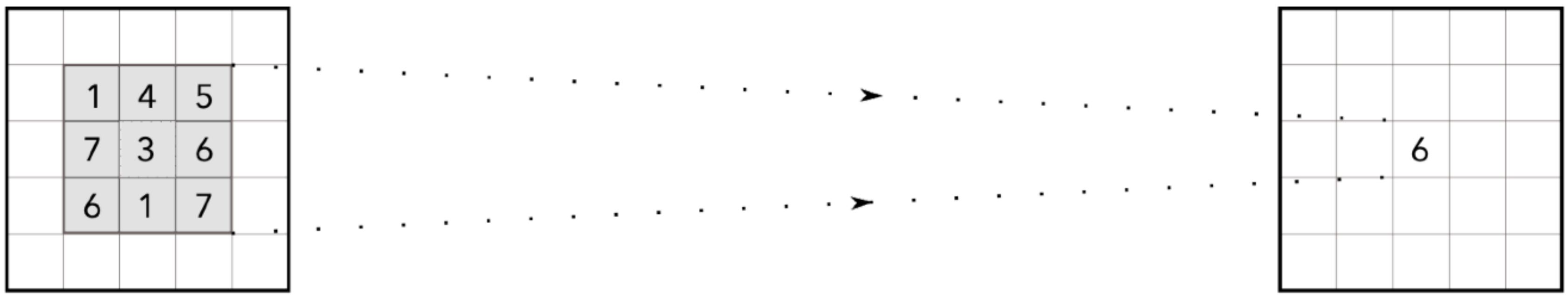


Convolution in CNNs

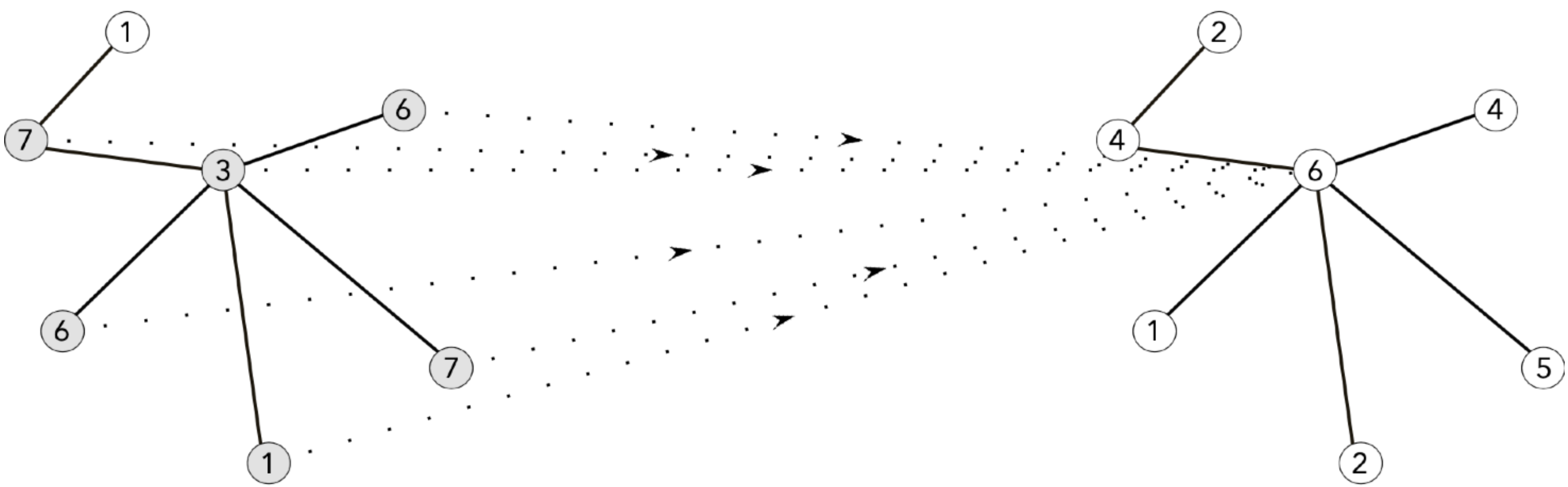


Localized Convolution in GNNs

Extending Convolutions to Graphs



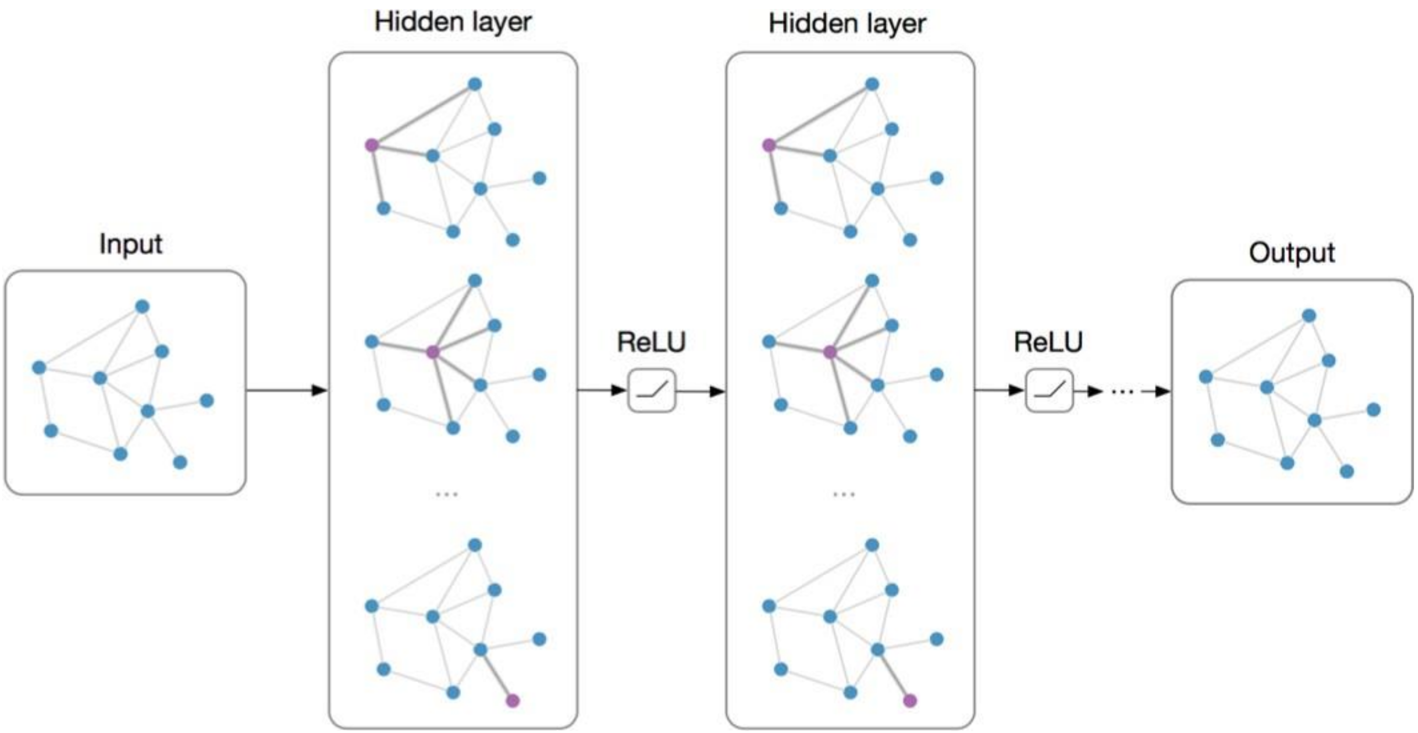
Convolution in CNNs

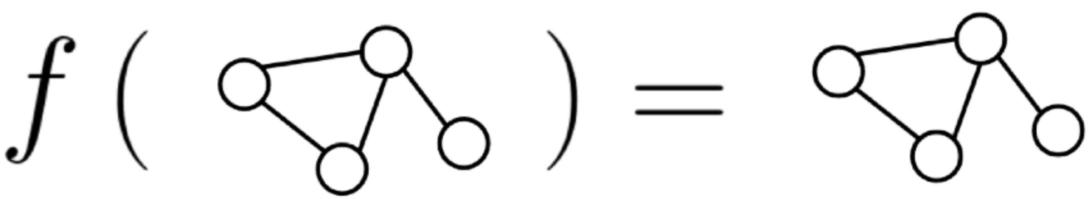


Localized Convolution in GNNs

Graph Neural Network

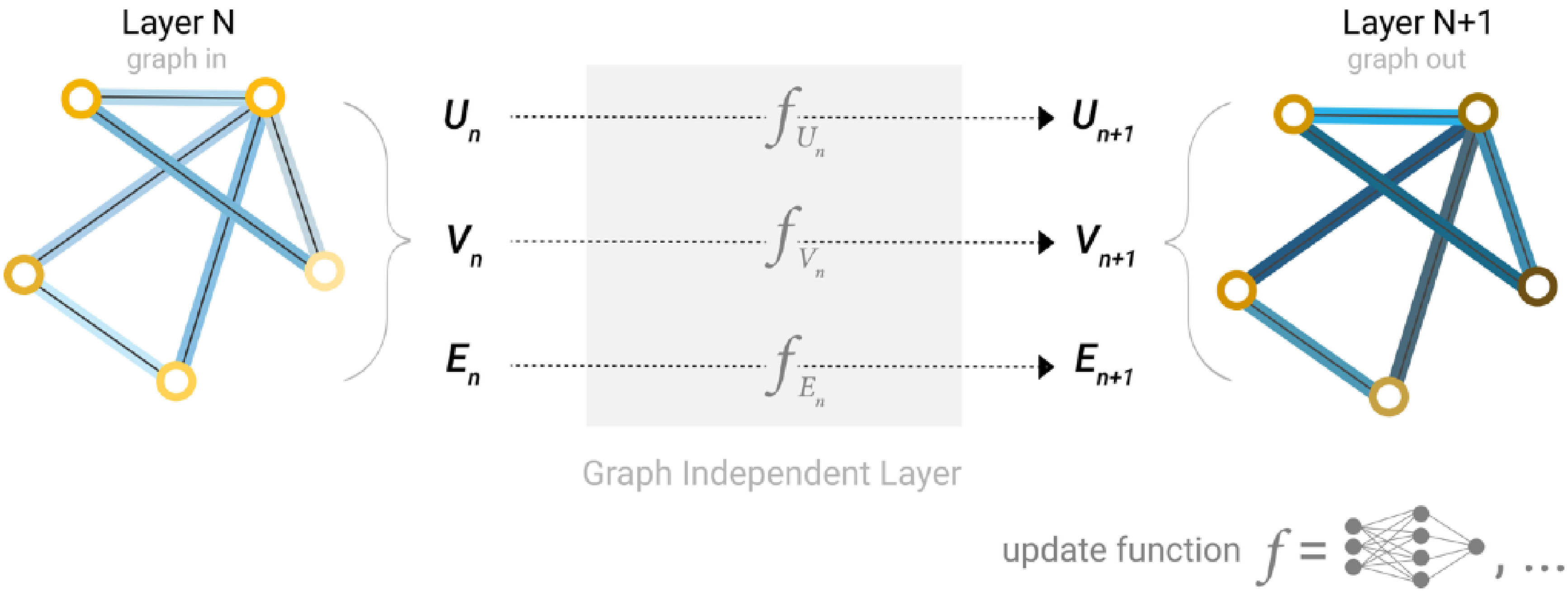
Graph Neural Network





* slide from Thomas Kipf, **University of Amsterdam**

Graph Neural Network



- GNN **does not update the connectivity** of the input graph
- Output graph of a GNN has the **same adjacency** and the **same number of feature** vectors as the input graph.
- But, the output graph has **updated embeddings**, since the GNN has updated each of the node, edge and global-context representations.

Graph Convolutional Neural Network

- Kipf and Welling (2017)

Attempt #1: $$\mathbf{h}_i^{l+1} = \sigma\left(\sum_{j \in \mathcal{N}(i)} W^l \mathbf{h}_j^l\right)$$

Graph Convolutional Neural Network

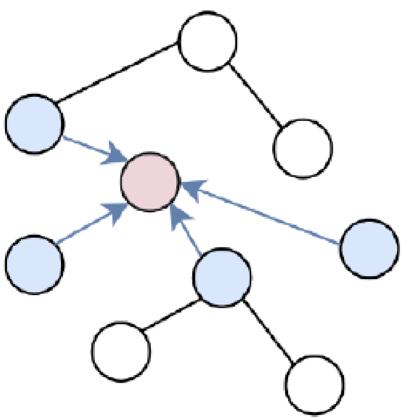
- Kipf and Welling (2017)

Attempt #1: $\mathbf{h}_i^{l+1} = \sigma\left(\sum_{j \in \mathcal{N}(i)} W^l \mathbf{h}_j^l\right)$

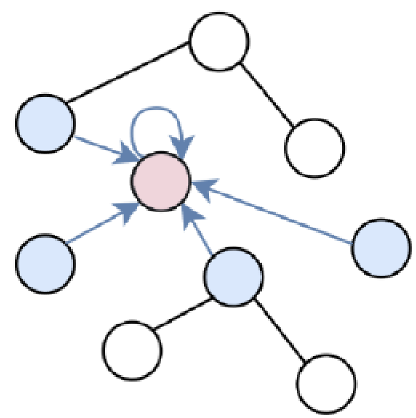
Attempt #2: $\mathbf{h}_i^{l+1} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} W^l \mathbf{h}_j^l\right)$

Normalised propagation matrices

- The GC layer remains valid if we replace the adjacency matrix with another matrix having the same sparsity pattern (i.e., 0 for pairs of nodes not connected).
  - Adjacency matrix with self-loops: $\hat{A} = A + I$ and $[\hat{D}]_{i,i} = \sum_j \hat{A}_{ij}$.
  - Normalised adjacency matrix (with or without self-loops): $L = D - A$ or $L = \hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}$



Without self-loop



With self-loop

  - Some of these matrices can have better properties when training.

Graph Convolutional Neural Network

- Kipf and Welling (2017)

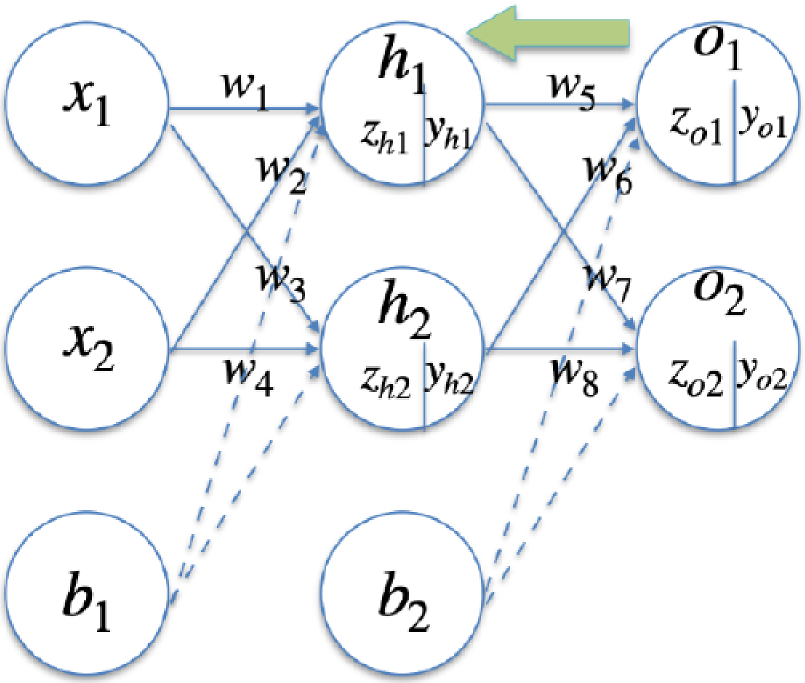
Attempt #1: $\mathbf{h}_i^{l+1} = \sigma\left(\sum_{j \in \mathcal{N}(i)} W^l \mathbf{h}_j^l\right)$

Attempt #2: $\mathbf{h}_i^{l+1} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} W^l \mathbf{h}_j^l\right)$

Attempt #3: $\mathbf{h}_i^{l+1} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{|\mathcal{N}(i)| + 1}\sqrt{|\mathcal{N}(j)| + 1}} W^l \mathbf{h}_j^l\right)$

Recap: MLP

- $\mathbf{x} = \{1, x_1, x_2, \cdots, \}^T$
- $\mathbf{w} = \{w_0, w_1, w_2, \cdots, \}^T$
- $z = \sigma(\mathbf{w}^T \mathbf{x})$
- $H^1 = f(X, W)$
- $H^{l+1} = f(H^l, W)$



Graph Layer

- We consider graph layers of the form

$$f(X, A, W) = (H, A)$$

- acting on the node features, but keeping the connectivity the same. For simplicity, we can omit the second output and write

$$H = f(X, A, W)$$

A convolution-like operation for graphs

- A graph convolutional (GC) layer is defined as:

$$[H]_i = \phi\Big(\sum_{i \in \mathcal{N}_i} A_{ij} W^T \mathbf{x}_j \Big),$$

- which can be written compactly as:

$$H = \phi(AXW)$$

- The GC layer is easily composable into a multi-layered architecture, e.g., with two layers:

$$f(X, A) = \phi(A\phi(AXW)Z)$$
$$H^{l+1} = \phi(AH^l W^l)$$

https://www.sscardapane.it/assets/files/nnds2021/Lecture_6_graphs.pdf

A convolution-like operation for graphs

- The GC layer can be understood as a sequence of three operations:
  - 1. A local **node-wise** operation $\hat{\mathbf{x}}_i = W^T \mathbf{x}_i$.
  - 2. An aggregation with respect to the neighbourhood. This is called the **message-passing** phase.
  - 3. A standard non-linearity.

- In an image convolution, points (1)-(2) are combined in the filtering operation, because each pixel has a fixed neighbourhood. Here, this is not possible because we cannot easily initialise trainable parameters to weight the neighbourhood.

Summary / Recap

Graph

- A graph is a pair $G = (V, E)$, where $V = \{1, \cdots, n\}$ is a set of n vertices (nodes), and $E = \{(i, j) \mid i, j \in N\}$ is a set of edges between them.
- Adjacency matrix A where: $A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$
- If $A^T = A$, we say the graph is undirected.
- Neighbourhood $\mathcal{N}_i = \{j \mid A_{ij} = 1\}$
- Degree $[D]_{i,i} = \sum_j A_{ij}$
- Adjacency matrix with self-loops: $\hat{A} = A + I$ and $[\hat{D}]_{i,i} = \sum_j \hat{A}_{ij}$.
- Normalised adjacency matrix / Laplacian: $L = D - A$ or $L = \hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2}$

A convolution-like operation for graphs

- A graph convolutional (GC) layer is defined as:

$$[H]_i = \phi\Big(\sum_{i \in \mathcal{N}_i} A_{ij} W^T \mathbf{x}_j\Big),$$

- which can be written compactly as:

$$H = \phi(AXW)$$

- The GC layer is easily composable into a multi-layered architecture, e.g., with two layers:

$$f(X, A) = \phi(A\phi(AXW)Z)$$
$$H^{l+1} = \phi(AH^l W^l)$$

How do we use GNN / GCN for real problems?

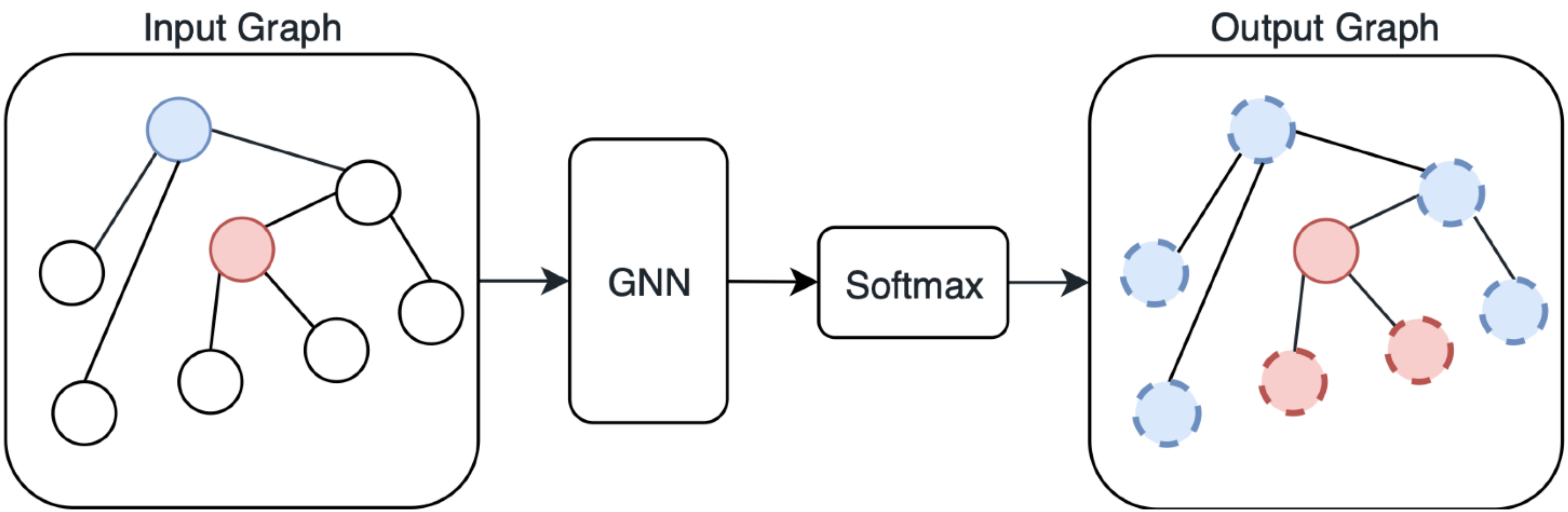
GNN Tasks

- **Node Classification**: label data samples by taking into account their neighbours.

- **Link Prediction**: predict most likely links in the graph.

Task #1: Node classification

- For the node classification, we can stack two graph convolutional layer to form a Graph Neural Network. A softmax() applied on each node embedding of the last layer will provide the prediction y.



Kipf, T.N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907

Task #1: Node classification

- Suppose a subset $\mathcal{T} \subset \mathcal{N}$ of nodes has a known class y (e.g., fake or certified users in a social network). We can use a GCN to classify all the nodes of the graph simultaneously:

$$\hat{Y} = \text{softmax}(A\varphi(AXW)Z)$$

- We optimise the weights of the network with gradient descent:

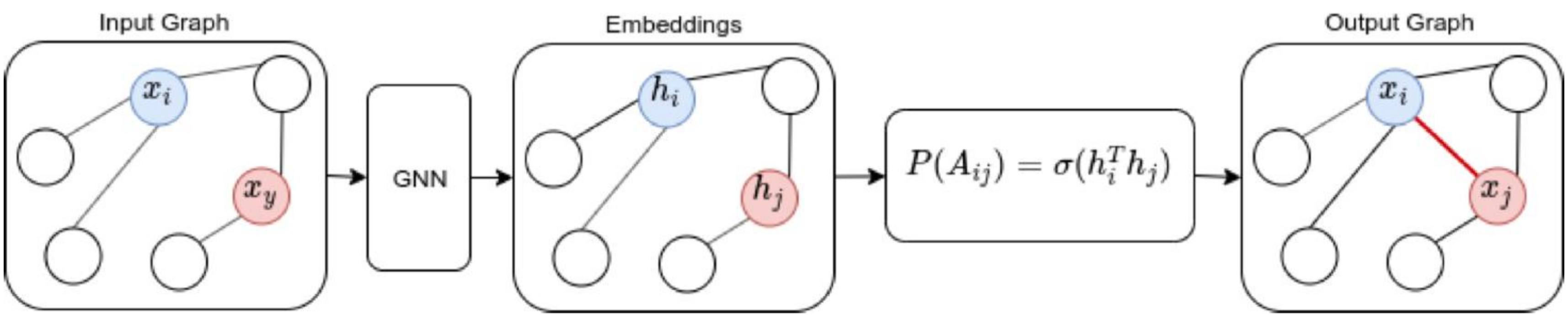
$$W^*, Z^* = \arg\min \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \text{cross-entropy}(y_i, \hat{y}_i) \, .$$

Kipf, T.N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907

Task #2: Link Prediction

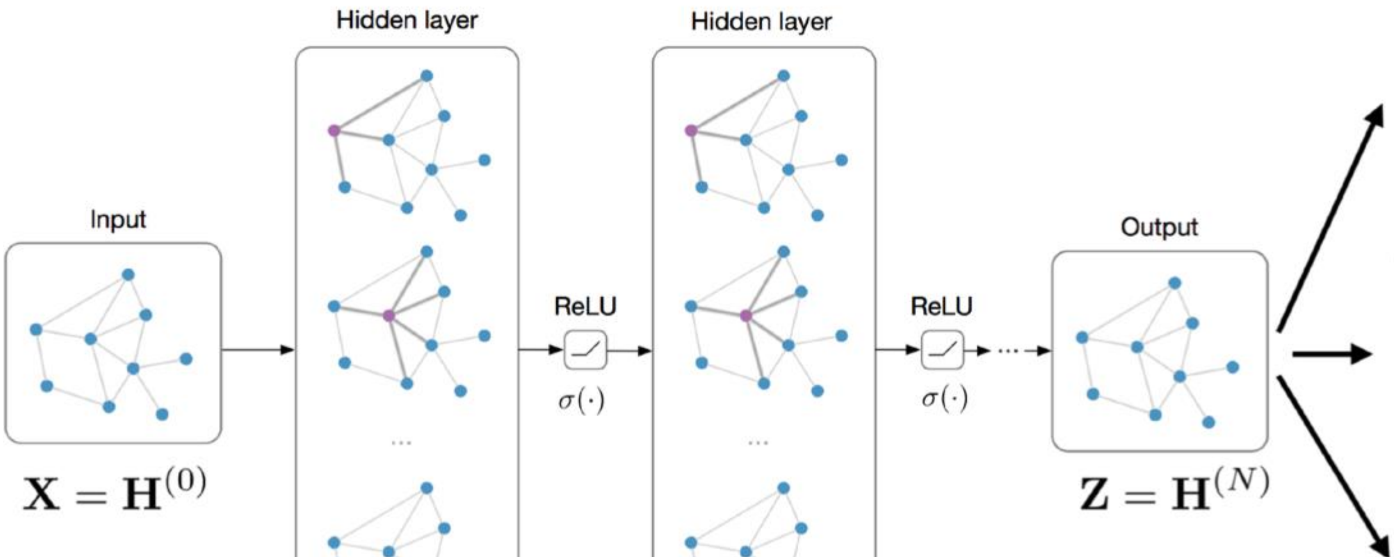
- In link prediction, the objective is to predict whether two nodes in a network are likely to have a link.

- We can get the probability for each edge by taking the inner product of the embedding produced by a GNN.



- Also, in this case the model is optimised by minimising the **cross-entropy loss** function.

Node classification, Graph Classification, Link Prediction



$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

Node classification:

$$\mathrm{softmax}(\mathbf{z_n})$$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$$\mathrm{softmax}(\textstyle\sum_n \mathbf{z_n})$$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

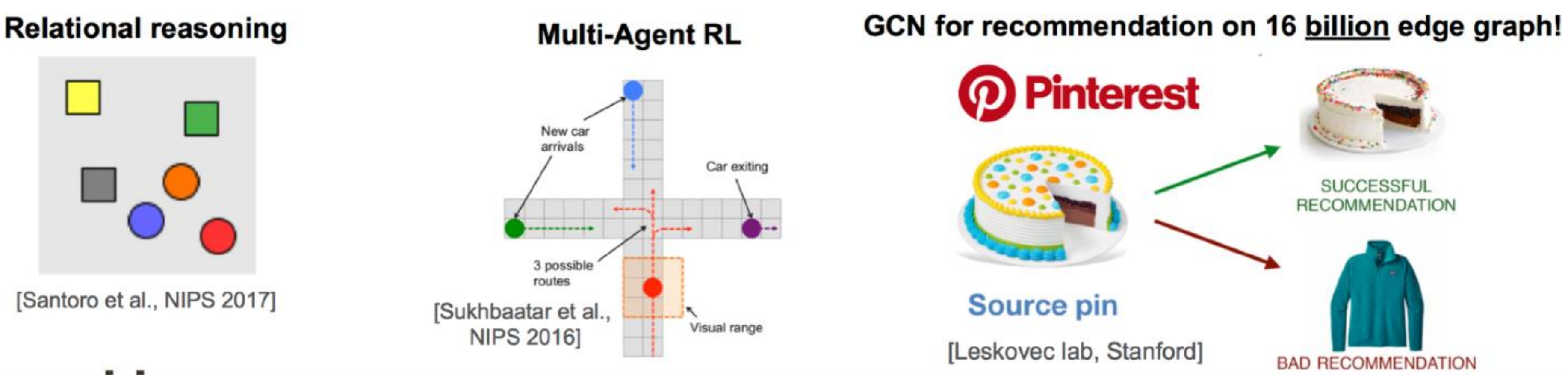
$$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$$

Kipf & Welling (NIPS BDL 2016)
"Graph Auto-Encoders"

https://www.cs.ubc.ca/~lsigal/532S_2018W2/Lecture18a.pdf

Conclusions

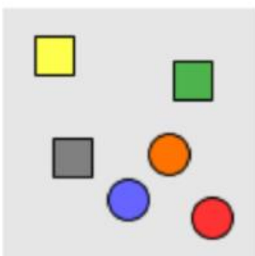
- Deep learning on graphs works and is very effective!
- Exciting area: lots of new applications and extensions (hard to keep up)



Conclusions

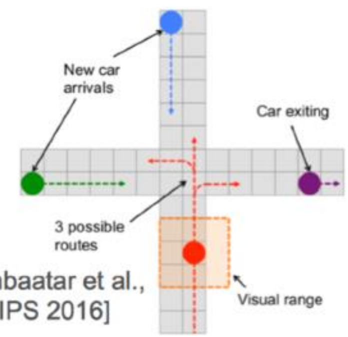
- Deep learning on graphs works and is very effective!
- Exciting area: lots of new applications and extensions (hard to keep up)

Relational reasoning



[Santoro et al., NIPS 2017]

Multi-Agent RL



[Sukhbaatar et al., NIPS 2016]

GCN for recommendation on 16 billion edge graph!

[Leskovec lab, Stanford]

- Open problems:
  - Theory
  - Scalable, stable generative models
  - Learning on large, evolving data
  - Multi-modal and cross-model learning (e.g., sequence2graph)

Software Libraries

- Multiple libraries build on standard deep learning frameworks to provide GNN functionalities:
  - Deep Graph Library (framework agnostic, very scalable):
  - https://www.dgl.ai/

  - PyTorch Geometric (great for reproducing results):
  - https://github.com/pyg-team/pytorch_geometric

  - Spektral (TensorFlow, smaller than the other two):
  - https://graphneural.network/