Graph Neural Networks

M. Soleymani
Sharif University of Technology
Spring 2024

Most slides have been adapted from:
Jure Leskovec, Machine Learning with Graphs, CS224W
and Thomas Kipf, University of Amsterdam

Our current DL toolbox

Doubt thou the stars are fire,
Doubt that the sun doth move,
Doubt truth to be a liar,
But never doubt I love...

Text

Audio signals

Images

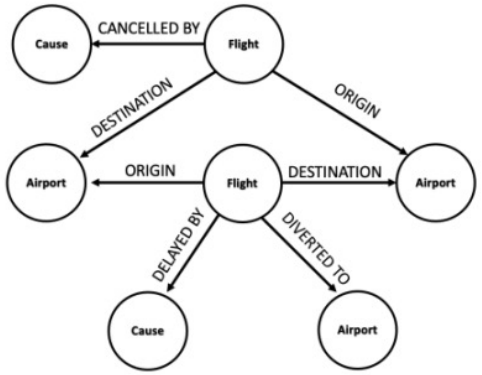
Our current deep learning toolbox is designed for sequences & grids

But, not everything can be represented as a sequence or a grid

Why graphs ?

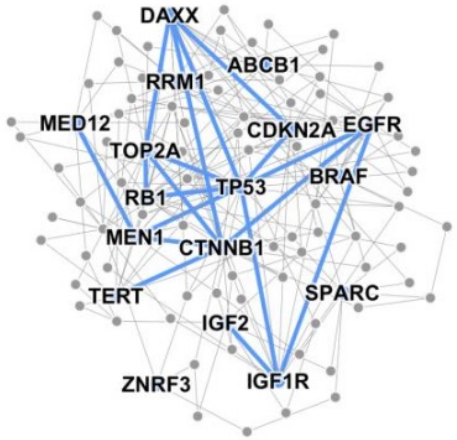
Graphs are a general language for describing and analyzing entities with relations/interactions

Many types of graphs (1)

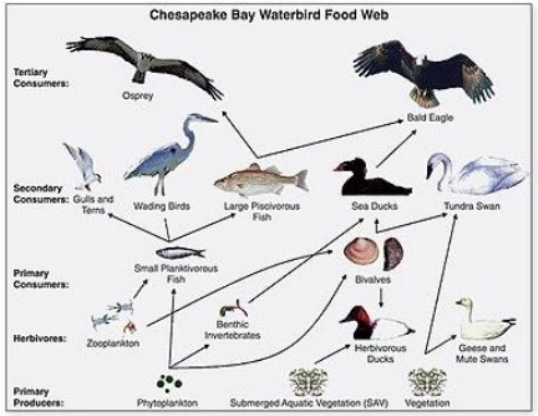


Event Graphs

Computer Networks



Disease Pathways



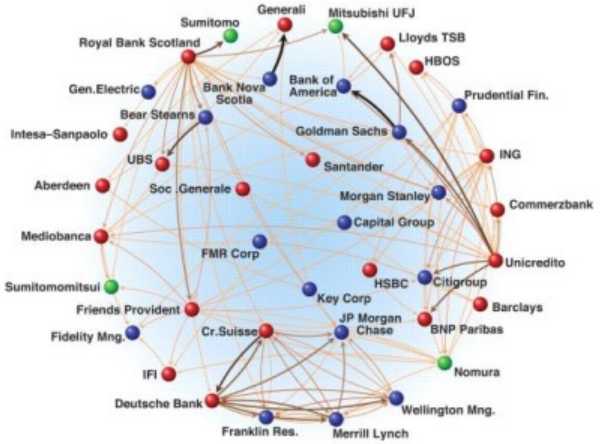
Food Webs

Particle Networks

Underground Networks

Many types of graphs (2)

Social Networks

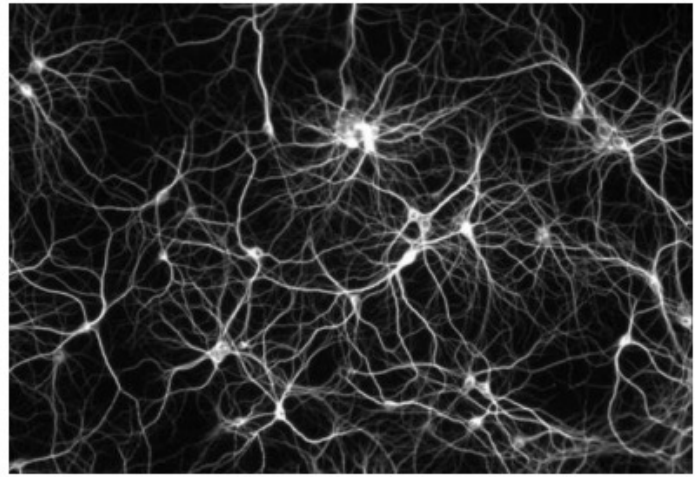


Economic Networks

Communication Networks

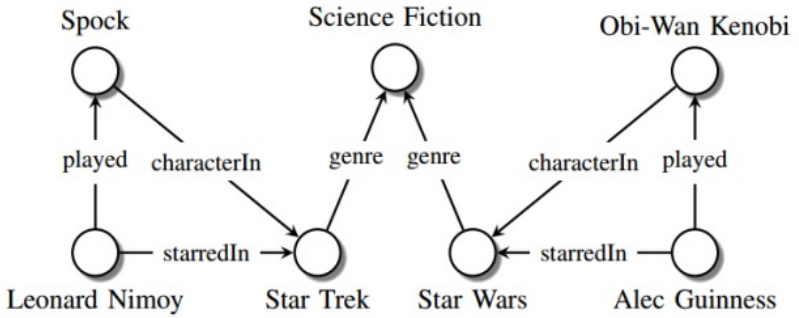
Citation Networks

Internet

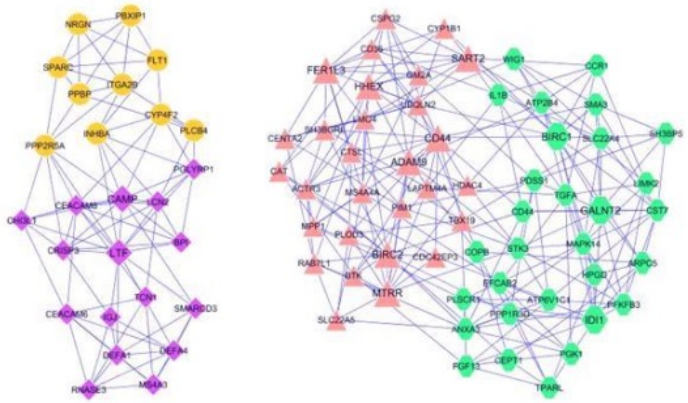


Networks of Neurons

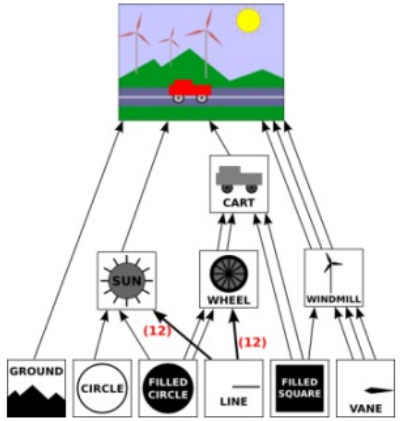
Many types of graphs (3)



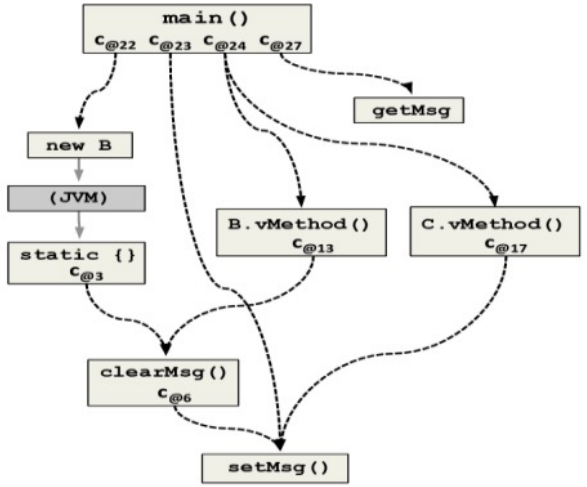
Knowledge Graphs



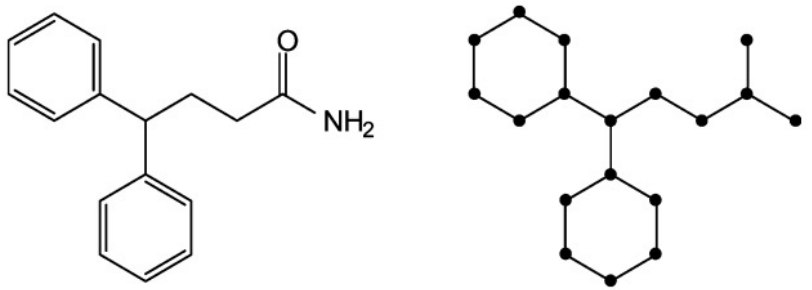
Regulatory Networks



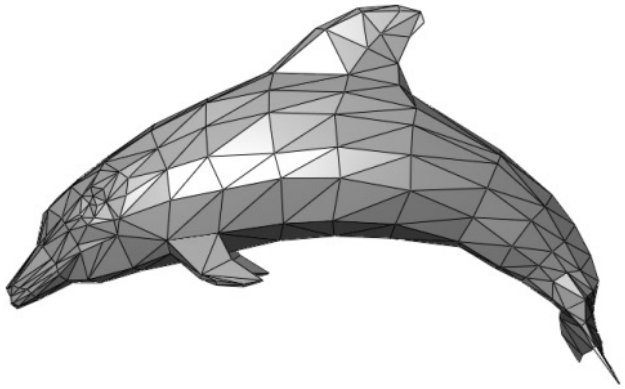
Scene Graphs



Code Graphs



Molecules

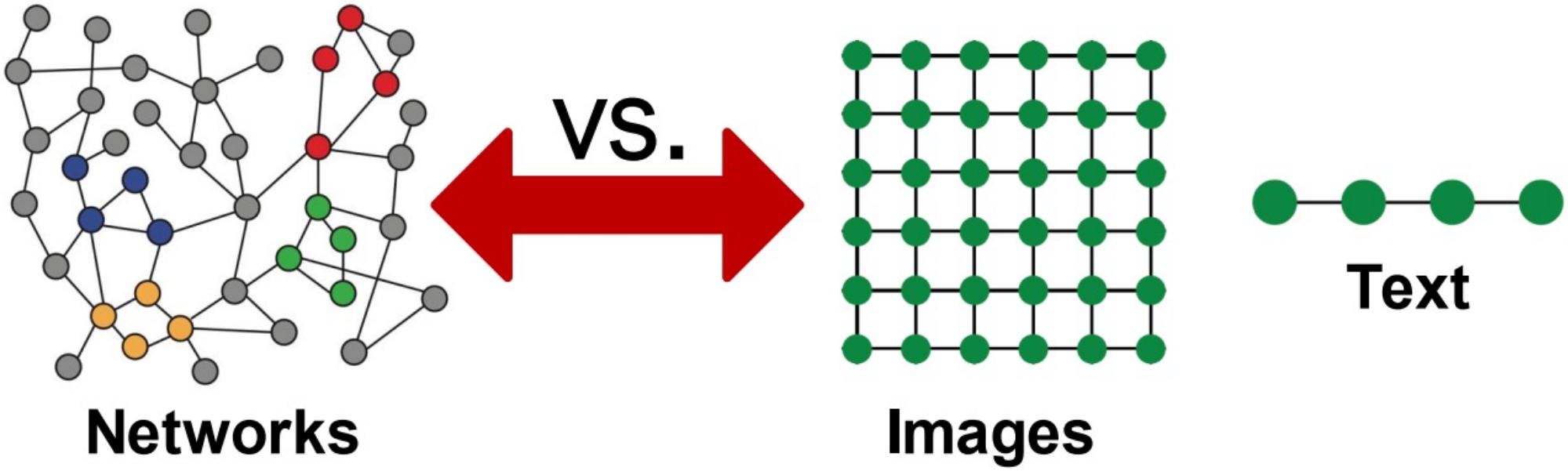


3D Shapes

What is different in graphs ?

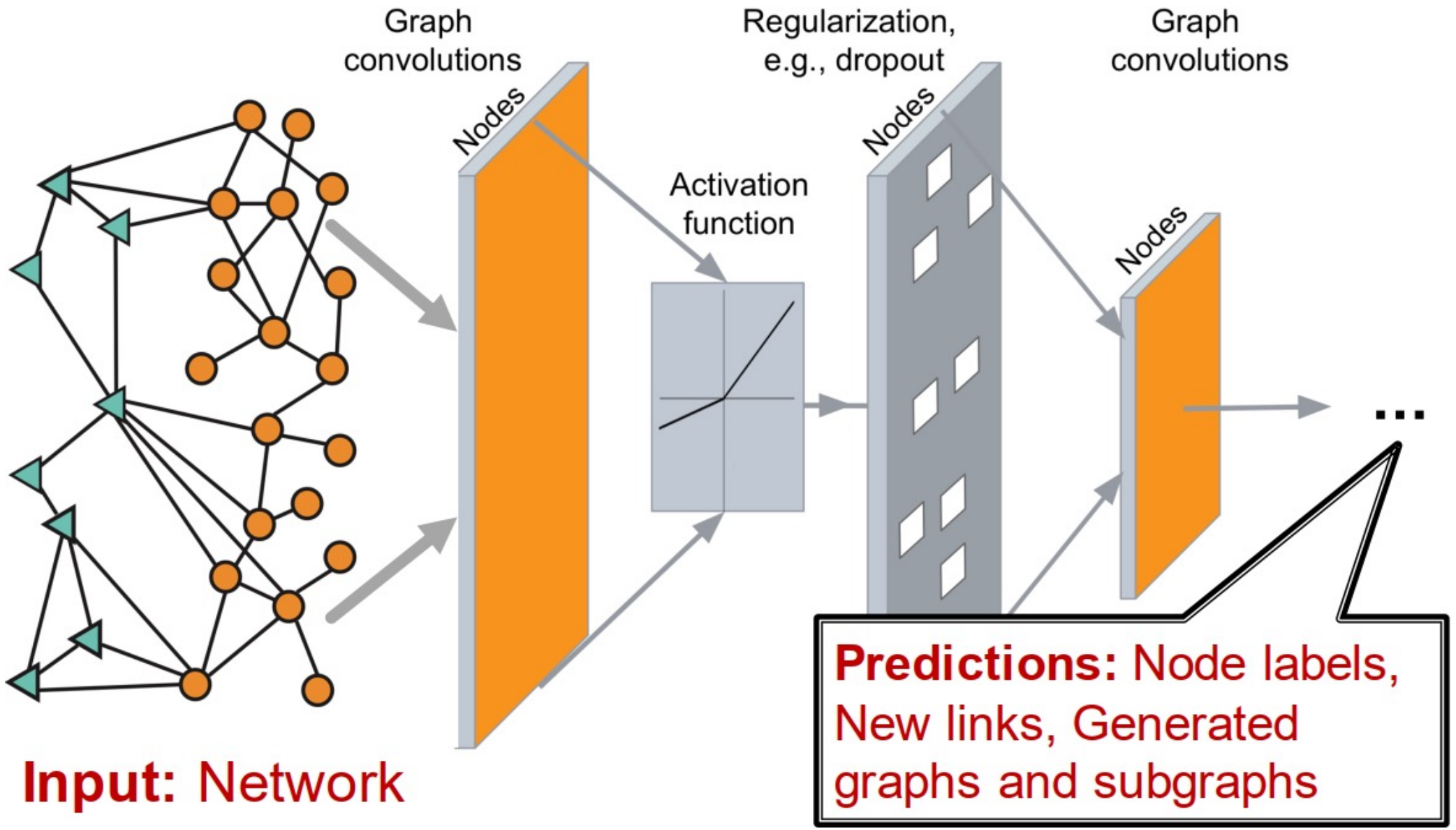
Graphs are complex:

- Arbitrary size and complex topological structure (i.e., no spatial locality like grids)



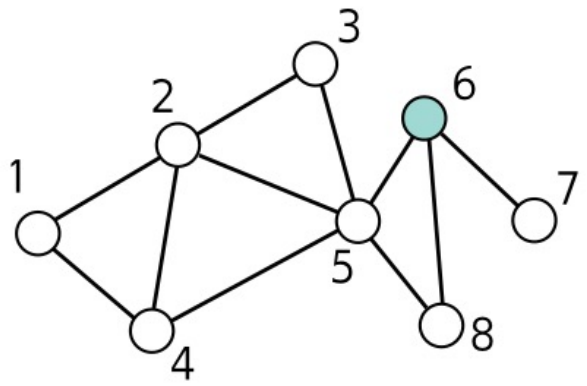
- No fixed node ordering or reference point

ML with graphs (Big picture)



Properties of the adjacency matrix

a)



b)

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

c)

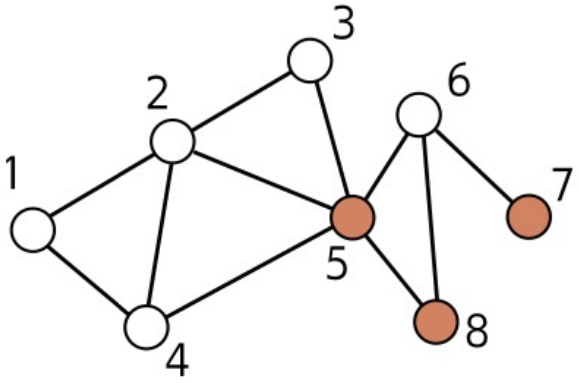
$$\mathbf{A}^2 = \begin{bmatrix} 2 & 1 & 1 & 1 & 2 & 0 & 0 & 0 \\ 1 & 4 & 1 & 2 & 2 & 1 & 0 & 1 \\ 1 & 1 & 2 & 2 & 1 & 1 & 0 & 1 \\ 1 & 2 & 2 & 3 & 1 & 1 & 0 & 1 \\ 2 & 2 & 1 & 1 & 5 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \end{bmatrix}$$

d)

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

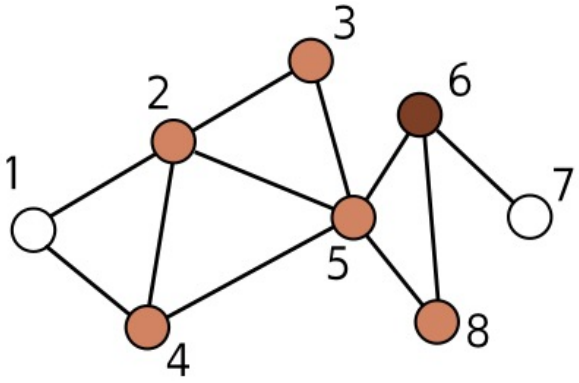
e)

$$\mathbf{Ax} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



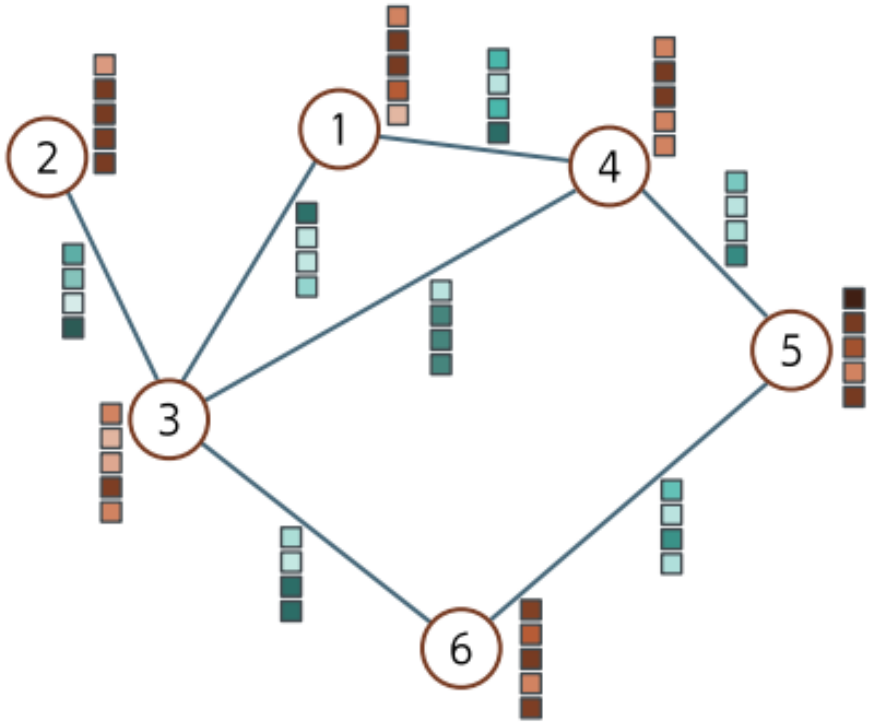
f)

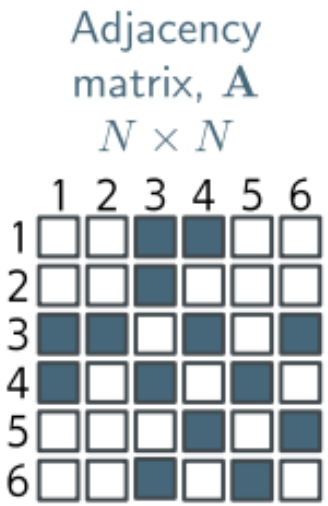
$$\mathbf{A}^2\mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 3 \\ 0 \\ 1 \end{bmatrix}$$

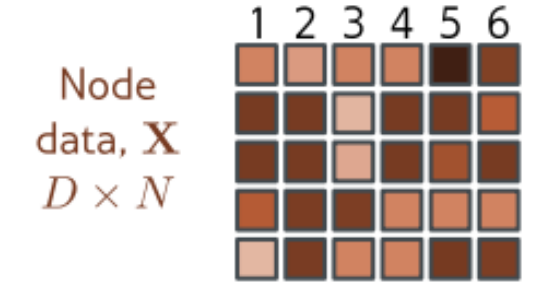


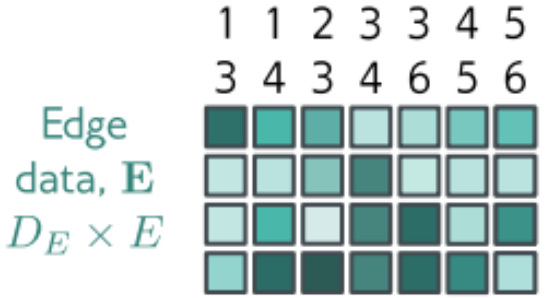
Price, Understanding Deep Learning, 2023

Graph representation





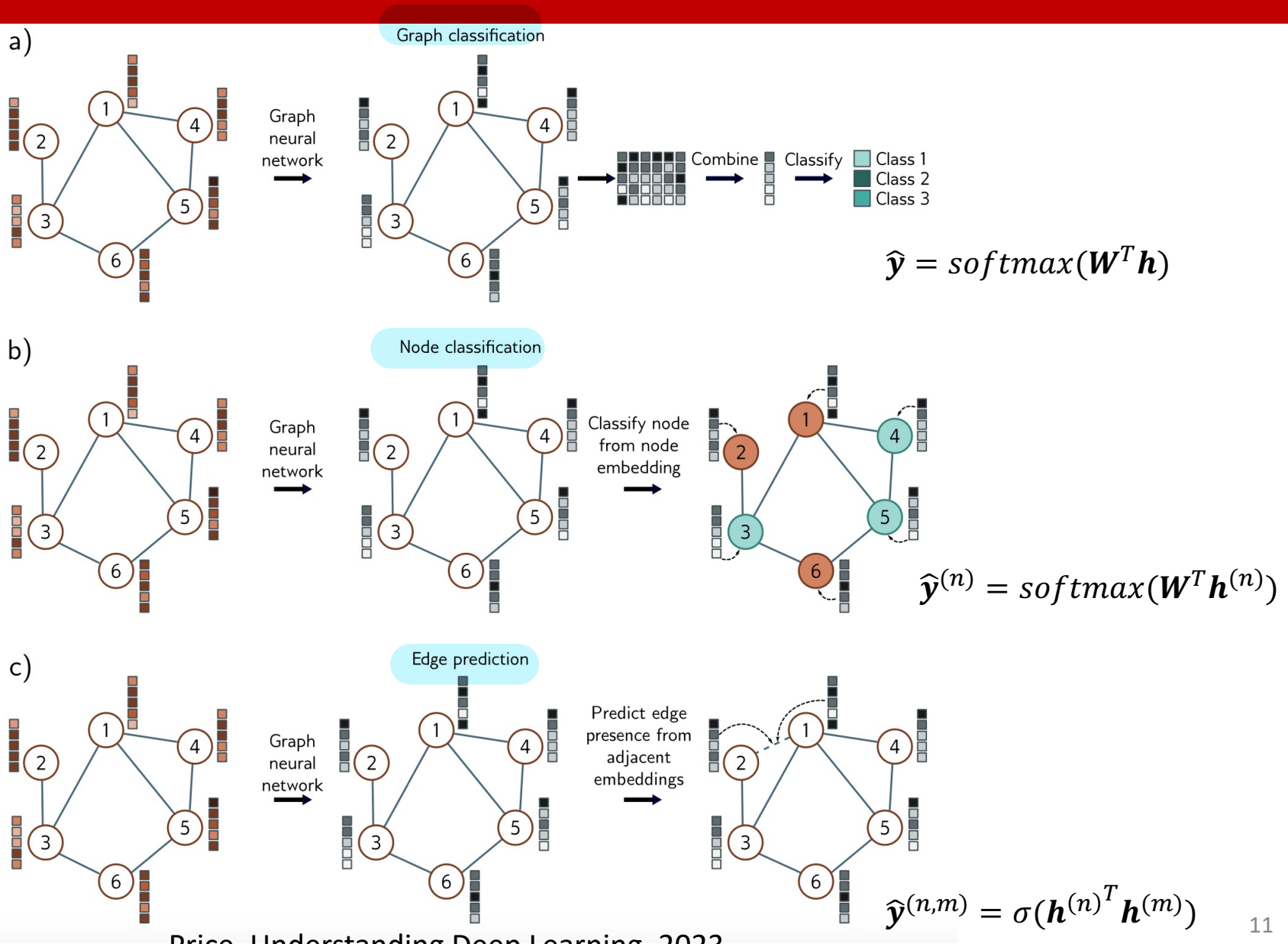




Graph representation: Example graph with six nodes and seven edges. Each node has an associated embedding of length five (brown vectors). Each edge has an associated embedding of length four (blue vectors). This graph can be represented by three matrices. The adjacency matrix is a binary matrix where element (m, n) is set to one if node m connects to node n. The node data matrix X contains the concatenated node embeddings. The edge data matrix E contains the edge embeddings.

Price, Understanding Deep Learning, 2023

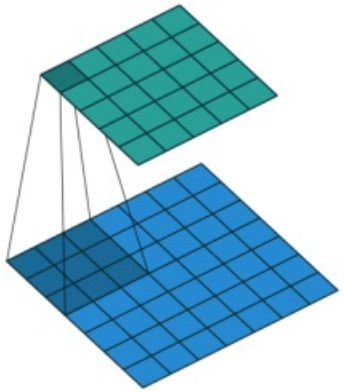
Common tasks for graphs

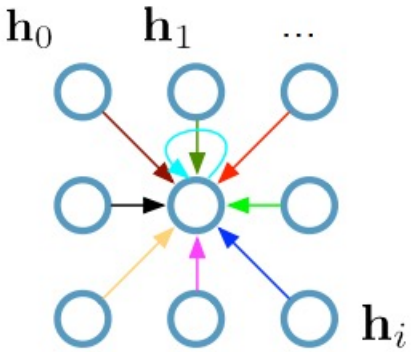


Price, Understanding Deep Learning, 2023

Review (Convolutional neural networks)

Single CNN layer with 3x3 filter:

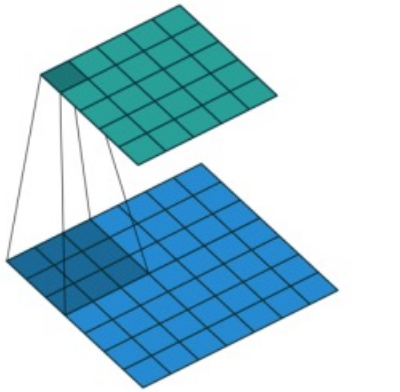


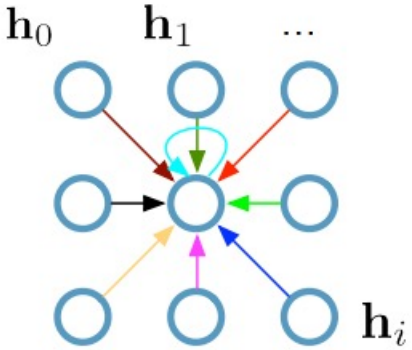


$h_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Review (Convolutional neural networks)

Single CNN layer with 3x3 filter:





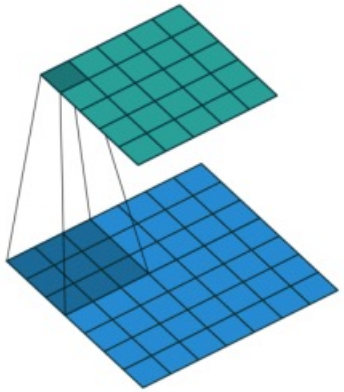
$h_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

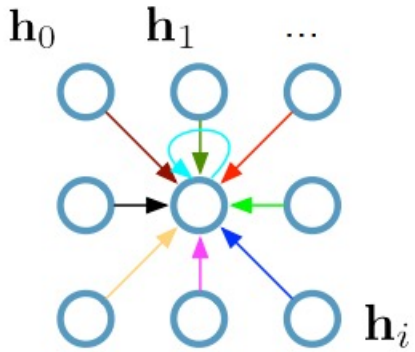
Updates for a single pixel:

- Transform messages individually $W_i h_i$
- Add everything up $\sum W_i h_i$

Review (Convolutional neural networks)

Single CNN layer with 3x3 filter:





$h_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Updates for a single pixel:

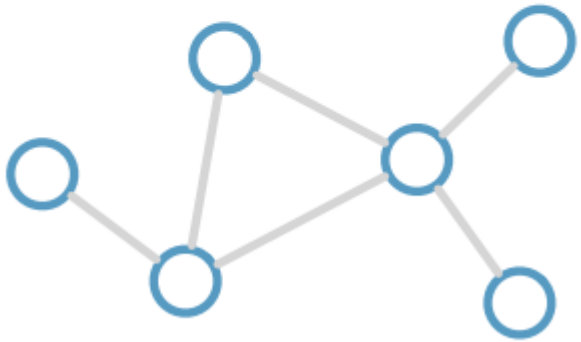
- Transform messages individually $W_i h_i$
- Add everything up $\sum W_i h_i$

Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

Graph convolutional networks (GCNs)

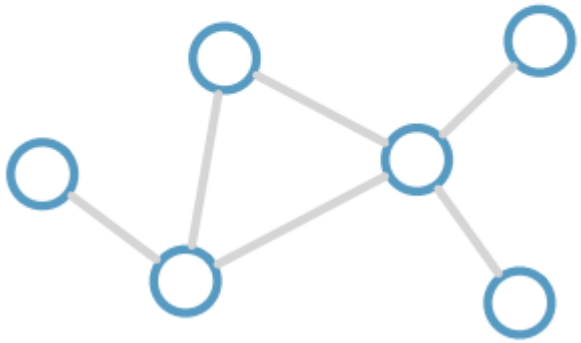
Consider this undirected graph:



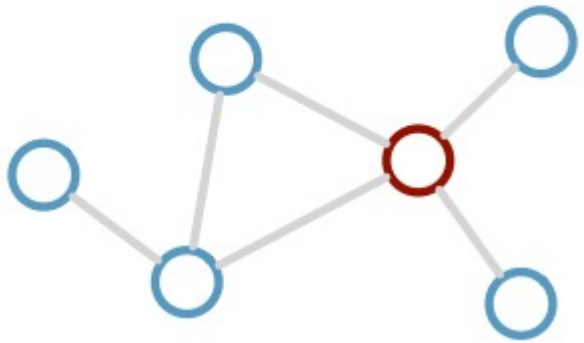
Kipf & Welling (ICLR 2017), related previous works by Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Graph convolutional networks (GCNs)

Consider this undirected graph:



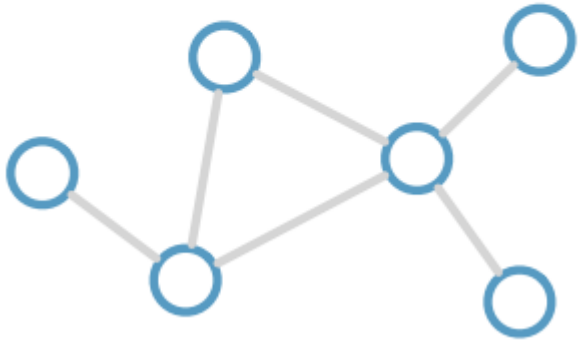
Calculate update for node in red:



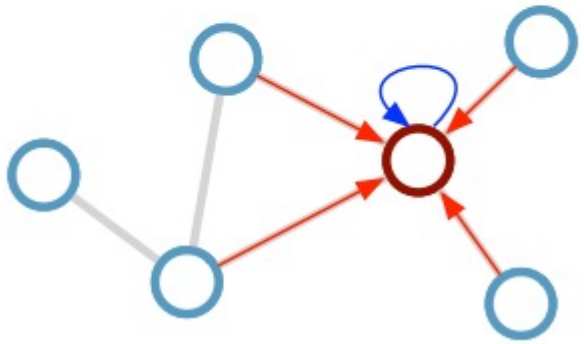
Kipf & Welling (ICLR 2017), related previous works by Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Graph convolutional networks (GCNs)

Consider this undirected graph:



Calculate update for node in red:



Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

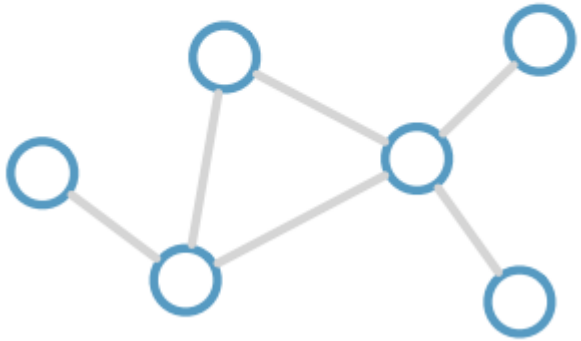
Parameter sharing: uses the same parameters at every node, reducing the number of parameters

$\mathcal{N}_i$: neighbor indices c_{ij} : norm. constant (fixed/trainable)

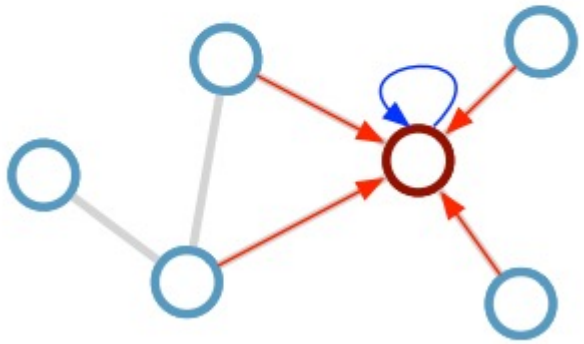
Kipf & Welling (ICLR 2017), related previous works by Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Graph convolutional networks (GCNs)

Consider this undirected graph:



Calculate update for node in red:



Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

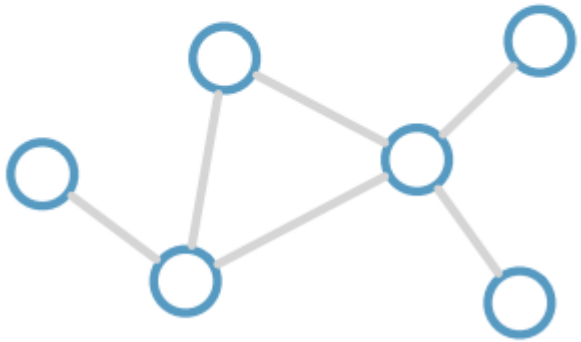
$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

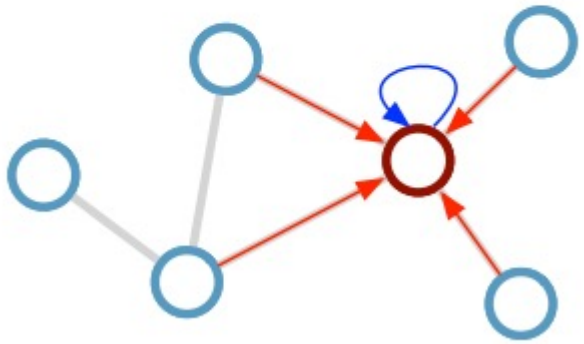
Kipf & Welling (ICLR 2017), related previous works by Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Graph convolutional networks (GCNs)

Consider this undirected graph:



Calculate update for node in red:



Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

Desirable properties:

- Weight sharing over all locations
- Invariance to permutations
- Linear complexity O(E)
- Applicable both in transductive and inductive settings

Limitations:

- Requires gating mechanism / residual connections for depth
- Only indirect support for edge features

$\mathcal{N}_i$: neighbor indices c_{ij} : norm. constant (fixed/trainable)

Kipf & Welling (ICLR 2017), related previous works by Duvenaud et al. (NIPS 2015) and Li et al. (ICLR 2016)

Message-passing neural network

Input: Undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

Initial node embeddings $\{\mathbf{h}_n^{(0)} = \mathbf{x}_n\}$

Aggregate$(\cdot)$ function

Update$(\cdot, \cdot)$ function

Output: Final node embeddings $\{\mathbf{h}_n^{(L)}\}$

// Iterative message-passing

for $l \in \{0, \ldots, L-1\}$ **do**

$\mathbf{z}_n^{(l)} \leftarrow \text{Aggregate}\left(\left\{\mathbf{h}_m^{(l)} : m \in \mathcal{N}(n)\right\}\right)$

$\mathbf{h}_n^{(l+1)} \leftarrow \text{Update}\left(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l)}\right)$

end for

return $\{\mathbf{h}_n^{(L)}\}$

aggregation function gets a variable number of neighboring nodes and does not depend on the ordering of them

can also be a differentiable function of a set of learnable parameters

Bishop, Deep Learning, 2024

Graph convolutional networks (GCNs)

- Instantiation:
  - $\boldsymbol{H}^{(l+1)} = \sigma(\widehat{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)})$
  - $\widehat{\boldsymbol{A}} = \widetilde{\boldsymbol{A}} + \boldsymbol{I}$
  - $\tilde{A}_{ij} = \frac{A_{ij}}{\sqrt{d_i}\sqrt{d_j}}$

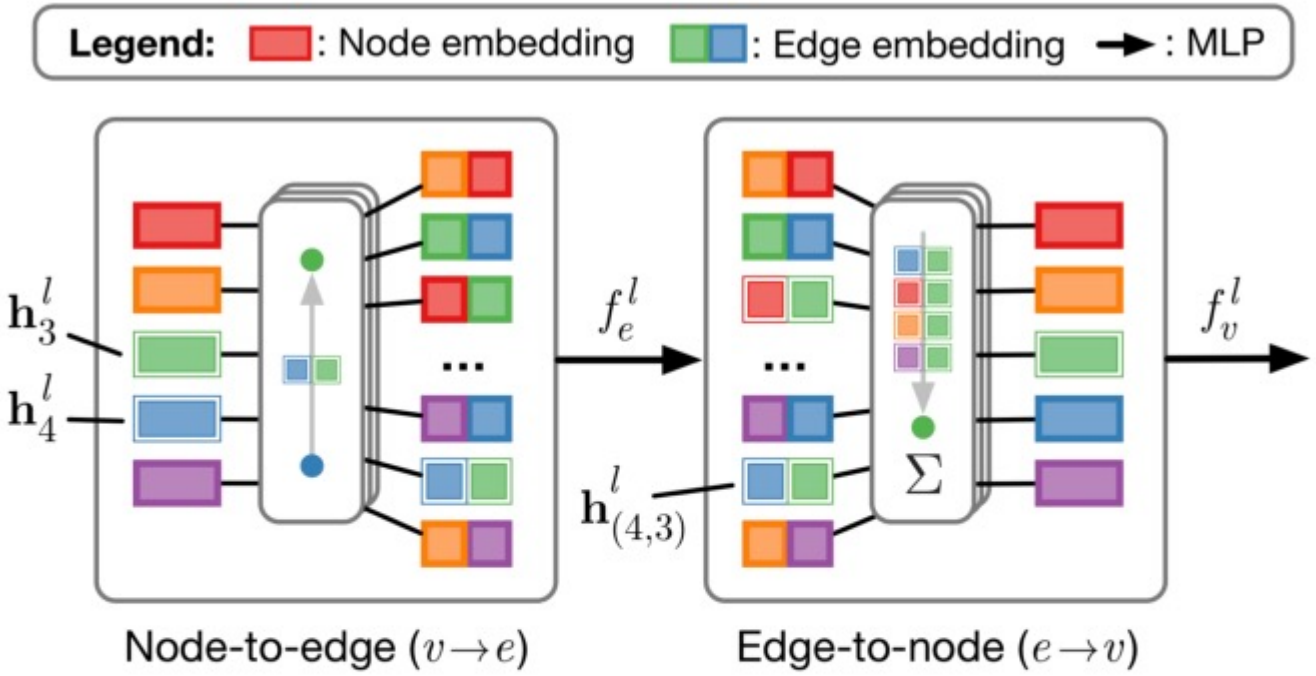
Kipf normalization

$$\mathbf{H}^{(1)} = \mathbf{F}\left(\mathbf{X}, \mathbf{A}, \mathbf{W}^{(1)}\right)$$
$$\mathbf{H}^{(2)} = \mathbf{F}\left(\mathbf{H}^{(1)}, \mathbf{A}, \mathbf{W}^{(2)}\right)$$
$$\vdots \quad = \quad \vdots$$
$$\mathbf{H}^{(L)} = \mathbf{F}\left(\mathbf{H}^{(L-1)}, \mathbf{A}, \mathbf{W}^{(L)}\right)$$

Considering k-hop neighborhood by applying k GCN layers

Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017

GNNs with edge embeddings



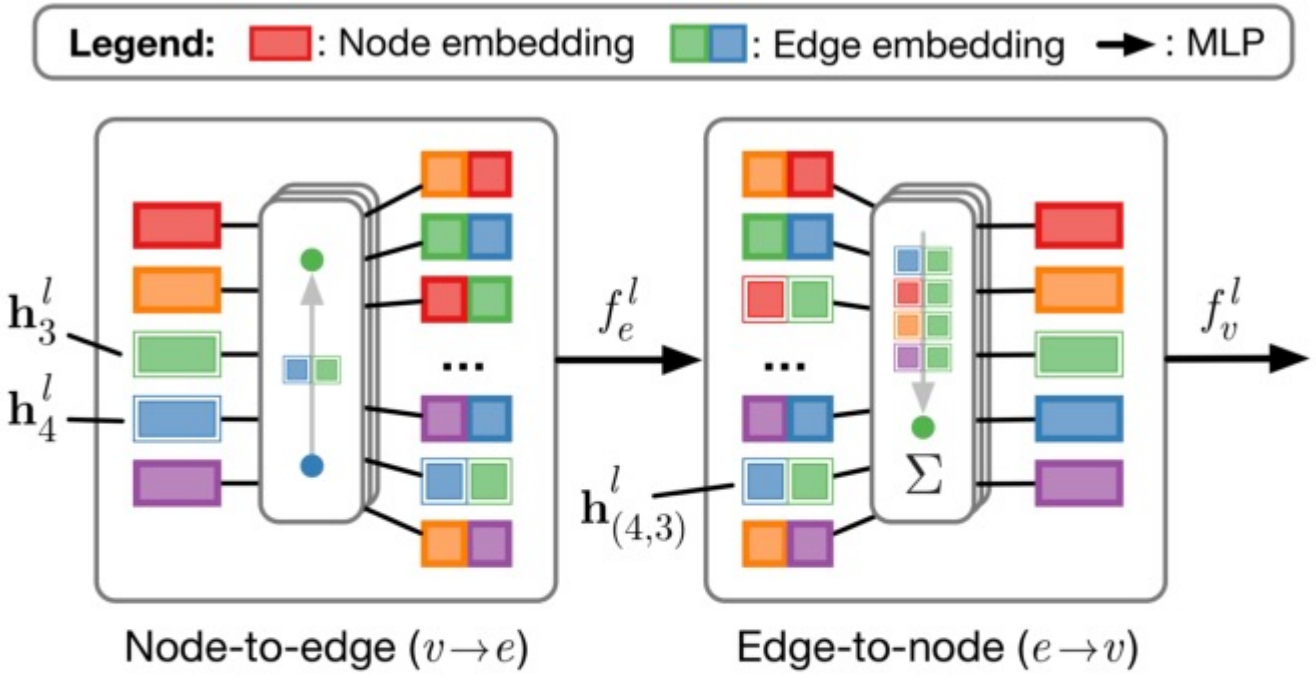
Formally:

$$v \to e: \quad \mathbf{h}_{(i,j)}^l = f_e^l([\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{x}_{(i,j)}])$$

$$e \to v: \quad \mathbf{h}_j^{l+1} = f_v^l([\textstyle\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^l, \mathbf{x}_j])$$

Battaglia et al. (NIPS 2016), Gilmer et al. (ICML 2017), Kipf et al. (ICML 2018)

GNNs with edge embeddings



Formally:

$$v \to e: \quad \mathbf{h}_{(i,j)}^l = f_e^l([\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{x}_{(i,j)}])$$

$$e \to v: \quad \mathbf{h}_j^{l+1} = f_v^l([\textstyle\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^l, \mathbf{x}_j])$$

Pros:

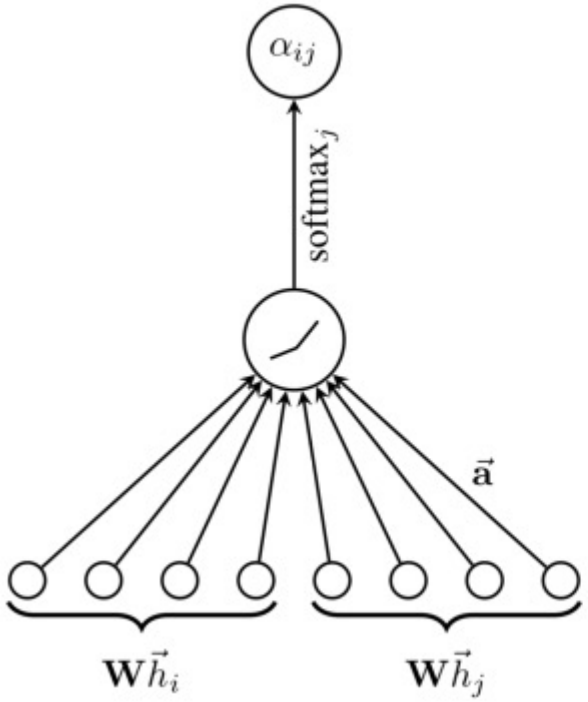
- Supports edge features
- More expressive than GCN
- As general as it gets (?)
- Supports sparse matrix ops

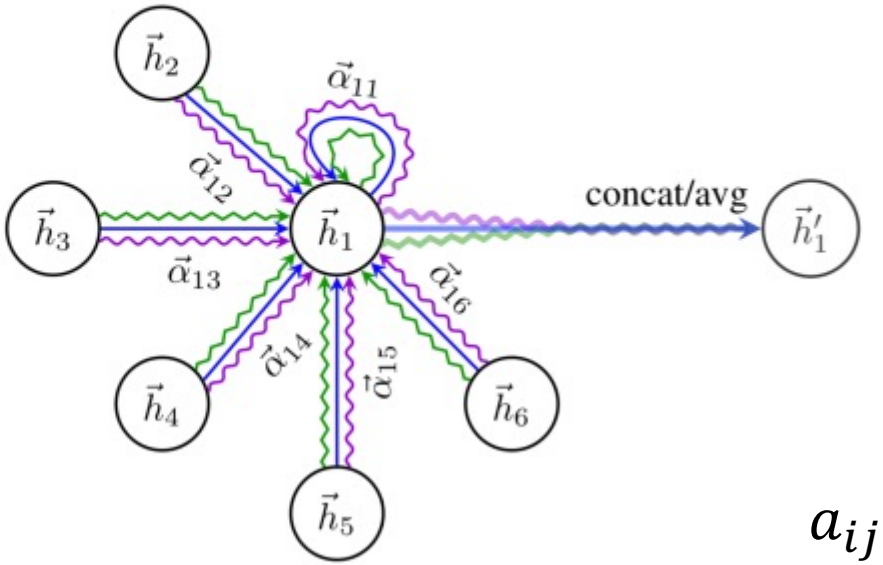
Cons:

- Need to store intermediate edge-based activations
- Difficult to implement with subsampling
- In practice limited to small graphs

Battaglia et al. (NIPS 2016), Gilmer et al. (ICML 2017), Kipf et al. (ICML 2018)

Graph neural networks with attention



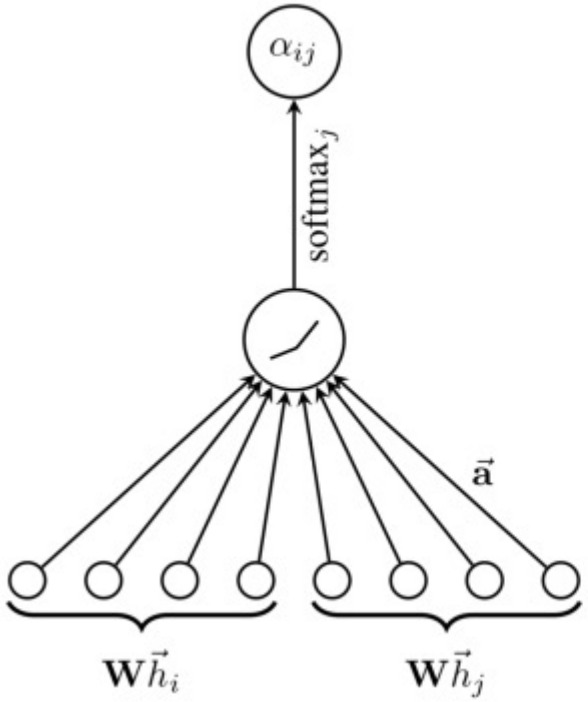


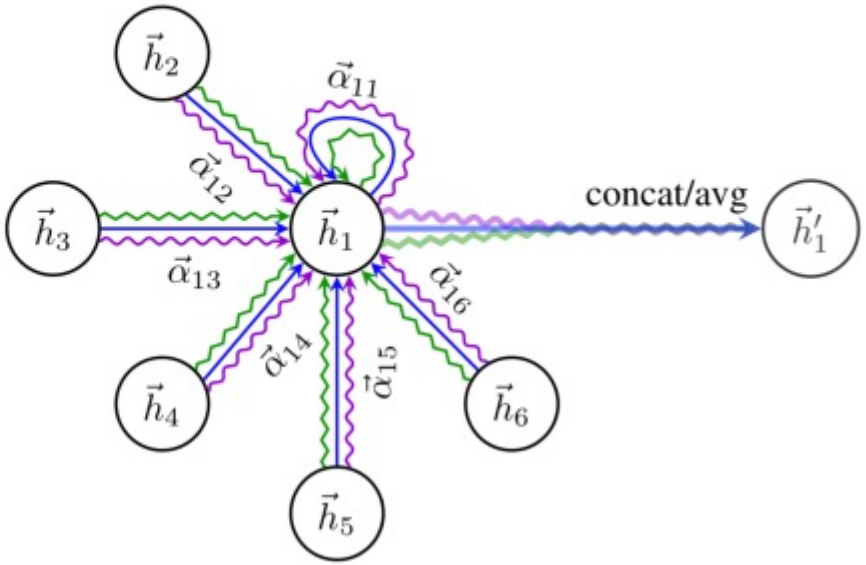
$$a_{ij} = \frac{\exp(h_i^T W h_j)}{\sum_{k \in \mathcal{N}_i} \exp(h_i^T W h_k)}$$

$$h_i^{l+1} = \sum_{j \in \mathcal{N}_i} a_{ij} h_j^l$$

Monti et al. (CVPR 2017), Hoshen (NIPS 2017), Veličkovićet al. (ICLR 2018)

Graph neural networks with attention



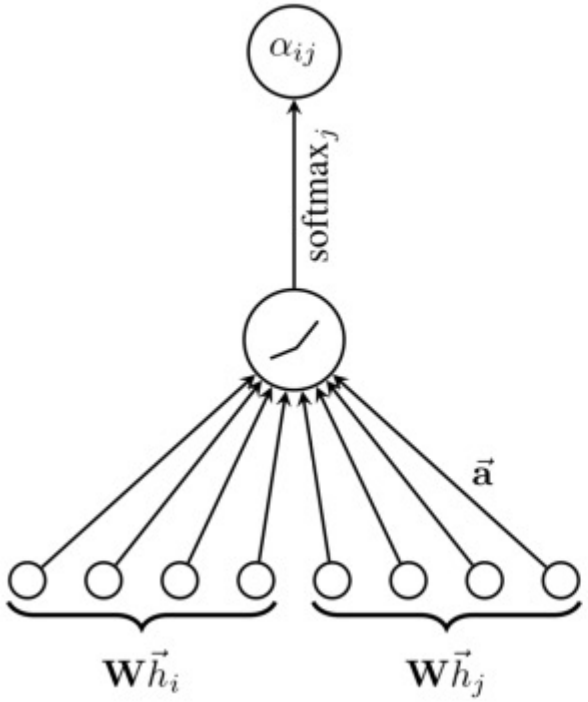


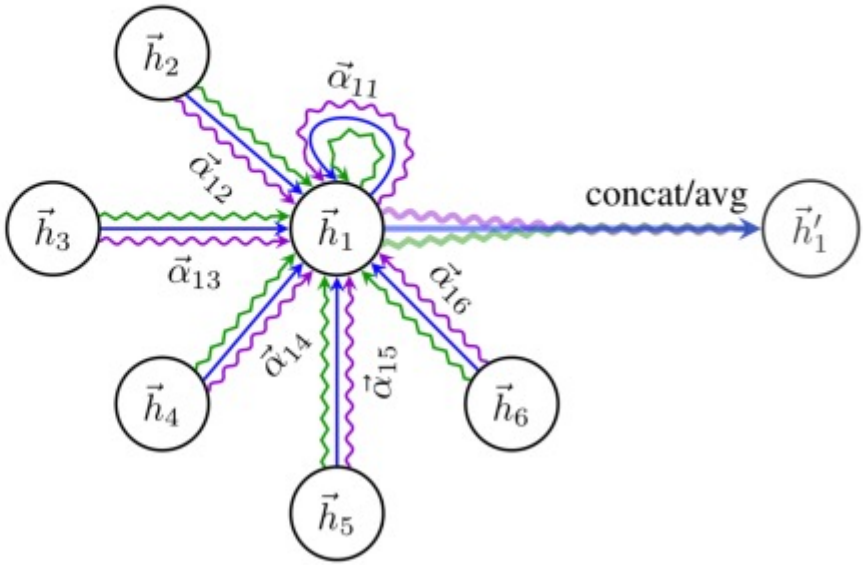
$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

Multi-head attention

Monti et al. (CVPR 2017), Hoshen (NIPS 2017), Veličkovićet al. (ICLR 2018)

Graph neural networks with attention





Pros:

- No need to store intermediate edge-based activation vectors (when using dot-product attn.)
- Slower than GCNs but faster than GNNs with edge embeddings

Cons:

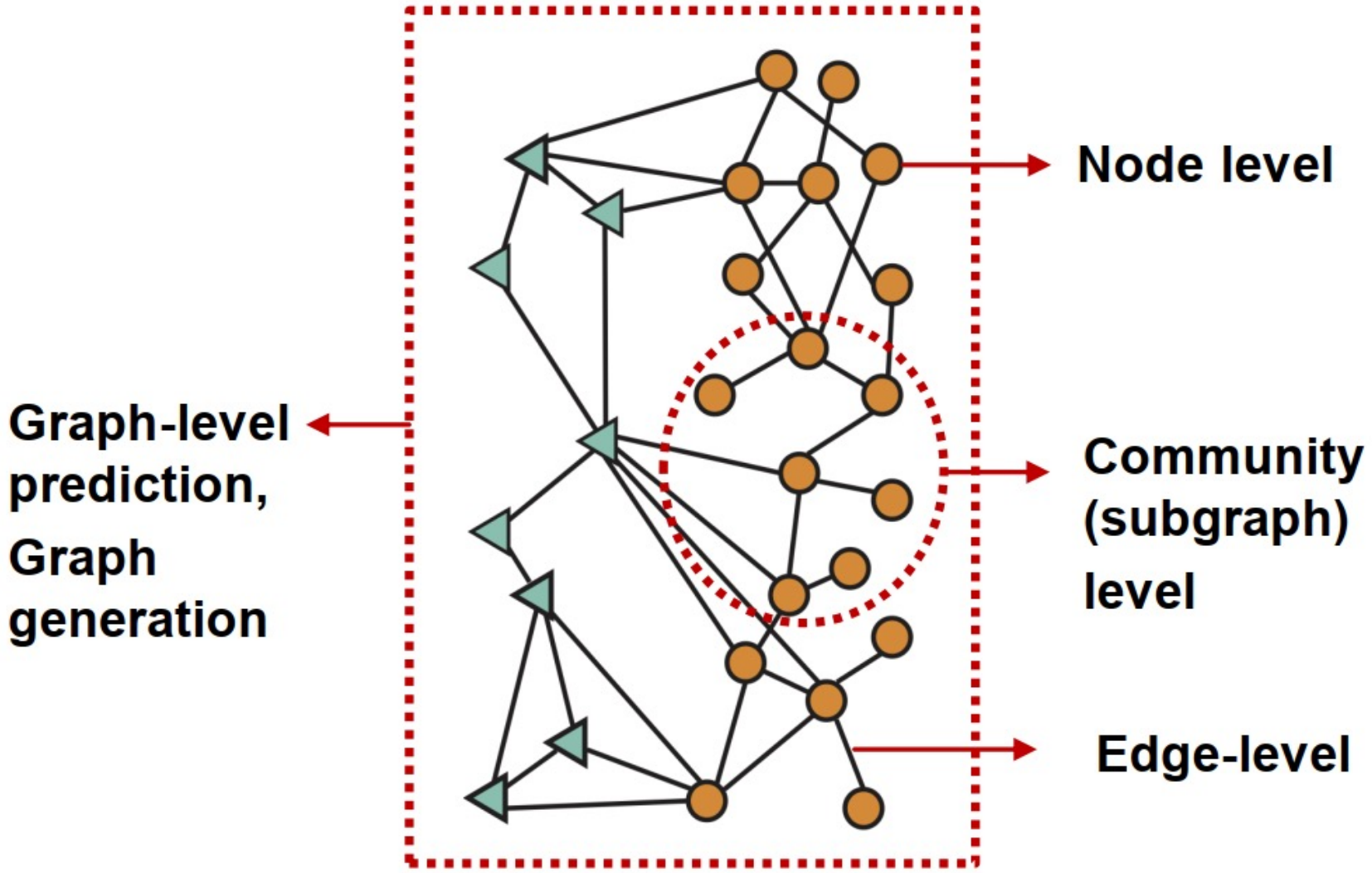
- (Most likely) less expressive than GNNs with edge embeddings
- Can be more difficult to optimize

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_k]\right)\right)}$$

Monti et al. (CVPR 2017), Hoshen (NIPS 2017), Veličkovićet al. (ICLR 2018)

Different types of tasks



Classic graph ML tasks

Node classification: Predict a property of a node

- **Example**: Categorize online users / items

Link prediction: Predict whether there are missing links between two nodes

- **Example**: Knowledge graph completion

Graph classification: Categorize different graphs

- **Example**: Molecule property prediction

Clustering: Detect if nodes form a community

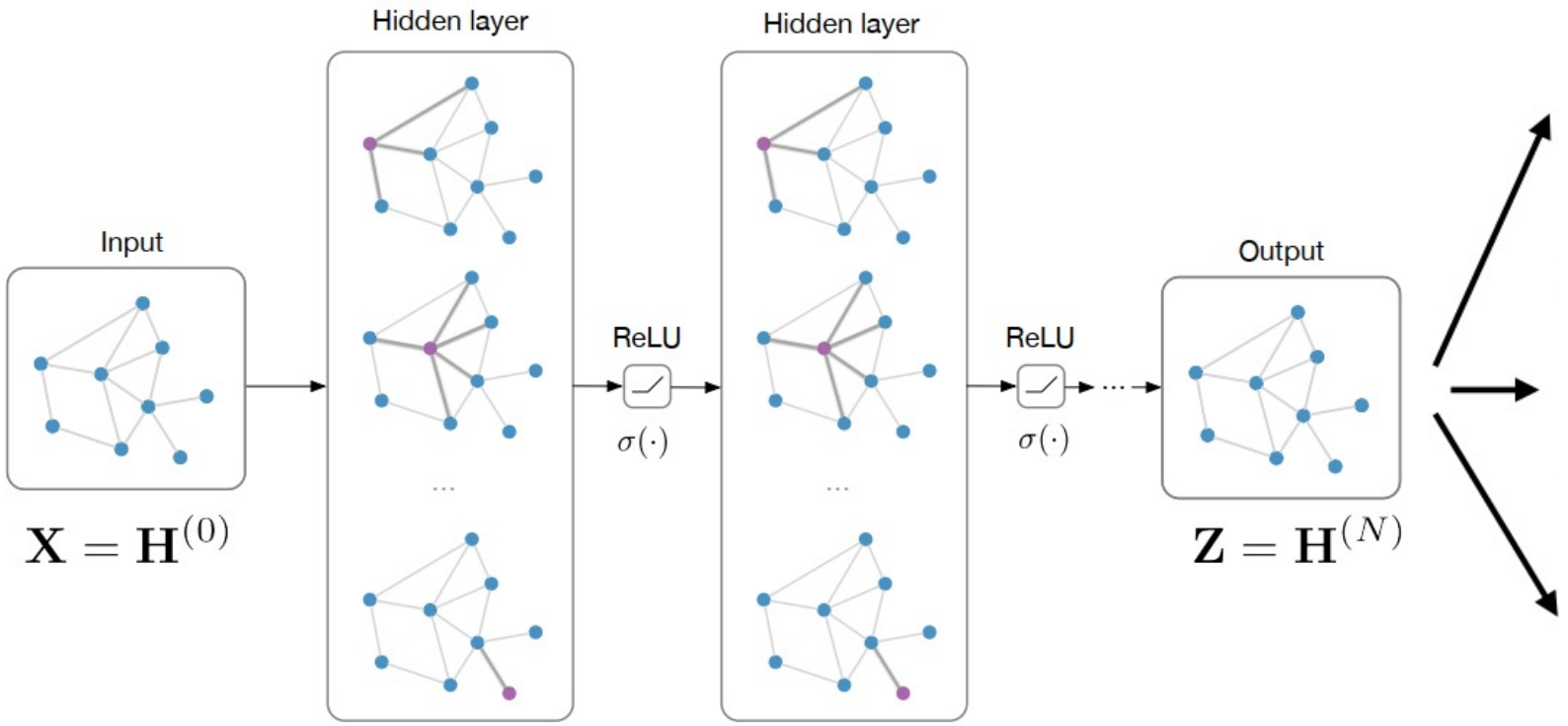
- **Example**: Social circle detection

Other tasks:

- Graph generation: Drug discovery
- Graph evolution: Physical simulation

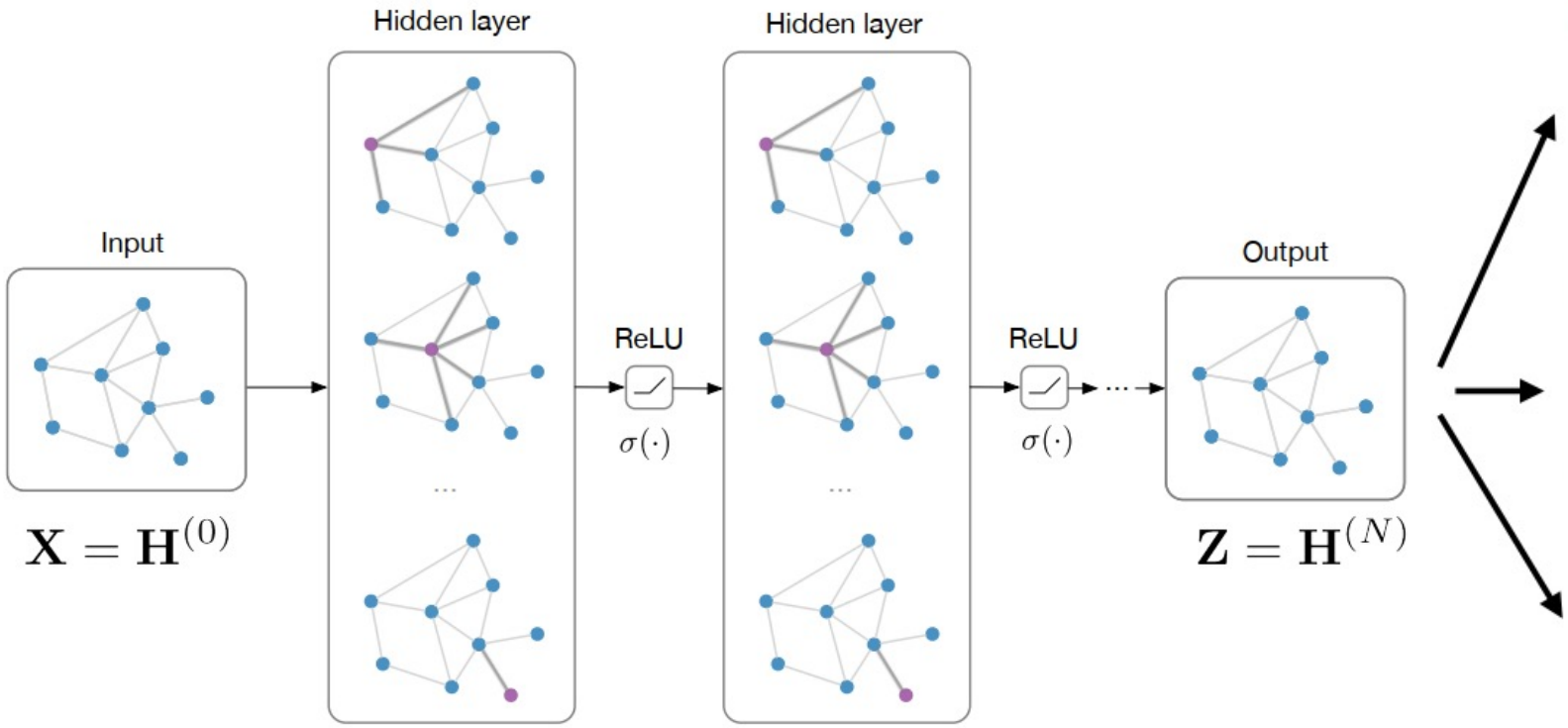
One fits all: Classification and link prediction with GNNs/GCNs

Input: Feature matrix $X \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{A}$



One fits all: Classification and link prediction with GNNs/GCNs

Input: Feature matrix $X \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{A}$



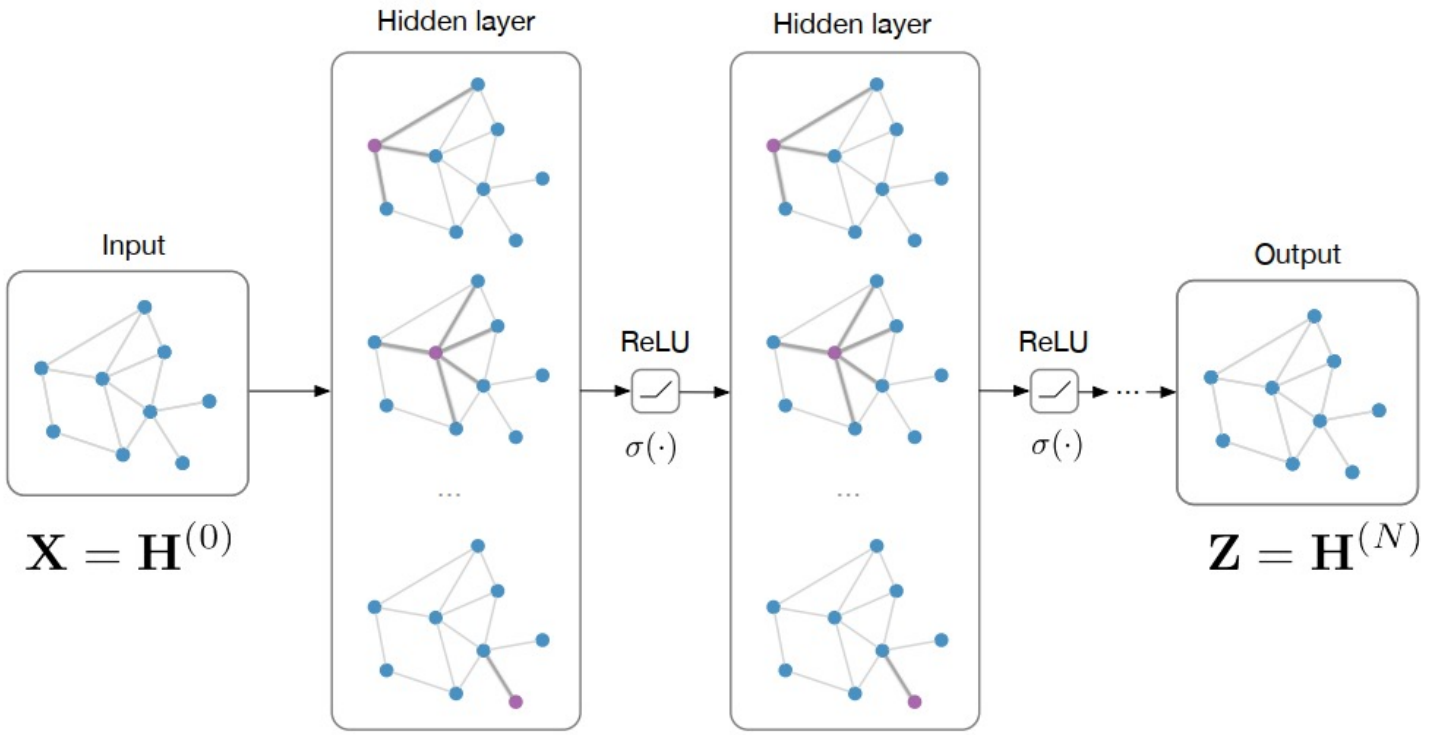
Node classification:

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

e.g. Kipf & Welling (ICLR 2017)

One fits all: Classification and link prediction with GNNs/GCNs

Input: Feature matrix $X \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{A}$



Node classification:

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

e.g. Kipf & Welling (ICLR 2017)

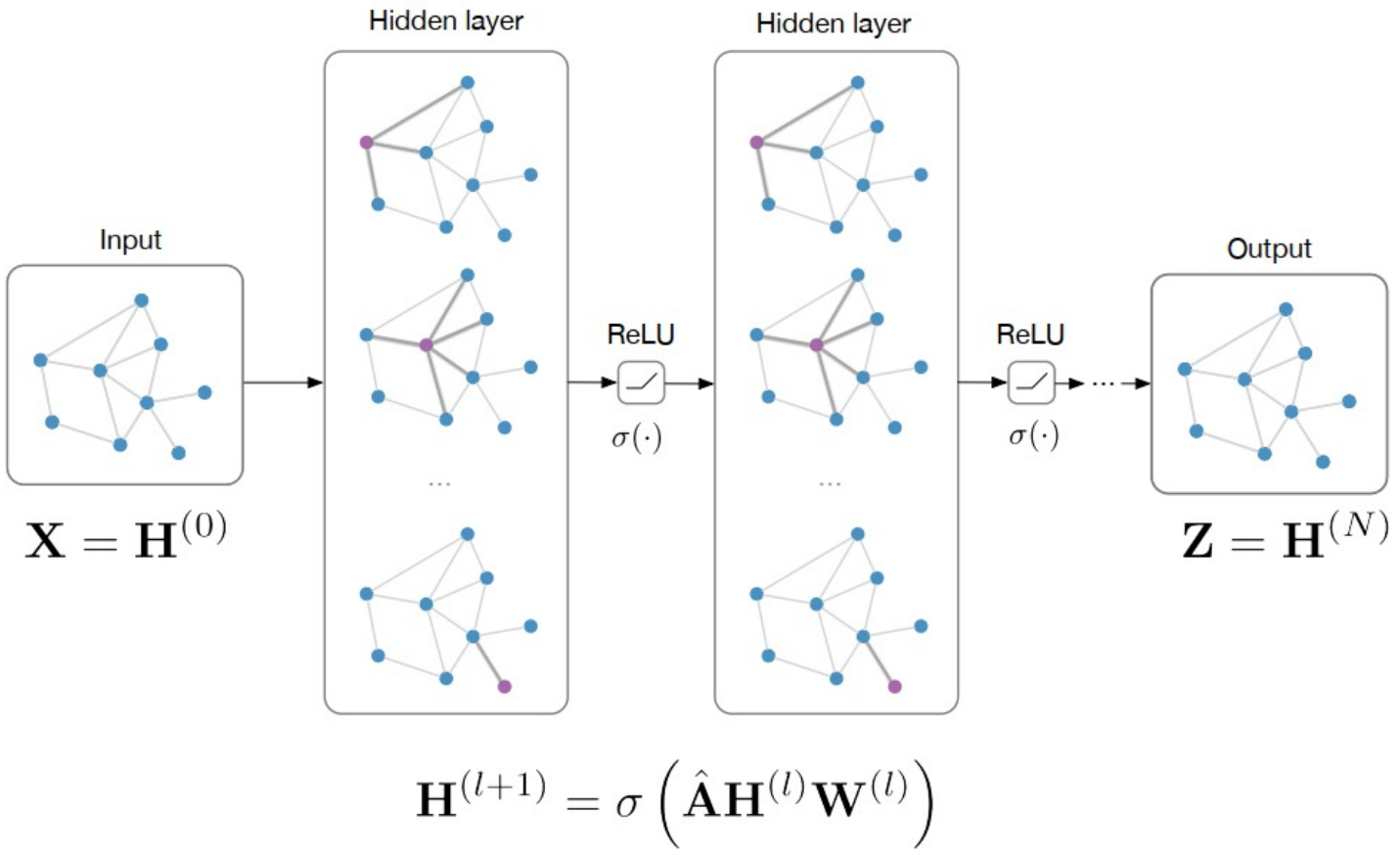
Graph classification:

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

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

One fits all: Classification and link prediction with GNNs/GCNs

Input: Feature matrix $X \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{A}$



Node classification:

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

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

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

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

Link prediction:

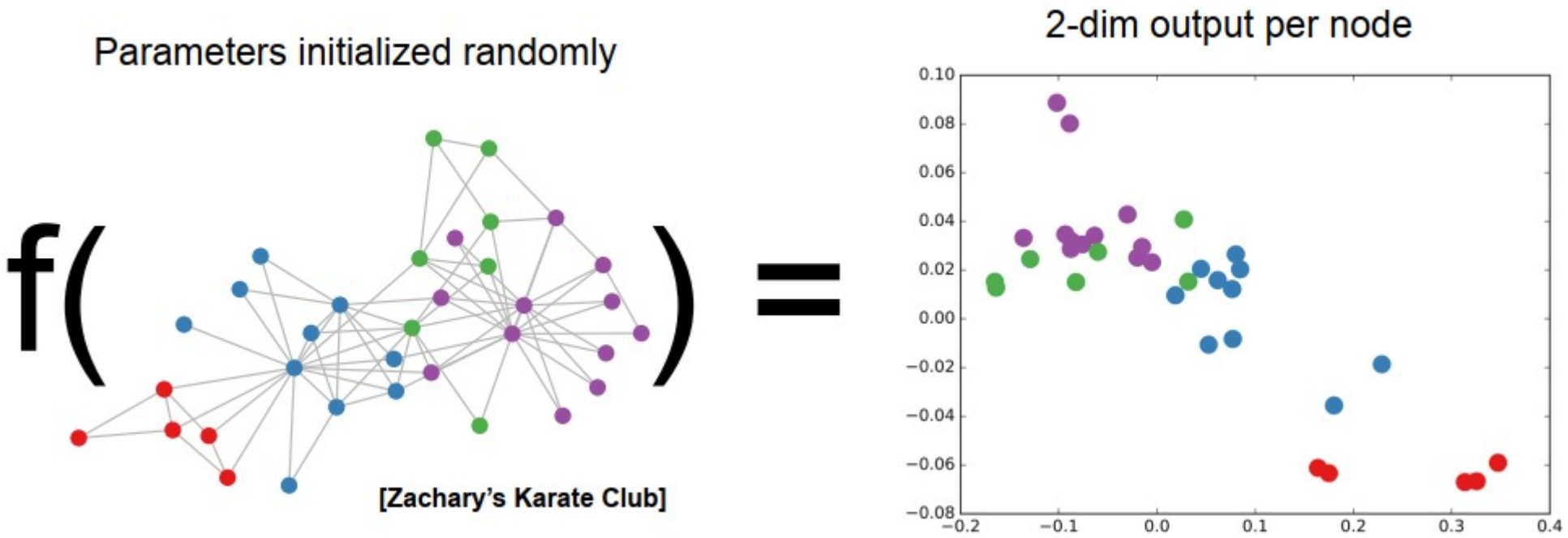
$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$

Kipf & Welling (NIPS BDL 2016)

"Graph Auto-Encoders"

Visualization of representations

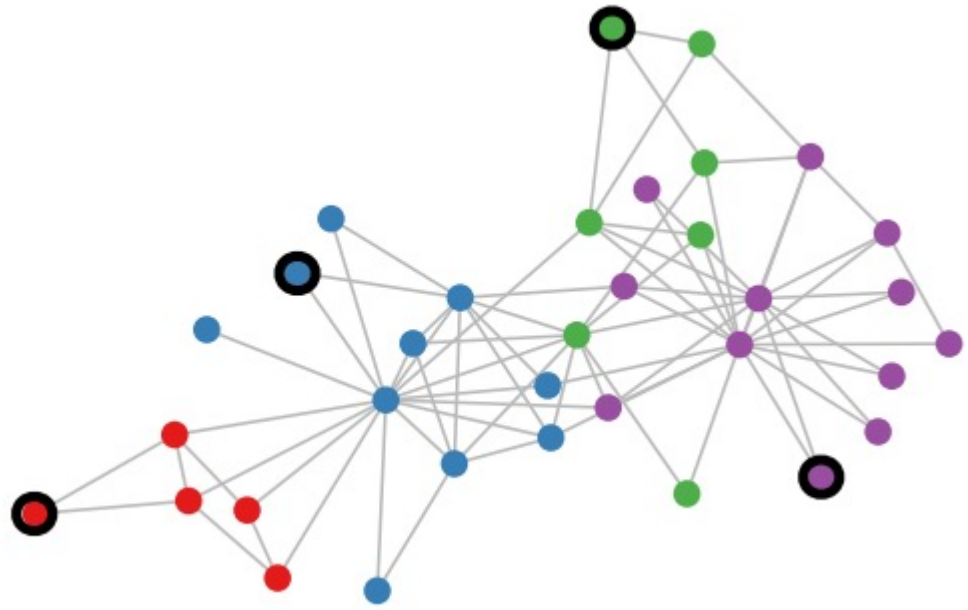
Forward pass through untrained 3-layer GCN model



Semi-supervised classification on graphs

Setting: Some nodes are labeled (black circle) All other nodes are unlabeled

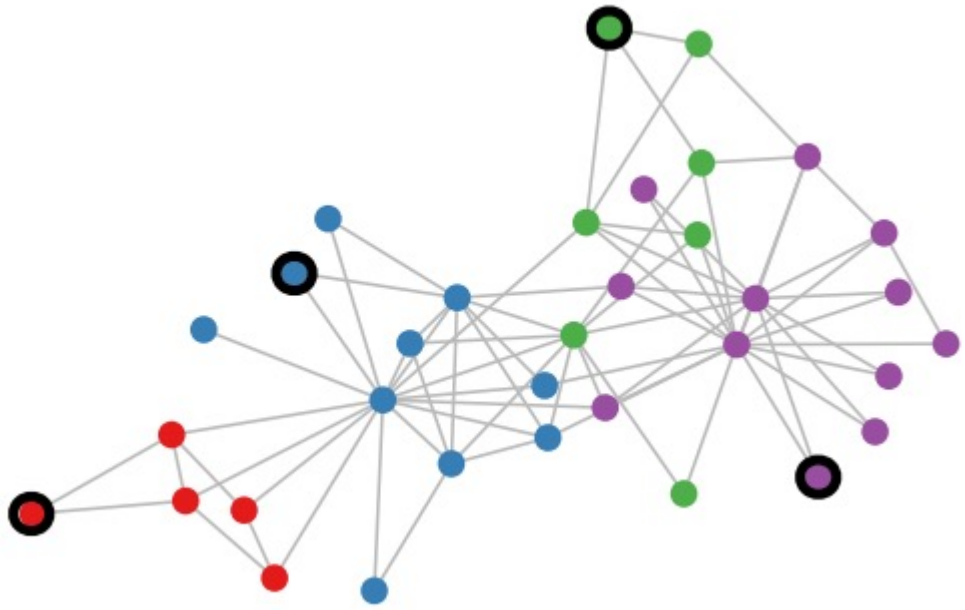
Task: Predict node label of unlabeled nodes



Semi-supervised classification on graphs

Setting: Some nodes are labeled (black circle) All other nodes are unlabeled

Task: Predict node label of unlabeled nodes



Evaluate loss on labeled nodes only:

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

$\mathcal{Y}_L$ set of labeled node indices

$\mathbf{Y}$ label matrix

$\mathbf{Z}$ GCN output (after softmax)

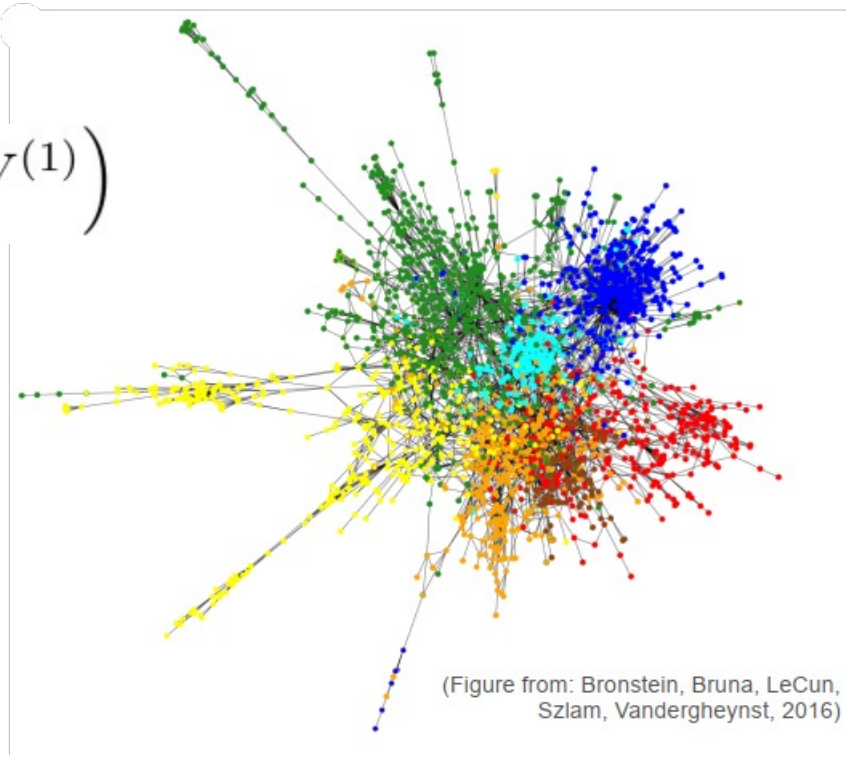
Application: Classification on citation networks

Input: Citation networks (nodes are papers, edges are citation links, optionally bag-of-words features on nodes)

Target: Paper category (e.g. stat.ML, cs.LG, …)

Model: 2-layer GCN

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\ W^{(1)}\Big)$$



(Figure from: Bronstein, Bruna, LeCun, Szlam, Vandergheynst, 2016)

Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017

Application: Classification on citation networks

Input: Citation networks (nodes are papers, edges are citation links, optionally bag-of-words features on nodes)

Target: Paper category (e.g. stat.ML, cs.LG, …)

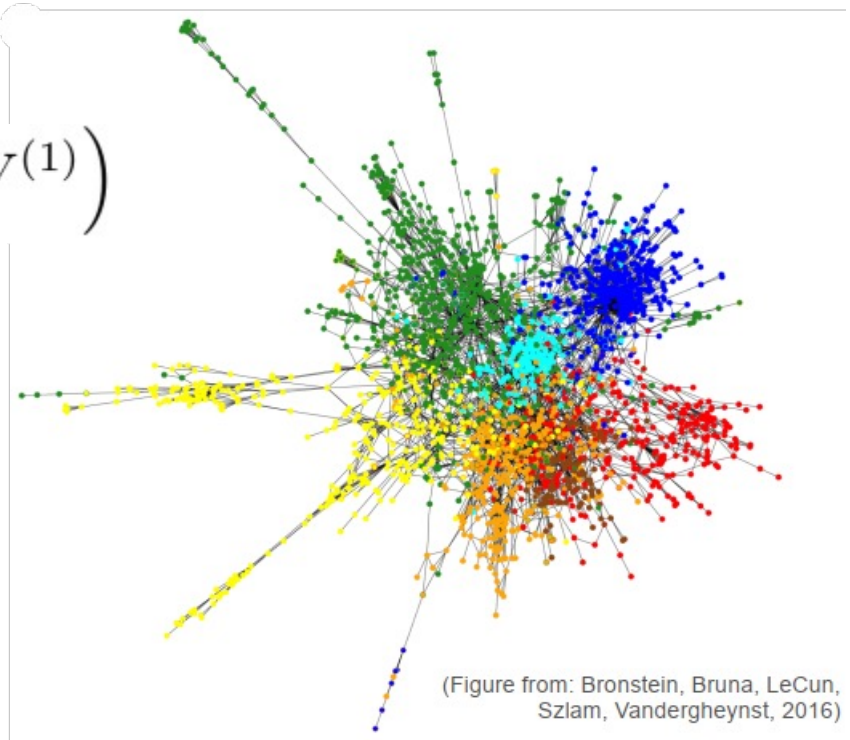
Model: 2-layer GCN

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\ W^{(1)}\Big)$$

Classification results (accuracy)

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [24]	59.6	59.0	71.1	26.7
LP [27]	45.3	68.0	63.0	26.5
DeepWalk [18]	43.2	67.2	65.3	58.1
Planetoid* [25]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

no input features

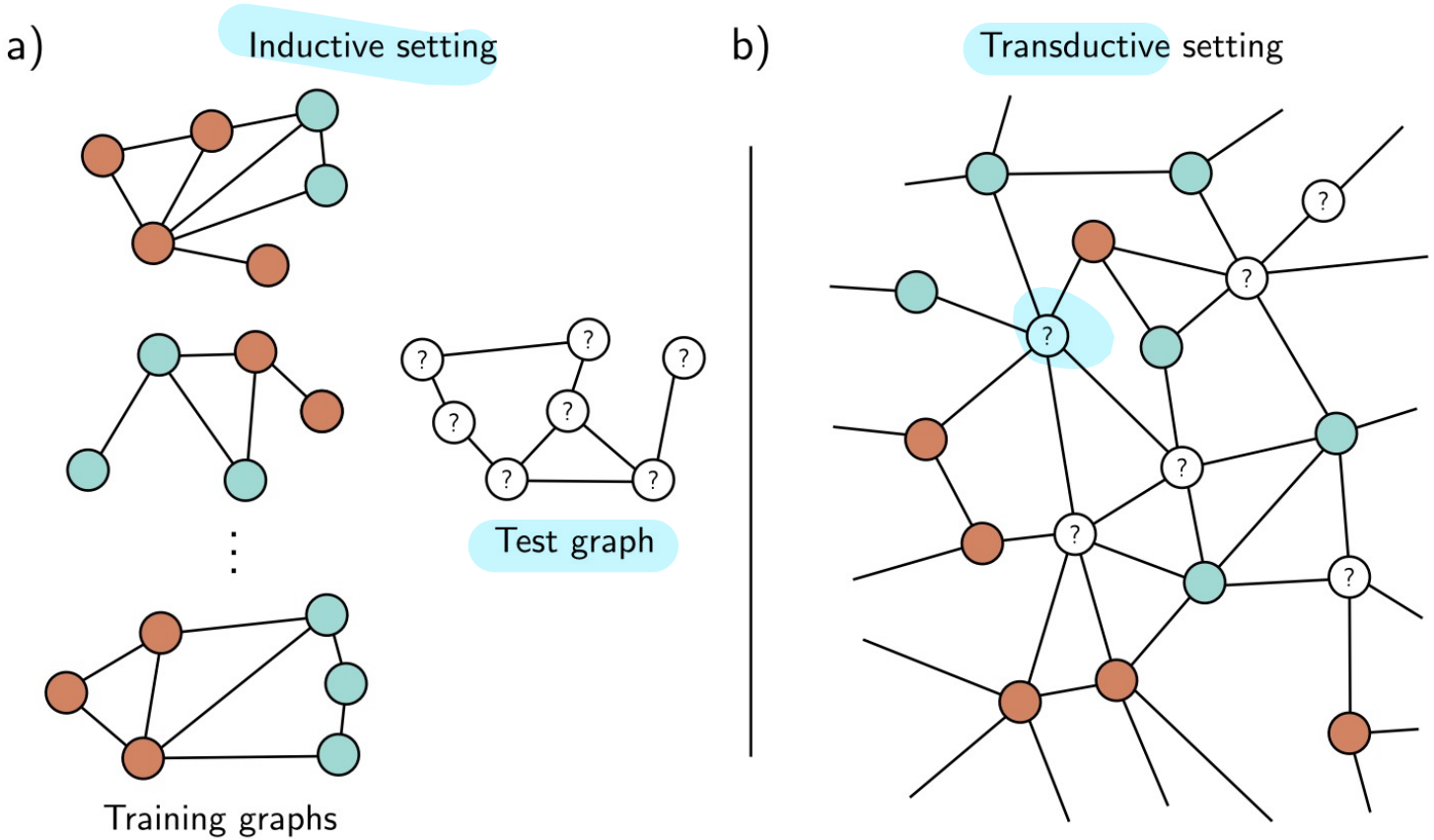


(Figure from: Bronstein, Bruna, LeCun, Szlam, Vandergheynst, 2016)

Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017

More on Training of GNNs

Inductive vs. transductive problems



Price, Understanding Deep Learning, 2023

Training with batches

- Each graph may have a different number of nodes
  - the matrices X_i and A_i have different sizes

- How to process the whole batch in parallel?
  - The graphs in the batch can be treated as disjoint components of a single large graph.
  - The network can then be run as a single instance of the network equations.
  - The mean pooling is carried out only over the individual graphs to make a single representation per graph

How to form batches

- Choose a random subset of labeled nodes at each training step

- Two approaches:
  - Neighborhood sampling
    - start with the batch nodes and randomly sample a fixed number of their neighbors
  - Graph partitioning
    - cluster the original graph into disjoint subsets of nodes
    - smaller graphs can each be treated as batches, or a random subset of them can be combined to form a batch

Price, Understanding Deep Learning, 2023

Oversmoothing

- Node-embedding vectors tend to become very similar to each other after a number of iterations of message-passing
  - effectively limits the depth of the network

- Alleviating oversmoothing:
  - Residual connections
  - jump knowledge connections: allow the output layer to take information from all previous layers of the network