Deep Learning on Graphs

fuyw

2019.10

Outline

- Intro
- Graph Neural Network
- Summary

Deep Learning on Graphs: A Survey

Ziwei Zhang, Peng Cui and Wenwu Zhu

Abstract—Deep learning has been shown successful in a number of domains, ranging from acoustics, images to natural language processing. However, applying deep learning to the ubiquitous graph data is non-trivial because of the unique characteristics of graphs. Recently, a significant amount of research efforts have been devoted to this area, greatly advancing graph analyzing techniques. In this survey, we comprehensively review different kinds of deep learning methods applied to graphs. We divide existing methods into three main categories: semi-supervised methods including Graph Neural Networks and Graph Convolutional Networks, unsupervised methods including Graph Autoencoders, and recent advancements including Graph Recurrent Neural Networks and Graph Reinforcement Learning. We then provide a comprehensive overview of these methods in a systematic manner following their history of developments. We also analyze the differences of these methods and how to composite different architectures. Finally, we briefly outline their applications and discuss potential future directions.

Index Terms—Graph Data, Deep Learning, Graph Neural Network, Graph Convolutional Network, Graph Autoencoder.

2018

Zhang, Ziwei, Peng Cui, and Wenwu Zhu. "Deep learning on graphs: A survey." *arXiv preprint arXiv:1812.04202* (2018).

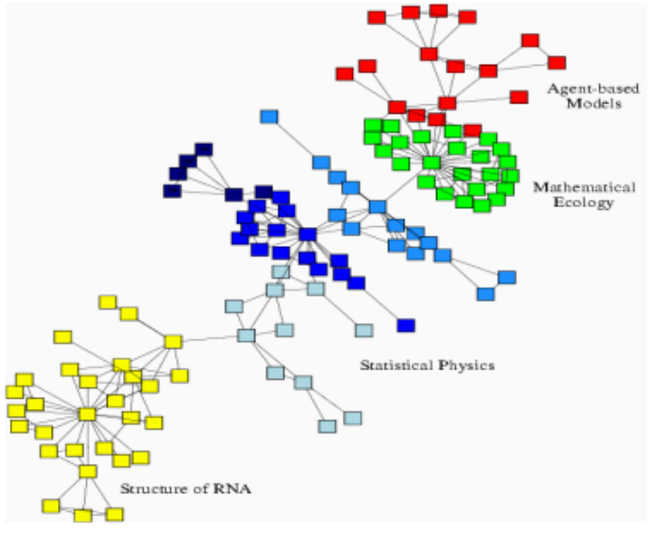
A Comprehensive Survey on Graph Neural Networks

Zonghan Wu, Shirui Pan, *Member, IEEE*, Fengwen Chen, Guodong Long,
Chengqi Zhang, *Senior Member, IEEE*, Philip S. Yu, *Fellow, IEEE*

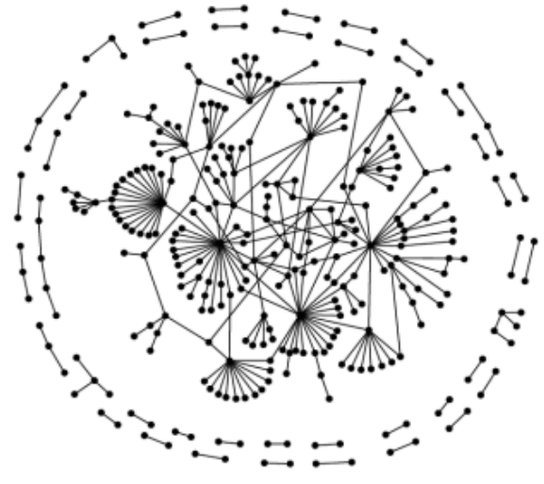
Wu, Zonghan, et al. "A comprehensive survey on graph neural networks." *arXiv preprint arXiv:1901.00596* (2019).

Many Data are Networks

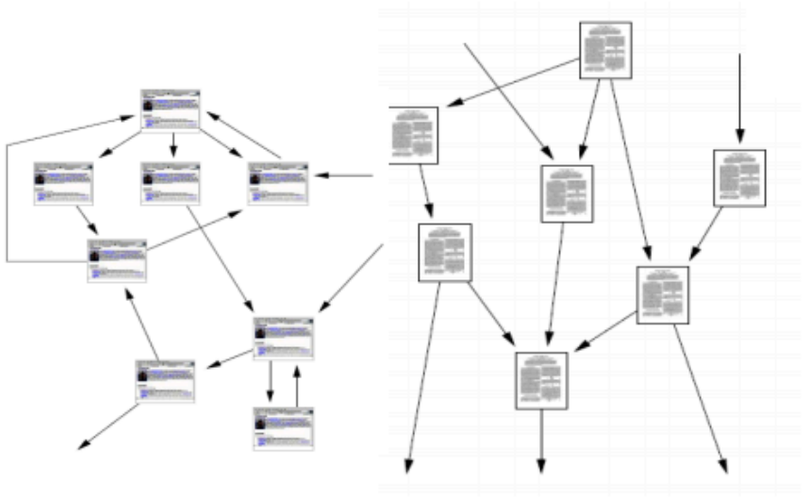
Social networks



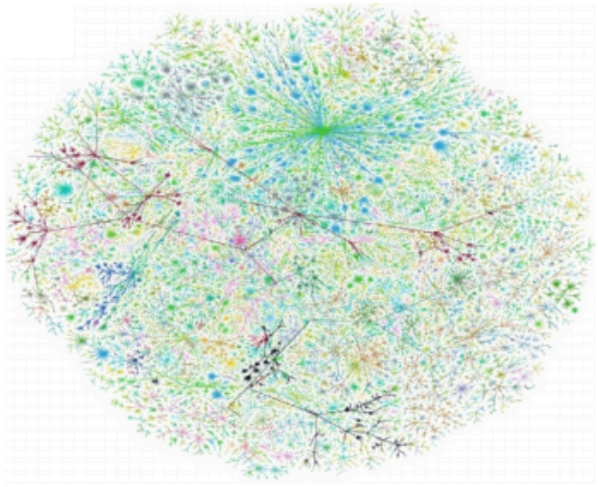
Economic networks



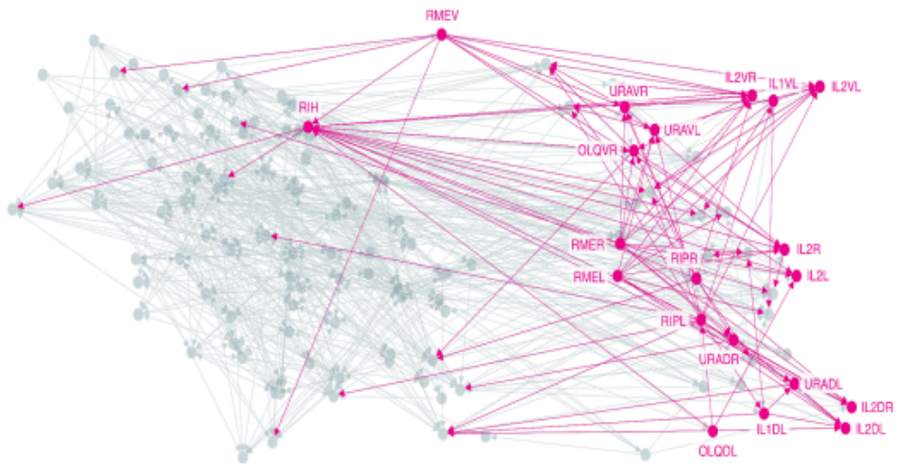
Biomedical networks



Information networks:
Web & citations



Internet



Networks of neurons

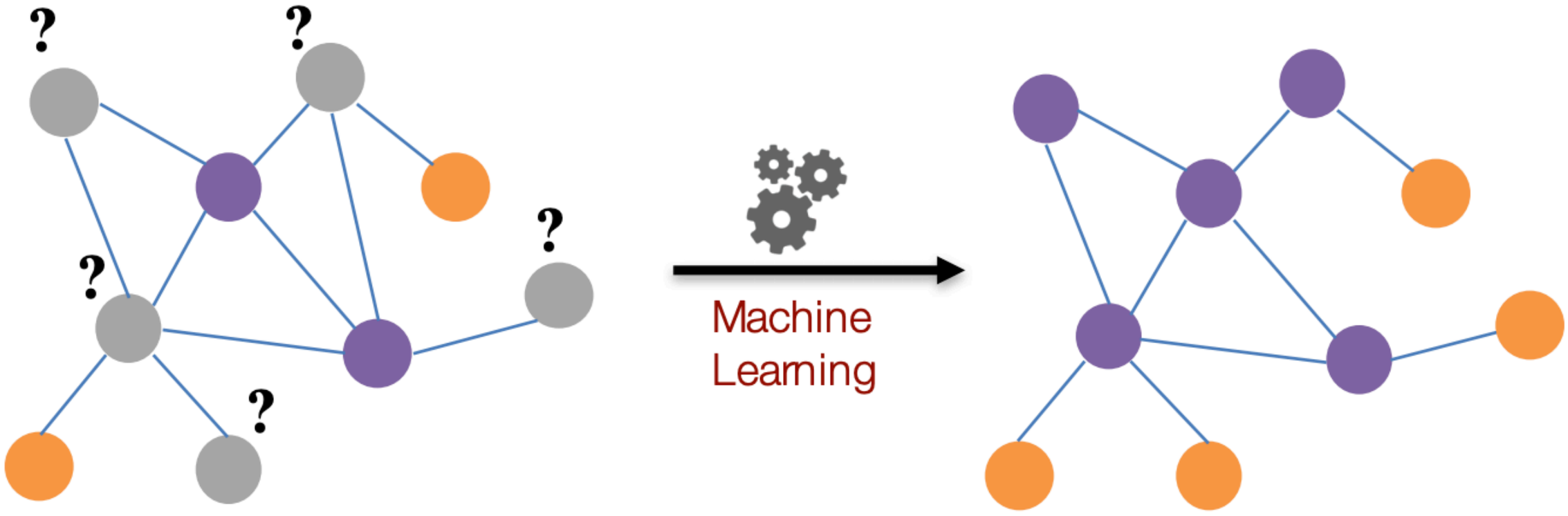
Why Graphs? Why Now?

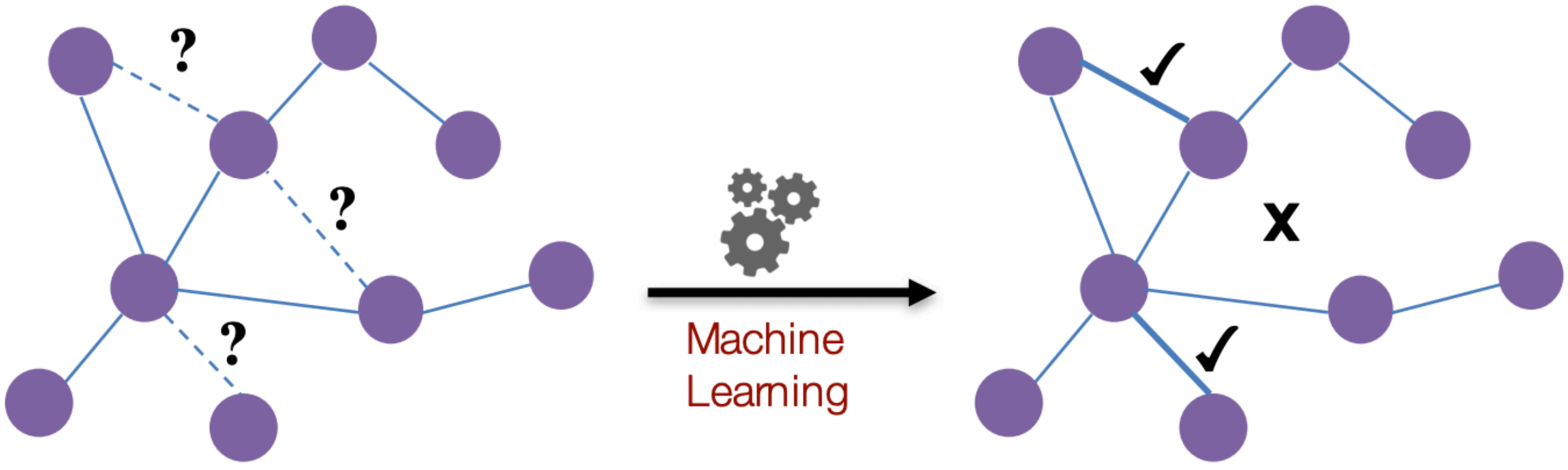
- Universal language for describing complex data
  - Networks/graphs from science, nature, and technology are more similar than one would expect
- Shared vocabulary between fields
  - Computer Science, Social science, Physics, Economics, Statistics, Biology
- Data availability (+computational challenges)
  - Web/mobile, bio, health, and medical
- Impact!
  - Social networking, Social media, Drug design

Machine Learning with Graphs

Classical ML tasks in graphs:

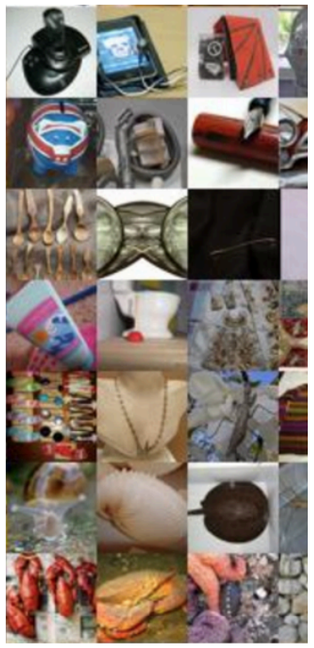
- Node classification
  - Predict a type of a given node
- Link prediction
  - Predict whether two nodes are linked
- Community detection
  - Identify densely linked clusters of nodes
- Network similarity
  - How similar are two (sub)networks


?
?
?
?
?
Machine
Learning


?
?
?
✓
X
✓
Machine
Learning

Why Is It Hard?

IMAGENET



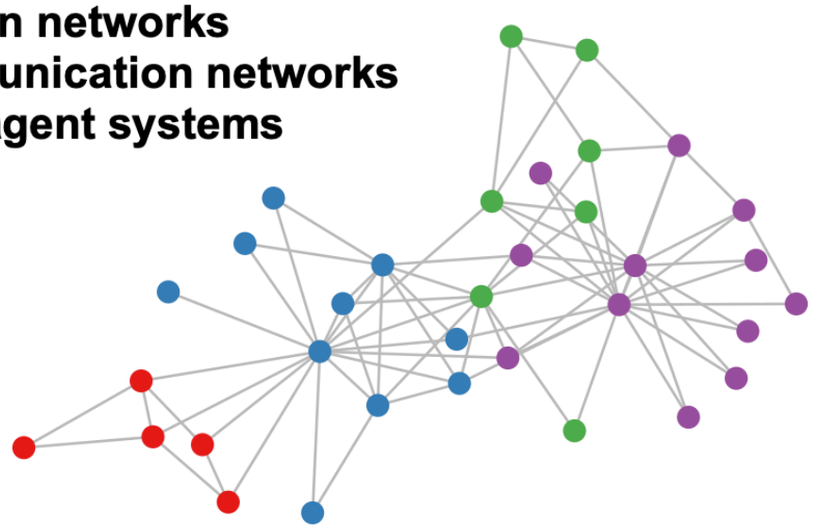
Speech data

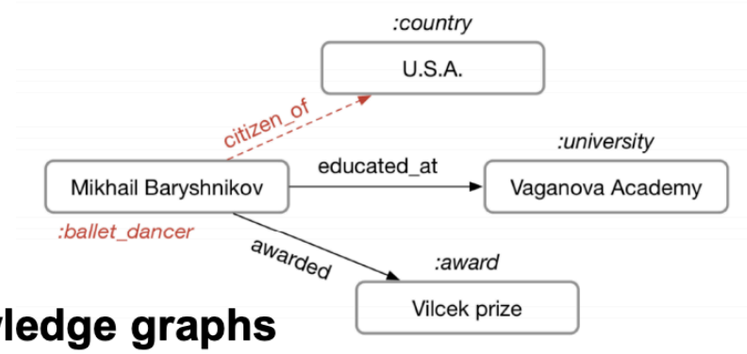
Grid games

Graph-structured data

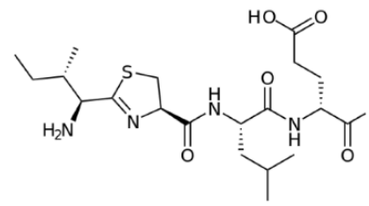
A lot of real-world data does not “live” on grids

Social networks
Citation networks
Communication networks
Multi-agent systems





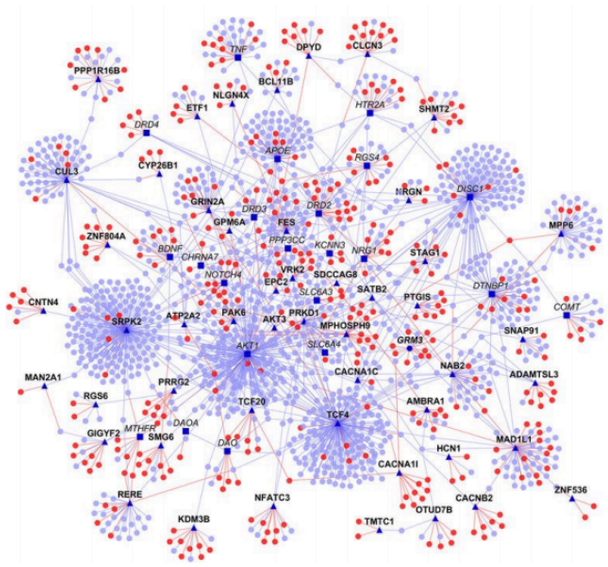
Knowledge graphs



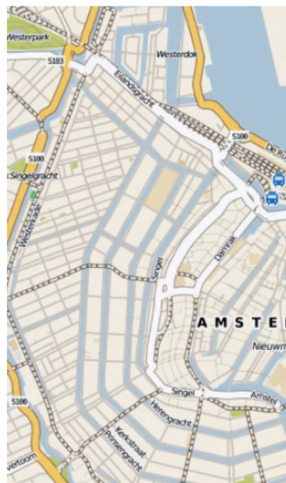
Molec

Deep neu

- translatio
- hierarchi



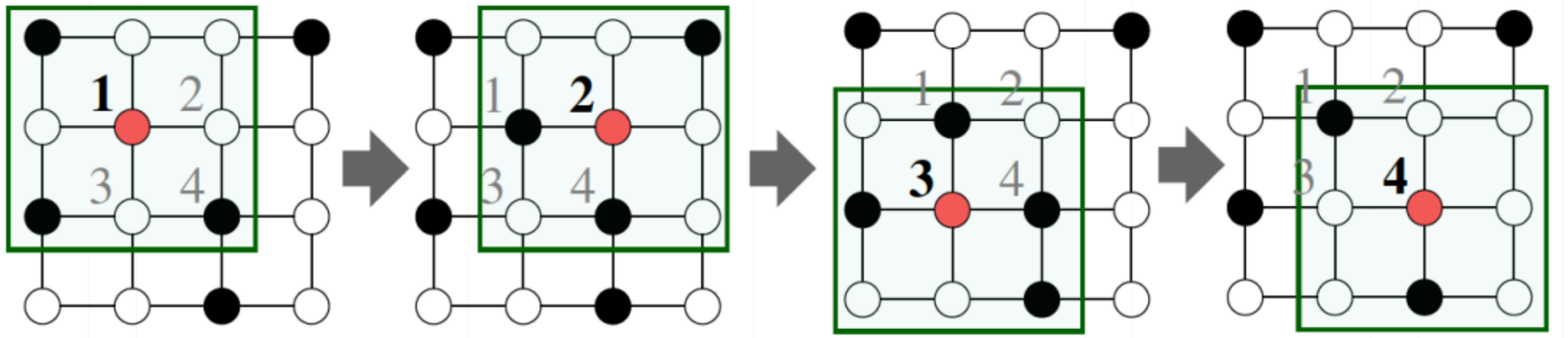
Protein interaction networks



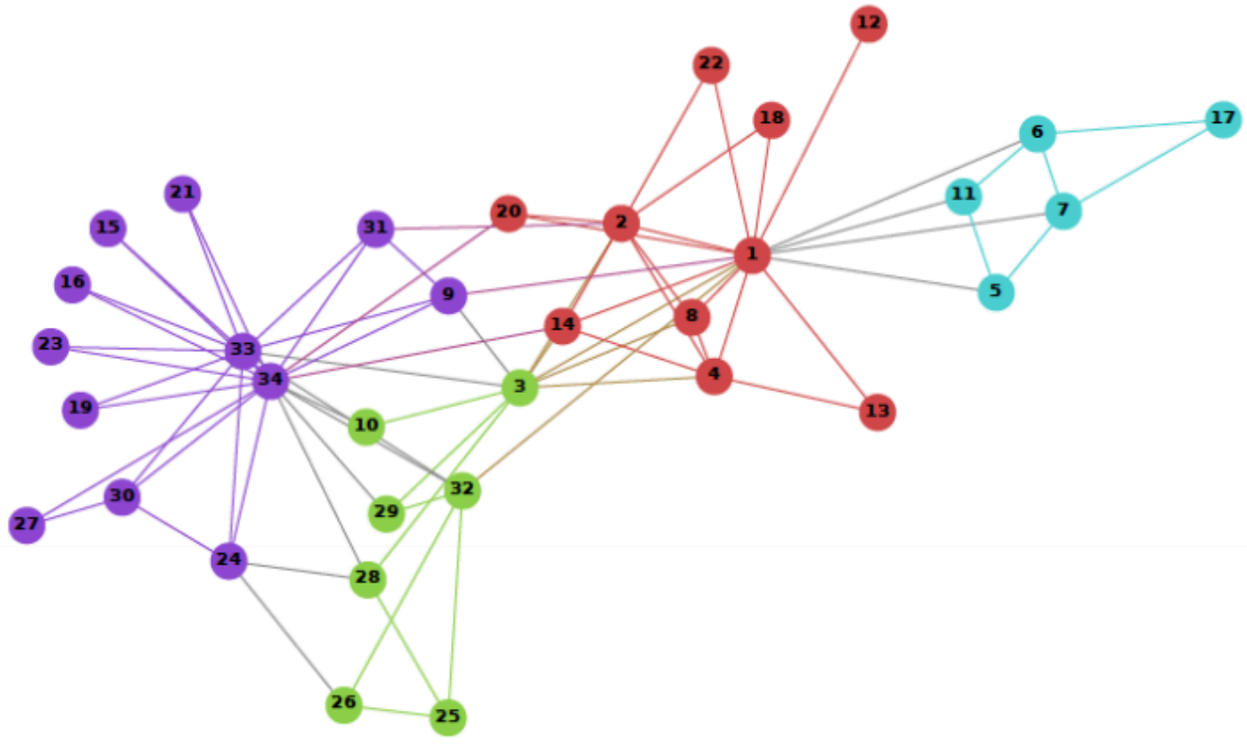
Standard deep learning architectures like CNNs and RNNs don’t work here!

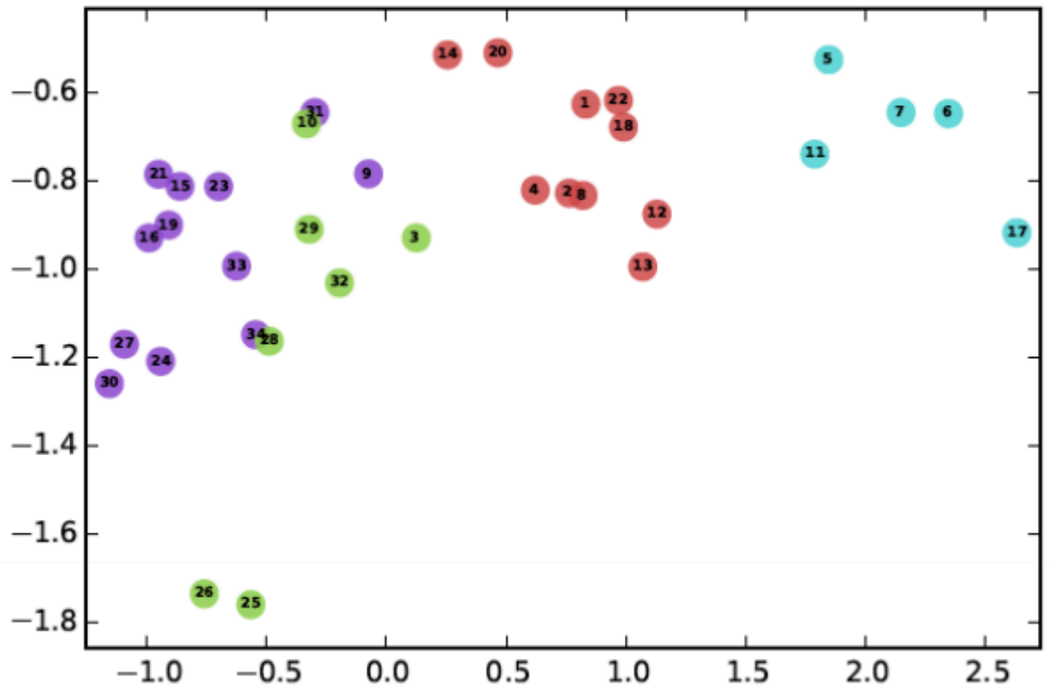
Why Is It Hard?

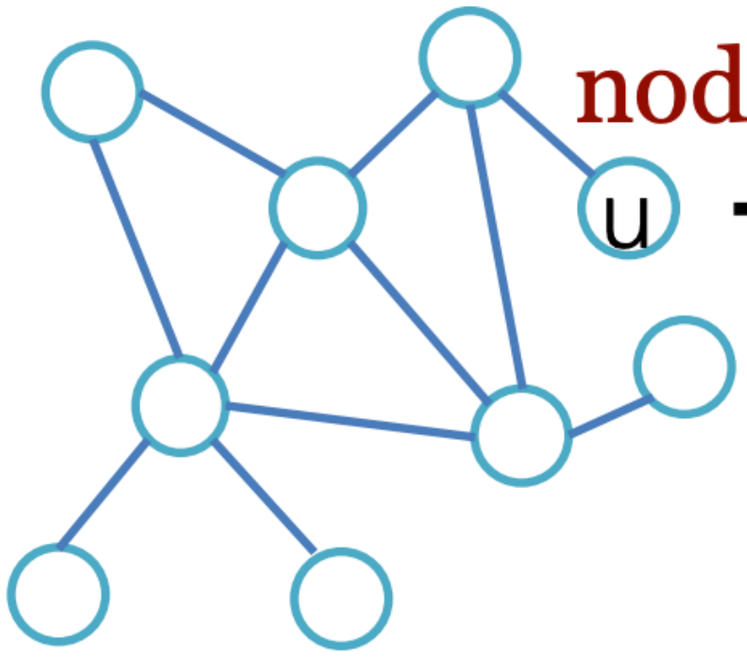
- But graphs are far more complex!
  - Complex topographical structure (i.e., no spatial locality like grids)

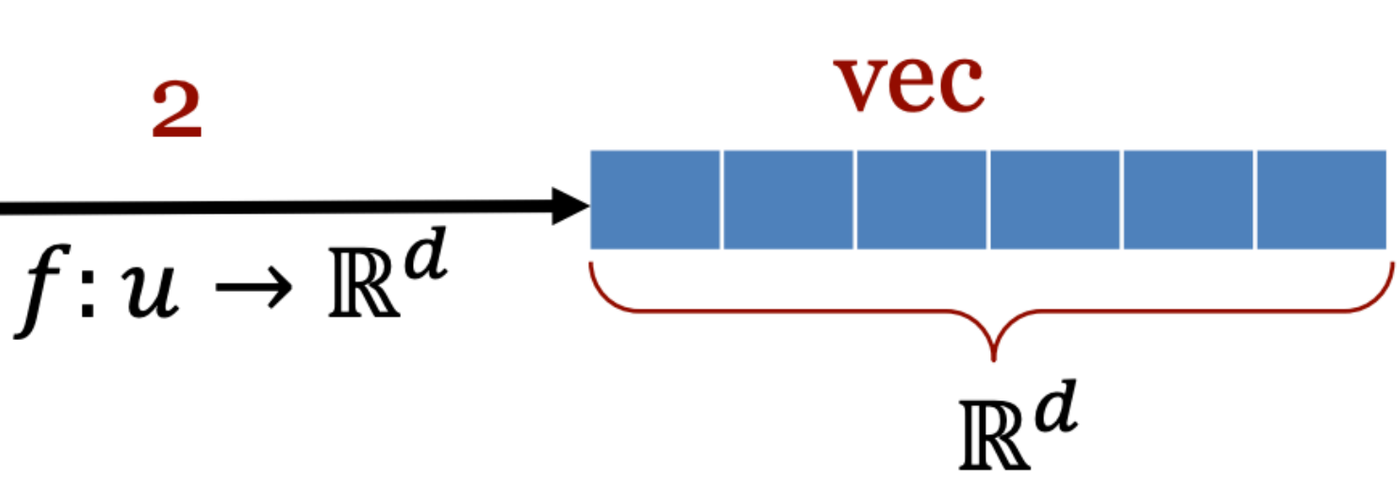


  - No fixed node ordering or reference point (i.e., the isomorphism problem)
  - Often dynamic and have multimodal features.


12
22
18
17
6
21
20
2
11
7
15
31
1
16
9
5
14
8
23
33
34
3
4
19
13
10
32
30
29
27
24
28
26
25


14
20
5
1
22
18
7
6
11
21
15
23
9
4
2
8
12
16
19
29
3
17
33
32
13
34
28
27
24
30
26
25
−0.6
−0.8
−1.0
−1.2
−1.4
−1.6
−1.8
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


node
u


2
$f: u \rightarrow \mathbb{R}^d$
vec
$\mathbb{R}^d$
Feature representation,
embedding

Two Key Components

- **Encoder** maps each node to a low-dimensional vector.

$$\text{ENC}(v) = \mathbf{z}_v$$

$\mathbf{d}$-dimensional embedding

node in the input graph

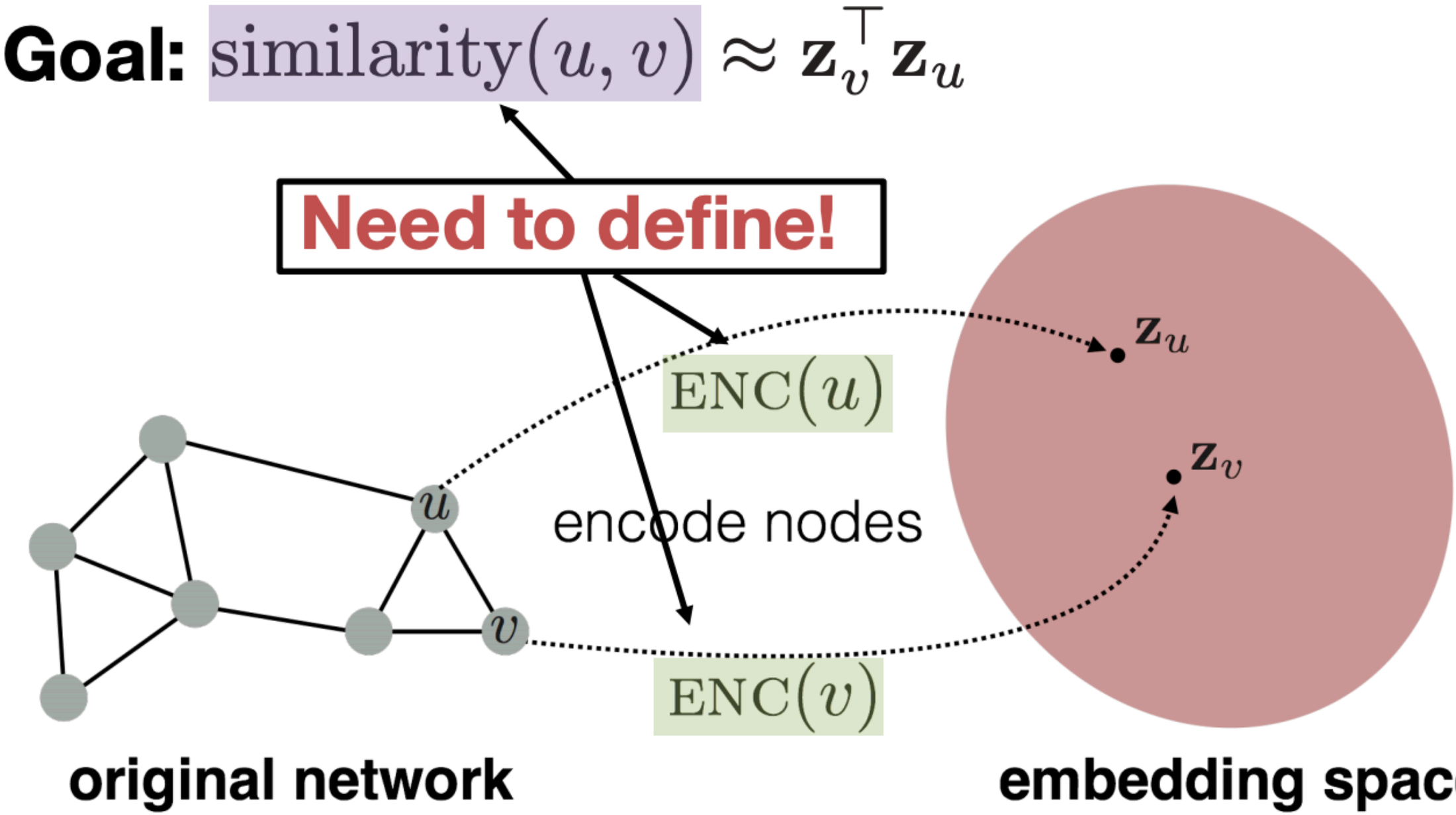
- **Similarity function** specifies how relationships in vector space map to relationships in the original network.

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Similarity of u and v in the original network

dot product between node embeddings

Similarity in the embedding space approximates similarity in the original network.



How to Define Node Similarity?

- Key distinction between "shallow" methods is **how they define node similarity.**

- E.g., should two nodes have similar embeddings if they….
  - are connected?
  - share neighbors?
  - have similar "structural roles"?
  - …?

Adjacency-based Similarity

- **Similarity function** is just the edge weight between u and v in the original network.
- **Intuition:** Dot products between node embeddings approximate edge existence.

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v} \|^2$$

loss (what we want to minimize)

sum over all node pairs

embedding similarity

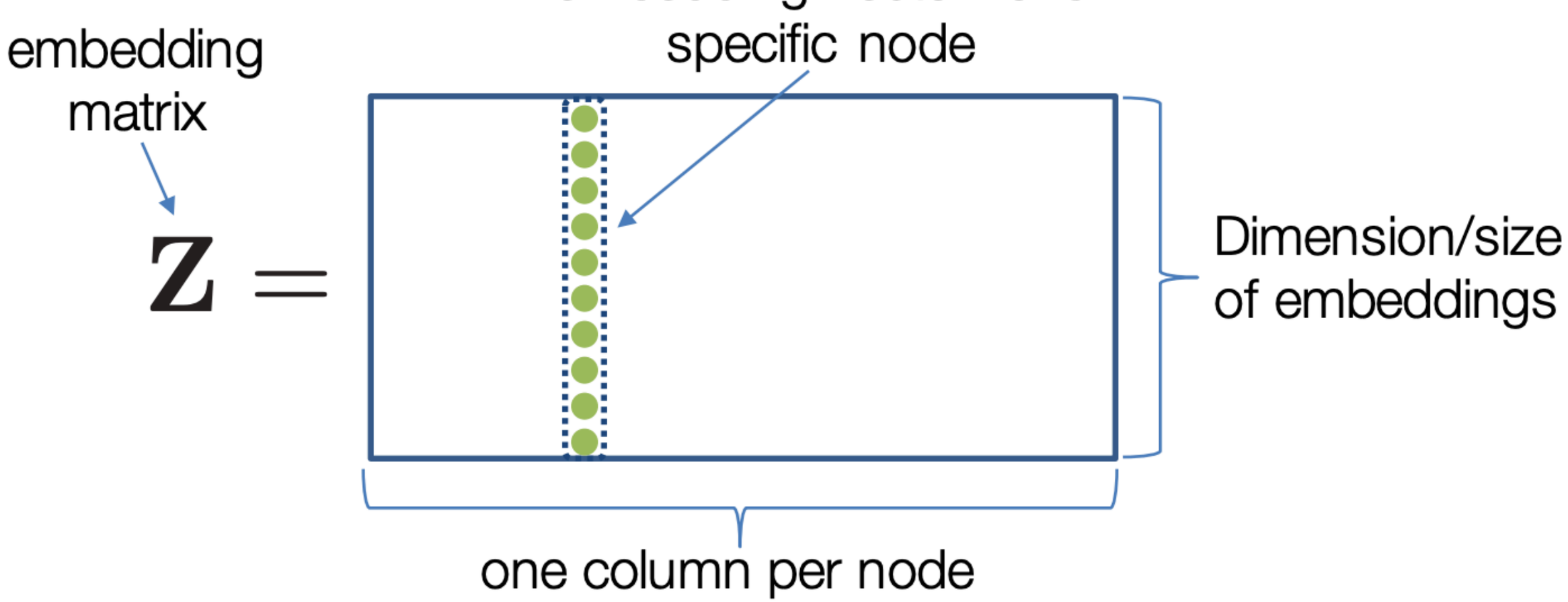
(weighted) adjacency matrix for the graph

$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is node embedding **[what we learn!]**

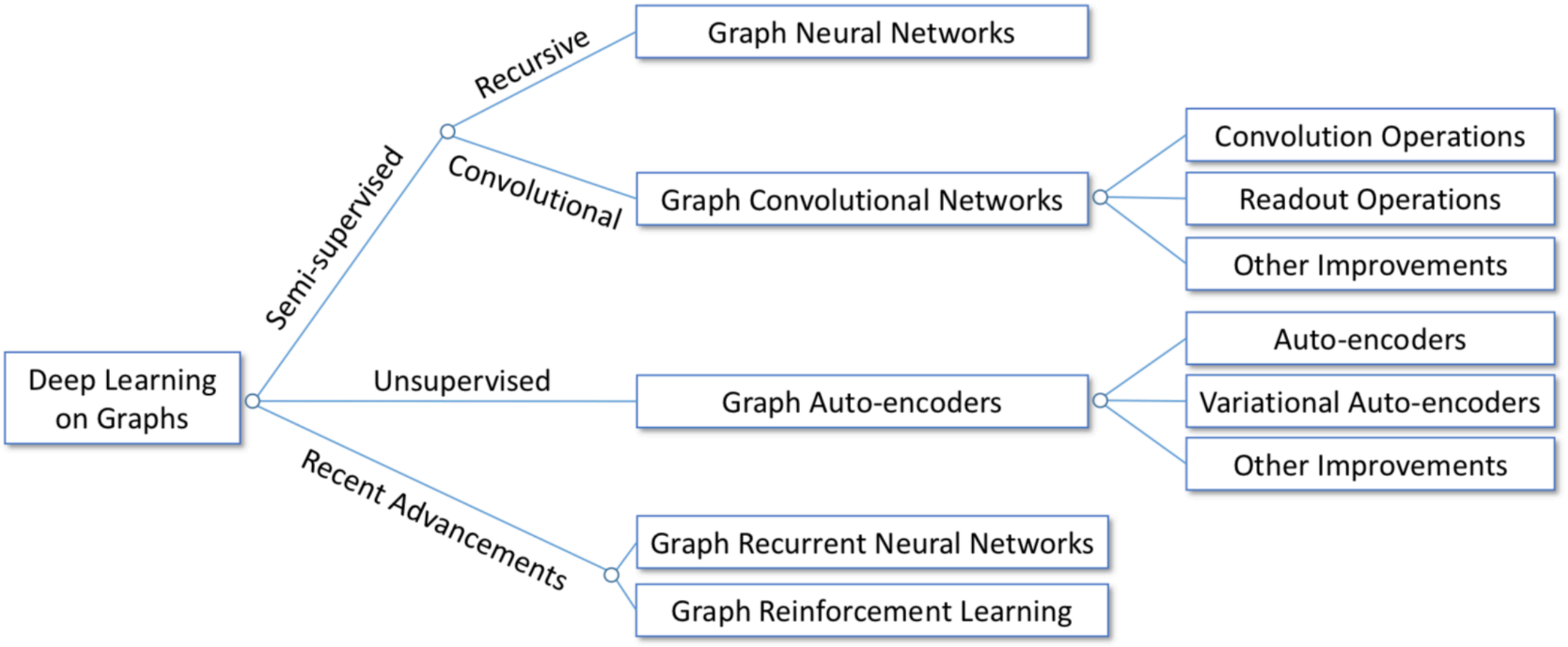
$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node *v*

node2vec
DeepWalk
LINE

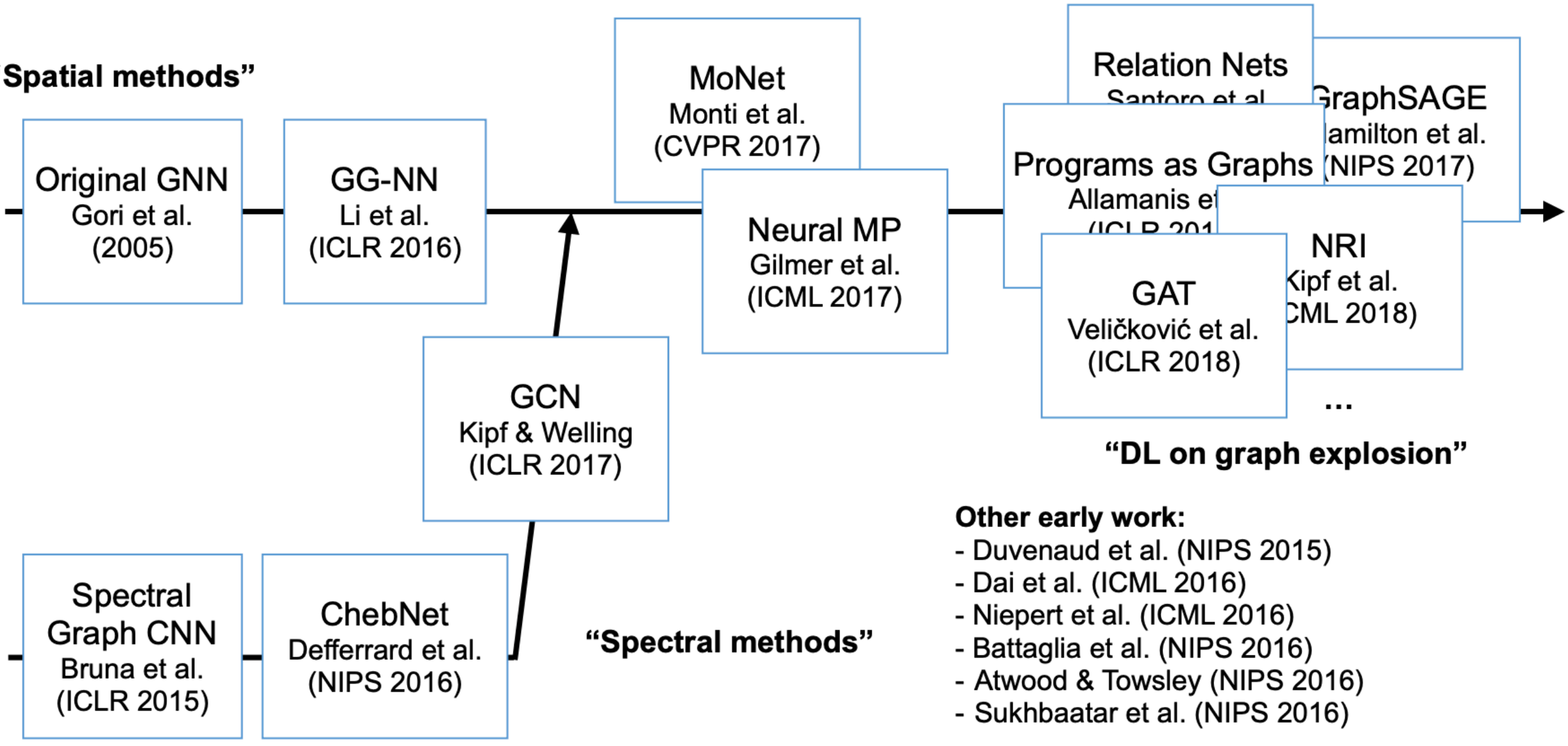


Outline

- Intro
- Graph Neural Network
    - graph neural network
- Summary


Deep Learning on Graphs
Semi-supervised
Recursive
Convolutional
Graph Neural Networks
Graph Convolutional Networks
Convolution Operations
Readout Operations
Other Improvements
Unsupervised
Graph Auto-encoders
Auto-encoders
Variational Auto-encoders
Other Improvements
Recent Advancements
Graph Recurrent Neural Networks
Graph Reinforcement Learning

A brief history of graph neural nets



Other early work:

- Duvenaud et al. (NIPS 2015)
- Dai et al. (ICML 2016)
- Niepert et al. (ICML 2016)
- Battaglia et al. (NIPS 2016)
- Atwood & Towsley (NIPS 2016)
- Sukhbaatar et al. (NIPS 2016)

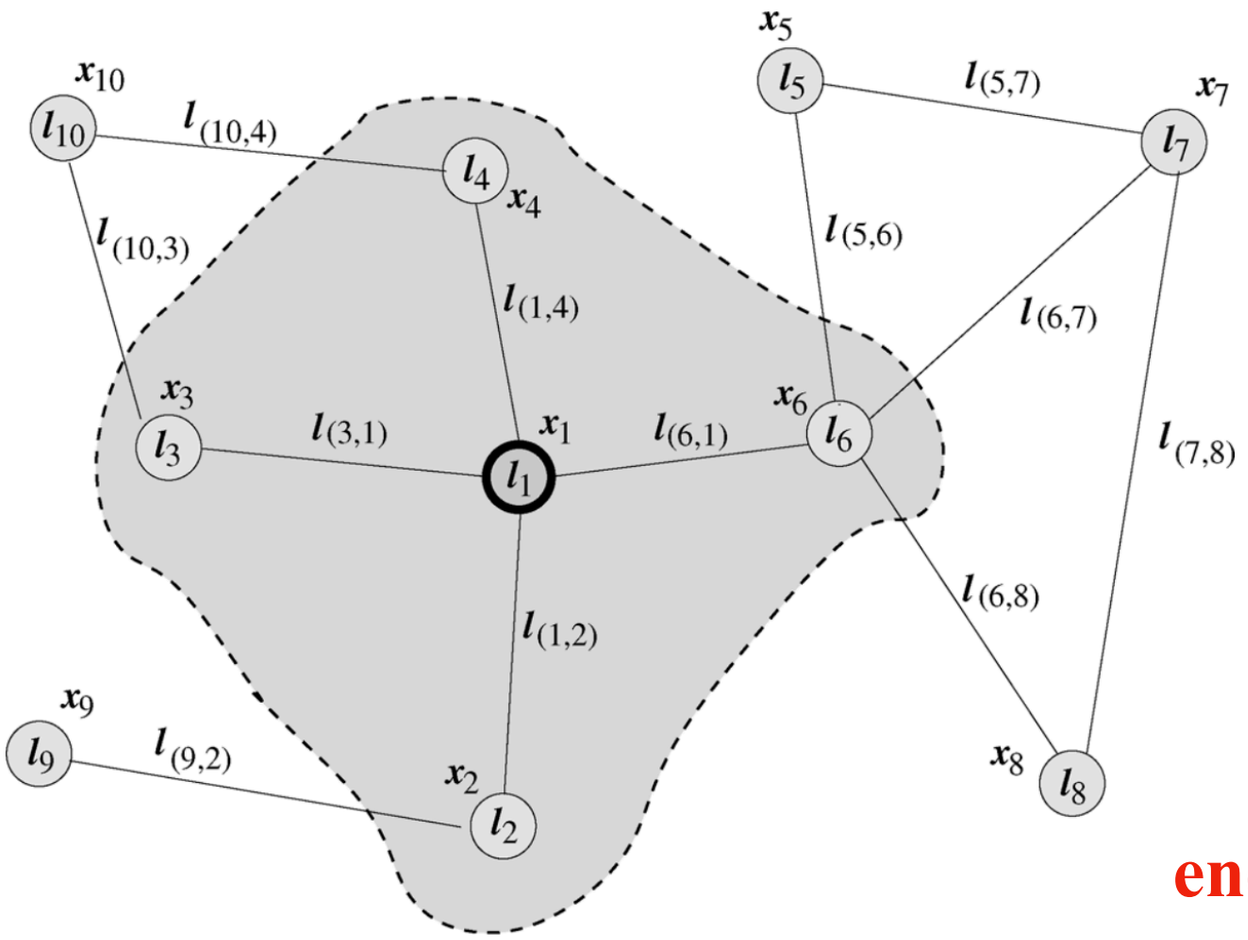
(slide inspired by Alexander Gaunt's talk on GNNs)

Definition

- Assume we have a graph G:
  - $\mathbf{V}$ is the vertex set.
  - $\mathbf{A}$ is the adjacency matrix (assume binary).
  - $\mathbf{X} \in \mathbf{R}^{m \times |V|}$ **is a matrix of node features.**
    - Categorical attributes, text, image data
      - E.g., profile information in a social network.
    - Node degrees, clustering coefficients, etc.
    - Indicator vectors (i.e., one-hot encoding of each node)

The Graph Neural Network Model

Franco Scarselli, Marco Gori, *Fellow, IEEE*, Ah Chung Tsoi, Markus Hagenbuchner, *Member, IEEE*, and Gabriele Monfardini



$$x_1 = f_w(l_1, l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}, x_2, x_3, x_4, x_6, l_2, l_3, l_4, l_6)$$

$l_{co[1]}$ $x_{ne[1]}$ $l_{ne[n]}$

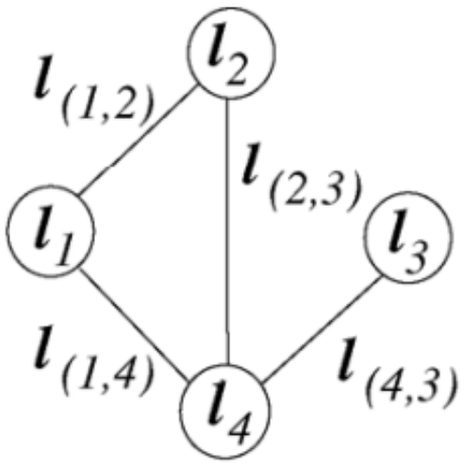
$$\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})$$

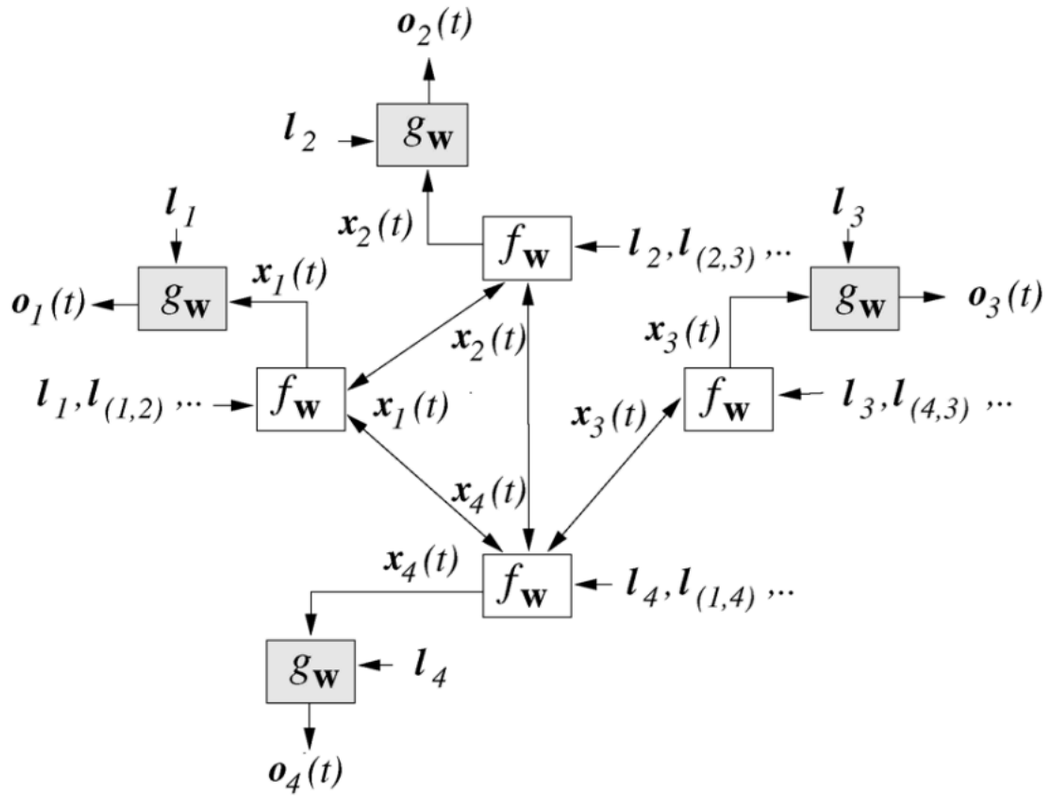
$$\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$$

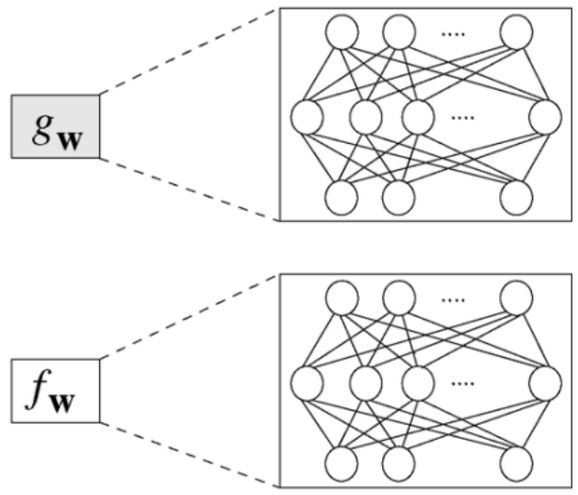
recursive — contraction map (unique)

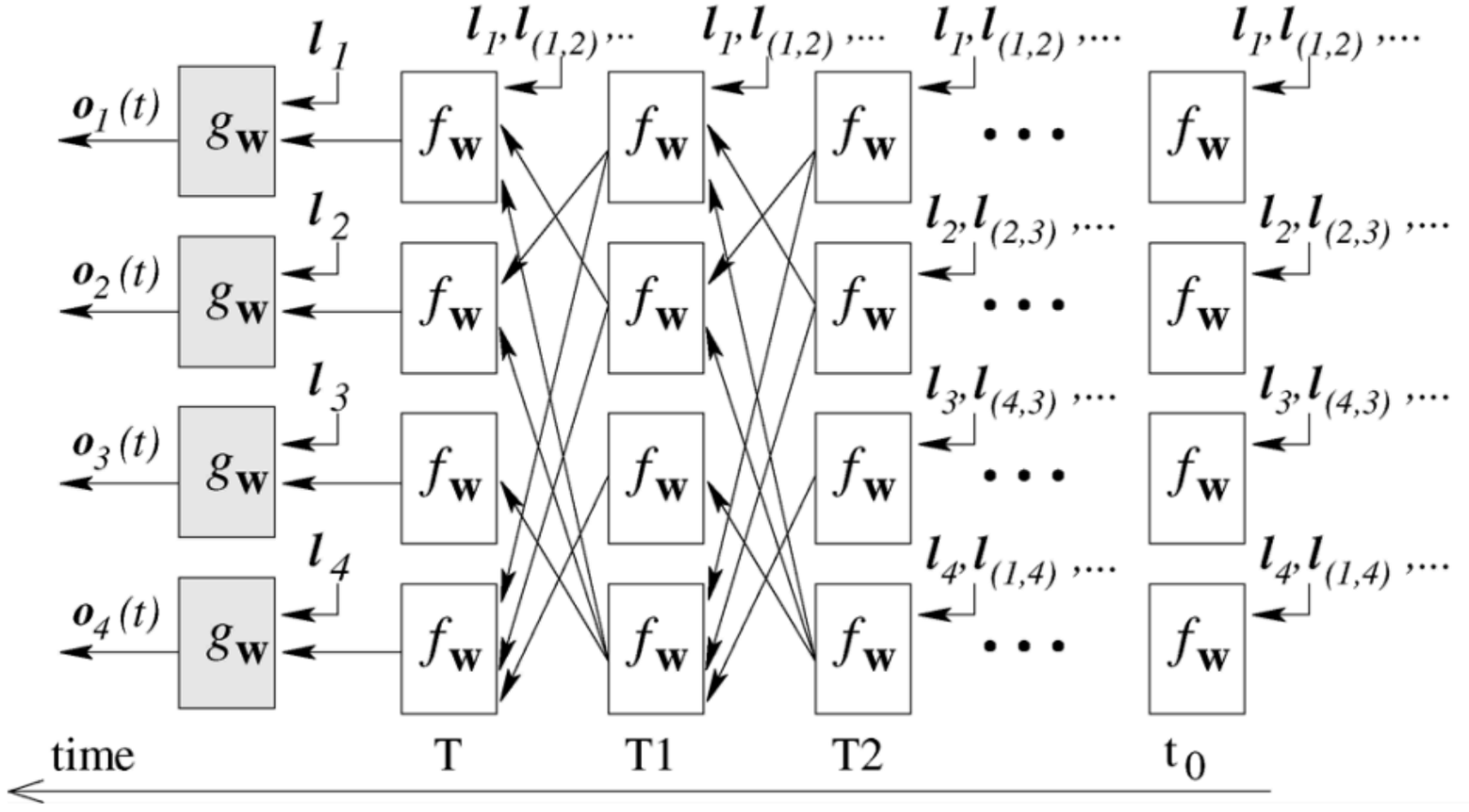
$$\mathbf{s}_i = \sum_{j \in \mathcal{N}(i)} \mathcal{F}\left(\mathbf{s}_i, \mathbf{s}_j, \mathbf{F}_i^V, \mathbf{F}_j^V, \mathbf{F}_{i,j}^E\right)$$

encode structural information of the graph


l(1,2)
l2
l(2,3)
l1
l3
l(1,4)
l4
l(4,3)

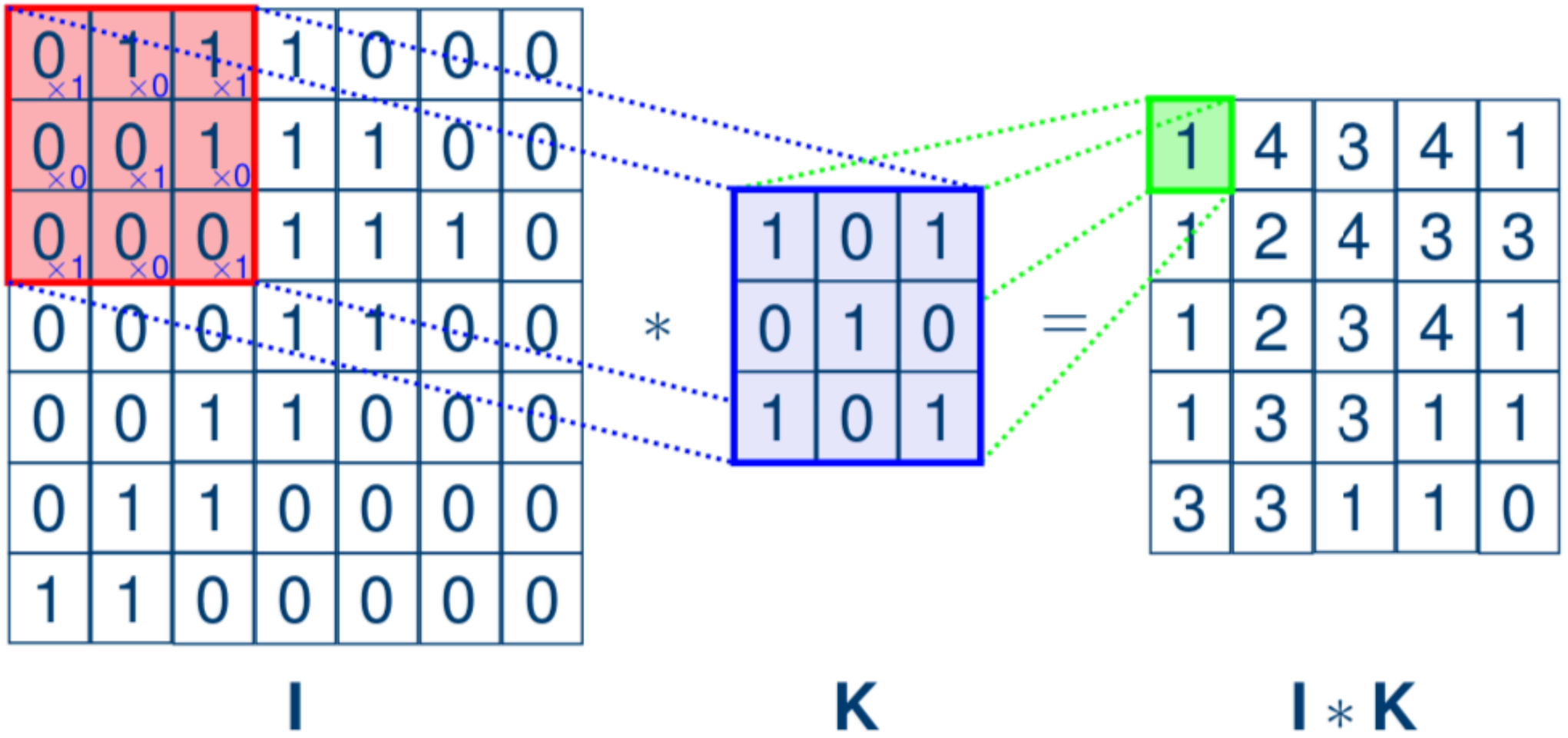

o2(t)
l2
gw
l1
x2(t)
fw
l2,l(2,3) ...
l3
o1(t)
x1(t)
x2(t)
x3(t)
o3(t)
l1,l(1,2) ...
x1(t)
x3(t)
l3,l(4,3) ...
x4(t)
x4(t)
l4,l(1,4) ...
l4
o4(t)

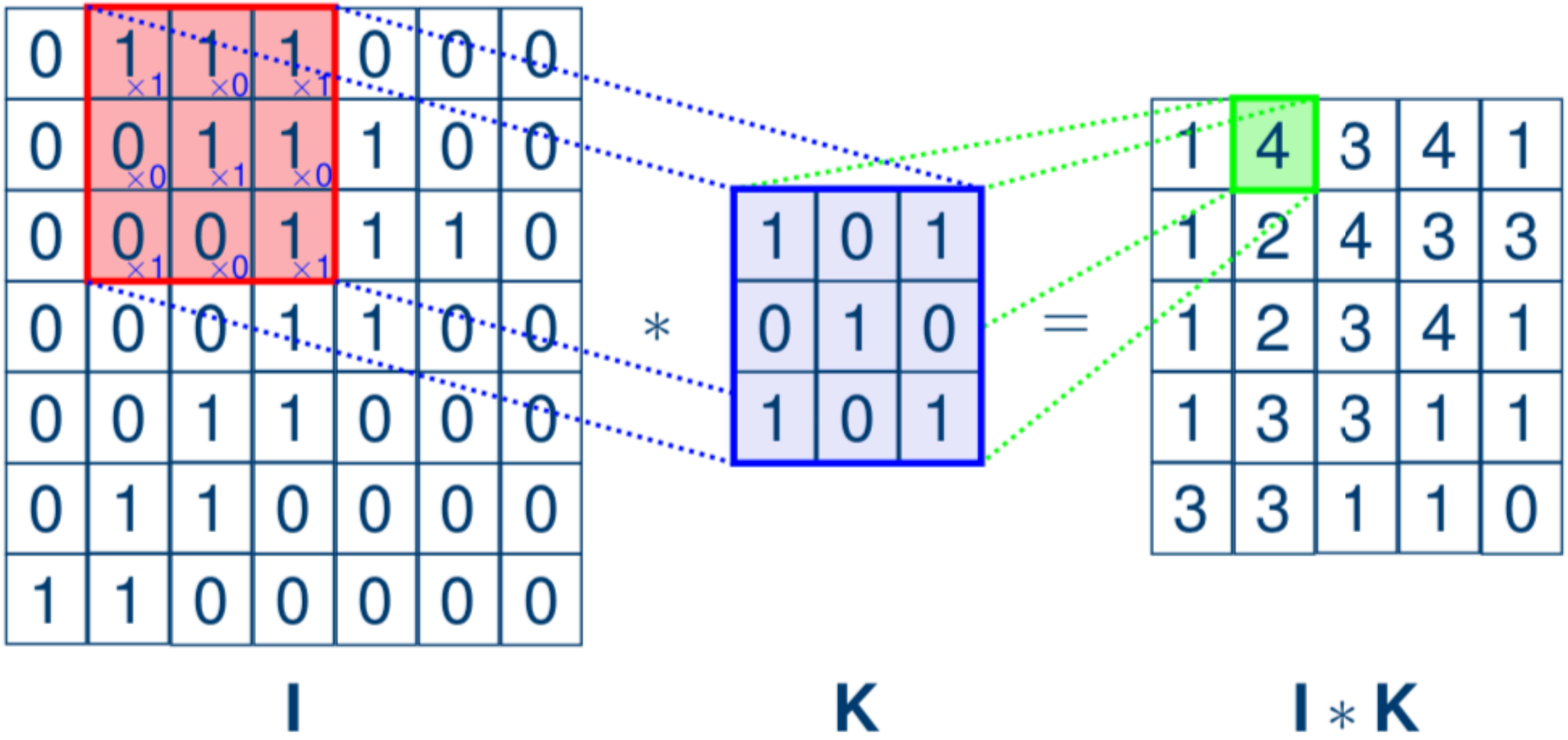

gw
fw

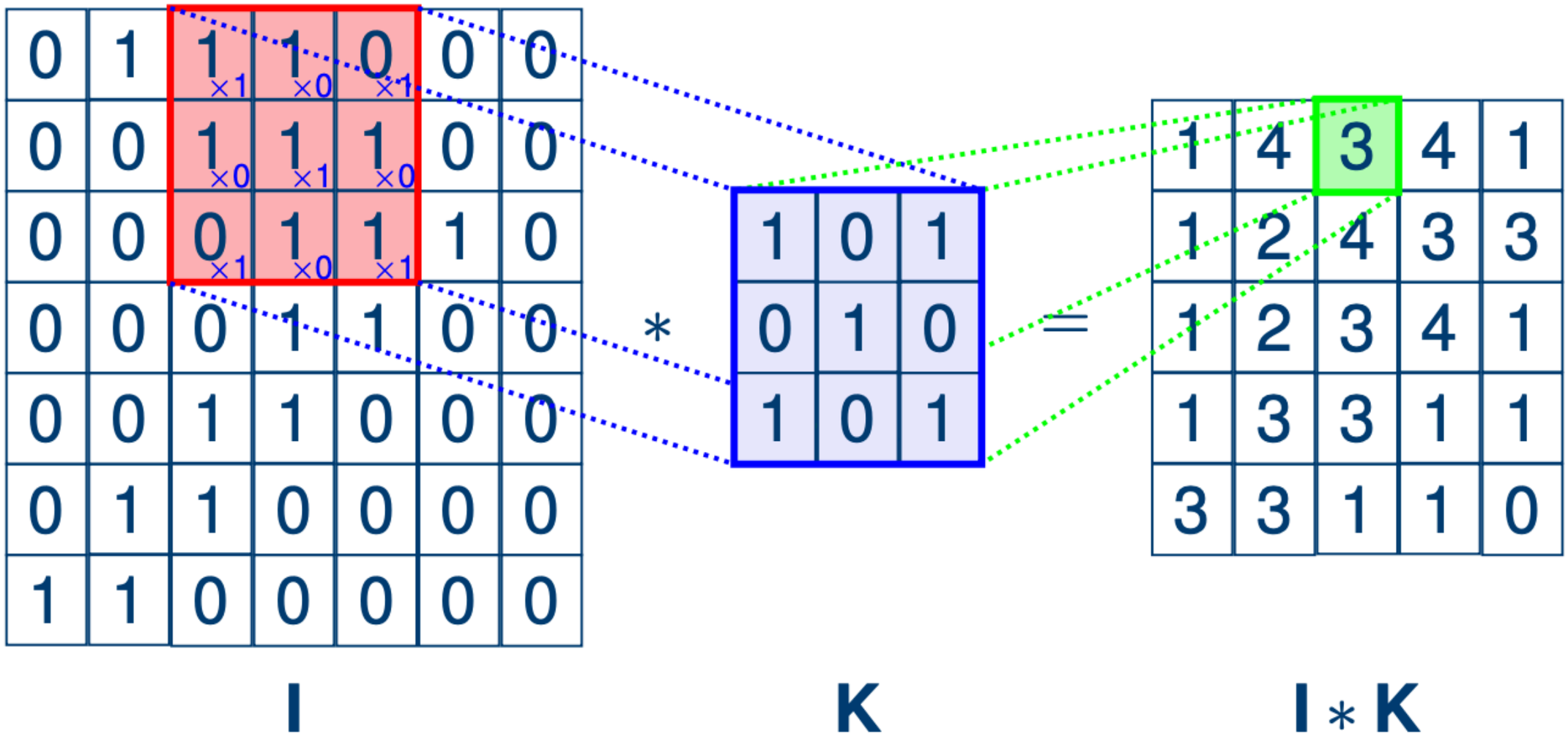

o1(t)
o2(t)
o3(t)
o4(t)
l1
l2
l3
l4
l1,l(1,2) ...
l2,l(2,3) ...
l3,l(4,3) ...
l4,l(1,4) ...
time
T
T1
T2
t0

Outline

- Intro
- Graph Neural Network
    - graph neural network
    - graph convolutional network
- Summary


0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 1 1 0 0 0 0
1 1 0 0 0 0 0
×1 ×0 ×1
×0 ×1 ×0
×1 ×0 ×1
*
1 0 1
0 1 0
1 0 1
=
1 4 3 4 1
1 2 4 3 3
1 2 3 4 1
1 3 3 1 1
3 3 1 1 0
I
K
I ∗ K


0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 1 1 0 0 0 0
1 1 0 0 0 0 0
×1 ×0 ×1
×0 ×1 ×0
×1 ×0 ×1
1 0 1
0 1 0
1 0 1
1 4 3 4 1
1 2 4 3 3
1 2 3 4 1
1 3 3 1 1
3 3 1 1 0
I
K
I ∗ K


0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 0 1
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 1 1 0 0 0 0
1 1 0 0 0 0 0
×1 ×0 ×1
×0 ×1 ×0
×1 ×0 ×1
*
1 0 1
0 1 0
1 0 1
=
1 4 3 4 1
1 2 4 3 3
1 2 3 4 1
1 3 3 1 1
3 3 1 1 0
I
K
I ∗ K

Spatial method
Spectral method

Spectral Networks and Deep Locally Connected Networks on Graphs

Joan Bruna
New York University
bruna@cims.nyu.edu

Wojciech Zaremba
New York University
woj.zaremba@gmail.com

Arthur Szlam
The City College of New York
aszlam@ccny.cuny.edu

Yann LeCun
New York University
yann@cs.nyu.edu

$$x_{k+1,j} = L_k h\left(\sum_{i=1}^{f_{k-1}} F_{k,i,j} x_{k,i}\right) \quad (j = 1 \dots f_k)$$

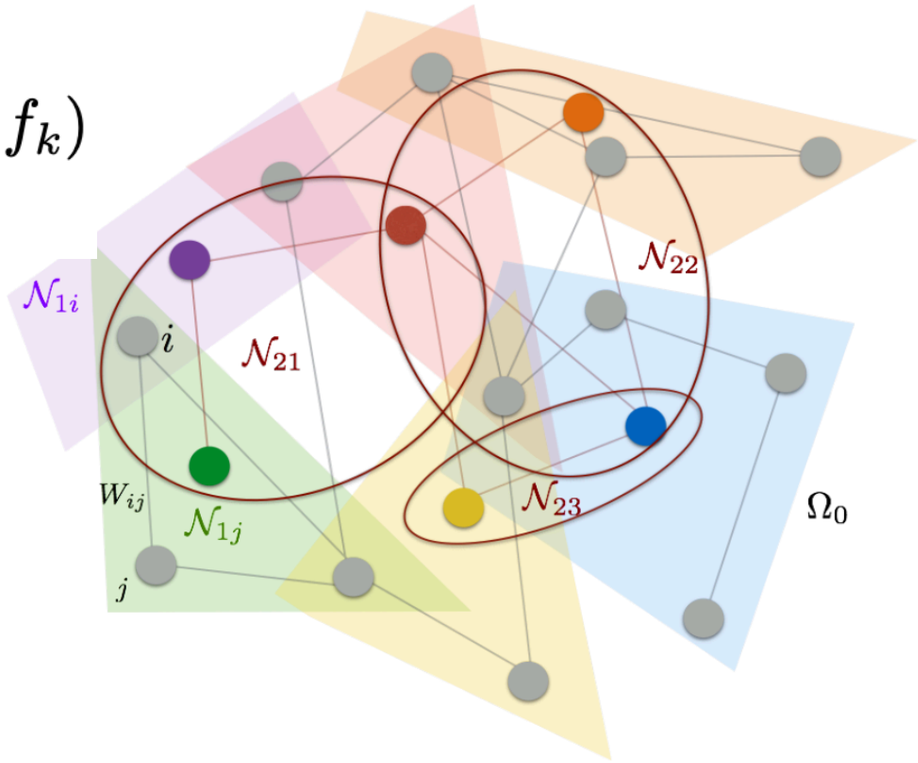


Figure 1: Undirected Graph $G = (\Omega_0, W)$ with two levels of clustering. The original points are drawn in gray.

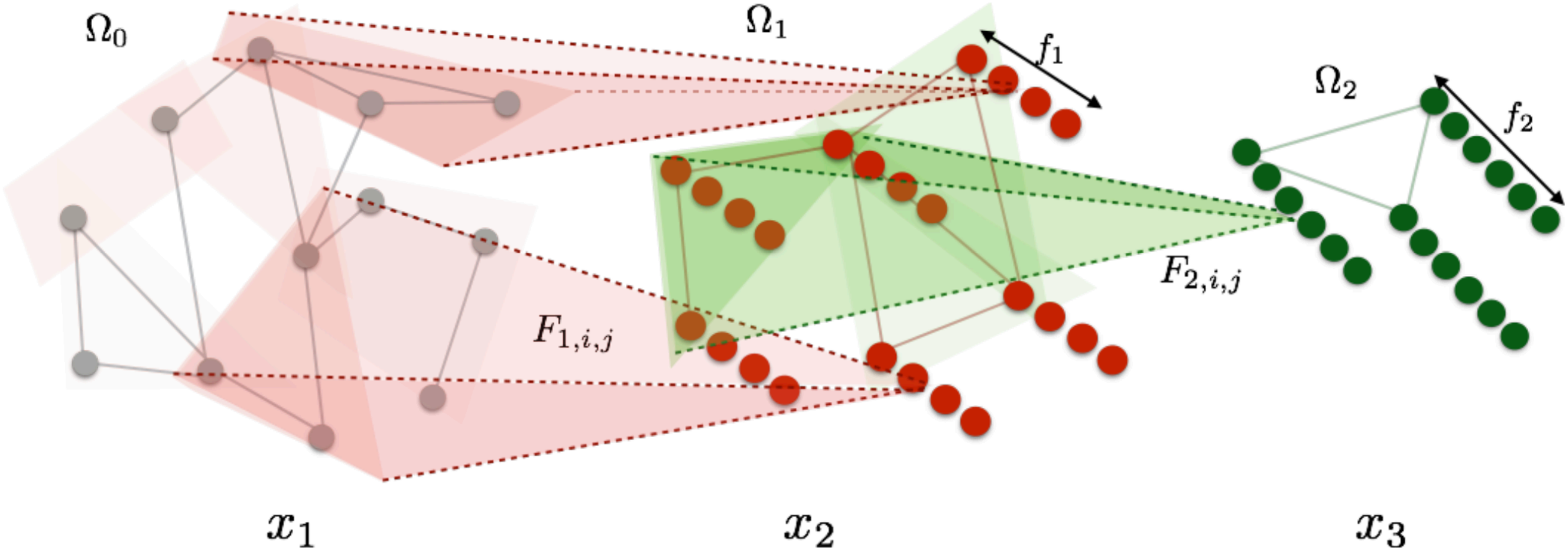


Figure 2: Spatial Construction as described by (2.1), with $K = 2$. For illustration purposes, the pooling operation is assimilated with the filtering stage. Each layer of the transformation loses spatial resolution but increases the number of filters.

$$
\begin{aligned}
W_0 &= W \\
A_k(i,j) &= \sum_{s \in \Omega_k(i)} \sum_{t \in \Omega_k(j)} W_{k-1}(s,t)\ ,\ (k \le K) \\
W_k &= \text{rownormalize}(A_k)\ ,\ (k \le K) \\
\mathcal{N}_k &= \text{supp}(W_k)\ .\ (k \le K)
\end{aligned}
$$

The global structure of the graph can be exploited with the spectrum of its graph-Laplacian to generalize the convolution operator.

Graph Signal Processing:

Normalized Laplacian matrix ==> Real symmetric positive semidefinite.

Eigenvalues (spectrum) of diagonal matrix:

$$U = [u_0, u_1, \cdots, u_{n-1}] \in \mathbb{R}^{n \times n}$$

$$L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^{\top}$$

$$x_{k+1,j} = h\left(V \sum_{i=1}^{f_{k-1}} F_{k,i,j} V^T x_{k,i} \right) \quad (j = 1 \ldots f_k)$$

Convolution Theorem: $\mathbf{U}^\top(\mathbf{y} * \mathbf{x}) = (\mathbf{U}^\top\mathbf{y}) \odot (\mathbf{U}^\top\mathbf{x})$

Element-wise product => diagonal matrix $(\mathbf{U}^\top\mathbf{y}) \odot (\mathbf{U}^\top\mathbf{x}) = g_\theta\mathbf{U}^\top\mathbf{x}$ **(learnable filters)**

Graph convolution: $g_\theta * \mathbf{x} = \mathbf{U}g_\theta\mathbf{U}^\top\mathbf{x}$ **Filtering a signal x**

Graph Laplacian: $L = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = U\Lambda U^\top$

Chebyshev Polynomial: $g_\theta * \mathbf{x} \approx \sum_{k=0}^{K} \theta'_k T_k(L_{sym})\mathbf{x}$ $T_k(\mathbf{x}) = 2\mathbf{x}T_{k-1}(\mathbf{x}) - T_{k-2}(\mathbf{x})$

1st order approximation: $g_\theta * \mathbf{x} = \theta\left(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}\right)\mathbf{x}$

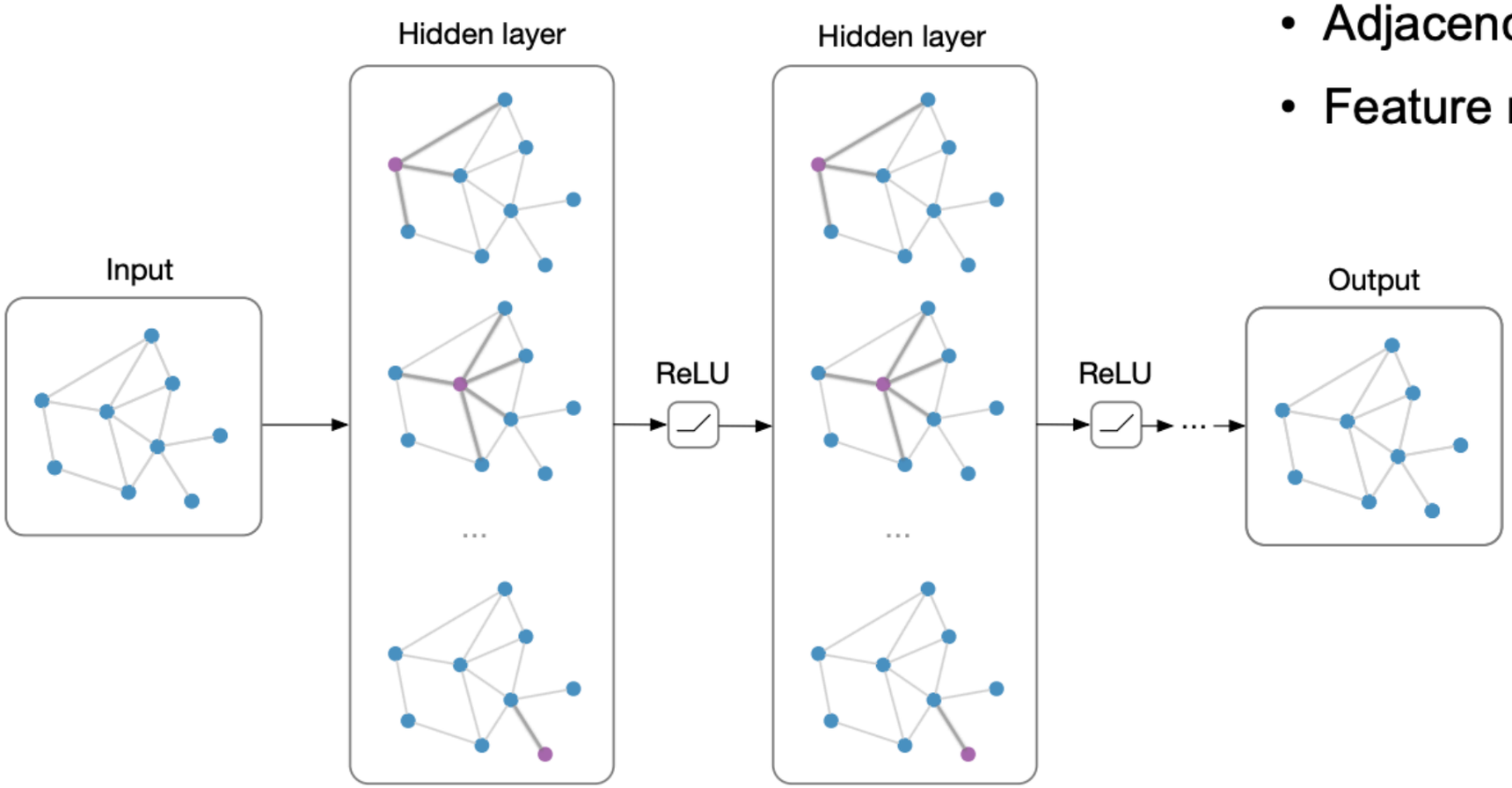
Renormalization: $I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \longrightarrow \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$

Propagation: $H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}\Theta^{(l)}\right)$

Output: $Z = f(X, A) = \text{softmax}\left(\hat{A}\ \text{ReLU}\left(\hat{A}X\Theta^{(0)}\right)\Theta^{(1)}\right)$

Graph Neural Networks (GNNs)

The bigger picture:



Notation: $\mathcal{G} = (\mathbf{A}, \mathbf{X})$

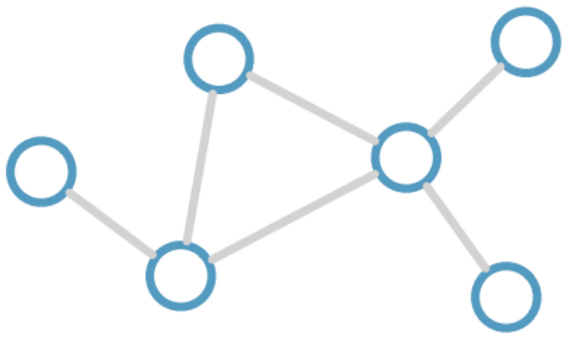
- Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$
- Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$

Main idea: Pass messages between pairs of nodes & agglomerate

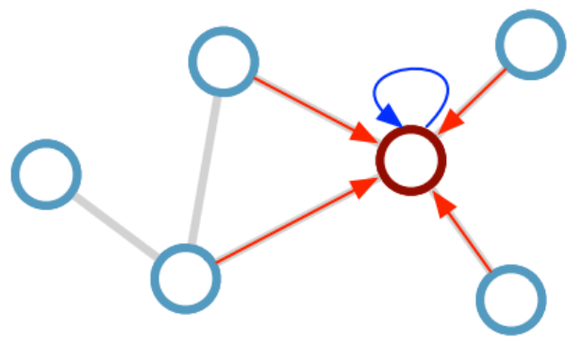
Graph convolutional networks (GCNs)

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

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

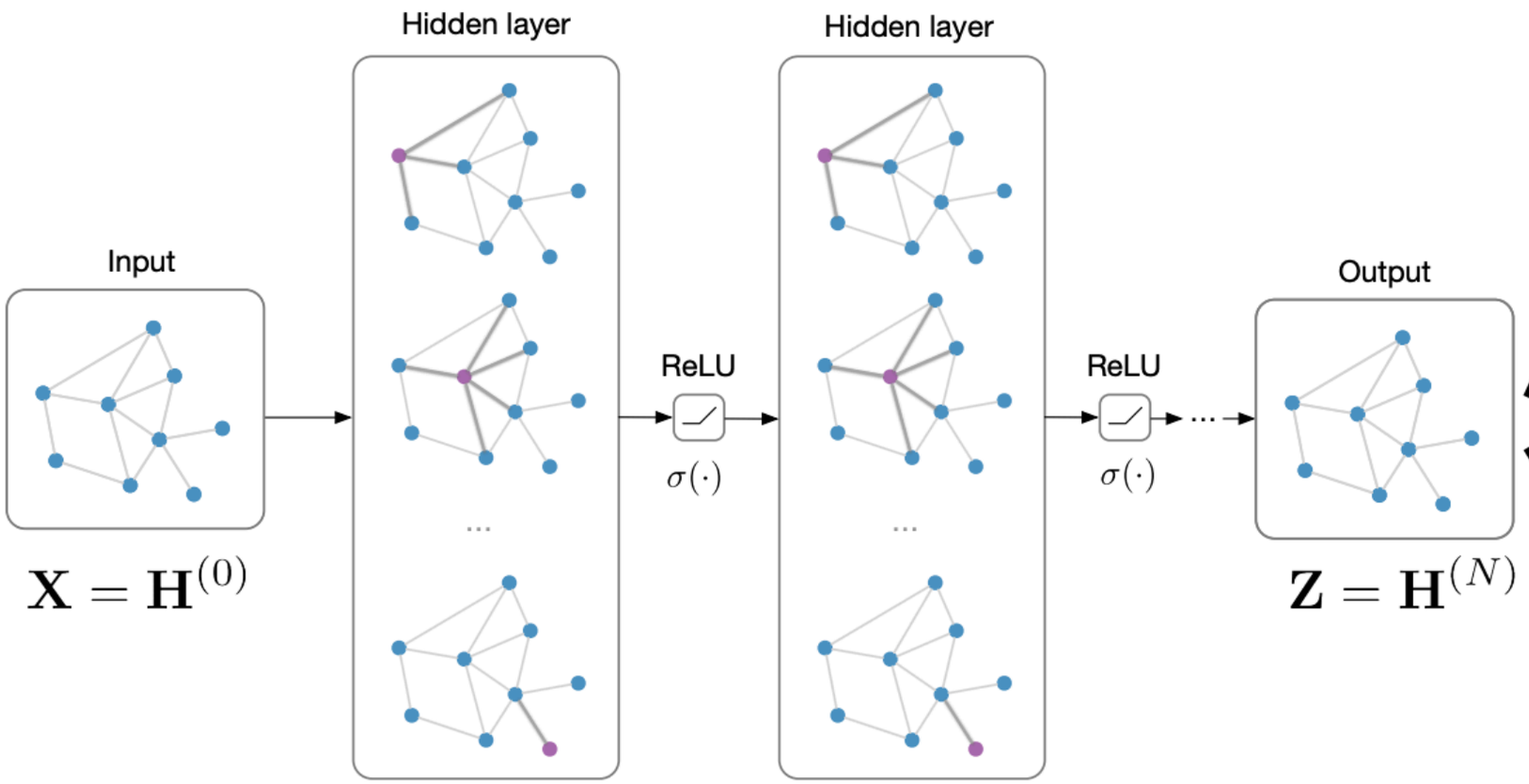
Limitations:

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

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

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

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



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

Node classification:

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

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

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

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

Link prediction:

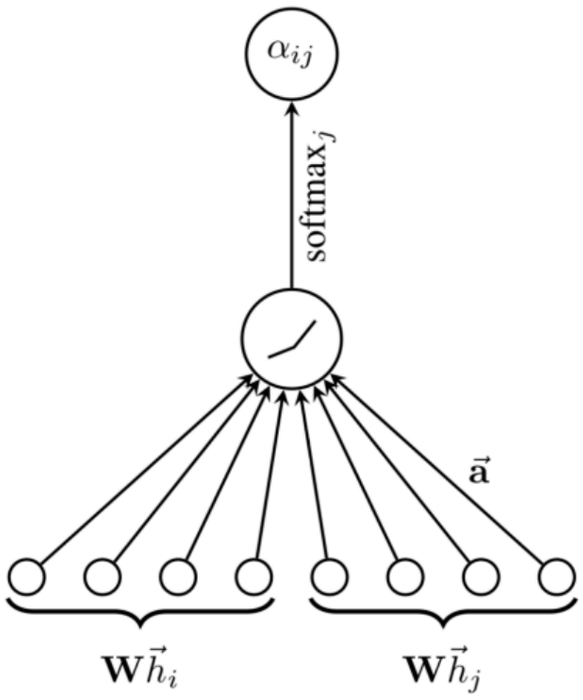
$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$

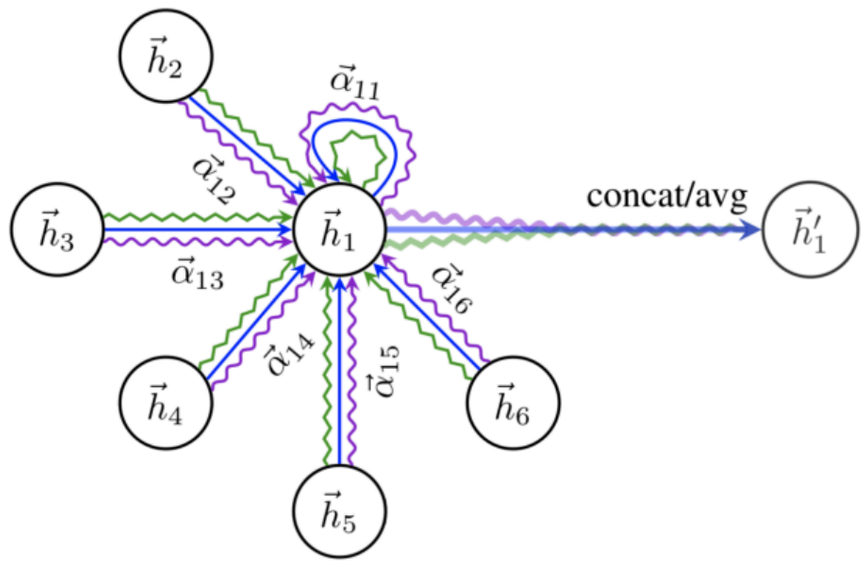
Kipf & Welling (NIPS BDL 2016)

"Graph Auto-Encoders"

Graph neural networks with attention

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





[Figure from Veličković et al. (ICLR 2018)]

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:

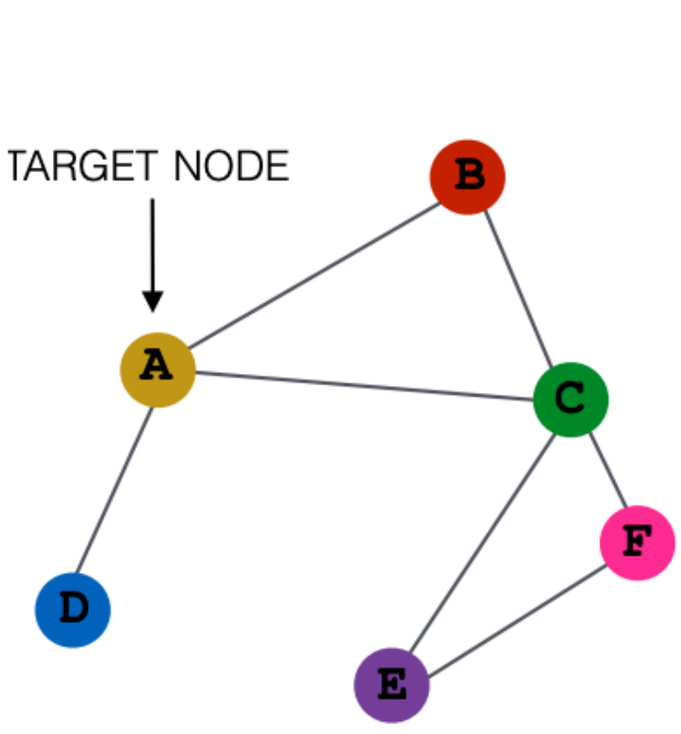
- (Most likely) less expressive than GNNs with edge embeddings
- Can be more difficult to optimize

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

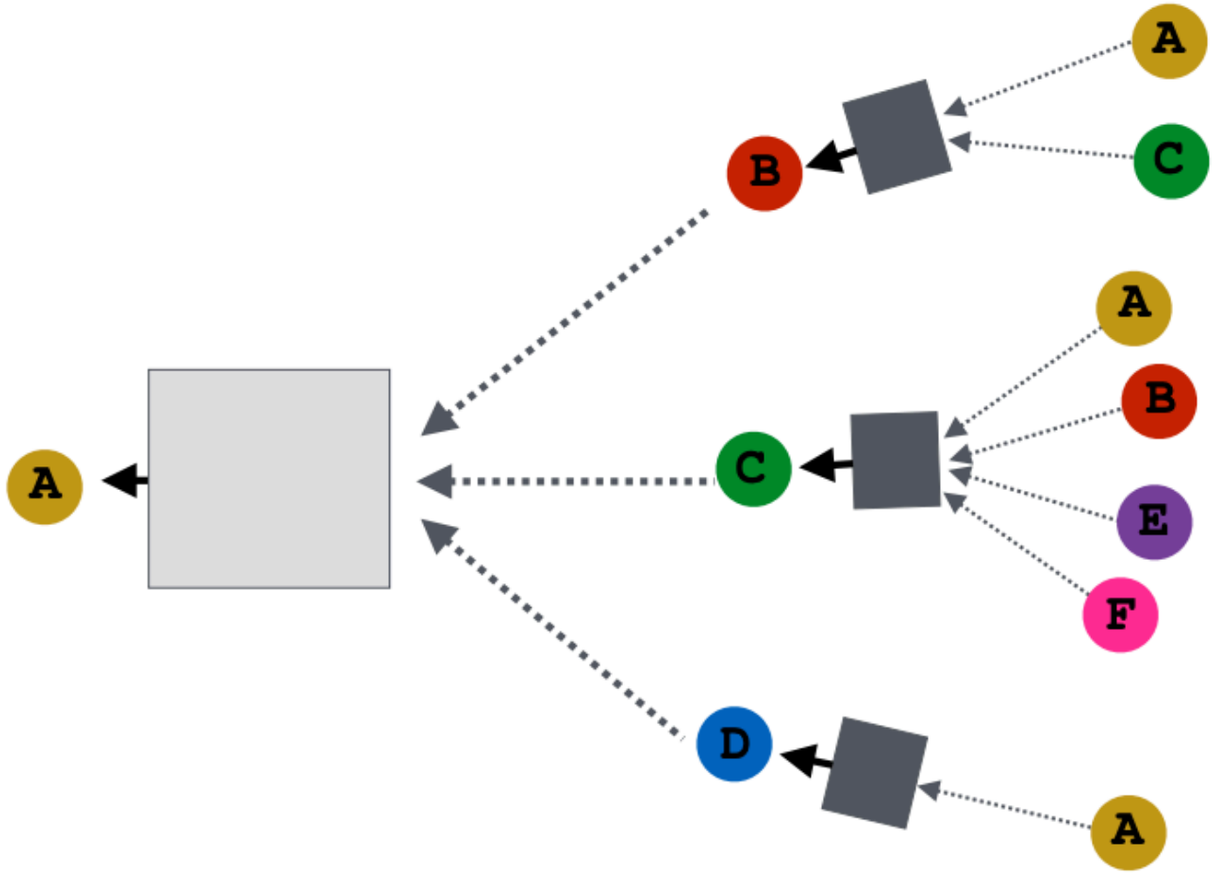
$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_k]\right)\right)}$$

Neighborhood Aggregation

- **Key idea:** Generate node embeddings based on local neighborhoods.



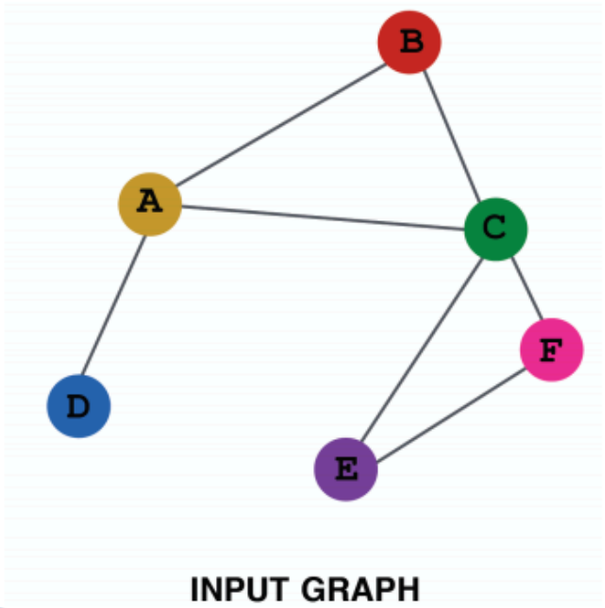
INPUT GRAPH



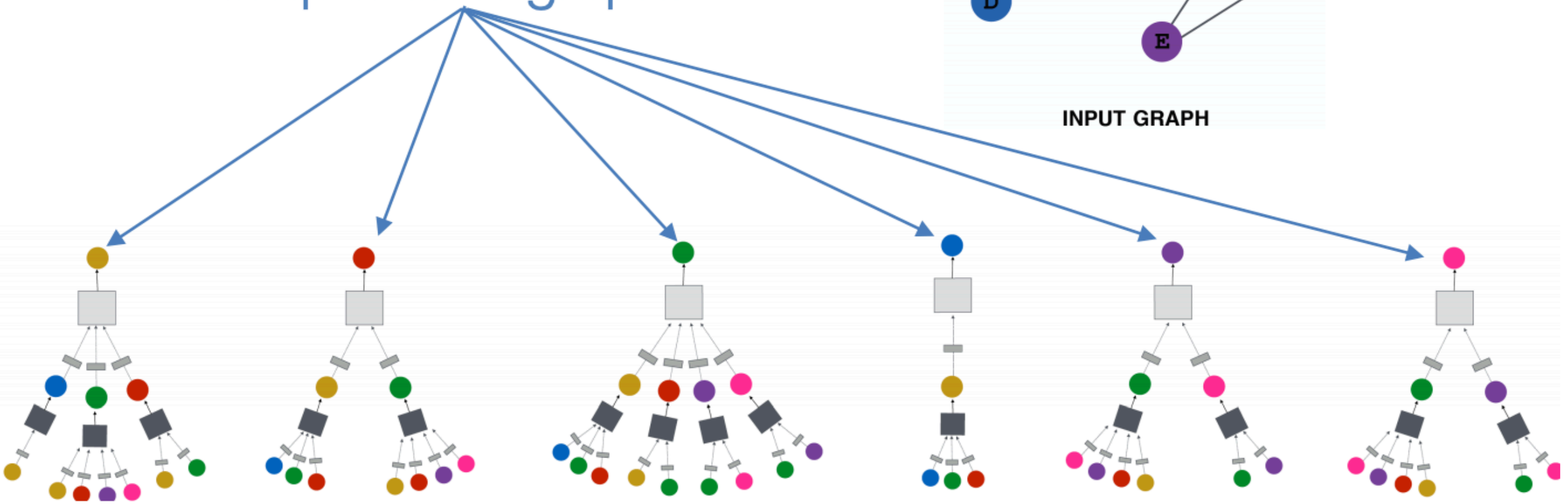
Neighborhood Aggregation

- **Intuition:** Network neighborhood defines a computation graph

Every node defines a unique computation graph!



INPUT GRAPH



The Math

- **Basic approach:** Average neighbor messages and apply a neural network.

Initial "layer 0" embeddings are equal to node features

previous layer embedding of v

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k > 0$$

kth layer embedding of v

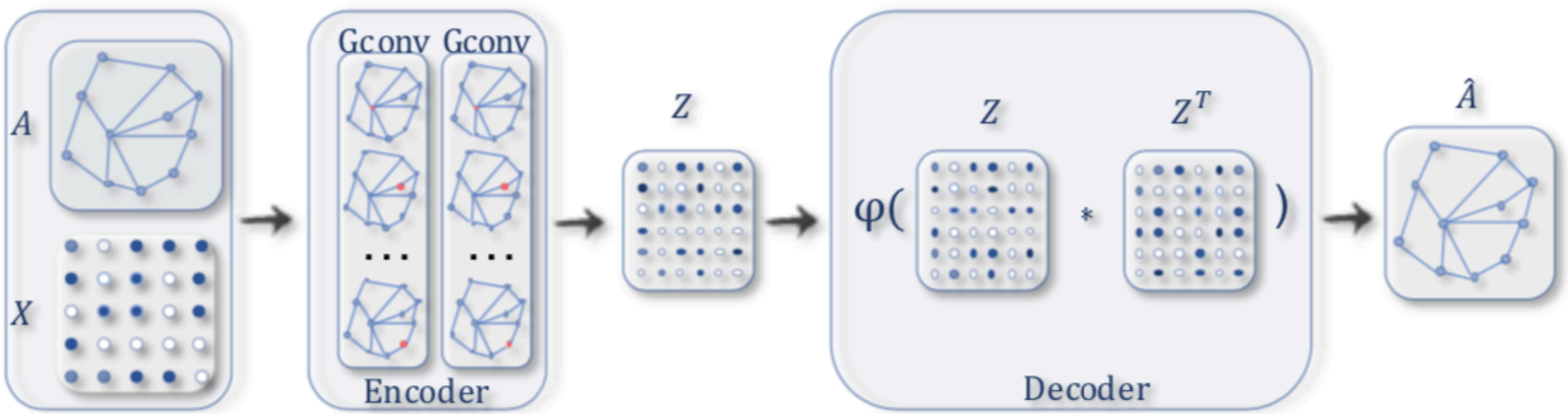
non-linearity (e.g., ReLU or tanh)

average of neighbor's previous layer embeddings

- Spectral Model
    - Solid theoretical foundation.
    - Need to compute eigenvectors.
    - Need to compute the whole graph.
    - Graph Fourier basis can not apply to new graph.
- Spatial Model (Information Aggregation)
    - Efficiency, generality, flexibility.
    - Can compute on a batch.

Outline

- Intro
- Graph Neural Network
    - graph neural network
    - graph convolutional network
    - graph auto-encoder
- Summary



(c) A GAE for network embedding [61]. The encoder uses graph convolutional layers to get a network embedding for each node. The decoder computes the pair-wise distance given network embeddings. After applying a non-linear activation function, the decoder reconstructs the graph adjacency matrix. The network is trained by minimizing the discrepancy between the real adjacency matrix and the reconstructed adjacency matrix.

$$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^{\top}\right), \quad \text{with} \quad \mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A})$$

Outline

- Intro
- Graph Neural Network
    - graph neural network
    - graph convolutional network
    - graph auto-encoder
    - graph reinforcement learning
- Summary

GRAPH CONVOLUTIONAL REINFORCEMENT LEARNING

Jiechuan Jiang
Peking University
jiechuan.jiang@pku.edu.cn

Chen Dun*
Rice University
cd46@rice.edu

Tiejun Huang & Zongqing Lu†
Peking University
{tjhuang,zongqing.lu}@pku.edu.cn

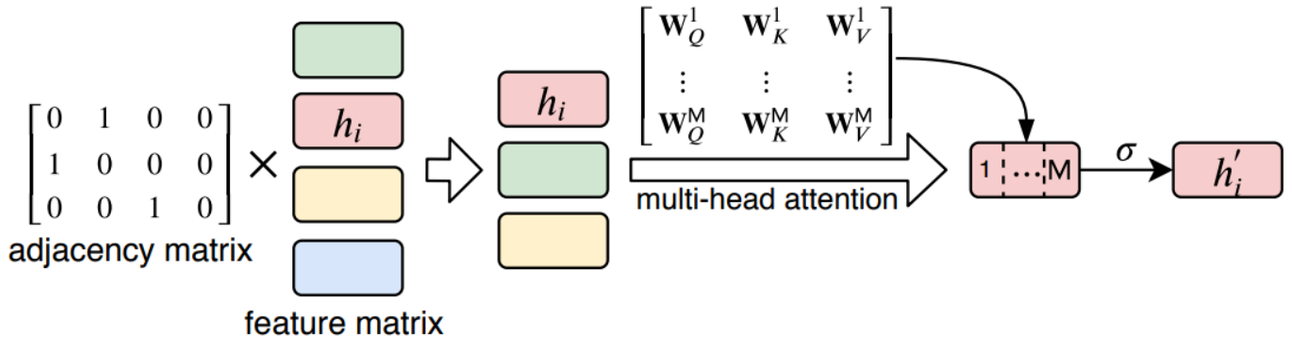


Figure 2: Illustration of computation of the convolutional layer with relation kernel of multi-head attention.

$$\mathcal{L}(\theta) = \frac{1}{\mathsf{S}} \sum_{\mathsf{S}} \frac{1}{\mathsf{N}} \sum_{i=1}^{\mathsf{N}} \left(y_i - Q\left(O_{i,\mathcal{C}}, a_i; \theta\right)\right)^2$$

adjacency matrix $\mathcal{C} = \{C_1, \cdots, C_{\mathsf{N}}\}$

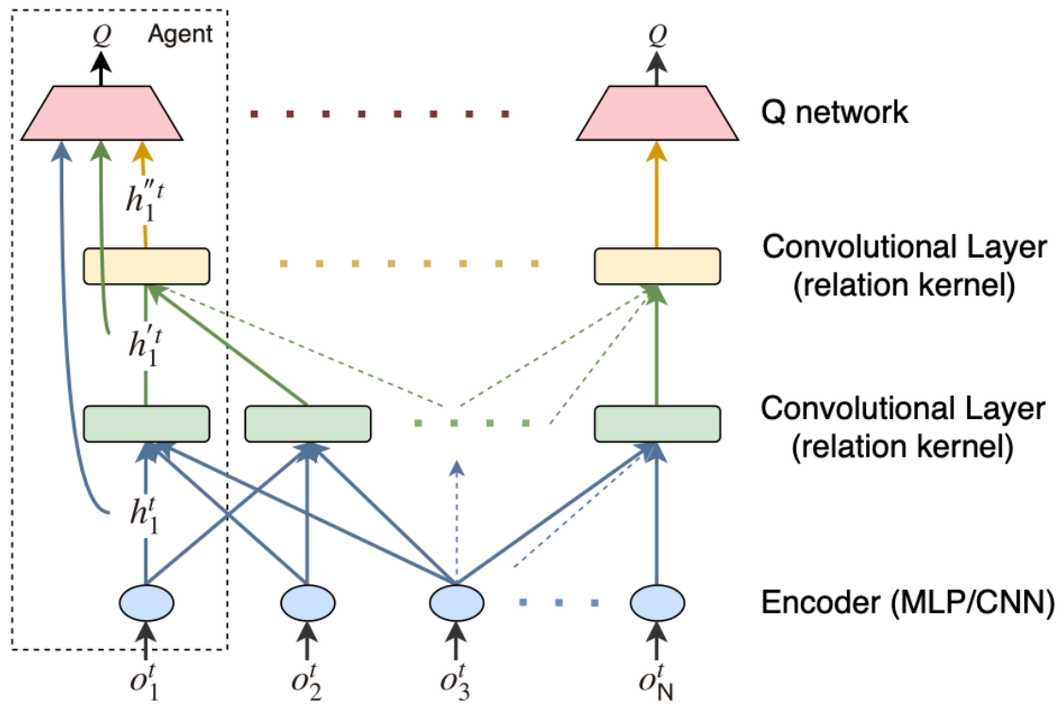
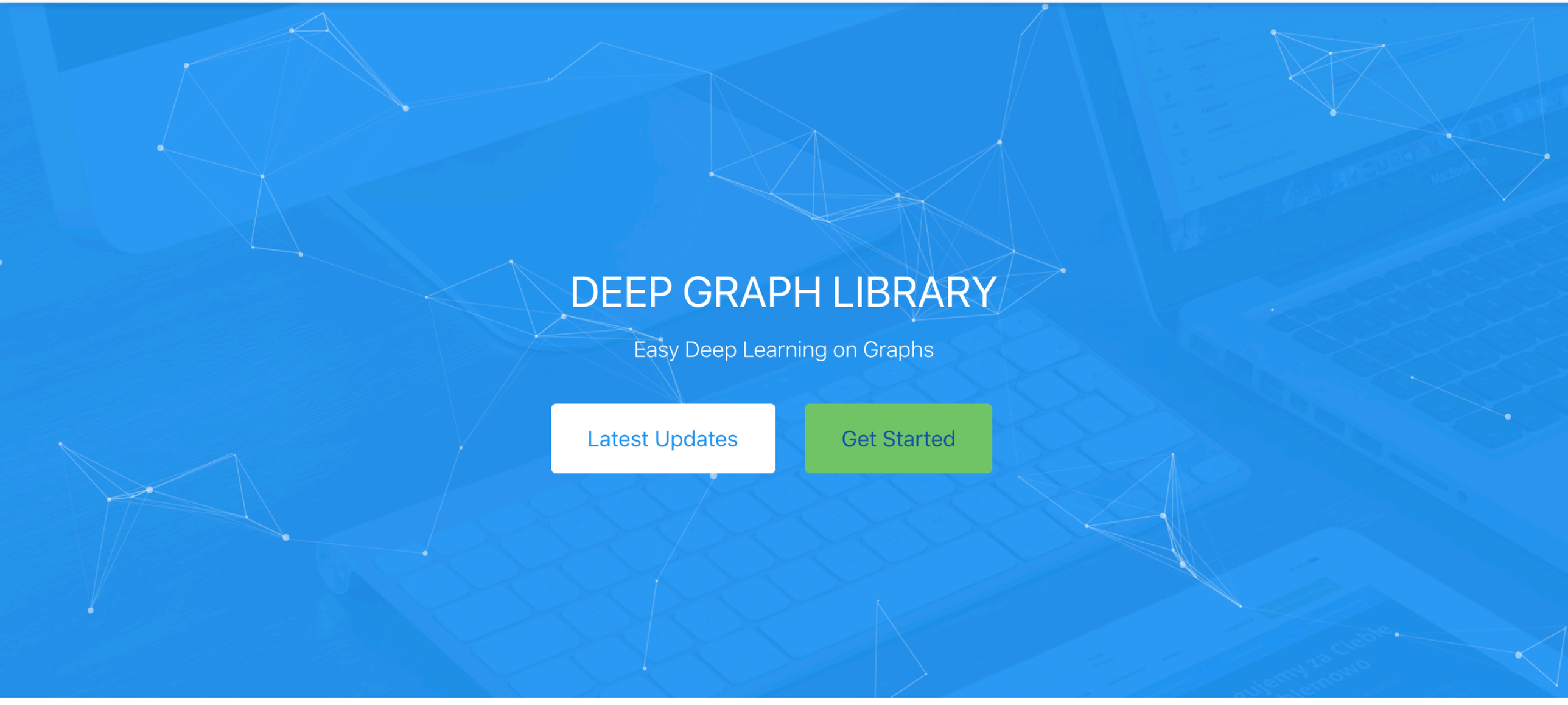


Figure 1: DGN consists of three modules: encoder, convolutional layer, and Q network. All agents share weights and gradients are accumulated to update the weights.

Multi-agent environment ==> Graph

Neighboring agents are more likely to interact with each other.



https://www.dgl.ai/

https://github.com/dmlc/dgl

Reference

- [WWW18] (http://snap.stanford.edu/proj/embeddings-www/)
- [TKipf] (http://tkipf.github.io/misc/SlidesCambridge.pdf)
- [AAAI19] (https://jian-tang.com/files/AAAI19/aaai-grltutorial-part0-intro.pdf)
- [Tutorial] (https://www.cl.cam.ac.uk/~pv273/slides/MILA-attn.pdf)
- [Tutorial] (http://snap.stanford.edu/proj/embeddings-www/files/nrltutorial-part2-gnns.pdf)