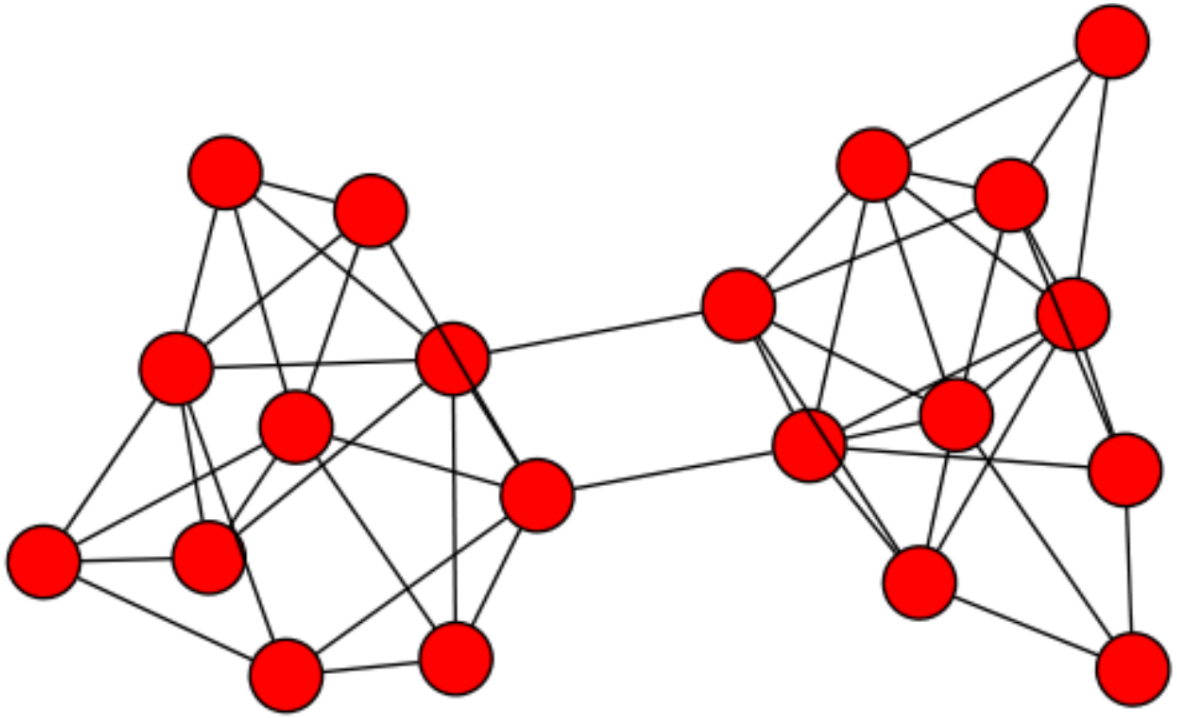
Recent Advances in Machine learning on Graphs

Chanyoung Park

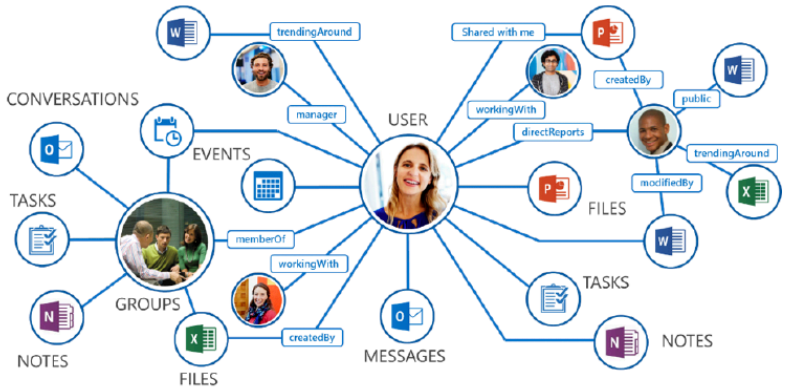
Department of Industrial & Systems Engineering
KAIST
cy.park@kaist.ac.kr

GRAPH (NETWORK)

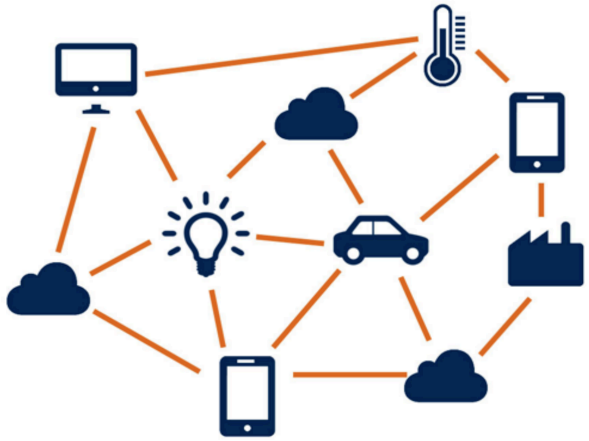
- A general description of data and their relations



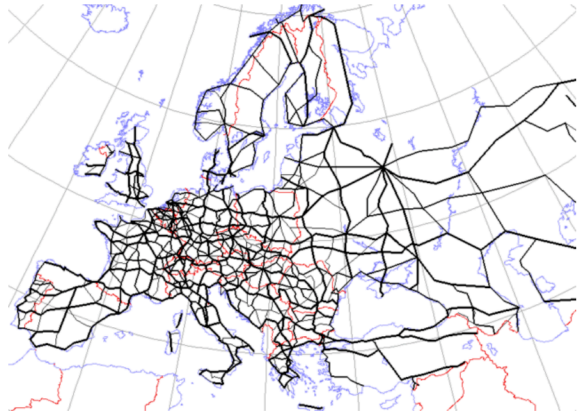
VARIOUS REAL-WORLD GRAPHS (NETWORKS)



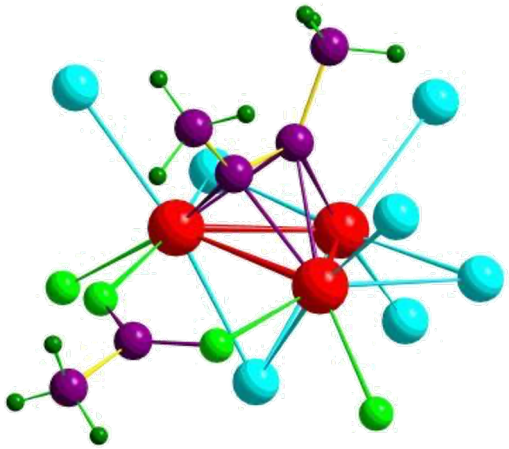
Social graph



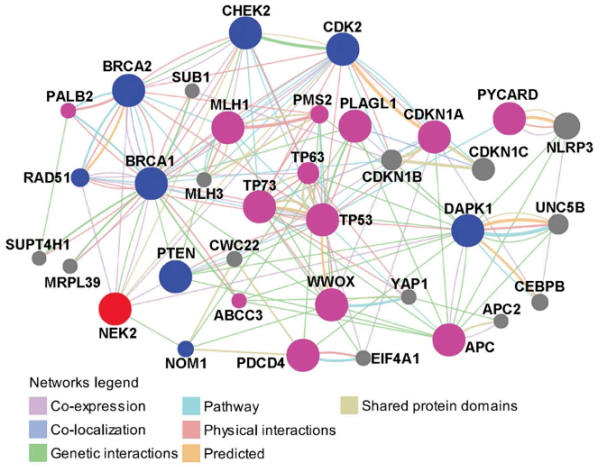
Internet-of-Things



Transportation



Molecular graph



Gene network

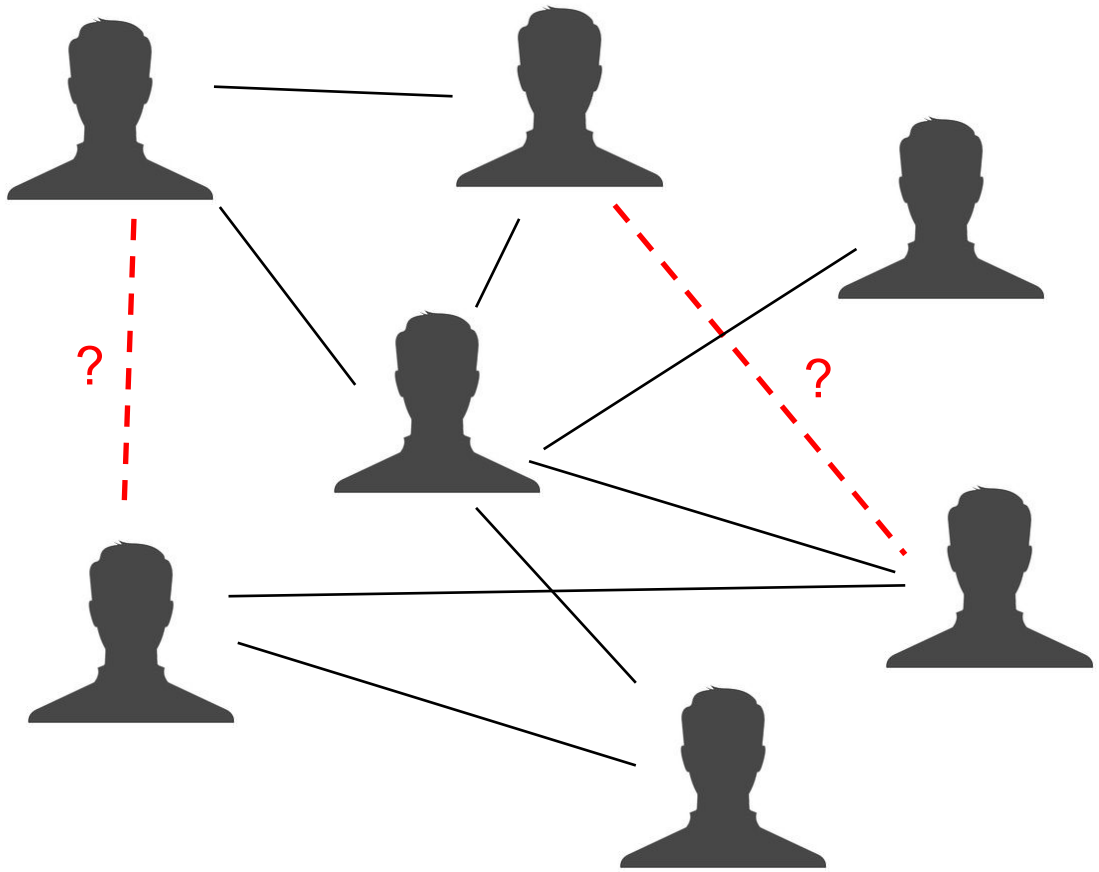
Web graph

(Figure credit) Web

MACHINE LEARNING ON GRAPHS

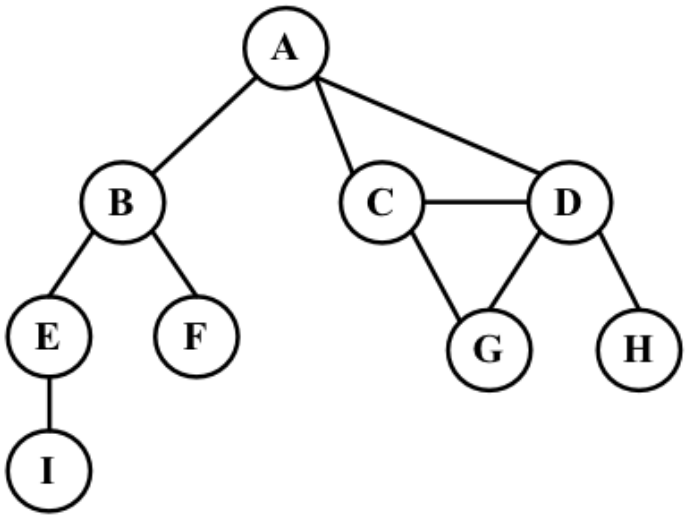
Classical ML tasks in graphs:

- 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



Link Prediction
(Friend Recommendation)

TRADITIONAL GRAPH REPRESENTATION



	A	B	C	D	E	F	G	H	I
A	0	1	1	1	0	0	0	0	0
B	1	0	0	0	1	1	0	0	0
C	1	0	0	1	0	0	1	0	0
D	1	0	1	0	0	0	1	1	0
E	0	1	0	0	0	0	0	0	1
F	0	1	0	0	0	0	0	0	0
G	0	0	1	1	0	0	0	0	0
H	0	0	0	1	0	0	0	0	0
I	0	0	0	0	1	0	0	0	0

Adjacency matrix

Problems

- Suffer from data sparsity
- Suffer from high dimensionality
- High complexity for computation
- Does not represent “semantics”
- …

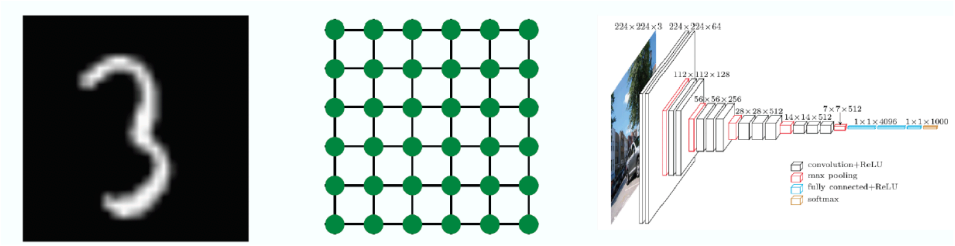
How to effectively and efficiently represent graphs is the key!

→ Deep learning-based approach?

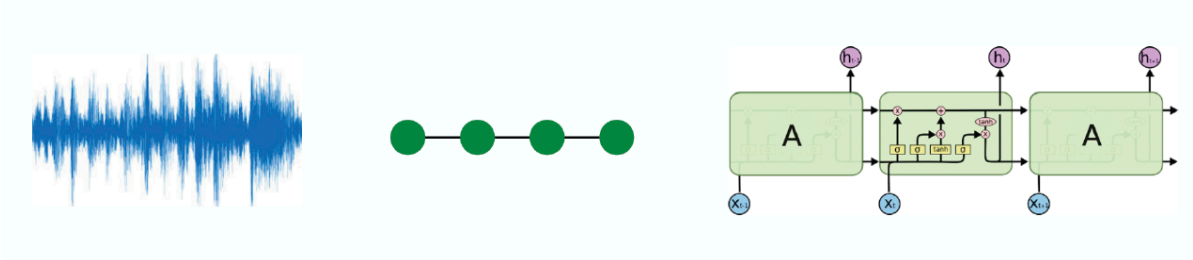
(Figure credit) https://www.oreilly.com/library/view/learning-javascript-data/9781788623872/8a7d3187-7c57-418c-a426-3aceab96f47f.xhtml

CHALLENGES OF GRAPH REPRESENTATION LEARNING

- Existing deep neural networks are designed for data with regular-structure (grid or sequence)
  - CNNs for fixed-size images/grids …



  - RNNs for text/sequences …



- **Graphs are very complex**
  - Arbitrary structures (no spatial locality like grids / no fixed orderings)
  - Heterogeneous: Directed/undirected, binary/weighted/typed, multimodal features
  - Large-scale: More than millions of nodes and billions of edges

(Figure credit) Tutorial on Graph Representation Learning, AAAI 2019

THIS TALK

- How to learn graph representation in **various types of graphs**?
  - Homogeneous Network Embedding
  - Attributed Network Embedding
  - Multi-aspect Network Embedding
  - Heterogeneous Network Embedding
- How to effectively **train GNNs**?
  - Self-supervised learning
  - Going deeper with GNN

- What is not covered in this talk?
  - Dynamic graph
  - Expressiveness of GNN (e.g., isomorphism)
  - Robustness of GNN (i.e., adversarial attack)
  - Graph-level operations (i.e., graph pooling)

OUTLINE

- Homogeneous Network Embedding
- Multi-aspect Network Embedding
- Attributed Network Embedding
- Heterogeneous Network Embedding
- Training GNN
  - Self-supervised learning
  - Going deeper with GNN
- Applications of Graph Machine Learning

OUTLINE

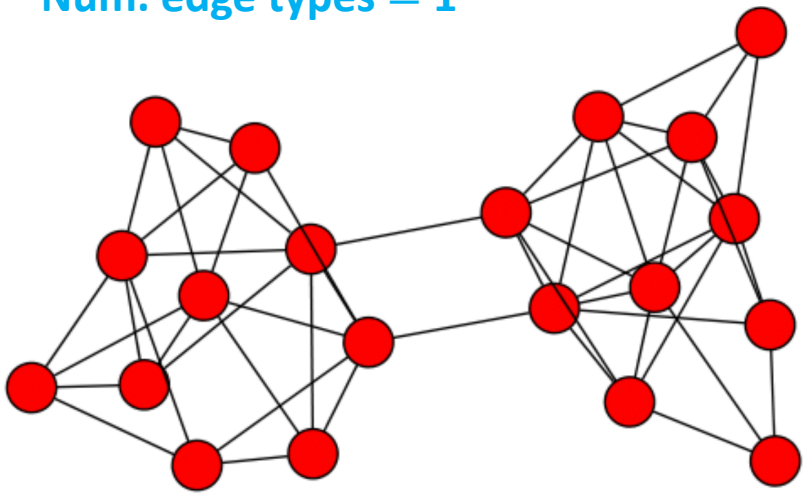
- Homogeneous Network Embedding
- Multi-aspect Network Embedding
- Attributed Network Embedding
- Heterogeneous Network Embedding
- Training GNN
  - Self-supervised learning
  - Going deeper with GNN
- Applications of Graph Machine Learning

HOMOGENEOUS NETWORK

- A graph with a single type of node and a single type of edge

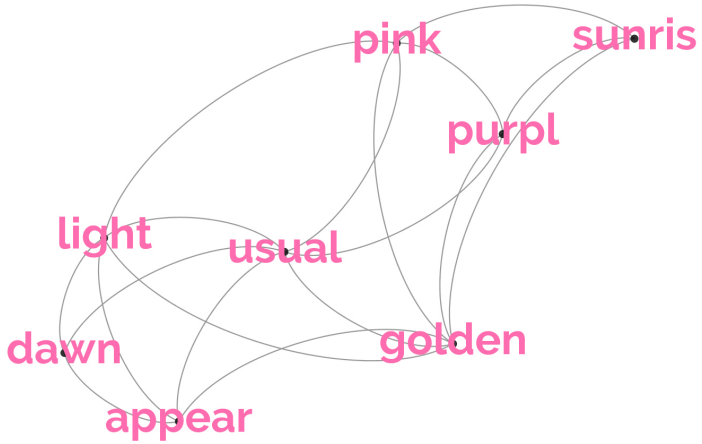
Num. node types = 1

Num. edge types = 1

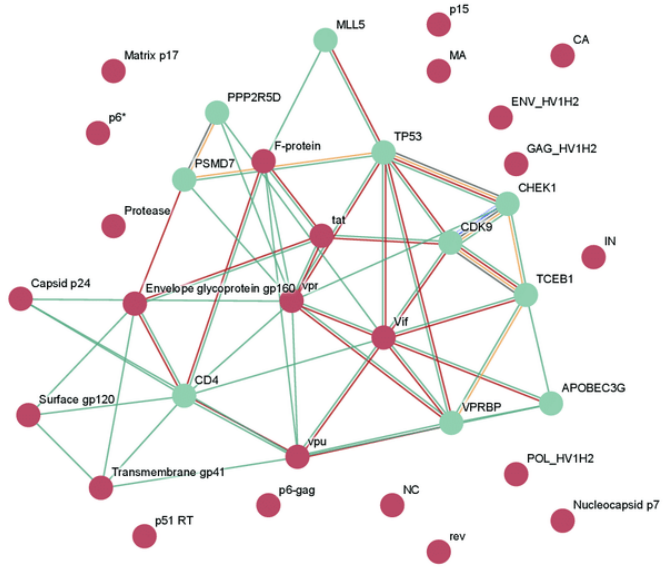


Homogeneous network

Social graph



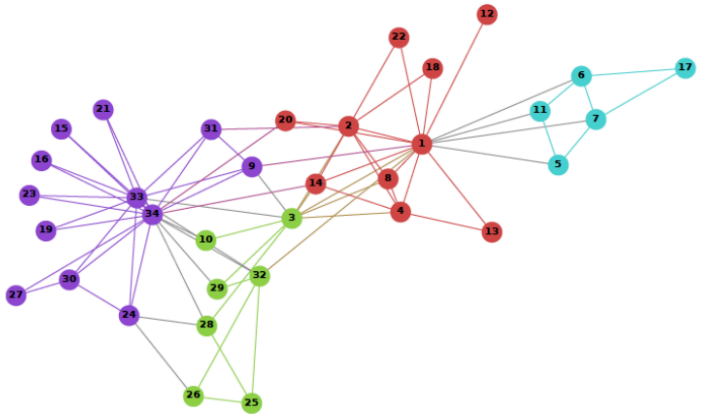
Word cooccurrence graph



Protein-Protein Interaction Graph

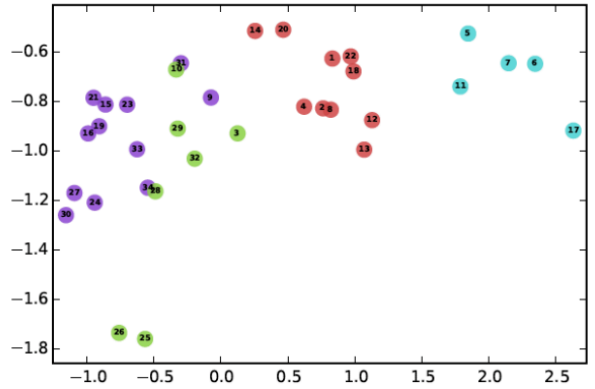
(Figure credit) https://medium.com/analytics-vidhya/social-network-analytics-f082f4e21b16
https://www.researchgate.net/publication/327854066/figure/fig2/AS:674567748075520@1537840892354/HIV-1-and-Homo-sapiens-interaction-network-in-virusesSTRING-HIV-1-and-Homo-sapiens.png
https://commons.wikimedia.org/wiki/File:Word_co-occurrence_network_(range_3_words)_-_ENG.jpg

PROBLEM DEFINITION: NODE EMBEDDING



Input

Node embedding



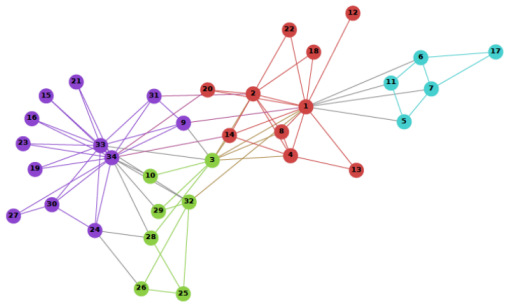
Output

- **Formal definition**
  - **Given:** A graph $G = (V, E, W)$,
    - V is the set of nodes
    - E is the set of edges between the nodes
    - W is the set of weights of the edges,
  - **Goal:** To represent each node i with a vector, which preserves the structure of networks.
- **Idea**: Similar nodes in a graph have similar vector representations

(Figure credit) https://arxiv.org/pdf/1403.6652.pdf

DEEPWALK

- Deepwalk converts a graph into a collection of node sequences through random walk
- Treat random walks on networks as sentences
- Distributional hypothesis
  - **Word embedding**: Words in similar contexts have similar meanings (e.g., skip-gram in word embedding)
  - **Node embedding**: Nodes in similar structural contexts are similar



Random walk

$$\mathcal{W}_{v_4} = v_4 \rightarrow v_3 \rightarrow v_1 \rightarrow v_5 \rightarrow v_1 \rightarrow v_{46} \rightarrow v_{51} \rightarrow v_{89}$$

$$\mathcal{W}_{v_4} = \begin{matrix} 4 \\ u_k \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{bmatrix} 3 \\ 1 \\ 5 \\ 1 \\ \vdots \end{bmatrix} v_j \longrightarrow \begin{matrix} d \\ \Phi \end{matrix} \; j$$

Maximize: $\Pr(v_3|\Phi(v_1))$

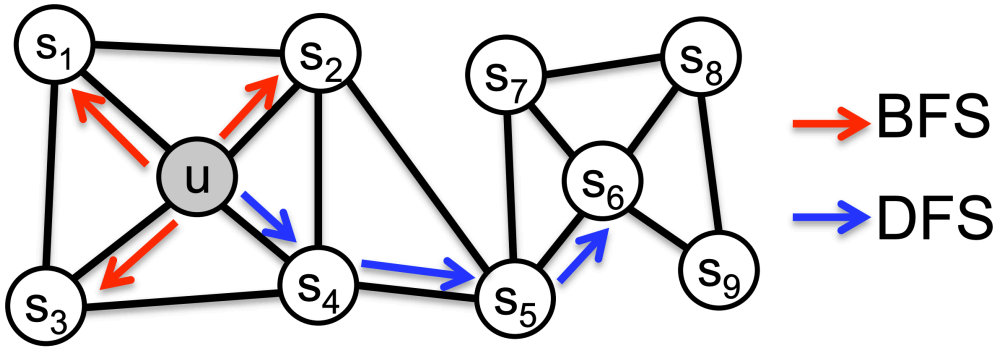
$\Pr(v_5|\Phi(v_1))$

$$p(v_j \mid v_i) = \frac{\exp(\vec{u}'^{T}_{i}\,\vec{u}_j)}{\sum_{k \in V} \exp(\vec{u}'^{T}_{k}\,\vec{u}_i)}$$

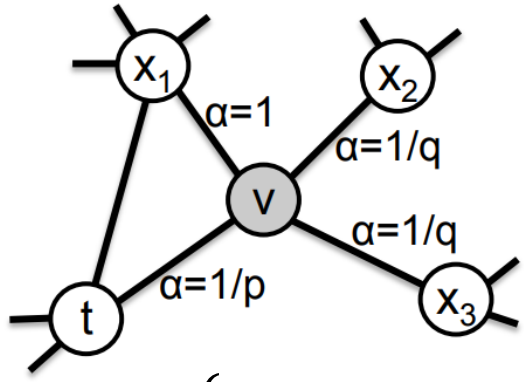
Deepwalk: Online learning of social representations, KDD2014

NODE2VEC

- **Idea**: Find the node context with a hybrid strategy of
  - Breadth-first Sampling (BFS): **Structural equivalence**
  - Depth-first Sampling (DFS): **Homophily**



- **Biased random walk** with two parameters ***p*** and ***q***
  - ***p***: controls the probability of **revisiting** a node in the walk
  - ***q***: controls the probability of **exploring** "outward" nodes
- Find optimal ***p*** and ***q*** through cross-validation on labeled data
- Interpolate between BFS and DFS



$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & if\ d_{tx} = 0 \\ 1 & if\ d_{tx} = 1 \\ \frac{1}{q} & if\ d_{tx} = 2 \end{cases}$$

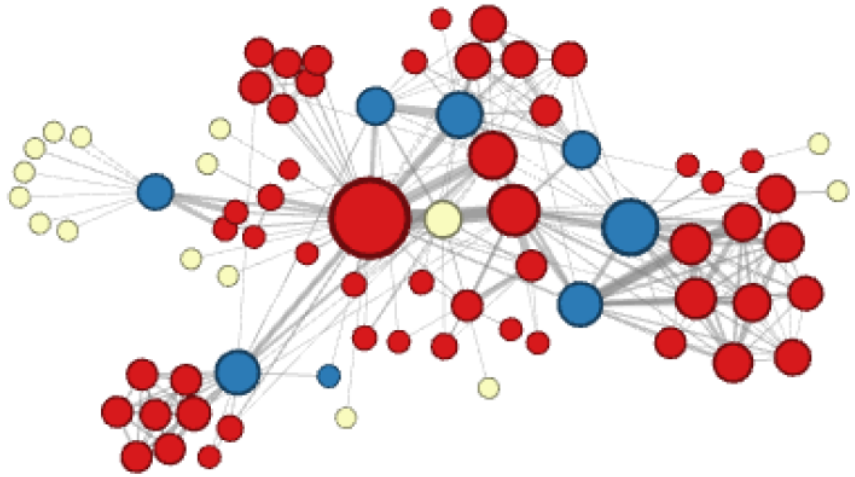
node2vec: Scalable Feature Learning for Networks, KDD2016

NODE2VEC: CASE STUDY

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & if\ d_{tx} = 0 \\ 1 & if\ d_{tx} = 1 \\ \frac{1}{q} & if\ d_{tx} = 2 \end{cases}$$

- p = 1, q = 2 → BFS-like behavior
  - Discovers structure roles

$$\alpha_{pq}(t,x) = \begin{cases} 1 & if\ d_{tx} = 0 \\ 1 & if\ d_{tx} = 1 \\ 0.5 & if\ d_{tx} = 2 \end{cases}$$



- p = 1, q = 0.5 → DFS-like behavior
  - Discovers clusters/communities

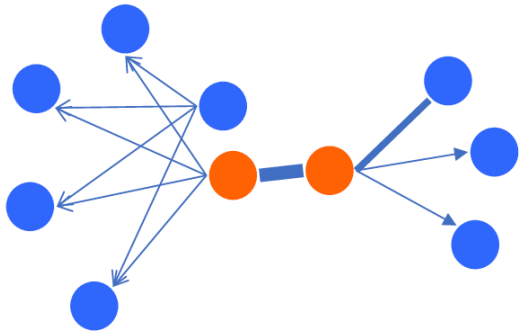
$$\alpha_{pq}(t,x) = \begin{cases} 1 & if\ d_{tx} = 0 \\ 1 & if\ d_{tx} = 1 \\ 2 & if\ d_{tx} = 2 \end{cases}$$

node2vec: Scalable Feature Learning for Networks, KDD2016

LINE: LARGE-SCALE INFORMATION NETWORK EMBEDDING

- **Idea:** To preserve the **first-order** and **second-order proximity**
- Deepwalk: DFS
- Node2vec: DFS + BFS
- LINE: BFS

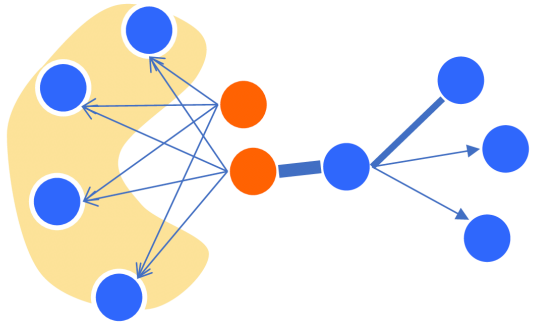
First-order Proximity



The local pairwise proximity between the nodes

- However, many links between the nodes are not observed
- Not sufficient for preserving the entire network structure

Second-order Proximity

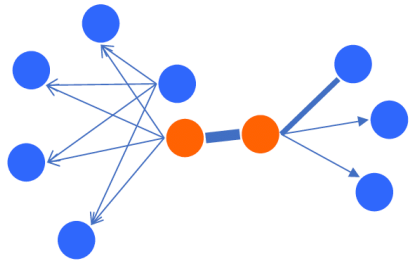


Proximity between the neighborhood structures of the nodes

LINE: Large-scale Information Network Embedding, WWW2015

PRESERVING THE PROXIMITY

First-order Proximity



Empirical distribution of first-order proximity:

$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{(m,n)\in E} w_{mn}}$$

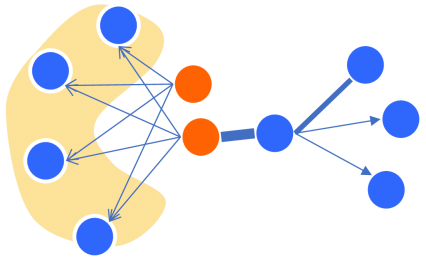
Model distribution of first-order proximity:

$$p_1(v_i, v_j) = \frac{\exp(\vec{u}_i^T \vec{u}_j)}{\sum_{(m,n)\in V\times V} \exp(\vec{u}_m^T \vec{u}_n)}$$

Objective

$$O_1 = KL(\hat{p}_1, p_1) = -\sum_{(i,j)\in E} w_{ij} \log p_1(v_i, v_j)$$

Second-order Proximity



Empirical distribution of neighborhood structure:

$$\hat{p}_2(v_j \mid v_i) = \frac{w_{ij}}{\sum_{k\in V} w_{ik}}$$

Model distribution of neighborhood structure:

$$p_2(v_j \mid v_i) = \frac{\exp(\vec{u}'^T_i \vec{u}_j)}{\sum_{k\in V} \exp(\vec{u}'^T_k \vec{u}_i)}$$

Objective

$$O_2 = \sum_i KL(\hat{p}_2(\cdot \mid v_i), p_2(\cdot \mid v_i))$$

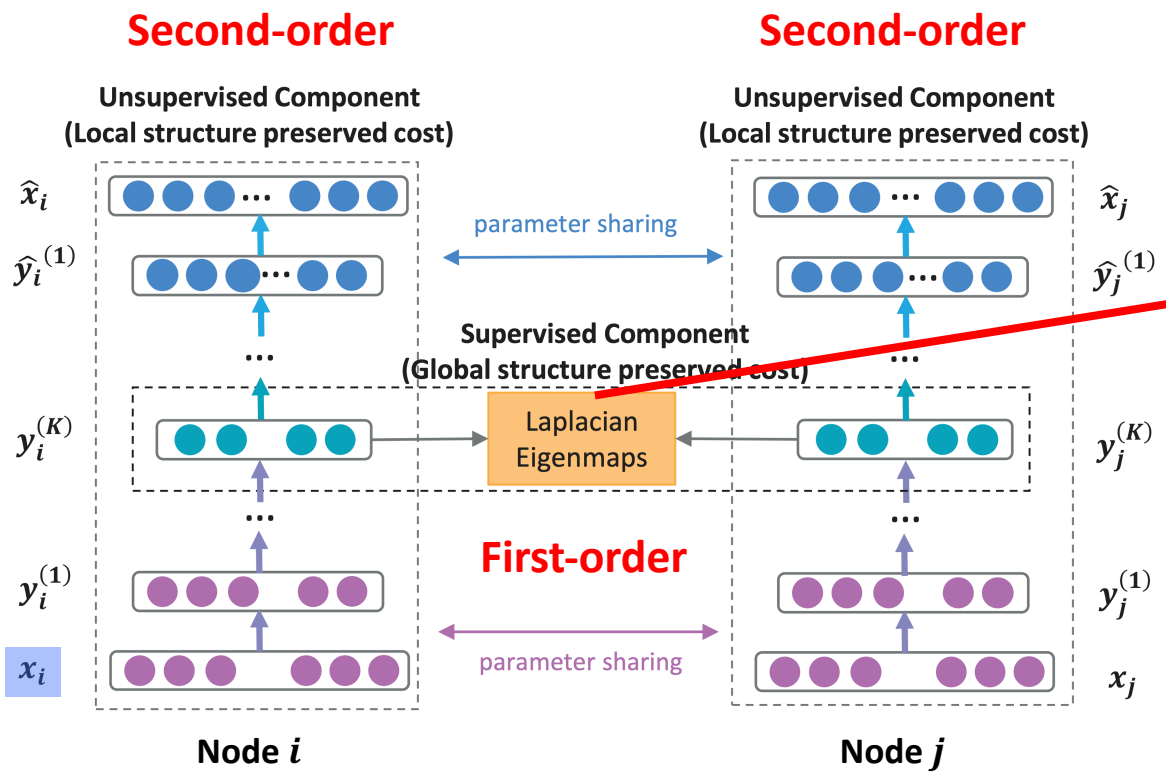
$$= -\sum_{(i,j)\in E} w_{ij} \log p_2(v_j \mid v_i)$$

LINE: Large-scale Information Network Embedding, WWW2015

SDNE - STRUCTURAL DEEP NETWORK EMBEDDING

- **Idea:** Shallow models cannot capture the highly non-linear network structure
  - Deepwalk, node2vec, LINE are all shallow models

$$\mathcal{L}_{2nd} = \sum_{i=1}^{n} \|\hat{\mathbf{x}}_i - \mathbf{x}_i\|_2^2$$



Neighborhood structure of node i

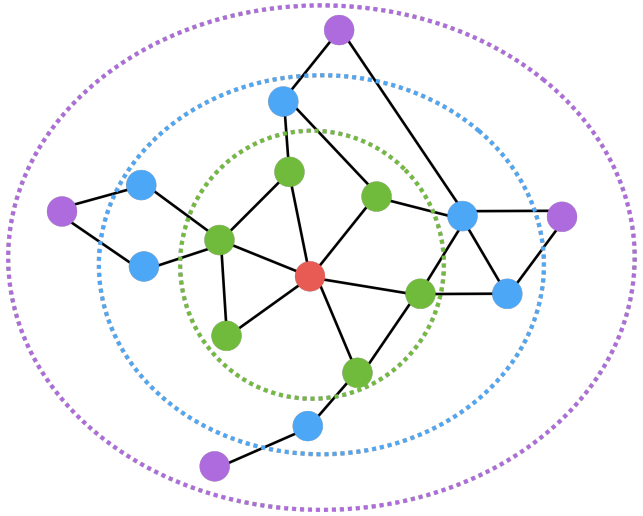
$$\mathcal{L}_{1st} = \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i^{(K)} - \mathbf{y}_j^{(K)}\|_2^2 = \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2$$

$$\mathcal{L}_{mix} = \mathcal{L}_{2nd} + \alpha \mathcal{L}_{1st}$$

Structural Deep Network Embedding, KDD2016

GRAREP

- **Idea**: Consider **k-hop node neighbors**



- Red: Target node
- Green: 1-hop neighbors
  - $\mathbf{A}$ (i.e., adjacency matrix)
- Blue: 2-hop neighbors
  - $\mathbf{A^2}$
- Purple: 3-hop neighbors
  - $\mathbf{A^3}$

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

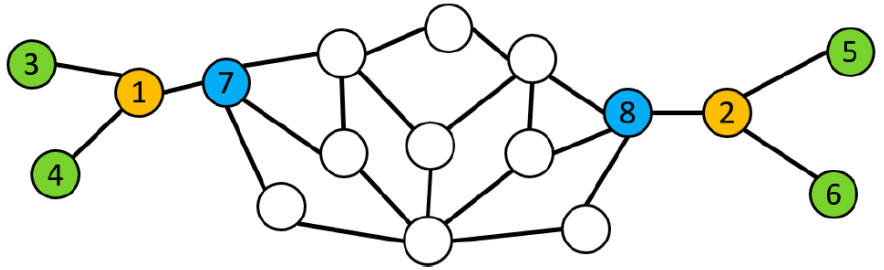
Concatenate all the k-step representations $[Z^1, Z^2, \ldots, Z^K]$

GraRep: Learning Graph Representations with Global Structural Information, CIKM2015

(Figure credit) Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

DEEP RECURSIVE NETWORK EMBEDDING WITH REGULAR EQUIVALENCE

- **Idea**: To preserve **regular equivalence**
  - Example of regular equivalence (node 7 and node 8 are regularly equivalent)



- **Approach**: The definition of regular equivalence is **recursive**
  - If two nodes are regularly equivalent, then their neighbors are also regularly equivalent
  - **Solution**: Represent embedding of one node by the aggregation of its neighbors' embeddings.

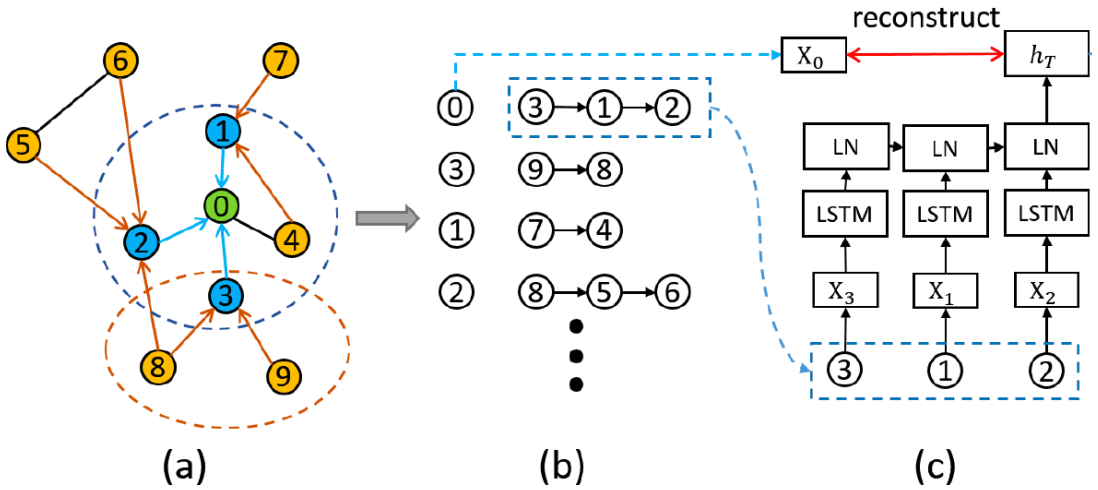
$$\mathcal{L}_1 = \sum_{v \in V} ||\mathbf{X}_v - Agg(\{\mathbf{X}_u | u \in \mathcal{N}(v)\})||_F^2,$$

Deep Recursive Network Embedding with Regular Equivalence, KDD2018

DEEP RECURSIVE NETWORK EMBEDDING WITH REGULAR EQUIVALENCE

- How to design the aggregation function? **LSTM!** (Variable size of neighbors)

$$\mathcal{L}_1 = \sum_{v \in V} ||\mathbf{X}_v - Agg(\{\mathbf{X}_u | u \in \mathcal{N}(v)\})||_F^2,$$



- **Regularization**
  - The model may degenerate to the trivial solution with all the embeddings being 0.
  - **How can we avoid the trivial solution?**
  - Use node degree as the weakly guided information
    - The learned embedding of a node should be able to approximate the degree of the node

$$\mathcal{L}_{reg} = \sum_{v \in V} ||\log(d_v + 1) - MLP(Agg(\{\mathbf{X}_u | u \in \mathcal{N}(v)\}))||_F^2,$$

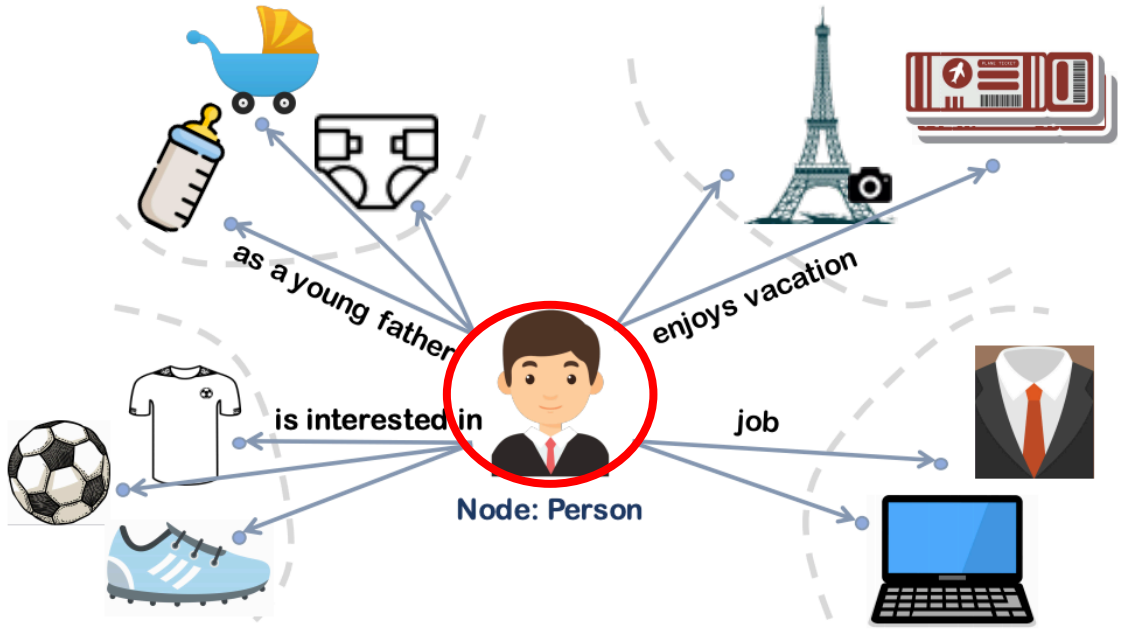
$$\mathcal{L} = \mathcal{L}_1 + \lambda \mathcal{L}_{reg}$$

Deep Recursive Network Embedding with Regular Equivalence, KDD2018

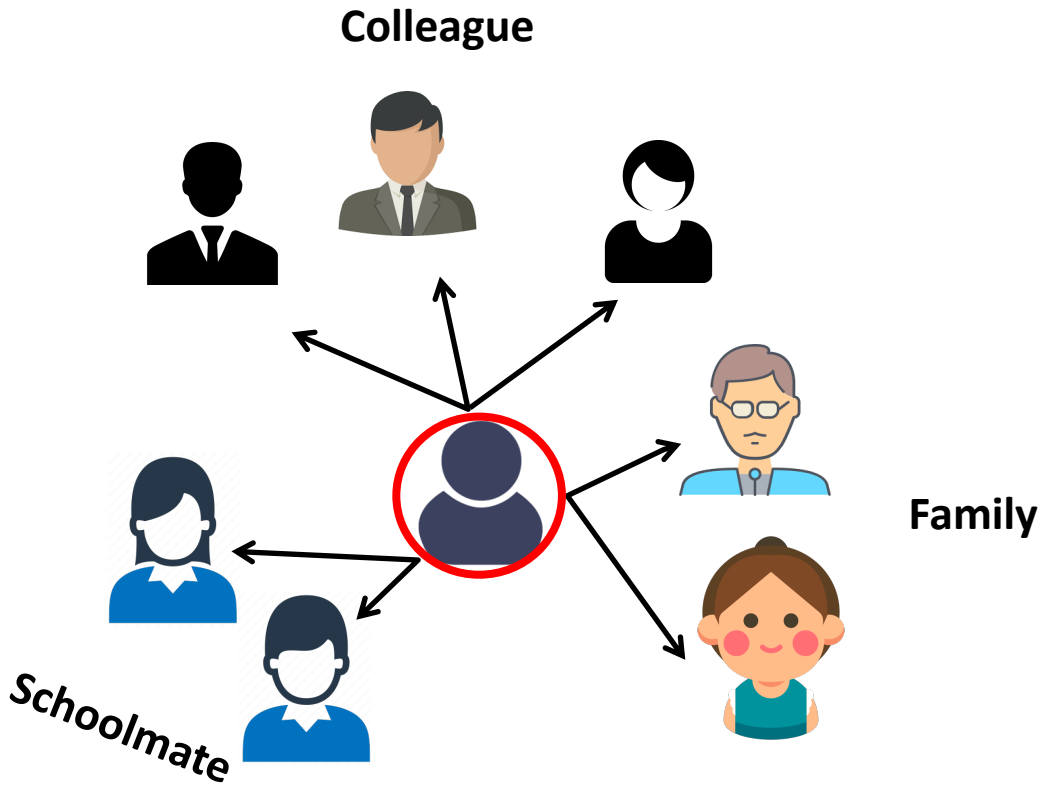
OUTLINE

- Homogeneous Network Embedding
- Multi-aspect Network Embedding
- Attributed Network Embedding
- Heterogeneous Network Embedding
- Training GNN
  - Self-supervised learning
  - Going deeper with GNN
- Applications of Graph Machine Learning

IS A SINGLE REPRESENTATION ENOUGH?



Purchase history



Social network

How to differentiate among multiple aspects?

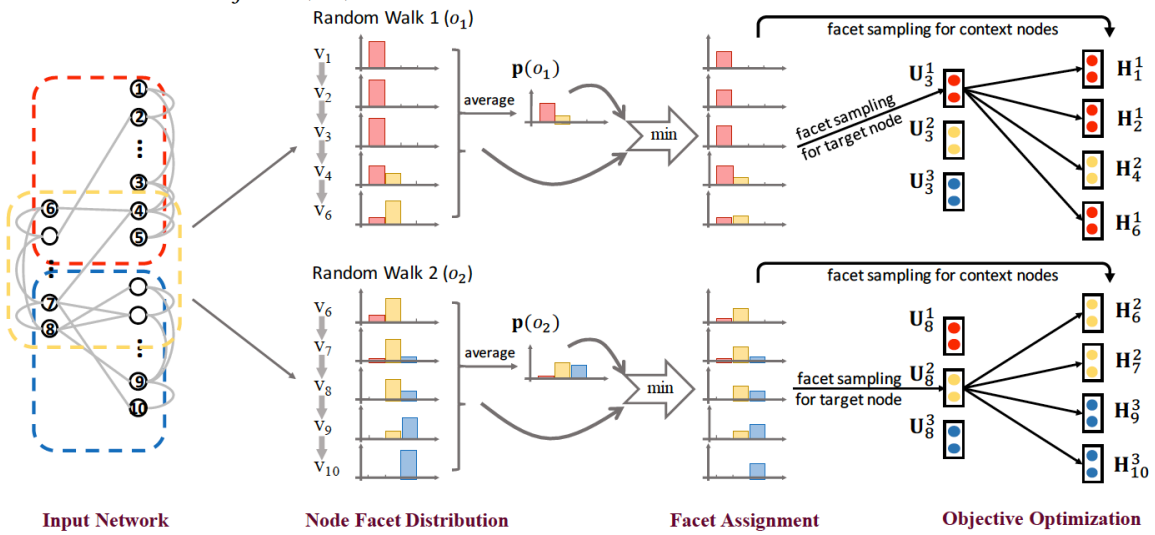
(Figure credit) Is a single vector enough? exploring node polysemy for network embedding, KDD19

POLYDW

- Define the aspect (sense) of each node by clustering the adjacency matrix (offline clustering)
- For each node and its context nodes, sample an aspect
- Update the node embeddings of the sampled aspect only

Deepwalk

$$\mathcal{L}_{DW}(\theta) = \sum_{o \in O} \log\ p(o|\theta) = \sum_{o \in O} \log\ p((\mathcal{N}(v_i), v_i)|\theta)$$
$$= \sum_{o \in O} \sum_{v_j \in \mathcal{N}(v_i)} \log\ p(v_j|v_i),$$



PolyDW

$$\mathcal{L}_{PolyDW}(\theta) = \sum_{o \in O} \log\ p(o|\mathcal{P}, \theta)$$

Prior (obtained from clustering)

$$= \sum_{o \in O} \log\ [\sum_{s(o)} p(o|s(o), \mathcal{P}, \theta) \cdot p(s(o)|\mathcal{P}, \theta)]$$
$$\geq \sum_{o \in O} \sum_{s(o)} p(s(o)|\mathcal{P}, \theta) \cdot \log\ p(o|s(o), \mathcal{P}, \theta)$$
$$= \sum_{o \in O} \sum_{s(o)} p(s(o)|\mathcal{P}) \cdot [\sum_{v_j \in \mathcal{N}(v_i)} \log\ p(v_j|v_i, s(o))]$$

Cluster membership

$o = (v, N(v))$
$N(v)$: context of node v
$s(o)$: A set of possible aspects within an observation o

ASP2VEC

- Adopt the **Gumbel-softmax trick** to **dynamically sample aspects based on the context**
  - PolyDW: Offline clustering

Gumbel-softmax

Embedding of v_i

Embedding of $N(v_i)$ regarding aspect s

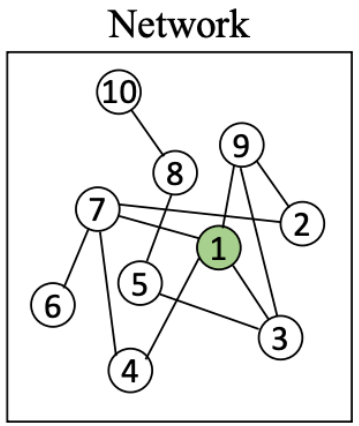
Aspect of node v_i

$$p(\quad | \mathcal{N}(v_i)) = \frac{\exp[\langle \mathbf{P}_i, \text{Readout}^{(s)}(\mathcal{N}(v_i))\rangle + g_s]/\tau}{\sum_{s'=1}^{K} \exp[\langle \mathbf{P}_i, \text{Readout}^{(s')}(\mathcal{N}(v_i))\rangle + g_{s'}]/\tau}$$

Local context of v_i

Sample the aspect that gives the highest value
(Continuous relaxation of discrete random variable **)**

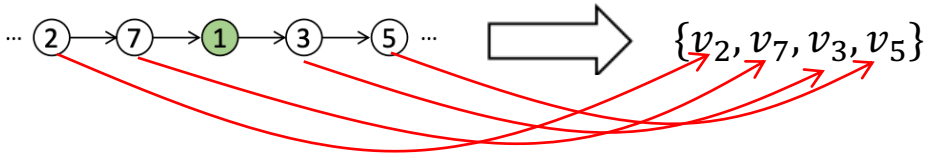
Probability of v_i being selected as aspect s given its context $N(v_i)$



Random walk

Target: v_1

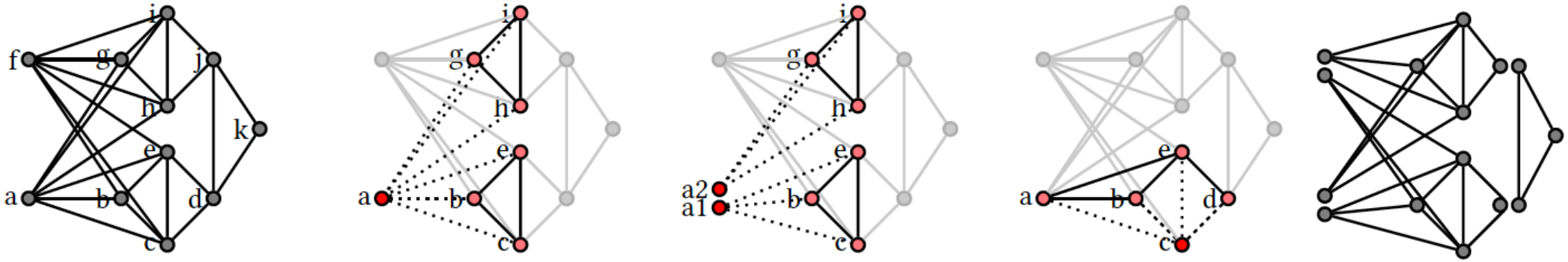
Context ($N(v_1)$)



Unsupervised Differentiable Multi-aspect Network Embedding, KDD2020

SPLITTER

- Given an original graph, compute a **persona graph**
  - Add constraints on Deepwalk to relate the persona graph with the original graph



(a) original graph G (b) ego-net of a (c) splitting a in two personas (d) ego-net of c (one persona) (e) persona graph

$$\underset{\Phi_{G_P}}{\text{minimize}} \quad -\log \Pr\left(\{v_{i-w}, \cdots, v_{i+w}\} \setminus v_i \mid \Phi_{G_P}(v_i)\right) - \lambda \log \Pr\left(v_o \mid \Phi_{G_P}(v_i)\right).$$

Predict the context of v_i using the persona of v_i

Predict the original embedding of v_i using its persona

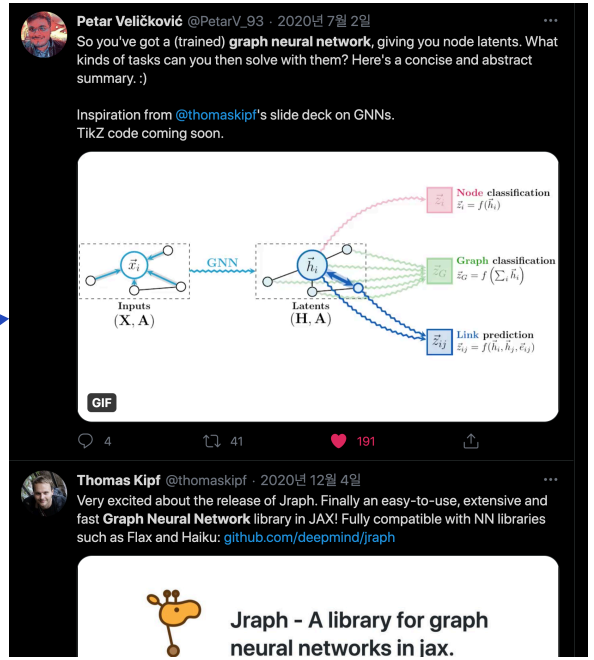
Is a single embedding enough? learning node representations that capture multiple social contexts, WWW2019

OUTLINE

- Homogeneous Network Embedding
- Multi-aspect Network Embedding
- Attributed Network Embedding
- Heterogeneous Network Embedding
- Training GNN
  - Self-supervised learning
  - Going deeper with GNN
- Applications of Graph Machine Learning

ATTRIBUTED NETWORK: EXAMPLE OF NODE ATTRIBUTES

- Any type of information that is related to a node within a graph
- **Example**
  - User content in social graph
  - Reviews in e-commerce graph
  - Paper abstracts in citation graph
  - Product image in e-commerce graph
  - …



Top reviews from the United States

Ralph E.

★☆☆☆☆ **Do not recommend....VERY short life!**

Reviewed in the United States on September 16, 2017

Verified Purchase

I was very pleased with the quality of the product as I received it. But after about one week of charging it became intermittent for a couple of days and then stopped working. I think it is important to point out that this cord is in one location, I sit my phone on a table surface and let it charge all night. So no moving, no using it while connected or such. I also an not the type of person that unplugs it by pulling on the cored or such. This just proved to me that they cord was either defective as received or poor quality (or maybe a combo of both). Now it is much cheaper than an APPLE cord, but I would expect more. Wish it would have done better....I had I hopes being an Amazon product!

392 people found this helpful

Helpful | Report abuse

Unsupervised Differentiable Multi-aspect Network Embedding

Chanyoung Park[1], Carl Yang[2], Qi Zhu[1], Donghyun Kim[4], Hwanjo Yu[3*], Jiawei Han[1]
[1]Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA
[2]Emory University, GA, USA, [3]POSTECH, Pohang, South Korea, [4]Yahoo! Research, CA, USA
{pcy1302,qiz3,hanj}@illinois.edu,j.carlyang@emory.edu,donghyun.kim@verizonmedia.com,hwanjoyu@postech.ac.kr

ABSTRACT

Network embedding is an influential graph mining technique for representing nodes in a graph as distributed vectors. However, the majority of network embedding methods focus on learning a single vector representation for each node, which has been recently criticized for not being capable of modeling multiple aspects of a node. To capture the multiple aspects of each node, existing studies mainly rely on offline graph clustering performed prior to the actual embedding, which results in the cluster membership of each node (i.e., node aspect distribution) fixed throughout training of the embedding model. We argue that this not only makes each node always have the same aspect distribution regardless of its dynamic context, but also hinders the end-to-end training of the model that eventually leads to the final embedding quality largely dependent on the clustering. In this paper, we propose a novel end-to-end framework for multi-aspect network embedding, called asp2vec, in which the aspects of each node are dynamically assigned based on its local context. More precisely, among multiple aspects, we dynamically assign a *single* aspect to each node based on its current context, and our aspect selection module is end-to-end differentiable via the Gumbel-Softmax trick. We also introduce the aspect regularization framework to capture the interactions among the multiple aspects in terms of relatedness and diversity. We further demonstrate that our proposed framework can be readily extended to heterogeneous networks. Extensive experiments towards various

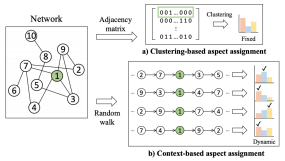


Figure 1: a) Clustering–based aspect assignment that fixes the aspect distribution during the embedding learning b) asp2vec dynamically selects a single aspect based on the local context nodes.

1 INTRODUCTION

Networks constitute a natural paradigm to represent real-world relational data that contain various relationships between entities ranging from online social network of users, and academic publication network of authors, to protein-protein interaction (PPI) network in the physical world. Due to the pervasive nature of networks, analyzing and mining useful knowledge from networks has been an actively researched topic for the past decades. Among various tools for network analysis, *network embedding*, which learns continuous vector representations for nodes in a network, has re-

CHALLENGE IN ATTRIBUTED NETWORK EMBEDDING

- Node **attributes** and **network** influence each other and are **inherently correlated**
  - Ex. High correlation of user posts and friend relationships

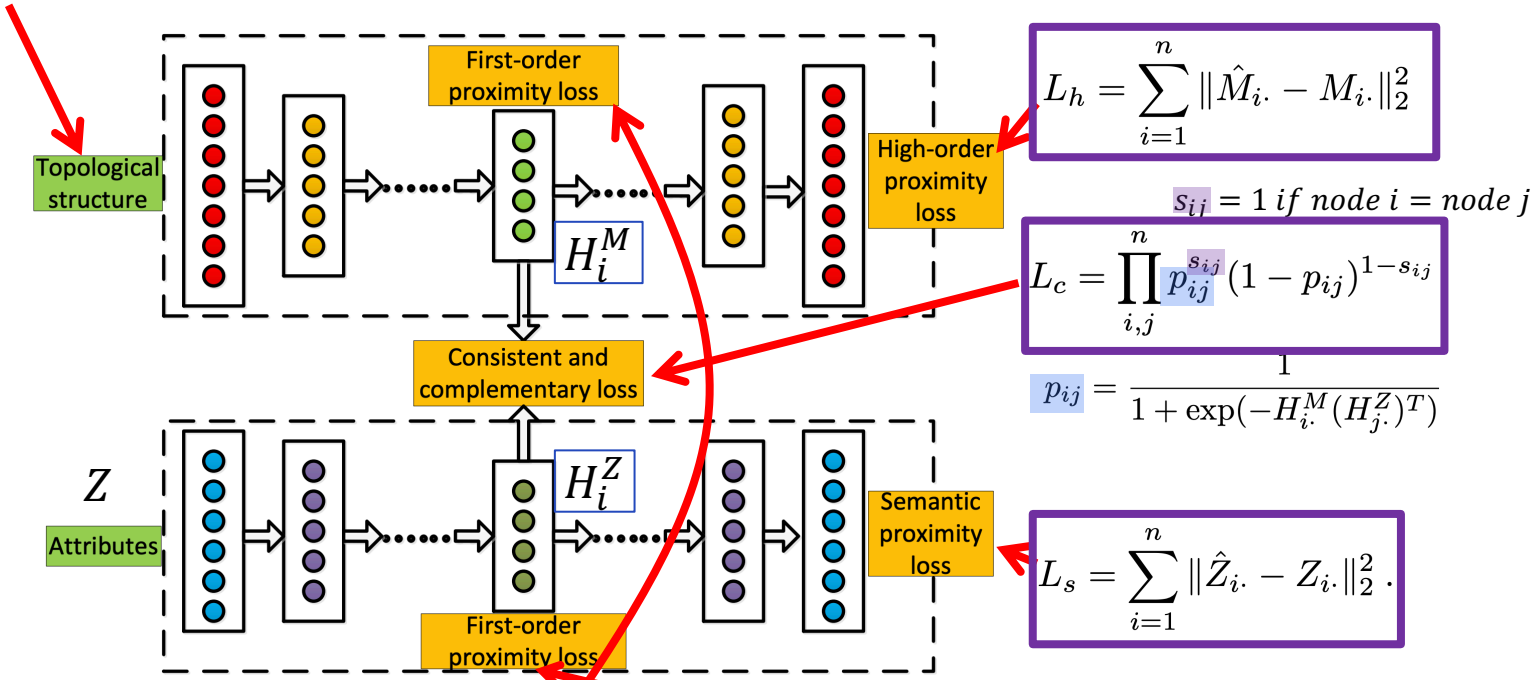
- How to jointly model two different modalities: **graph topology** + **node attributes**

- **Highly complex** (large scale)
  - Number of nodes and dimension of attributes could be large

DEEP ATTRIBUTED NETWORK EMBEDDING (DANE)

- **Idea**: The proximity in an attributed network depends on not only the **topological** structure but also the **attribute**

$M = A + A^2 + \cdots + A^t$ **(Capture up to *t*-hop neighbors)**



$$L_h = \sum_{i=1}^{n} \|\hat{M}_{i\cdot} - M_{i\cdot}\|_2^2$$

$s_{ij} = 1 \; if \; node \; i = node \; j$

$$L_c = \prod_{i,j}^{n} p_{ij}^{s_{ij}} (1 - p_{ij})^{1 - s_{ij}}$$

$$p_{ij} = \frac{1}{1 + \exp(-H_{i\cdot}^M (H_{j\cdot}^Z)^T)}$$

$$L_s = \sum_{i=1}^{n} \|\hat{Z}_{i\cdot} - Z_{i\cdot}\|_2^2 .$$

$$L_f = -\sum_{E_{ij}>0} \log p_{ij}^M - \sum_{E_{ij}>0} \log p_{ij}^Z$$

$$p_{ij}^M = \frac{1}{1 + \exp(-H_{i\cdot}^M (H_{j\cdot}^M)^T)}$$

$$p_{ij}^Z = \frac{1}{1 + \exp(-H_{i\cdot}^Z (H_{j\cdot}^Z)^T)}$$

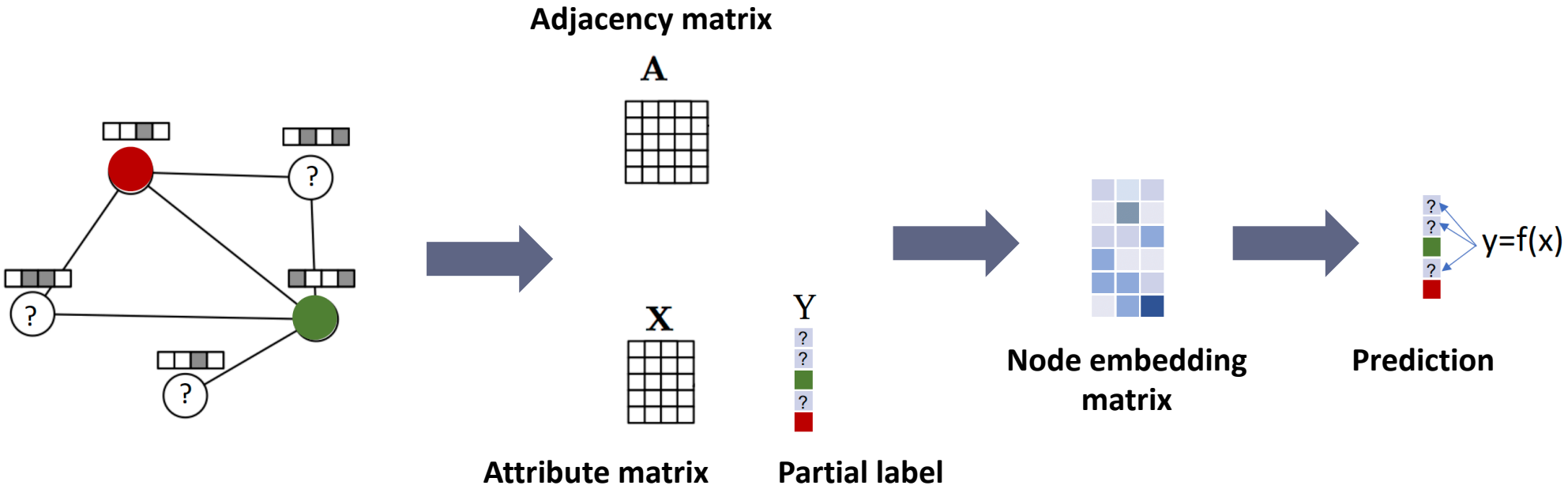
$$L = L_f + L_h + L_c + L_s$$

- The topological structure and attributes are the <u>two modal information of the same network</u>
  - → Learned representations should be **consistent** and **complementary**

Deep Attributed Network Embedding, IJCAI2018

SEMI-SUPERVISED/SUPERVISED GRAPH REPRESENTATION LEARNING

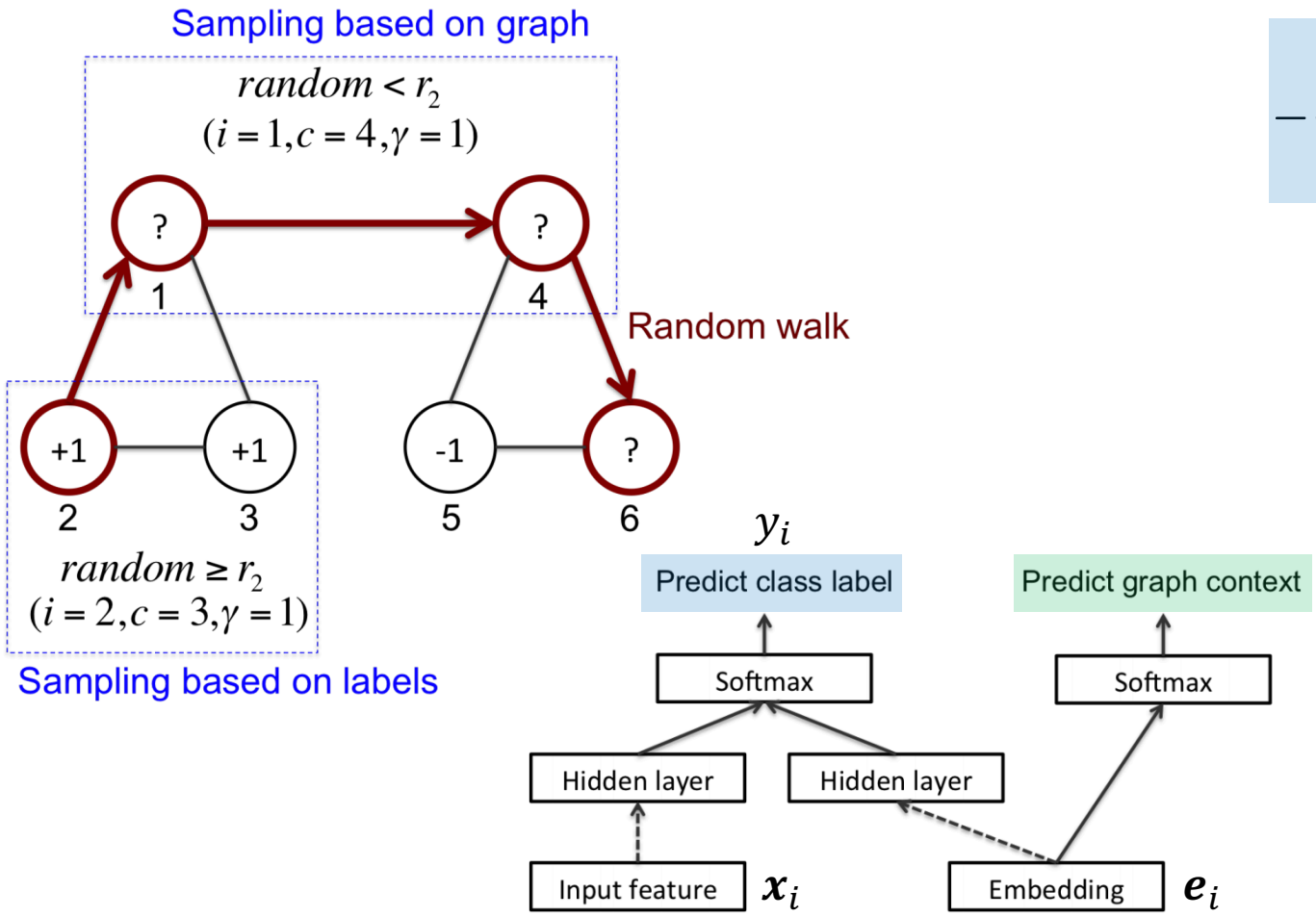
- So far, we have looked at **unsupervised** graph representation learning methods
  - Mainly focused on preserving the proximity or graph structure
- What about when we are given some supervised tasks?
  - E.g., node classification
- **Goal**: Learning node representations for specific tasks considering **node label** and **attribute information**



(Figure credit) https://chuxuzhang.github.io/KDD20_Tutorial.html

PLANETOID

- **Idea**: Random walk based on **graph structure** and **labels** (An extension of Deepwalk)



$$-\frac{1}{L}\sum_{i=1}^{L}\log p(y_i|\mathbf{x}_i,\mathbf{e}_i) - \lambda\mathbb{E}_{(i,c,\gamma)}\log\sigma(\gamma\mathbf{w}_c^T\mathbf{e}_i),$$

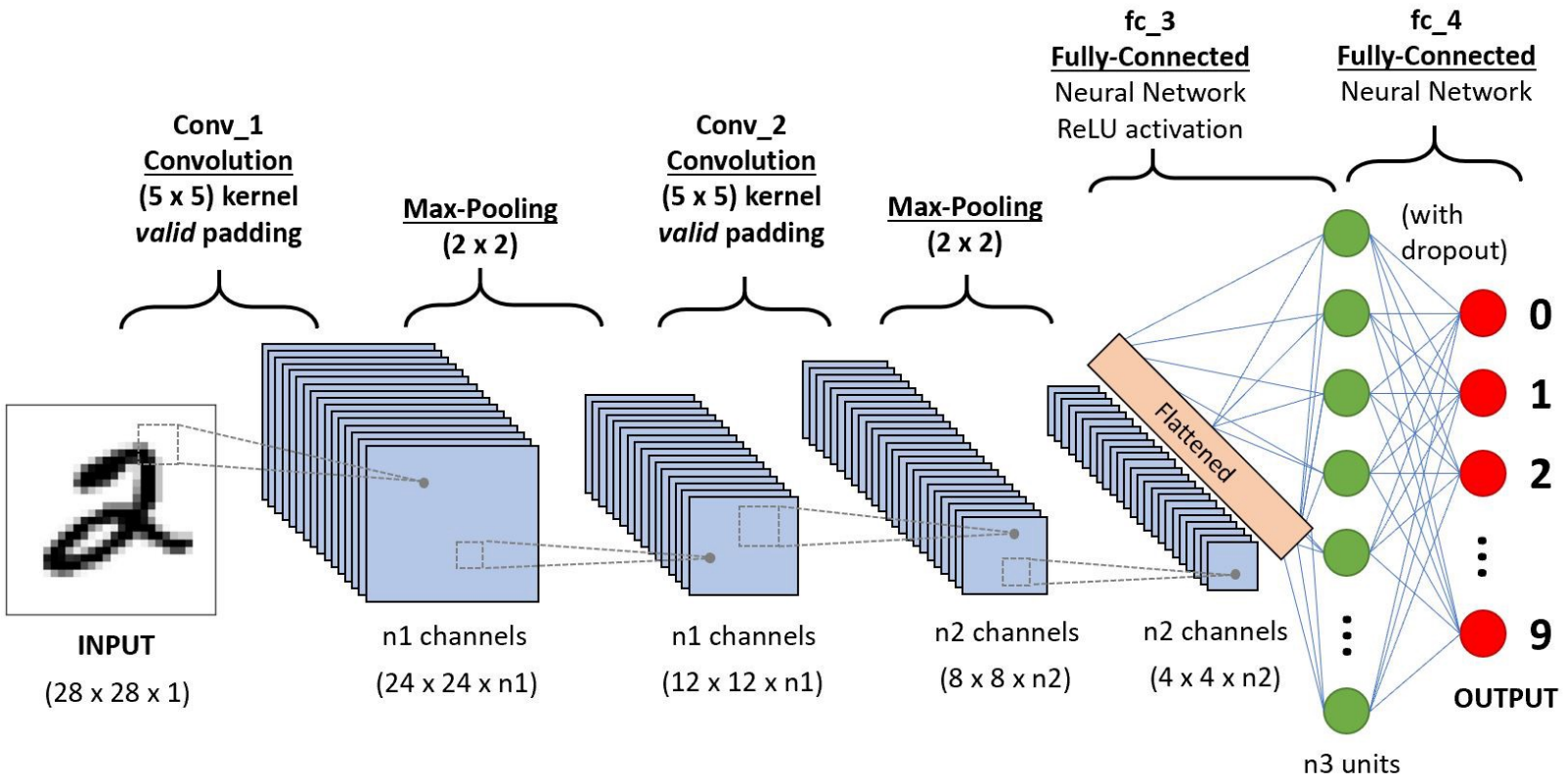
Supervised part **Deepwalk**

$$p(y|\mathbf{x},\mathbf{e}) = \frac{\exp[\mathbf{h}^k(\mathbf{x})^T,\mathbf{h}^l(\mathbf{e})^T]\mathbf{w}_y}{\sum_{y'}\exp[\mathbf{h}^k(\mathbf{x})^T,\mathbf{h}^l(\mathbf{e})^T]\mathbf{w}_{y'}},$$

Revisiting Semi-Supervised Learning with Graph Embeddings, ICML2016

BACKGROUND: CONVOLUTIONAL NEURAL NETWORKS FOR IMAGES

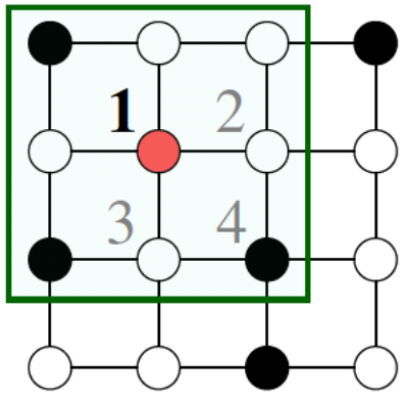
- Convolutional filters
  - Local feature detectors
  - A feature is learned in each **local receptive field** by a convolutional filter



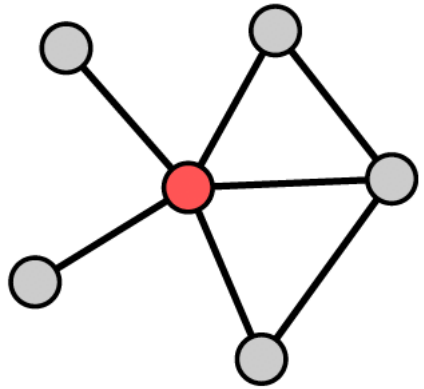
(Figure credit) https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

LOCAL RECEPTIVE FIELD ON GRAPHS

- How should we define local receptive fields on graphs?
  - Local subgraphs
- However, there are no orders between the neighbors
  - In images, the neighbors of a node can follow specific order



Image

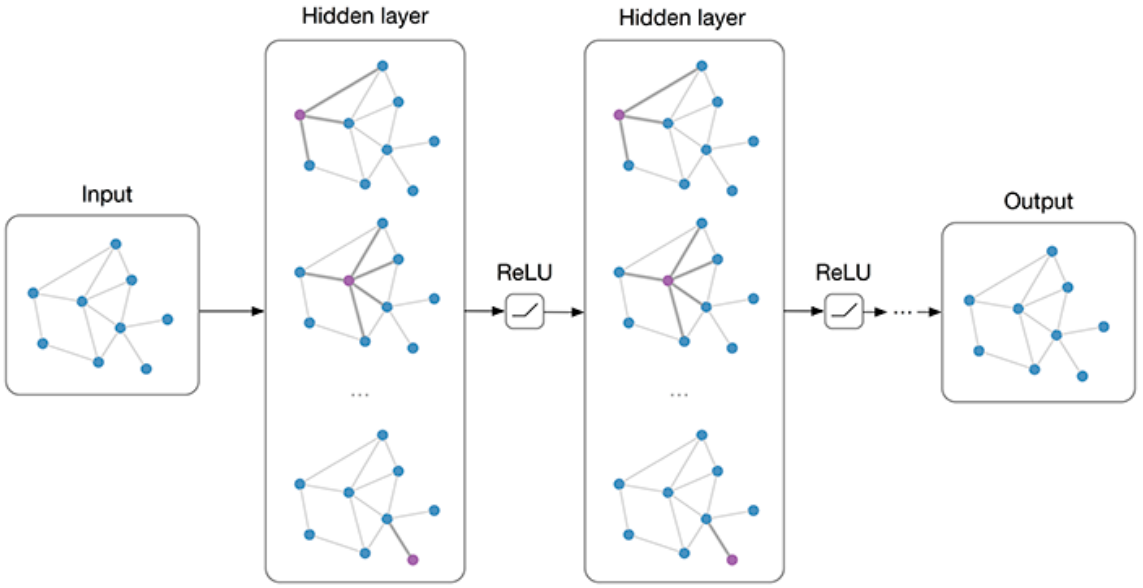


Graph

(Figure credit) https://deepgraphlearning.github.io/coursewebsite/schedule

GRAPH CONVOLUTIONAL NETWORK (GCN)

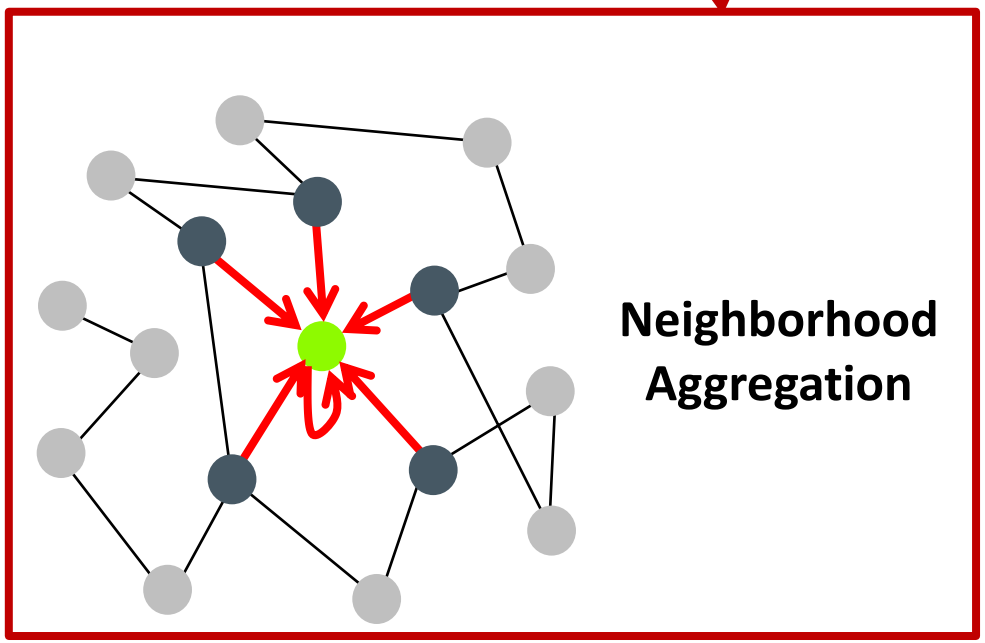
- Node features: $X \in \mathbb{R}^{n \times F}$
- Adjacency matrix: $A \in \mathbb{R}^{n \times n}$
- Add self link: $\tilde{A} = A + I_N$
- Degree matrix: $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
- Weight matrix at layer l: $W^{(l)} \in \mathbb{R}^{F \times d}$



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

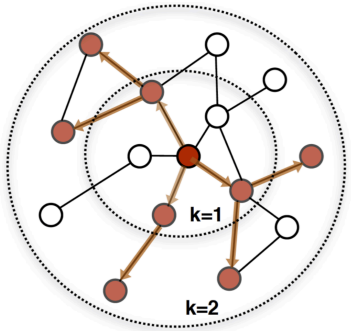
$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

Set of node indices with labels

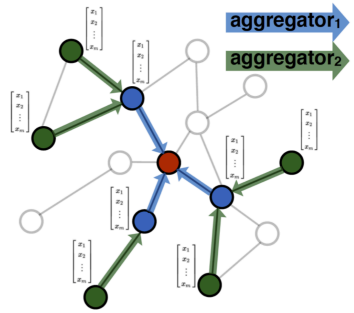


GRAPH SAMPLE AND AGGREGATE (GRAPHSAGE)

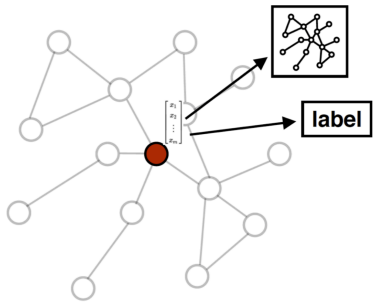
- **Motivation**: Can we train GCN more efficiently?
- **Idea**: Sample neighbors



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

Concatenate

Neighborhood embedding | Self embedding

$$\boldsymbol{h}_v^k = \sigma([\boldsymbol{W}^k \cdot AGG(\{\boldsymbol{h}_u^{k-1}, \forall u \in N(v)\}), \boldsymbol{B}^k \boldsymbol{h}_v^{k-1}])$$

Generalized aggregation

- **Variants of AGG**
  - Mean:

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$$

  - Pool
    - Transform neighbor vectors and apply symmetric vector function.

element-wise mean/max

$$\text{AGG} = \gamma\left(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)$$

  - LSTM:
    - Apply LSTM to random permutation of neighbors.

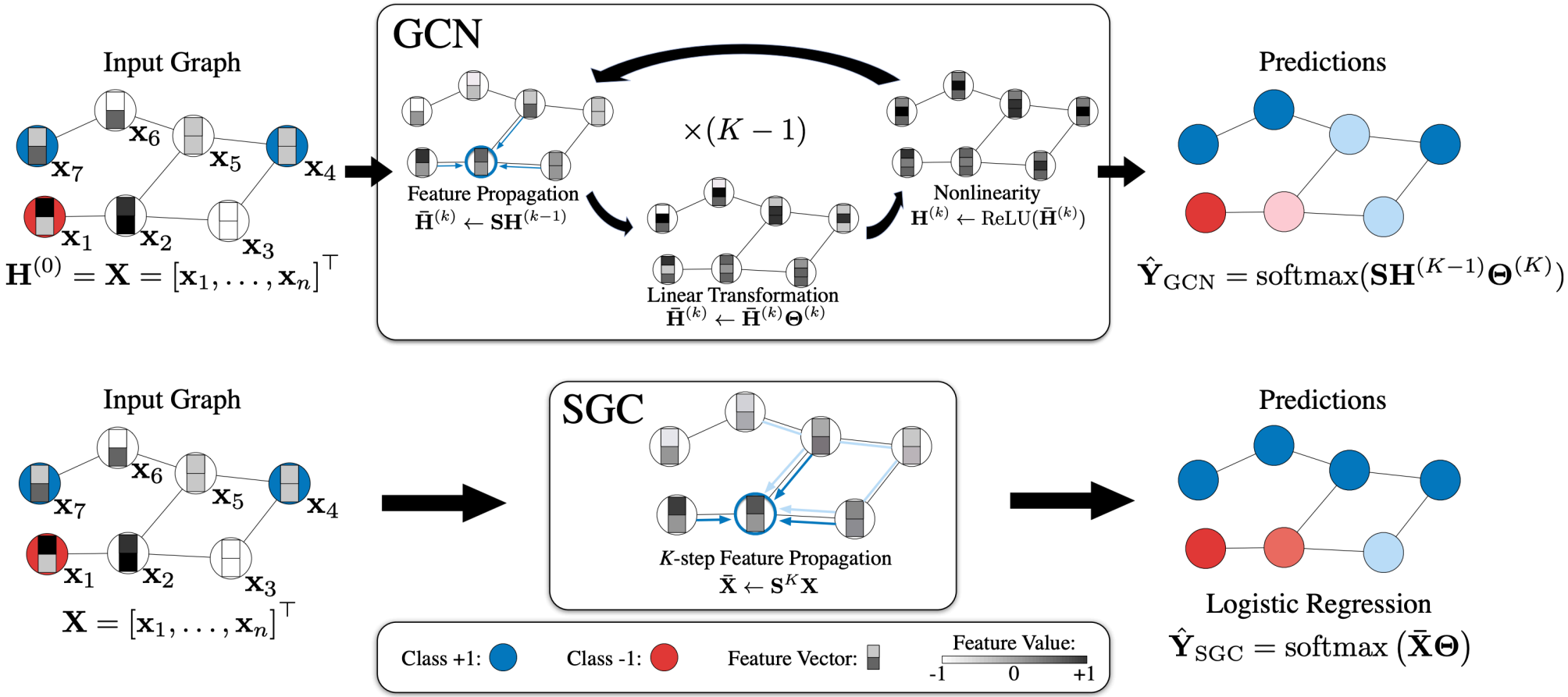
$$\text{AGG} = \text{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

Inductive Representation Learning on Large Graphs, NeurIPS2017

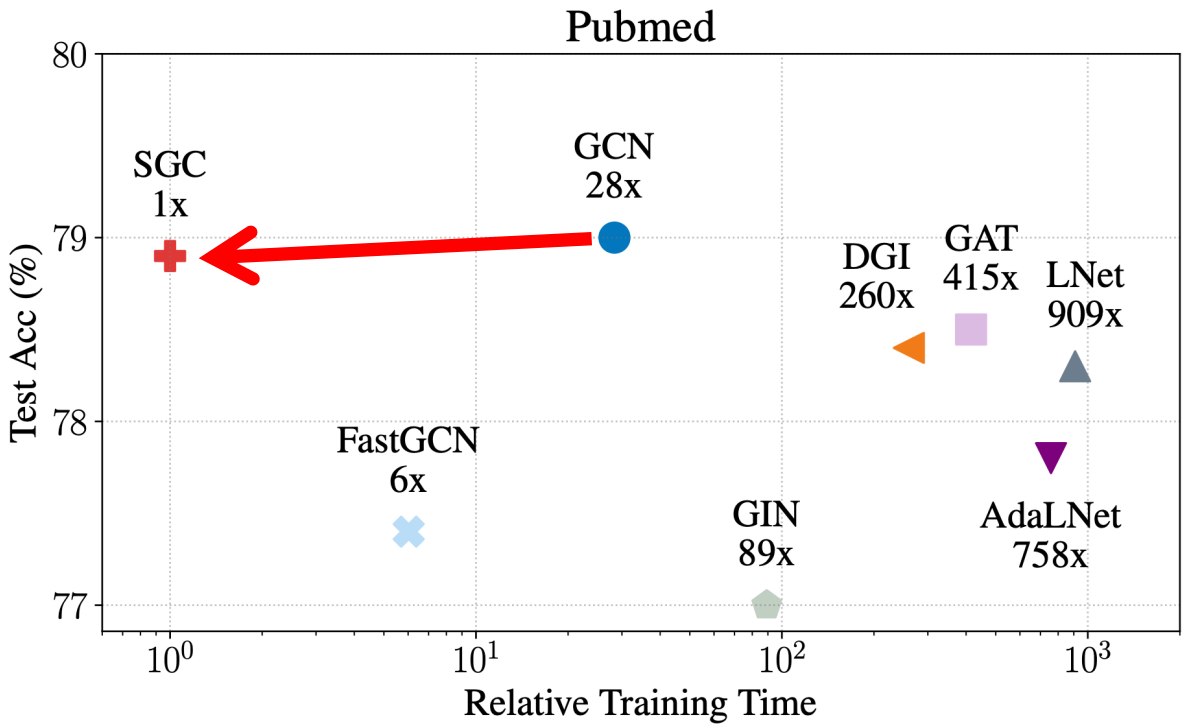
(Figure credit) Slide snipping from "Hamiltion & Tang, AAAI 2019 Tutorial on Graph Representation Learning"

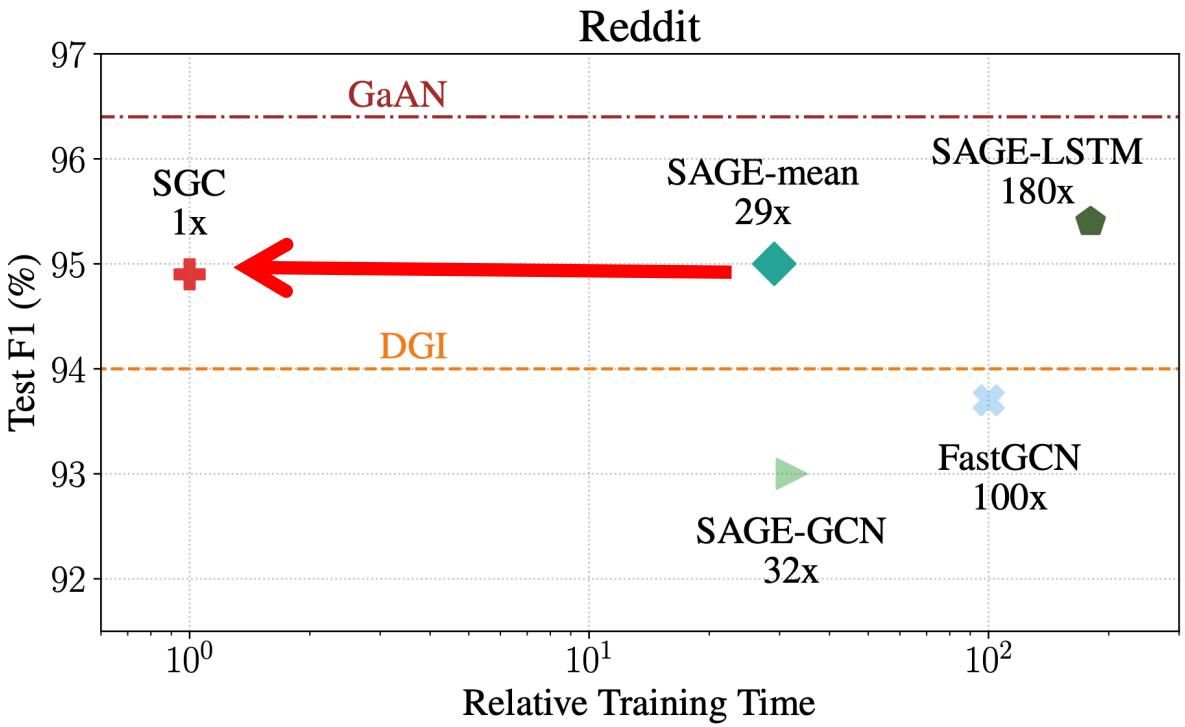
SGCN

- **Motivation**: Can we train GCN more efficiently?
- **Idea**: Remove unnecessary complexity and redundant computation (non-linearity and weight matrices)

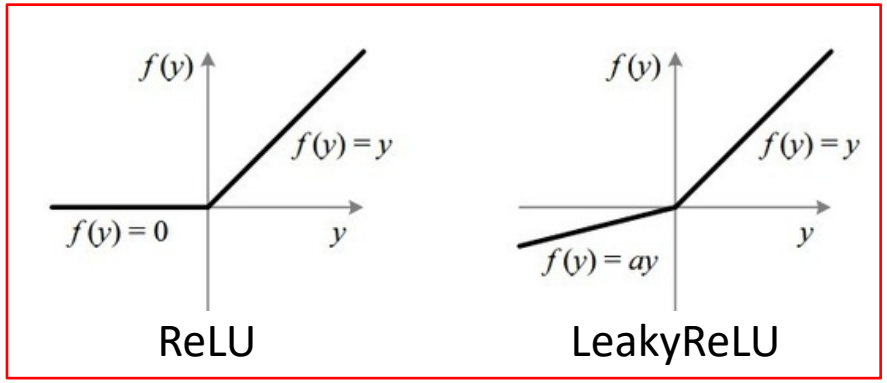


SGCN


Pubmed
Test Acc (%)
80
79
78
77
SGC
1x
GCN
28x
DGI
260x
GAT
415x
LNet
909x
FastGCN
6x
GIN
89x
AdaLNet
758x
10^0
10^1
10^2
10^3
Relative Training Time


Reddit
Test F1 (%)
97
96
95
94
93
92
GaAN
DGI
SGC
1x
SAGE-mean
29x
SAGE-LSTM
180x
FastGCN
100x
SAGE-GCN
32x
10^0
10^1
10^2
Relative Training Time

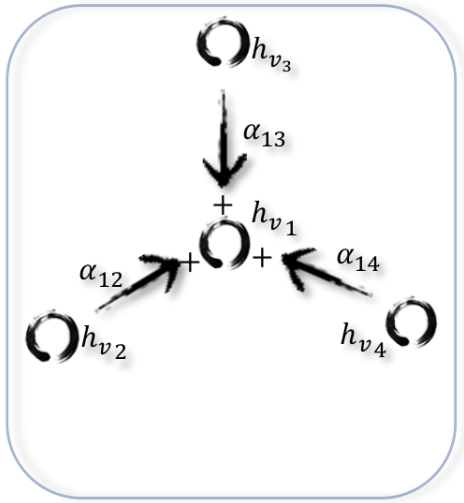
GRAPH ATTENTION NETWORKS (GAT)



ReLU LeakyReLU

- **Idea:** We assign *higher weights to more important nodes*

$$\boldsymbol{H}^{(l+1)} = \sigma\left(\sum_{\boldsymbol{j}\in N(i)} \alpha_{ij}\boldsymbol{H}_j^{(l)}\boldsymbol{W}^{(l)}\right)$$



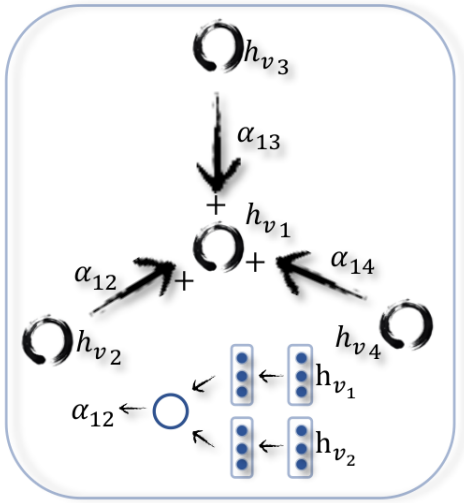
Aggregate info from neighborhood via the **normalized Laplacian matrix**

GCN

$$\alpha_{ij} = \frac{1}{\sqrt{\deg(v_i)\deg(v_j)}}$$

Non-parametric weights

Aggregate info from neighborhood via the **learned attention**



GAT

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

Parametric weights

Graph Attention Networks, ICLR2018

(Figure credit) https://www.programmersought.com/article/55025669012/

BACKGROUND: MUTUAL INFORMATION (MI)

- Measures the amount of information that two variables share
- If X and Y are independent, then $P_{XY} = P_X P_Y \rightarrow$ in this case, MI = 0

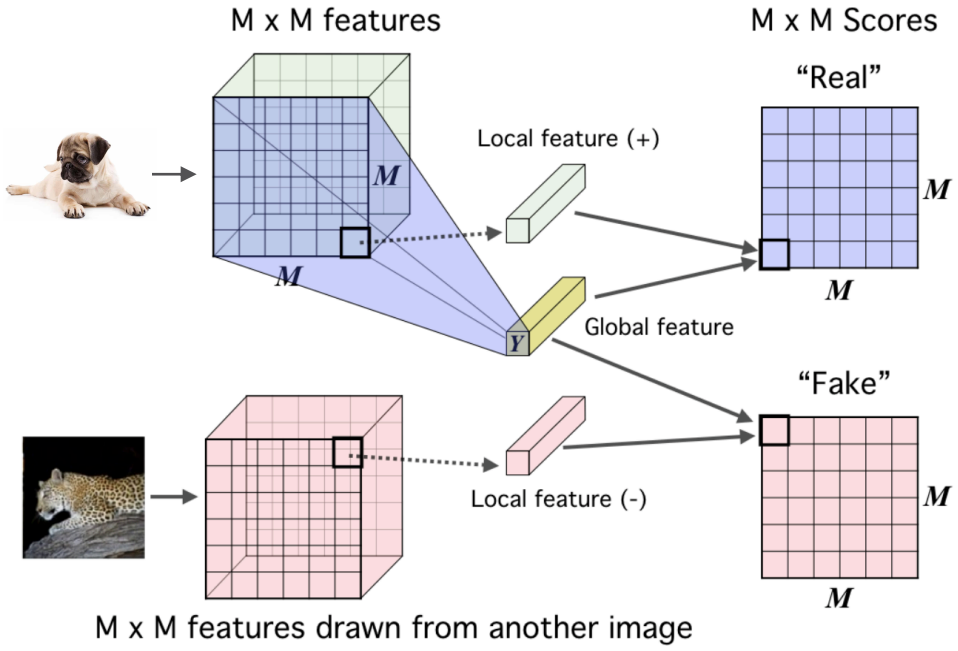
$$I(X;Y) = \mathbb{E}_{P_{XY}}\left[\log \frac{P_{XY}}{P_X P_Y}\right]$$
$$= D_{KL}(P_{XY} || P_X P_Y)$$

- High MI? → One variable is always indicative of the other variable
- Recently, scalable estimation of mutual information was made both possible and practical through **Mutual Information Neural Estimation** (MINE)

MINE: Mutual Information Neural Estimation, ICML2018

BACKGROUND: DEEP INFOMAX

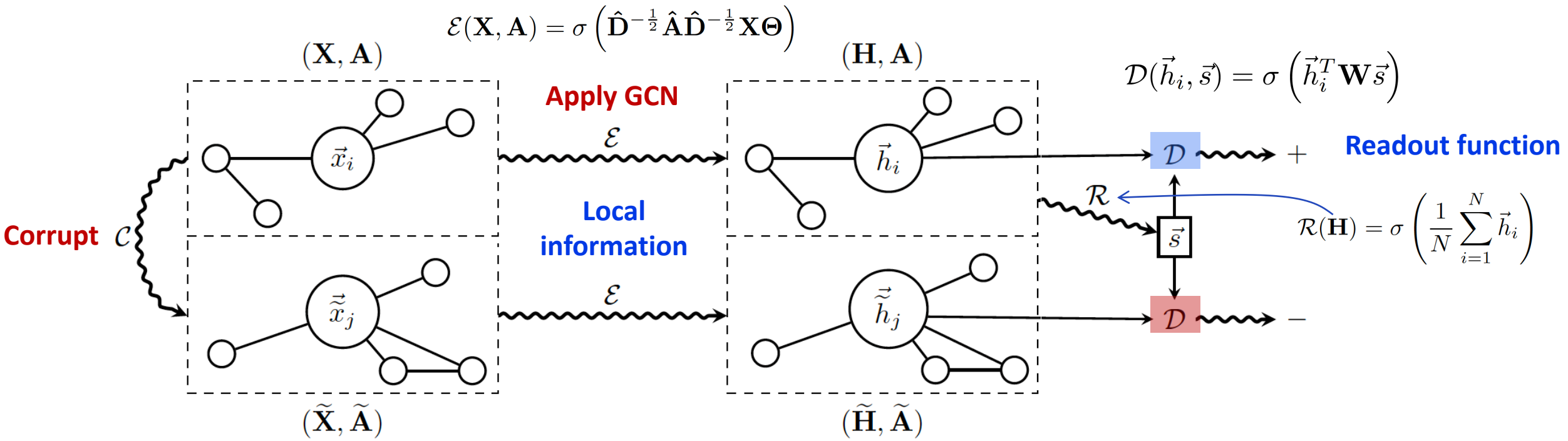
- Unsupervised representation learning method for image data
- Idea: **Maximize mutual information (MI)** between local patches and the global representation of an image



Discriminator tries to discriminate between "Real" and "Fake"

Learning deep representations by mutual information estimation and maximization, ICLR2019

DEEP GRAPH INFOMAX

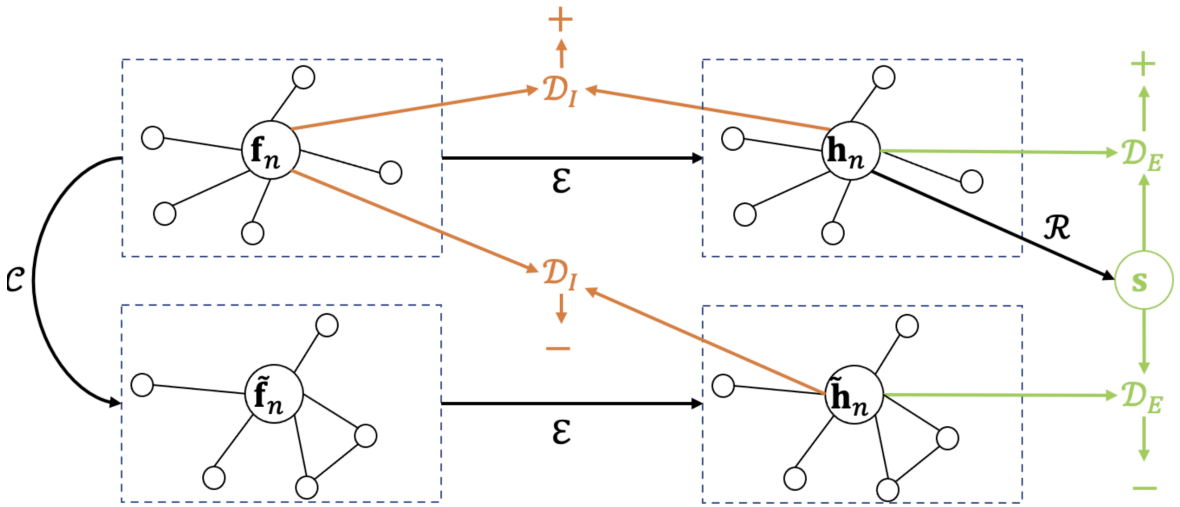


$$\mathcal{L} = \frac{1}{N+M}\left(\sum_{i=1}^{N}\mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\log \mathcal{D}\left(\vec{h}_i, \vec{s}\right)\right] + \sum_{j=1}^{M}\mathbb{E}_{(\widetilde{\mathbf{X}},\widetilde{\mathbf{A}})}\left[\log\left(1 - \mathcal{D}\left(\tilde{\vec{h}}_j, \vec{s}\right)\right)\right]\right)$$

Maximizes the mutual information between the local patches (h_i) and the graph-level global representation (s)

HIGH-ORDER DEEP GRAPH INFOMAX

- **Idea: High-order Mutual Information**
  - We should not only consider the extrinsic supervision signal, i.e., $\boldsymbol{s} \leftrightarrow \boldsymbol{h}$, but also **intrinsic signal**, i.e., $\boldsymbol{f} \leftrightarrow \boldsymbol{h}$



$$I(\mathbf{h}_n; \mathbf{s}; \mathbf{f}_n) = I(\mathbf{h}_n; \mathbf{s}) + I(\mathbf{h}_n; \mathbf{f}_n) - I(\mathbf{h}_n; \mathbf{s}, \mathbf{f}_n)$$

$$\max I(\mathbf{h}_n; \mathbf{s}; \mathbf{f}_n) = \max I(\mathbf{h}_n; \mathbf{s}) + \max I(\mathbf{h}_n; \mathbf{f}_n) - \max I(\mathbf{h}_n; \mathbf{s}, \mathbf{f}_n)$$

Difference-based estimation (Mukherjee et al, 2020)

$$\mathcal{L} = \lambda_E I(\mathbf{h}_n; \mathbf{s}) + \lambda_I I(\mathbf{h}_n; \mathbf{f}_n) + \lambda_J I(\mathbf{h}_n; \mathbf{s}, \mathbf{f}_n)$$

$$I(X; Y) = H(X) + H(Y) - H(X, Y)$$ **DGI**

$$I(X_1; X_2; X_3) = H(X_1) + H(X_2) + H(X_3) - H(X_1, X_2) - H(X_1, X_3) - H(X_2, X_3) + H(X_1, X_2, X_3)$$

$$= H(X_1) + H(X_2) - H(X_1, X_2) + H(X_1) + H(X_3) - H(X_1, X_3) - H(X_1) - H(X_2, X_3) + H(X_1, X_2, X_3)$$

$$= I(X_1; X_2) + I(X_1; X_3) - I(X_1; X_2, X_3)$$

HDMI: High-order Deep Multiplex Infomax, WWW2021

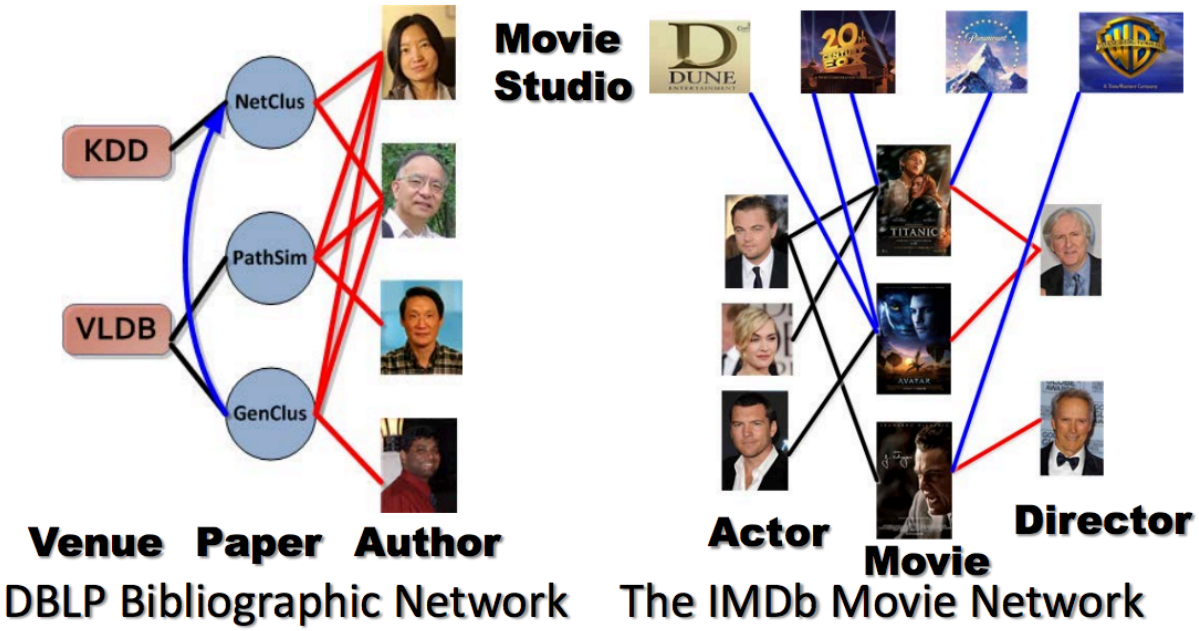
OUTLINE

- Homogeneous Network Embedding
- Multi-aspect Network Embedding
- Attributed Network Embedding
- Heterogeneous Network Embedding
- Training GNN
  - Self-supervised learning
  - Going deeper with GNN
- Applications of Graph Machine Learning

HETEROGENEOUS NETWORK (HETNET)

- So far, we have look at graphs with a single type of node and a single type of edges
- However, in reality a lot of graphs have **multiple types of nodes** and **multiple types of edges**
- Such networks are called **"heterogeneous network"**

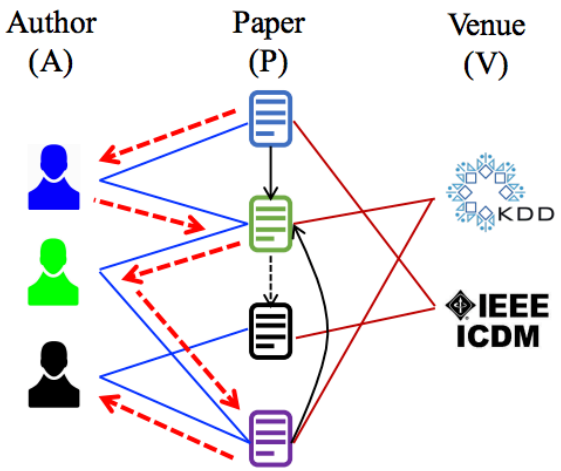
Num. node types > 1
Num. edge types > 1



How do we embed nodes in a heterogeneous network?

METAPATH2VEC: SCALABLE REPRESENTATION LEARNING FOR HETEROGENEOUS NETWORKS

- **Motivation**: Deepwalk assumes that each node has a single type → Extend Deepwalk to HetNet!

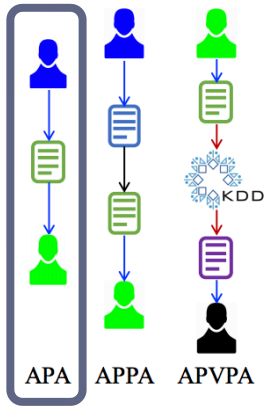


① Meta-path guided random walk (APA)

② Skip-gram model (like Deepwalk)

Maximize: Pr(|) Pr(|)

Meta-paths



Deepwalk

$$\arg\max_{\theta} \sum_{v \in V} \sum_{c \in N(v)} \log p\left(c \mid v; \theta\right)$$

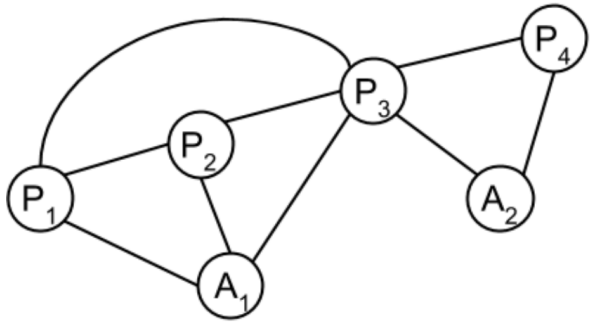
metapath2vec

$$\arg\max_{\theta} \sum_{v \in V} \sum_{t \in T_V} \sum_{c_t \in N_t(v)} \log p(c_t \mid v; \theta)$$

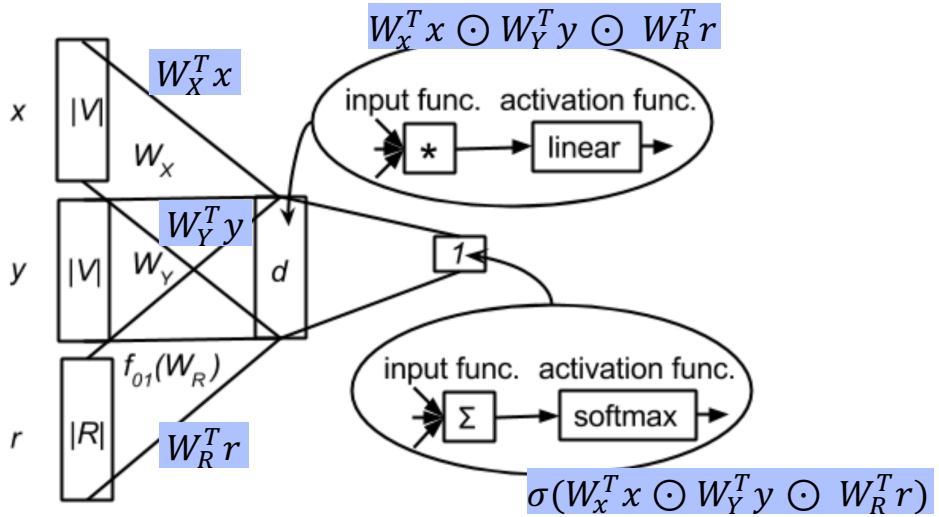
metapath2vec: Scalable Representation Learning for Heterogeneous Networks, KDD2017

HIN2VEC

- **Motivation**: Do we need predefined metapaths?
- **Idea**: Learn latent vectors of both nodes and the targeted relationships



$$R = \{\text{P-P, P-A, A-P, P-P-P, P-P-A, P-A-P, A-P-P, A-P-A}\}$$



Relation exists between x and y

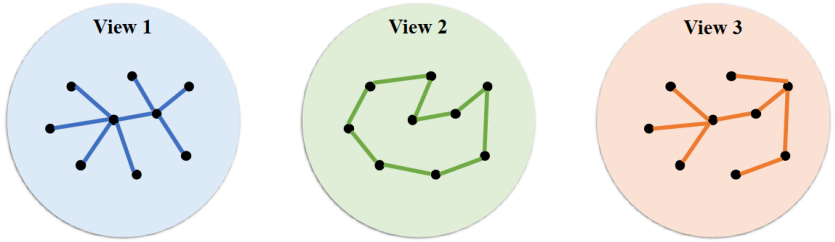
$$O_{x,y,r}(x,y,r) = \begin{cases} P(r|x,y), & \text{if } L(x,y,r) = 1 \\ 1 - P(r|x,y), & \text{if } L(x,y,r) = 0 \end{cases}$$

$$\log O_{x,y,r}(x,y,r) = L(x,y,r) \log P(r|x,y) + [1 - L(x,y,r)] \log[1 - P(r|x,y)]$$

$$P(r|x,y) = sigmoid\left(\sum W'_X \vec{x} \odot W'_Y \vec{y} \odot f_{01}(W'_R \vec{r})\right)$$

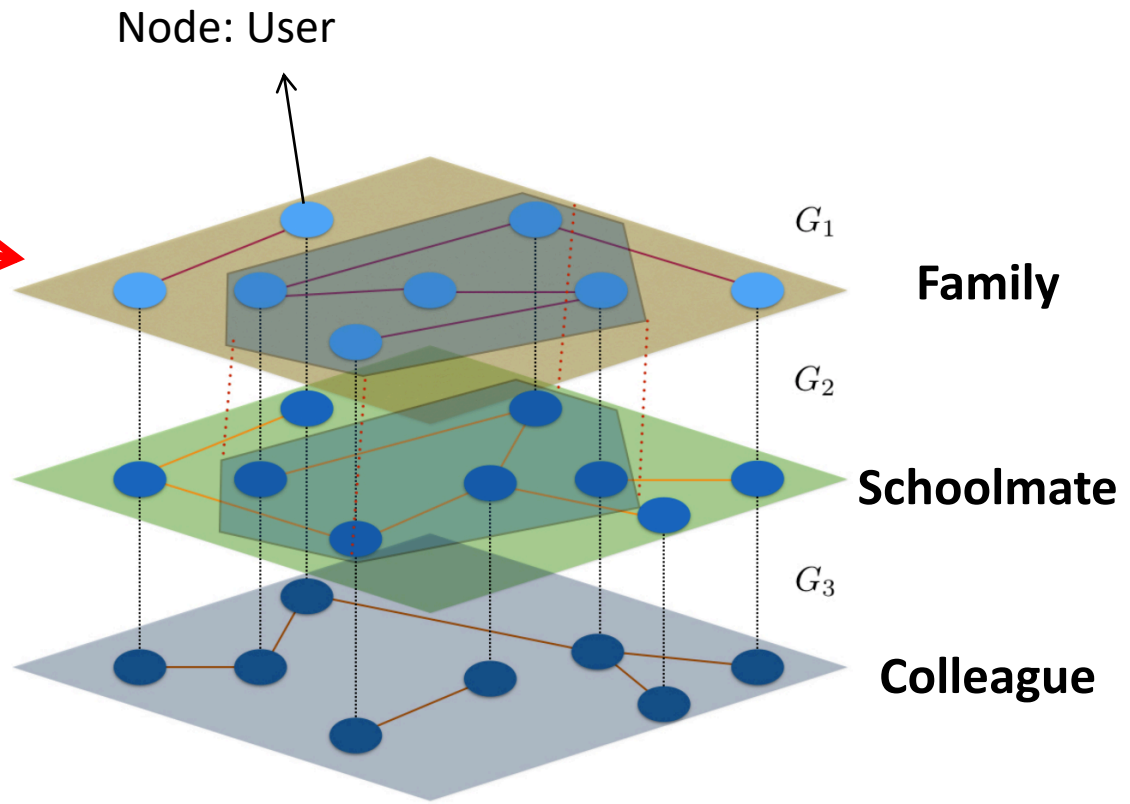
$$O \propto \log O = \sum_{x,y,r \in D} \log O_{x,y,r}(x,y,r)$$

MULTI-LAYER (MULTIPLEX) NETWORK



- A type of heterogeneous network
  - A single node type, multiple edge types
- **Example 1: Social network**
  - Relationship between users
- **Example 2: E-commerce**
  - Relationship between items
- **Example 3: Publication network**
  - Relationship between papers (Citation, share authors)
  - Relationship between authors (Co-author, co-citation)
- **Example 4: Movie database**
  - Relationship between movies
    - Common director, common actor
- **Example 5: Transportation network in a city**
  - Relation between locations in a city
    - Bus, train, car, taxi

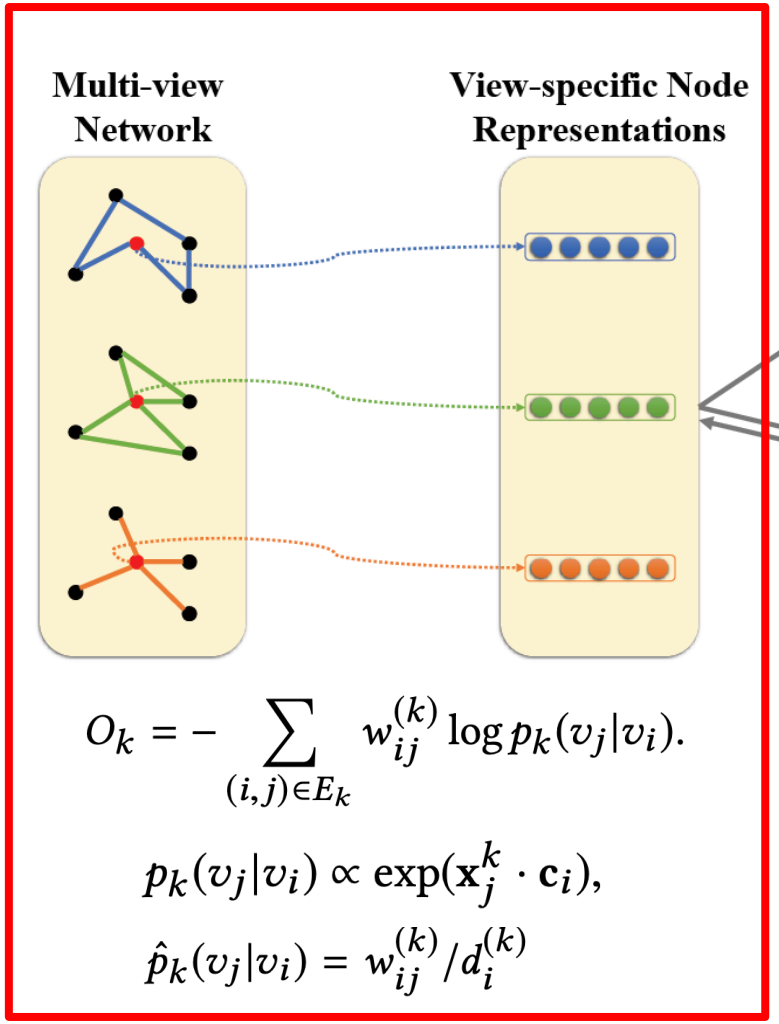
Num. node types $= 1$
Num. edge types > 1



Social Network

AN ATTENTION-BASED COLLABORATION FRAMEWORK FOR MULTI-VIEW NETWORK REPRESENTATION LEARNING

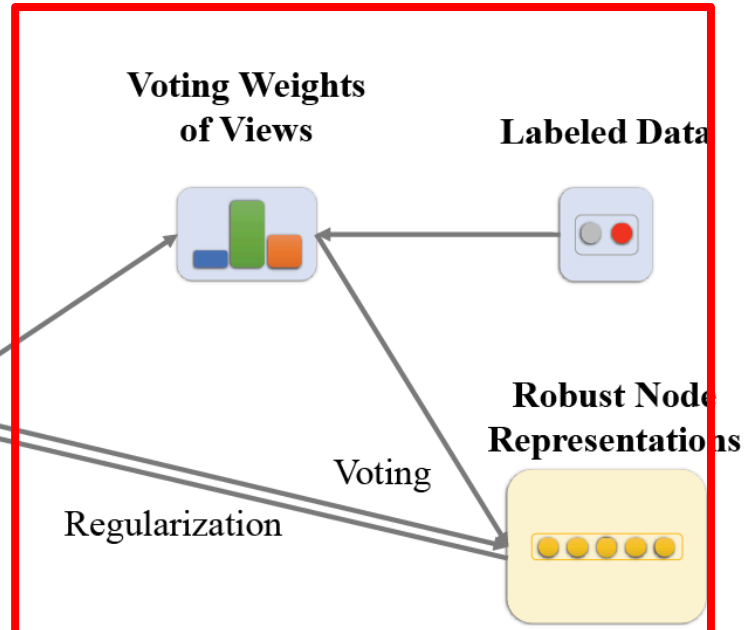
- **Idea**: Promote the collaboration of different views and let them vote for the robust representations.



$$O_k = -\sum_{(i,j)\in E_k} w_{ij}^{(k)} \log p_k(v_j|v_i).$$

$$p_k(v_j|v_i) \propto \exp(\mathbf{x}_j^k \cdot \mathbf{c}_i),$$

$$\hat{p}_k(v_j|v_i) = w_{ij}^{(k)} / d_i^{(k)}$$



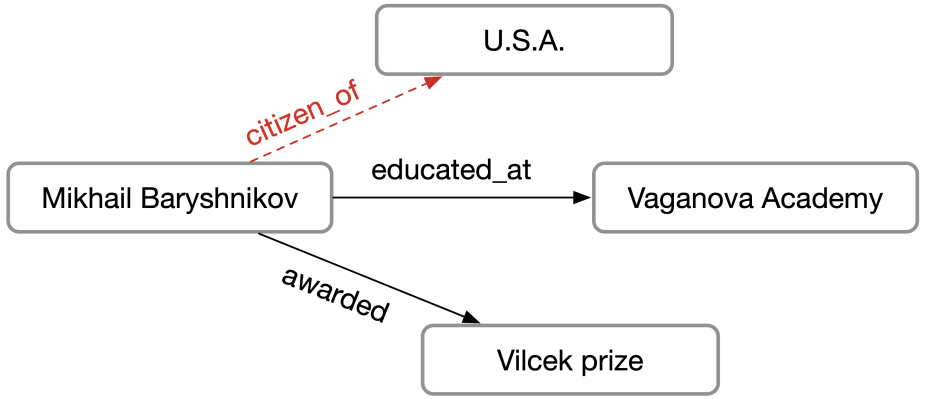
$$R = \sum_{i=1}^{|V|} \sum_{k=1}^{K} \lambda_i^k ||\mathbf{x}_i^k - \mathbf{x}_i||_2^2,$$

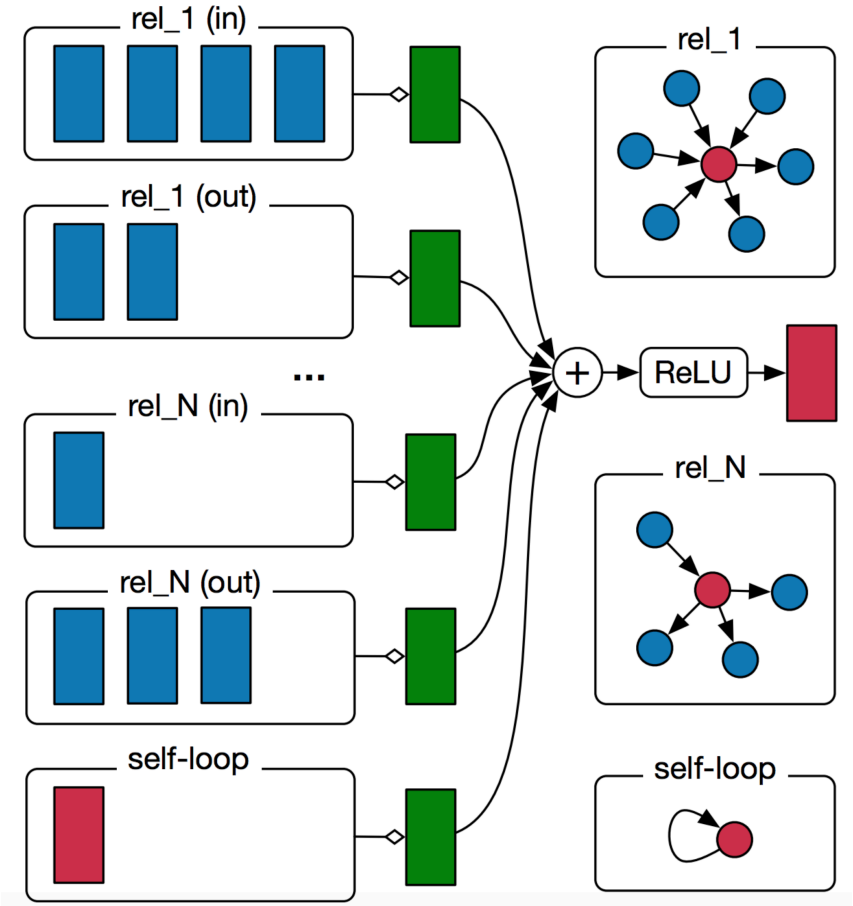
$$\lambda_i^k = \frac{\exp(\mathbf{z}_k \cdot \mathbf{x}_i^C)}{\sum_{k'=1}^{K} \exp(\mathbf{z}_{k'} \cdot \mathbf{x}_i^C)} \qquad \mathbf{x}_i = \sum_{k=1}^{K} \lambda_i^k \mathbf{x}_i^k.$$

$$O_{collab} = \sum_{k=1}^{K} O_k + \eta R,$$

R-GCN: RELATIONAL GCN

- Knowledge graph is a type of multiplex network
  - Nodes are entities, the edges are relations labeled with their types





GCN

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathrm{N_i}} \frac{1}{c_i} \mathrm{W}^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right)$$

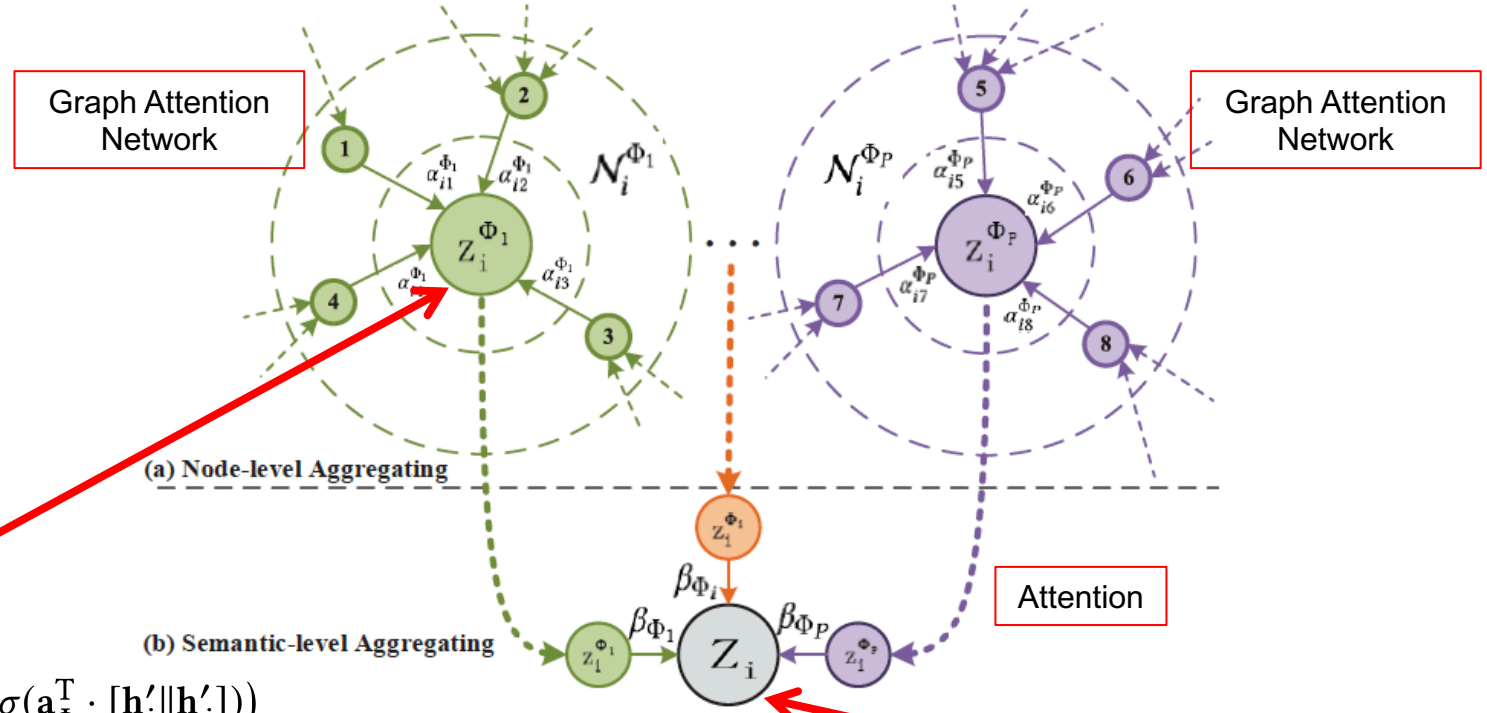
R-GCN

$$h_i^{(l+1)} = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right)$$

Modeling Relational Data with Graph Convolutional Networks, ESWC2018

HAN: HETEROGENEOUS GRAPH ATTENTION NETWORK

- **Idea:** Apply graph attention networks to each network and then aggregate through attention



$$\mathbf{z}_i^{\Phi} = \sigma\left(\sum_{j \in \mathcal{N}_i^{\Phi}} \alpha_{ij}^{\Phi} \cdot \mathbf{h}'_j\right)$$

$$\alpha_{ij}^{\Phi} = softmax_j(e_{ij}^{\Phi}) = \frac{\exp\left(\sigma(\mathbf{a}_{\Phi}^{\mathrm{T}} \cdot [\mathbf{h}'_i \| \mathbf{h}'_j])\right)}{\sum_{k \in \mathcal{N}_i^{\Phi}} \exp\left(\sigma(\mathbf{a}_{\Phi}^{\mathrm{T}} \cdot [\mathbf{h}'_i \| \mathbf{h}'_k])\right)},$$

$$e_{ij}^{\Phi} = att_{node}(\mathbf{h}'_i, \mathbf{h}'_j; \Phi).$$

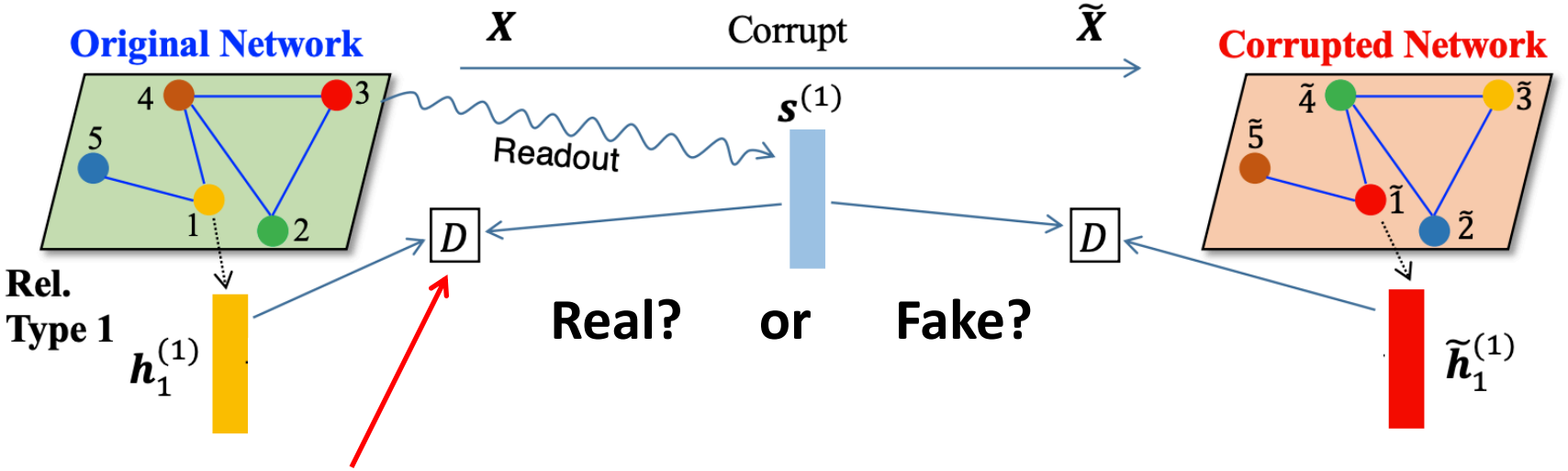
$$\mathbf{Z} = \sum_{p=1}^{P} \beta_{\Phi_p} \cdot \mathbf{Z}_{\Phi_p}$$

Heterogeneous Graph Attention Network, WWW2019

DMGI: UNSUPERVISED ATTRIBUTED MULTIPLEX NETWORK EMBEDDING

- **Idea:** Adopt infomax principal to multiplex network

Schoolmate

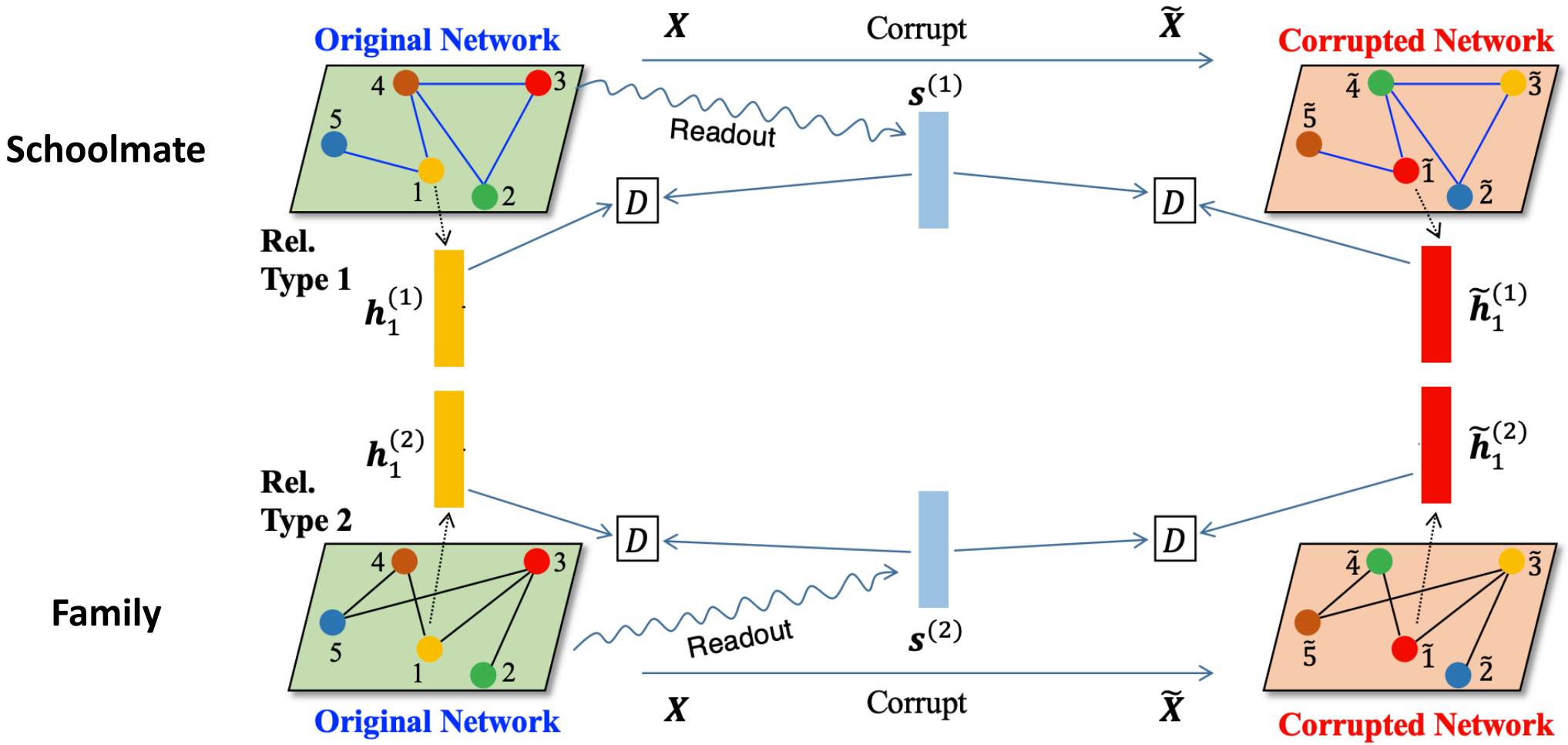


DGI

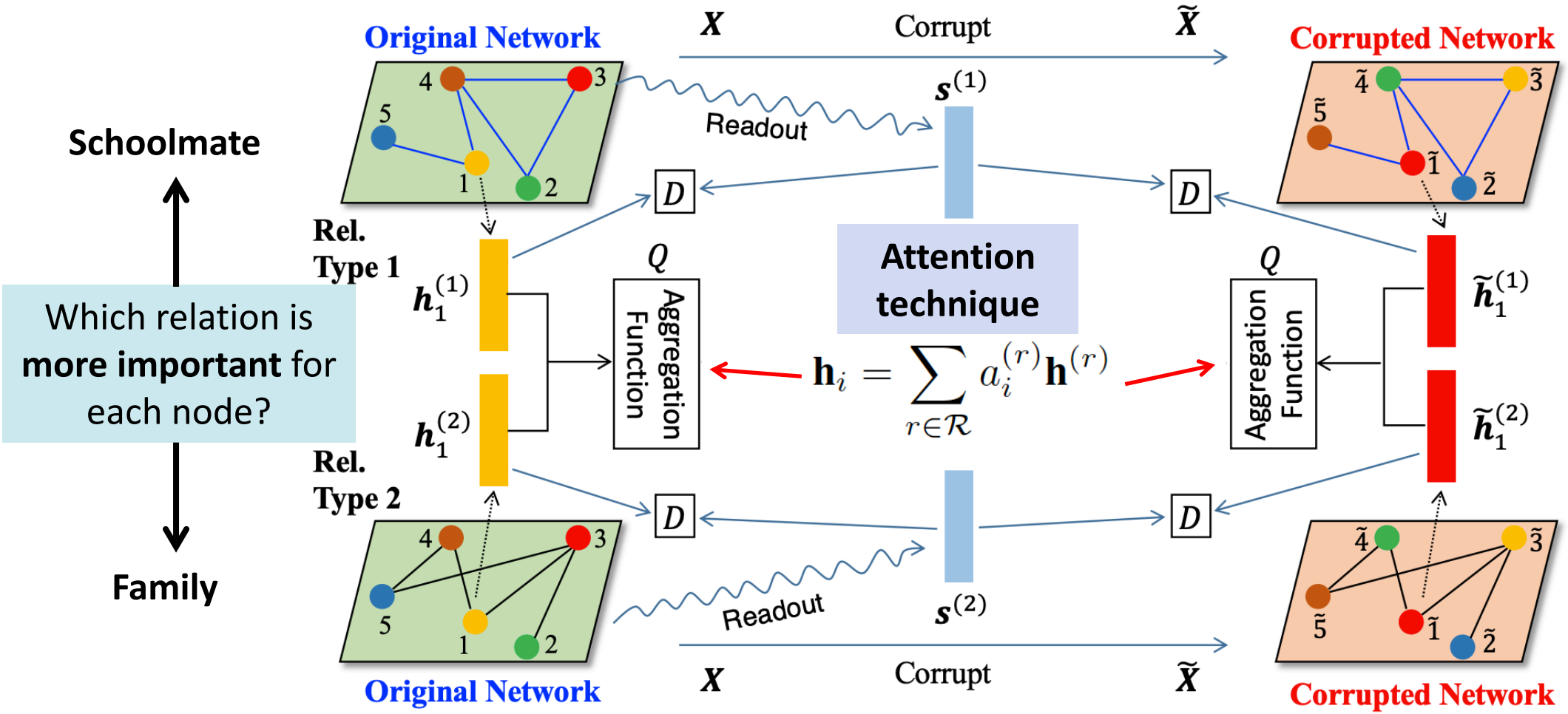
$$\mathcal{D}\left(\mathbf{h}_i^{(r)}, \mathbf{s}^{(r)}\right) = \sigma(\mathbf{h}_i^{(r)T}\mathbf{M}^{(r)}\mathbf{s}^{(r)})$$

Score matrix w.r.t. relation r

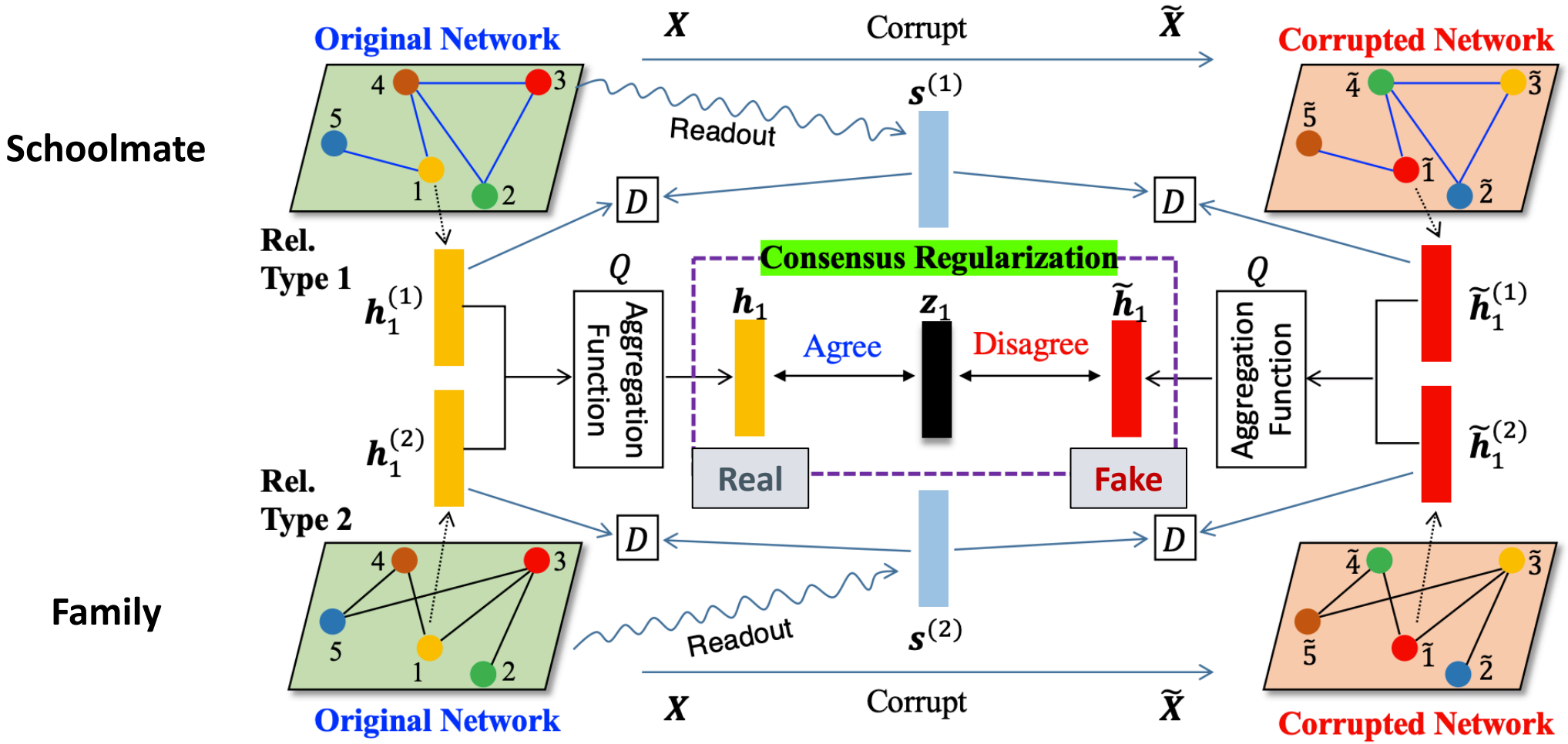
DMGI: UNSUPERVISED ATTRIBUTED MULTIPLEX NETWORK EMBEDDING



DMGI: UNSUPERVISED ATTRIBUTED MULTIPLEX NETWORK EMBEDDING

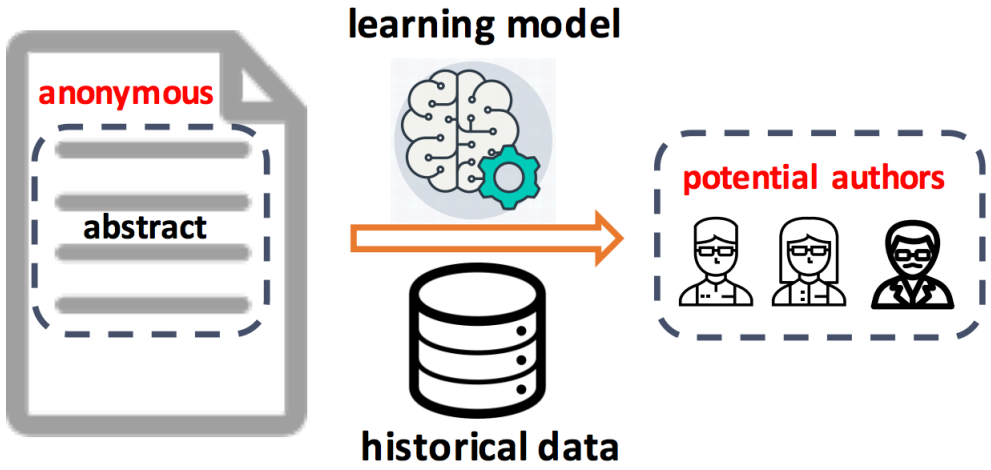


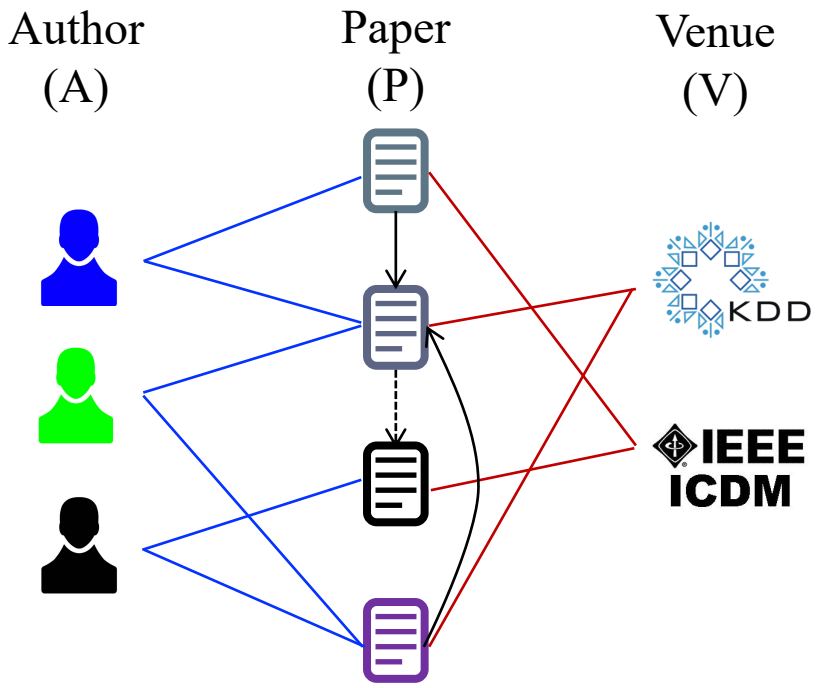
DMGI: UNSUPERVISED ATTRIBUTED MULTIPLEX NETWORK EMBEDDING



TASK-GUIDED METHODS

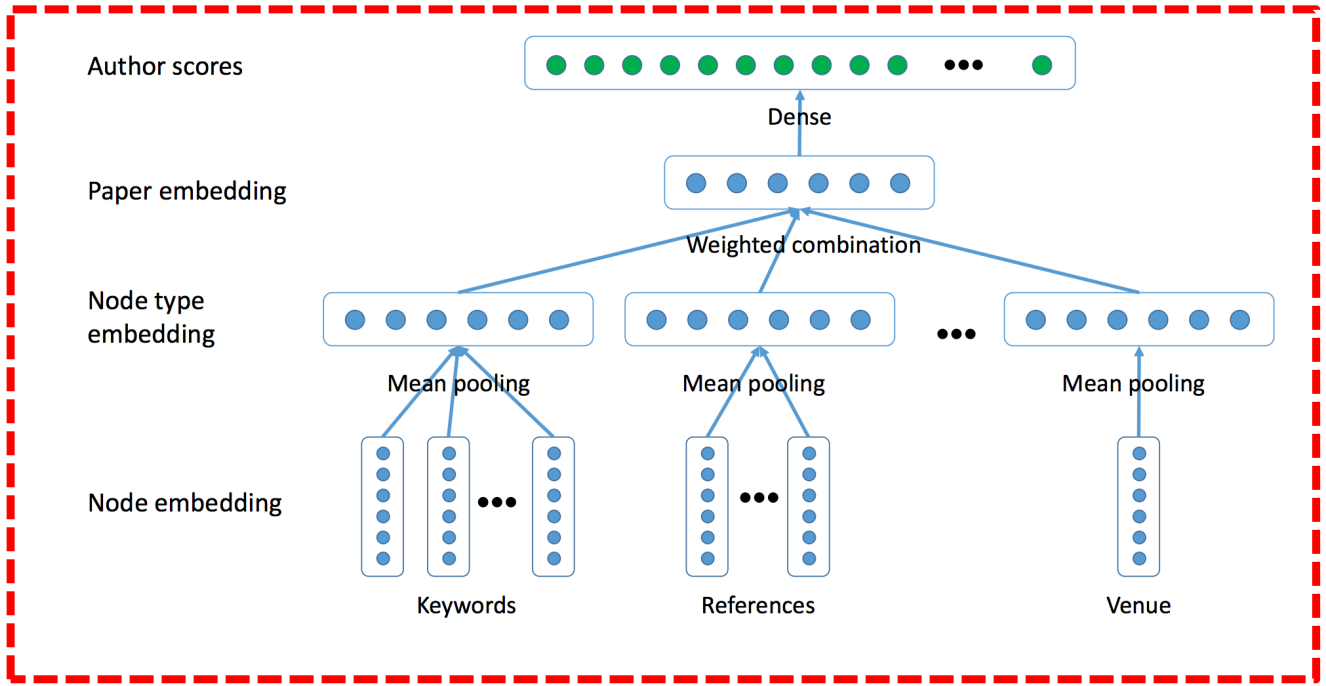
- Instead of learning general node embeddings, what about we focus on a specific task?
- Example: Author Identification
  - Predict the true authors of an anonymized paper given
    - Paper abstract
    - Venue (e.g., KDD, ICDM)
    - References
- Can we predict the true authors?



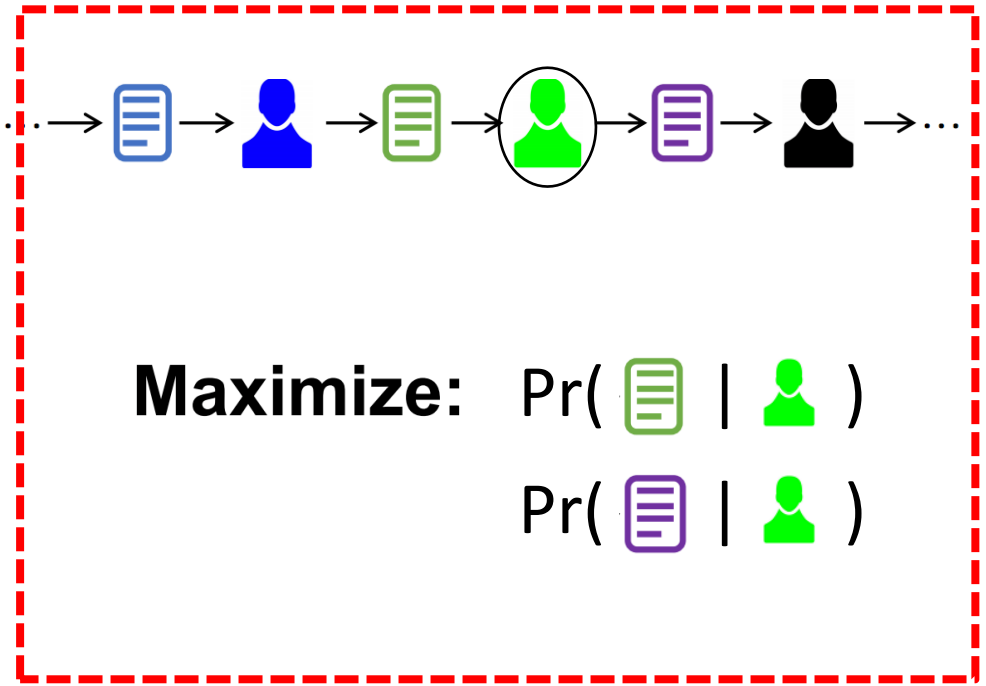


HETNETE

- **Step 1**: Combine keywords, venue and references related to a paper to obtain the paper embedding
- **Step 2**: Perform metapath2vec using embeddings learned in step 1



+

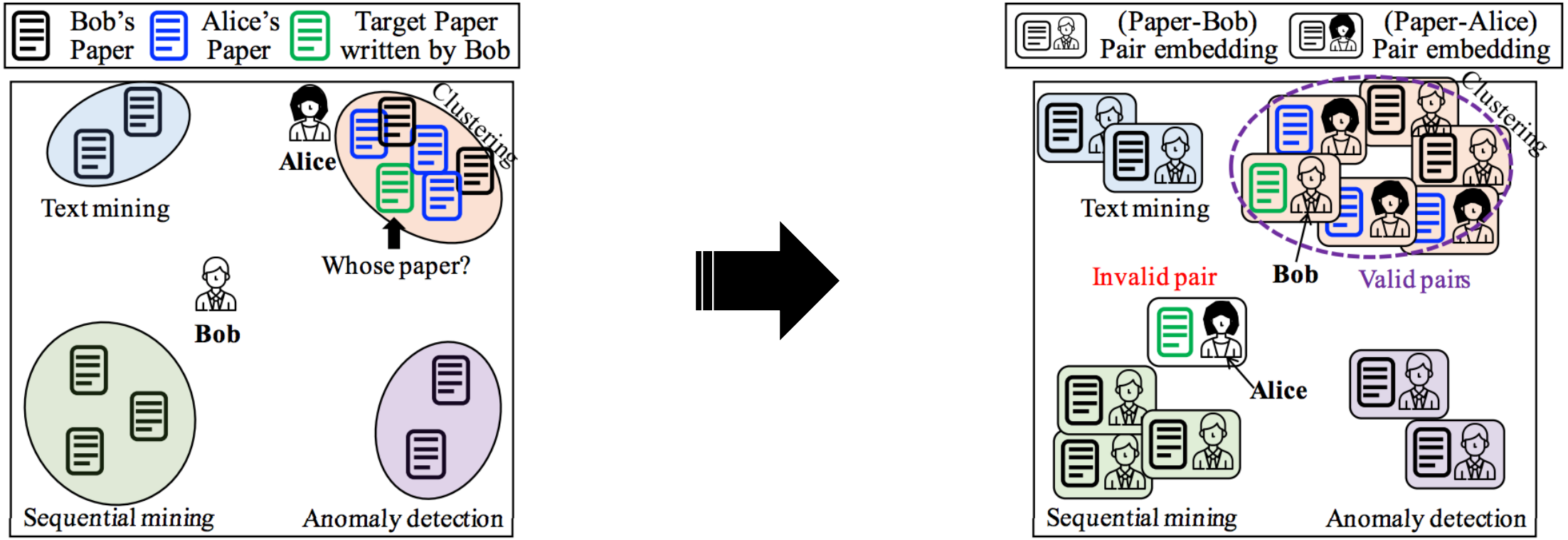


Supervised part:
Task-specific part

Unsupervised part:
metapath2vec

TAPEM

- Idea: Let's generate **pair embeddings**

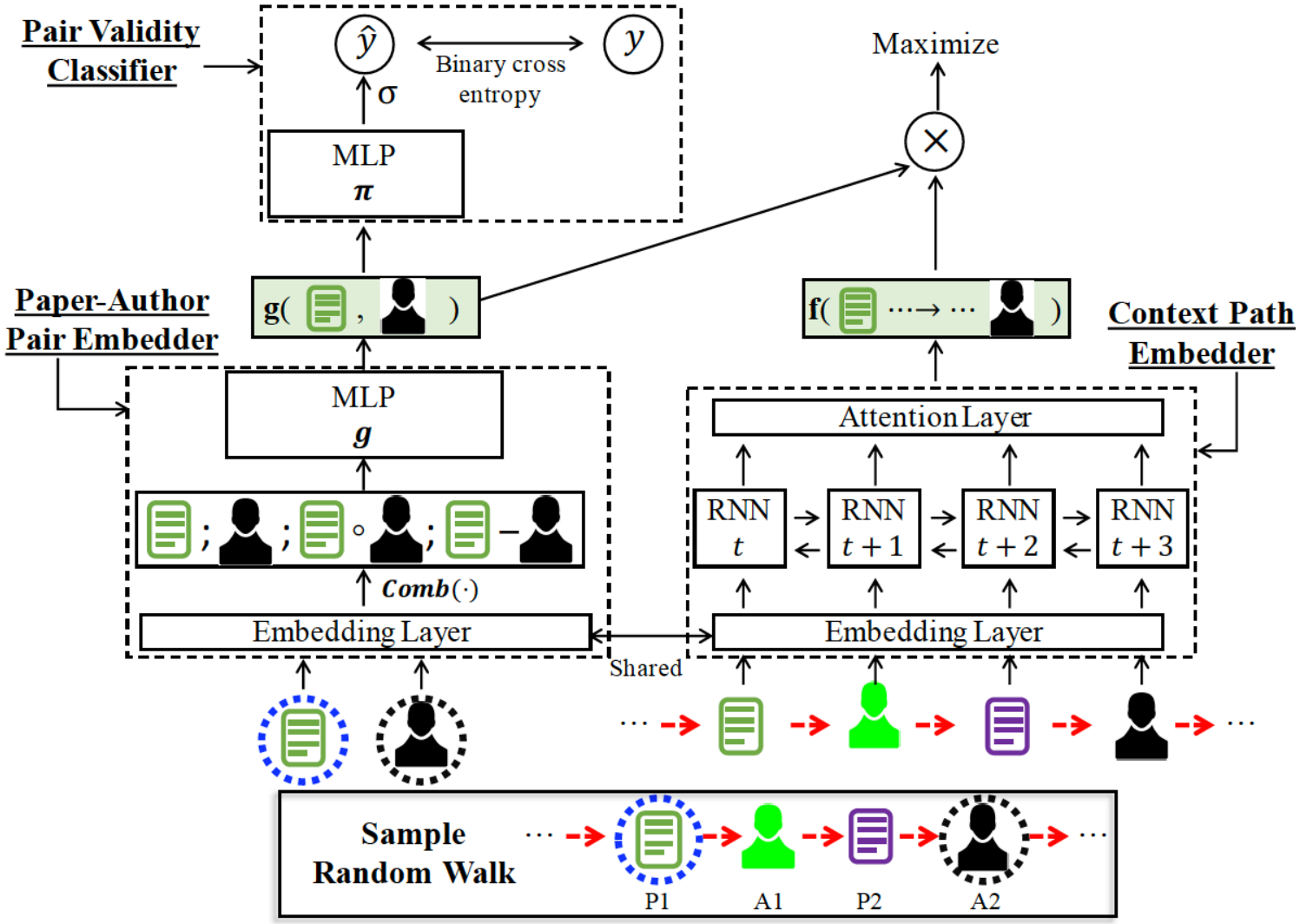


Associated research topic

Pair validity information

- whether is written by

TAPEM


Pair Validity Classifier
Binary cross entropy
MLP
Maximize
Paper-Author Pair Embedder
Context Path Embedder
MLP
Attention Layer
RNN
Comb(·)
Embedding Layer
Shared
Embedding Layer
Sample Random Walk
P1
A1
P2
A2

OUTLINE

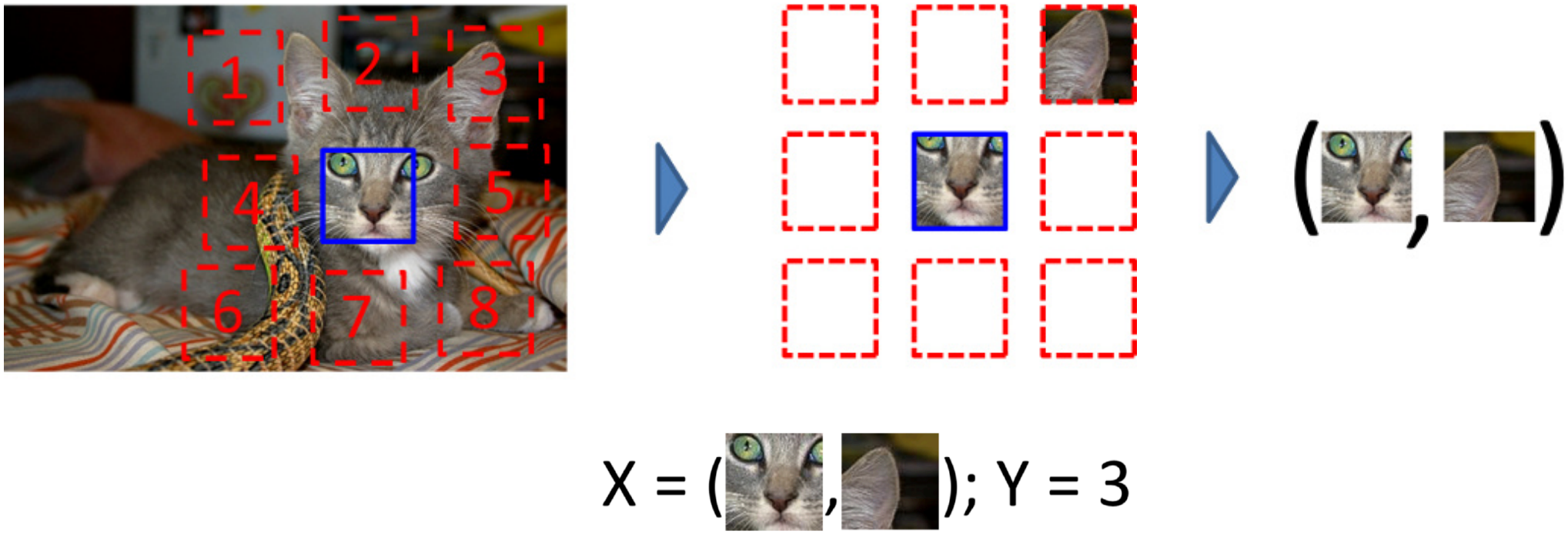
- Homogeneous Network Embedding
- Multi-aspect Network Embedding
- Attributed Network Embedding
- Heterogeneous Network Embedding
- Training GNN
  - Self-supervised learning
  - Going deeper with GNN
- Applications of Graph Machine Learning

OUTLINE

- Homogeneous Network Embedding
- Multi-aspect Network Embedding
- Attributed Network Embedding
- Heterogeneous Network Embedding
- Training GNN
  - Self-supervised learning
  - Going deeper with GNN
- Applications of Graph Machine Learning

WHAT IS SELF-SUPERVISED LEARNING?

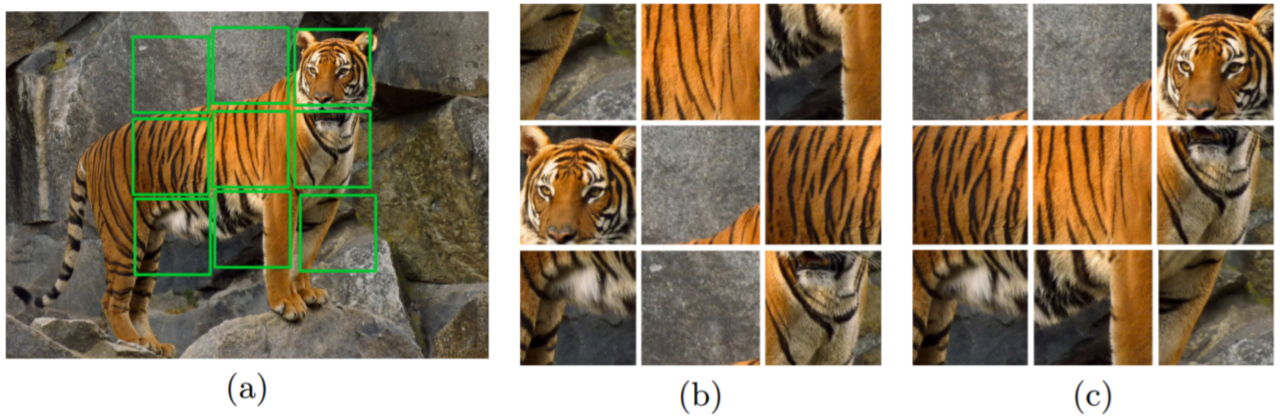
- A form of unsupervised learning where the data provides the supervision
- In general, withhold some part of the data, and task the network with predicting it
- An example of **pretext task**: **Relative positioning**
  - Train network to predict relative position of two regions in the same image



(Figure credit) Unsupervised visual representation learning by context prediction, ICCV2015

WHAT IS SELF-SUPERVISED LEARNING?

- Pretext task: **Jigsaw puzzle**

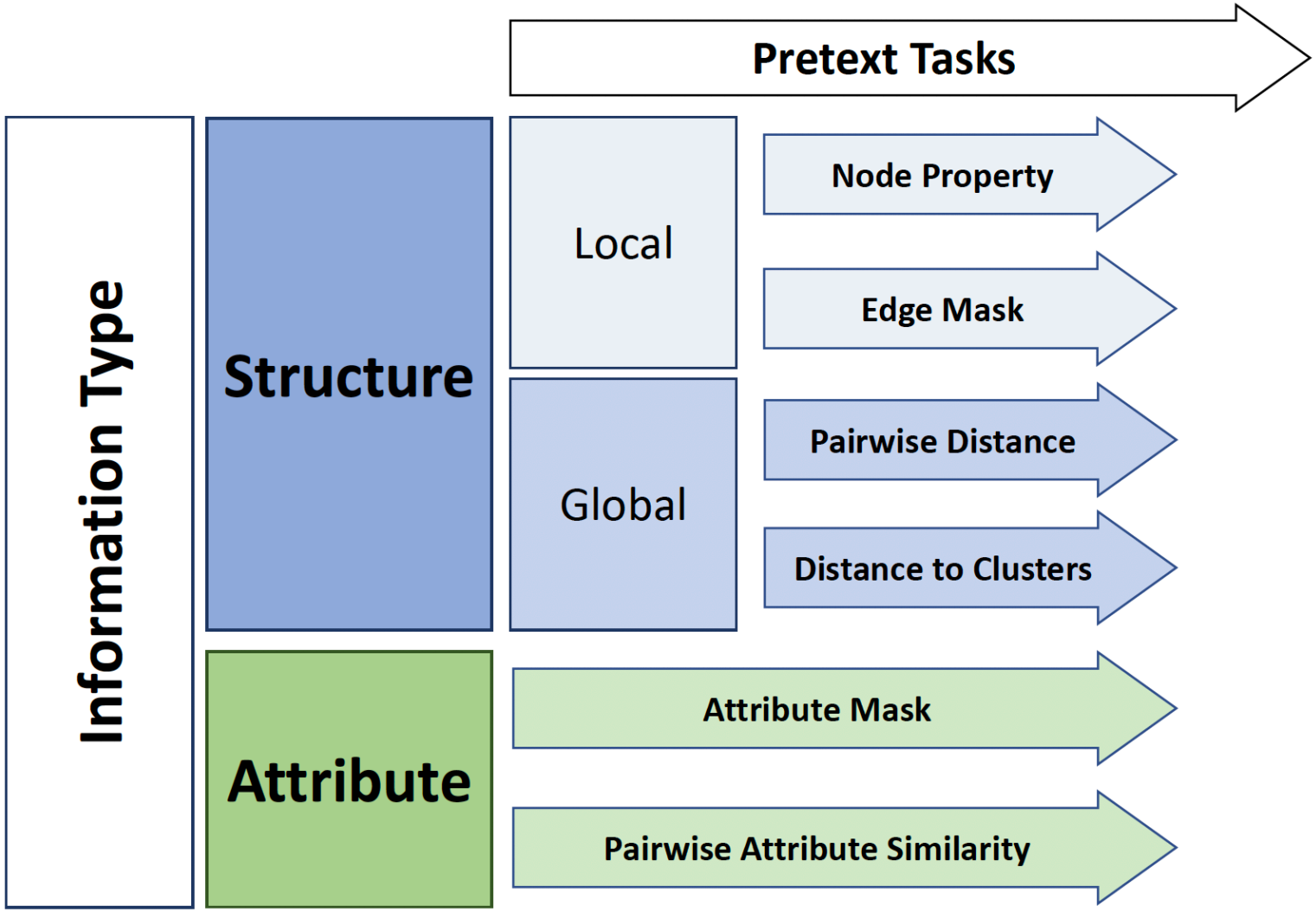


- Pretext task : **Colorization**

- Pretext task : **Rotation**
  - Which one has the correct rotation?

(Figure credit) Self-Supervised Learning, Andrew Zisserman

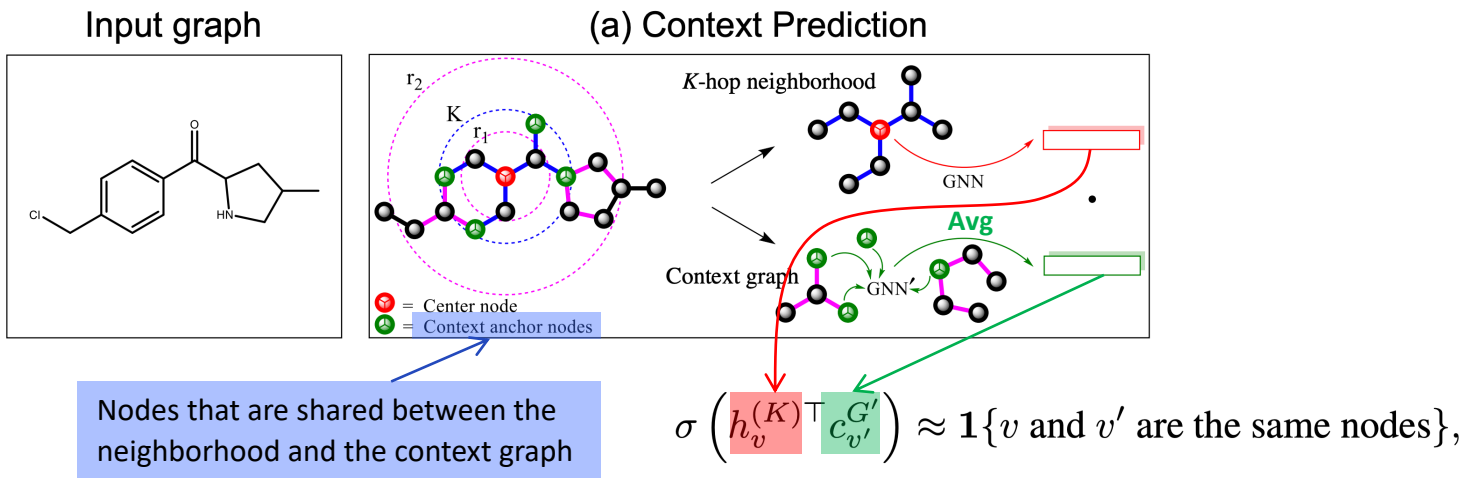
EXAMPLES OF PRETEXT TASKS ON GRAPHS



(Figure credit) Tutorial in Graph Neural Networks: Models and Applications, AAAI2021

CONTEXT PREDICTION

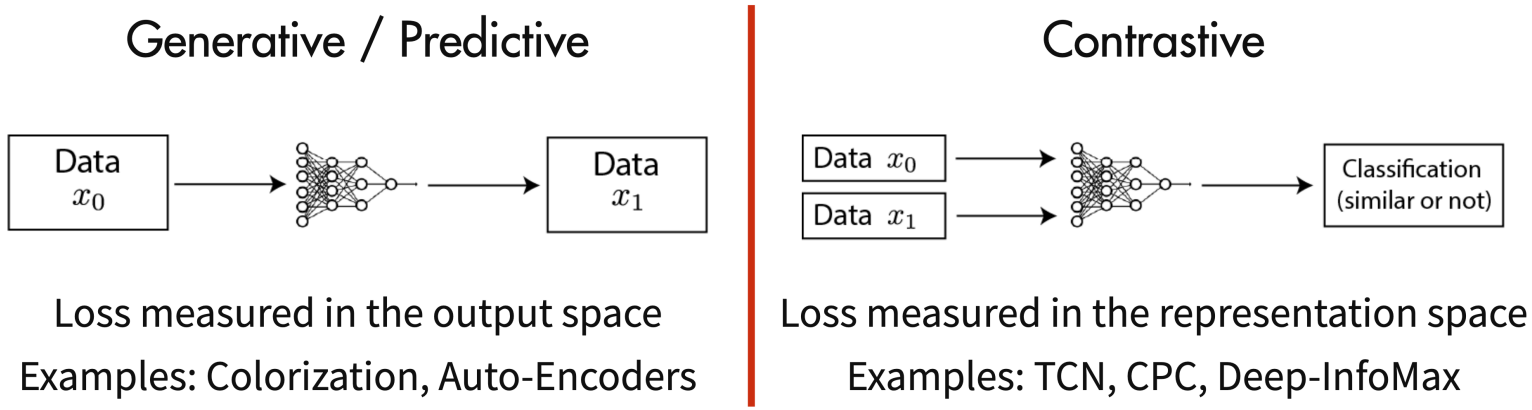
- **Pretext task:** Context prediction



	Chemistry			Biology		
	Non-pre-trained	Pre-trained	Gain	Non-pre-trained	Pre-trained	Gain
GIN	67.0	**74.2**	**+7.2**	64.8 ± 1.0	**74.2 ± 1.5**	**+9.4**
GCN	**68.9**	72.2	+3.4	63.2 ± 1.0	70.9 ± 1.7	+7.7
GraphSAGE	68.3	70.3	+2.0	65.7 ± 1.2	68.5 ± 1.5	+2.8
GAT	66.8	60.3	-6.5	**68.2 ± 1.1**	67.8 ± 3.6	-0.4

Strategies for Pre-training Graph Neural Networks, ICLR2020

CONTRASTIVE LEARNING



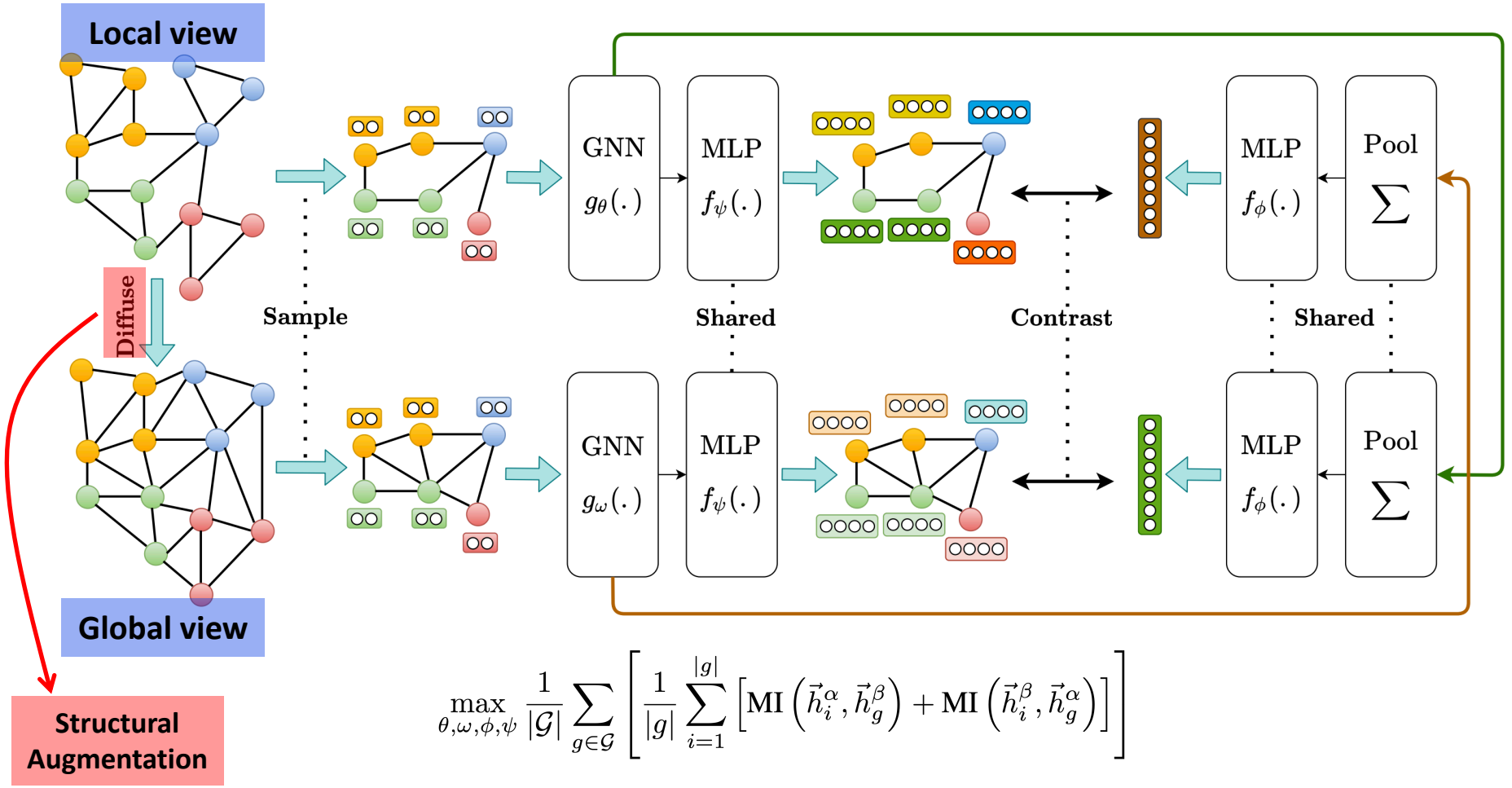
- **Given:** $X = \{x, \quad x^+, \quad x_1^-, \ldots, x_{N-1}^-\}$; Similarity function $s(\cdot)$ (e.g., cosine similarity)
- **Goal:** $s\big(f(x), f(x^+)\big) > s(f(x), f(x^-))$
- **Contrastive/InfoNCE Loss**

$$\mathcal{L}_N = -\mathbb{E}_X \left[\log \frac{\exp\left(s\big(f(x), f(x^+)\big)\right)}{\exp\left(s\big(f(x), f(x^+)\big)\right) + \sum_{j=1}^{N-1} \exp\left(s\left(f(x), f\left(x_j^-\right)\right)\right)} \right]$$

(Figure credit) https://ankeshanand.com/blog/2020/01/26/contrative-self-supervised-learning.html

CONTRASTIVE MULTI-VIEW REPRESENTATION LEARNING ON GRAPHS

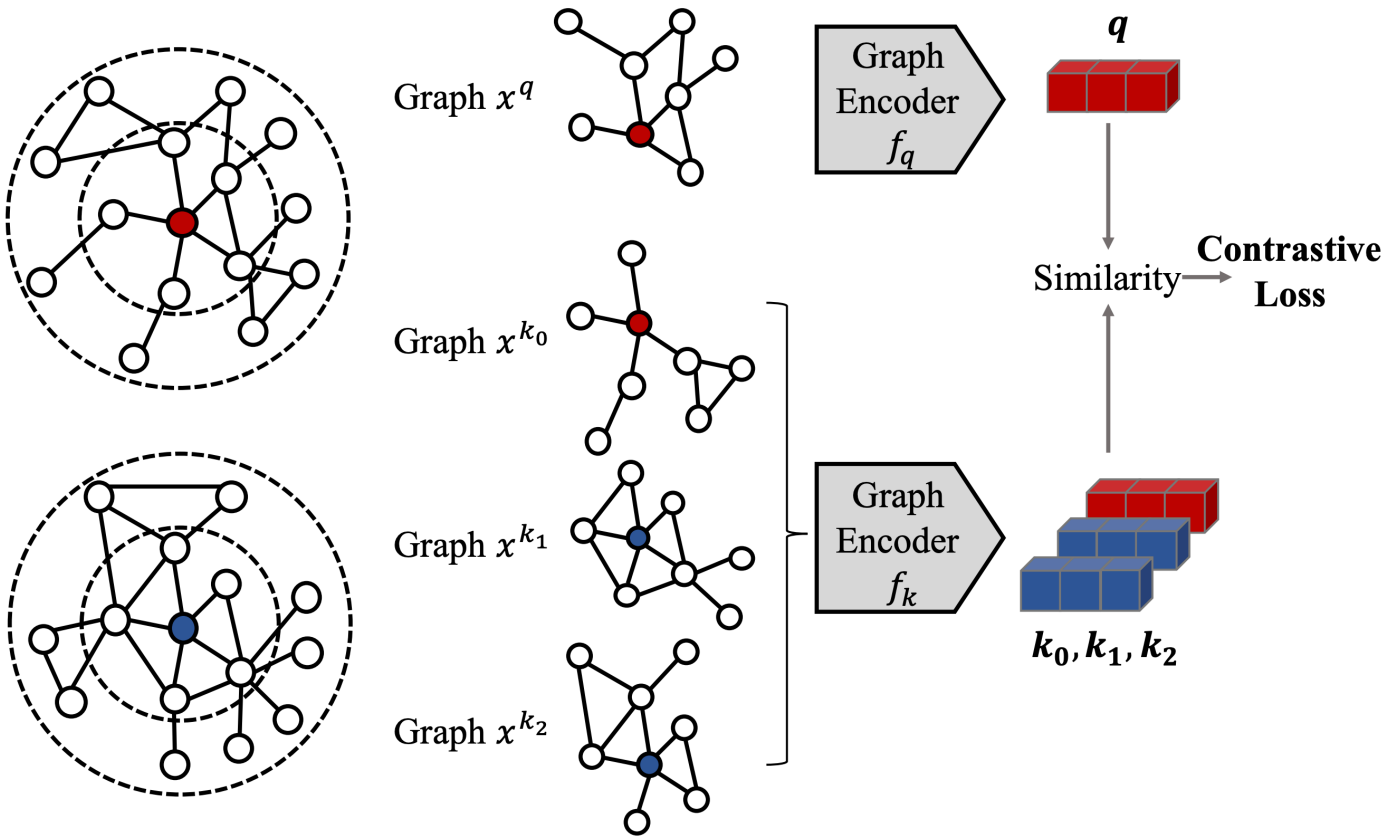
- **Idea**: Contrast encodings from first-order neighbors and a general graph diffusion
  - Maximize MI between node representations of one view and graph representation of another view and vice versa



$$\max_{\theta,\omega,\phi,\psi} \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} \left[\frac{1}{|g|} \sum_{i=1}^{|g|} \left[\mathrm{MI}\left(\vec{h}_i^{\,\alpha}, \vec{h}_g^{\,\beta}\right) + \mathrm{MI}\left(\vec{h}_i^{\,\beta}, \vec{h}_g^{\,\alpha}\right) \right] \right]$$

GCC: GRAPH CONTRASTIVE CODING FOR GRAPH NEURAL NETWORK PRE-TRAINING

- **Idea**: **Subgraph instance discrimination** in and across networks



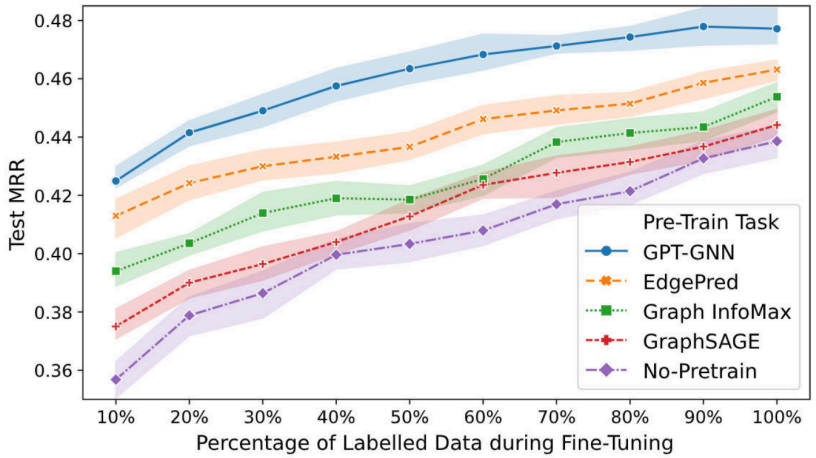
$$\mathcal{L} = -\log \frac{\exp\left(\boldsymbol{q}^\top \boldsymbol{k}_+ / \tau\right)}{\sum_{i=0}^{K} \exp\left(\boldsymbol{q}^\top \boldsymbol{k}_i / \tau\right)}$$

- Query instance x^q
- Key instances $\{x^{k_0}, x^{k_1}, x^{k_2}\}$
- Embedding
  - $\boldsymbol{q}$ (embedding of x^q)
    - i.e., $\boldsymbol{q} = f(x^q)$
  - $\boldsymbol{k_0}, \boldsymbol{k_1}, \boldsymbol{k_2}$ (embedding of $\{x^{k_0}, x^{k_1}, x^{k_2}\}$)
    - i.e., $\boldsymbol{k_i} = f(x^{k_i})$

GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training, KDD2020

GPT-GNN

- **Idea**: Model the graph distribution $p(G;\theta)$ by learning to reconstruct the input graph
  - Factorize the graph likelihood: $\boldsymbol{p(G;\theta) = p(X,E;\theta) = p(X;\theta)p(E;\theta)}$
    - Attribute generation $p(X;\theta)$ / Edge generation $p(E;\theta)$
  - Autoregressive generative process of an attributed graph



$$\log p_\theta(X,E) = \sum_{i=1}^{|\mathcal{V}|} \log p_\theta(X_i, E_i \mid X_{<i}, E_{<i})$$

$$= \sum_{i=1}^{|\mathcal{V}|} \log p_\theta(X_i \mid X_{<i}, E_{<i}) \cdot p_\theta(E_i \mid X_{<i}, E_{<i})$$

This ignores the dependency between attributes (X) and edges (E)

$$= \sum_{i=1}^{|\mathcal{V}|} \sum\nolimits_o p_\theta(X_i, E_{i,\neg o} \mid E_{i,o}, X_{<i}, E_{<i}) \cdot p_\theta(E_{i,o} \mid X_{<i}, E_{<i})$$

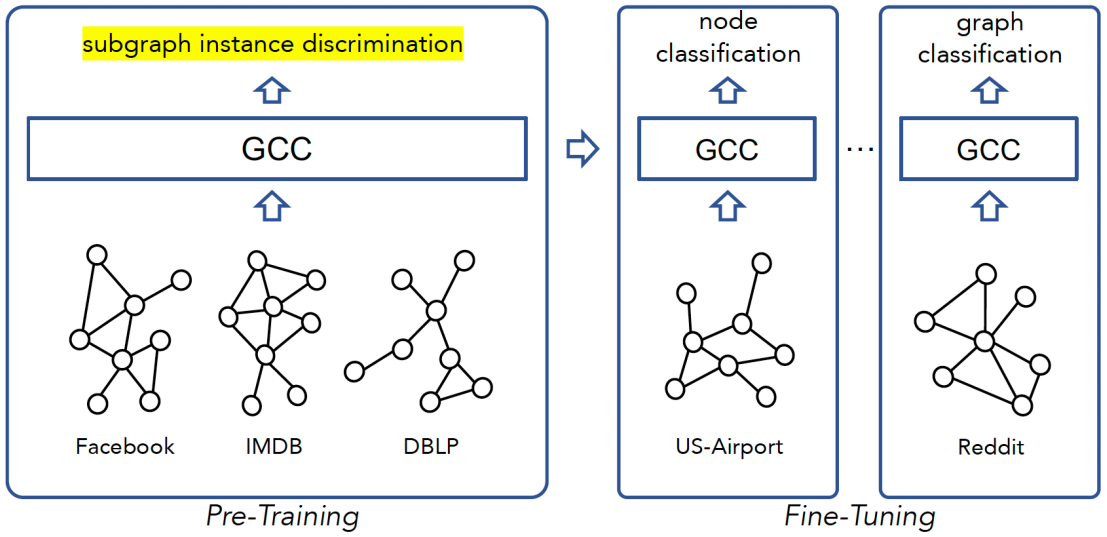
1) Given the observed edges, generate node attributes
2) Given the observed edges and generated node attributes, generate the remaining edges

$$= \sum_{i=1}^{|\mathcal{V}|} \mathbb{E}_o \Big[\underbrace{p_\theta(X_i \mid E_{i,o}, X_{<i}, E_{<i})}_{\text{1) generate attributes}} \cdot \underbrace{p_\theta(E_{i,\neg o} \mid E_{i,o}, X_{\leq i}, E_{<i})}_{\text{2) generate edges}} \Big]$$

Now we can consider the dependency between edges and attributes

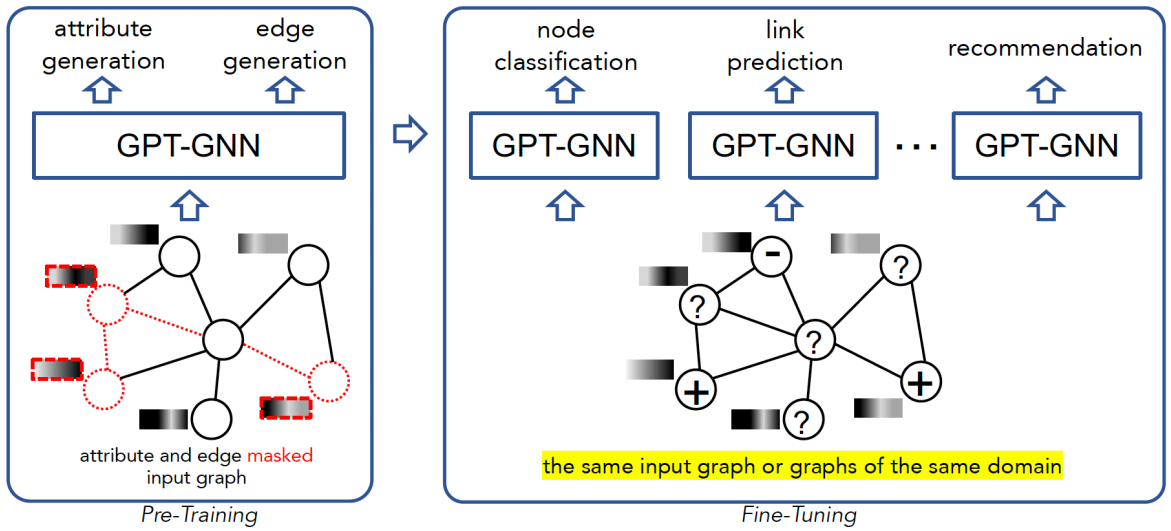
GCC VS. GPT-GNN : TWO DIFFERENT SETTINGS

GCC



- To pre-train from **some graphs**
- To fine-tune for **unseen tasks on unseen graphs**

GPT-GNN



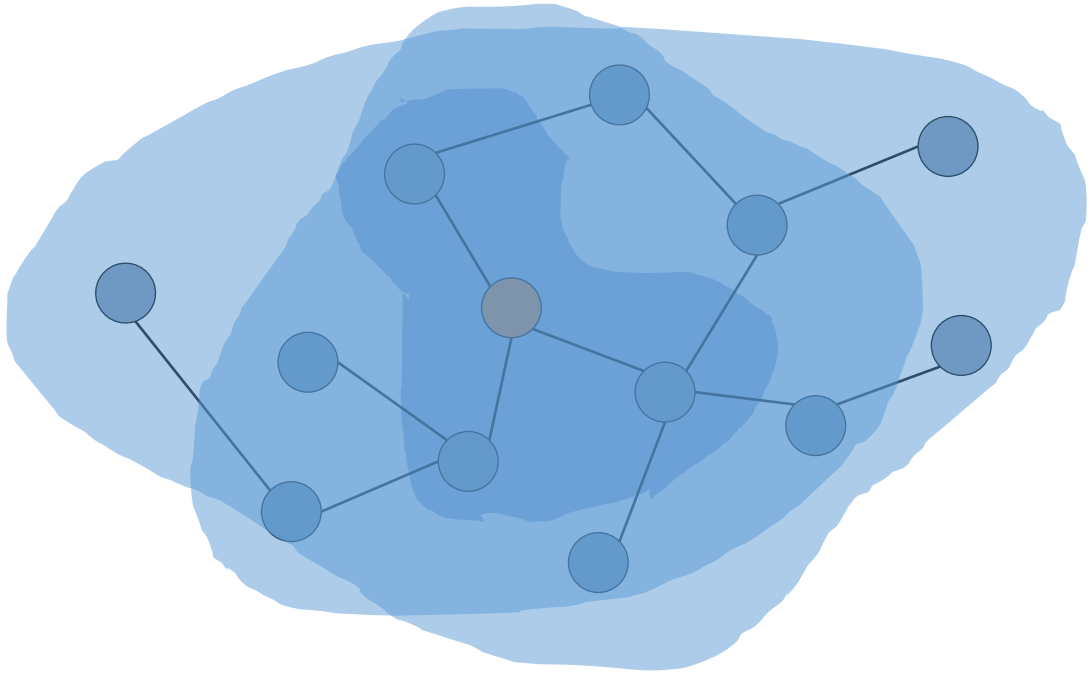
- To pre-train from **one graph**
- To fine-tune for **unseen tasks on the same graph or graphs of the same domain**

OUTLINE

- Homogeneous Network Embedding
- Multi-aspect Network Embedding
- Attributed Network Embedding
- Heterogeneous Network Embedding
- Training GNN
  - Self-supervised learning
  - Going deeper with GNN
- Applications of Graph Machine Learning

WHY DEEP GNNS?

- Lager receptive field (more global view)

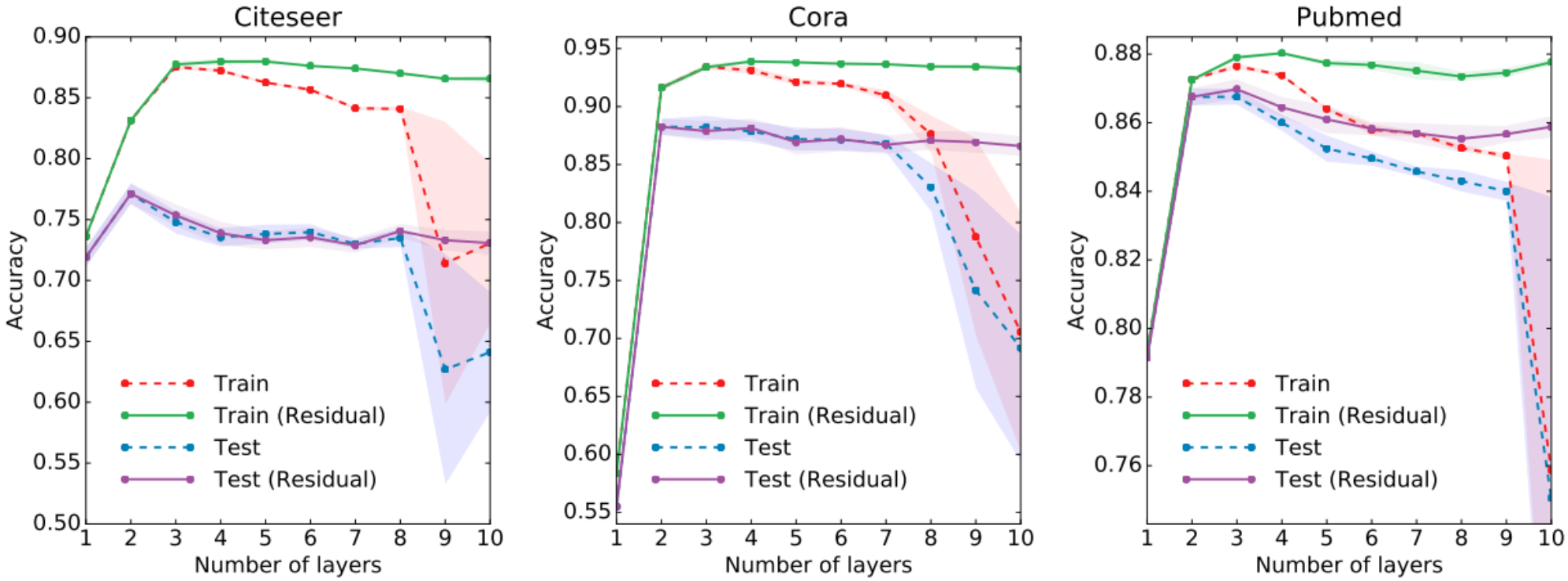


- Larger model capacity

(Figure credit) Tutorial in Graph Neural Networks: Models and Applications, AAAI2021

WHAT HAPPENS WHEN GCN GOES DEEP?

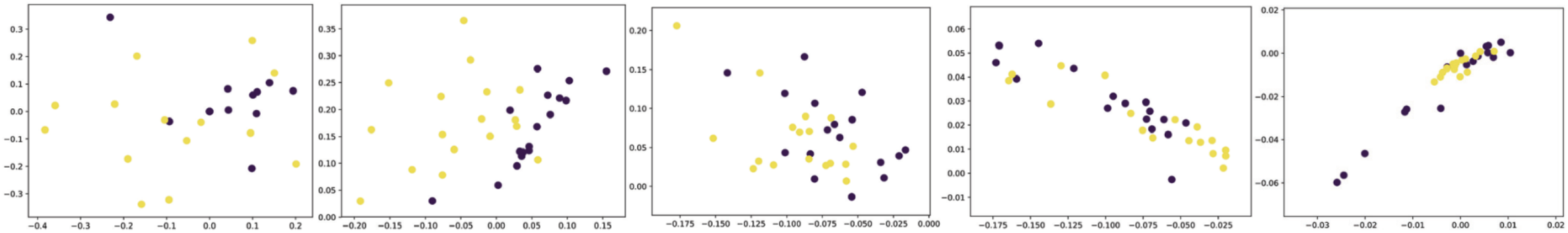
- The performance drops as we go deeper



Semi-Supervised Classification with Graph Convolutional Networks, ICLR2017

WHAT IS GOING ON?

- **Challenge**: Over-smoothing
- Repeated graph convolutions eventually make node embeddings indistinguishable



(a) 1-layer (b) 2-layer (c) 3-layer (d) 4-layer (e) 5-layer

Figure 2: Vertex embeddings of Zachary's karate club network with GCNs with 1,2,3,4,5 layers.

Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning, AAAI2018

WHY OVER-SMOOTHING?

- **Multiple propagation makes representation inseparable!**

Theorem. In a connected graph, given a propagation matrix P and node features $x \in \mathbb{R}^F$, we have

$$\lim_{k\to\infty} P^k x \propto u_1,$$

where u_1 is the eigenvector of P corresponds to its largest eigenvalue.

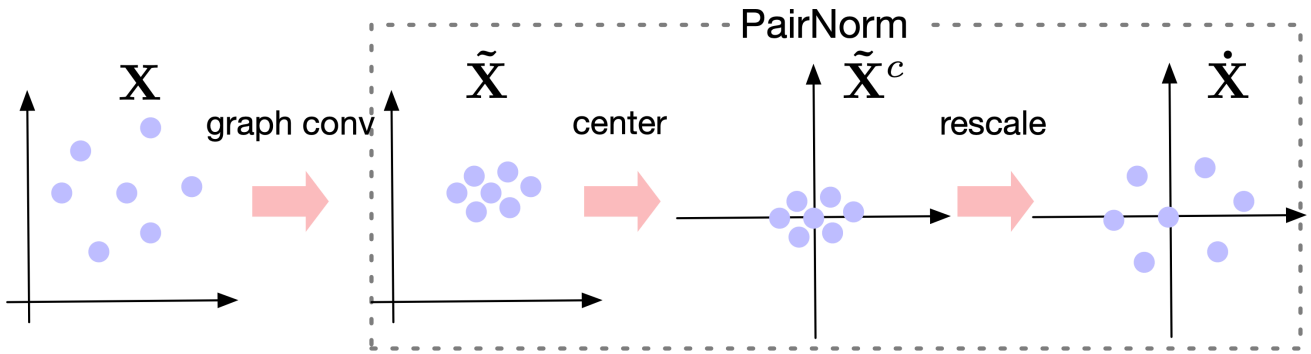
(1) When $P = D^{-1}A$, u_1 is $\mathbf{1}$

(2) When $P = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, u_1 is $D^{-\frac{1}{2}}\mathbf{1}$

Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning, AAAI2018

(Figure credit) Tutorial in Graph Neural Networks: Models and Applications, AAAI2021

PAIRNORM



- **Idea**: Normalization layer for GNNs
  - The total pairwise squared distances (TPSD) remains a constant across layers

$$
\begin{aligned}
\mathrm{TPSD}(\tilde{\mathbf{X}}) &= \sum_{i,j\in[n]} \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2 = \sum_{i,j\in[n]} (\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j)^T(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j) \\
&= \sum_{i,j\in[n]} (\tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_i + \tilde{\mathbf{x}}_j^T\tilde{\mathbf{x}}_j - 2\tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_j) \\
&= 2n\sum_{i\in[n]} \tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_i - 2\sum_{i,j\in[n]} \tilde{\mathbf{x}}_i^T\tilde{\mathbf{x}}_j \\
&= 2n\sum_{i\in[n]} \|\tilde{\mathbf{x}}_i\|_2^2 - 2\mathbf{1}^T\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T\mathbf{1} \\
&= 2n\sum_{i\in[n]} \|\tilde{\mathbf{x}}_i\|_2^2 - 2\|\mathbf{1}^T\tilde{\mathbf{X}}\|_2^2 \\
&= 2n^2\left(\frac{1}{n}\sum_{i=1}^n \|\tilde{\mathbf{x}}_i\|_2^2 - \|\frac{1}{n}\sum_{i=1}^n \tilde{\mathbf{x}}_i\|_2^2\right) .
\end{aligned}
$$

Step1: Compute centered representation
(This does not affect TPSD)

$$
\begin{aligned}
\tilde{\mathbf{x}}_i^c &= \tilde{\mathbf{x}}_i - \frac{1}{n}\sum_i^n \tilde{\mathbf{x}}_i \\
\|\frac{1}{n}\sum_{i=1}^n \tilde{\mathbf{x}}_i\|_2^2 &\to 0 \\
\mathrm{TPSD}(\tilde{\mathbf{X}}) &= \mathrm{TPSD}(\tilde{\mathbf{X}}^c) = 2n\|\tilde{\mathbf{X}}^c\|_F^2
\end{aligned}
$$

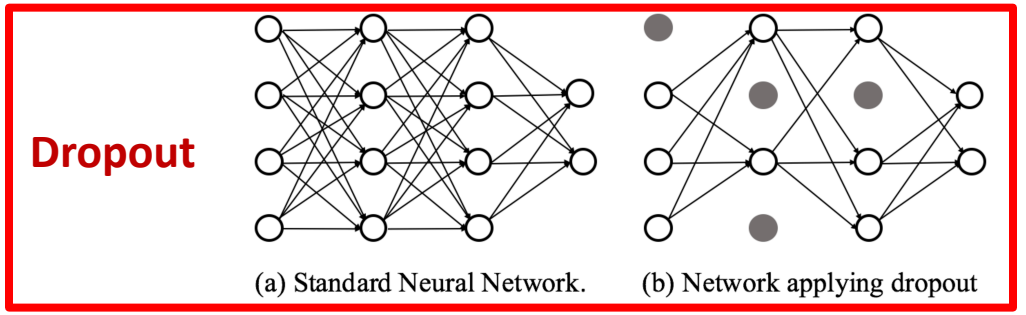
Step2: Scale the centered representation

$$
\dot{\mathbf{x}}_i = s\cdot\frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\frac{1}{n}\sum_{i=1}^n \|\tilde{\mathbf{x}}_i^c\|_2^2}} = s\sqrt{n}\cdot\frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\|\tilde{\mathbf{X}}^c\|_F^2}}
$$

$$
\mathrm{TPSD}(\dot{\mathbf{X}}) = 2n\|\dot{\mathbf{X}}\|_F^2 = 2n\sum_i \|s\cdot\frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\frac{1}{n}\sum_i \|\tilde{\mathbf{x}}_i^c\|_2^2}}\|_2^2 = 2n\frac{s^2}{\frac{1}{n}\sum_i \|\tilde{\mathbf{x}}_i^c\|_2^2}\sum_i \|\tilde{\mathbf{x}}_i^c\|_2^2 = 2n^2s^2
$$

PairNorm: Tackling Oversmoothing in GNNs, ICLR2020

DROPEDGE



- Idea: **Randomly remove a certain number of edges** from the input graph at each training epoch

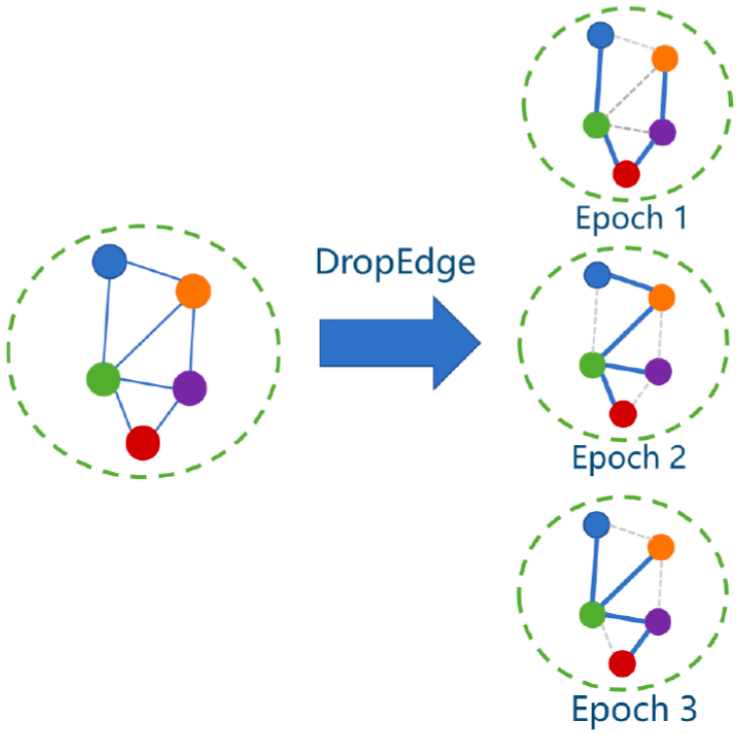


Table 1: Testing accuracy (%) comparisons on different backbones w and w/o DropEdge.

Dataset	Backbone	2 layers		8 layers		32 layers	
		Orignal	DropEdge	Orignal	DropEdge	Orignal	DropEdge
Cora	GCN	86.10	**86.50**	78.70	**85.80**	71.60	**74.60**
	ResGCN	-	-	85.40	**86.90**	85.10	**86.80**
	JKNet	-	-	86.70	**87.80**	87.10	**87.60**
	IncepGCN	-	-	86.70	**88.20**	87.40	**87.70**
	GraphSAGE	87.80	**88.10**	84.30	**87.10**	31.90	**32.20**
Citeseer	GCN	75.90	**78.70**	74.60	**77.20**	59.20	**61.40**
	ResGCN	-	-	77.80	**78.80**	74.40	**77.90**
	JKNet	-	-	79.20	**80.20**	71.70	**80.00**
	IncepGCN	-	-	79.60	**80.50**	72.60	**80.30**
	GraphSAGE	78.40	**80.00**	74.10	**77.10**	37.00	**53.60**
Pubmed	GCN	90.20	**91.20**	90.10	**90.90**	84.60	**86.20**
	ResGCN	-	-	89.60	**90.50**	90.20	**91.10**
	JKNet	-	-	90.60	**91.20**	89.20	**91.30**
	IncepGCN	-	-	90.20	**91.50**	OOM	**90.50**
	GraphSAGE	90.10	**90.70**	90.20	**91.70**	41.30	**47.90**
Reddit	GCN	96.11	**96.13**	96.17	**96.48**	45.55	**50.51**
	ResGCN	-	-	96.37	**96.46**	93.93	**94.27**
	JKNet	-	-	96.82	**97.02**	OOM	OOM
	IncepGCN	-	-	96.43	**96.87**	OOM	OOM
	GraphSAGE	96.22	**96.28**	96.38	**96.42**	96.43	**96.47**

DropEdge: Towards Deep Graph Convolutional Networks on Node Classification, ICLR2020

GCN II

- **Idea**: **Initial residual** and **Identity mapping**

GCN $\mathbf{H}^{(\ell+1)} = \sigma\left(\tilde{\mathbf{P}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}\right) \qquad \tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$

GCN II $\mathbf{H}^{(\ell+1)} = \sigma\left(\left((1-\alpha_\ell)\tilde{\mathbf{P}}\mathbf{H}^{(\ell)} + \alpha_\ell\mathbf{H}^{(0)}\right)\left((1-\beta_\ell)\mathbf{I}_n + \beta_\ell\mathbf{W}^{(\ell)}\right)\right)$

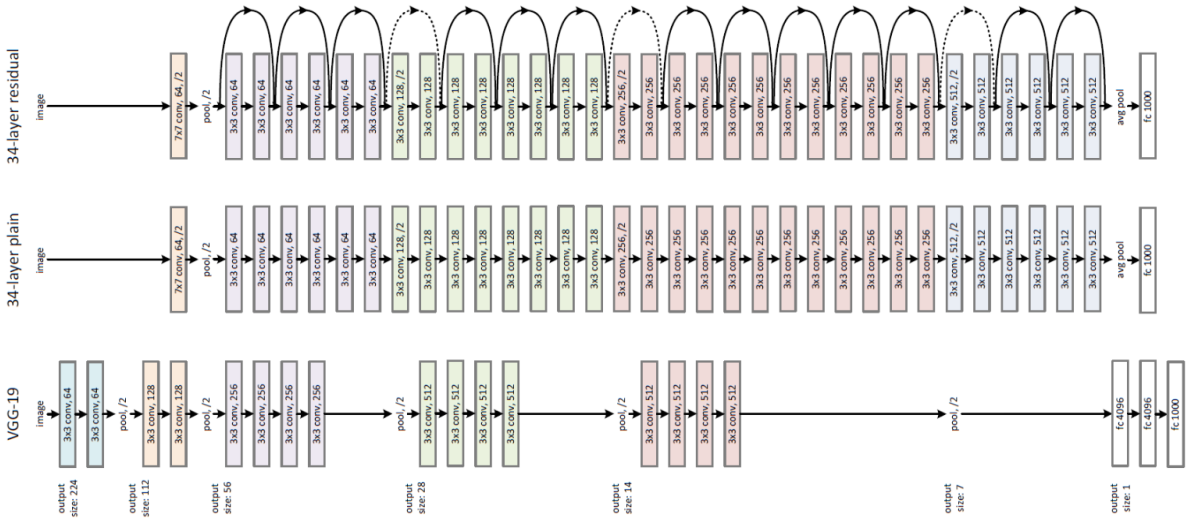
Add residual connection

Maintain a portion of information from the initial input

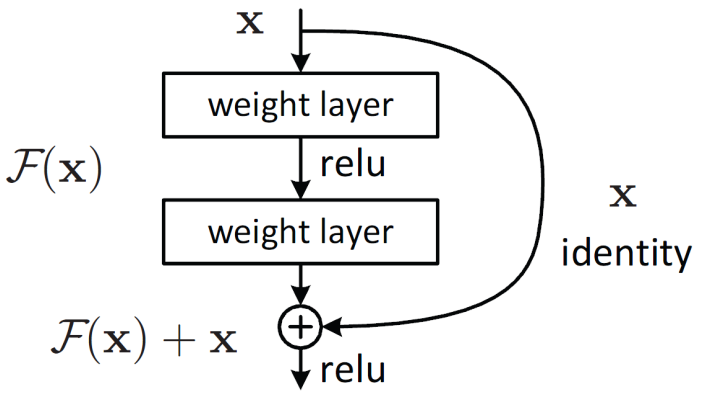
Identity mapping

The weights are shown to have small norms

Method	Cora	Citeseer	Pubmed
GCN	81.5	71.1	79.0
GAT	83.1	70.8	78.5
APPNP	83.3	71.8	80.1
JKNet	81.1 (4)	69.8 (16)	78.1 (32)
JKNet(Drop)	83.3 (4)	72.6 (16)	79.2 (32)
Incep(Drop)	83.5 (64)	72.7 (4)	79.5 (4)
GCNII	**85.5** $\pm$ **0.5** (64)	**73.4** $\pm$ **0.6** (32)	80.2 $\pm$ 0.4 (16)
GCNII*	85.3 $\pm$ 0.2 (64)	73.2 $\pm$ 0.8 (32)	**80.3** $\pm$ **0.4** (16)



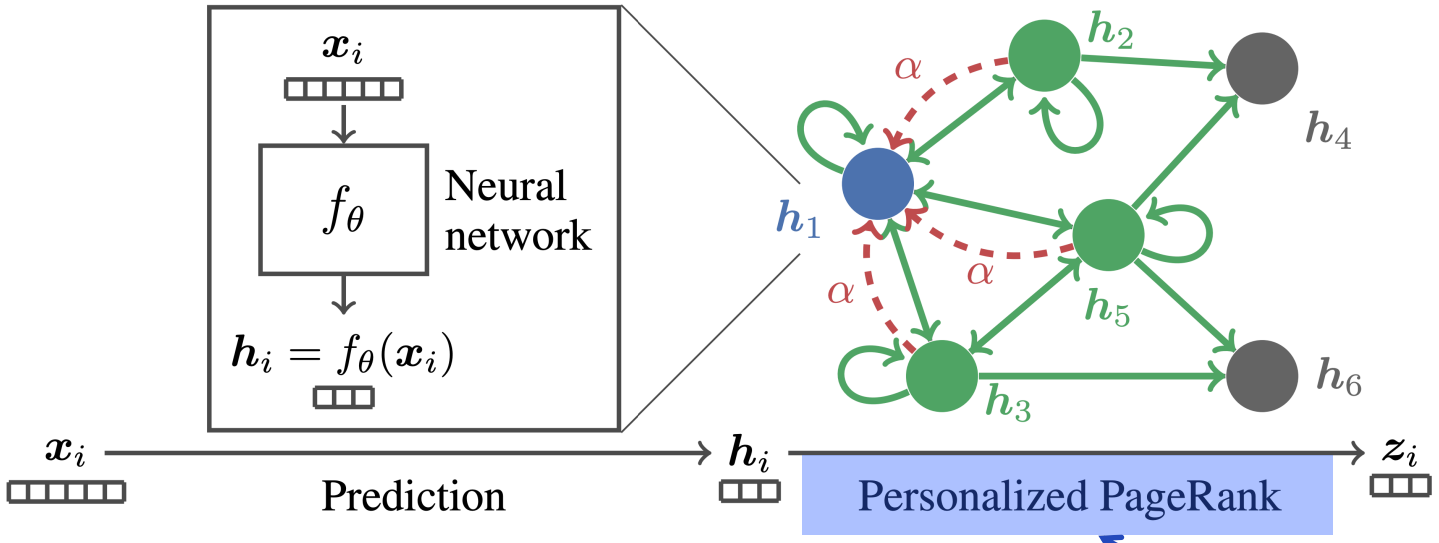
ResNet (He et al., 2016)



Simple and Deep Graph Convolutional Networks, ICML2020

APPNP

- **Idea**: Incorporate the **personalize page rank** to capture the better locality of the target node
  - Introduce the **teleport probability** α
    - Staying close to the root node to avoid oversmoothing, and leveraging the information from a large neighborhood
  - Separate the neural network from the propagation scheme



$$Z^{(0)} = H = f_\theta(X),$$

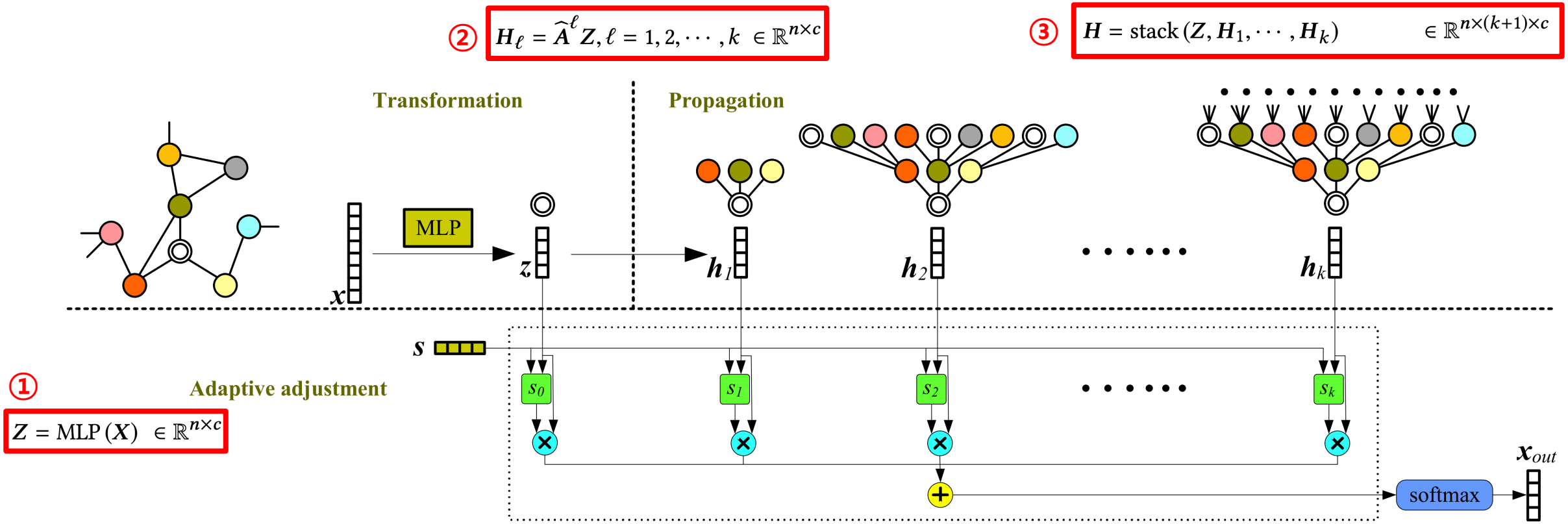
$$Z^{(k+1)} = (1-\alpha)\hat{\tilde{A}}Z^{(k)} + \alpha H,$$

$$Z^{(K)} = \text{softmax}\left((1-\alpha)\hat{\tilde{A}}Z^{(K-1)} + \alpha H\right),$$

Predict then Propagate: Graph Neural Networks meet Personalized Pagerank, ICLR2019

DEEP ADAPTIVE GRAPH NEURAL NETWORK (DAGNN)

- **Idea**: Decouple *transformation* and *propagation*



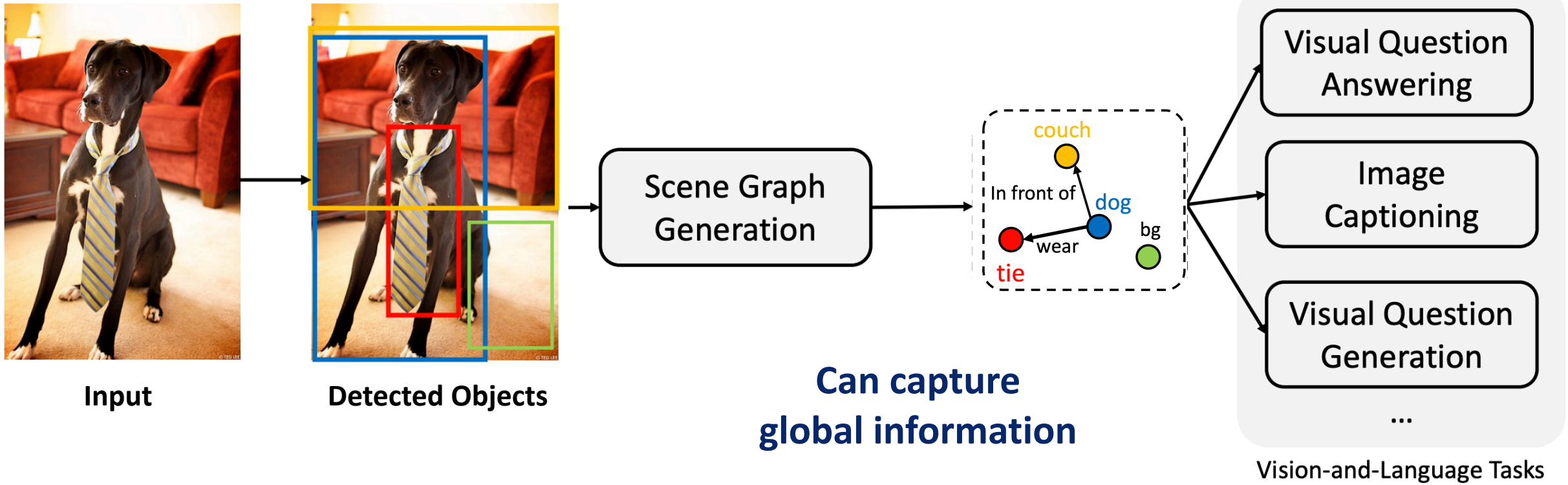
Towards Deeper Graph Neural Networks, KDD2020

OUTLINE

- Homogeneous Network Embedding
- Multi-aspect Network Embedding
- Attributed Network Embedding
- Heterogeneous Network Embedding
- Training GNN
  - Self-supervised learning
  - Going deeper with GNN
- Applications of Graph Machine Learning

APPLICATIONS OF GRAPH MACHINE LEARNING: COMPUTER VISION

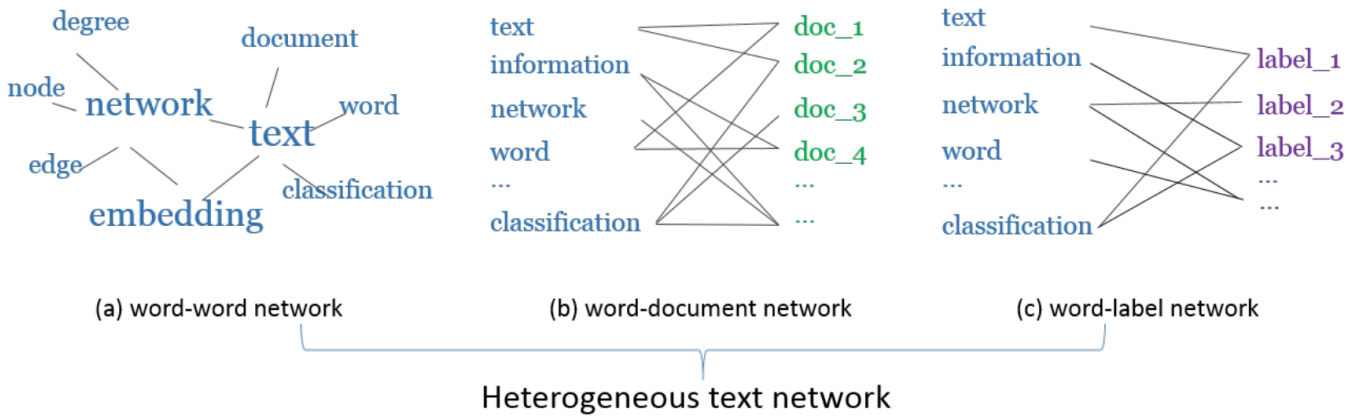
- **Scene graph**
  - **Objects** in a scene usually have **relationships** with each other



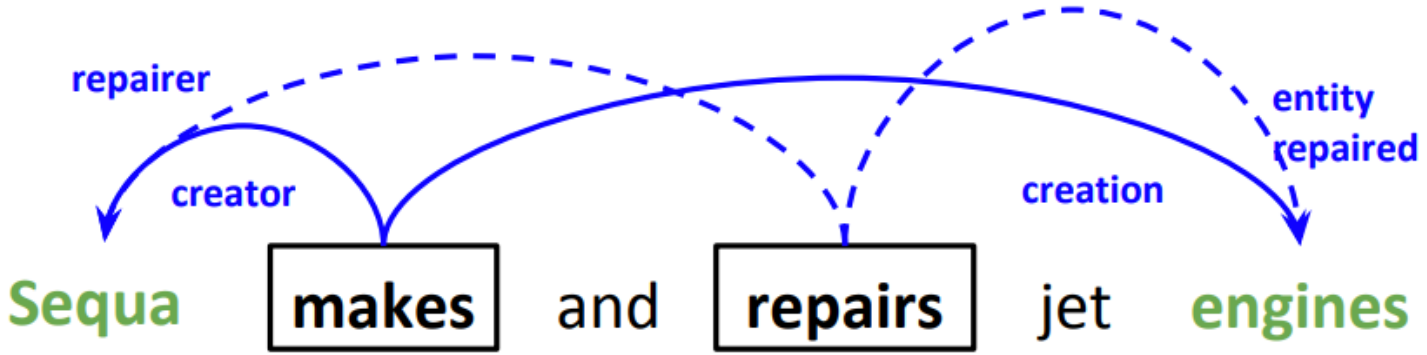
Can capture global information

APPLICATIONS OF GRAPH MACHINE LEARNING: NATURAL LANGUAGE PROCESSING

- **Document classification**, Sentiment analysis



- **Semantic role labeling**



(Figure credit): https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/fp292-Tang.pdf
(Figure credit): https://shikhar-vashishth.github.io/assets/pdf/emnlp19_tutorial.pdf

APPLICATIONS OF GRAPH MACHINE LEARNING: BIO-MEDICAL DOMAIN

Classifying the function of proteins in the interactome!

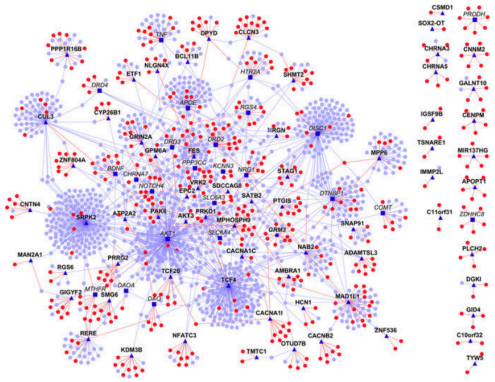
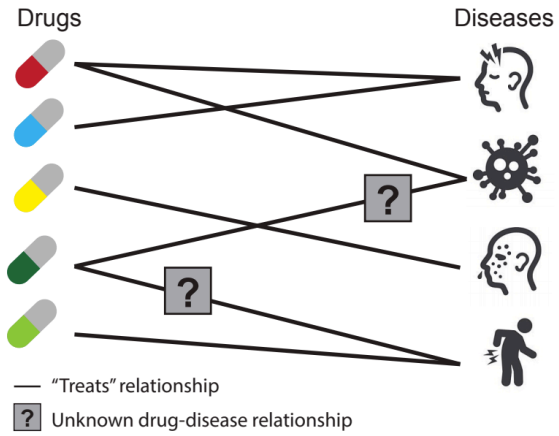


Image from: Ganapathiraju et al. 2016. Schizophrenia interactome with 504 novel protein–protein interactions. *Nature*.

Node Classification

Predicting which diseases a new molecule might treat!



Link Prediction

Identifying disease proteins in the interactome!

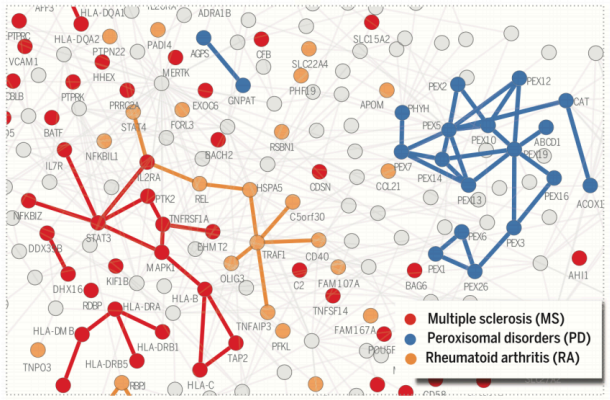


Image from: Menche et al. 2015. Uncovering disease-disease relationships through the incomplete interactome. *Science*.

Community Detection

CONCLUSION

- How to model different types of graphs?
  - Homogeneous Network Embedding
  - Multi-aspect Network Embedding
  - Attributed Network Embedding
  - Heterogeneous Network Embedding
- How to effectively training GNN?
  - Self-supervised learning for GNN
  - Going deep with GNN

REFERENCES

Homogeneous Network Embedding

- Deepwalk: Online learning of social representations, KDD2014
- node2vec: Scalable Feature Learning for Networks, KDD2016
- LINE: Large-scale Information Network Embedding, WWW2015
- Structural Deep Network Embedding, KDD2016
- GraRep: Learning Graph Representations with Global Structural Information, CIKM2015
- Deep Recursive Network Embedding with Regular Equivalence, KDD2018

Multi-aspect Network Embedding

- Is a single vector enough? Exploring node polysemy for network embedding, KDD2019
- Is a single embedding enough? learning node representations that capture multiple social contexts, WWW2019
- Unsupervised Differentiable Multi-aspect Network Embedding, KDD2020
- Categorical reparameterization with gumbel-softmax, ICLR2017
- Disentangled Graph Convolutional Networks, ICML2019

REFERENCES

Attributed Network Embedding

- Deep Attributed Network Embedding, IJCAI2018
- Revisiting Semi-Supervised Learning with Graph Embeddings, ICML16
- Semi-Supervised Classification with Graph Convolutional Networks, ICLR2017
- Inductive Representation Learning on Large Graphs, NeurIPS2017
- Simplifying Graph Convolutional Networks, ICML2019
- Graph Attention Networks, ICLR2018
- MINE: Mutual Information Neural Estimation, ICML2018
- Learning deep representations by mutual information estimation and maximization, ICLR2019
- Deep Graph Infomax, ICLR2019
- HDMI: High-order Deep Multiplex Infomax, WWW2021

REFERENCES

Heterogeneous Network Embedding

- metapath2vec: Scalable Representation Learning for Heterogeneous Networks, KDD2017
- HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning, CIKM2019
- Graph Transformer Networks, NeurIPS2019
- An Attention-based Collaboration Framework for Multi-View Network Representation Learning, CIKM2017
- Modeling Relational Data with Graph Convolutional Networks, ESWC2018
- Heterogeneous Graph Attention Network, WWW2019
- Unsupervised Attributed Multiplex Network Embedding, AAAI2020
- Task-guided and path-augmented heterogeneous network embedding for author identification, WSDM2017
- Camel: Content-Aware and Meta-path Augmented Metric Learning for Author Identification, WWW 2018
- Task-guided Pair Embedding in Heterogeneous Network, CIKM2019

REFERENCES

Training GNN (Self-supervised learning)

- Self-supervised Learning on Graphs: Deep Insights and New Directions, arxiv2020
- Self-Supervised Graph Representation Learning via Global Context Prediction, arxiv2020
- Strategies for Pre-training Graph Neural Networks, ICLR2020
- Contrastive Multi-View Representation Learning on Graphs, ICML2020
- GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training, KDD2020
- GPT-GNN: Generative Pre-Training of Graph Neural Networks, KDD 2020
- Multi-stage self-supervised learning for graph convolutional networks on graphs with few labels, AAAI2020

Training GNN (Going deeper with GNN)

- Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning, AAAI2018
- PairNorm: Tackling Oversmoothing in GNNs, ICLR2020
- DropEdge: Towards Deep Graph Convolutional Networks on Node Classification, ICLR2020
- Simple and Deep Graph Convolutional Networks, ICML2020
- Predict then Propagate: Graph Neural Networks meet Personalized Pagerank, ICLR2019
- Towards Deeper Graph Neural Networks, KDD2020

THANK YOU

- Contact: cy.park@kaist.ac.kr
- Lab homepage: http://dsail.kaist.ac.kr

GUMBEL-SOFTMAX

- A simple way to draw a one-hot sample $\boldsymbol{z}$ from the **categorical distribution**
- **Given**: A K-dimensional **categorical distribution** with class probability $\pi_1, \pi_2, \ldots, \pi_K$

Non-differentiable

Gumbel noise drawn from Gumbel(0,1)

$$g_i = -\log(-\log(u_i))$$
$$u_i \sim Uniform(0,1)$$

☹ Gumbel-max

$$z = \text{one-hot}\left(\text{argmax}_i \left[\log \pi_i + g_i\right]\right)$$

Differentiable

☺ Gumbel-softmax

$$z_i = \text{softmax}\left[\log \pi_i + g_i\right]$$
$$= \frac{\exp\left(\left(\log \pi_i + g_i\right)/\tau\right)}{\sum_{j=1}^{K} \exp\left(\left(\log \pi_j + g_j\right)/\tau\right)}$$
$$\text{for } i = 1, \ldots, K$$

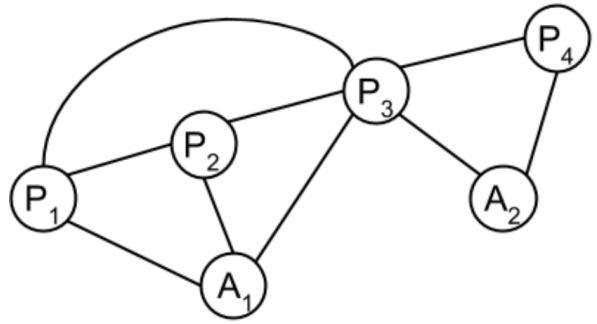
Temperature parameter

As $\tau \to 0$, samples from the Gumbel-Softmax distribution become one-hot

Continuous relaxation of discrete random variable

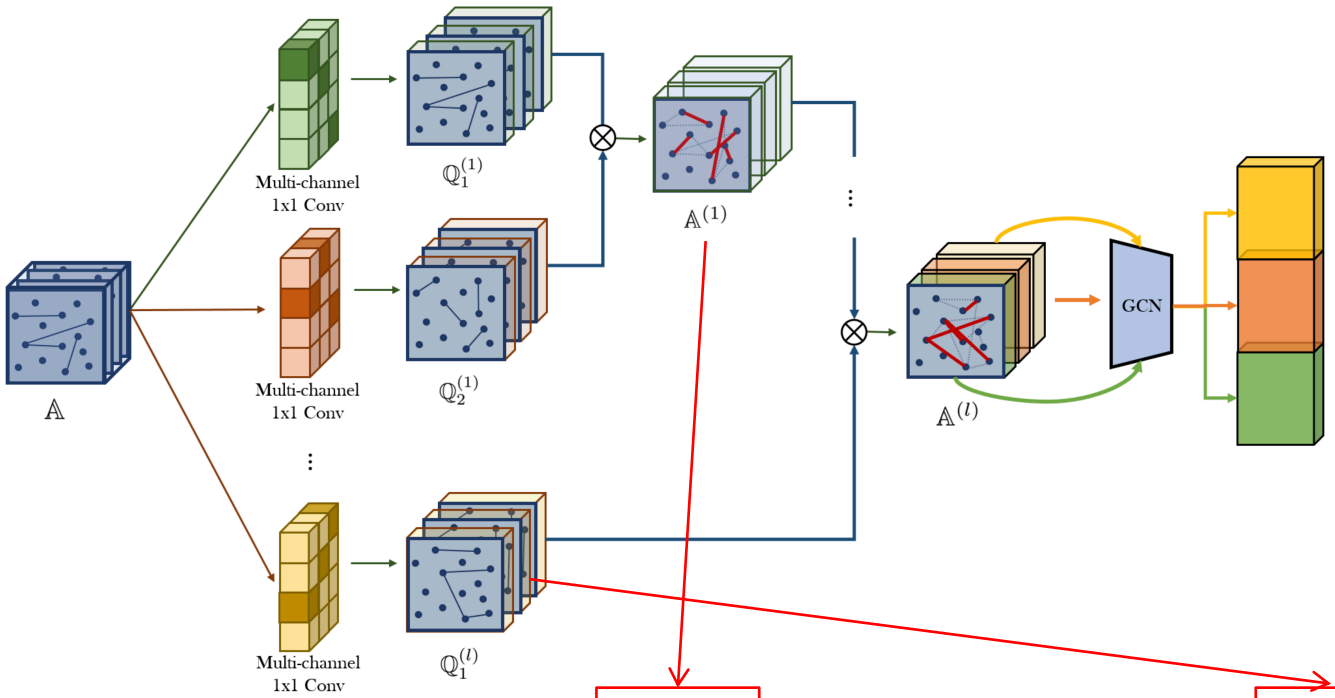
Categorical reparameterization with gumbel-softmax, ICLR2017

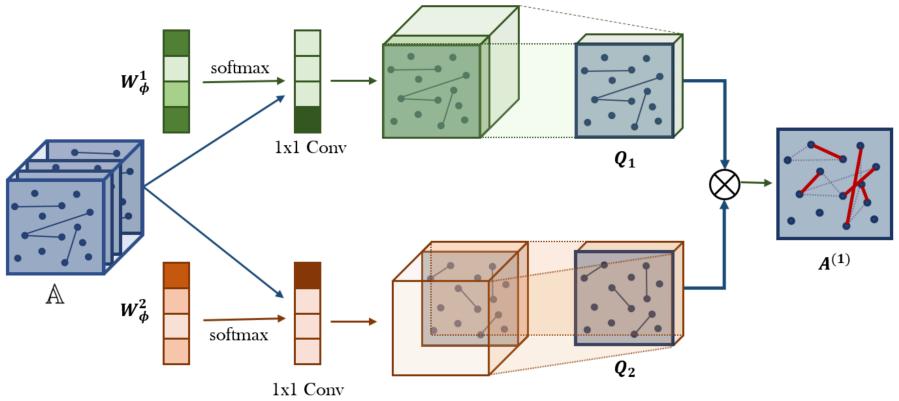
GRAPH TRANSFORMER NETWORKS



$R = \{\text{P-P, P-A, A-P}\}$

- **Motivation**: Do we need predefined metapaths?
- **Idea**: Automatically learn useful meta-paths for given data and tasks





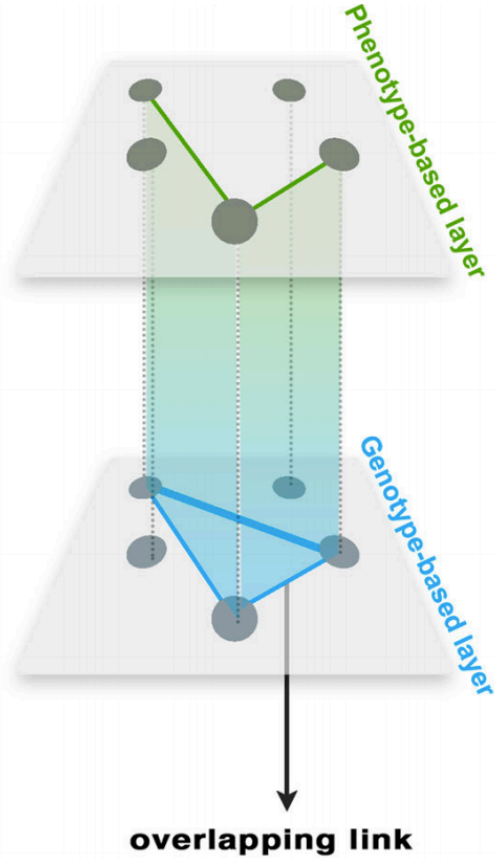
$$A_P = \left(\sum_{t_1 \in \mathcal{T}^e} \alpha_{t_1}^{(1)} A_{t_1}\right)\left(\sum_{t_2 \in \mathcal{T}^e} \alpha_{t_2}^{(2)} A_{t_2}\right)\cdots\left(\sum_{t_l \in \mathcal{T}^e} \alpha_{t_l}^{(l)} A_{t_l}\right)$$

$$Z = \Big\Vert_{i=1}^{C} \sigma(\tilde{D}_i^{-1}\tilde{A}_i^{(l)}XW),$$

Final node embeddings

Graph Transformer Networks, NeurIPS2019

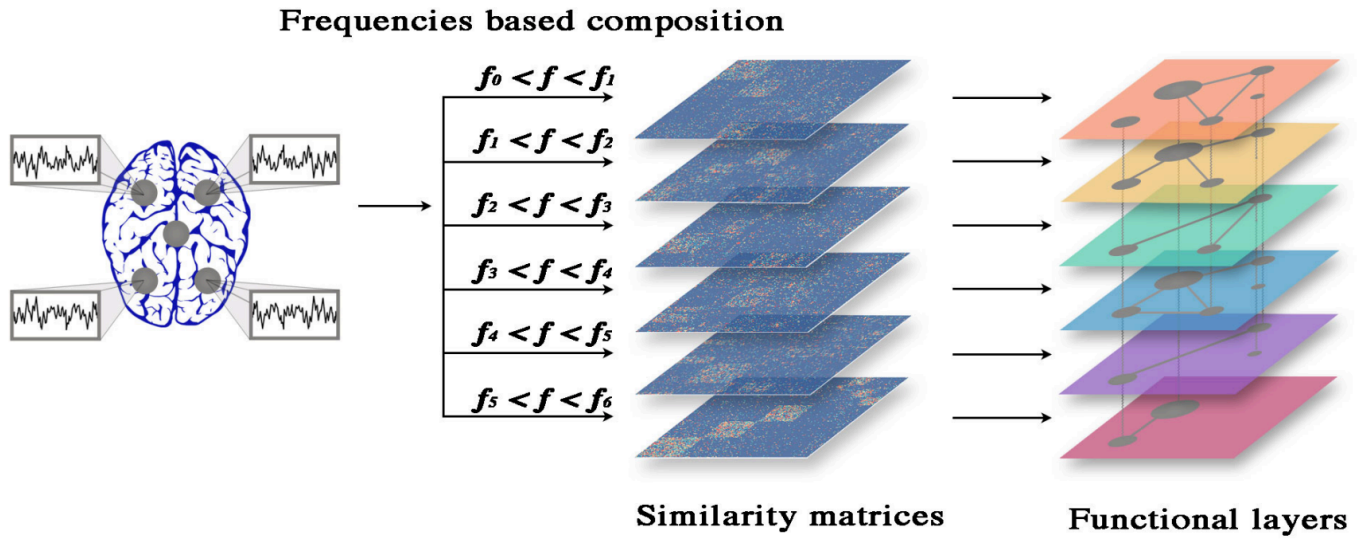
OTHER EXAMPLES OF MULTIPLEX NETWORK



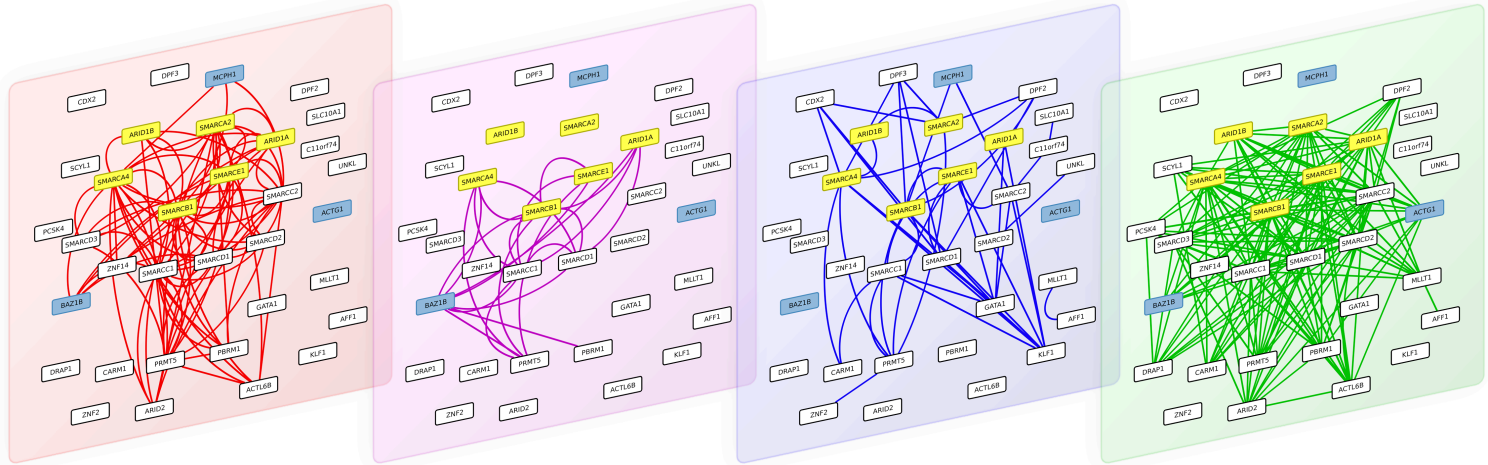
Share symptom

Share gene

Disease-Disease network



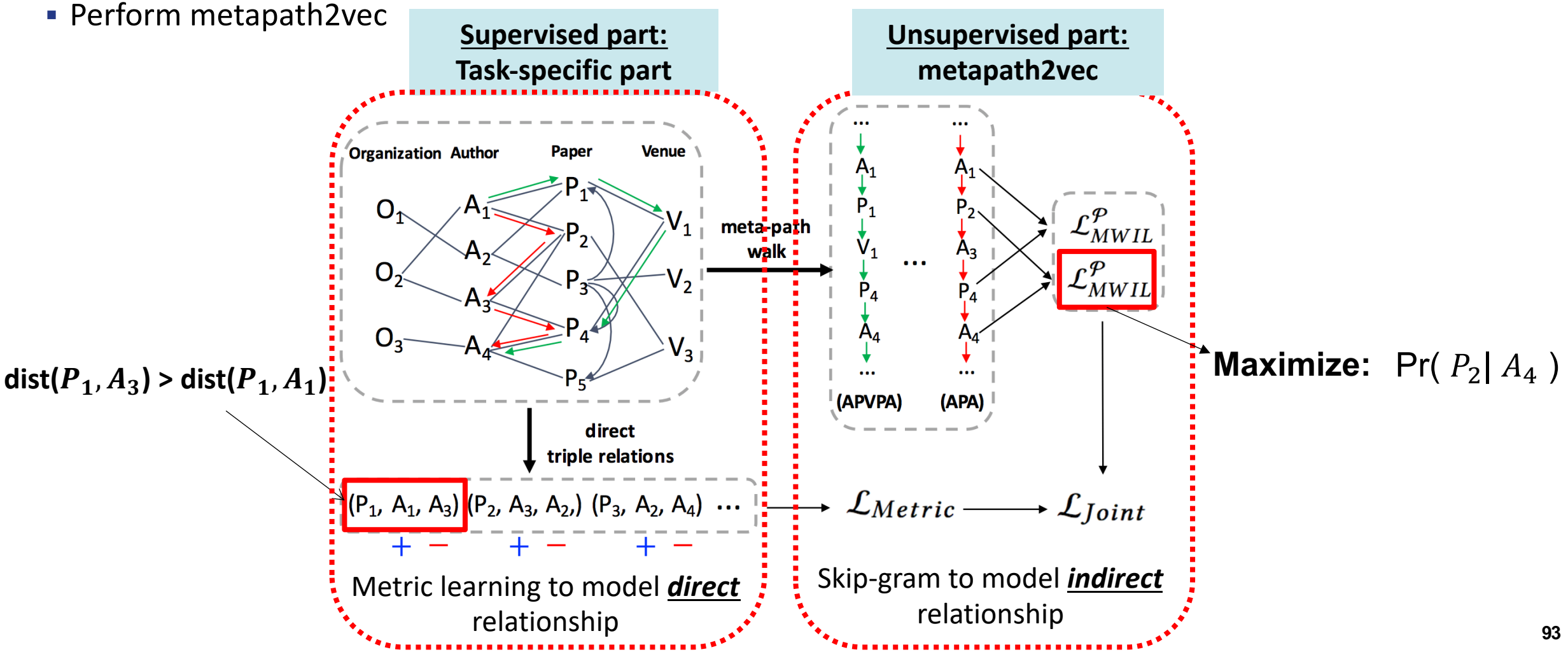
Frequency based composition of brain



Multiplex biological networks

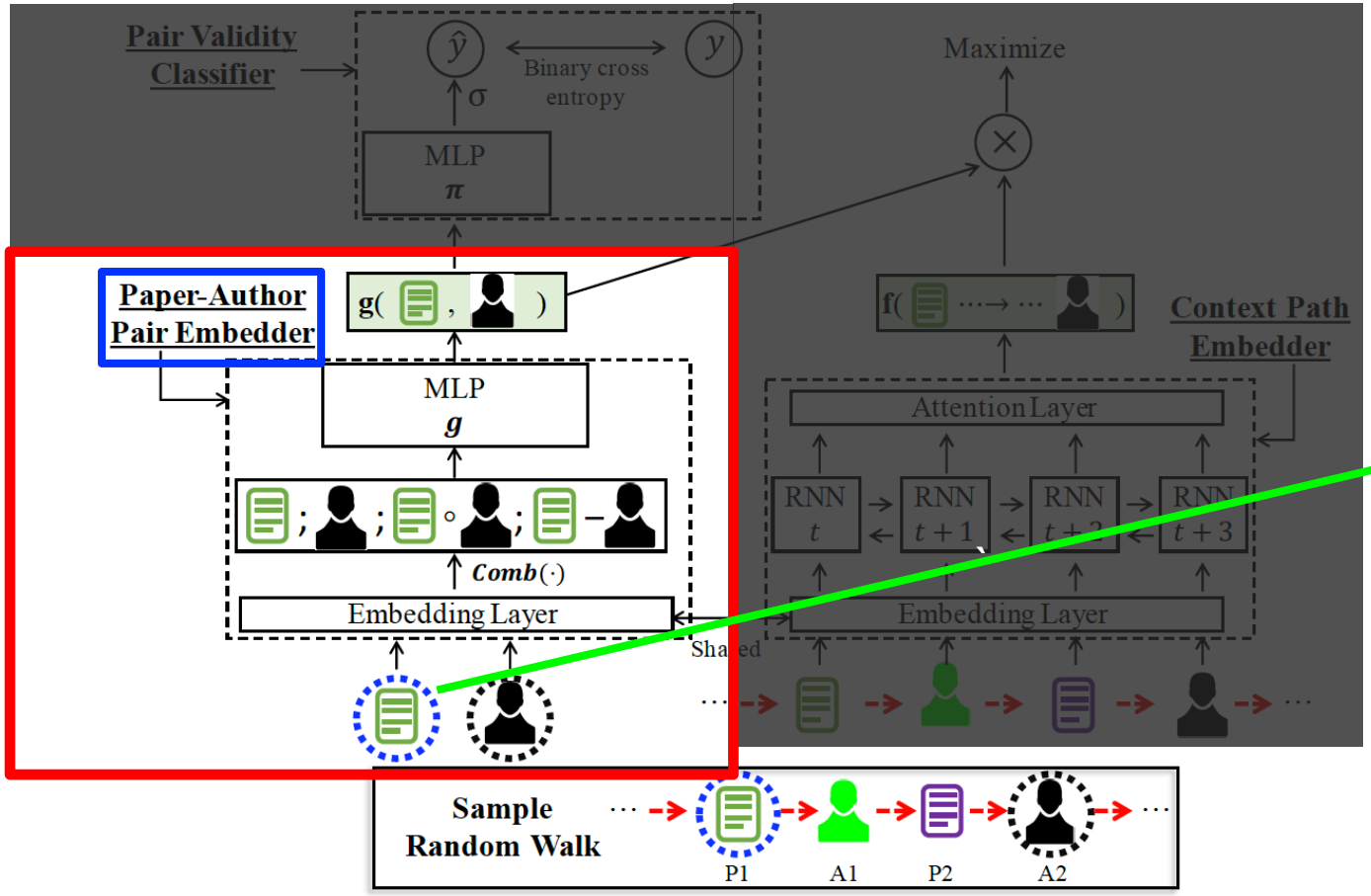
CAMEL

- Model the paper abstract using a GRU-based encoder
- Perform metapath2vec

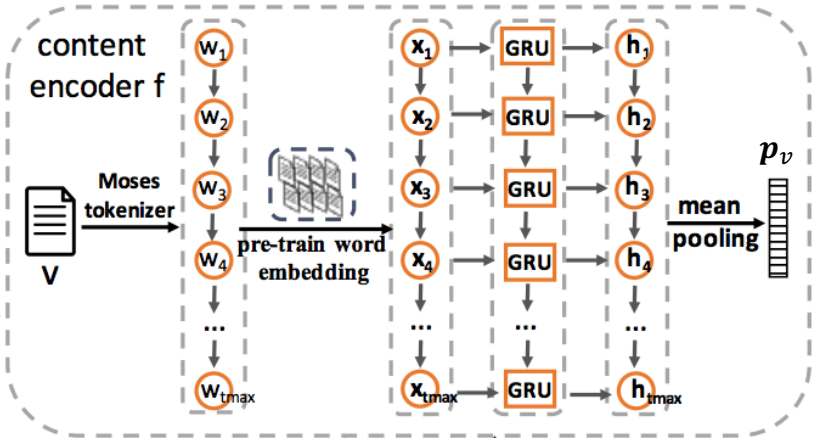


TAPEM

- **1) Context Path-aware Pair Embedder**
  - Step 1: Pair Embedder (Embedding Paper–Author Pair)

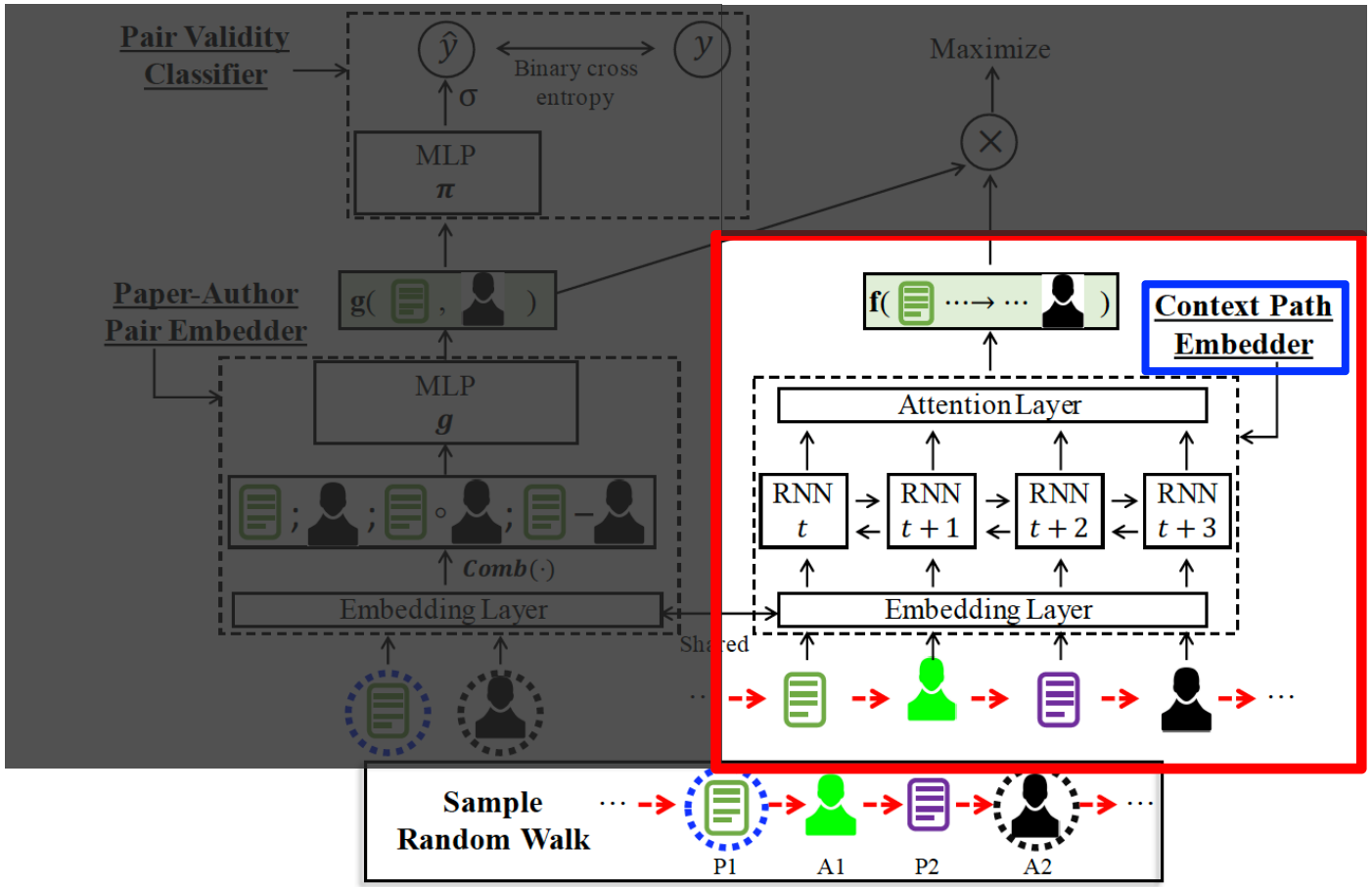


$$\mathbf{p}_v = \text{PaperEncoder}(O_v)$$



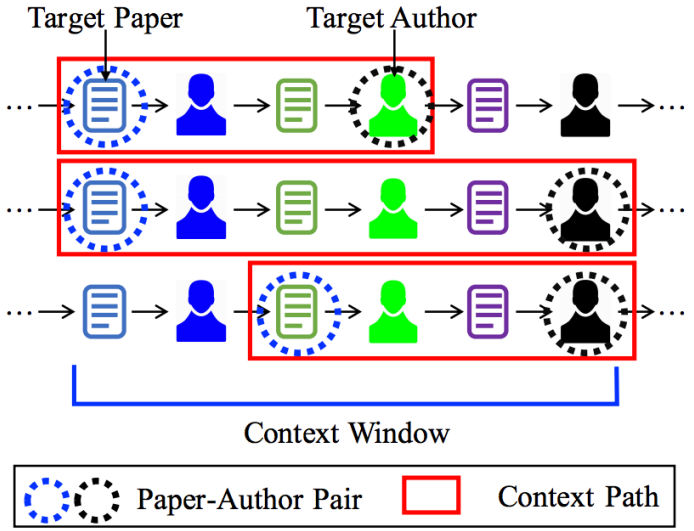
TAPEM

- **1) Context Path-aware Pair Embedder**
  - Step 2: Context Path Embedder (Embedding Context Path)



What is a **context path**?

A sequence of nodes between a target node pair

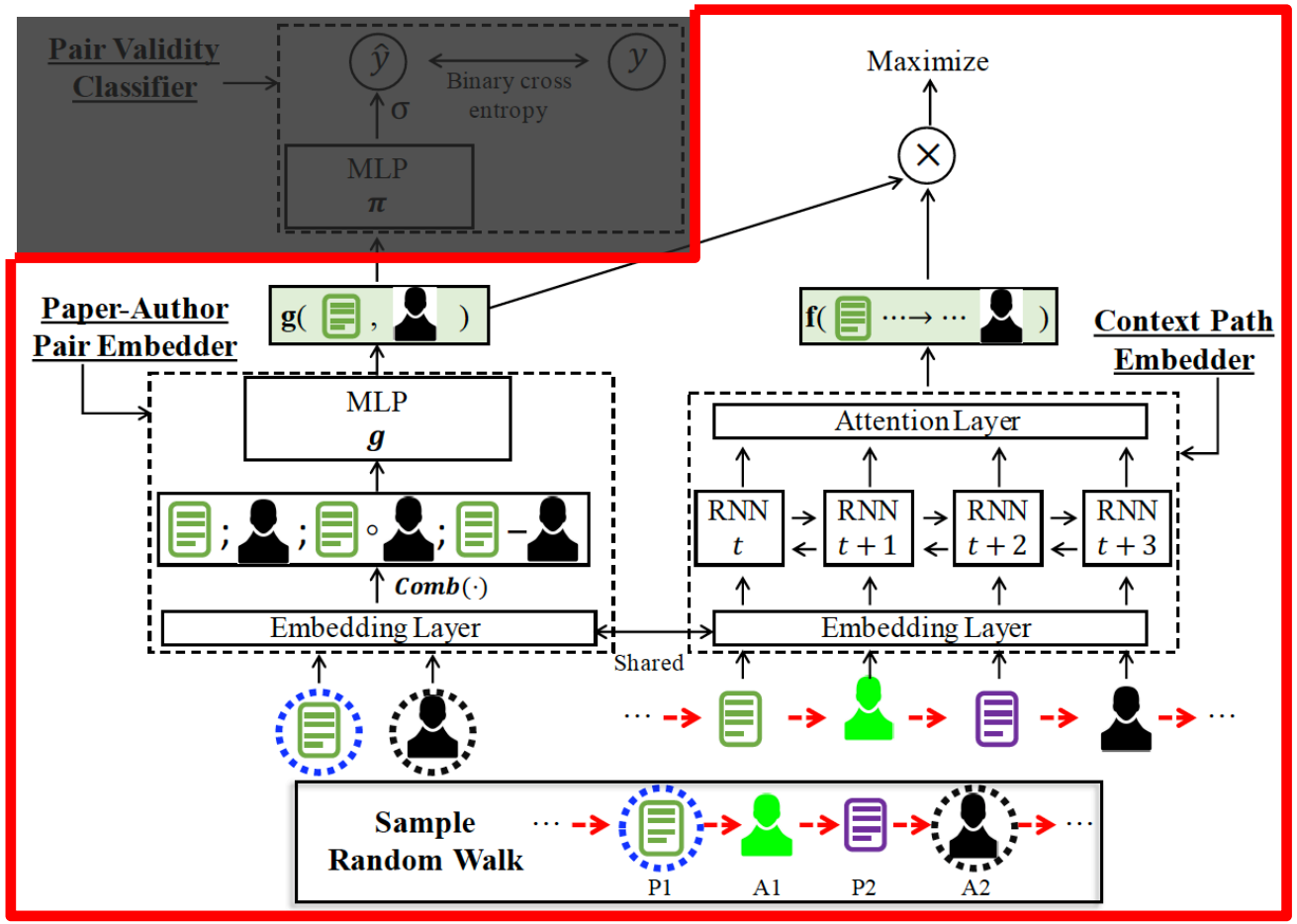


Why do we consider the **context path**?

We can infer the research topic related to the pair (v, u) by examining the path between paper v and author u

TAPEM

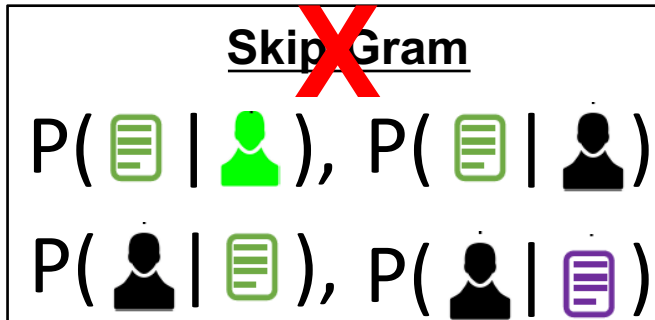
- **1) Context Path-aware Pair Embedder**
  - Step 3: Injecting Context Information into Pairs



Objective (Pair embedding)

Predict pair using its context path

~~Skip-Gram~~



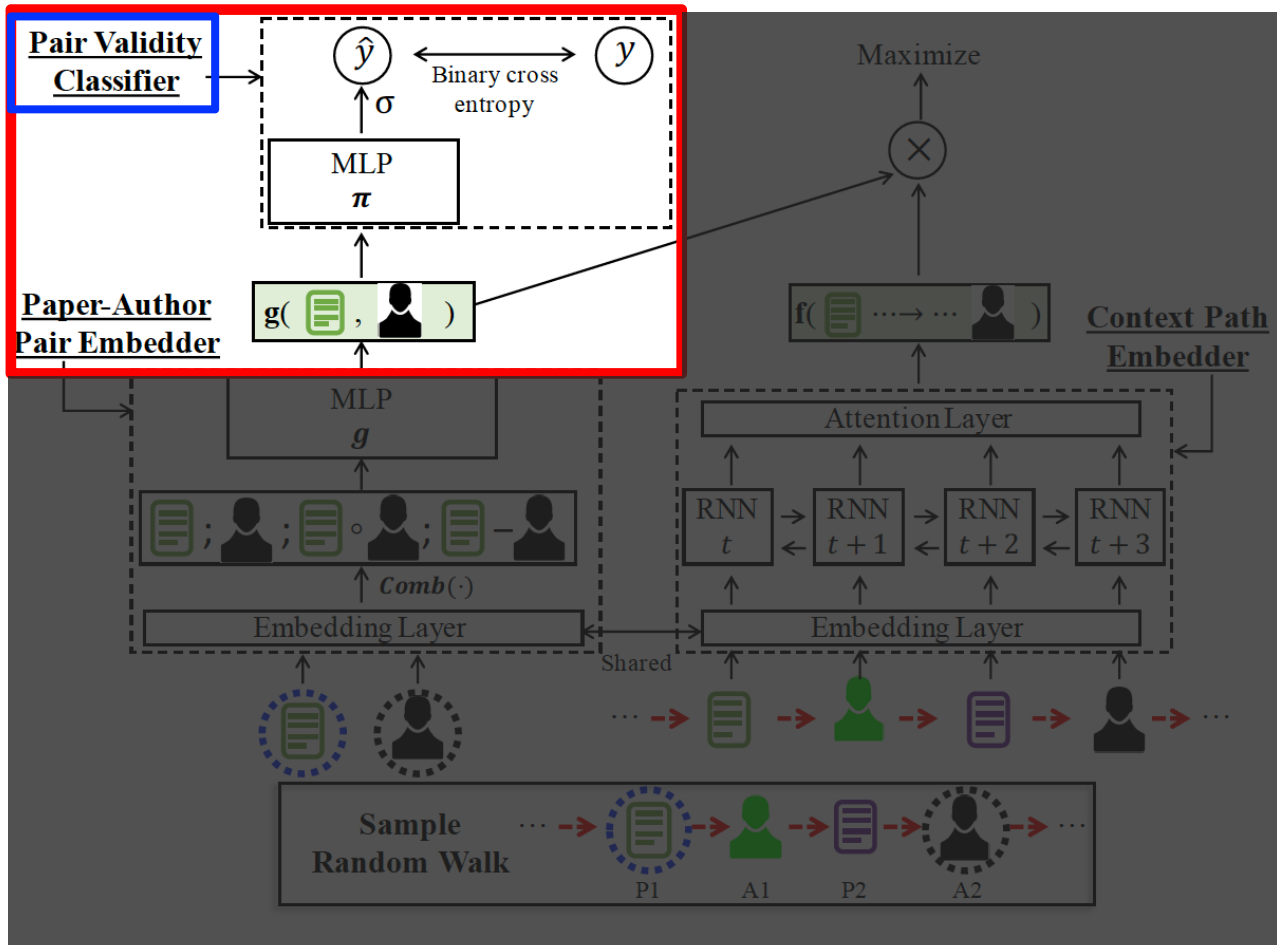
Benefit

Pair embedding ≈ Embeddings of frequent context paths

→ Pair embedding encodes its related research topic

TAPEM

- **2) Pair Validity Classifier** (Validity of Pair Embedding)



Objective

- Classify whether the pair is valid or not

$$\mathcal{L}_{\text{pv}}(v,u) = y_{v,u}\sigma(\boldsymbol{\pi}(\mathbf{g}(v,u))) + (1-y_{v,u})(1-\sigma(\boldsymbol{\pi}(\mathbf{g}(v,u))))$$

$$y_{v,u} = \begin{cases} 1, & \text{paper } v \text{ is written by author } u \\ 0, & \text{paper } v \text{ is not written by author } u \end{cases}$$

Benefit

- Enables to identify **relatively less active authors**
  - The training of the embedding is no longer solely based on the frequency (Limitation of Skip-Gram)
- Two nodes will be embedded close to each other if
  1. Related to a similar research topic
  2. **The pair itself is valid**

TAPEM: JOINT OBJECTIVE

$$\mathcal{L} = \sum_{\mathcal{P} \in \mathcal{S}(\mathcal{P})} \sum_{\boldsymbol{w} \in \mathcal{W}_{\mathcal{P}}} \sum_{v \in \boldsymbol{w}} \sum_{u \in \boldsymbol{w}[C_v - \tau : C_v + \tau]} \left[\mathcal{L}_{\text{ctx}}(v, u) + \mathcal{L}_{\text{pv}}(v, u) \right]$$

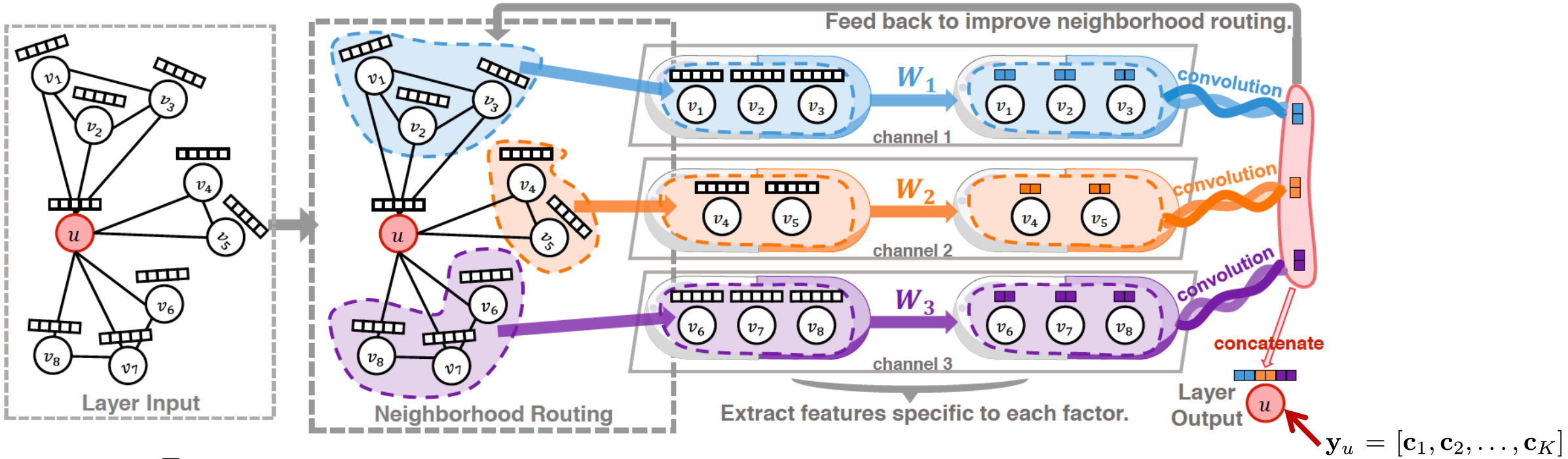
Context Path-aware Pair Embedder — $\mathcal{L}_{\text{ctx}}(v, u)$

Pair Validity Classifier — $\mathcal{L}_{\text{pv}}(v, u)$

- $S(P)$: a set of meta-path scheme
- W_p: a set of random walks guided by meta-path p
- τ: window size
- C_v: position of paper v in walk w

DISENGCN

- **Challenge**: How to identify the subset of neighbors that are actually connected by node u due to factor k?



$$\mathbf{y}_u = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K]$$

$$\mathbf{z}_{i,k} = \frac{\sigma(\mathbf{W}_k^\top \mathbf{x}_i + \mathbf{b}_k)}{\left\|\sigma(\mathbf{W}_k^\top \mathbf{x}_i + \mathbf{b}_k)\right\|_2},$$

GCN

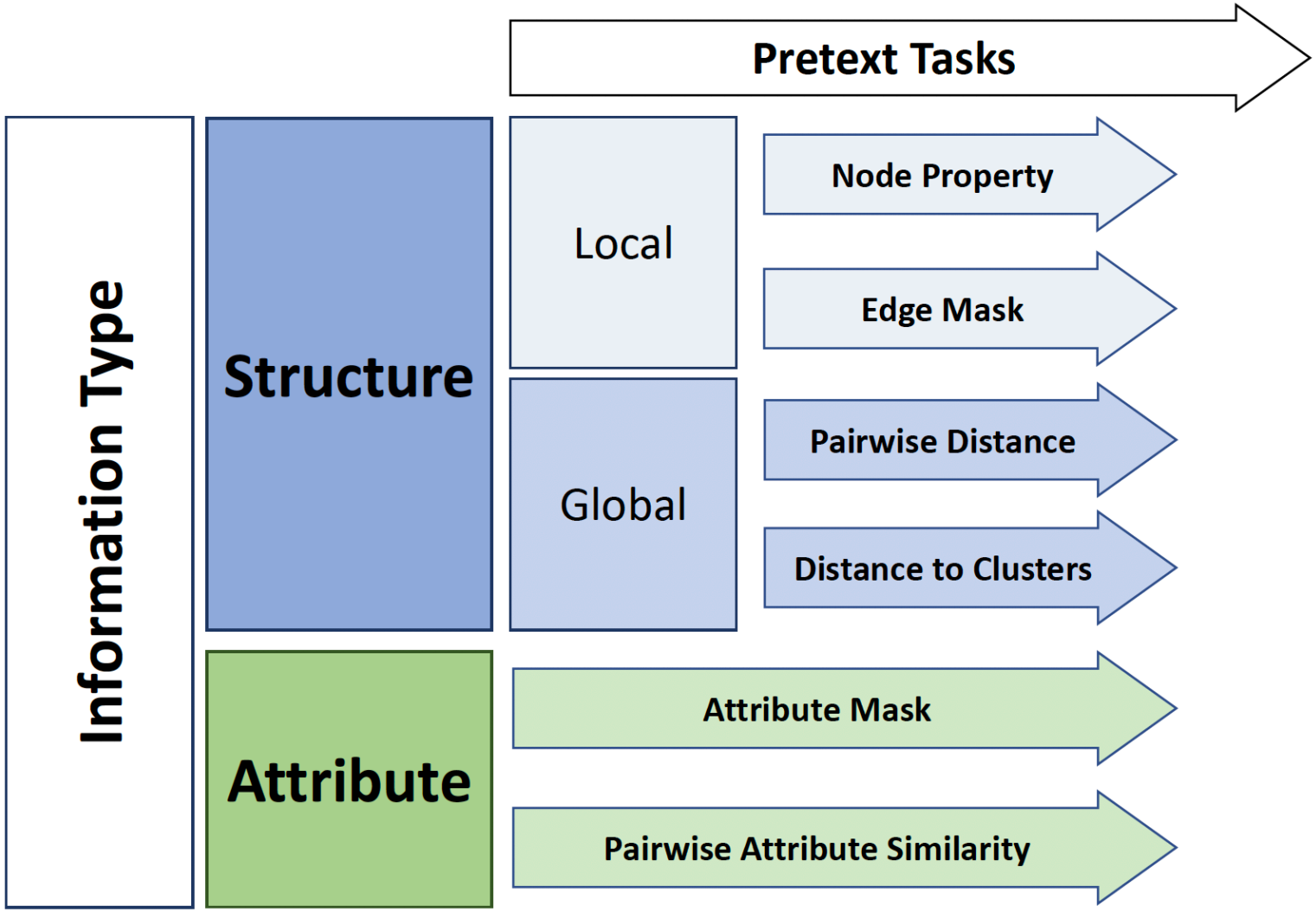
$$\mathbf{W}_k \in \mathbb{R}^{d_{in} \times \frac{d_{out}}{K}}$$

$$\mathbf{c}_k^{(t)} = \frac{\mathbf{z}_{u,k} + \sum_{v:(u,v)\in G} p_{v,k}^{(t-1)}\, \mathbf{z}_{v,k}}{\left\|\mathbf{z}_{u,k} + \sum_{v:(u,v)\in G} p_{v,k}^{(t-1)}\, \mathbf{z}_{v,k}\right\|_2},$$

$$p_{v,k}^{(t)} = \frac{\exp(\mathbf{z}_{v,k}{}^\top \mathbf{c}_k^{(t)}/\tau)}{\sum_{k'=1}^{K} \exp(\mathbf{z}_{v,k'}{}^\top \mathbf{c}_{k'}^{(t)}/\tau)},$$

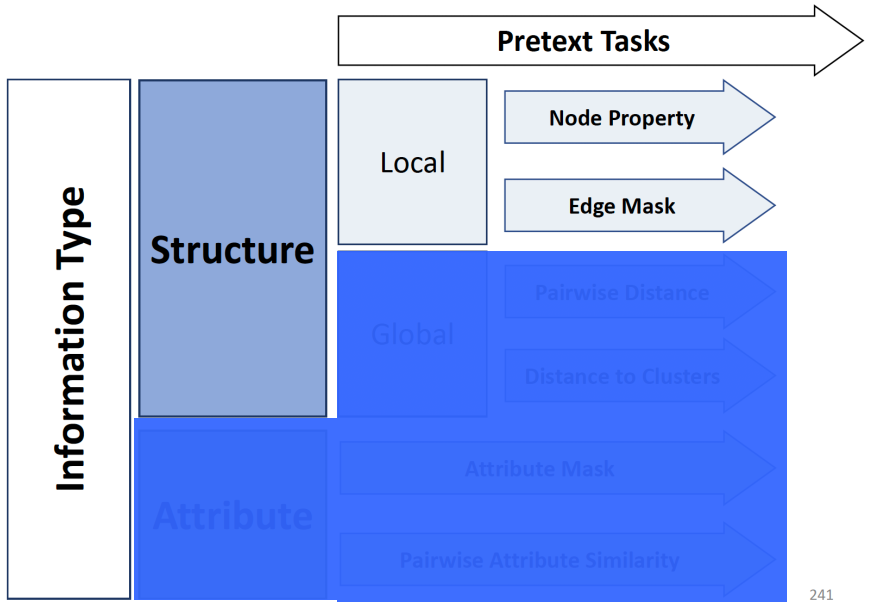
Probability that factor k is the reason why node u reaches neighbor v

EXAMPLES OF PRETEXT TASKS ON GRAPHS



(Figure credit) Tutorial in Graph Neural Networks: Models and Applications, AAAI2021

LOCAL STRUCTURE BASED PRETEXT TASK



- Node property
  - **Goal**: To predict the property for each node in the graph such as their *degree*, *local node importance*, and *local clustering coefficient*.

Predicted degree of node v_i

$$\mathcal{L}_{self}(\theta', \mathbf{A}, \mathbf{X}, \mathcal{D}_U) = \frac{1}{|\mathcal{D}_U|} \sum_{v_i \in \mathcal{D}_U} (f_{\theta'}(\mathcal{G})_{v_i} - d_i)^2$$

Degree of node v_i

- Edge mask
  - **Goal**: To predict *whether or not there exists a link between a given node pair*

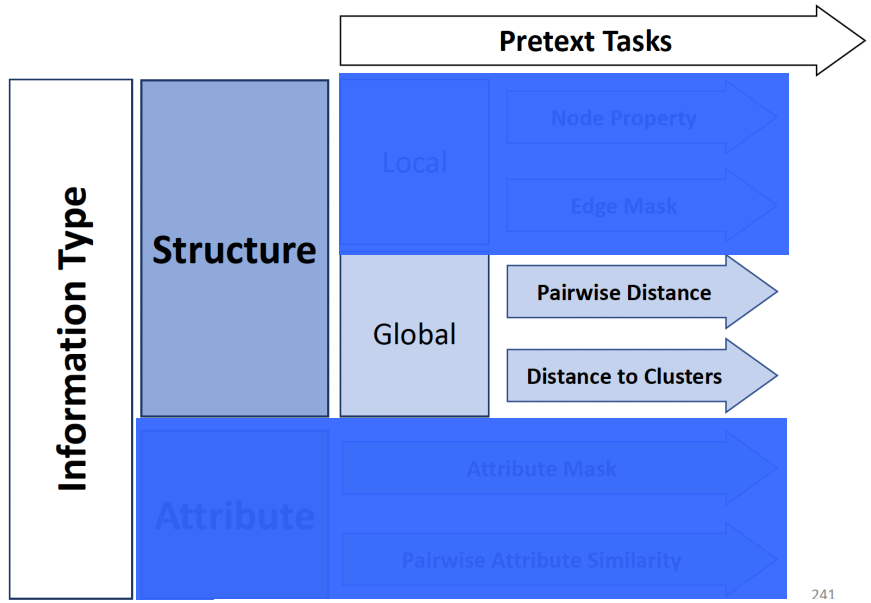
Cross-entropy loss

$$\mathcal{L}_{self}(\theta', \mathbf{A}, \mathbf{X}, \mathcal{D}_U) = \frac{1}{|\mathcal{M}_e|} \sum_{(v_i, v_j) \in \mathcal{M}_e} \ell\left(f_w(|f_{\theta'}(\mathcal{G})_{v_i} - f_{\theta'}(\mathcal{G})_{v_j}|), 1\right) + \frac{1}{|\overline{\mathcal{M}}_e|} \sum_{(v_i, v_j) \in \overline{\mathcal{M}}_e} \ell\left(f_w(|f_{\theta'}(\mathcal{G})_{v_i} - f_{\theta'}(\mathcal{G})_{v_j}|), 0\right)$$

Connected edges

Not connected edges

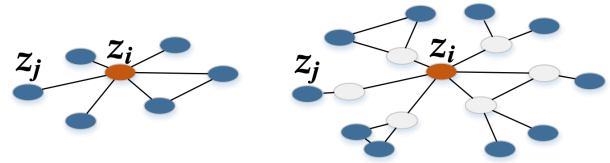
GLOBAL STRUCTURE BASED PRETEXT TASK



- Pairwise distance
  - **Goal**: To predict the distance between different node pair.

$$\mathcal{L}_{self}(\theta', \mathbf{A}, \mathbf{X}, \mathcal{D}_U) = \frac{1}{|\mathcal{S}|} \sum_{(v_i, v_j) \in \mathcal{S}} \ell\left(f_w(|f_{\theta'}(\mathcal{G})_{v_i} - f_{\theta'}(\mathcal{G})_{v_j}|), C_{p_{ij}}\right)$$

Pairwise distance between node v_i and v_j



1-hop context $h(\langle z_i, z_j \rangle, y=0)$

2-hop context $h(\langle z_i, z_j \rangle, y=1)$

- Distance2Clusters
  - **Goal**: To predict the distance from the unlabeled nodes to predefined graph clusters
  - Step 1: Apply graph clustering to get k clusters $\{C_1, C_2, \ldots, C_k\}$
  - Step 2: In each cluster C_j, assume the node with the highest degree as the center node

$$\mathcal{L}_{self}(\theta', \mathbf{A}, \mathbf{X}, \mathcal{D}_U) = \frac{1}{|\mathcal{D}_U|} \sum_{v_i \in \mathcal{D}_U} \|f_{\theta'}(\mathcal{G})_{v_i} - \mathbf{d}_i\|^2$$

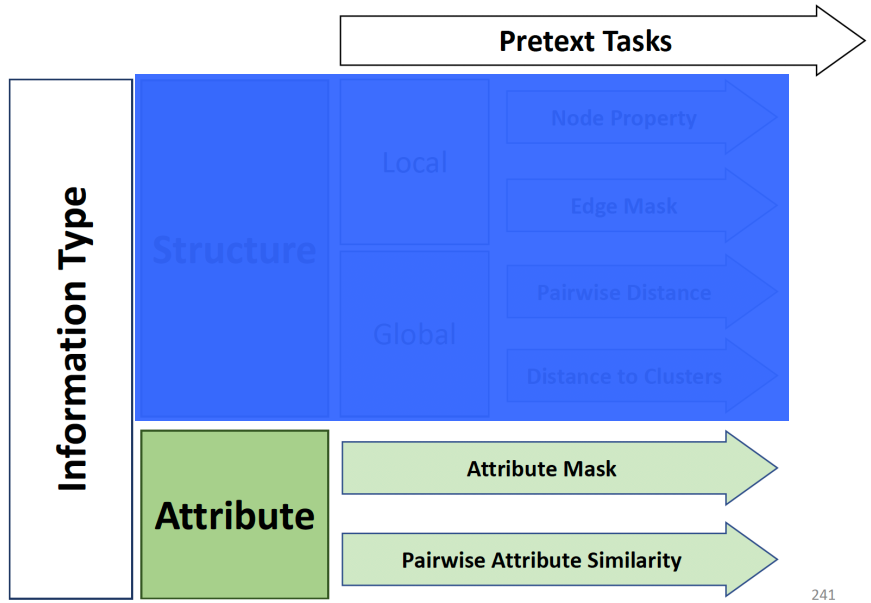
$$\boldsymbol{d_i} = [d_{i1}, d_{i2}, \ldots, d_{ik}]$$

Distance from node v_i to cluster c_2

Self-Supervised Graph Representation Learning via Global Context Prediction, arxiv2020

Self-supervised Learning on Graphs: Deep Insights and New Directions, arxiv2020

ATTRIBUTE BASED PRETEXT TASK



- Attribute mask
  - **Goal**: To predict the masked attribute
  - Apply PCA to reduce the dimensionality of features

$$\mathcal{L}_{self}(\theta', \mathbf{A}, \mathbf{X}, \mathcal{D}_U) = \frac{1}{|\mathcal{M}_a|} \sum_{v_i \in \mathcal{M}_a} \| f_{\theta'}(\mathcal{G})_{v_i} - \mathbf{x}_i \|^2$$

Feature of node v_i

- Pairwise attribute similarity
  - **Goal**: To predict the similarity of pairwise node features

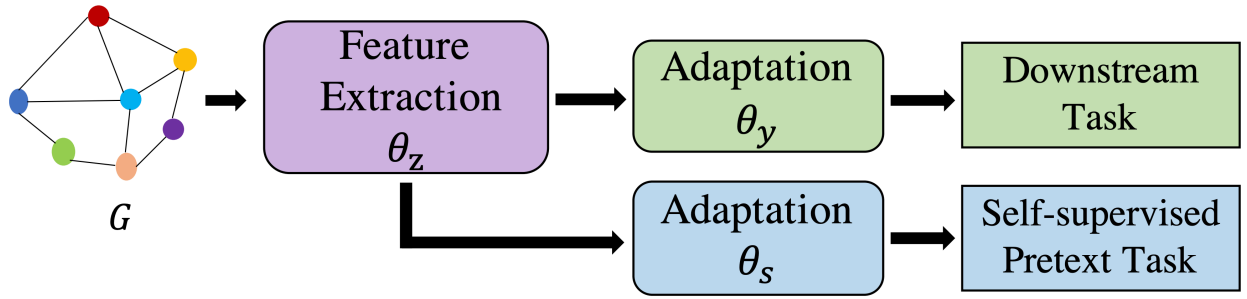
$$\mathcal{L}_{self}(\theta', \mathbf{A}, \mathbf{X}, \mathcal{D}_U) = \frac{1}{|\mathcal{T}|} \sum_{(v_i, v_j) \in \mathcal{T}} \| f_w(|f_{\theta'}(\mathcal{G})_{v_i} - f_{\theta'}(\mathcal{G})_{v_j}|) - s_{ij} \|^2$$

Self-supervised Learning on Graphs: Deep Insights and New Directions, arxiv2020

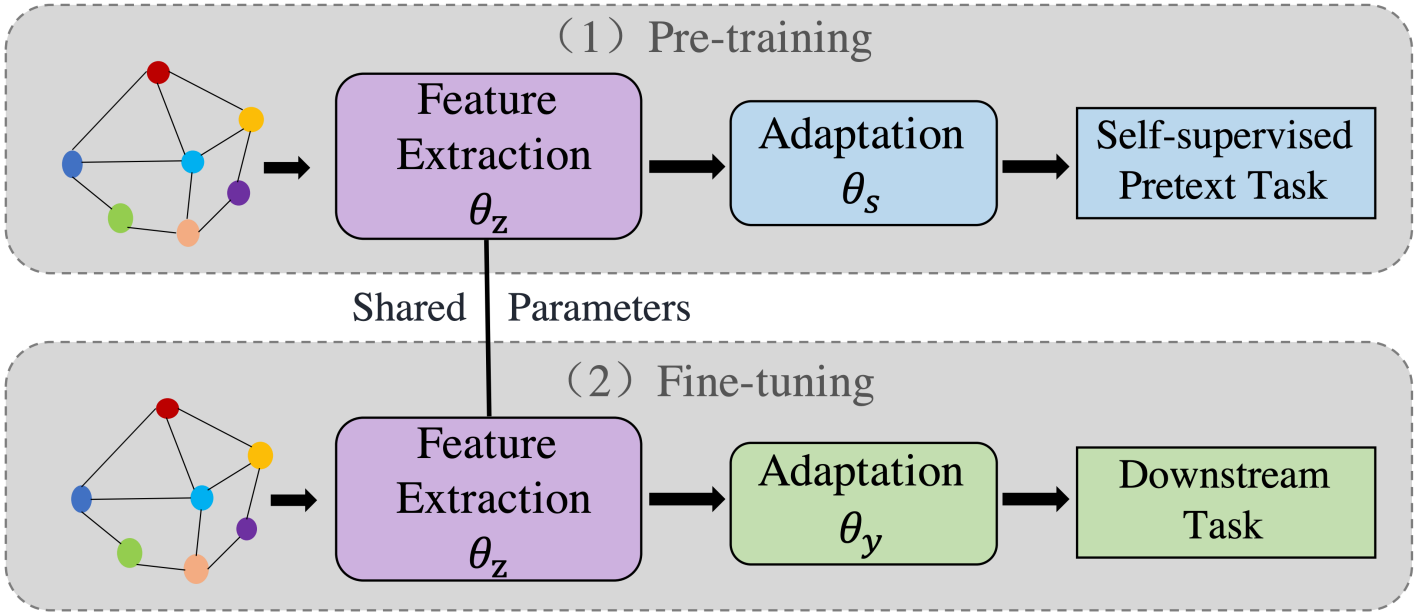
MAIN STRATEGIES FOR SSL IN GNN

- **Joint training**

$$\min_{\theta,\theta'} \mathcal{L}_{task}\left(\theta, \mathbf{A}, \mathbf{X}, \mathcal{D}_L\right) + \lambda \mathcal{L}_{self}\left(\theta', \mathbf{A}, \mathbf{X}, \mathcal{D}_U\right)$$

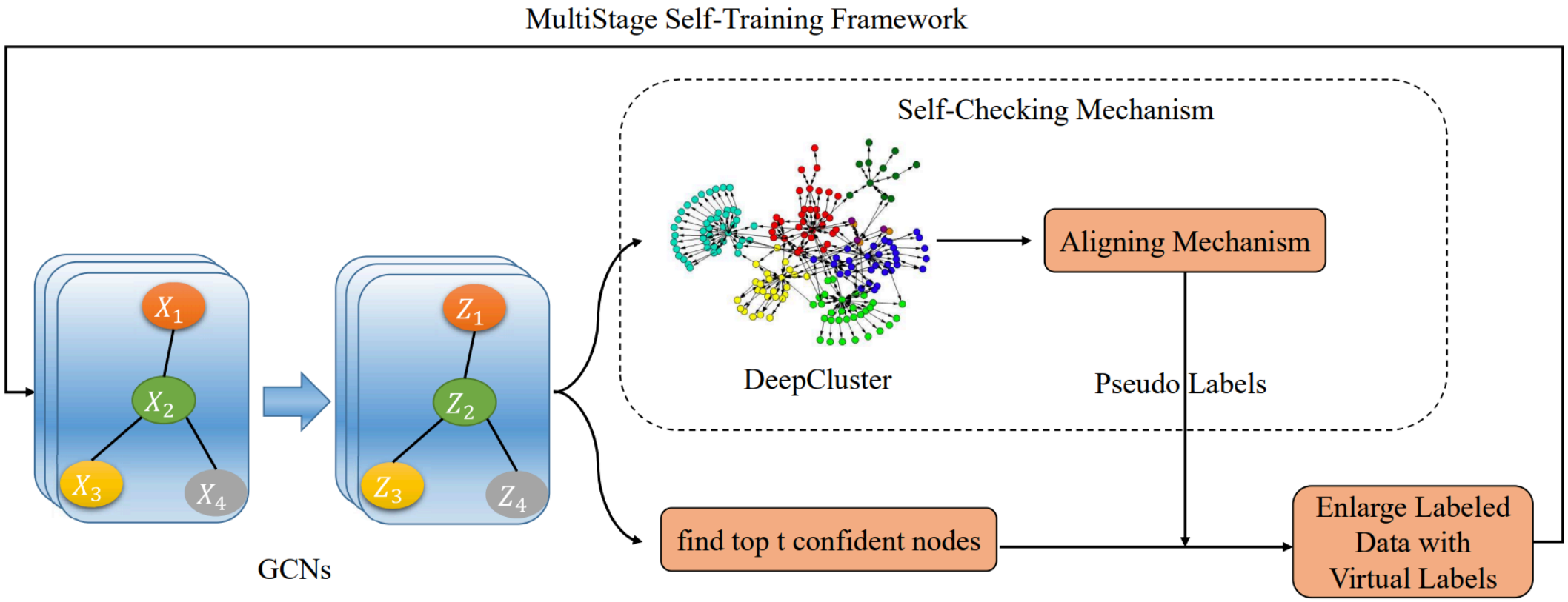


- **Two-stage training**



M3S

- **Idea**: Enlarge training dataset through **self-training**



Multi-stage self-supervised learning for graph convolutional networks on graphs with few labels, AAAI2020