

CS249: GRAPH NEURAL NETWORKS

Graph Basics

Instructor: Yizhou Sun

yzsun@cs.ucla.edu

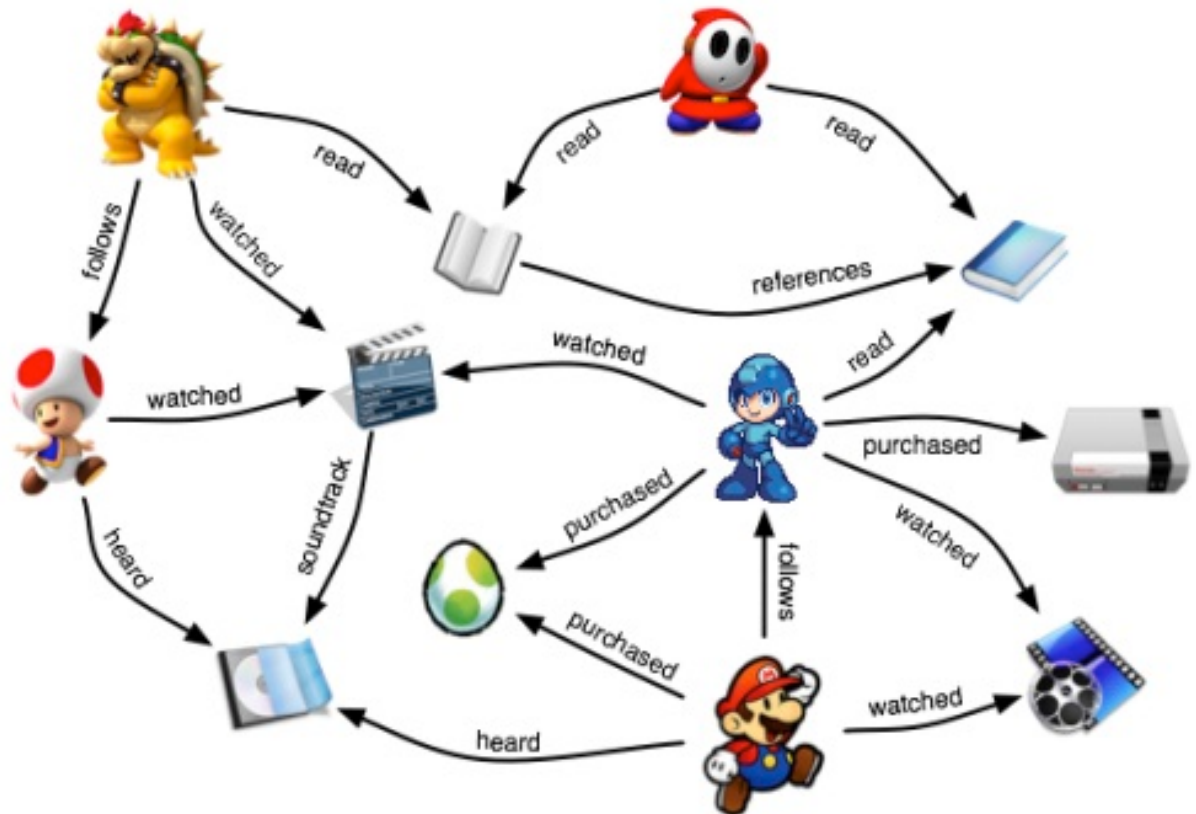
March 28, 2022

Content

- Introduction to Graphs 
- Spectral analysis
- Shallow Embedding
- Graph Neural Networks

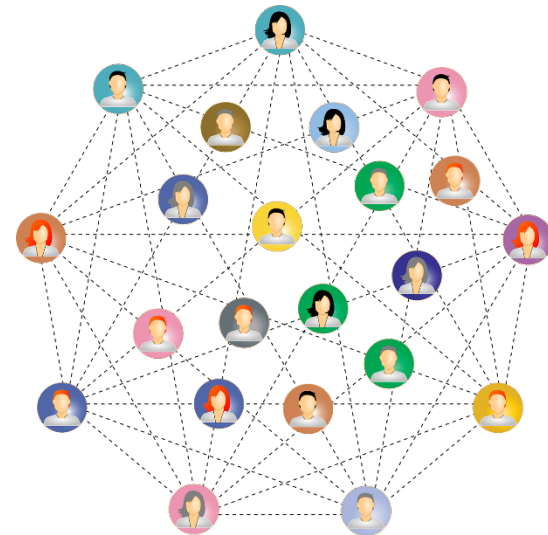
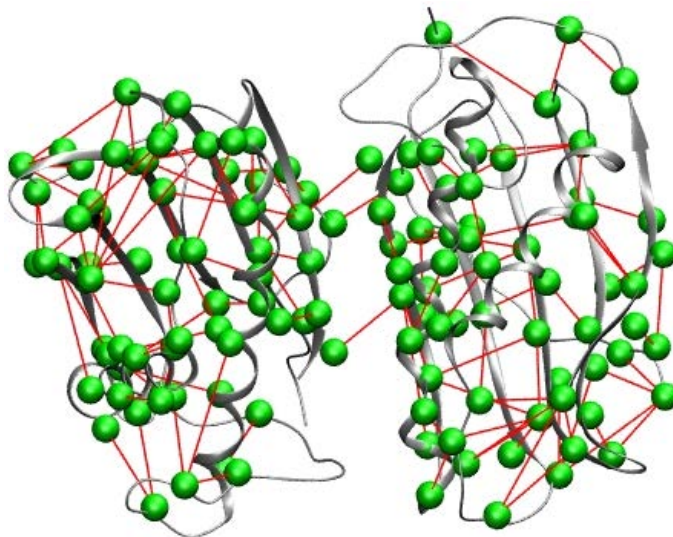
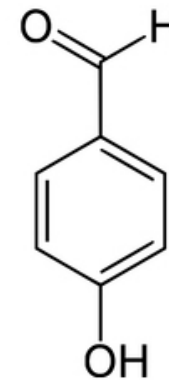
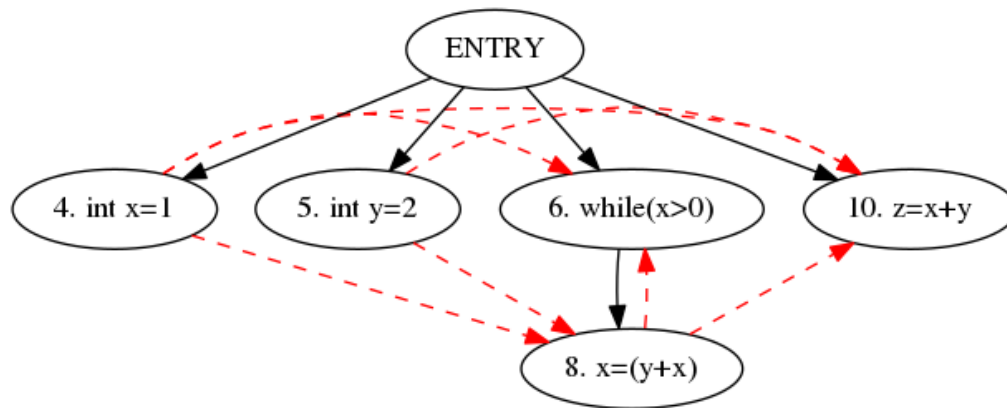
Why Graphs

- We are living in a connected world!
 - Entities
 - attributes
 - Relations
 - attributes



Graphs as modern infrastructure

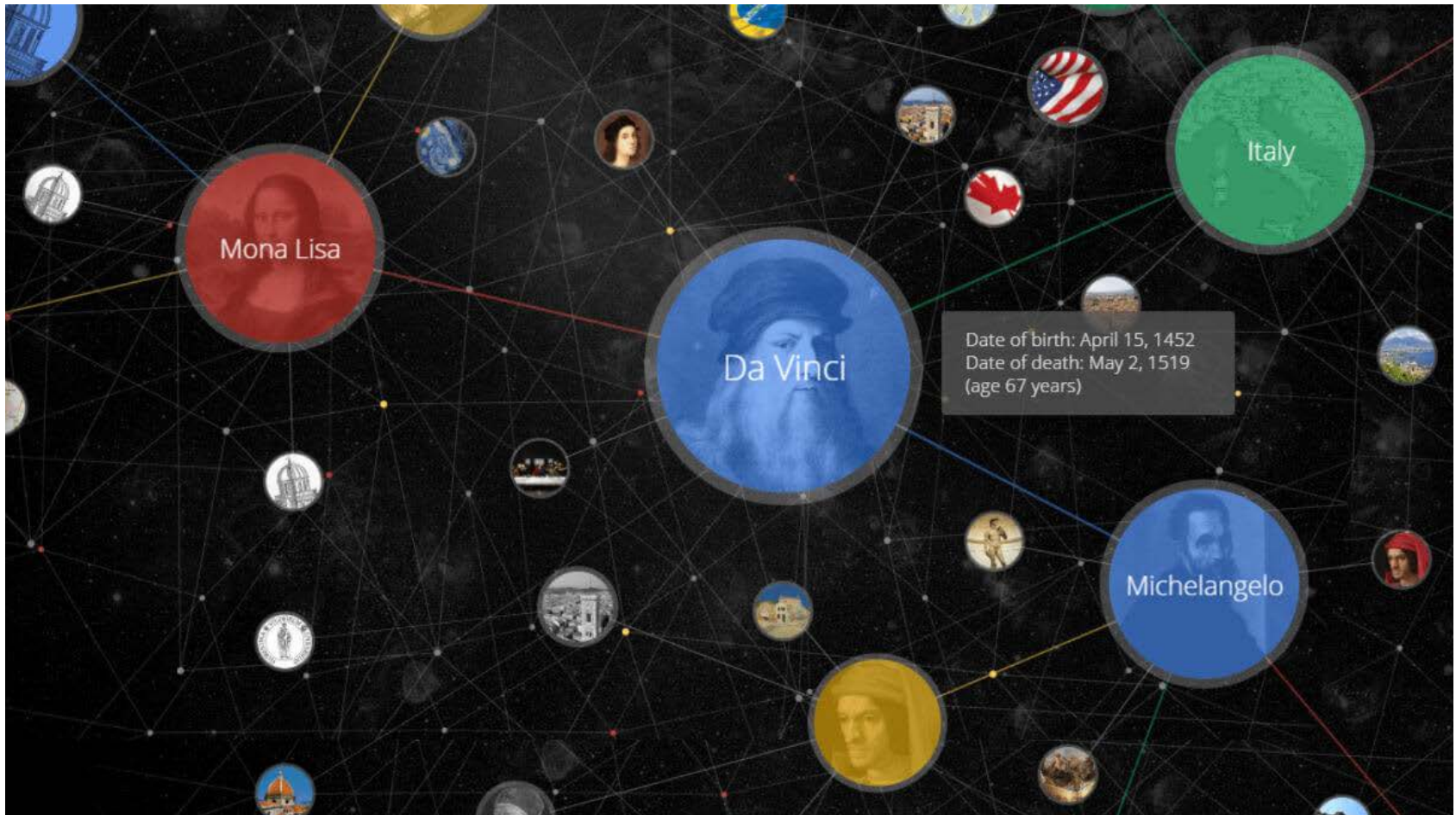
- Widely used in social network, e-commerce, healthcare, software engineer, hardware design, ...



Google's Knowledge Graph

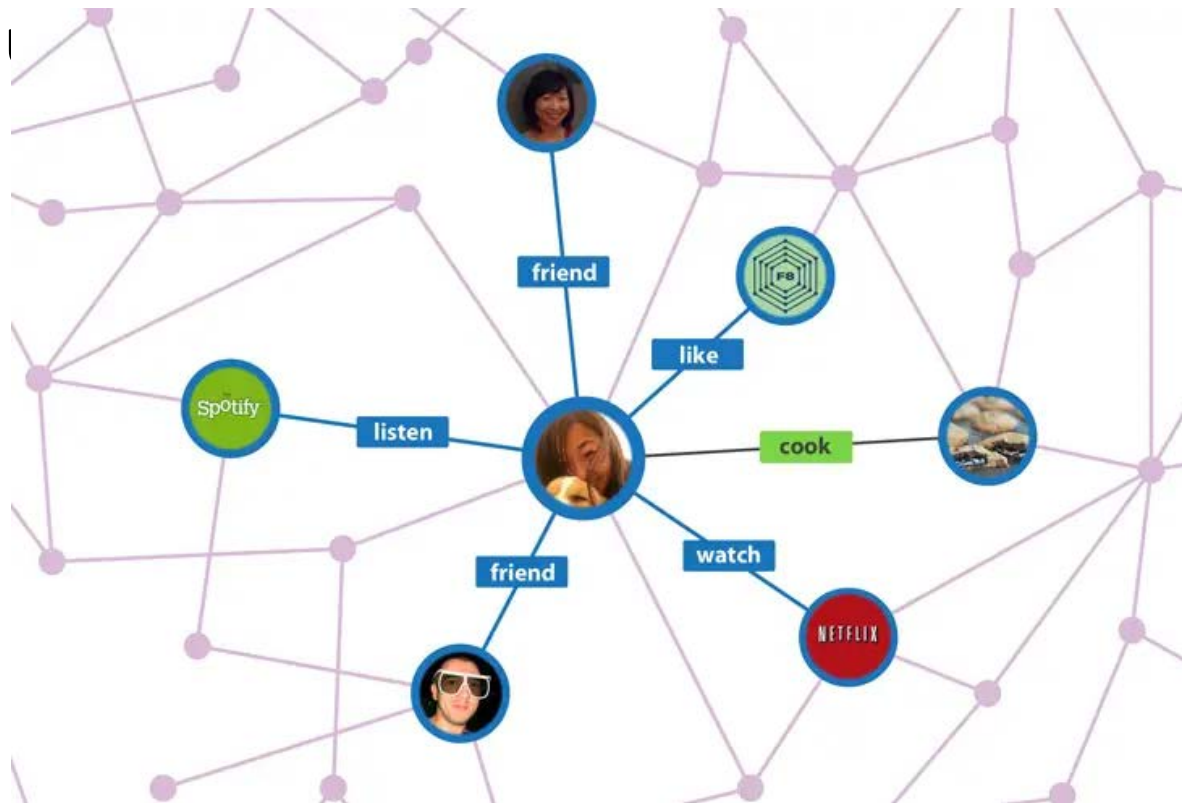
Dramatically enhanced search function:

- From web page search to knowledge search



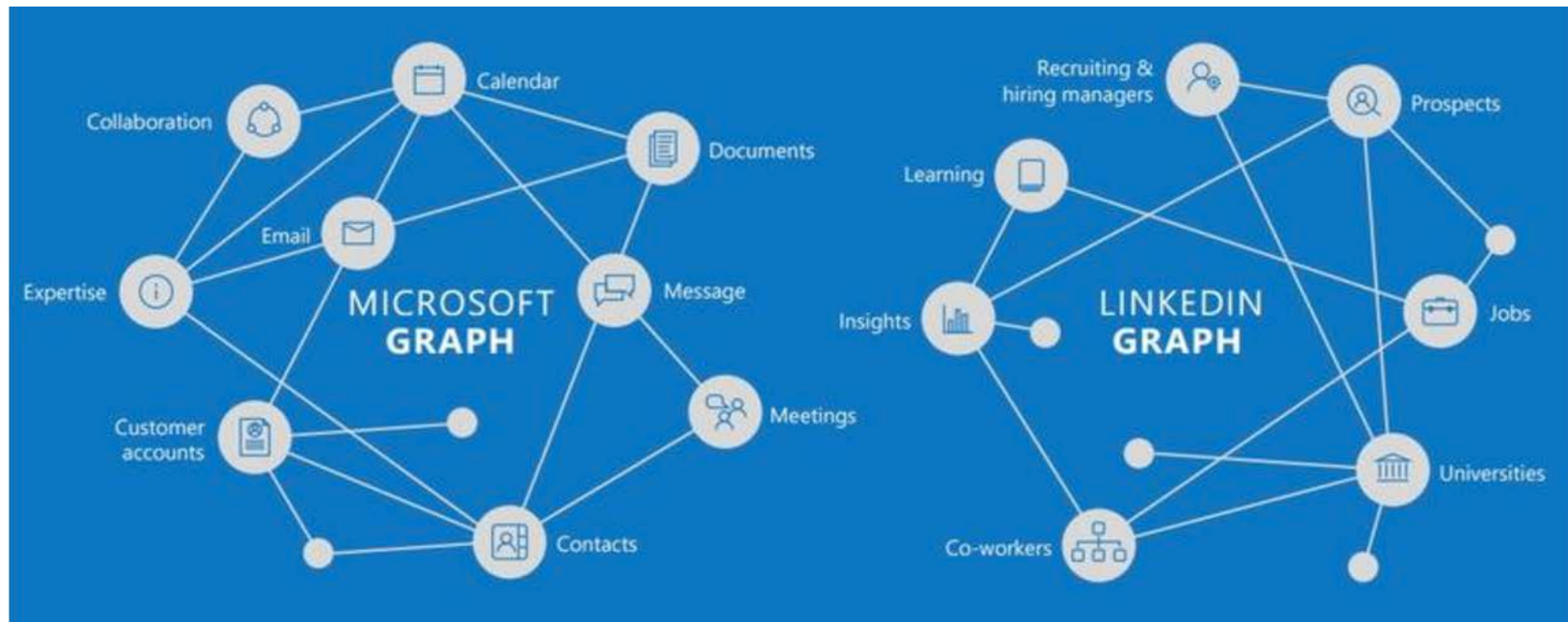
Facebook's Social/Open Graph

- Beyond social network
 - User and their interactions with other **objects**
 - Could be: articles, blogs, videos, photos, book, music



LinkedIn's Economic/Knowledge Graph

(by Oct. 2017)



Amazon's Product Graph

- Facilitate recommendation

TAXONOMY



USER LOGS



CATALOGUE

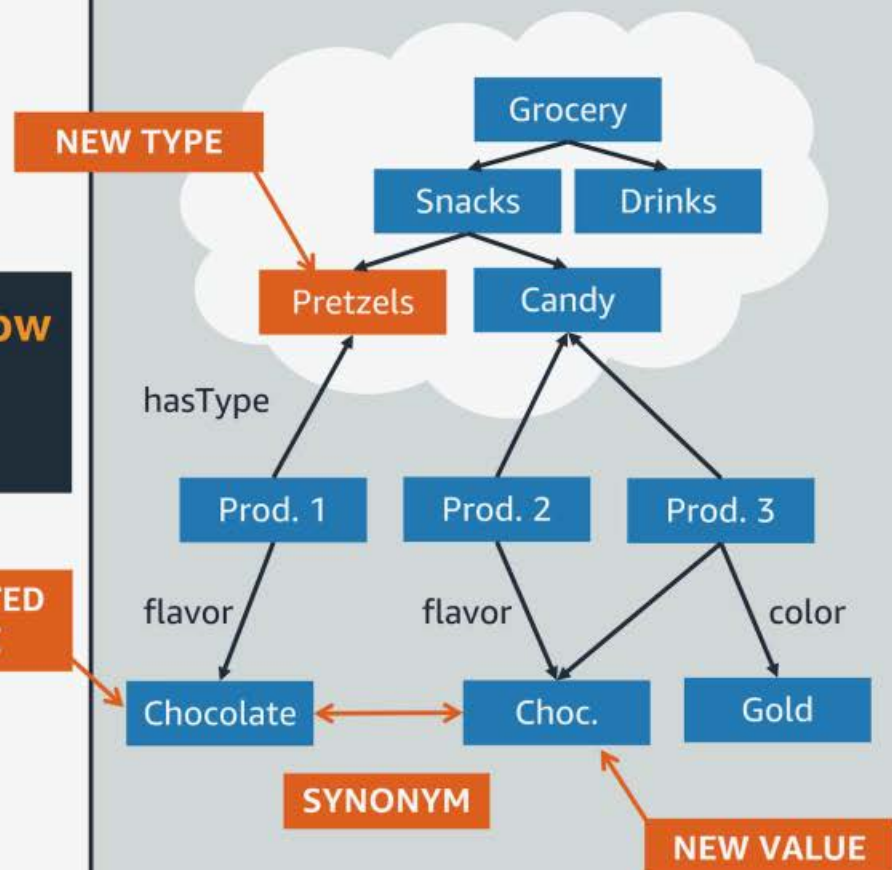
Product	Type	Flavor	Color
Product 1	Snacks	Cherry	
Product 2	Candy	?	?
Product 3	Candy	Choc.	Gold

AutoKnow

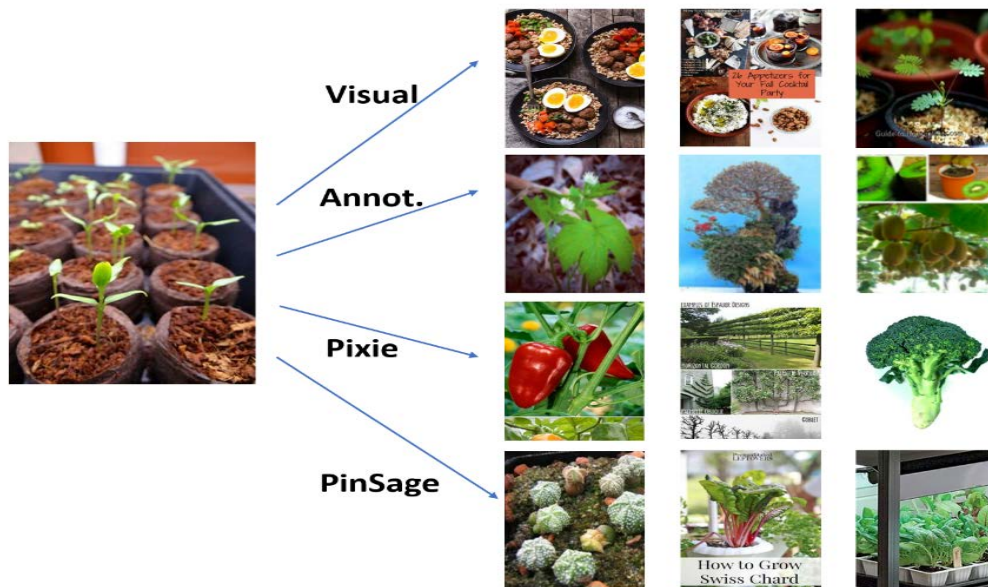
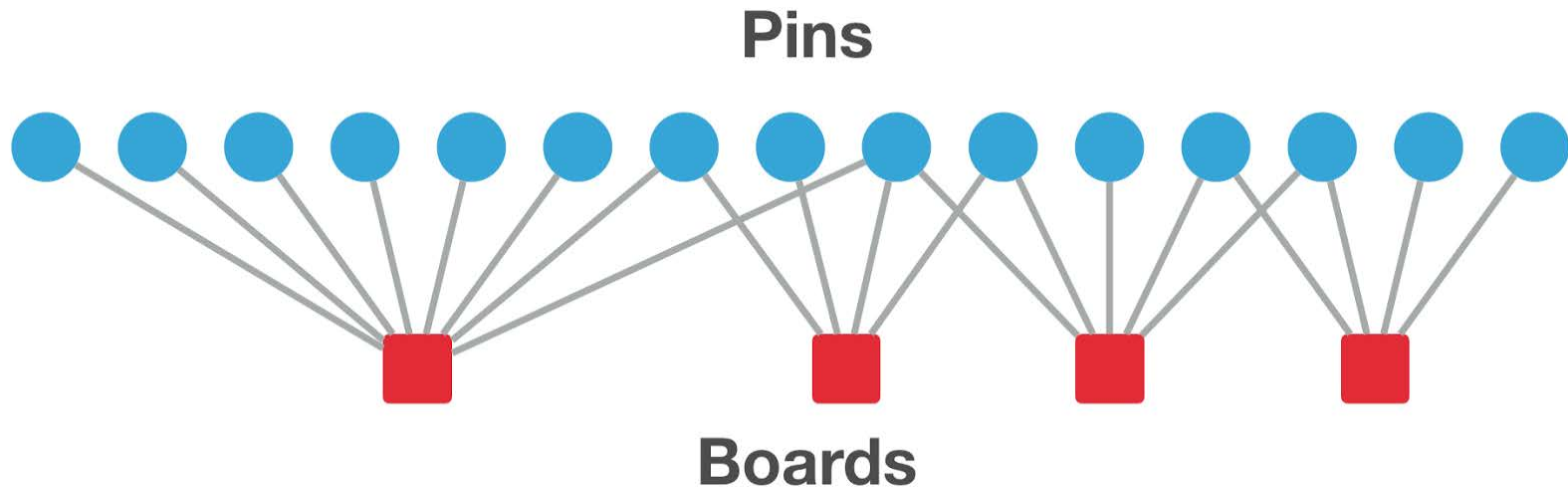


CORRECTED VALUE

PRODUCT KNOWLEDGE GRAPH



Pinterest

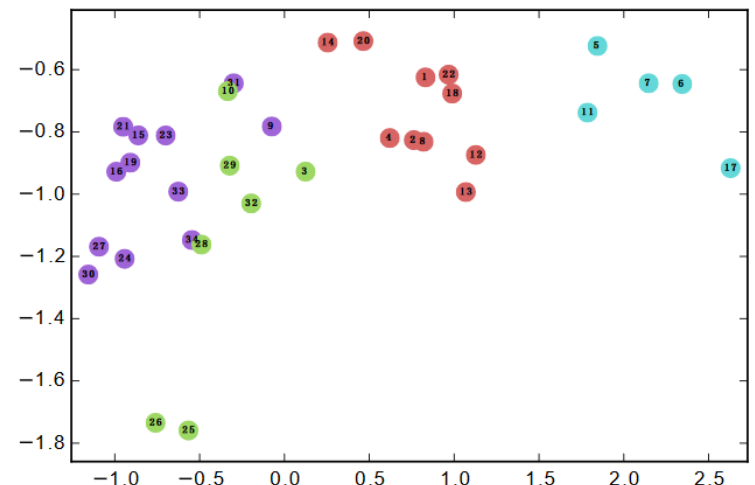
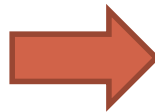
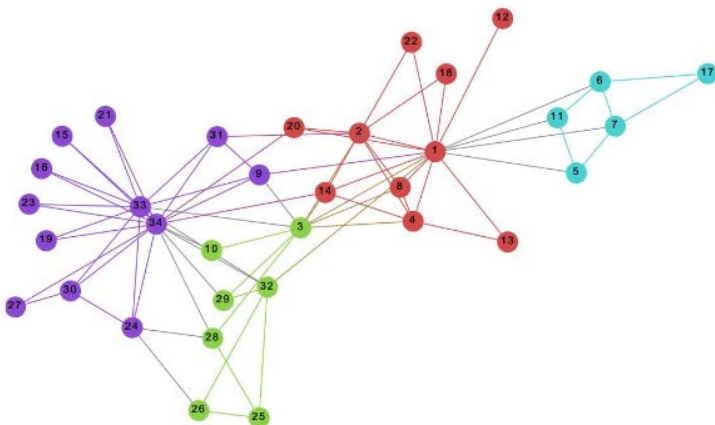
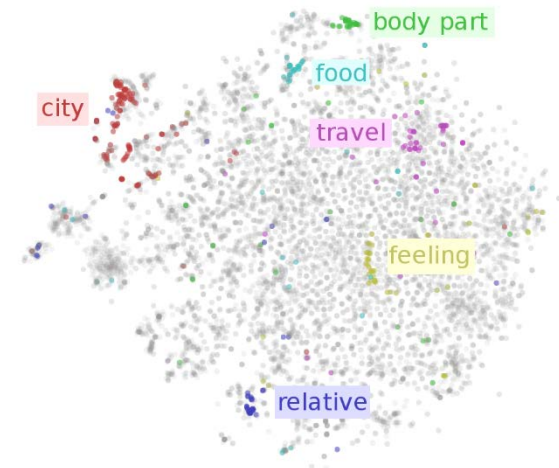


**Graph-based
Recommendation
(Pixie and PinSage)
better than baselines**

Why Representation Learning

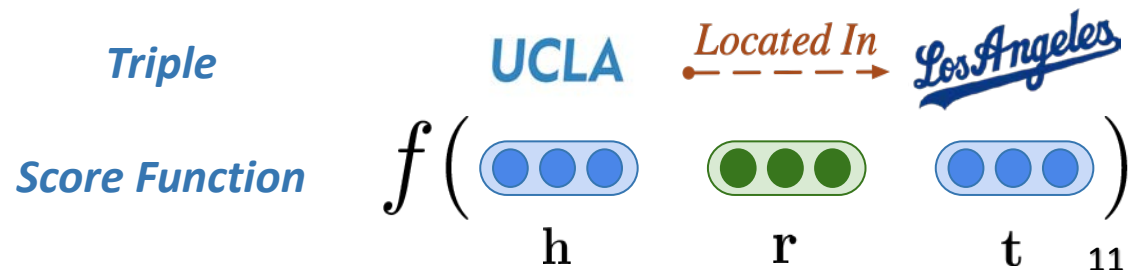
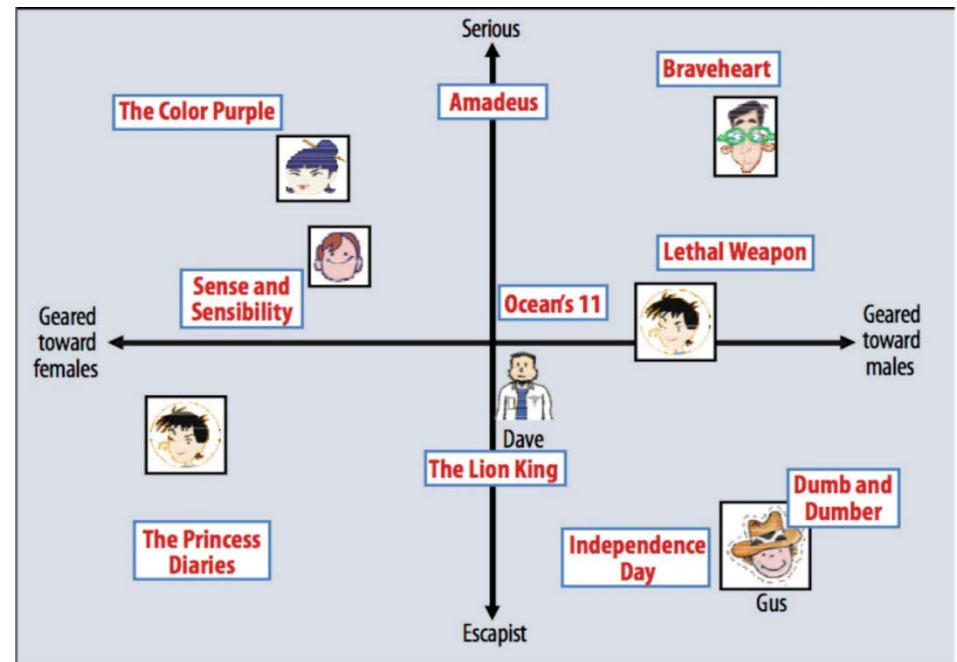
- Turn discrete symbols into continuous vectors

motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0



Examples of Graph Tasks – Node Level

- Node Classification
 - User profiling
- Link Prediction
 - Recommendation
 - Knowledge completion



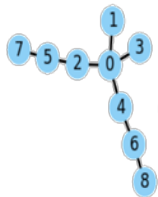
Examples of Graph Tasks – Graph Level

- Graph Classification

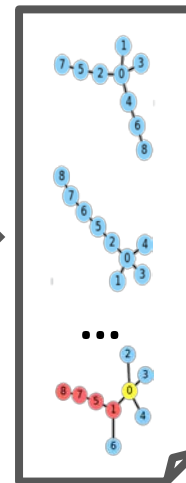
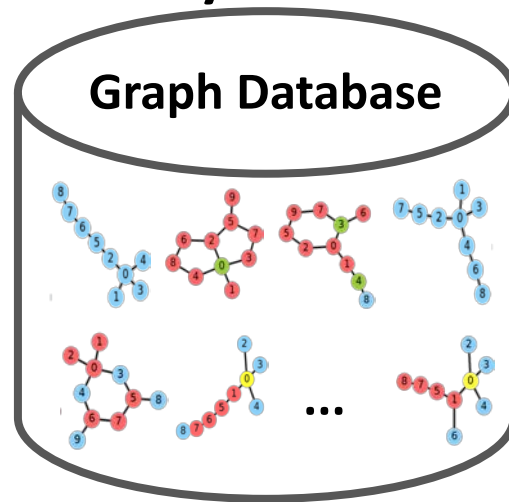
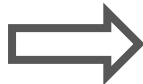


Alzheimer's disease?

- Graph similarity search



Query
Graph



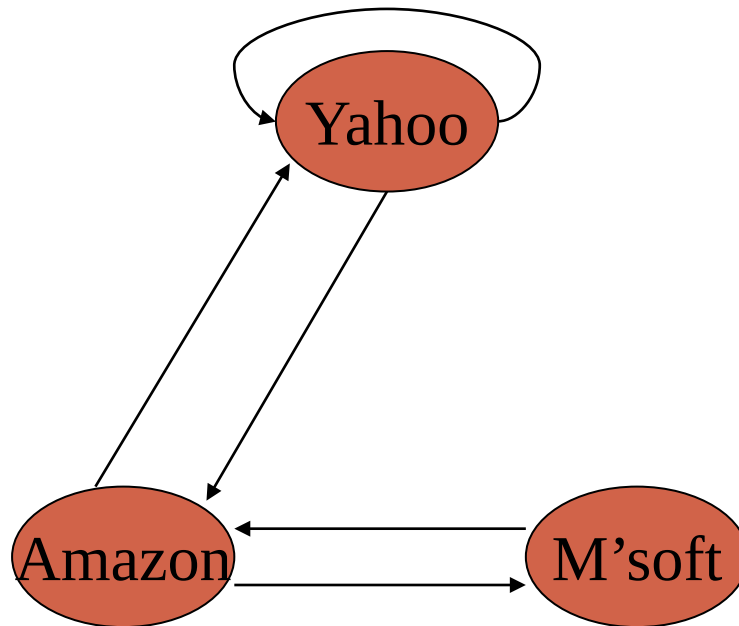
Descending
Similarity



Representation of a Graph

- $G = \langle V, E \rangle$
 - $V = \{u_1, \dots, u_n\}$: node set
 - $E \subseteq V \times V$: edge set
- Adjacency matrix
 - $A = \{a_{ij}\}, i, j = 1, \dots, N$
 - $a_{ij} = 1, \text{ if } \langle u_i, u_j \rangle \in E$
 - $a_{ij} = 0, \text{ if } \langle u_i, u_j \rangle \notin E$
 - Undirected graph vs. Directed graph
 - $A = A^T$ vs. $A \neq A^T$
 - Weighted graph
 - Use W instead of A , where w_{ij} represents the weight of edge $\langle u_i, u_j \rangle$

Example



	y	a	m
y	1	1	0
a	1	0	1
m	0	1	0

Adjacency matrix A

Typical Graph Tasks

- **Node level**

- **Link prediction**
- **Node classification**
- **Similarity search**
- **Community detection**
- **Ranking**

- **Graph level**

- **Graph Prediction**
- **Graph similarity search**
- **Frequent pattern mining**
- **MCS detection**
- **Clustering**

- **Node level, Graph Property**

- **Betweenness score prediction**
- **Travelling salesman problem**
- **Network attack problem**
- **Set cover problem**
- **Maximum clique detection**

Graph Techniques

- Approaches before GNNs
 - Heuristics
 - Graph signal processing
 - Graph Kernels
 - Graphical models

Content

- Introduction to Graphs
- Spectral analysis 
- Shallow Embedding
- Graph Neural Networks

Spectral Analysis

- Graph Laplacians are keys to understand graphs
 - Unnormalized graph Laplacians
 - Normalized graph Laplacians
- Spectral clustering
 - Leverage eigenvectors of graph Laplacians to conduct clustering on graphs
 - Has close relationship with graph cuts
- Label propagation
 - Semi-supervised node classification on graphs
 - Also related to graph Laplacians

What are Graph Laplacian Matrices?

- They are matrices defined as functions of graph adjacency or weight matrix
- A tool to study graphs
- There is a field called spectral graph theory studying those matrices

Examples of Graph Laplacians

- Given an undirected and weighted graph $G = (V, E)$, with weight matrix W
 - $W_{ij} = W_{ji} \geq 0$
 - n : total number of nodes
 - Degree for node $v_i \in V$: $d_i = \sum_j w_{ij}$
 - Degree matrix D : a diagonal matrix with degrees on the diagonal, i.e., $D_{ii} = d_i$ and $D_{ij} = 0$ if $i \neq j$
- Three examples of graph Laplacians
 - The unnormalized graph Laplacian
 - $L = D - W$
 - The normalized graph Laplacians
 - Symmetric: $L_{sym} = D^{-1/2} L D^{-1/2}$
 - Random walk related: $L_{rw} = D^{-1} L$

The Unnormalized Graph Laplacian

- Definition: $L = D - W$

- Properties of L

- For any vector $f \in R^n$

$$f' L f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2.$$

- L is symmetric and positive semi-definite
 - The smallest eigenvalue of L is 0, and the corresponding eigenvector is the constant one vector **1**
 - **1** is an all-one vector with n dimensions
 - An eigenvector can be scaled by multiplying a nonzero scalar α
 - L has n non-negative, real-valued eigenvalues:

$$0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$$

Question

- Will self loops in graph change L ?

Number of Connected Components

- The multiplicity k of the eigenvalue 0 of L equals the number of connected components
 - Consider $k = 1$, i.e., a connected graph, and f , the corresponding eigenvector for 0

$$0 = f' L f = \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2$$

- Every element of f has to be equal to each other

K connected components

- The graph can be represented as a block diagonal matrix, and so do matrix L

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_k \end{pmatrix}$$

- For each block L_i , it has eigenvalue 0 with corresponding constant one eigenvector
- For L , it has k eigenvalues with 0, with corresponding eigenvectors as indicator vectors
 - $\mathbf{1}_{A_i}$ is an indicator vector, with ones for nodes in A_i , i.e., the nodes in component i, and zeros elsewhere

The normalized graph Laplacians

- Symmetric:

- $L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$

- Random walk related:

- $L_{rw} = D^{-1} L = I - D^{-1} W$

Properties of L_{sym} and L_{rw}

- For any vector $f \in R^n$

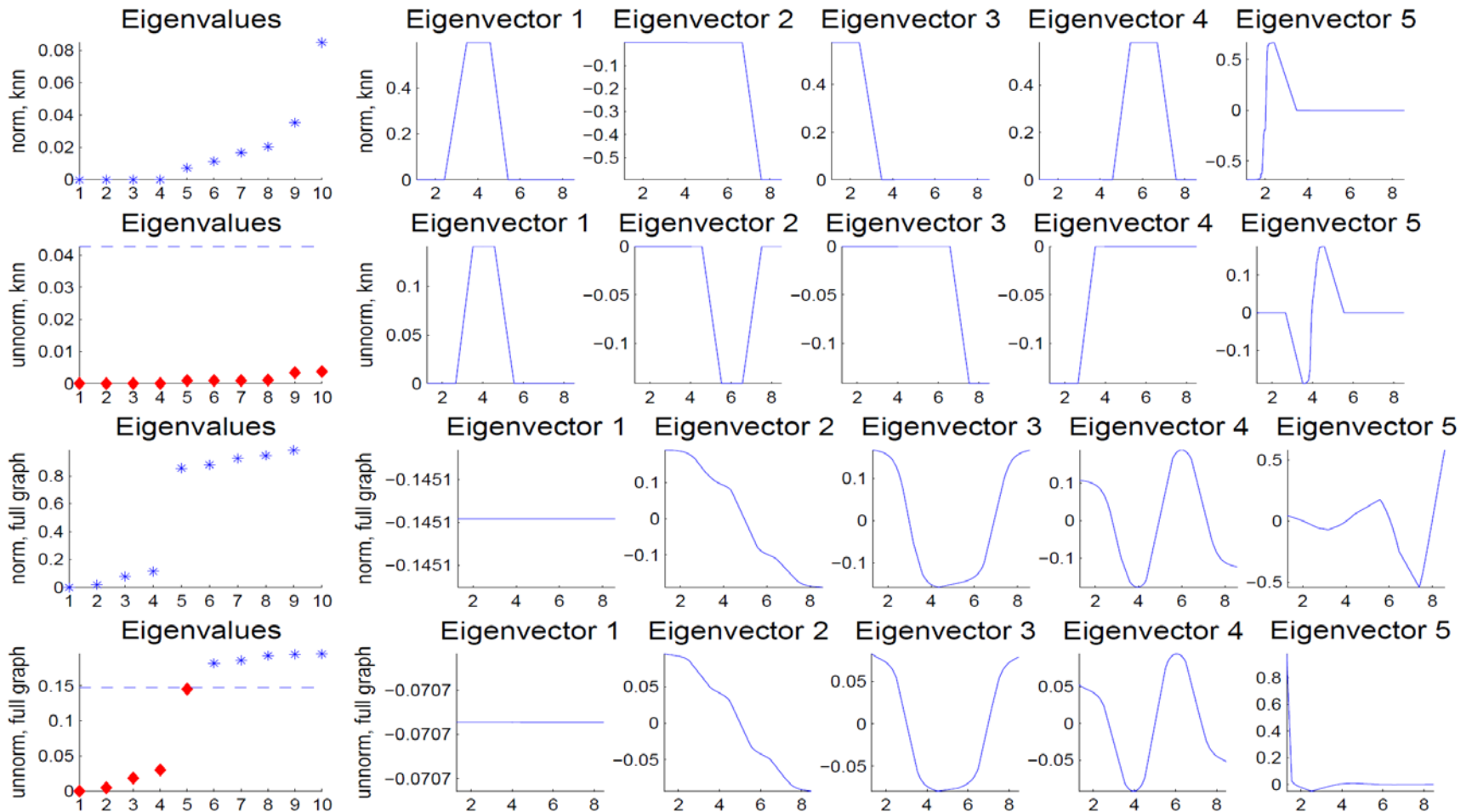
$$f' L_{sym} f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2$$

- λ is an eigenvalue of L_{rw} with eigenvector $v \Leftrightarrow \lambda$ is an eigenvalue of L_{sym} with eigenvector $w = D^{1/2}v$
- λ is an eigenvalue of L_{rw} with eigenvector $v \Leftrightarrow \lambda$ and v solves the generalized eigen problem $Lv = \lambda Dv$
- 0 is an eigenvalue of L_{rw} with eigenvector $\mathbf{1}$. 0 is an eigenvalue of L_{sym} with eigenvector $D^{1/2}\mathbf{1}$
- L_{sym} and L_{rw} are positive semi-definite, and have n non-negative real-valued eigenvalues

$$0 = \lambda_1 \leq \dots \leq \lambda_n$$

Example

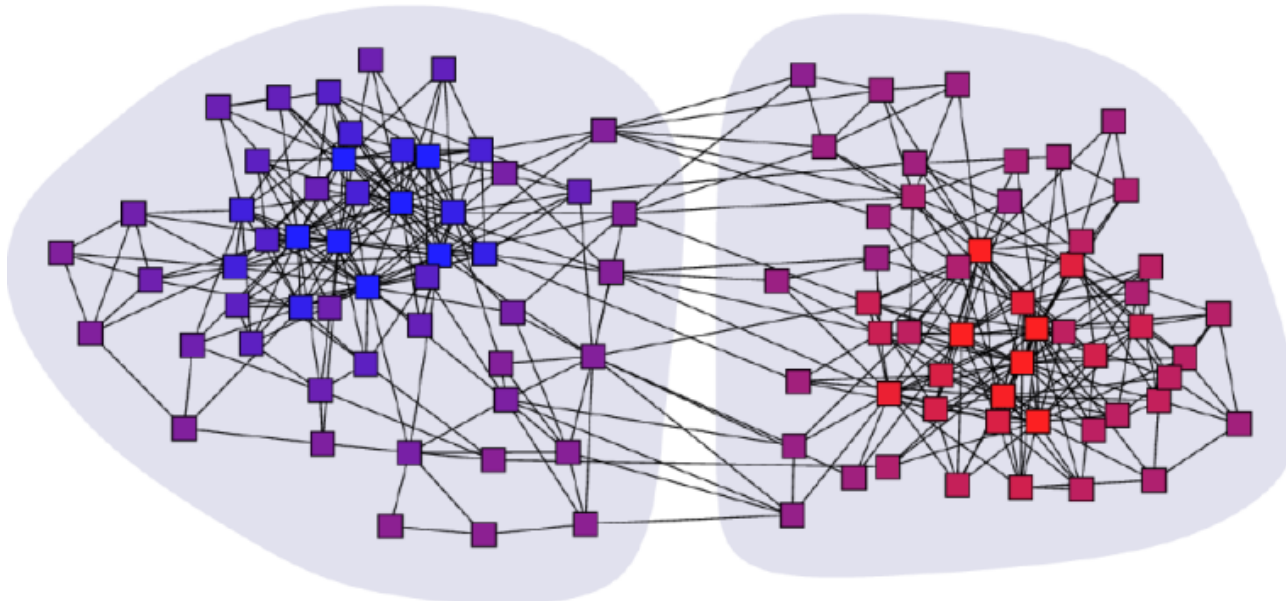
- Graph built upon Gaussian mixture model with four components



Spectral Clustering

Clustering Graphs and Network Data

- Applications
 - Bi-partite graphs, e.g., customers and products, authors and conferences
 - Web search engines, e.g., click through graphs and Web graphs
 - Social networks, friendship/coauthor graphs



Clustering books about politics [Newman, 2006]

Example of Graph Clustering

- Reference: ICDM'09 Tutorial by Chris Ding
- Example:
 - Clustering supreme court justices according to

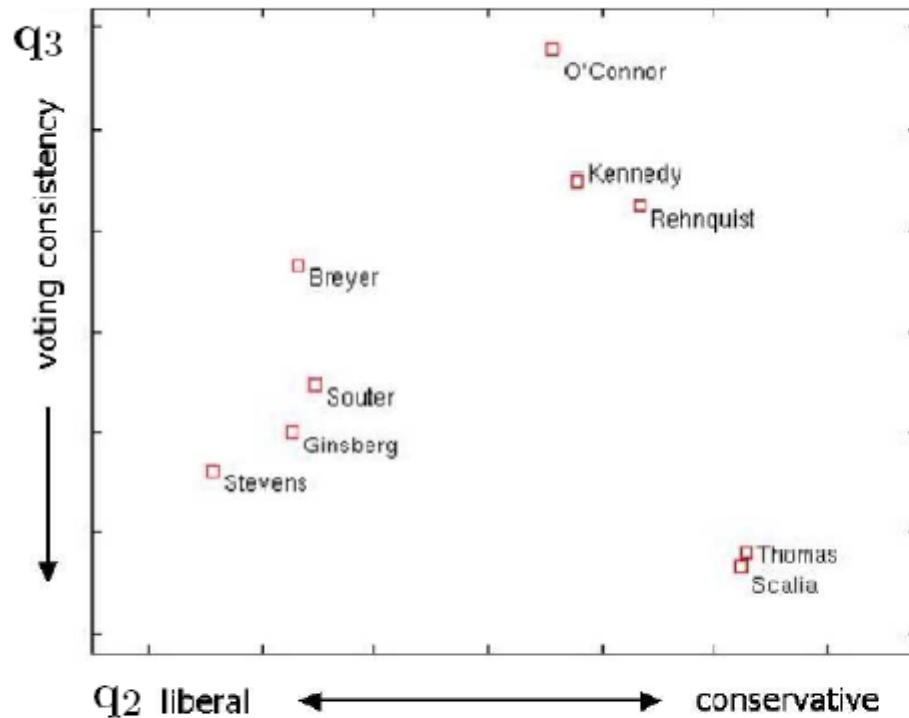
Number of times (%) two Justices voted in agreement

	Ste	Bre	Gin	Sou	O'Co	Ken	Reh	Sca	Tho
Stevens	–	62	66	63	33	36	25	14	15
Breyer	62	–	72	71	55	47	43	25	24
Ginsberg	66	72	–	78	47	49	43	28	26
Souter	63	71	78	–	55	50	44	31	29
O'Connor	33	55	47	55	–	67	71	54	54
Kennedy	36	47	49	50	67	–	77	58	59
Rehnquist	25	43	43	44	71	77	–	66	68
Scalia	14	25	28	31	54	58	66	–	79
Thomas	15	24	26	29	54	59	68	79	–

$W =$

Table 1: From the voting record of Justices 1995 Term – 2004 Term, the number of times two justices voted in agreement (in percentage). (Data source: from July 2, 2005 *New York Times*. Originally from *Legal Affairs; Harvard Law Review*)

Example: Continue



$$C = q_2 q_2^T + q_3 q_3^T$$

	Stevens	Breyer	Ginsberg	Souter	O'Connor	Kennedy	Rehnquist	Scalia	Thomas
Stevens	Green	Green	Green	Green	Red	Red	Red	Red	Red
Breyer	Green	Green	Green	Green	Green	Red	Red	Red	Red
Ginsberg	Green	Green	Green	Green	Red	Red	Red	Red	Red
Souter	Green	Green	Green	Green	Red	Red	Red	Red	Red
O'Connor	Red	Green	Red	Red	Green	Green	Green	Red	Red
Kennedy	Red	Red	Red	Red	Green	Green	Green	Red	Red
Rehnquist	Red	Red	Red	Red	Green	Green	Green	Red	Red
Scalia	Red	Red	Red	Red	Red	Red	Red	Green	Green
Thomas	Red	Red	Red	Red	Red	Red	Red	Green	Green

- Three groups in the Supreme Court:
 - Left leaning group, center-right group, right leaning group.

Spectral Clustering Algorithms

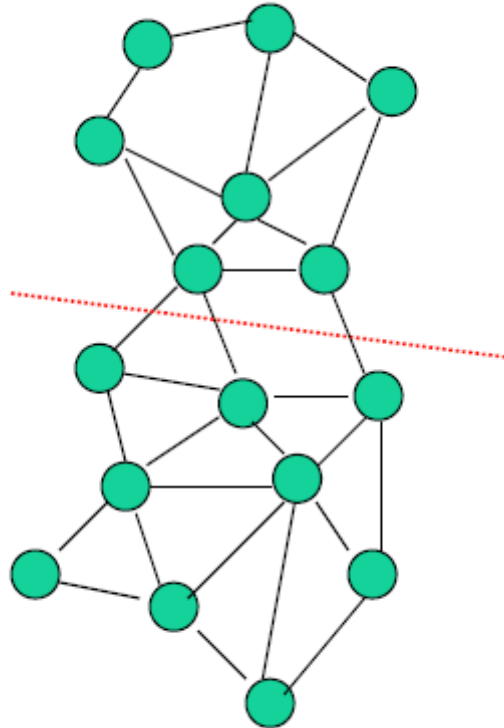
- Goal: cluster nodes in the graph into k clusters
- Idea: Leverage the first k eigenvectors of L
- Major steps:
 - Compute the first k eigenvectors, $v_1, v_2, \dots, v_k$, of L
 - Each node i is then represented by k values $x_i = (v_{1i}, v_{2i}, \dots, v_{ki})$
 - Cluster x_i 's using k -means

Variants of Spectral Clustering Algorithms

- Normalized spectral clustering according to Shi and Malik (2000)
 - Compute the first k eigenvectors, $v_1, v_2, \dots, v_k$, of L_{rw}
- Normalized spectral clustering according to Ng, Jordan, and Weiss (2002)
 - Compute the first k eigenvectors, $v_1, v_2, \dots, v_k$, of L_{sym}
 - Normalizing x_i to have norm 1

Connections to Graph Cuts

- Min-Cut
 - Minimize the # of cut of edges to partition a graph into 2 disconnected components



Objective Function of Min-Cut

2-way Spectral Graph Partitioning

Partition membership indicator: $q_i = \begin{cases} 1 & \text{if } i \in A \\ -1 & \text{if } i \in B \end{cases}$

$$\begin{aligned} J = \text{CutSize} &= \frac{1}{4} \sum_{i,j} w_{ij} [q_i - q_j]^2 \\ &= \frac{1}{4} \sum_{i,j} w_{ij} [q_i^2 + q_j^2 - 2q_i q_j] = \frac{1}{2} \sum_{i,j} q_i [d_i \delta_{ij} - w_{ij}] q_j \\ &= \frac{1}{2} q^T (D - W) q \end{aligned}$$

Relax indicators q_i from discrete values to continuous values,
the solution for $\min J(q)$ is given by the eigenvectors of

$$(D - W)q = \lambda q$$

(Fiedler, 1973, 1975)

(Pothen, Simon, Liou, 1990)

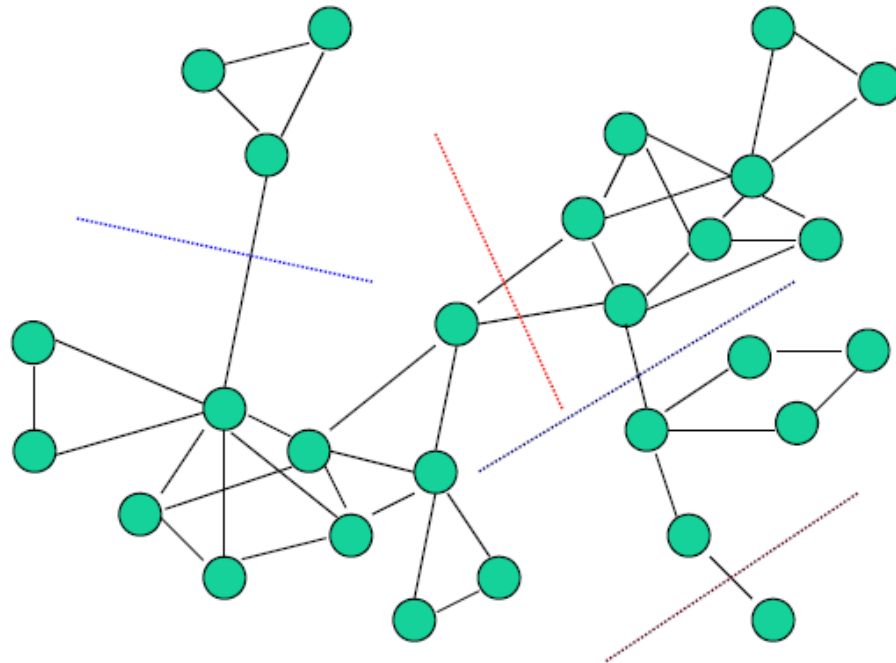
Algorithm

- Step 1:
 - Calculate Graph Laplacian matrix: $L = D - W$
- Step 2:
 - Calculate the **second** eigenvector q
 - Q: Why second?
- Step 3:
 - Bisect q (e.g., 0) to get two clusters

Minimum Cut with Constraints

minimize cutsize without explicit size constraints

But where to cut ?



Need to balance sizes

Other Objective Functions

- Ratio Cut (Hagen & Kahng, 1992)

$$s(A,B) = \sum_{i \in A} \sum_{j \in B} w_{ij}$$

$$J_{Rcut}(A,B) = \frac{s(A,B)}{|A|} + \frac{s(A,B)}{|B|}$$

- Normalized Cut (Shi & Malik, 2000)

$$d_A = \sum_{i \in A} d_i$$

$$J_{Ncut}(A,B) = \frac{s(A,B)}{d_A} + \frac{s(A,B)}{d_B}$$

$$= \frac{s(A,B)}{s(A,A) + s(A,B)} + \frac{s(A,B)}{s(B,B) + s(A,B)}$$

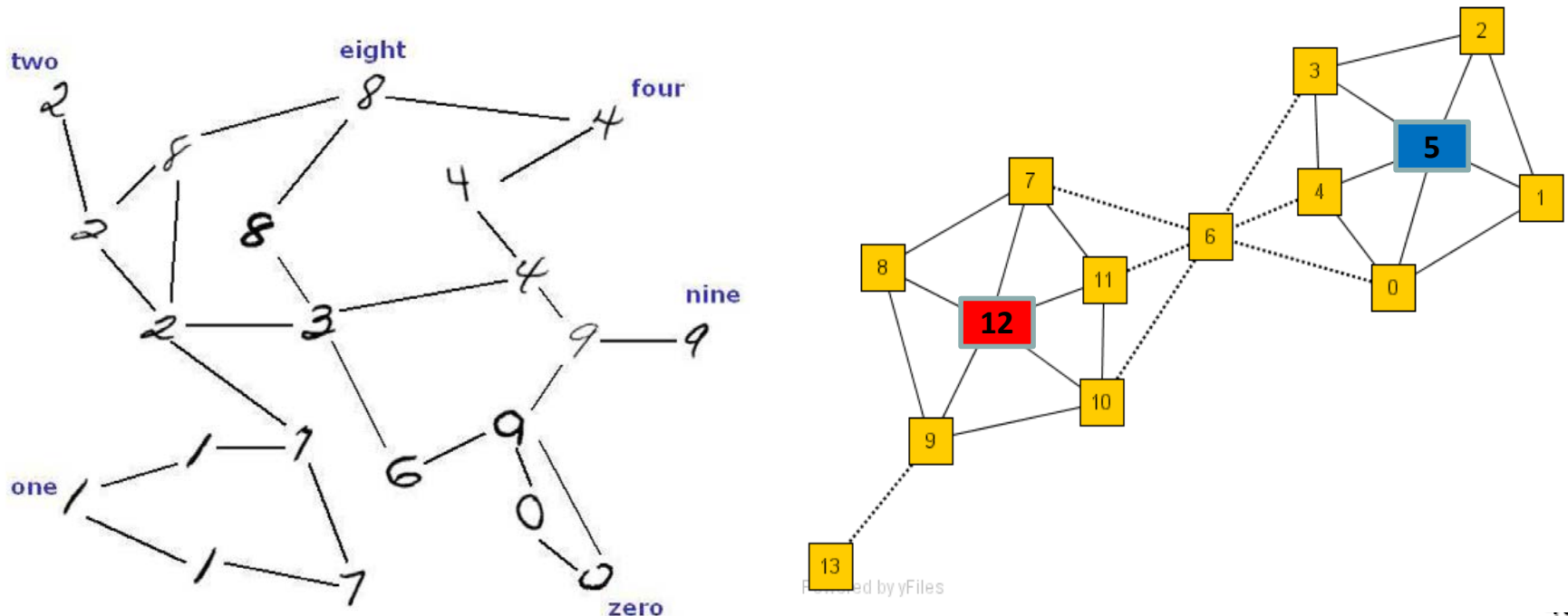
- Min-Max-Cut (Ding et al, 2001)

$$J_{MMC}(A,B) = \frac{s(A,B)}{s(A,A)} + \frac{s(A,B)}{s(B,B)}$$

Label Propagation

Label Propagation in the Network

- Given a network, some nodes are given labels, can we classify the unlabeled nodes by using link information?
- E.g., Node 12 belongs to **Class 1**, Node 5 Belongs to **Class 2**



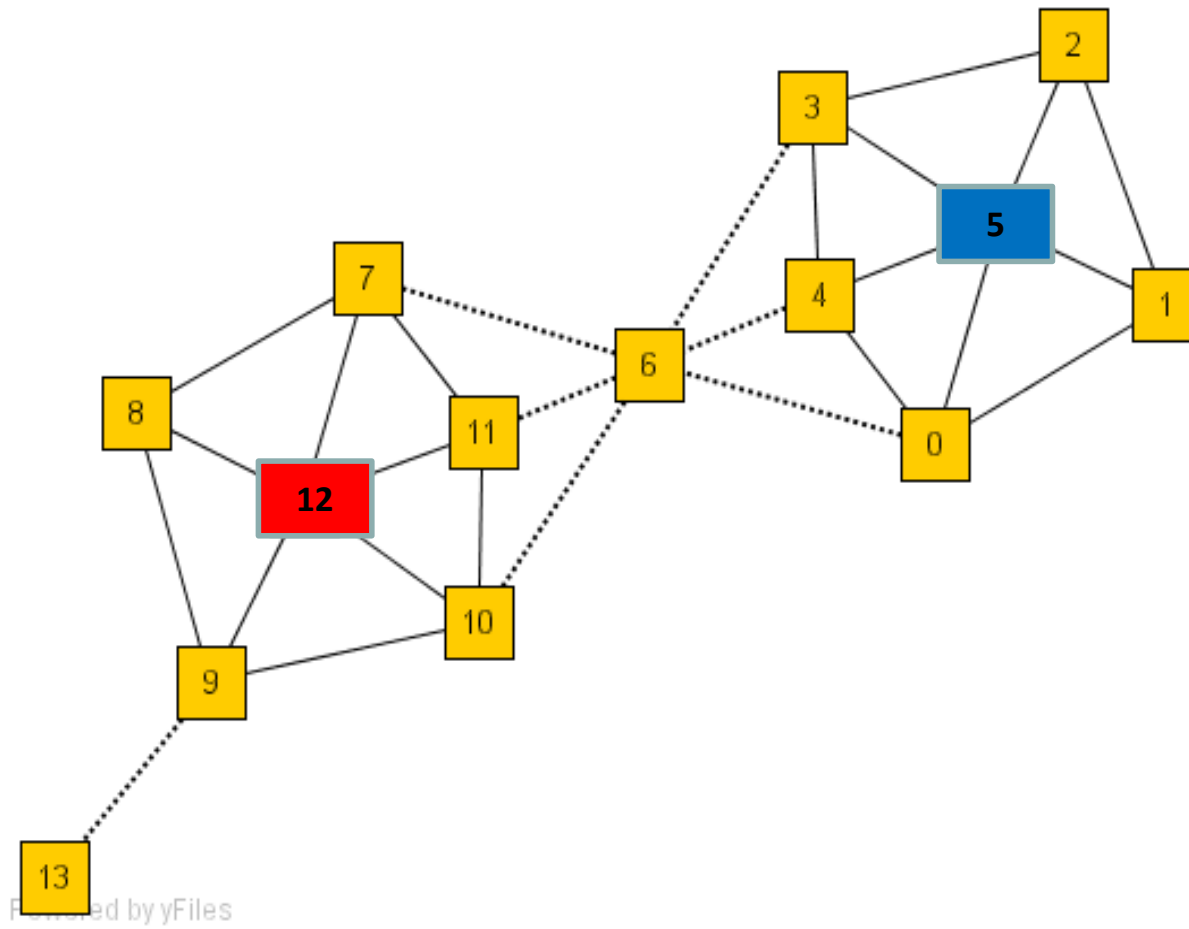
Problem Formalization for Label Propagation

- Given n nodes
 - l with labels (e.g., $Y_1, Y_2, \dots, Y_l$ are known)
 - u without labels (e.g., $Y_{l+1}, Y_{l+2}, \dots, Y_n$ are unknown)
 - Y is the $n \times K$ label matrix
 - K is the number of labels (classes)
 - Y_{ik} denotes the probability node i belonging to class k
- The weighted adjacency matrix is W
- The probabilistic transition matrix T
 - $T_{ij} = P(j \rightarrow i) = \frac{w_{ij}}{\sum_{i'} w_{i'j}}$

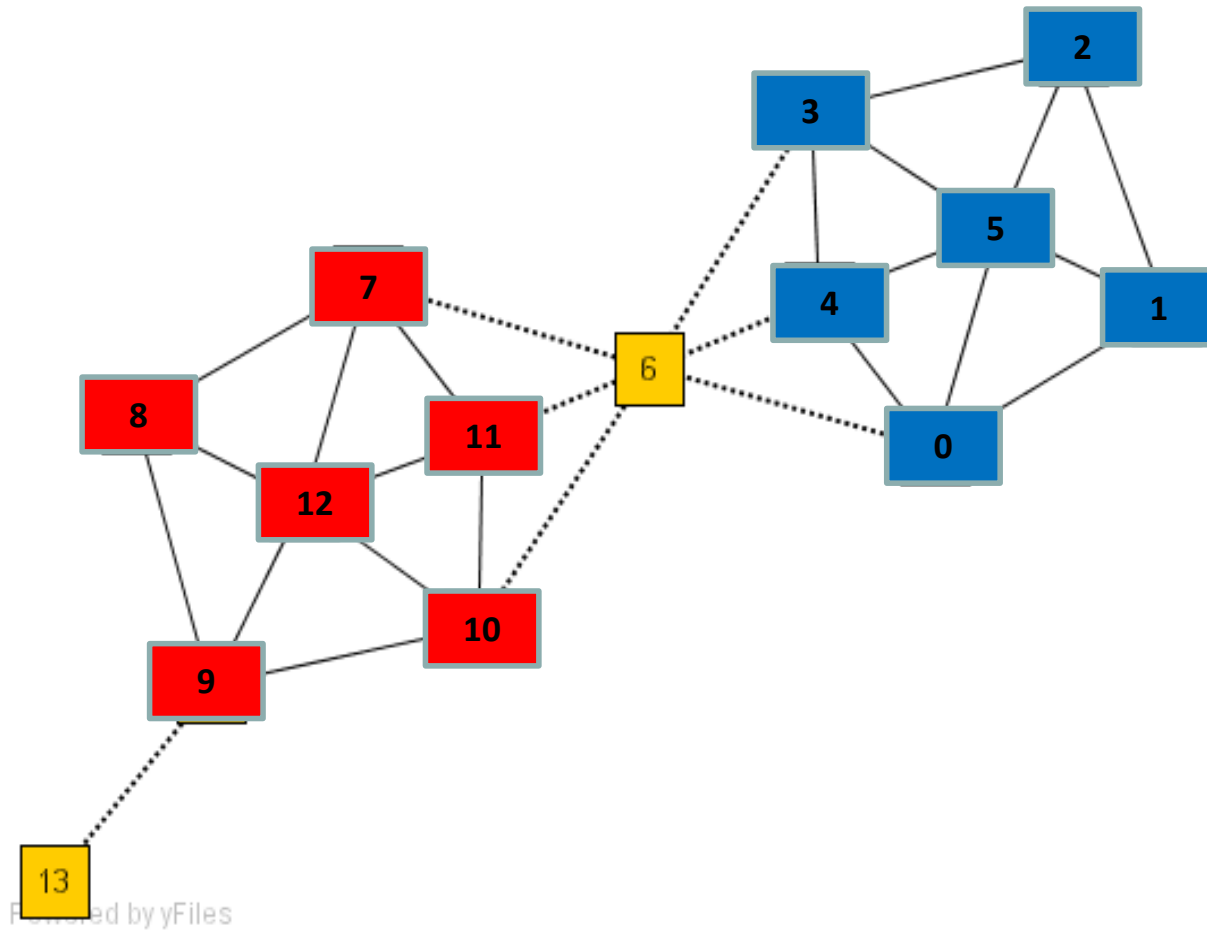
The Label Propagation Algorithm

- Step 1: Propagate $Y \leftarrow TY$
 - $Y_i = \sum_j T_{ij} Y_j = \sum_j P(j \rightarrow i) Y_j$
 - Initialization of Y for unlabeled ones is not important
- Step 2: Row-normalize Y
 - The summation of the probability of each object belonging to each class is 1
- Step 3: Reset the labels for the labeled nodes. Repeat 1-3 until Y converges

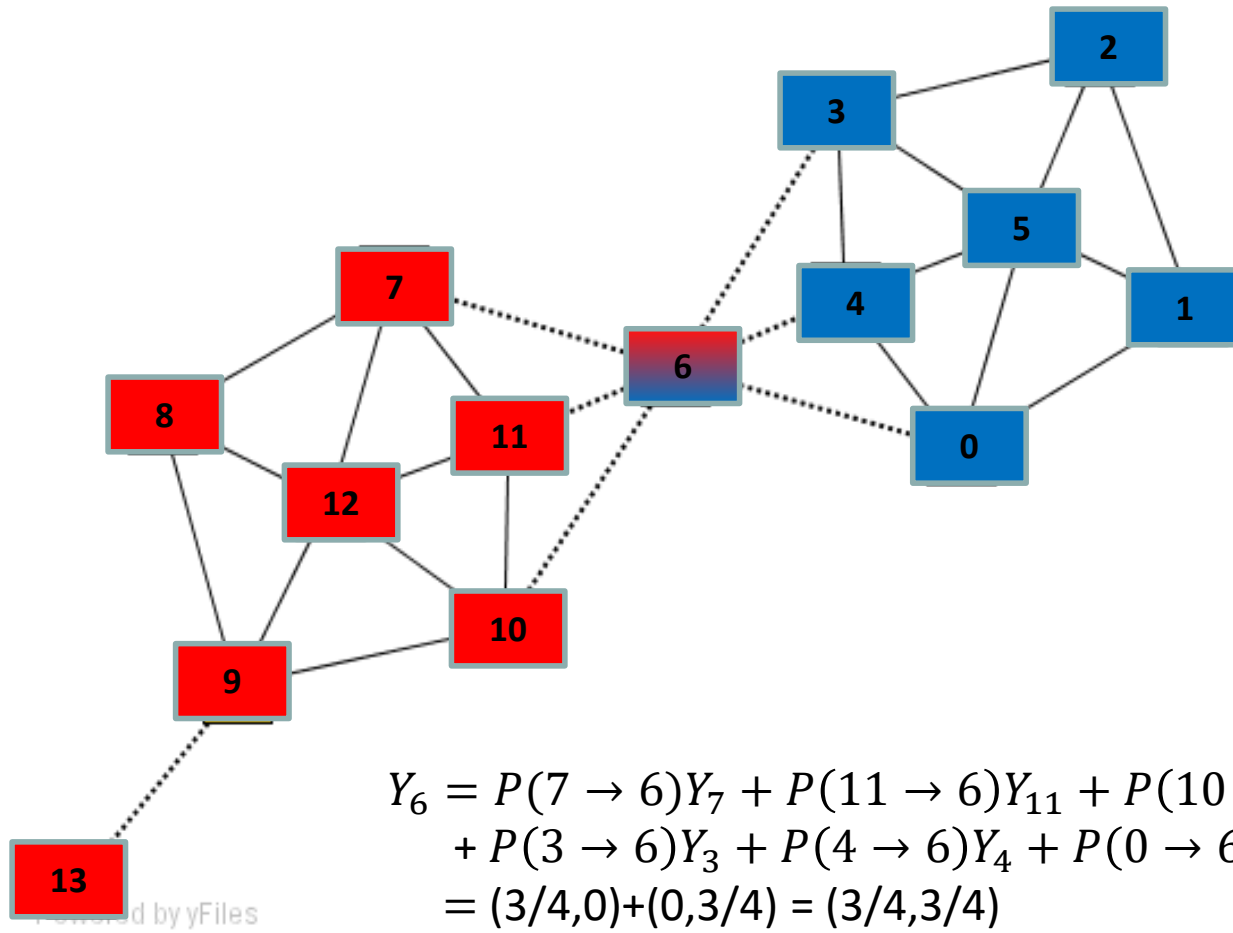
Example: Iter = 0



Example: Iter = 1



Example: Iter = 2



Other Label Propagation Algorithms

- Energy minimizing and harmonic function
 - Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions
 - By Xiaojin Zhu et al., ICML'03
 - <https://www.aaai.org/Papers/ICML/2003/ICML03-118.pdf>
- Graph regularization
 - Learning with Local and Global Consistency
 - By Denny Zhou et al., NIPS'03
 - <http://papers.nips.cc/paper/2506-learning-with-local-and-global-consistency.pdf>

From Energy Minimizing Perspective

- Consider a binary classification problem
 - $y \in \{0,1\}$
- Let $f: V \rightarrow R$, which maps a node to a real number
 - $f(i) = y_i$, if i is labeled
 - $f = \begin{pmatrix} f_l \\ f_u \end{pmatrix}$
- $\text{TI } E(f) = \frac{1}{2} \sum_{i,j} w_{ij} (f(i) - f(j))^2$
 - Intuition: if two nodes are connected, they should share similar labels

Minimizing Energy Function Results in Harmonic Function

- Note $E(f) = f'(D - W)f = f'Lf!$
- *Goal: find f such that $E(f)$ is minimized, and f_l is fixed*
- *Solution:*
 - $(D - W)f = 0$

Solve f_u

- Consider W as a block matrix

$$W = \begin{bmatrix} W_{ll} & W_{lu} \\ W_{ul} & W_{uu} \end{bmatrix}$$

- Closed form solution of f_u :

$$f_u = (D_{uu} - W_{uu})^{-1} W_{ul} f_l = (I - P_{uu})^{-1} P_{ul} f_l$$

- Where $P = D^{-1}W$
- Iterative solution of f_u :
 - $f_u = (\sum_{t=0} P_{uu}^t) P_{ul} f_l \Rightarrow f_u^t = P_{uu} f_u^{t-1} + P_{ul} f_l$
 - As $(I - P_{uu})^{-1} = I + P_{uu} + P_{uu}^2 + \dots$

From Graph Regularization Perspective

- Let F be $n \times K$ matrix with nonnegative entries
 - For node i , pick the label k such that F_{ik} is the maximum on among all the k
- Let Y be $n \times K$ matrix with $y_{ik} = 1$, if node i is labeled as class k , and 0 otherwise.
- Cost function:
- **Smoothness constraint + fitting constraint**

$$\mathcal{Q}(F) = \frac{1}{2} \left(\sum_{i,j=1}^n W_{ij} \left\| \frac{1}{\sqrt{D_{ii}}} F_i - \frac{1}{\sqrt{D_{jj}}} F_j \right\|^2 + \mu \sum_{i=1}^n \|F_i - Y_i\|^2 \right)$$

Solve F

- Note the first component is related to $\text{trace}(F' L_{sym} F)$
- Closed form solution:
 - $\left. \frac{\partial Q}{\partial F} \right|_{F=F^*} = F^* - SF^* + \mu(F^* - Y) = 0$
 - $\Rightarrow F^* = (1 - \alpha)(I - \alpha S)^{-1}Y$
 - Where $S = D^{-1/2}WD^{-1/2}$, $\alpha = \frac{1}{1+\mu}$ is a value between 0 and 1
- Iterative solution:
 - $F(t+1) = \alpha SF(t) + (1 - \alpha)Y$

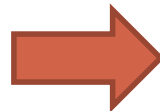
References

- A Tutorial on Spectral Clustering by U. Luxburg
http://www.kyb.mpg.de/fileadmin/user_upload/files/publications/attachments/Luxburg07_tutorial_4488%5B0%5D.pdf
- Learning from Labeled and Unlabeled Data with Label Propagation
 - By Xiaojin Zhu and Zoubin Ghahramani
 - <http://www.cs.cmu.edu/~zhuxj/pub/CMU-CALD-02-107.pdf>
- Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions
 - By Xiaojin Zhu et al., ICML'03
 - <https://www.aaai.org/Papers/ICML/2003/ICML03-118.pdf>
- Learning with Local and Global Consistency
 - By Denny Zhou et al., NIPS'03
 - <http://papers.nips.cc/paper/2506-learning-with-local-and-global-consistency.pdf>

Content

- Introduction to Graphs
- Spectral analysis
- Shallow Embedding 
- Graph Neural Networks

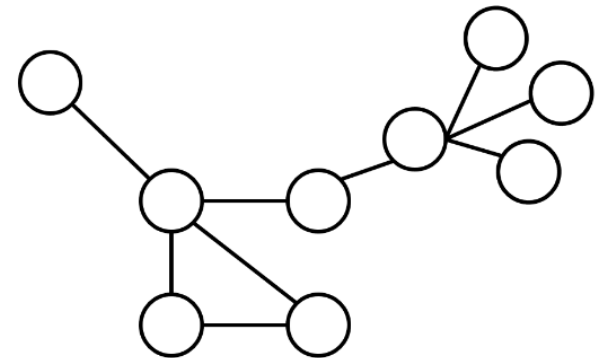
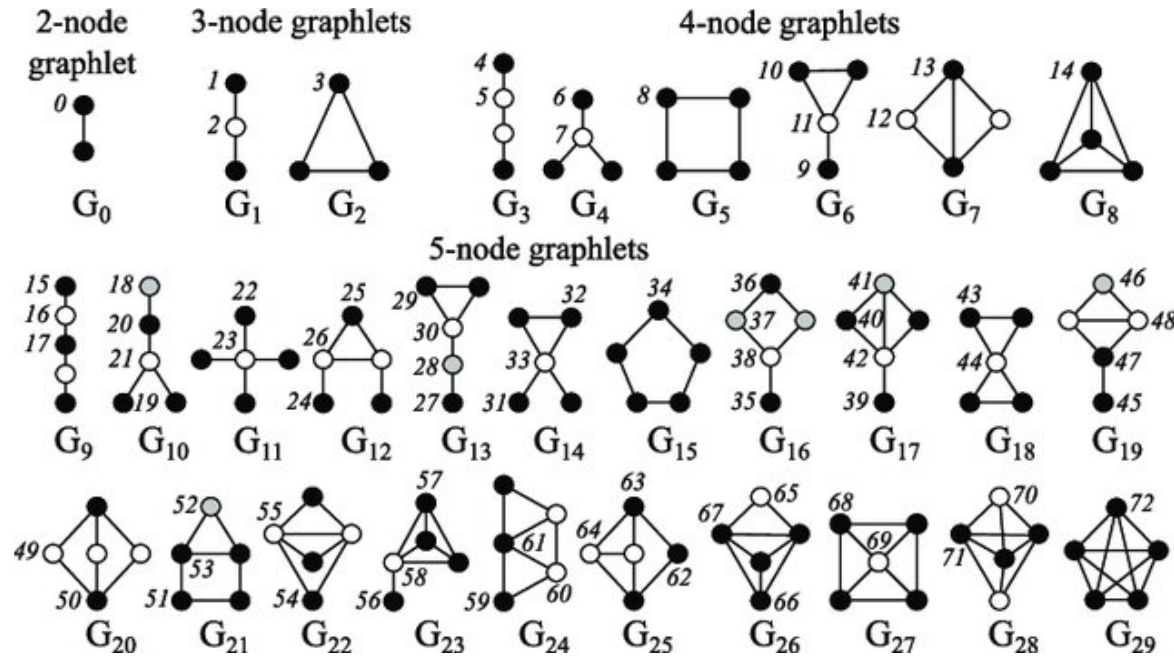
- Important for many graph related tasks
- Discrete nature makes it very challenging
- Naïve solutions



Permutation-variant

Even more challenging for graph representation

- Ex. Graphlet-based feature vector



Source: DOI: [10.1093/bioinformatics/btv130](https://doi.org/10.1093/bioinformatics/btv130)

						...
12	1	4	1	6	0	...

Requires subgraph isomorphism test: NP-hard

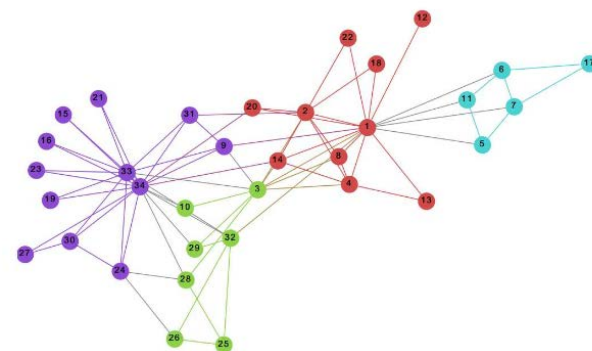
Source:

https://haotang1995.github.io/projects/robust_graph_level_representation_learning_using_graph_based_structural_attentional_learning55

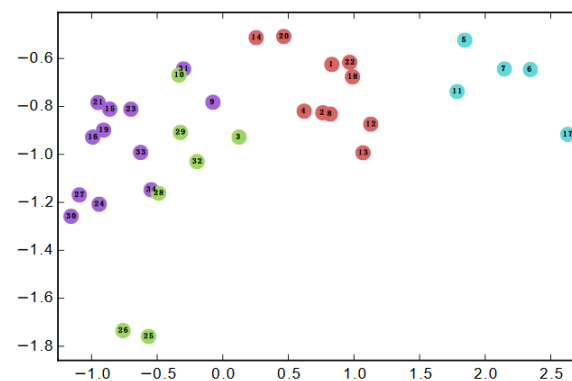
Automatic representation Learning

- Map each node/graph into a low dimensional vector
 - $\phi: V \rightarrow R^d$ or $\phi: \mathcal{G} \rightarrow R^d$
- Earlier methods
 - Shallow node embedding methods inspired by word2vec
 - DeepWalk [Perozzi, KDD'14]
 - LINE [Tang, WWW'15]
 - Node2Vec [Grover, KDD'16]

$\phi(v) = U^T x_v$, where U is the embedding matrix and x_v is the one-hot encoding vector



(a) Input: karate network



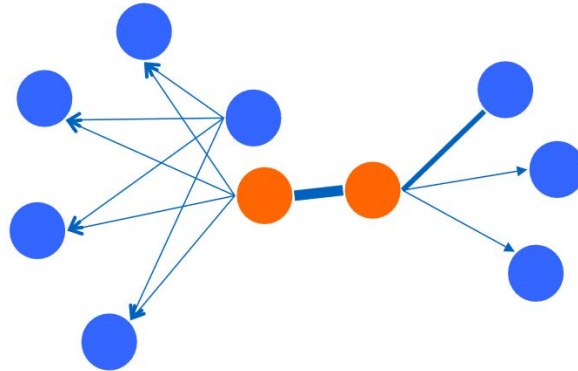
(b) Output: representations

Source: DeepWalk

LINE: Large-scale Information Network Embedding

Embedding

- First-order proximity



- Assumption: Two nodes are similar if they are connected

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

u_i : embedding vector for node i

- Limitation: links are sparse, not sufficient

Objective function for first-order proximity

- Minimize the KL divergence between empirical link distribution and modeled link distribution

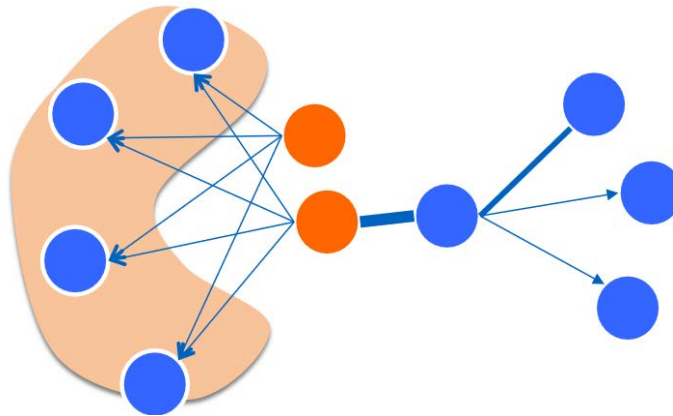
$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{(m,n) \in E} w_{mn}}$$

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

w_{ij} : weight over edge(i, j)

Second-Order Proximity

- Assumption:
 - Two nodes are similar if their neighbors are similar



$$p_2(v_j | v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)}$$

u_i : target embedding vector for node i

u_j' : context embedding vector for node j

Objective function for second-order proximity

- Minimize the KL divergence between empirical link distribution and modeled link distribution

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

- Objective function

$$O_2 = \sum_i d_i KL(\hat{p}_2(\cdot | v_i), p_2(\cdot | v_i)) = - \sum_{(i,j) \in E} w_{ij} \log p_2(v_j | v_i)$$
$$d_i = \sum_k w_{ik}$$

Negative Sampling for Optimization

- For second-order proximity derived objective function
 - For each positive link (i, j), sample K negative links (i, n)
 - An edge with weight w can be considered as w binary edges

$$\log \sigma(\vec{u}_j'^T \cdot \vec{u}_i) + \sum_{i=1}^K E_{v_n \sim P_n(v)} [\log \sigma(-\vec{u}_n'^T \cdot \vec{u}_i)]$$

negative distribution: $P_n(v) \propto d_v^{3/4}$

Limitation of shallow embedding techniques

- Too many parameters
 - Each node is associated with an embedding vector, which are parameters
- Not inductive
 - Cannot handle new nodes
- Cannot handle node attributes

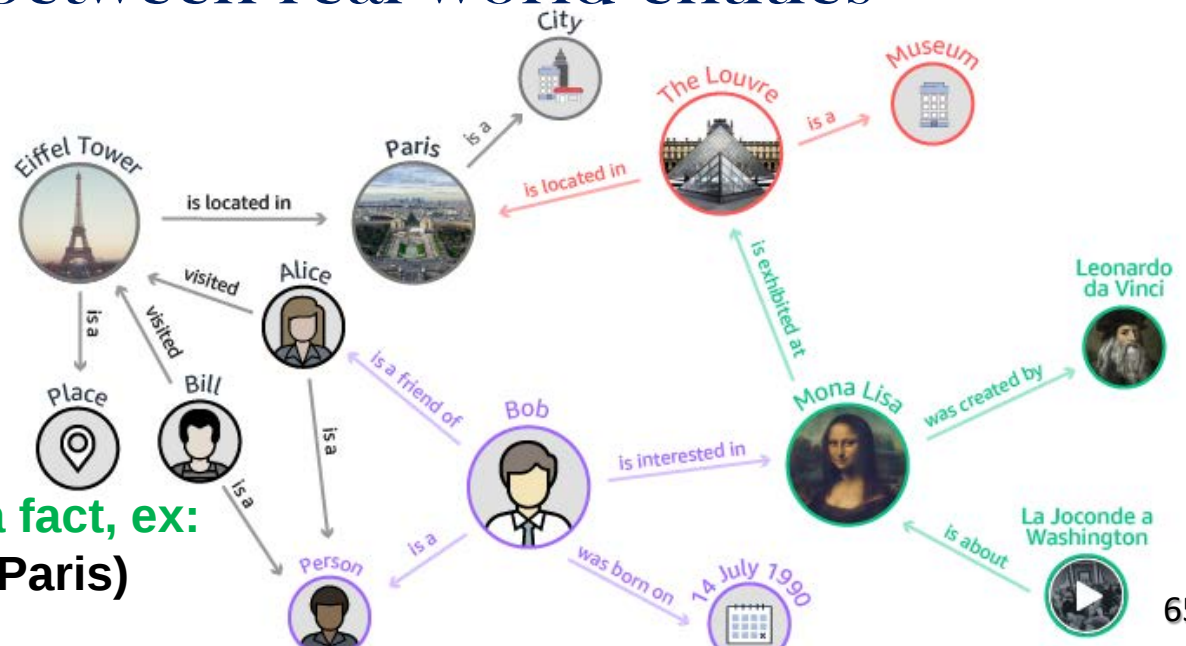
From shallow embedding to Graph Neural Networks

- The embedding function (encoder) is more complicated
 - Shallow embedding
 - $\phi(v) = U^T x_v$, where U is the embedding matrix and x_v is the one-hot encoding vector
 - Graph neural networks
 - $\phi(v)$ is a neural network depending on the graph structure

Knowledge Graph Embedding

Knowledge Graph

- What are knowledge graphs?
 - Multi-relational graph data
 - (heterogeneous information network)
 - Provide structured representation for semantic relationships between real-world entities



A triple (h, r, t) represents a fact, ex:
(Eiffel Tower, is located in, Paris)

Examples of KG

General-purpose KGs



Bio & Medical KGs



Product Graphs & E-commerce



Common-sense KGs & NLP



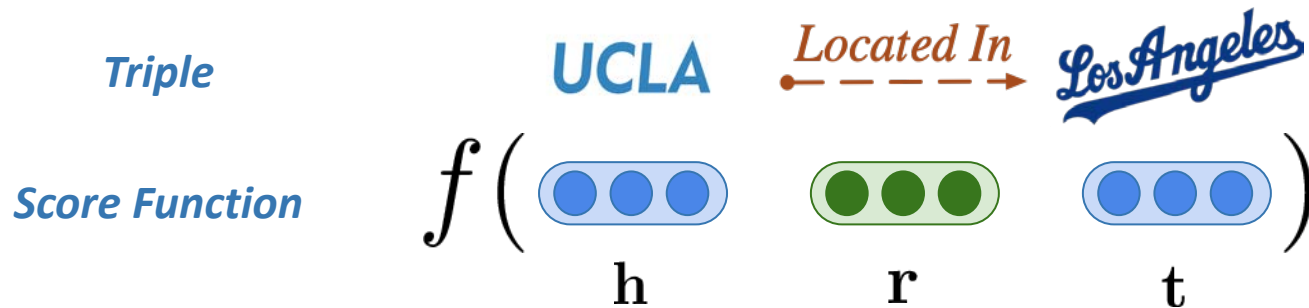
-

Knowledge Graph Embedding

- Goal:
 - Encode entities as low-dimensional vectors and relations as parametric algebraic operations
- Applications:
 - Dialogue agents
 - Question answering
 - Machine comprehension
 - Recommender systems
 - ...

Key Idea of KG embedding algorithms

- Define a score function for a triple: $f_r(\mathbf{h}, \mathbf{t})$
 - According to entity and relation representation



- Define a loss function to guide the training
 - E.g., an observed triple scores higher than a negative one

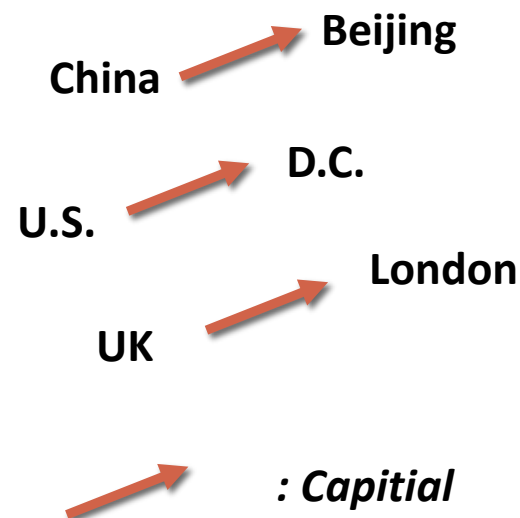
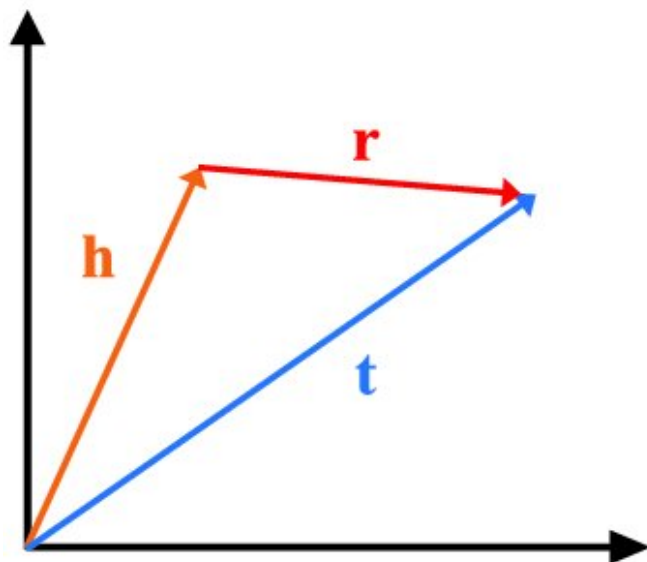
Summary of Existing Approaches

Model	Score Function	
SE (Bordes et al., 2011)	$-\ W_{r,1}h - W_{r,2}t\ $	$h, t \in \mathbb{R}^k, W_{r,\cdot} \in \mathbb{R}^{k \times k}$
TransE (Bordes et al., 2013)	$-\ h + r - t\ $	$h, r, t \in \mathbb{R}^k$
TransX	$-\ g_{r,1}(h) + r - g_{r,2}(t)\ $	$h, r, t \in \mathbb{R}^k$
DistMult (Yang et al., 2014)	$\langle r, h, t \rangle$	$h, r, t \in \mathbb{R}^k$
Complex (Trouillon et al., 2016)	$\text{Re}(\langle r, h, \bar{t} \rangle)$	$h, r, t \in \mathbb{C}^k$
HolE (Nickel et al., 2016)	$\langle r, h \otimes t \rangle$	$h, r, t \in \mathbb{R}^k$
ConvE (Dettmers et al., 2017)	$\langle \sigma(\text{vec}(\sigma([\bar{r}, \bar{h}] * \Omega))W), t \rangle$	$h, r, t \in \mathbb{R}^k$
RotatE	$-\ h \circ r - t\ ^2$	$h, r, t \in \mathbb{C}^k, r_i = 1$

Source: Sun et al., RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space (ICLR'19)

TransE: Score Function

- Relation: translating embedding



- Score function

$$f_r(\mathbf{h}, \mathbf{t}) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\| = -d(\mathbf{h} + \mathbf{r}, \mathbf{t})$$

TransE: Objective Function

- Objective Function
 - Margin-based ranking loss
 - $L = \sum_{(h,r,t) \in S} \sum_{(h',r,t') \in S'_{(h,r,t)}} [\gamma + d(\mathbf{h} + \mathbf{r}, \mathbf{t}) - d(\mathbf{h}' + \mathbf{r}, \mathbf{t}')]_+$
 - $[x]_+$ denotes the positive part of x , i.e., $\max(0, x)$
 - $\gamma > 0$ denotes the margin hyperparameter
 - The higher the bigger difference between positive triple and negative one
 - S : positive triple set; S' : corrupted triple set (negative triples)
- Optimization: stochastic gradient descent

TransE: Limitations

- One-one mapping: $t = \phi_r(h)$
 - Given (h,r) , t is unique
 - Given (r,t) , h is unique
- Anti-symmetric
 - If $r(h,t)$ then $r(t,h)$ is not true
 - Cannot model symmetric relation, e.g., friendship
- Anti-reflexive
 - $r(h,h)$ is not true
 - Cannot model reflexive relations, e.g., synonym

DistMult

- Bilinear score function
 - $f_r(\mathbf{h}, \mathbf{t}) = \mathbf{h}^T \mathbf{M}_r \mathbf{t}$
 - Where $\mathbf{M}_r$ is a diagonal matrix with diagonal vector $\mathbf{r}$
 - A simplification to neural tensor network (NTN)
- Objective function
 - $L = \sum_{(h,r,t) \in S} \sum_{(h',r,t') \in S'_{(h,r,t)}} [\gamma - f_r(\mathbf{h}, \mathbf{t}) + f_r(\mathbf{h}', \mathbf{t}')]_+$
- Limitation
 - Can only model symmetric relation
 - $f_r(\mathbf{h}, \mathbf{t}) = f_r(\mathbf{t}, \mathbf{h})$

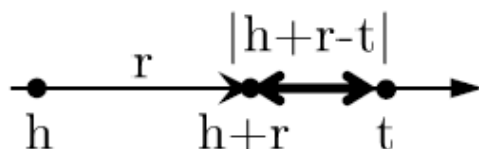
RotatE: Score Function

- Relation: rotation operation in complex space
 - head and tail entities in complex vector space, i.e., $\mathbf{h}, \mathbf{t} \in \mathbb{C}^k$
 - each relation r as an element-wise rotation from the head entity $\mathbf{h}$ to the tail entity $\mathbf{t}$, i.e.,
 - $\mathbf{t} = \mathbf{h} \circ \mathbf{r}$, i.e., $t_i = h_i r_i$, where $|r_i| = 1$
 - Equivalently, $r_i = e^{i\theta_{r,i}}$, i.e., rotate h_i with $\theta_{r,i}$
- Score function:
 - $f_r(h, t) = -||\mathbf{h} \circ \mathbf{r} - \mathbf{t}||$

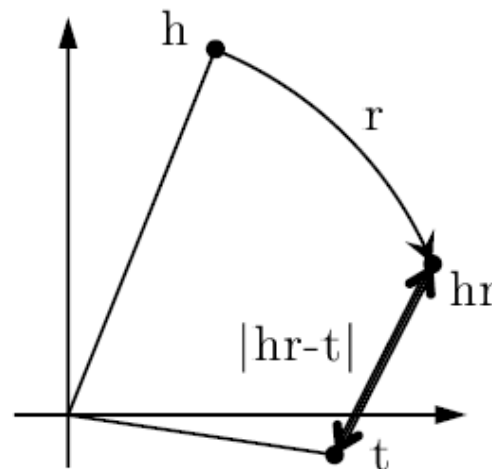
Zhiqing Sun, Zhihong Deng, Jian-Yun Nie, and Jian Tang. “RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space.” ICLR’19.

RotatE: Geometric Interpretation

- Consider 1-d case



(a) TransE models r as translation in real line.



(b) RotatE models r as rotation in complex plane.

RotatE: Objective function

- Smarter negative sampling
 - The negative triple with higher score is more likely to be sampled

$$p(h'_j, r, t'_j | \{(h_i, r_i, t_i)\}) = \frac{\exp \alpha f_r(\mathbf{h}'_j, \mathbf{t}'_j)}{\sum_i \exp \alpha f_r(\mathbf{h}'_i, \mathbf{t}'_i)}$$

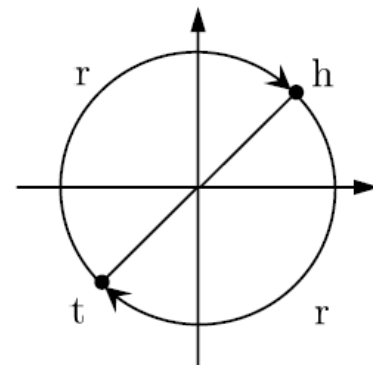
- Cross-entropy loss

$$L = -\log \sigma(\gamma - d_r(\mathbf{h}, \mathbf{t})) - \sum_{i=1}^n p(h'_i, r, t'_i) \log \sigma(d_r(\mathbf{h}'_i, \mathbf{t}'_i) - \gamma)$$

RotatE: Pros and Cons

- Pros:

- Can model relations with different properties
- Symmetric: $r_i = +1$ or -1
- Anti-symmetric: $\mathbf{r} \circ \mathbf{r} \neq \mathbf{1}$
- Inverse relations: $\mathbf{r}_2 = \bar{\mathbf{r}}_1$
 - E.g., hypernym is the **inverse** relation of hyponym
- Composition relations: $\mathbf{r}_3 = \mathbf{r}_1 \circ \mathbf{r}_2$, i.e., $\theta_3 = \theta_1 + \theta_2$, if $\mathbf{r}_j = e^{i\theta_j}$ for $j = 1, 2, 3$



- Cons:

- One-one mapping
- Relations are commutative: i.e., $\mathbf{r}_1 \circ \mathbf{r}_2 = \mathbf{r}_2 \circ \mathbf{r}_1$
 - Which is not always true, e.g., father's wife $\neq$ wife's father

Content

- Introduction to Graphs
- Spectral analysis
- Shallow Embedding
- Graph Neural Networks 

Notations

- An attributed graph $G = (V, E)$
 - V : vertex set
 - E : edge set
 - A : adjacency matrix
 - $X \in R^{d_0 \times |V|}$: feature matrix for all the nodes
 - $N(v)$: neighbors of node v
 - h_v^l : Representation vector of node v at Layer l
 - *Note* $h_v^0 = x_v$
 - $H^l \in R^{d_l \times |V|}$: representation matrix

The General Architecture of GNNs

- For a node v at layer t

$$h_v^{(t)} = f \left(\underbrace{h_v^{(t-1)}}_{\text{representation vector from previous layer for node } v}, \underbrace{\left\{ h_u^{(t-1)} \mid u \in \mathcal{N}(v) \right\}}_{\text{representation vectors from previous layer for node } v\text{'s neighbors}} \right)$$

representation vector
from previous layer for
node v

representation vectors
from previous layer for
node v 's neighbors

- A function of representations of neighbors and itself from previous layers
 - **Aggregation** of neighbors
 - **Transformation** to a different space
 - **Combination** of neighbors and the node itself

Compare with CNN

- Recall CNN
 - Regular graph

- GNN

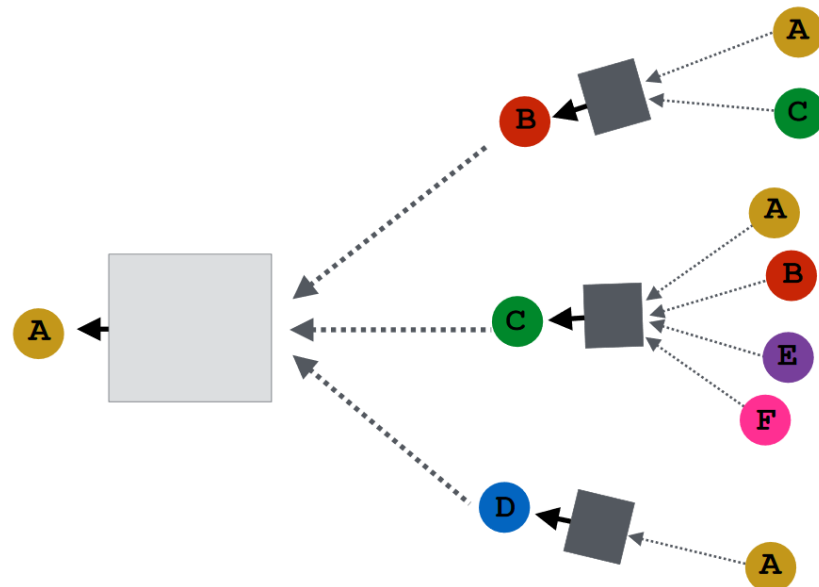
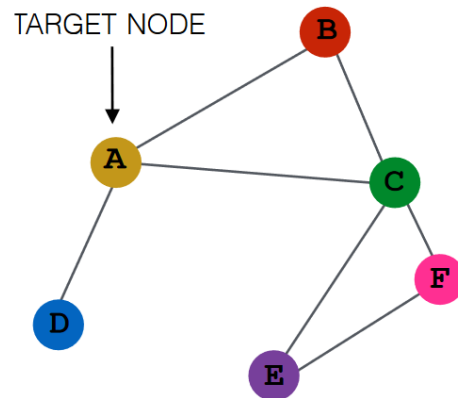
- Extend to irregular graph structure

1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature



Graph Convolutional Network (GCN)

- Kipf and Welling, ICLR'17

- $f(H^{(l)}, A) = \sigma \left(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H^{(l)} W^{(l)} \right), \hat{A} = A + I$

- f : graph filter

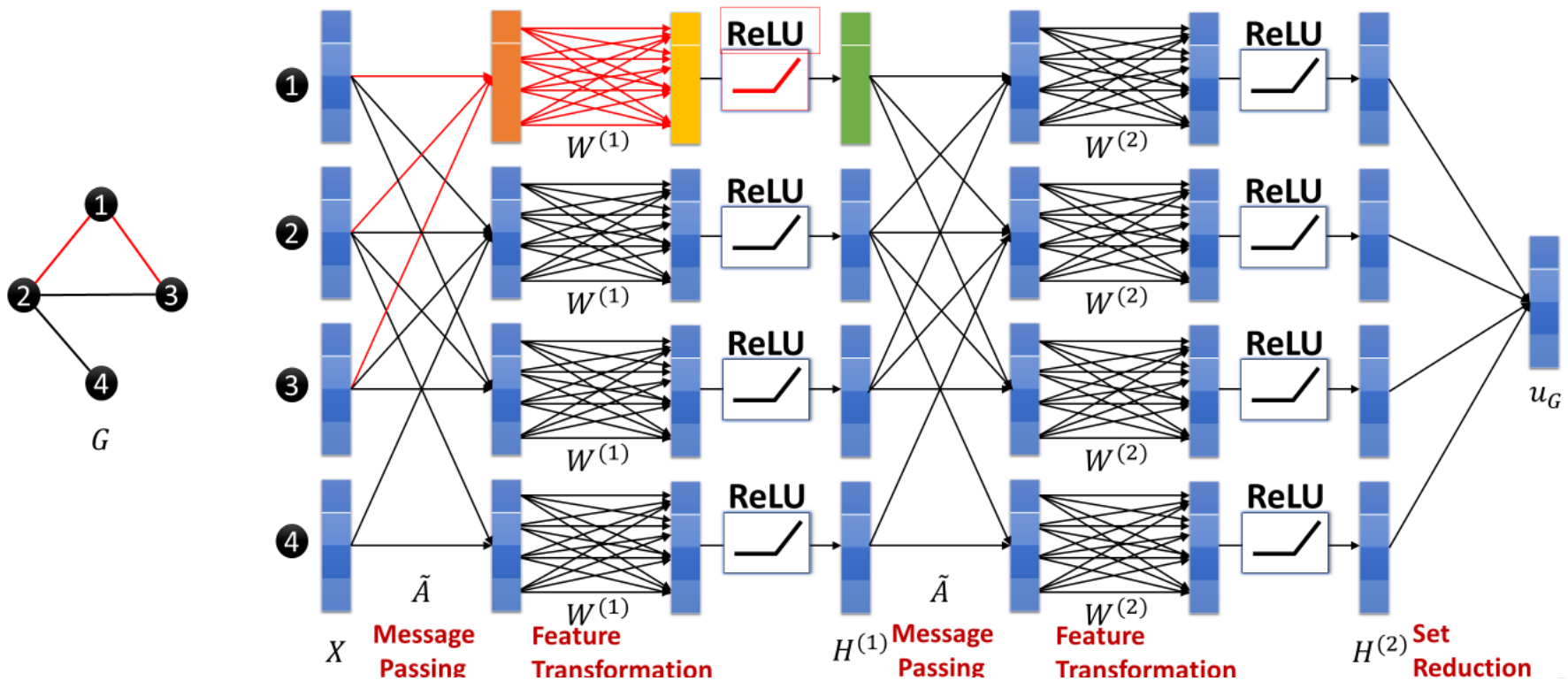
- From a node v 's perspective

$$\mathbf{h}_v^k = \sigma \left(\mathbf{W}_k \sum_{u \in N(v) \cup v} \frac{\mathbf{h}_u^{k-1}}{\sqrt{|N(u)| |N(v)|}} \right)$$

W_k : weight matrix at Layer k , shared across different nodes

A toy example of 2-layer GCN on a 4-node graph

- Computation graph



GraphSAGE

- Inductive Representation Learning on Large Graphs

William L. Hamilton*, Rex Ying*, Jure Leskovec,
NeurIPS'17

$$\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$$
$$\mathbf{h}_v^k \leftarrow \sigma \left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k) \right)$$

A more general form

$$\mathbf{h}_v^k = \sigma \left(\left[\mathbf{W}_k \cdot \overleftarrow{\text{AGG}} \left(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\} \right), \overrightarrow{\mathbf{B}_k \mathbf{h}_v^{k-1}} \right] \right)$$

More about AGG

- Mean
$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$$
- LSTM
$$\text{AGG} = \text{LSTM}([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))])$$
 - $\pi(\cdot)$: a random permutation
- Pool
$$\text{AGG} = \gamma \{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}$$
 - $\gamma(\cdot)$: Element-wise mean/max pooling of neighbor set

Message-Passing Neural Network

- Gilmer et al., 2017. Neural Message Passing for Quantum Chemistry. *ICML*.
- *A general framework that subsumes most GNNs*
 - Can also include **edge** information
- Two steps
 - Get messages from neighbors at step k

$$\mathbf{m}_v^k = \sum_{u \in N(v)} M(\mathbf{h}_u^{k-1}, \mathbf{h}_v^{k-1}, \mathbf{e}_{u,v}) \quad \text{e.g., Sum or MLP}$$

- Update the node latent represent based on the msg $\mathbf{h}_v^k = U(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$ e.g., LSTM, GRU

A special case: GGNN, Li et al., Gated graph sequence neural networks, ICLR 2015

Graph Attention Network (GAN)

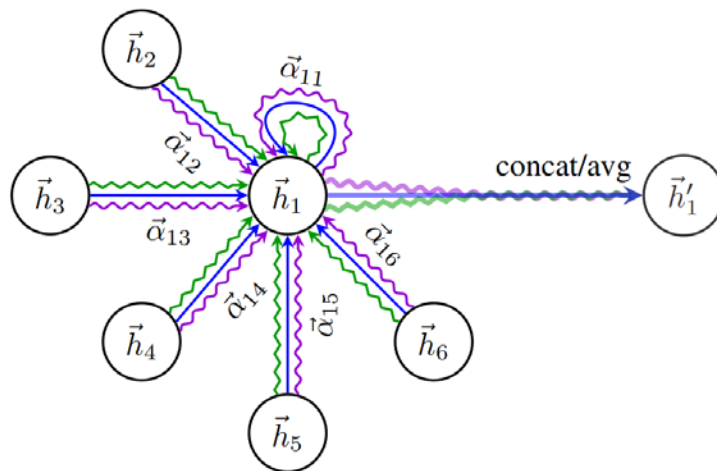
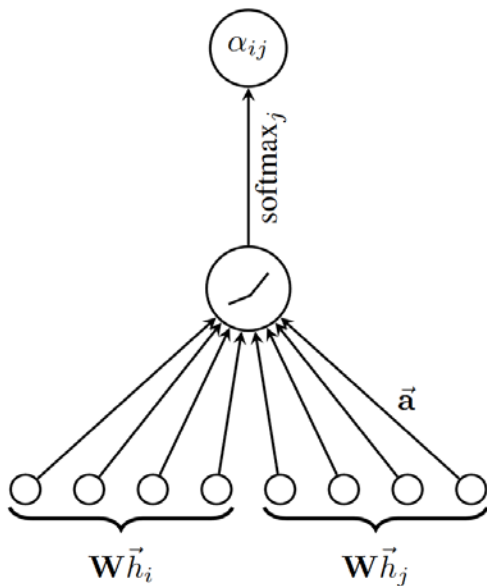
- How to decide the importance of neighbors?
 - GCN: a predefined weight
 - Others: no differentiation
- GAN: decide the weights using learnable attention
 - Velickovic et al., 2018. Graph Attention Networks. *ICLR*.

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

The attention mechanism

- Potentially many possible designs

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



Challenges of GNNs

- Scalability
- Heterogeneity
- Label Scarcity

Challenge 1: Scalability

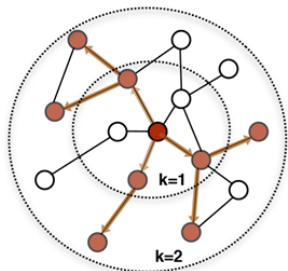
Existing Training Approaches and Challenges

- Full Batch

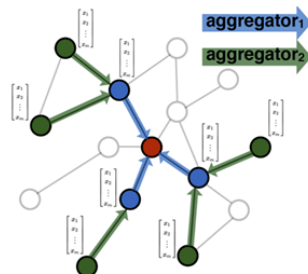
- Memory complexity: $O(LNd + Ld^2)$
- Time complexity: $O(L|E|d + LNd^2)$
 - L: number of layers; N: number of nodes; d: dimensions of embedding; |E|: number of edges

- Mini Batch

- GraphSage: node-wise neighbor sampling



(1) Sample sub-graph to determine a smaller computation graph.



(2) Propagate and transform message on smaller computation graph

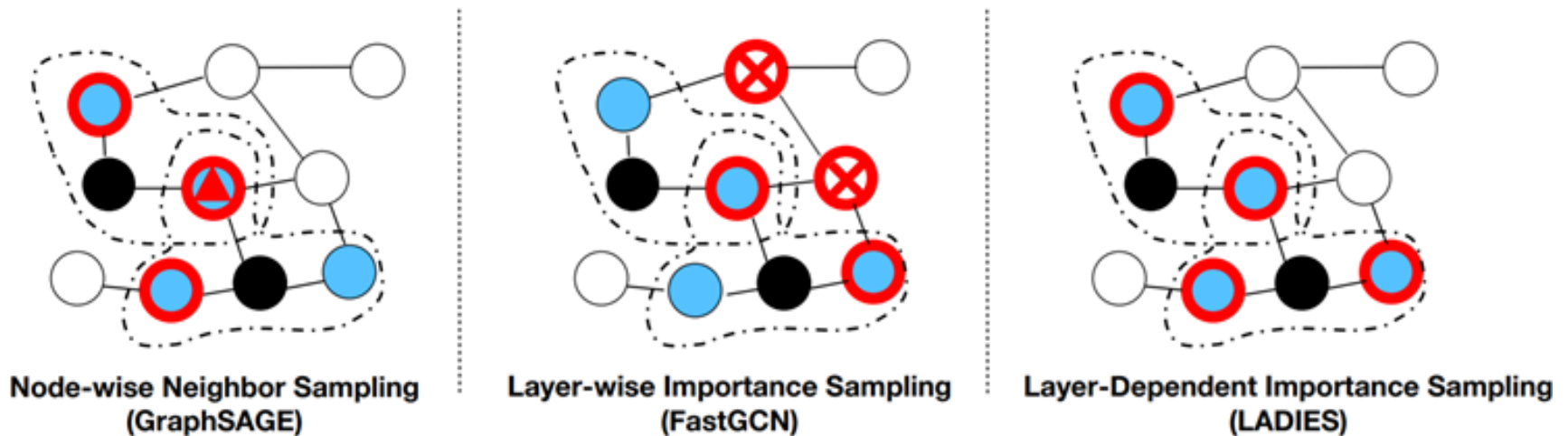
William L. Hamilton*, Rex Ying*, Jure Leskovec.
“Inductive Representation Learning on Large Graphs”, NeurIPS 2017

- FastGCN: layer-wise importance sampling

Chen et al., “Fastgcn: Fast learning with graph convolutional networks via importance sampling”, ICLR'18

Our Solution: Layer-Dependent Importance Sampling (LADIES), Zou, Hu et al., NeurIPS'19

- Conduct importance sampling based on previous sampled nodes
- The sampled graph is dense and non-redundant



Black nodes: selected nodes in minibatch; Circled nodes: sampled nodes;
Crossed nodes: isolated samples; Triangle nodes: duplicate samples

Memory: $O(bS^L d + Ld^2)$
Time: $O(bDS^L d + bS^L d^2)$

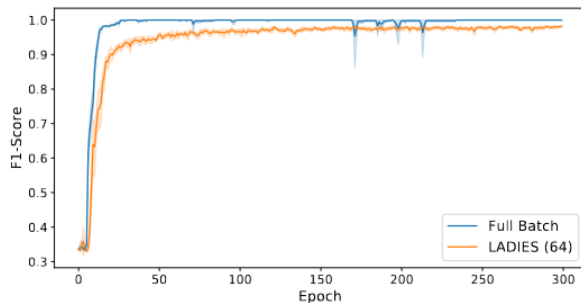
Memory: $O(LSd + Ld^2)$
Time: $O(LS^2 d + Ld^2)$

Memory: $O(LSd + Ld^2)$
Time: $O(LS^2 d + Ld^2)$

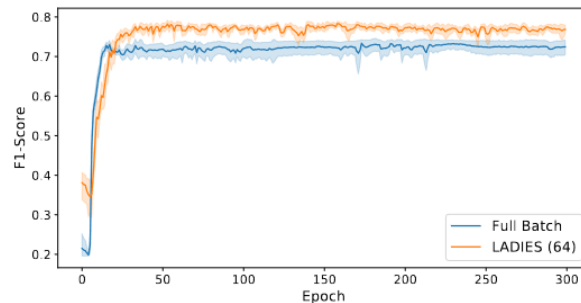
L: number of layers; N: number of nodes; d: dimensions of embedding; |E|: number of edges
S: number of samples per node or layer; b: size of mini-batch in terms of nodes; D: average degree of nodes

Results: Better Accuracy with Lower Time and Memory Cost

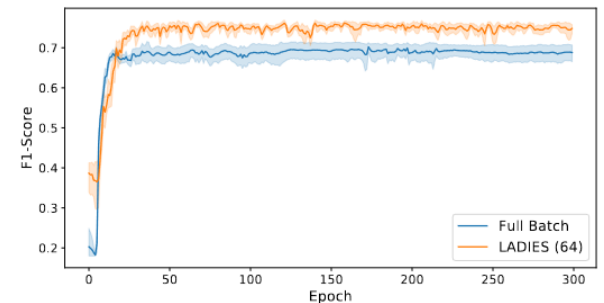
Dataset	Sample Method	F1-Score(%)	Total Time(s)	Mem(MB)	Batch Time(ms)	Batch Num
Pubmed (19717)	Full-Batch	71.9 ± 1.9	4.80 ± 1.53	137.93	44.69 ± 0.57	102 ± 33.4
	GraphSage (5)	70.1 ± 1.4	5.53 ± 2.57	453.58	44.73 ± 0.30	74.8 ± 31.7
	FastGCN (64)	38.5 ± 6.9	0.40 ± 0.69	1.92	7.42 ± 0.16	58.8 ± 94.8
	FastGCN (512)	39.3 ± 9.2	0.44 ± 0.61	4.53	10.06 ± 0.41	44.8 ± 55.0
	FastGCN (8192)	74.4 ± 0.8	3.47 ± 1.16	49.41	17.84 ± 0.33	195 ± 56.9
	LADIES (64)	76.8 ± 0.8	2.57 ± 0.72	1.92	9.43 ± 0.47	277 ± 82.2
	LADIES (512)	75.9 ± 1.1	2.27 ± 1.17	4.39	10.43 ± 0.36	245 ± 84.5
Reddit (232965)	Full-Batch	91.6 ± 1.6	474.3 ± 84.4	2370.48	1564 ± 3.41	179 ± 75.5
	GraphSage (5)	92.1 ± 1.1	13.12 ± 2.84	1234.63	121.47 ± 0.72	81.5 ± 42.3
	FastGCN (64)	27.8 ± 12.6	2.06 ± 1.29	3.75	7.85 ± 0.72	57.4 ± 43.7
	FastGCN (512)	17.5 ± 16.7	0.31 ± 0.41	6.91	10.01 ± 0.31	32.1 ± 72.3
	FastGCN (8192)	89.5 ± 1.2	5.63 ± 2.12	74.28	16.57 ± 0.58	278 ± 51.2
	LADIES (64)	83.5 ± 0.9	5.62 ± 1.58	3.75	9.42 ± 0.48	453 ± 88.2
	LADIES (512)	92.8 ± 1.6	6.87 ± 1.17	7.26	10.87 ± 0.63	393 ± 74.4



(a) Training (60 data samples)



(b) Validation (500 data samples)

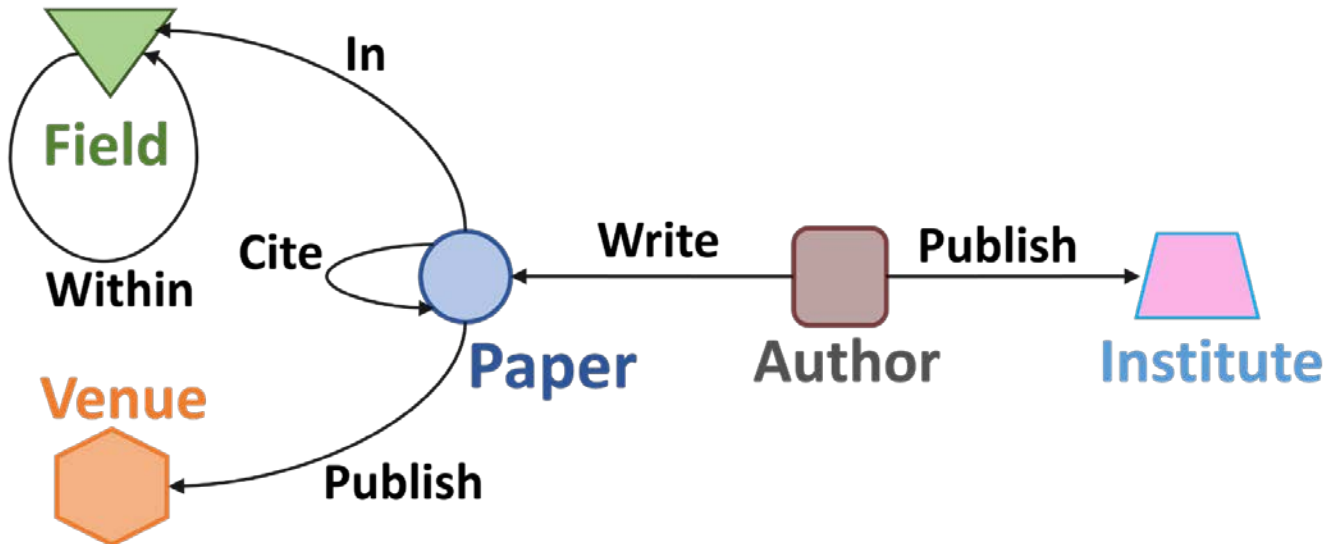


(c) Testing (1000 data samples)

Challenge 2: Heterogeneity

What are Heterogeneous Networks?

- Different types of nodes and edges

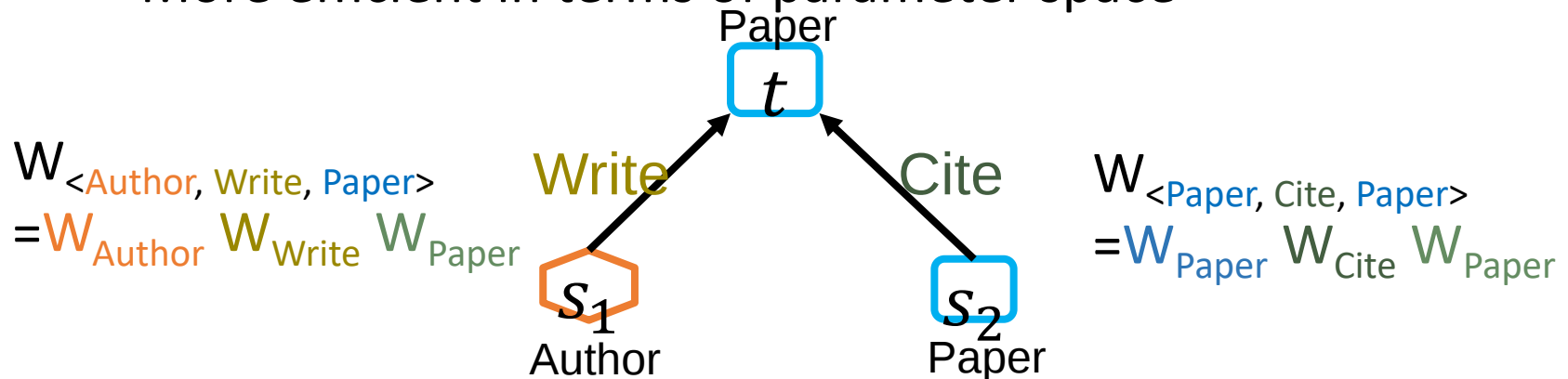


Example: Network Schema of Academic Networks

- Other examples:
 - E-Commerce
 - Knowledge graphs

Our Solution: Heterogeneous Graph Transformer (HGT), Hu et al., WWW'20

- Parameterization by Meta-Relation
 - Meta-Relation: $\langle \text{source_type}, \text{edge_type}, \text{target_type} \rangle$
 - E.g., $\langle \text{author}, \text{first_author_of}, \text{paper} \rangle$, $\langle \text{author}, \text{second_author_of}, \text{paper} \rangle$
 - Parameter Sharing
 - Capture the correlation between different meta-relations
 - More efficient in terms of parameter space



Message, Attention, and Aggregation of HGT

- Heterogeneous message for an edge $\langle s, e, t \rangle$

$$\mathbf{Message}_{HGT}(s, e, t) = \parallel_{i \in [1, h]} \text{MSG-head}^i(s, e, t)$$

$$\text{MSG-head}^i(s, e, t) = \text{M-Linear}_{\tau(s)}^i \left(H^{(l-1)}[s] \right) W_{\phi(e)}^{MSG}$$

- Heterogeneous mutual attention on edge $\langle s, e, t \rangle$

$$\mathbf{Attention}_{HGT}(s, e, t) = \text{Softmax}_{\forall s \in N(t)} \left(\parallel_{i \in [1, h]} \text{ATT-head}^i(s, e, t) \right) \quad (3)$$

$$\text{ATT-head}^i(s, e, t) = \left(K^i(s) W_{\phi(e)}^{ATT} Q^i(t)^T \right) \cdot \frac{\mu \langle \tau(s), \phi(e), \tau(t) \rangle}{\sqrt{d}}$$

$$K^i(s) = \text{K-Linear}_{\tau(s)}^i \left(H^{(l-1)}[s] \right)$$

$$Q^i(t) = \text{Q-Linear}_{\tau(t)}^i \left(H^{(l-1)}[t] \right)$$

Significant score for each meta-relation

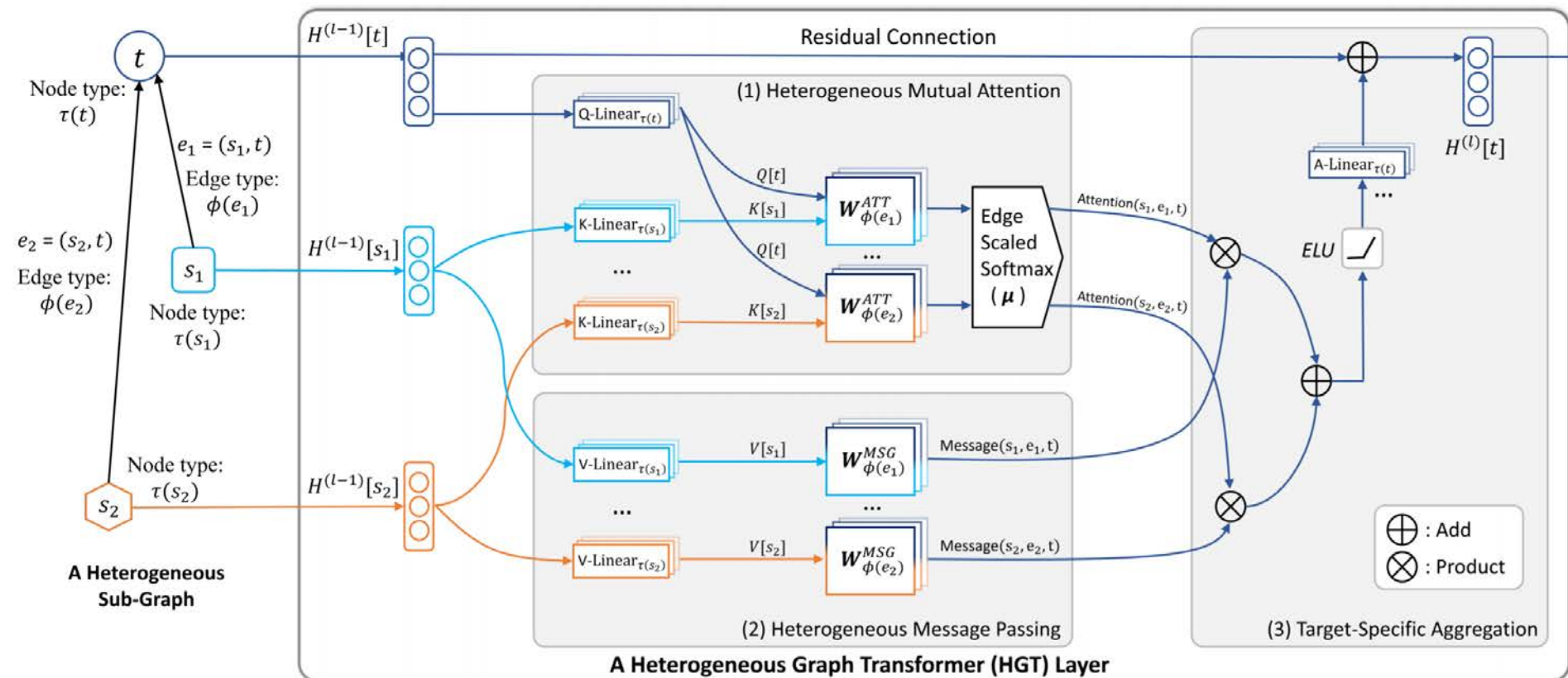
- Task-specific aggregation

$$\tilde{H}^{(l)}[t] = \bigoplus_{\forall s \in N(t)} \left(\mathbf{Attention}_{HGT}(s, e, t) \cdot \mathbf{Message}_{HGT}(s, e, t) \right)$$

$$H^{(l)}[t] = \text{A-Linear}_{\tau(t)} \left(\sigma \left(\tilde{H}^{(l)}[t] \right) \right) + H^{(l-1)}[t].$$

Architecture of HGT

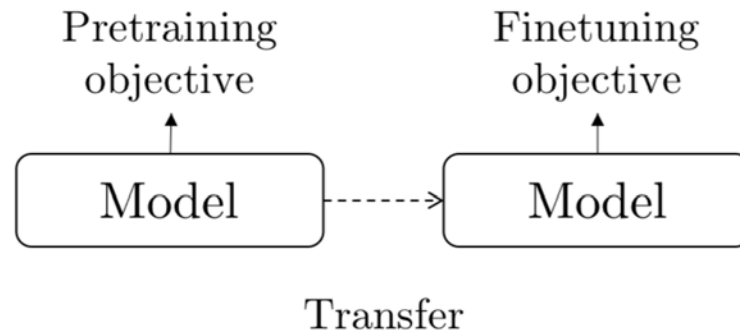
- Putting together



Challenge 3: Label Scarcity

Pre-Training of Graph Neural Networks

- Challenges on training GNNs
 - Requires abundant task-specific labeled data
- What is pre-training?
 - Train GNNs with self-supervision and then transfer learned model to downstream tasks with only a few labels
 - Popular in NLP: e.g., BERT

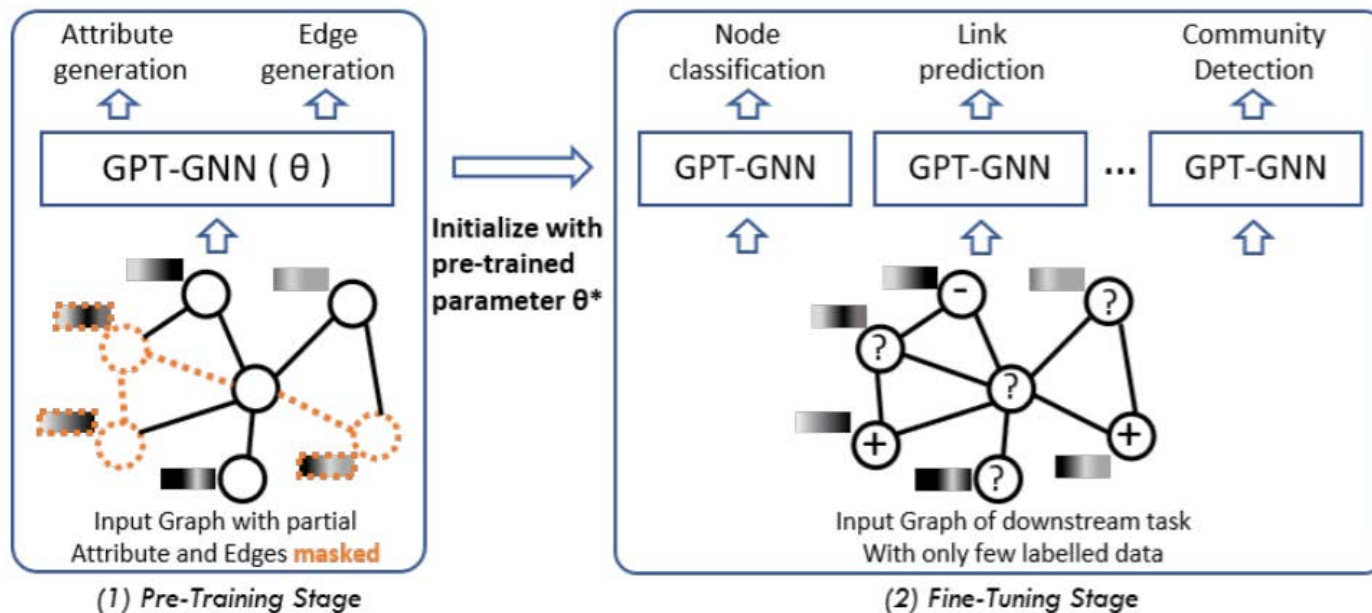


Key to the Success of Pre-Training

- Self-supervised Tasks
 - No additional labels are needed
 - General enough to different downstream tasks
- Existing self-supervised tasks for graphs
 - Link prediction [GAE, GraphSAGE (NIPS'17)]
 - Maximize mutual information between a patch and its super graph [DGI (ICLR'19)]

Our Solution: GPT-GNN (Hu et al., KDD'20)

- Pre-train GNNs via the generative task to generate both node attributes and edges
- Goal: find GNN parameter $\theta^* = \max_{\theta} p(G; \theta)$



The Power of Knowledge Transfer

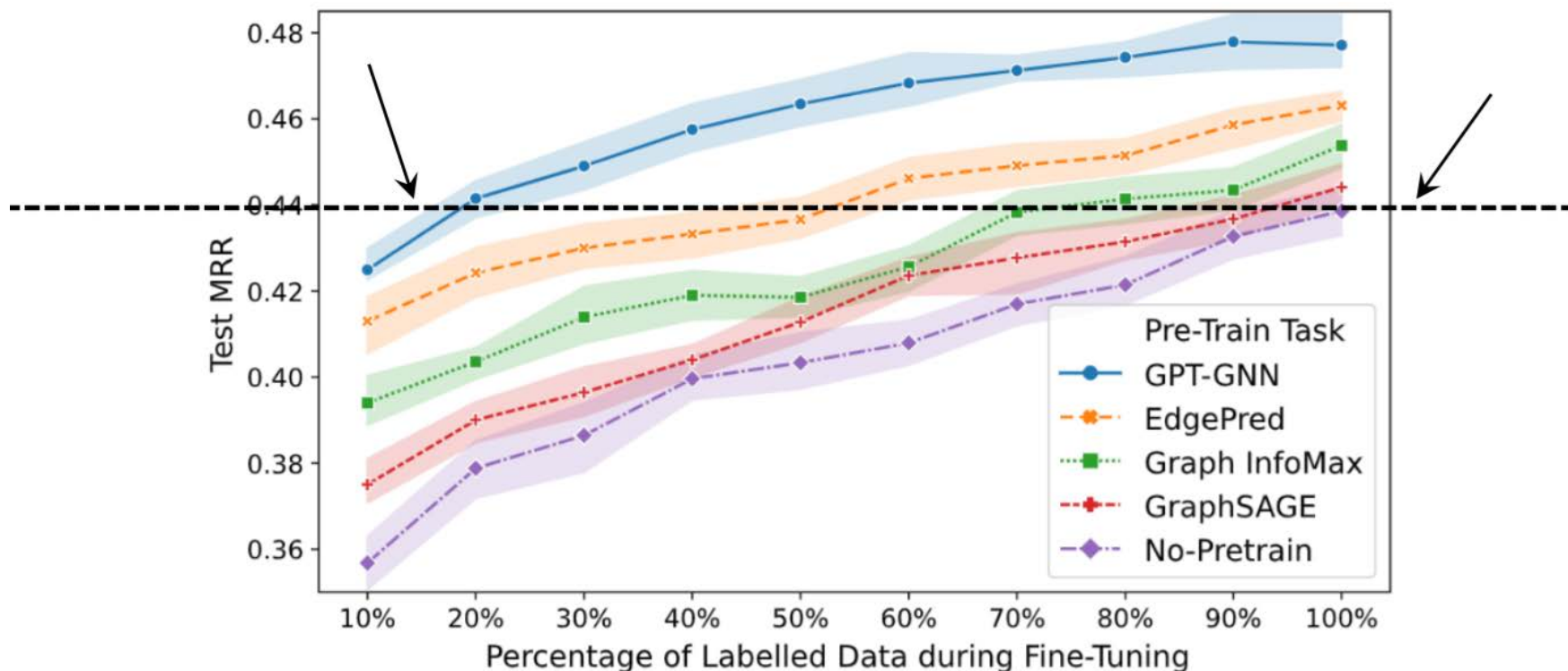
• Data 1: Open Academic Graph

	Pre-Train	Fine-Tune
No Transfer:	CS Academic Graph	CS Academic Graph
Field Transfer:	Med, Bio, Physics...	CS Academic Graph
Time Transfer:	CS before 2014	CS after 2014
Time + Field Transfer:	Med, Bio, Physics... before 2014	CS after 2014

- **All pre-training frameworks** help the performance of GNNs
 - GAE, GraphSage, Graph Infomax
 - GPT-GNN
- **GPT-GNN helps the most** by achieving a relative performance gain of 9.1% over the base model without pre-training
- **Both self-supervised tasks in GPT-GNN** help the pre-training framework
 - Attribute generation
 - Edge generation

Downstream Dataset		OAG		
Evaluation Task		Paper-Field	Paper-Venue	Author ND
Field Transfer	No Pre-train	.336±.149	.365±.122	.794±.105
	GAE	.403±.114	.418±.093	.816±.084
	GraphSAGE (unsp.)	.368±.125	.401±.096	.803±.092
	Graph Infomax	.387±.112	.404±.097	.810±.084
	GPT-GNN (Attr)	.396±.118	.423±.105	.818±.086
	GPT-GNN (Edge)	.401±.109	.428±.096	.826±.093
	GPT-GNN	.407±.107	.432±.098	.831±.102
	GAE	.384±.117	.412±.101	.812±.095
	GraphSAGE (unsp.)	.352±.121	.394±.105	.799±.093
Time Transfer	Graph Infomax	.369±.116	.398±.102	.805±.089
	GPT-GNN (Attr)	.382±.114	.414±.098	.811±.089
	GPT-GNN (Edge)	.392±.105	.421±.102	.821±.088
	GPT-GNN	.400±.108	.429±.101	.825±.093
Time + Field Transfer	GAE	.371±.124	.403±.108	.806±.102
	GraphSAGE (unsp.)	.349±.130	.393±.118	.797±.097
	Graph Infomax	.360±.121	.391±.102	.800±.093
	GPT-GNN (Attr)	.364±.115	.409±.103	.809±.094
	– (w/o node separation)	.347±.128	.391±.102	.791±.108
	GPT-GNN (Edge)	.386±.116	.414±.104	.815±.105
	– (w/o adaptive queue)	.376±.121	.410±.115	.808±.104
	GPT-GNN	.393±.112	.420±.108	.818±.102

Much Less labels are needed



The GNN model **w/o** pre-training by using **100%** training data
VS
The GNN model **with** pre-training by using **10-20%** training data

Downstream Tasks for Graphs

Typical Graph Functions

- Node level

- Similarity search
- **Link prediction**
- **Classification**
- Community detection
- Ranking

- Graph level

- **Similarity search**
- Frequent pattern mining
- Graph isomorphism test
- Graph matching
- **Classification**
- Clustering
- Graph generation

1. Semi-supervised Node Classification

- Decoder using $z_v = h_v^L$
 - Feed into another fully connected layer
 - $\hat{y}_v = \sigma(\theta^T z_v)$
- Loss function
 - Cross entropy loss
 - In a binary classification case
 - $l_v = y_v \log \hat{y}_v + (1 - y_v) \log(1 - \hat{y}_v)$

Applications of Node Classification

- Social network
 - An account is bot or not
- Citation network
 - A paper's research field
- A program-derived graph
 - The type of a variable

2. Link Prediction

- Decoder using $z_v = h_v^L$
 - Given a node pair (u, v)
 - Determine its probability $p_{uv} = z_u^T R z_v$
 - R could be different for different relation type
- Loss function
 - Cross entropy loss
 - $l_{uv} = y_{uv} \log p_{uv} + (1 - y_{uv}) \log(1 - p_{uv})$

Link Prediction Applications

- Social network
 - Friend recommendation
- Citation network
 - Citation recommendation
- Medical network
 - Drug and target binding or not
- A program-derived graph
 - Code autocomplete

3. Graph Classification

- Decoder using $h_G = g(\{z_v\}_{v \in V})$
 - $g(\cdot)$: a read out function, e.g., sum
 - Feed h_G into another fully connected layer
 - $\hat{y}_G = \sigma(\theta^T h_G)$
- Loss function
 - Cross entropy loss
 - In a binary classification case
 - $l_G = y_G \log \hat{y}_G + (1 - y_G) \log(1 - \hat{y}_G)$

Graph Classification Applications

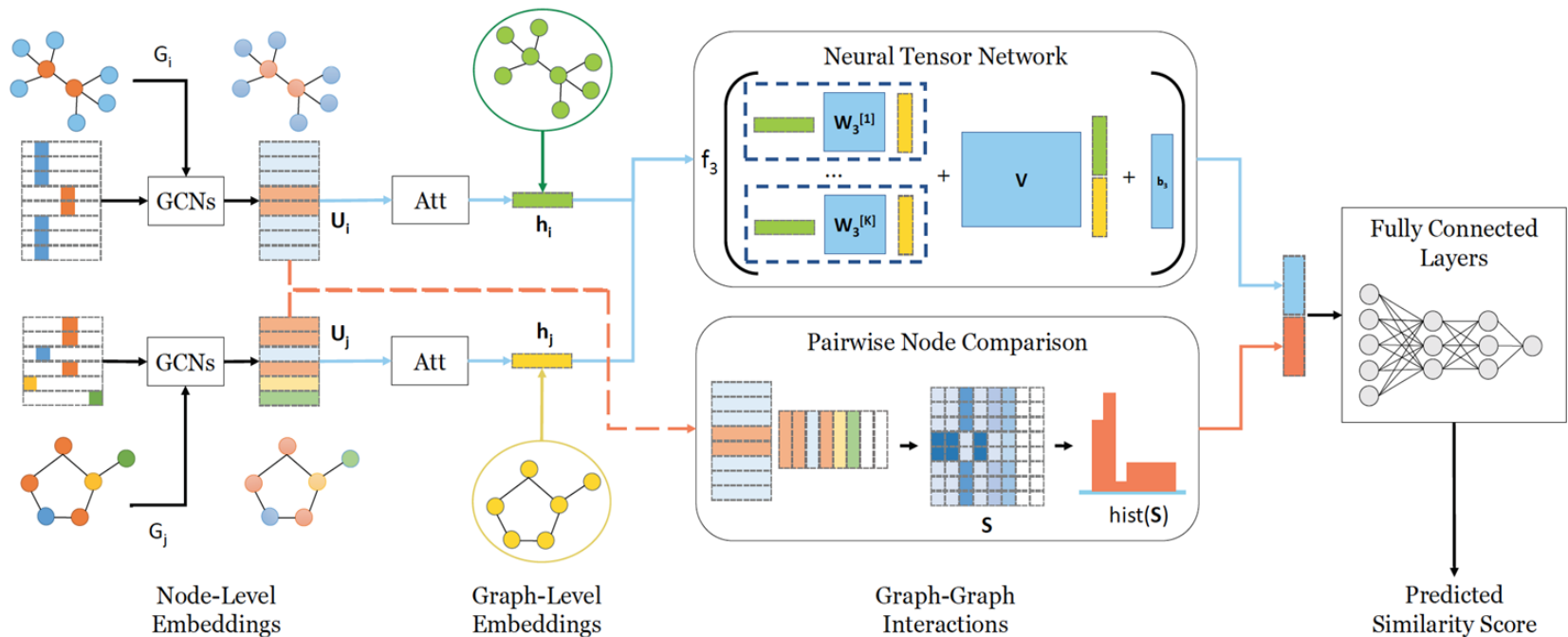
- Chemical compounds
 - Toxic or not
- Proteins
 - Has certain function or not
- Program-derived graphs
 - Contains bugs or not

4. Graph Similarity Computation

- Decoder using $h_G = g(\{z_v\}_{v \in V})$
 - Given a graph pair (G_1, G_2)
 - Determine its score $s_{G_1 G_2} = h_{G_1}^T R h_{G_2}$
- Loss function
 - E.g., Square loss
 - $l_{G_1 G_2} = (y_{G_1 G_2} - s_{G_1 G_2})^2$

A Concrete solution by SimGNN [Bai et al., AAAI 2019]

- Goal: learn a GNN $\phi: \mathcal{G} \times \mathcal{G} \rightarrow R^+$ to approximate Graph Edit Distance between two graphs



1. **Attention-based graph-level embedding**
2. **Histogram features from pairwise node similarities**

Graph Similarity Computation

Applications

- Drug database
 - Drug similarity search
- Program database
 - Code recommendation
 - Search ninja code for novice code
 - Search java code for COBOL code



Wanted urgently: People who know a half century-old computer language so states can process unemployment claims

By Alicia Lee, CNN

Updated 4:00 PM ET, Wed April 8, 2020

Summary

- Introduction to Graphs
- Spectral analysis
- Shallow Embedding
- Graph Neural Networks