CSE7850/CX4803 Machine Learning in Computational Biology

Lecture 13: Learning from Network Data

Yunan Luo

Acknowledgements: Some of the slides were adapted from CS224W, Jure Leskovec

- Learning from sequence data
- Learning from high-dim data
- **Learning from network data**
- Learning from structure data

Week	Date	Topic	Contents
1	01/08	Introduction	Introduction & Logistics
1	01/10	Basics in computational biology	Molecular biology
2	01/15		No class (MLK day)
2	01/17		Sequence alignment I
3	01/22		Sequence alignment II
3	01/24	ML foundations	No Class (PyTorch video + exercise)
4	01/29		Regression & Gradient descent
4	01/31		Classification & Toolbox for Applied ML
5	02/05		Neural networks
5	02/07		Deep learning
6	02/12	Learning from sequence data	Deep learning for Protein/DNA sequences
6	02/14		Large language models (LLMs)
7	02/19	Learning from high-dim data	Clustering and dimensionality reduction
7	02/21		Generative AI
8	02/26	Learning from network data	Network basics & ML for graphs
8	02/28		Graph neural network
9	03/04	Learning from structure data	Protein structure prediction & generation (AlphaFold, diffusion models)

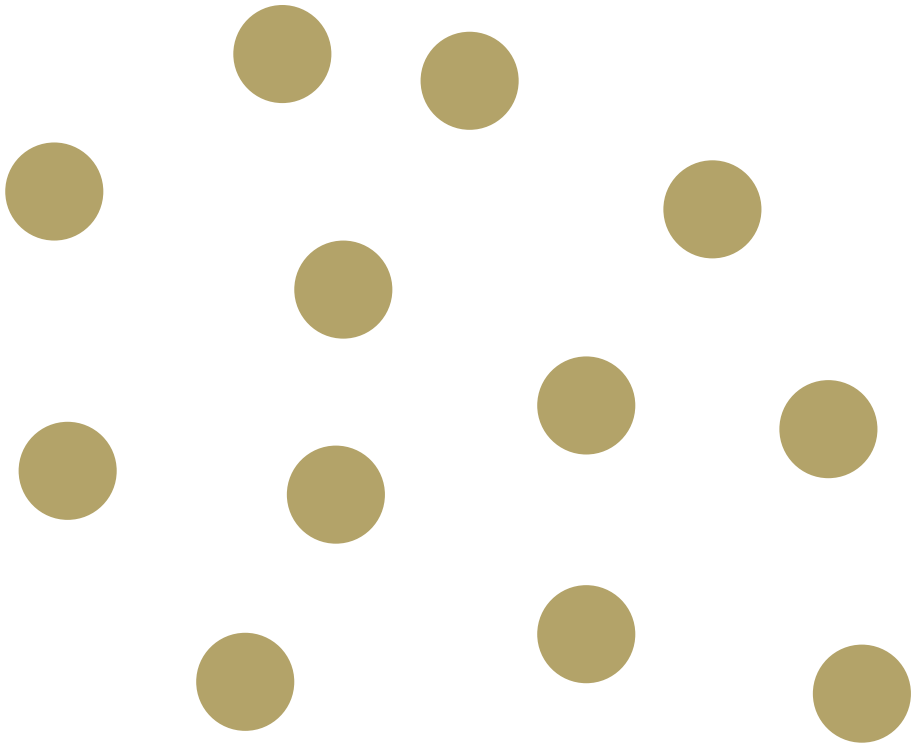
Outline

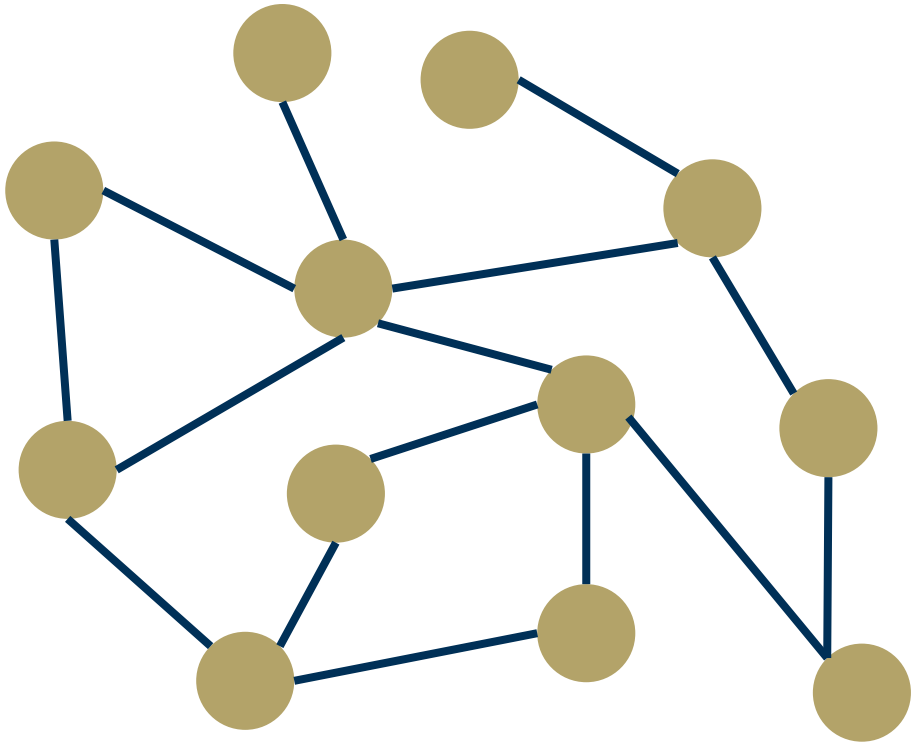
- Network (graph) data
  - Definitions & Examples
- Different graph ML tasks & features
  - Node-level
  - Edge-level
  - Graph-level
- Representations learning in graphs
  - Network embeddings

Outline

- Network (graph) data
  - Definitions & Examples
- Different graph ML tasks & features
  - Node-level
  - Edge-level
  - Graph-level
- Representations learning in graphs
  - Network embeddings

Network (graph) basics





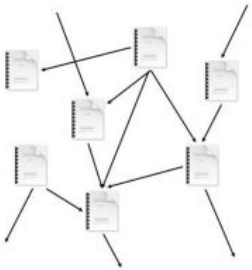
Networks (graphs)

Image credit: Medium

Social networks

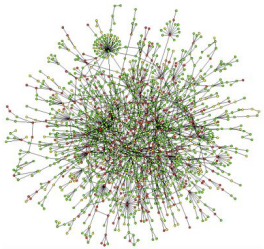
Image credit: Missoula Current News

Internet

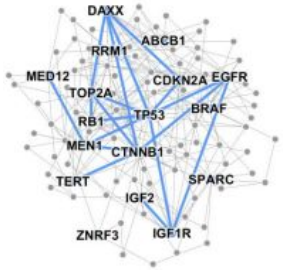


Citation networks

Underground networks



Protein-protein interaction networks



Disease pathways

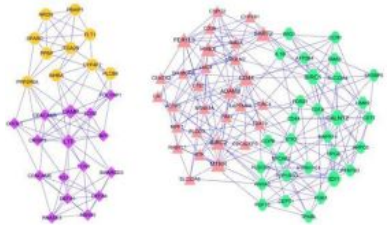


Image credit: ese.wustl.edu

Gene regulatory networks

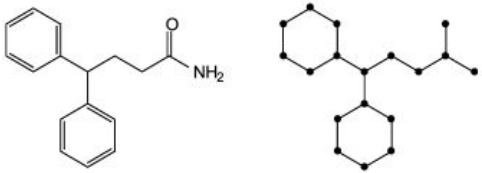
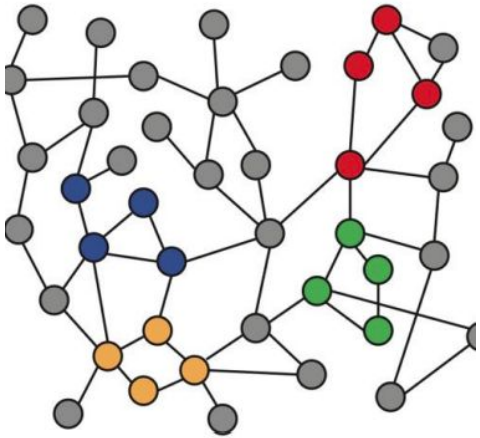


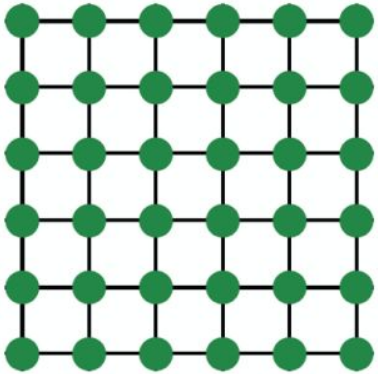
Image credit: MDPI

Molecules

Network is a general data representation

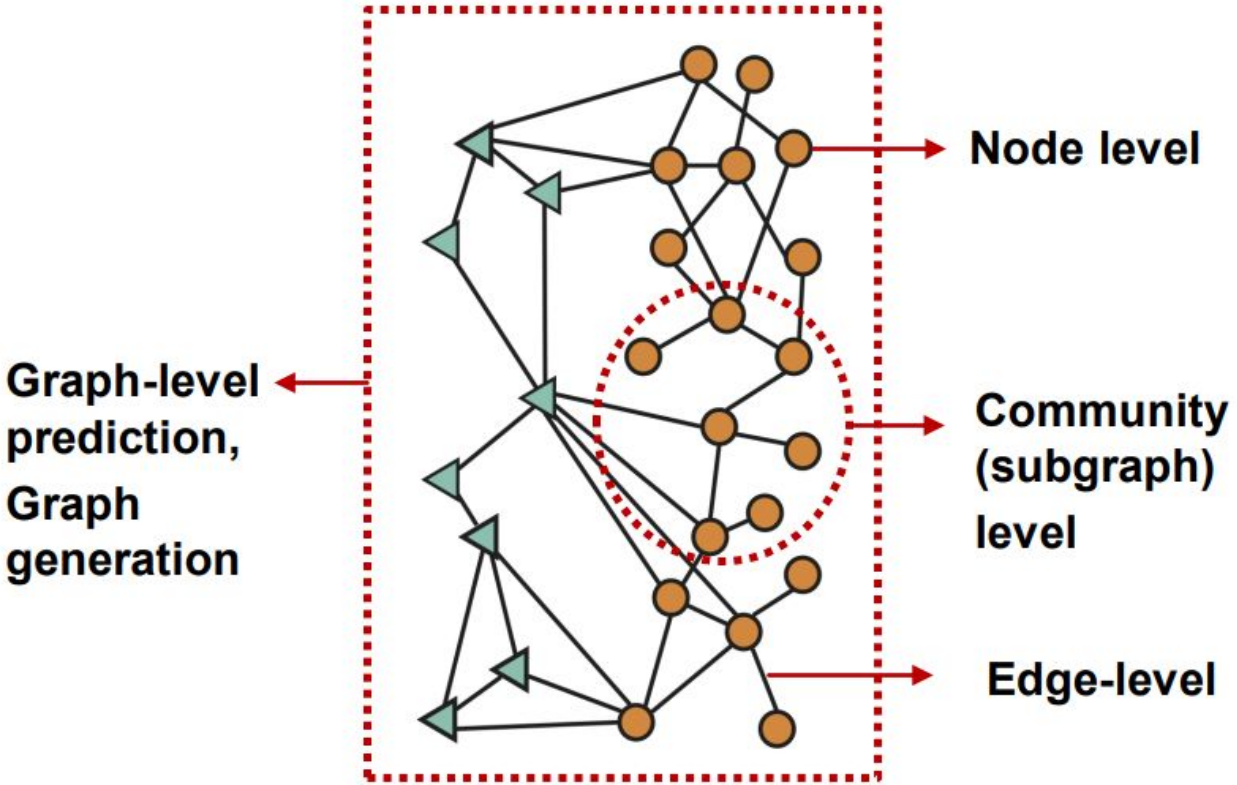


Special cases



Networks **Images** **Text**

Different types of ML tasks on graphs



Classic graph ML tasks

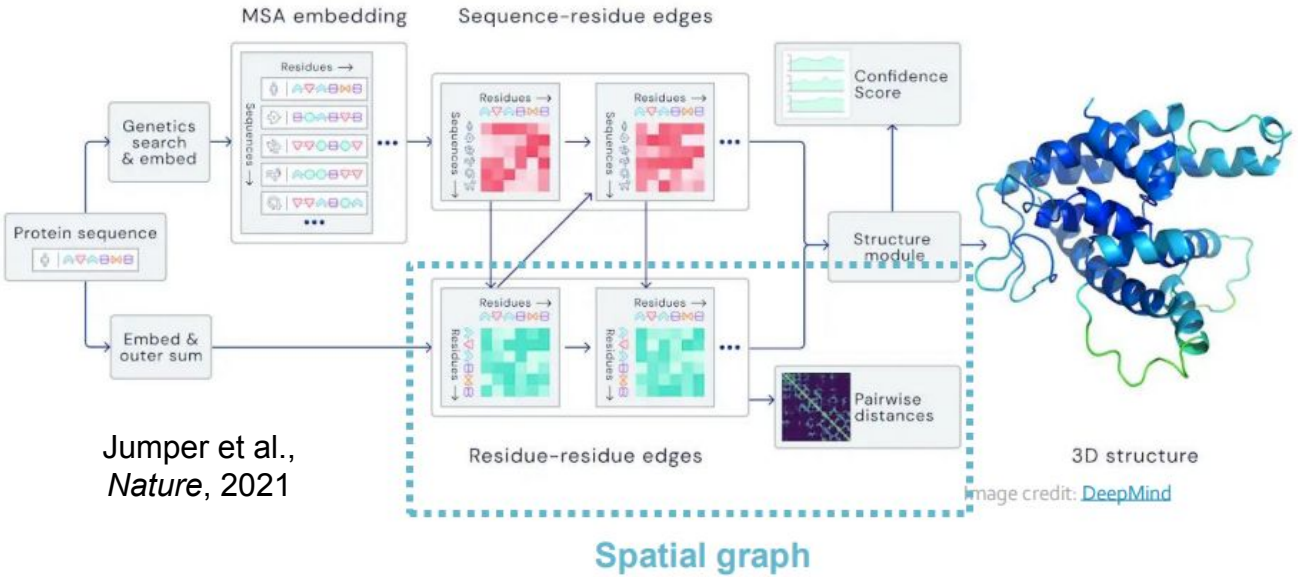
- **Node classification**: predict a property of a node
  - **Example**: Protein function prediction
- **Link prediction**: Predict whether a link exists between two nodes
  - **Example**: Recommendation (user <-> item), drug-target interaction prediction
- **Graph classification**: categorize different graphs
  - **Example**: Molecular property prediction
- **Clustering**: detect if nodes form a community
  - **Example**: Social circle detection, disease pathway detection
- **Graph generation**: generate new graphs
  - **Example**: drug design

These graph ML tasks lead to high-impact applications in real world

Example 1: Node-level ML tasks

Protein structure prediction (AlphaFold)

- **Nodes**: amino acids in a protein sequence
- **Edges**: proximity (distance) between amino acids (residues)

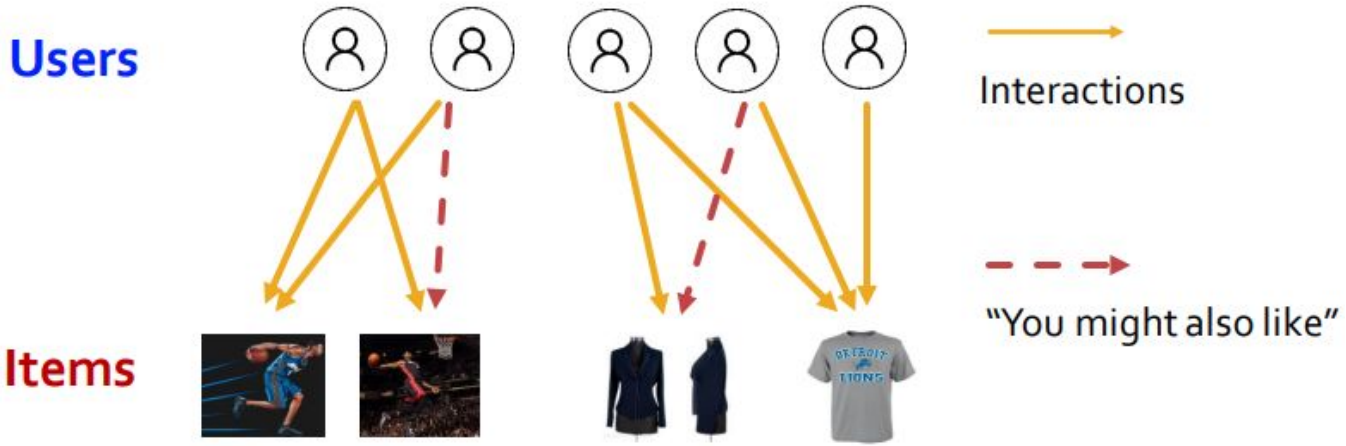


Jumper et al., *Nature*, 2021

Example 2: Edge-level ML tasks

Recommender Systems

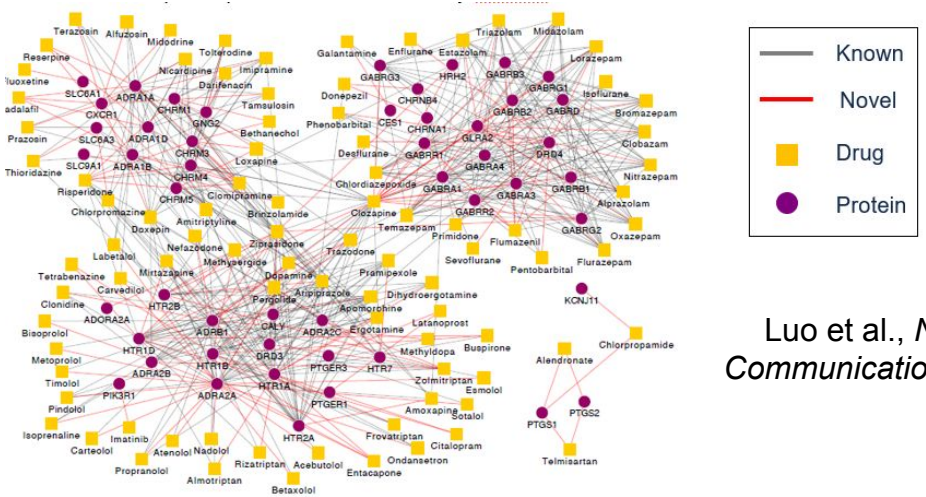
- **Nodes**: uses and items
- **Edges**: user-item interactions
- **Goal**: Recommend items users might like



Example 3: Edge-level ML tasks

Drug-target interaction

- **Nodes**: drugs and proteins (targets)
- **Edges**: drug-target interactions
- **Goal**: Predict unknown interactions between drugs and targets



Luo et al., *Nature Communications*, 2017

Example 4: Subgraph-level ML tasks

Traffic prediction

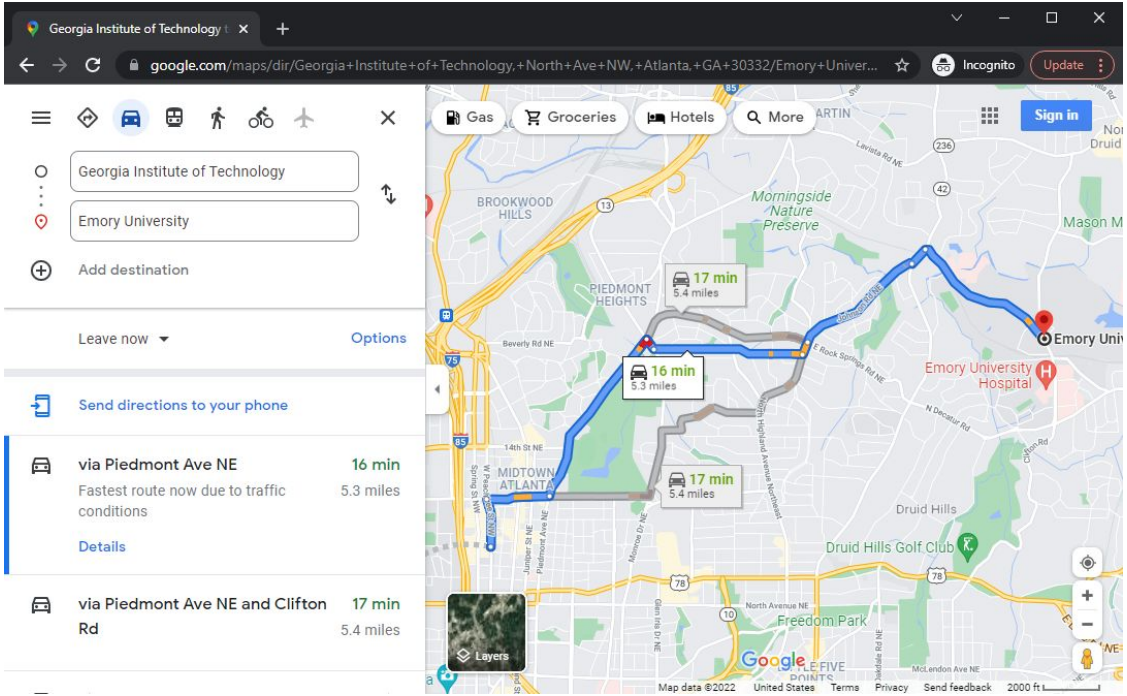
- **Nodes**: road segments
- **Edges**: connectivity between road segments
- **Goal**: predict time of arrival (ETA)

BLOG POST RESEARCH

03 SEP 2020

Traffic prediction with advanced Graph Neural Networks

DeepMind Blog

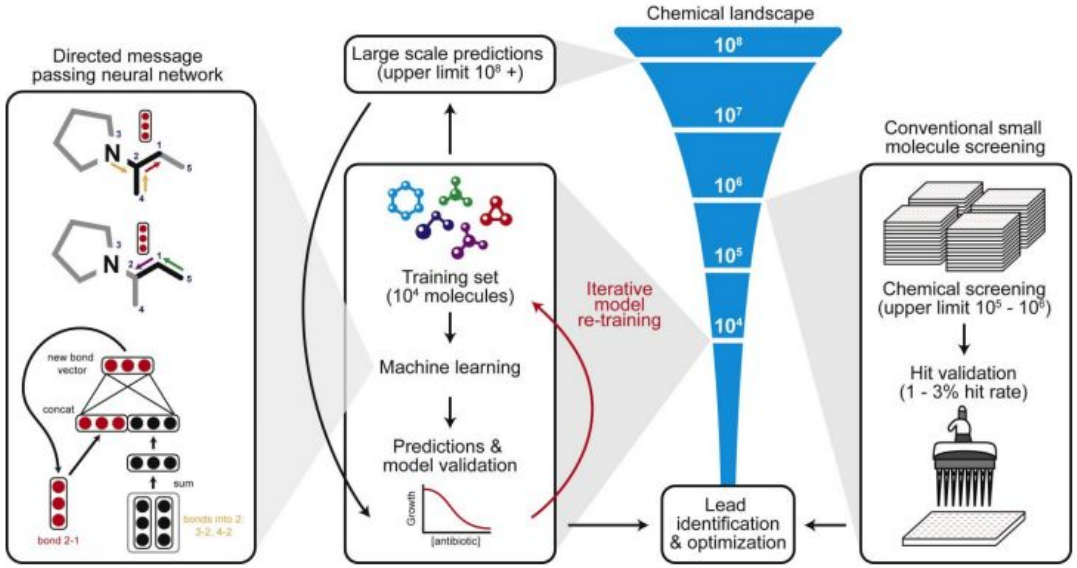


Deployed in Google Maps

Example 5: Graph-level ML tasks

Antibiotics discovery

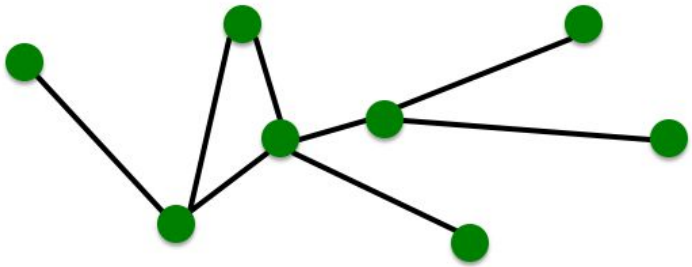
- **Nodes**: atoms
- **Edges**: bonds
- **Goal**: Predict novel antibiotics with a desired property



Stokes et al., *Cell*, 2020

What is a network (graph)?

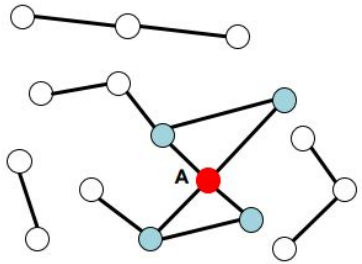
Graph: $G = (V, E)$



Objects: nodes, vertices	V
Interactions: links, edges	E
System: network, graph	$G(V, E)$

Node degree

Undirected

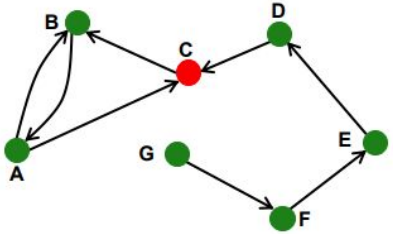


Node degree, k_i: the number of edges adjacent to node ***i***

$k_A = 4$

Avg. degree: $\bar{k} = \langle k \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i = \frac{2E}{N}$

Directed



Source: Node with $k^{in} = 0$
Sink: Node with $k^{out} = 0$

In directed networks we define an **in-degree** and **out-degree.**
The (total) degree of a node is the sum of in- and out-degrees.

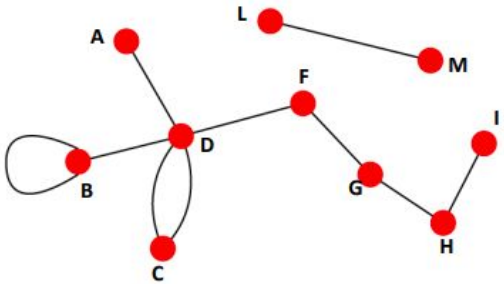
$k_C^{in} = 2 \qquad k_C^{out} = 1 \qquad k_C = 3$

$\bar{k} = \frac{E}{N}$ $\qquad\qquad$ $\overline{k^{in}} = \overline{k^{out}}$

Directed vs. Undirected Graphs

Undirected graphs

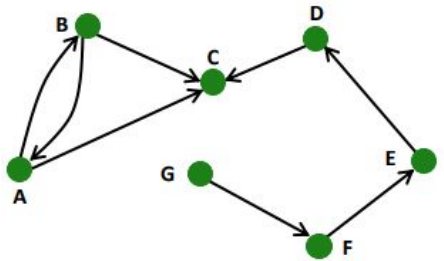
- **Links**: undirected (symmetrical, reciprocal)



- **Examples**:
  - Collaborations
  - Friendship on Facebook

Directed graphs

- **Links**: directed (arcs)



- **Examples**:
  - Paper citation
  - Flights between airports

Representing graphs: adjacency matrix

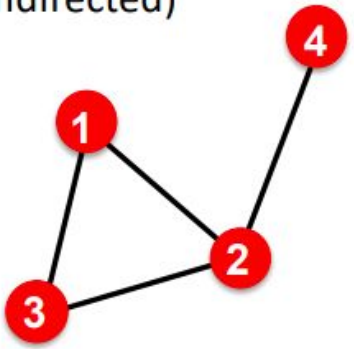
$A_{ij} = 1$ if there is a link from node i to node j

$A_{ij} = 0$ otherwise

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

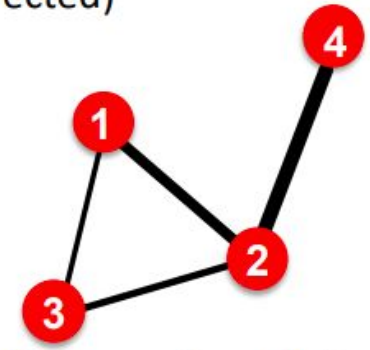
Weighted adjacency matrix

- **Unweighted**
  (undirected)



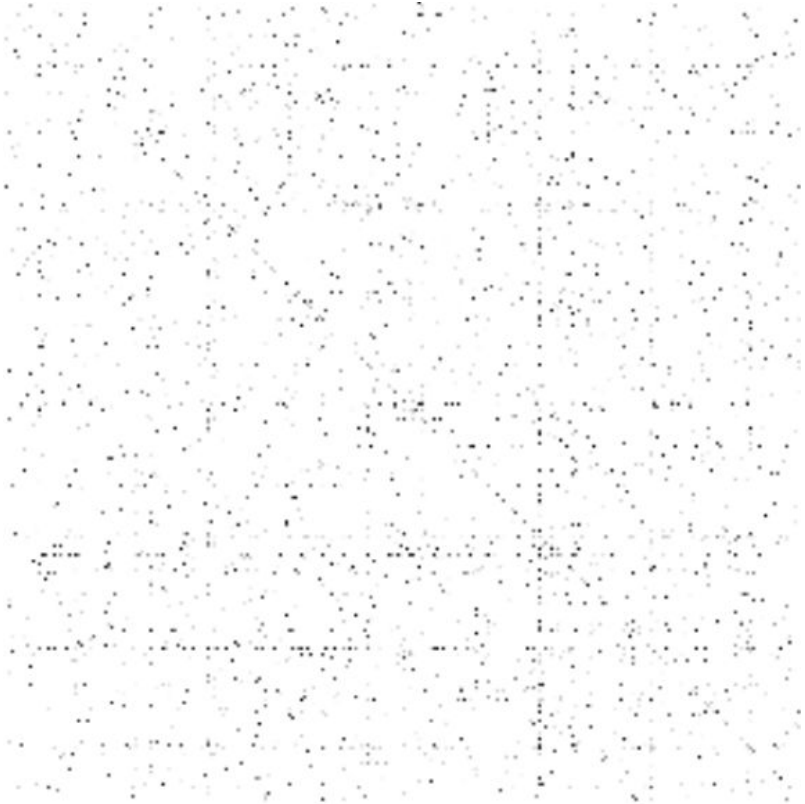
$$A_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

- **Weighted**
  (undirected)



$$A_{ij} = \begin{pmatrix} 0 & 2 & 0.5 & 0 \\ 2 & 0 & 1 & 4 \\ 0.5 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \end{pmatrix}$$

Adjacency matrices are sparse



Most real-world networks are sparse

Number of edges

Average degree

NETWORK	NODES	LINKS	DIRECTED/ UNDIRECTED	N	E	<k>
Internet	Routers	Internet connections	Undirected	192,244	609,066	6.33
WWW	Webpages	Links	Directed	325,729	1,497,134	4.60
Power Grid	Power plants, transformers	Cables	Undirected	4,941	6,594	2.67
Phone Calls	Subscribers	Calls	Directed	36,595	91,826	2.51
Email	Email Addresses	Emails	Directed	57,194	103,731	1.81
Science Collaboration	Scientists	Co-authorship	Undirected	23,133	93,439	8.08
Actor Network	Actors	Co-acting	Undirected	702,388	29,397,908	83.71
Citation Network	Paper	Citations	Directed	449,673	4,689,479	10.43
E. Coli Metabolism	Metabolites	Chemical reactions	Directed	1,039	5,802	5.58
Protein Interactions	Proteins	Binding interactions	Undirected	2,018	2,930	2.90

Consequence: Adjacency matrix is filled with zeros
(Density of the matrix $[E/N^2]$: WWW=1.51×10^{-5})

Demo: Python for network data using NetworkX

Google Colab

Outline

- Network (graph) data
  - Definitions & Examples
- Different graph ML tasks & features
  - Node-level
  - Edge-level
  - Graph-level
- Representations learning in graphs
  - Network embeddings

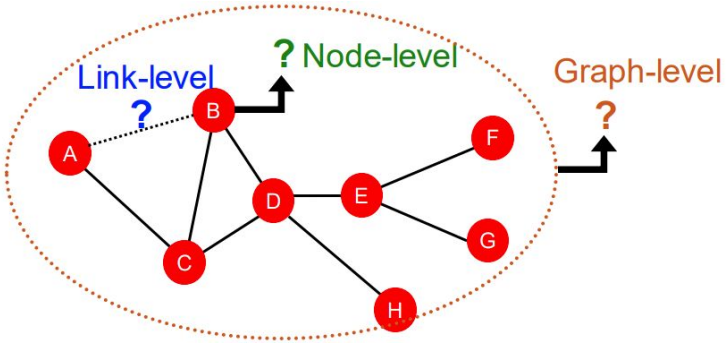
Traditional machine learning for graphs

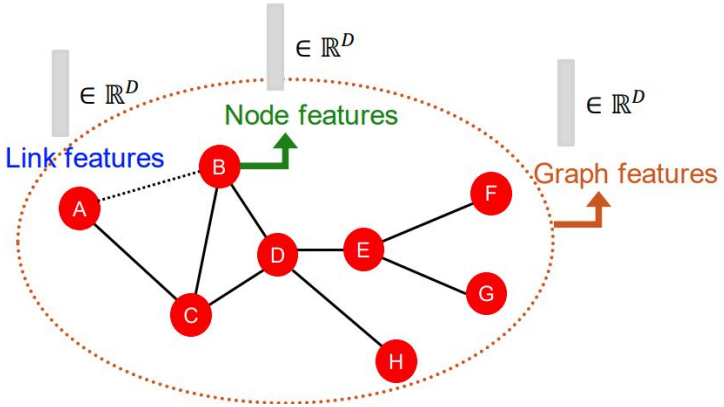
(for simplicity, we focus on undirected graphs in the lecture)

Machine learning tasks on graphs

- **Node-level** prediction
- **Link-level** prediction
- **Graph-level** prediction

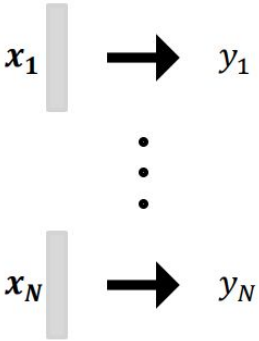
Design features for
nodes / links / graphs



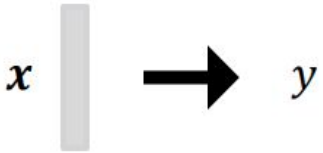


Traditional machine learning pipeline

- **Train an ML model**:
  - Obtain features of training data (node/edge/graph)
  - Train a ML model (SVM, NN, etc)



- **Apply the model**:
  - Given a new node/edge/graph, obtain its features and make a prediction

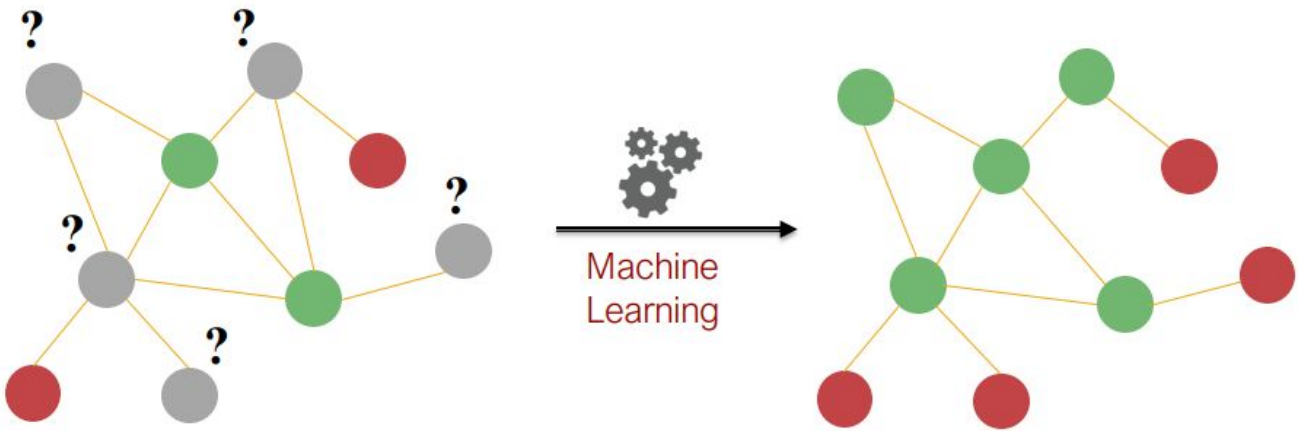


Traditional ML focuses on (manually) designing effective features over graphs

Node-level tasks and features

Node-level tasks

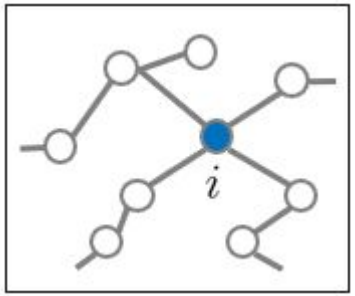
Node classification (e.g., protein function prediction)



Node-level features

Goal: design features that characterize the structure and position of a node in the network

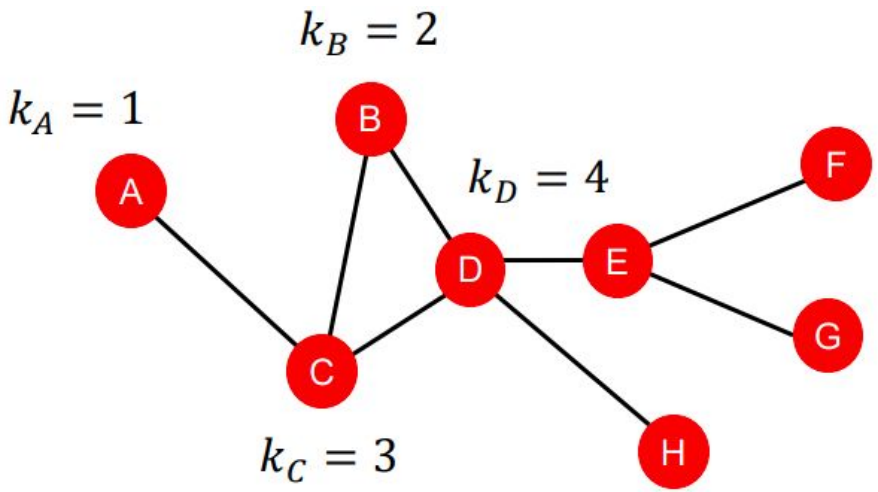
- Node degree
- Node centrality
- Clustering coefficient
- Graphlets



Node degree

Degree k_v of node v: the number of edges (neighboring nodes) the nodes has.

- Treats all neighboring nodes equally



Node centrality

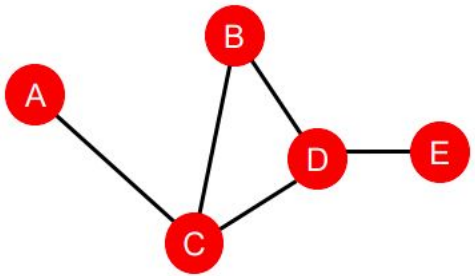
- **Node degree** counts the neighboring nodes without capturing their importance
- **Node centrality** c_v takes the node importance in a graph into account
- Different node centrality measures
  - Eigenvector centrality
  - Betweenness centrality
  - Closeness centrality
  - …

Node centrality: closeness centrality

- A node is important if it has small shortest path lengths to all other nodes

$$c_v = \frac{1}{\sum_{u \neq v} \text{shortest path length between } u \text{ and } v}$$

- Example:



$c_A = 1/(2+1+2+3) = 1/8$
(A-C-B, A-C, A-C-D, A-C-D-E)

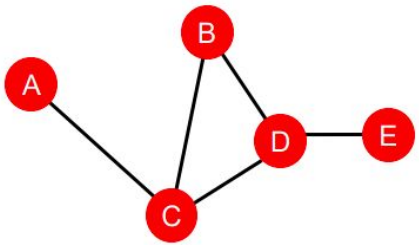
$c_D = 1/(2+1+1+1) = 1/5$
(D-C-A, D-B, D-C, D-E)

Node centrality: betweenness centrality

- A node is important if it lies on many shortest paths between other nodes

$$c_v = \sum_{s \neq v \neq t} \frac{\#(\text{shortest paths betwen } s \text{ and } t \text{ that contain } v)}{\#(\text{shortest paths between } s \text{ and } t)}$$

- **Unnormalized** version:



$$c_A = c_B = c_E = 0$$
$$c_C = 3$$
(A-C-B, A-C-D, A-C-D-E)

$$c_D = 3$$
(A-C-D-E, B-D-E, C-D-E)

- **Normalized** version:
  - Undirected graph
    - Normalized by (N-1)(N-2)/2
  - Directed graph
    - Normalized by (N-1)(N-2)
  - E.g.:

$$c_D = \frac{3}{\binom{N-1}{2}} = 0.5$$

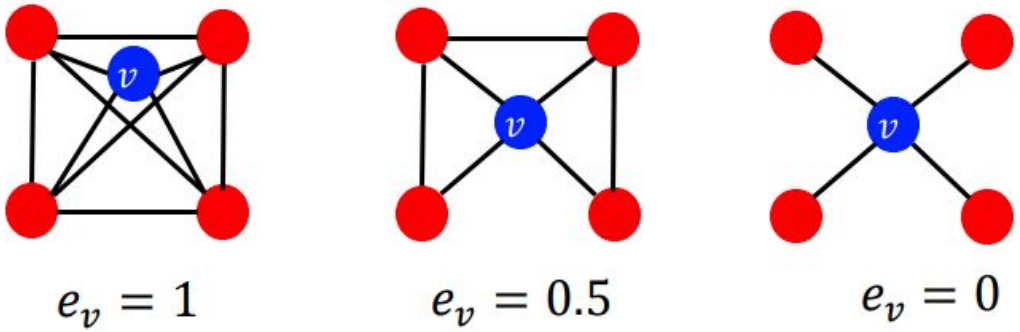
Clustering coefficient

- Measures how connected v's neighboring nodes are

$$e_v = \frac{\#(\text{edges among neighboring nodes})}{\binom{k_v}{2}} \in [0,1]$$

Total number of possible edges among ***v***'s neighboring nodes (***k_v***: degree of node ***k***)

- Example:



$e_v = 1$ $e_v = 0.5$ $e_v = 0$

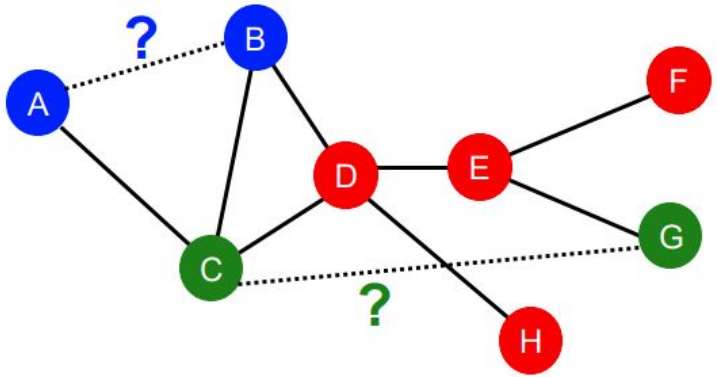
Node-level features: summary

- **Importance-based** features
  - Node degree
  - Node centrality
    - Betweenness centrality
    - Closeness centrality
- **Structure-based** features
  - Node degree
  - Clustering coefficient

Edge-level tasks and features

Recap: link-level prediction task

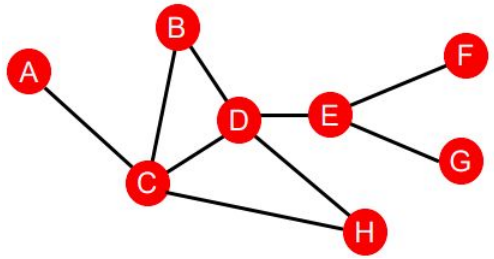
- **Link prediction**: predict new links based on existing links
- The key is to design feature for a pair of nodes



Distance-based features

Shortest-path distance between two nodes

- **Example**:



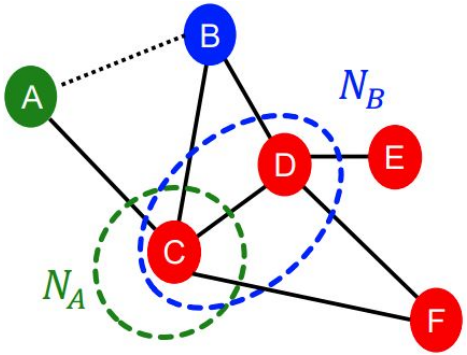
$$S_{BH} = S_{BE} = S_{AB} = 2$$
$$S_{BG} = S_{BF} = 3$$

- **Limitation**: does not capture the degree of neighborhood overlap
  - (B, H) have 2 shared neighbors, while (B, E) and (A, B) only have 1 such node

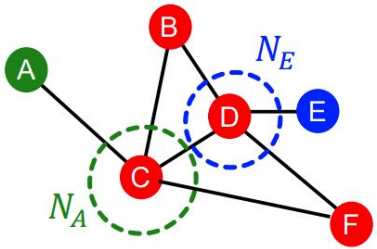
Local neighborhood overlap

#neighboring nodes shared between two nodes

- **Common neighbors:** $|N(v_1) \cap N(v_2)|$
  - Example: $|N(A) \cap N(B)| = |\{C\}| = 1$
- **Jaccard's coefficient:** $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$
  - Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{C,D\}|} = \frac{1}{2}$



- **Limitation:** Metric is always zero if the two nodes do not have any neighbors in common
  - However, the two nodes may still be connected in the future potentially



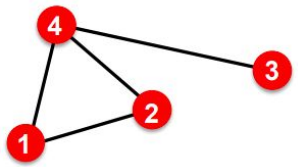
$N_A \cap N_E = \phi$
$|N_A \cap N_E| = 0$

Global neighborhood overlap

Katz index: #(walks) of all lengths between two nodes

- **Q**: how to compute #(walks) between two nodes?
- **A**: Use power of the graph adjacency matrix
  - **Recall**: $\boldsymbol{A_{uv}} = 1$ if $u \in N(v)$
  - Let $\boldsymbol{P_{uv}^{(K)}}$ = #walks of length $\boldsymbol{K}$ between $\boldsymbol{u}$ and $\boldsymbol{v}$
  - We will show $\boldsymbol{P^{(K)} = A^k}$
  - $\boldsymbol{P_{uv}^{(1)}}$ = #walks of length 1 (direct neighborhood) between u and $v = \boldsymbol{A_{uv}}$

$\boldsymbol{P_{12}^{(1)} = A_{12}}$



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Global neighborhood overlap

Katz index: #(walks) of all lengths between two nodes

- **How to compute $P_{uv}^{(2)}$?**
  - **Step 1:** Compute **#walks** of length 1 **between each of u's neighbor and v**
  - **Step 2**: **Sum up** these #walks across u's neighbors
  - $P_{uv}^{(2)} = \sum_i A_{ui} * P_{iv}^{(1)} = \sum_i A_{ui} * A_{iv} = A_{uv}^2$

Node 1's neighbors

#walks of length 1 between Node 1's neighbors and Node 2

$P_{12}^{(2)} = A_{12}^2$

$$A^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 3 \end{pmatrix}$$

Power of adjacency

Global neighborhood overlap

Katz index: #(walks) of all lengths between two nodes

- **Q**: how to compute #(walks) between two nodes?
- **A**: Use power of the graph adjacency matrix
  - $\boldsymbol{A_{uv}}$ specifies #(walks) of length 1 (direct neighbor) between ***u*** and ***v***
  - $\boldsymbol{A_{uv}^{(2)}}$ specifies #(walks) of length 2 (neighbor of neighbor) between ***u*** and ***v***
  - ***…***
  - $\boldsymbol{A_{uv}^{(L)}}$ specifies #(walks) of length *L* between ***u*** and ***v***

Global neighborhood overlap

Katz index: #(walks) of all lengths between two nodes

Sum over all walk lengths

$$S_{u,v} = \sum_{l=1}^{\infty} \beta^l A^l_{u,v}$$

#walks of length l between u and v

$0<\beta<1$: discount factor

- Katz index can be computed in closed-form:

$$\mathbf{S} = \sum_{l=1}^{\infty} \beta^i \mathbf{A}^i = \underbrace{(\mathbf{I} - \beta\mathbf{A})^{-1}}_{= \sum_{i=0}^{\infty} \beta^i \boldsymbol{A}^i} - \mathbf{I}$$

by geometric series of matrices

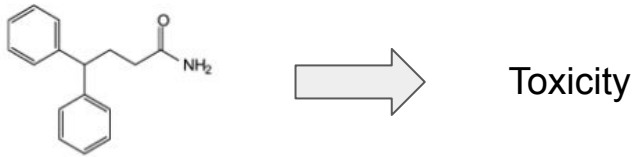
Summary of link-level features

- **Distance-based features**
  - Shortest-path distance between two nodes
    - Does not capture neighborhood overlap
- **Local neighborhood overlap**
  - #nodes shared between two nodes
    - Becomes 0 when no shared neighbors
- **Global neighborhood overlap**
  - Katz index: #walks of all lengths between two nodes
    - Captures global graph structure

Graph-level tasks and features

Graph-level tasks and features

- **Graph-level task**: predict property of the entire graph



- **Goal of graph-level feature**
  - We want features that characterize the structure of an entire graph
- **Examples**:
  - Traditional methods: Graph kernels (measure similarity between two graphs)
  - Learning-based methods: graph neural networks (GNNs)

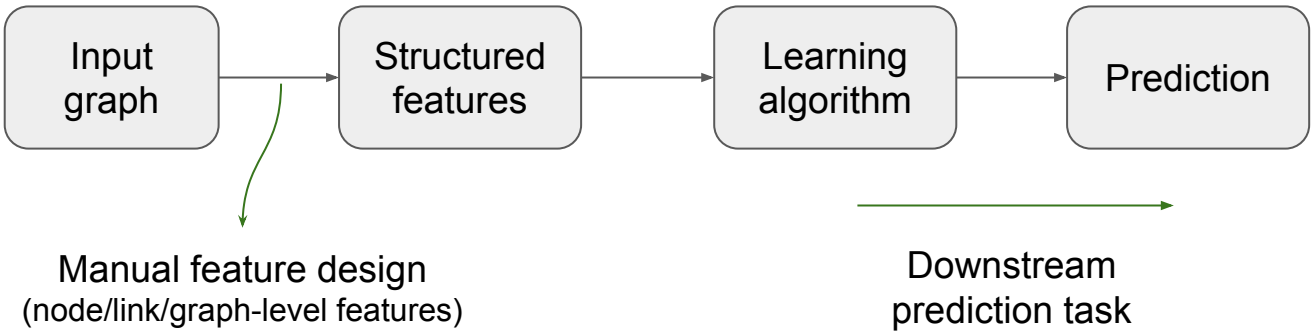
Outline

- Network (graph) data
  - Definitions & Examples
- Different graph ML tasks & features
  - Node-level
  - Edge-level
  - Graph-level
- Representations learning in graphs
  - Network embeddings

Graph representation learning

Traditional ML for graphs

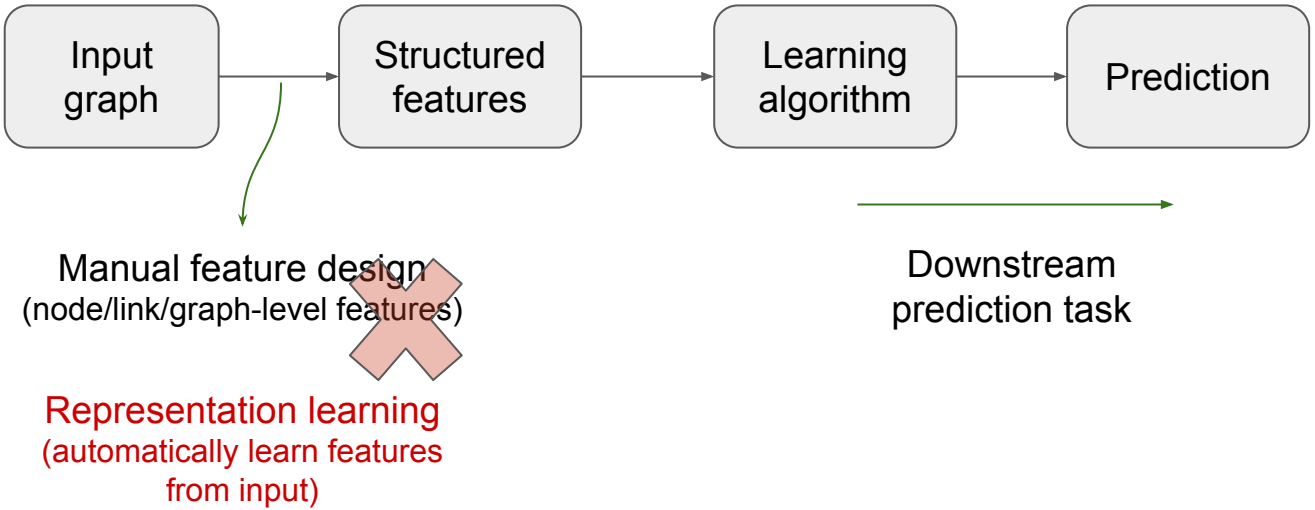
- Given an input graph, extract node, link and graph-level features, learn a model (SVM, neural network, etc.) that maps features to labels.



Adapted from CS224W, Jure Leskovec

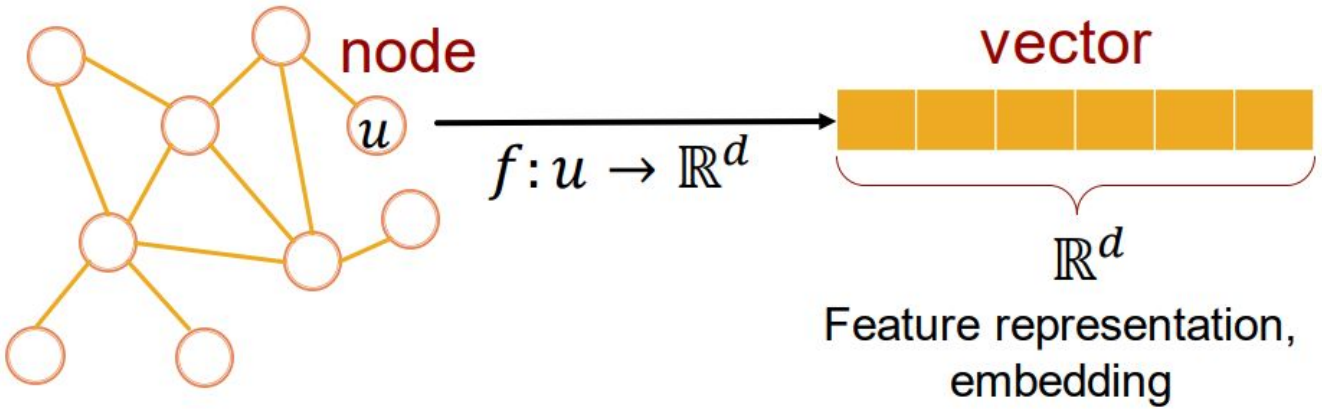
Graph representation learning

- Graph representation learning: learn a feature for each node from the graph input automatically



Graph representation learning

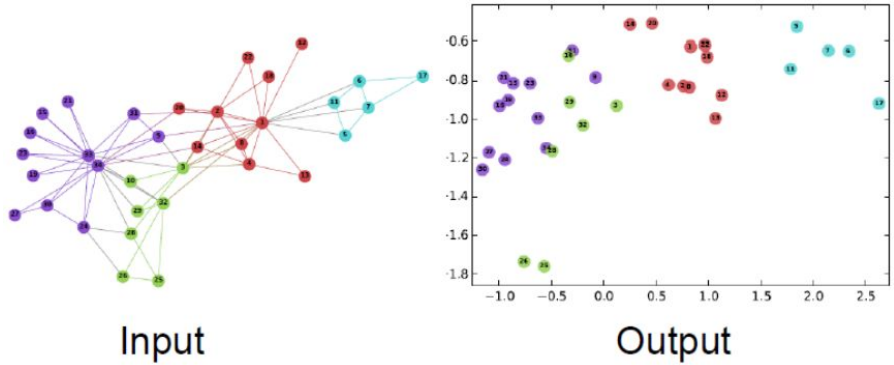
- **Goal**: Efficient task-independent feature learning for machine learning with graphs



Adapted from CS224W, Jure Leskovec

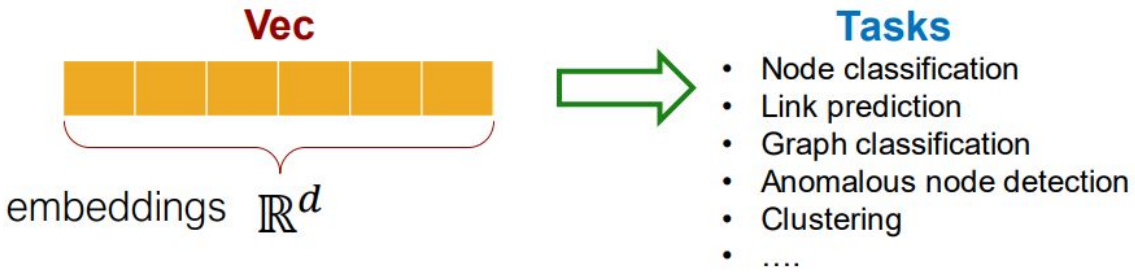
Why embedding?

- Similarity of embeddings between nodes indicates their similarity in the network.



DeepWalk: Online Learning of Social Representations. KDD 2014

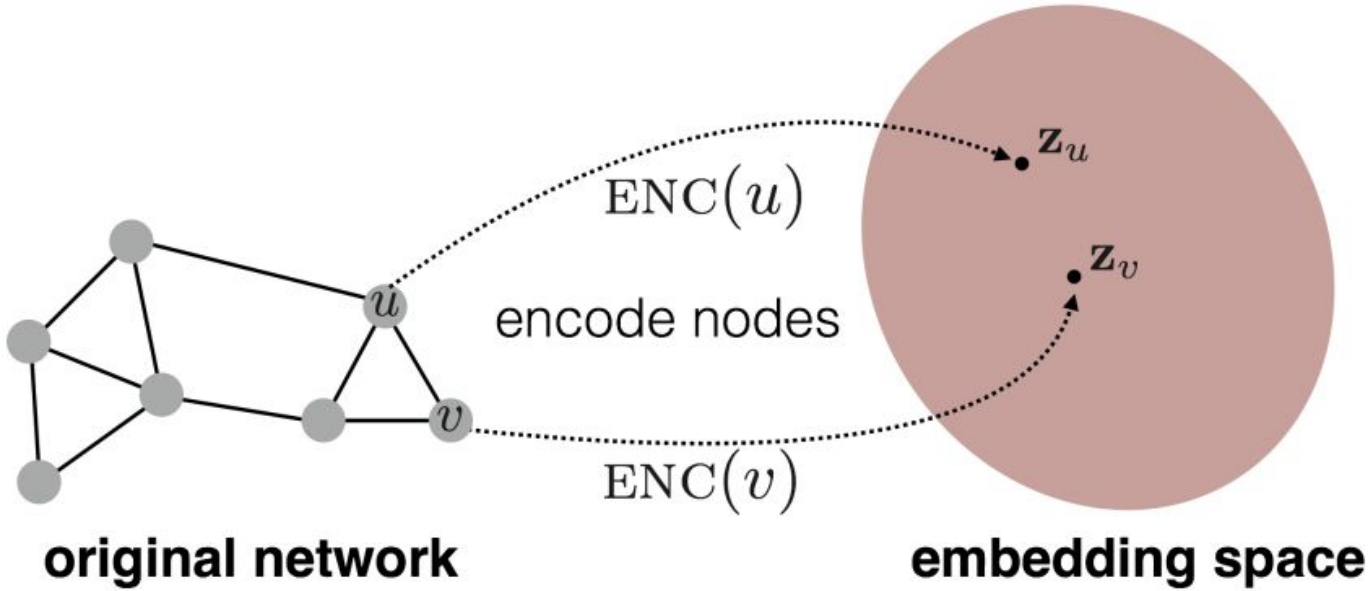
- Potentially used for many downstream predictions



Adapted from CS224W, Jure Leskovec

Embedding nodes

- **Goal**: encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph



Adapted from CS224W, Jure Leskovec

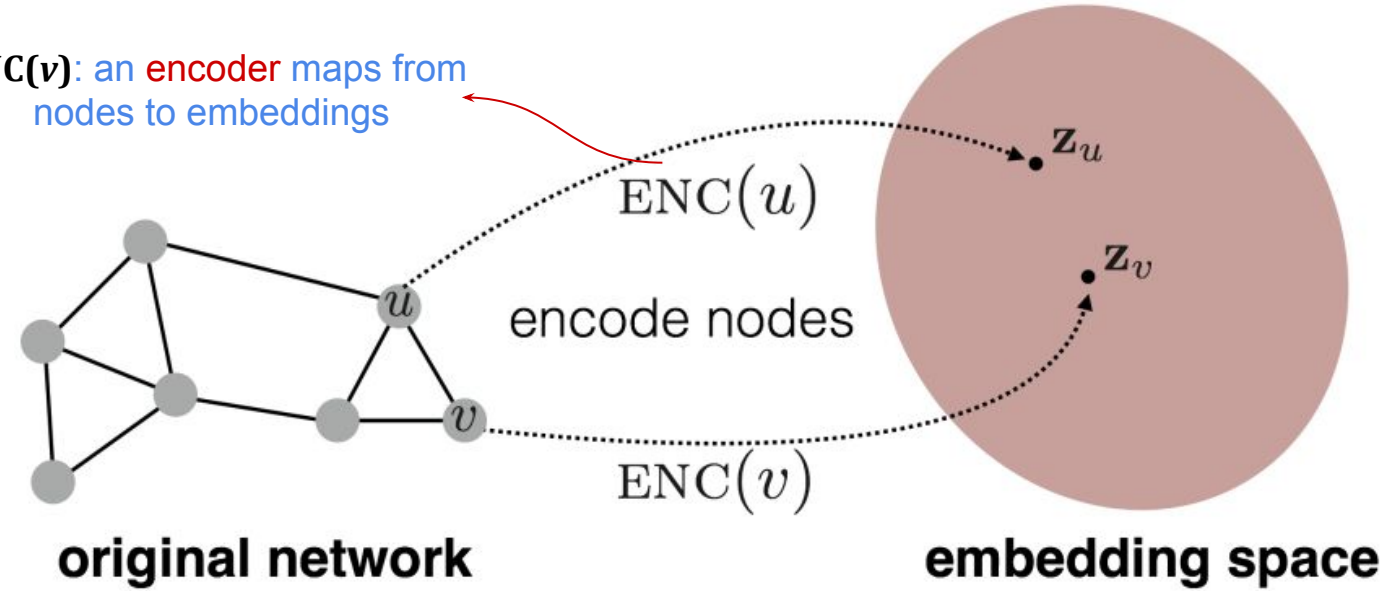
Embedding nodes

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

In the original network

Similarity of the embeddings

ENC(v): an encoder maps from nodes to embeddings



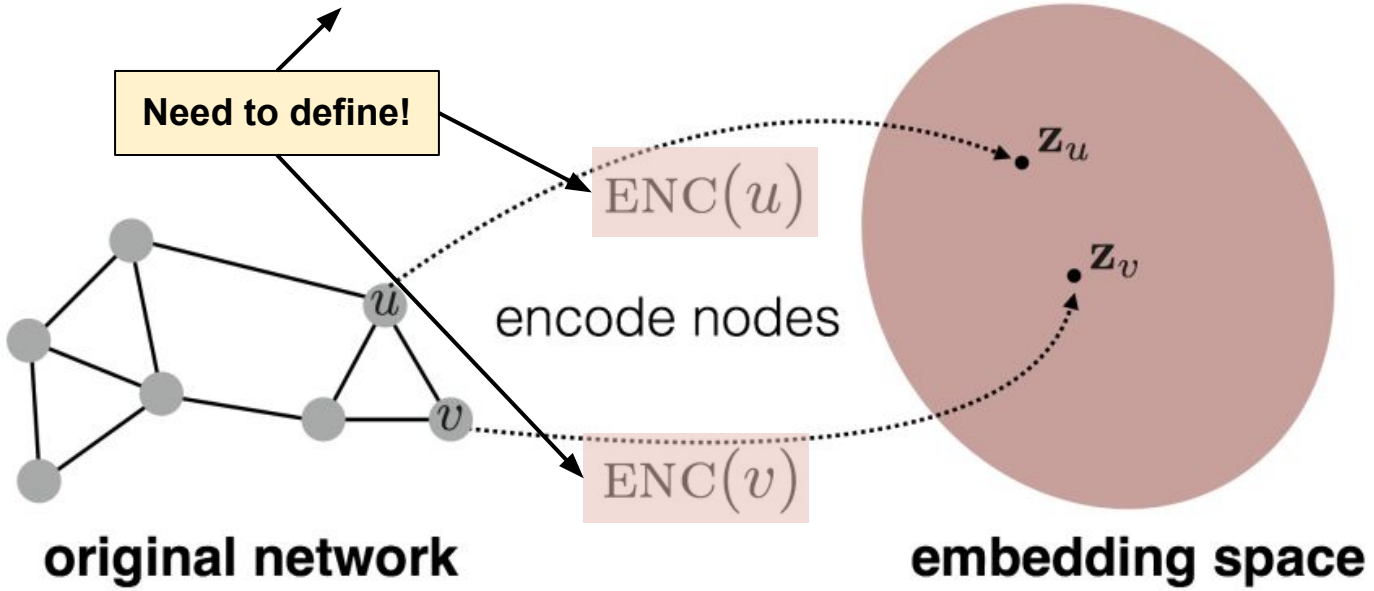
Adapted from CS224W, Jure Leskovec

Embedding nodes

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

In the original network

Similarity of the embeddings



Adapted from CS224W, Jure Leskovec

Two key components

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

$$\mathrm{ENC}(\boldsymbol{v}) = \mathbf{Z}_v$$

$\mathbf{Z}_v$: *d*-dimensional vector (embedding)

$\boldsymbol{v}$: Node in the input graph

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

$$\text{similarity}(u, v) \approx \mathbf{Z}_v{}^{\mathrm{T}}\mathbf{Z}_u$$

$\text{similarity}(u, v)$: Similarity of $\boldsymbol{u}$ and $\boldsymbol{v}$ in the original network

$\mathbf{Z}_v{}^{\mathrm{T}}\mathbf{Z}_u$: Dot product between node embeddings

Adapted from CS224W, Jure Leskovec

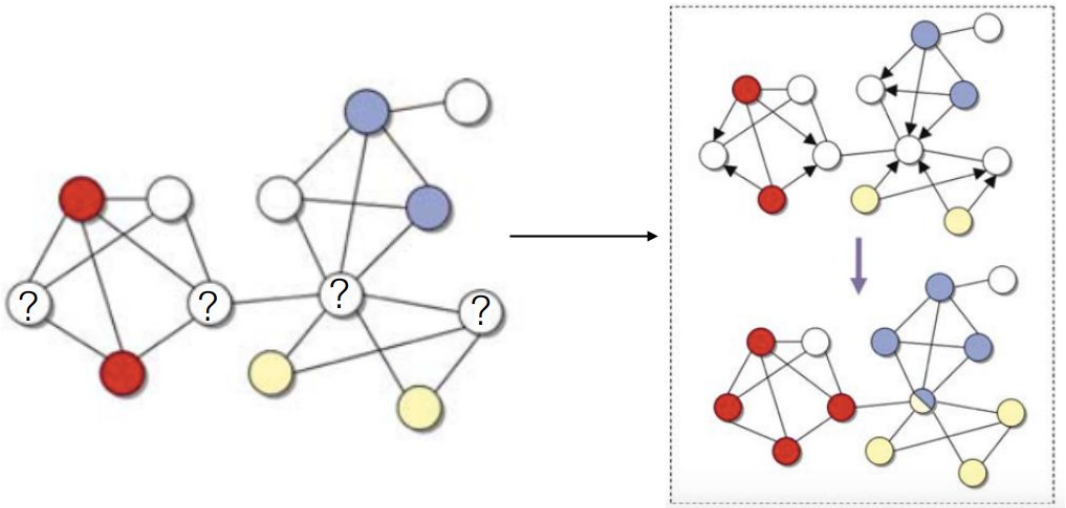
How to define node similarity in the network?

- Key choice of methods is how to define node similarity.
- Should two nodes have a similar embedding if they …
  - are linked?
  - share neighbors?
  - …
- This lecture: define node similarity based "topological roles" of each node with respect to other nodes.
- Two graph representation learning algorithms:
  - This lecture: Diffusion component analysis [DCA] (Cho et al, 2016, *Cell Systems*)
  - After-class reading: Node2vec (Grover et al, 2016, *KDD*)

Adapted from CS224W, Jure Leskovec

Motivating example

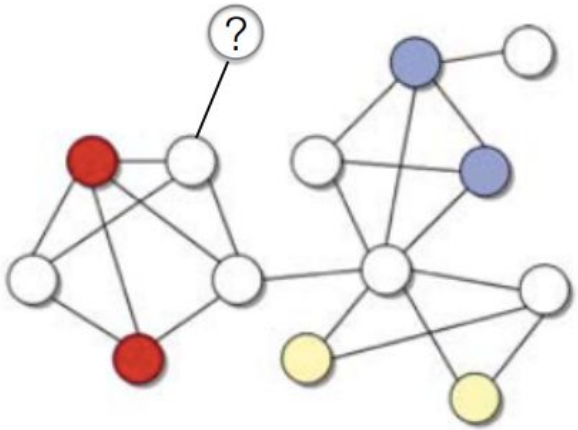
Example: protein function prediction



Voting by direct neighbors

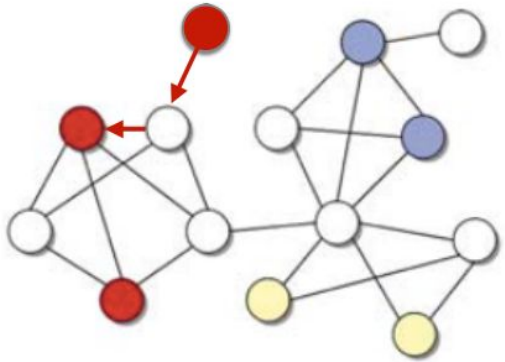
Adapted from CS598, Jian Peng

If there is no direct neighbor with known function



Adapted from CS598, Jian Peng

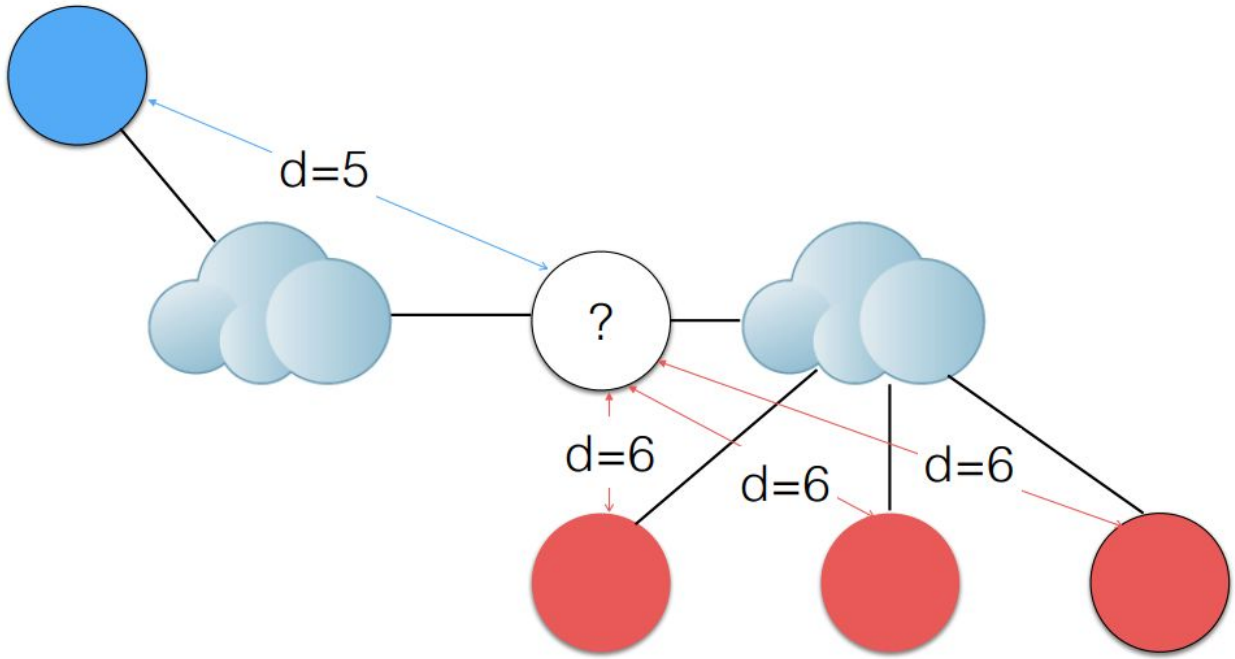
Shortest path



Floyd-Warshall algorithm: all pairwise distances
Computational Complexity: $O(n^3)$

Adapted from CS598, Jian Peng

Is shortest path a good metric?



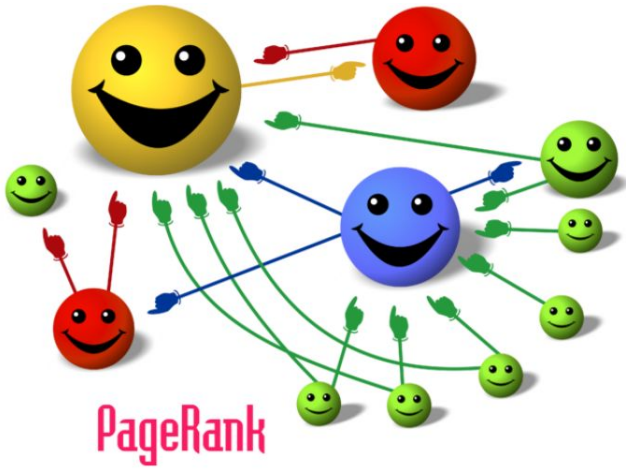
Adapted from CS598, Jian Peng

Diffusion component analysis (DCA)

[1] Cho, Hyunghoon, Bonnie Berger, and Jian Peng. "Diffusion component analysis: unraveling functional topology in biological networks." *International Conference on Research in Computational Molecular Biology*. Springer, Cham, 2015.

[2] Cho, Hyunghoon, Bonnie Berger, and Jian Peng. "Compact integration of multi-network topology for functional analysis of genes." *Cell systems* 3.6 (2016): 540-548.

Random walk and Pagerank



Article Talk Re

PageRank

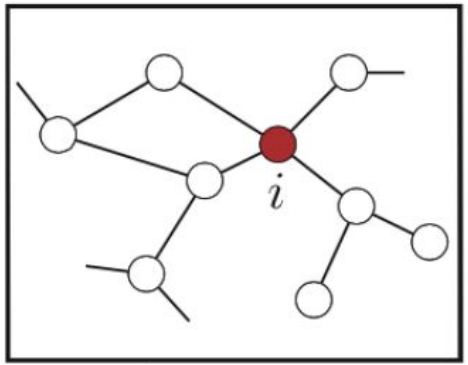
From Wikipedia, the free encyclopedia

PageRank (**PR**) is an algorithm used by Google Search to rank web pages in their search engine results. It is named after both the term "web page" and co-founder Larry Page. PageRank is a way of measuring the importance of website pages. According to Google:

> PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.[1]

Random walk with restart

Start from node i

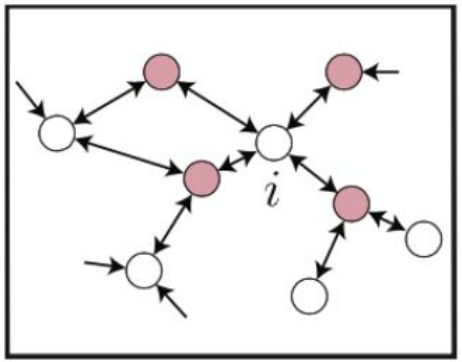


Initialization

$$s_i = (0, 0, ..., 1, ..., 0)$$

Random walk with restart

Distribute to neighbors



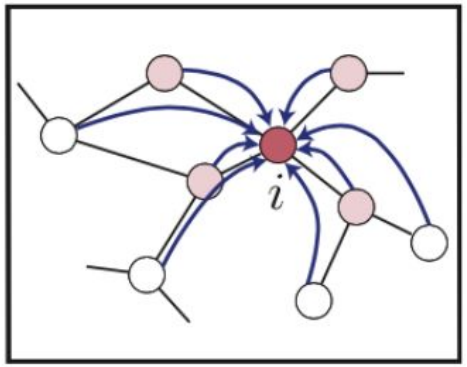
four neighbors

Propagation

$$s_i = (0, ..., 0.25, ..., 0.25, ..., 0.25, ..., 0.25, ..., 0)$$

Random walk with restart

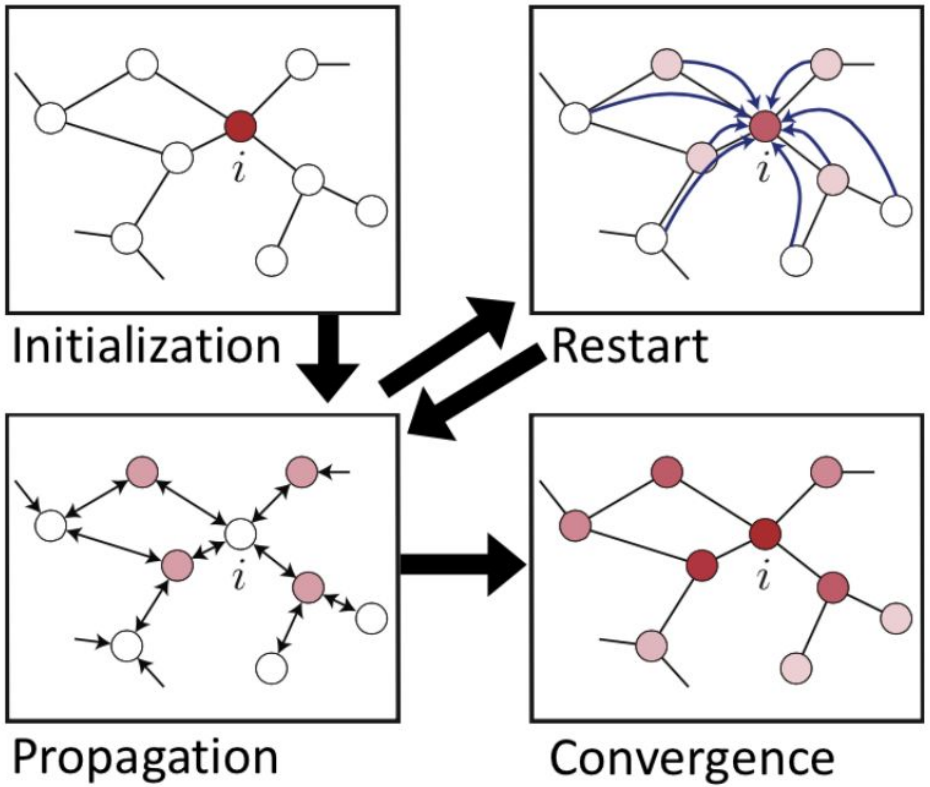
Shinkage and restart from node i



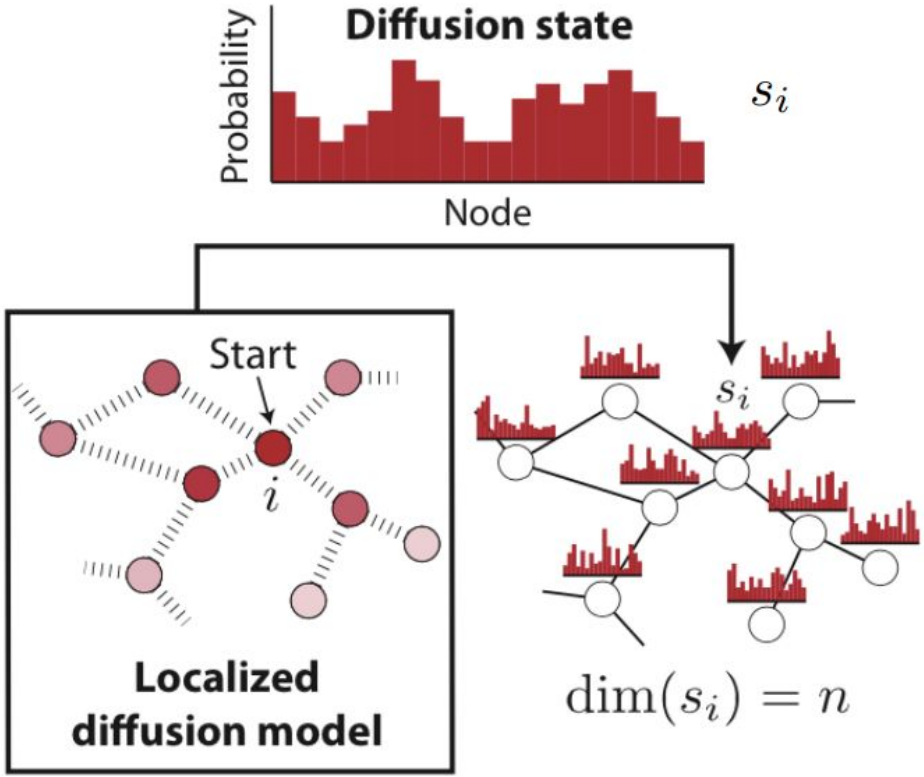
Restart

$$s_i = (0, ..., 0.125, ..., 0.125, ..., 0.5, ..., 0.125, ..., 0.125, ..., 0)$$

Random walk with restart



Random walk with restart



Random walk with restart

Adjacency matrix $A : A_{ij}$

Transition matrix $B \in R^{n \times n} : B_{ij} = A_{ij} / \sum_k A_{ik}$

Restart probability p

Algorithm: Repeat

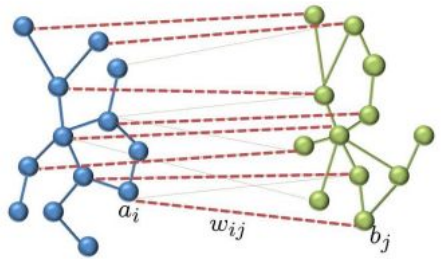
$$\forall i, j \qquad s_i(j) = (1-p) \sum_k s_i(k) B_{kj} + p\delta(i=j)$$

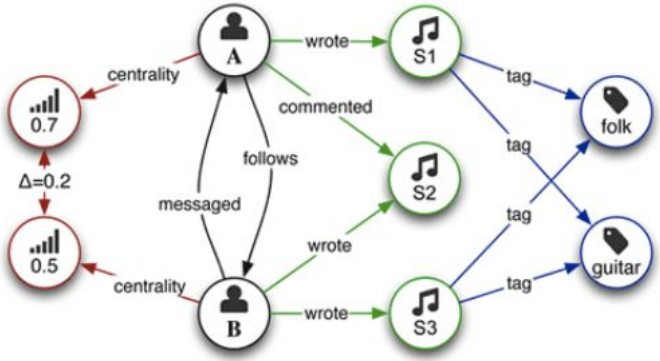
Matrix operation

$$S^{new} = (1-p) S^{old} B + p I_n$$

Random walk with restart

- **Comments**:
  - Very simple implementation
  - Capture long-range relationship in the graph
  - Robust to missing edges
  - Many applications in social network analysis, web data analysis, and bioinformatics





Weighted Voting

Assign color vector: $\forall i \in RED, f_{red}(i) = 1$

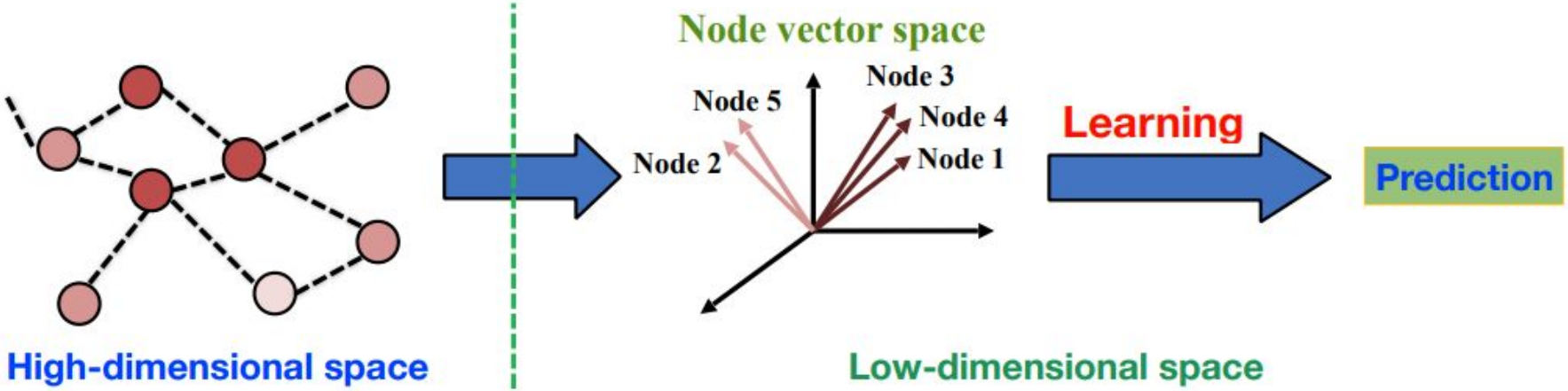
$\forall i \notin RED, f_{red}(i) = 0$

Majority voting $p_{red}(i) = \sum_j Sim(i,j) f_{red}(j)$

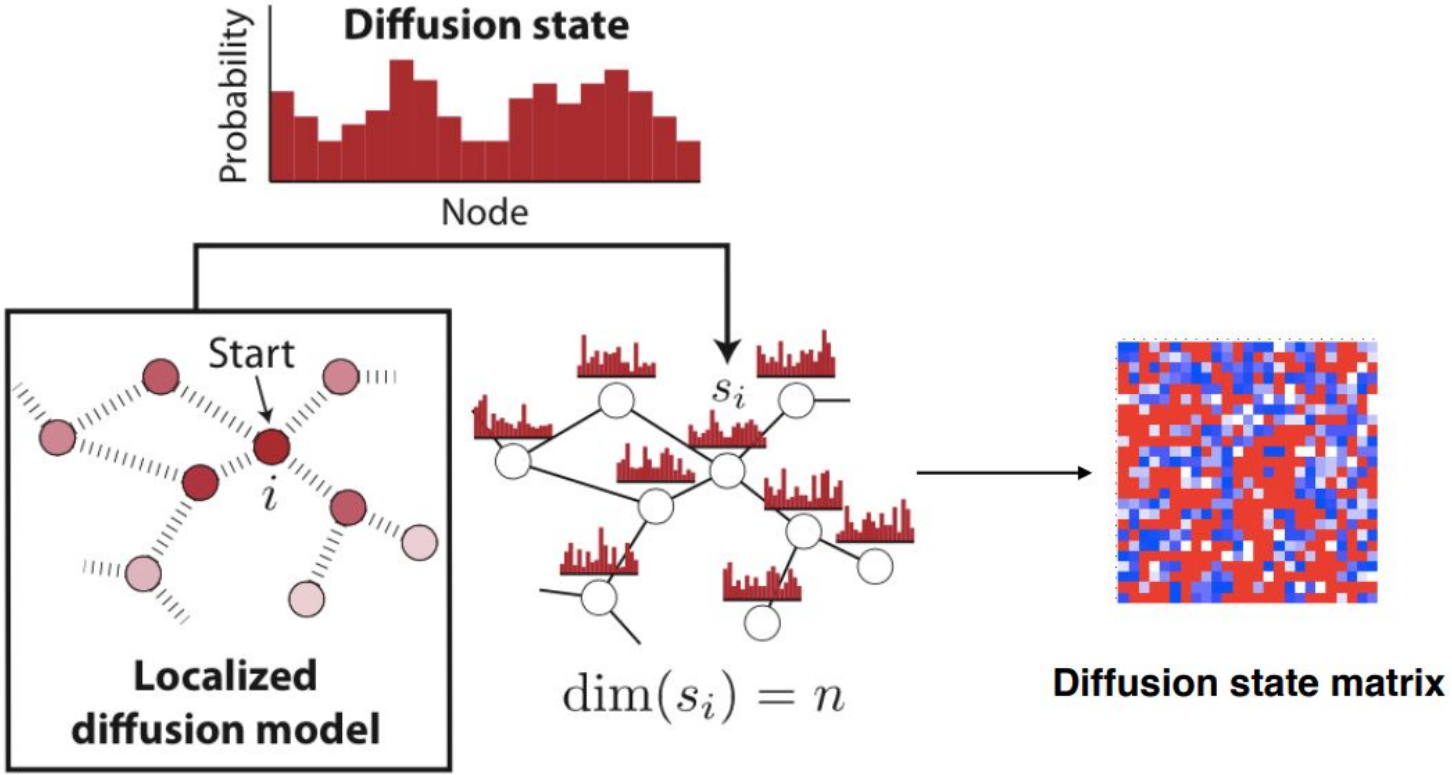
Some good similarity metrics:

1. reciprocal of shorted distance $1/d(i,j)$
2. random walk similarity $s_i(j) + s_j(i)$
3. diffusion state similarity $1/\|s_i - s_j\|_1$

Dimensionality reduction

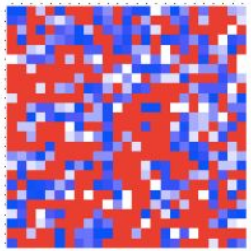


After we run random walk with restart



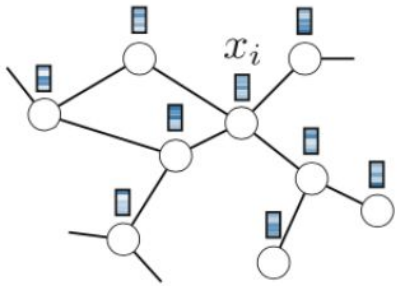
We can run SVD or NMF on *S*

- Let us assume that we don't know how this diffusion state matrix was generated



Diffusion state matrix

Low-dimensional representation



$$\dim(x_i) = d \ll n$$

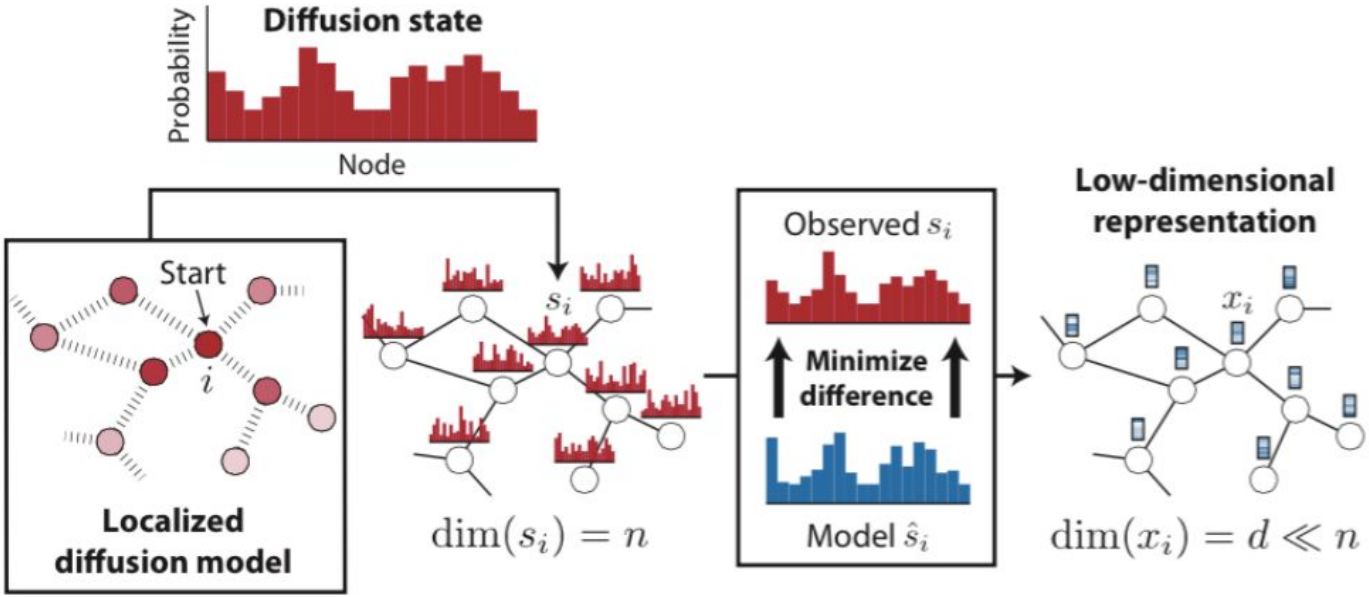
Any issues with these dimensionality reduction algorithms?

$$\min_{X,W} \|S - XW\|_2^2$$

$$X \in R^{n \times d}, W \in R^{d \times n}$$

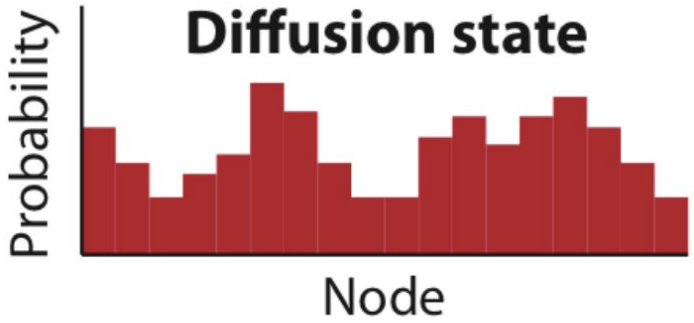
This paper: Diffusion component analysis (DCA)

- **Idea**: we hope to construct a model to approximate observed diffusion states



After random walk (diffusion)

- $s_i(j)$ gives the probability of reaching node j after the random walk from node i

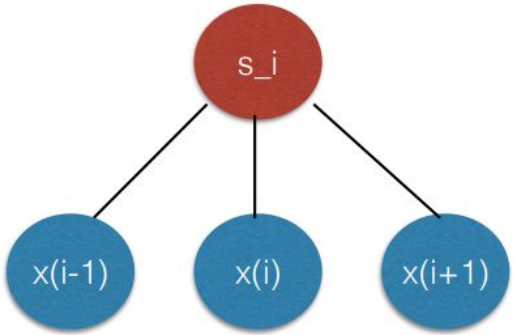


$$\sum_j s_i(j) = 1$$

$$\forall i, j \qquad s_i(j) \geq 0$$

What would be a good way to model diffusion states?

Logistic regression



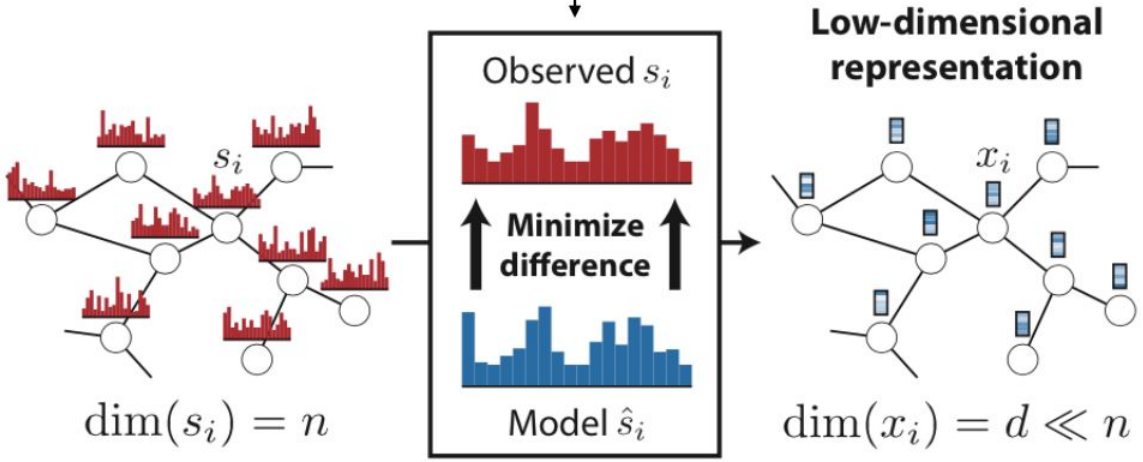
$$\hat{s}_i(j) = \frac{\exp(w_i^T x_j)}{\sum_k \exp(w_i^T x_k)}$$

Is it better than SVD/NMF?

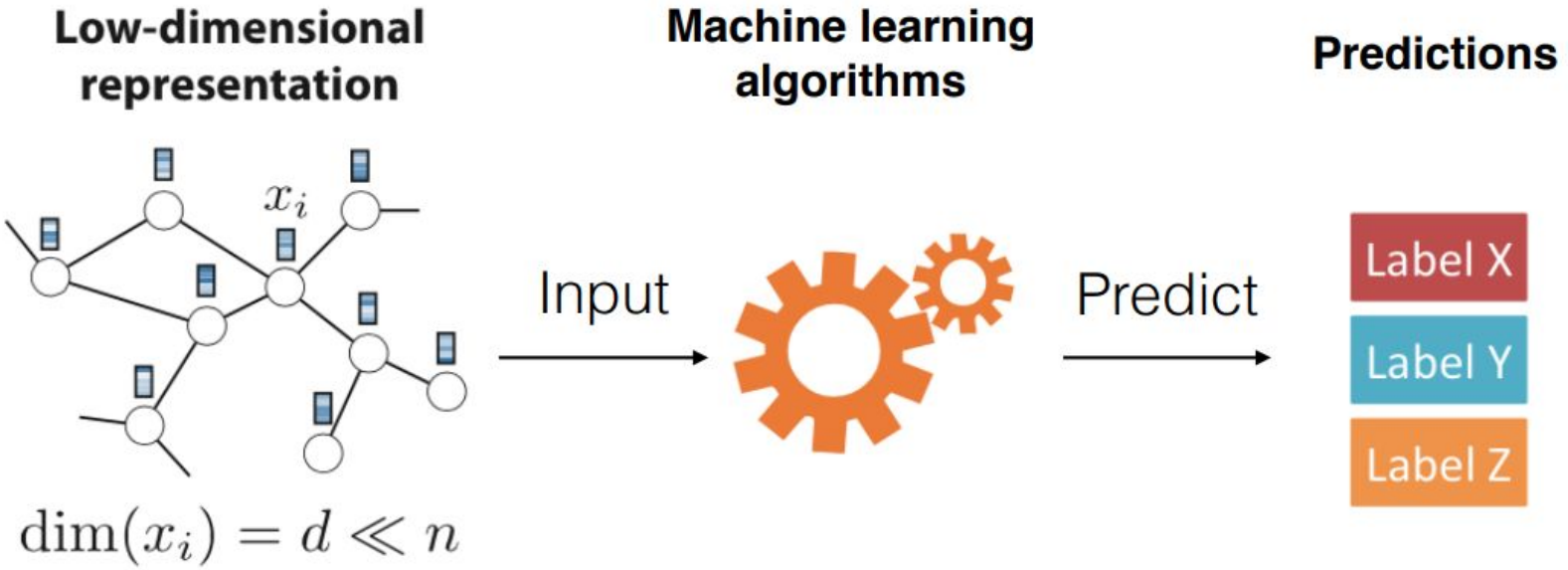
Matching observed and model data

$$\min_{w,x} \sum_i KL(s_i \| \hat{s}_i)$$

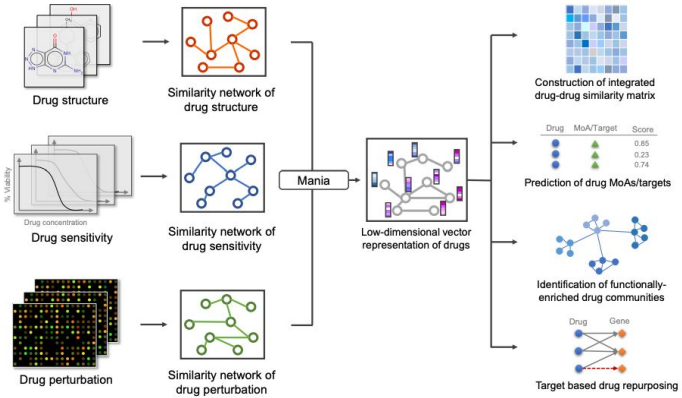
with $$\hat{s}_i(j) = \frac{\exp(w_i^T x_j)}{\sum_k \exp(w_i^T x_k)}$$



Machine learning with network data

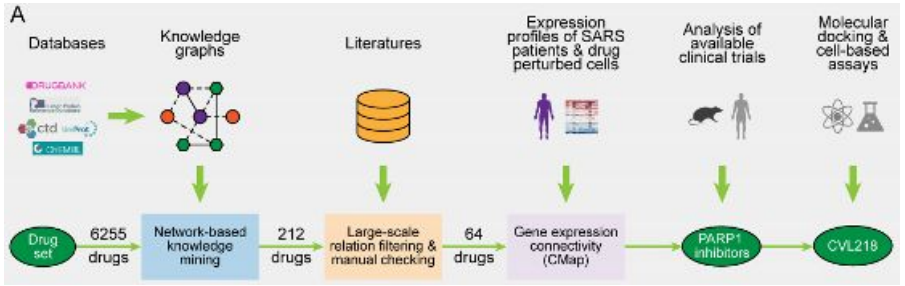


Application of PageRank: Drug Discovery



Predict new protein targets for existing drugs

Luo et al., *Nature Communications*, 2017



Predict potential drugs for COVID-19

Ge et al. *Signal Transduction and Targeted Therapy*, 2021

Analogy to PCA

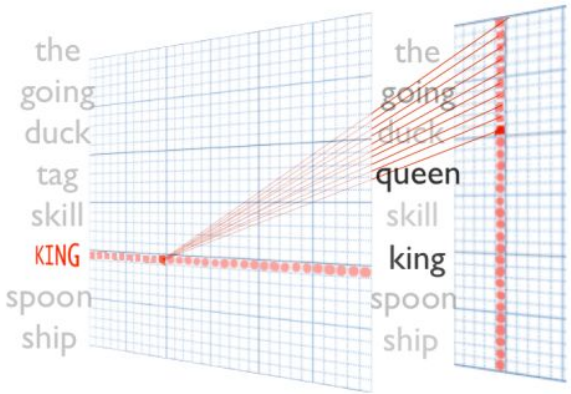
Principal component analysis (PCA)

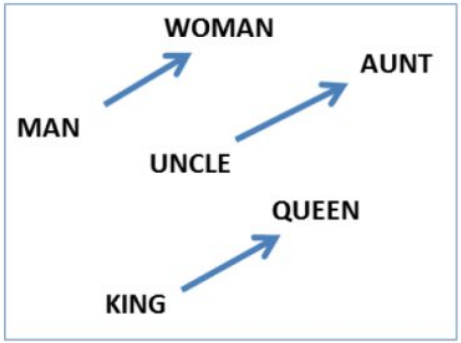
- Input: **matrix** data
- Goal: find low-rank approximation that best explain the **variance** of the matrix input

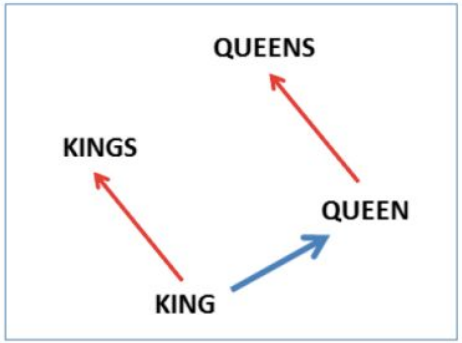
Diffusion component analysis (DCA)

- Input: **network** data
- Goal: find low-dim representations that best explain the **topological pattern** of the network input

Similarity to word2vec





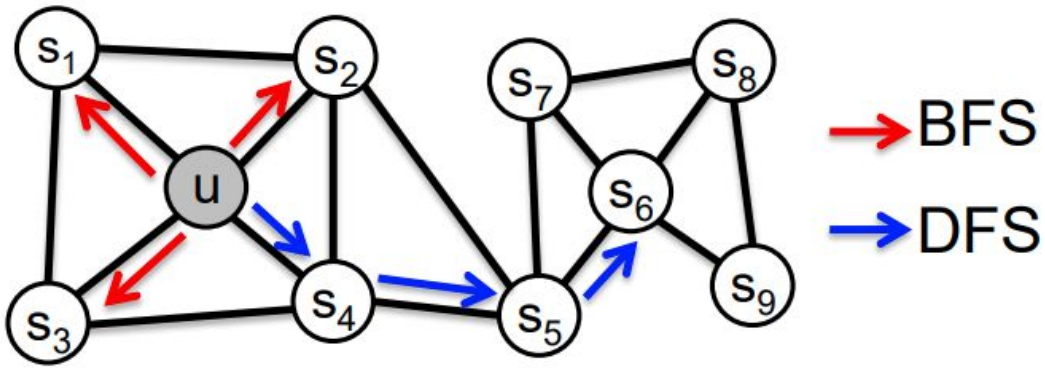


Reading after this class: node2vec

Grover et al. “node2vec: Scalable Feature Learning for Networks”. *KDD*, 2016

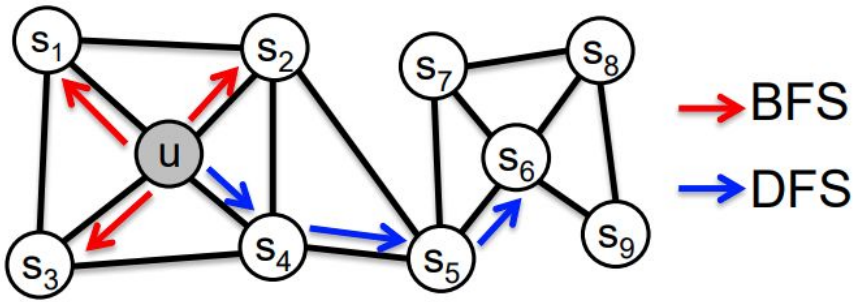
node2vec

- **Goal**: Embed nodes with similar network neighborhoods close in the feature space
- **Idea**: Use flexible, **biased random walks** that can trade off between **local** and **global** views of the network



Biased walks

Two classic strategies to define a neighborhood ***N(u)*** of a given node ***u***

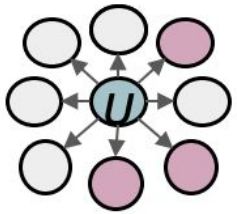


Walk of length 3 (***N(u)*** of size 3):

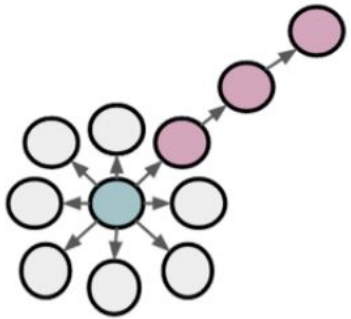
$$N_{BFS}(u) = \{ s_1, s_2, s_3 \}$$ Local microscopic view

$$N_{DFS}(u) = \{ s_4, s_5, s_6 \}$$ Global microscopic view

BFS & DFS



BFS:
Micro-view of neighbourhood



DFS:
Macro-view of neighbourhood

How to interpolate BFS & DFS?

Biased fixed-length random walk ***R*** that, given a node ***u***, generates neighborhood $N_R(u)$

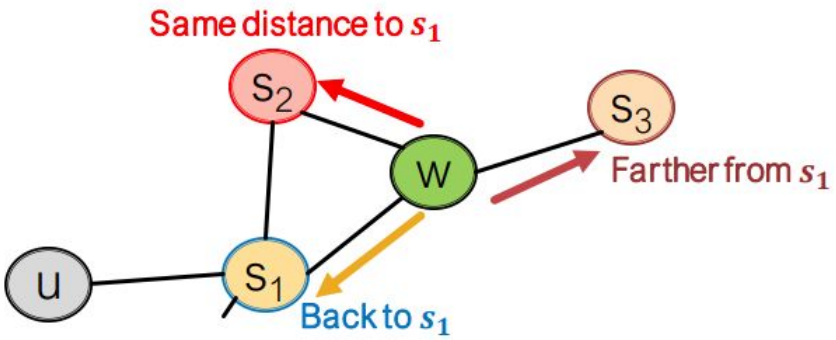
Two parameters:

- Return parameter ***p***:
  - Return back to the previous node
- In-out parameter ***q***:
  - Moving outwards (DFS) vs. inwards (BFS)
  - Intuitively, ***q*** is the “ratio” of BFS vs. DFS

Biased random walk

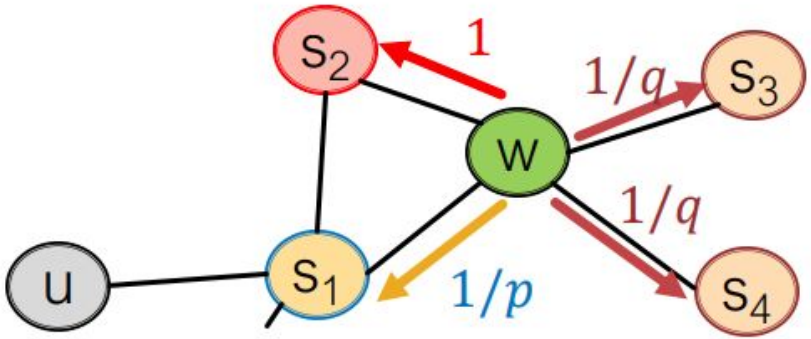
Biased 2nd-order random walks explore network neighborhoods

- Random walk just traversed edge (s_1, w) and is now at w
- Observation: Neighbors of w can only be



Biased random walk

Walker came over edge (***s*** $_1$, ***w***) and is now at ***w***. Where to go next?



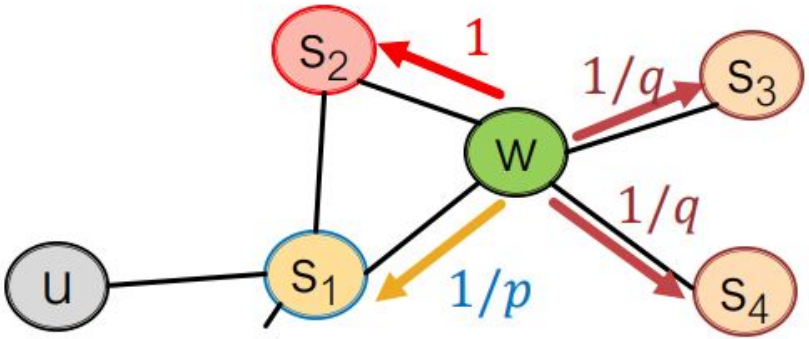
1/p, 1/q, and 1 are **unnormalized** probabilities

p, ***q*** model transition probabilities

- ***p***: return parameter
- ***q***: “walk away” parameter

Biased random walk

Walker came over edge (s_1, w) and is now at w. Where to go next?



	Target t	Prob.	Dist. (s_1, t)
w →	s_1	$1/p$	0
	s_2	1	1
	s_3	$1/q$	2
	s_4	$1/q$	2

1/p, 1/q, and 1 are **unnormalized** probabilities

- **DFS-like walk**: High value of ***1/p***
- **BFS-like walk**: High value of ***1/q***
- **Random walk $N_R(u)$**: nodes visited by the biased walk

Representation learning framework

- Given $G = (V, E)$
- **Goal**: learn a mapping f: u -> R^d: $f(u)=z_u$
- **Intuition**: learn representations such that given the representation z_u of node u, we can predict what its neighbors $N_R(u)$ are
- **Log-likelihood objective**

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$

← Maximum likelihood objective

$N_R(u)$ is the neighborhood of node u by strategy R

Optimization

- **Log-likelihood objective**

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$

- Equivalently

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

- **Intuition**: maximize the similarity between ***u*** and other nodes in the walk $\boldsymbol{N_R(u)}$
- Parameterize $\boldsymbol{P(v|z_u)}$ using softmax

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}$$

Optimization

- Putting it all together

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

- Finding representations $\boldsymbol{z_u}$ that minimize $\boldsymbol{L}$

Optimization

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

Expensive to compute.
Need approximation.

- **Solution**: Negative sampling

$$\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

$$\approx \log\left(\sigma(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)\right) - \sum_{i=1}^{k} \log\left(\sigma(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_{n_i})\right), n_i \sim P_V$$

Sigmoid function

(Makes each term a "probability" in [0, 1])

Random distribution voer nodes

- **Idea**: Instead of normalizing w.r.t all nodes, just normalize against ***k*** random negative samples
- Sample ***k*** negative nodes each with prob. proportional to its degree

node2vec algorithm

1. Compute random walk probabilities
2. Simulate r random walks of length l starting from each node u
3. Optimize the node2vec objective using Stochastic Gradient Descent

- Linear-time complexity
- All 3 steps are individually parallelizable

Other random walk-based methods

- Different kinds of biased random walks:
  - Based on node attributes (Dong et al., 2017)
  - Based on learned weights (Abu-El-Haija et al., 2017)
- Alternative optimization schemes:
  - Directly optimize based on 1-hop and 2-hop random walk probabilities (e.g., LINE from Tang et al. 2015)
- Network preprocessing techniques:
  - Run random walks on modified versions of the original network (e.g., Ribeiro et al. 2017's struct2vec, Chen et al. 2016's HARP)

Demo: Graph Representation Learning

Google Colab

Summary of today

- Network (graph) data
  - Nodes
  - Edges
- Different graph ML tasks & features
  - Node-level
  - Edge-level
  - Graph-level
- Representations learning in graphs
  - Learning node embeddings that capture structure/topological similarity between network nodes