Graph Learning

IFT6758 - Data Science

Sources:

http://snap.stanford.edu/proj/embeddings-www/

https://jian-tang.com/files/AAAI19/aaai-grltutorial-part2-gnns.pdf

Mila

Université de Montréal

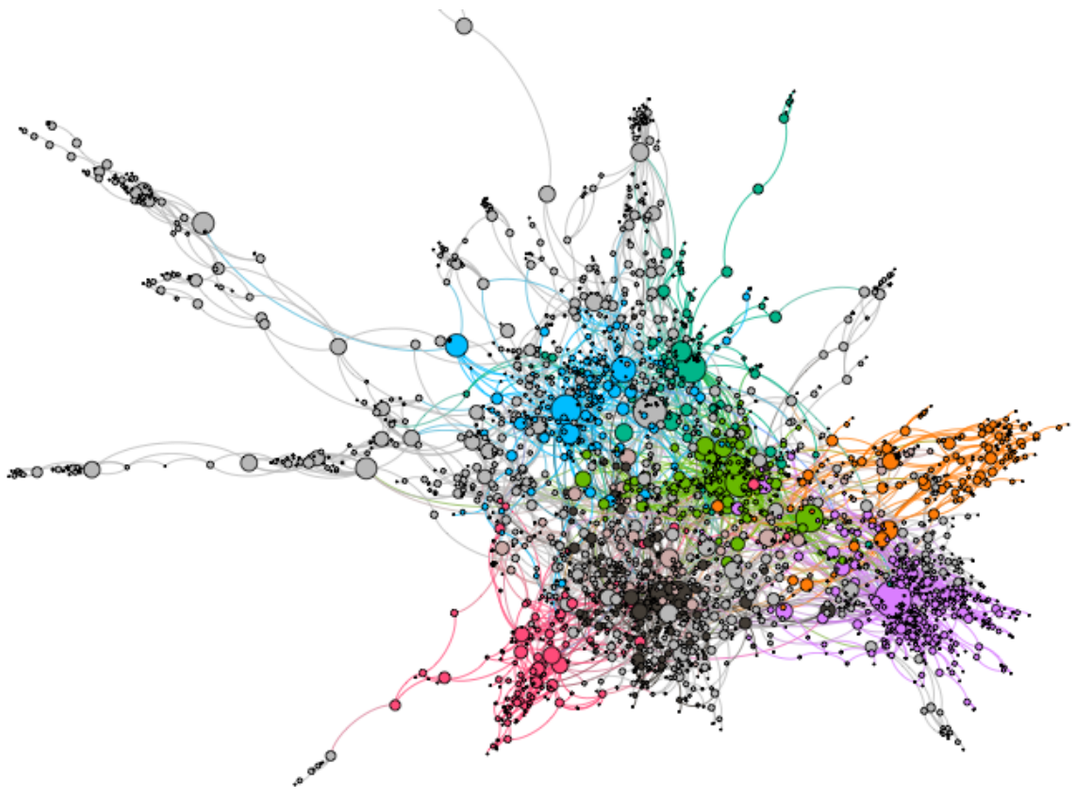
Announcements

- Mid-term exam grades will be published on Gradescope today.
- Assignment 4, final presentation, group and individual reports will be published on Gradescope on Wednesday.
- Presentation format is similar to mid-term, i.e., 7 Min for presentation (all team-members should present to get a score)

  BUT:

  - 3 Min questions about the presentation
  - 5 Min coding questions from all team-members
  - You should ONLY present the model that you will submit on December 2.

Graphs are everywhere

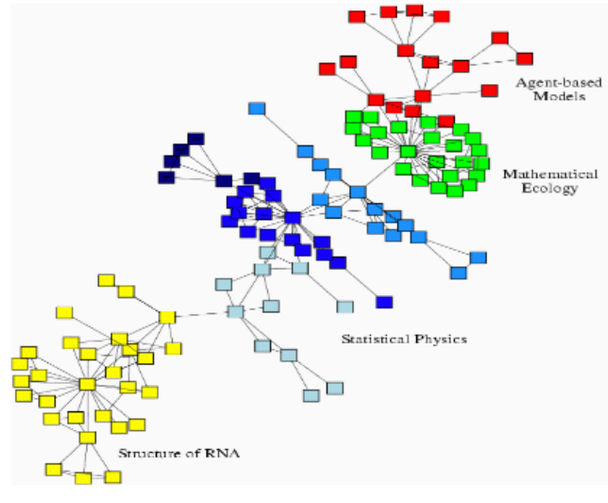


Graph G = (V, E)

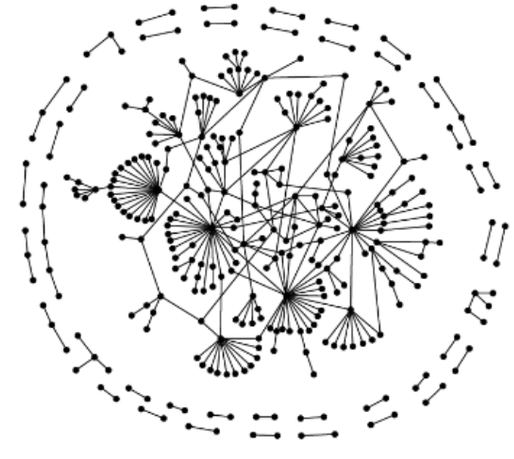
Graphs are a general language for describing and modeling complex systems

Graphs are everywhere

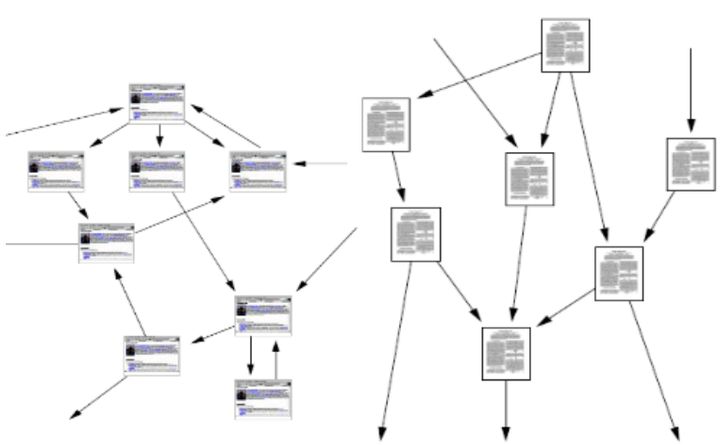
Social networks



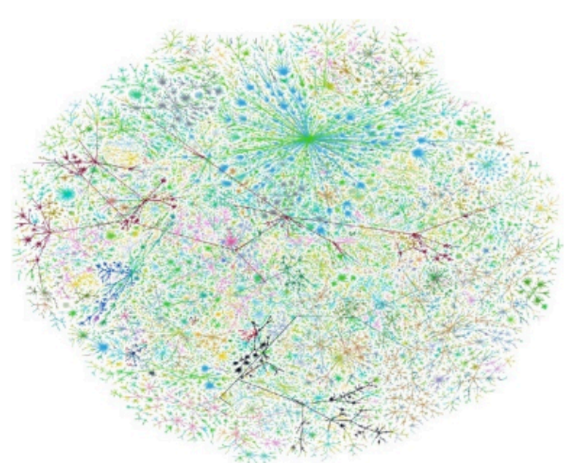
Economic networks



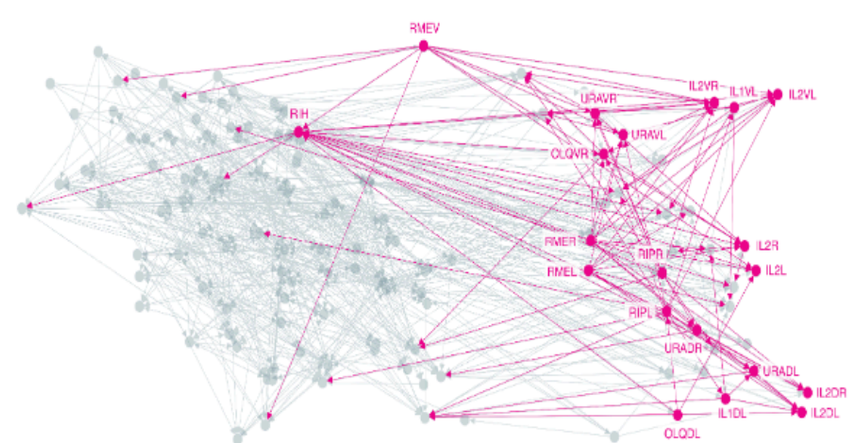
Biomedical networks



Information networks:
Web & citations



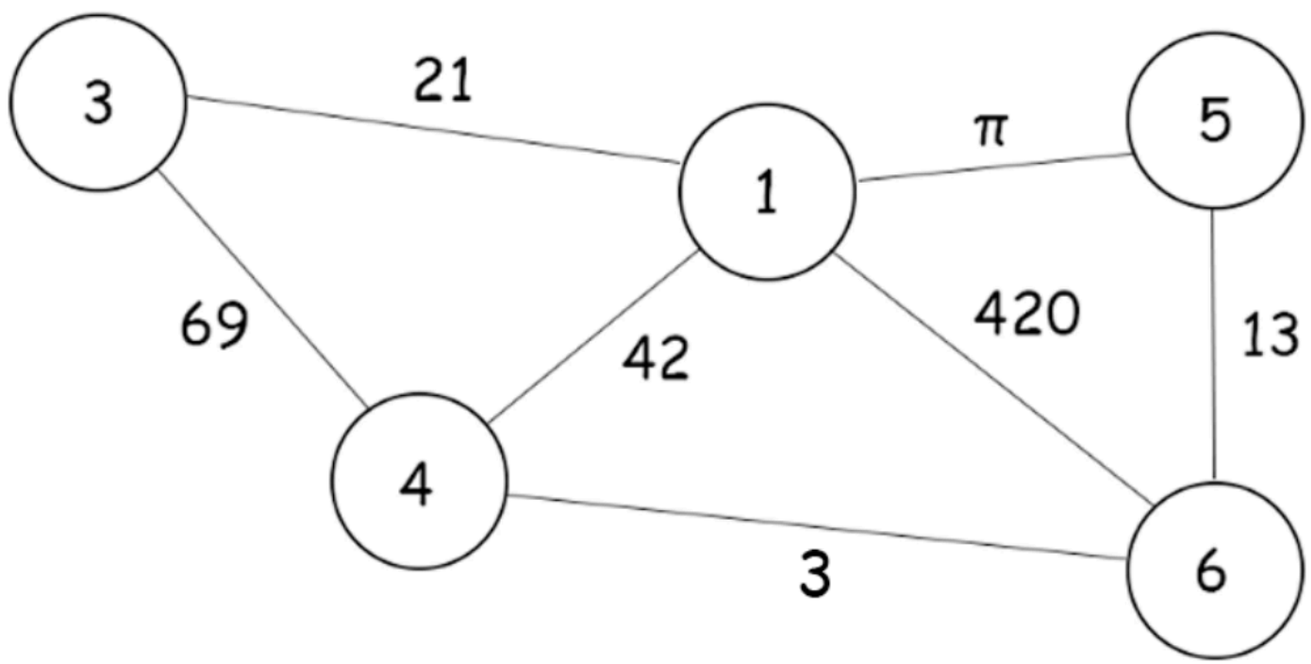
Internet



Networks of neurons

What is a graph?

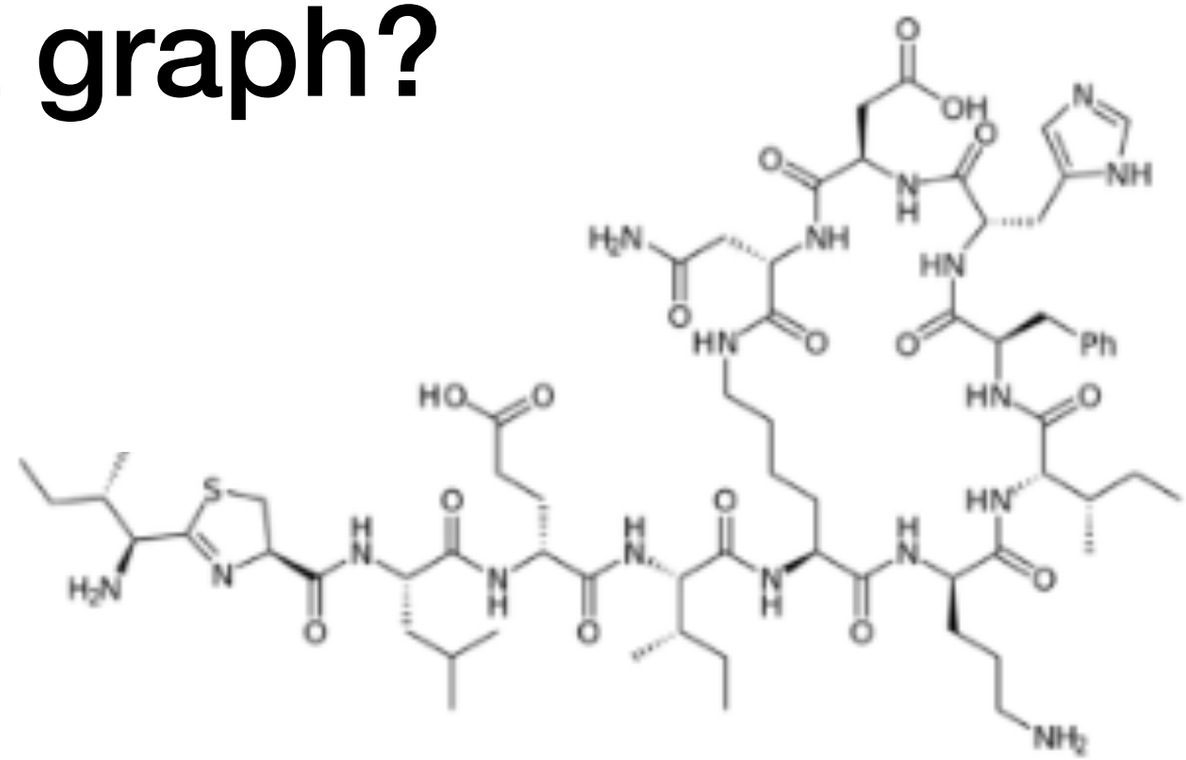
- Graphs can have labels on their edges and/or nodes



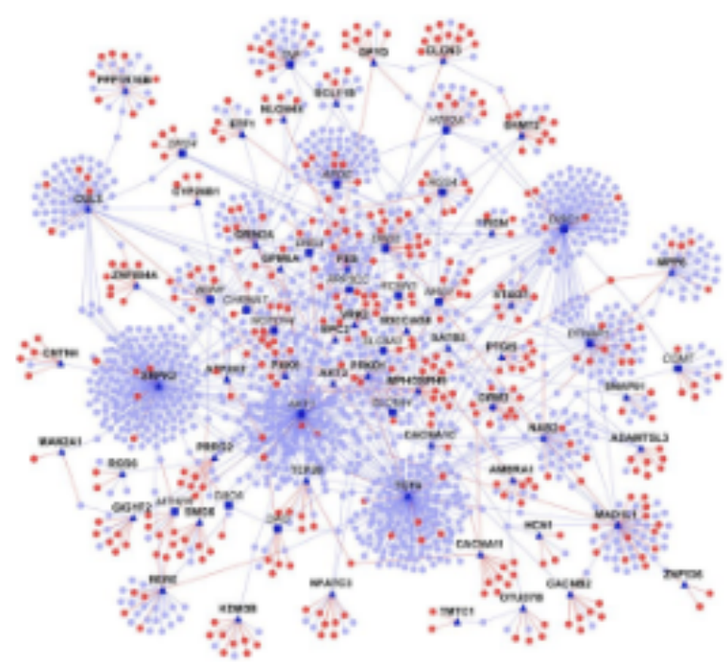
- Labels can also be considered **weights**

What is a graph?

Road maps



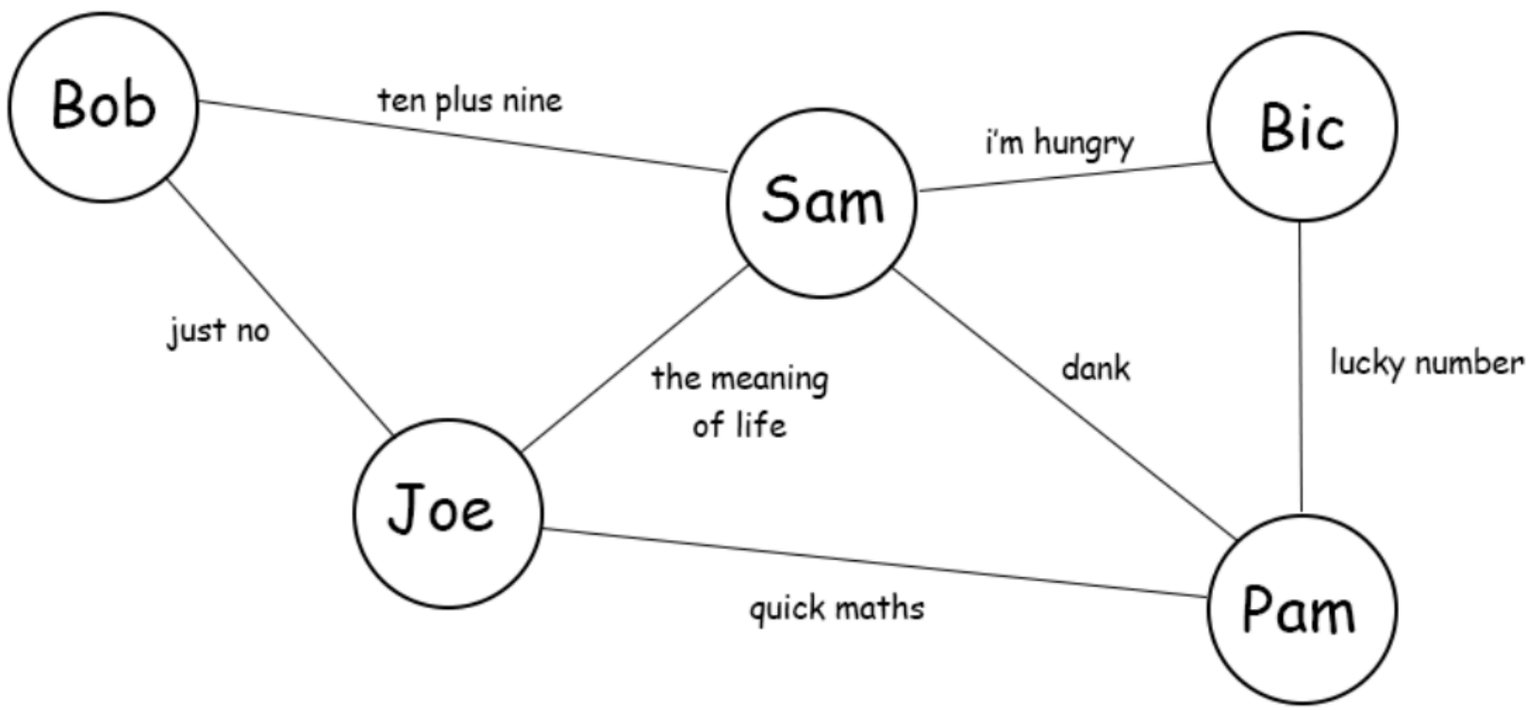
Molecules



Protein interation networks

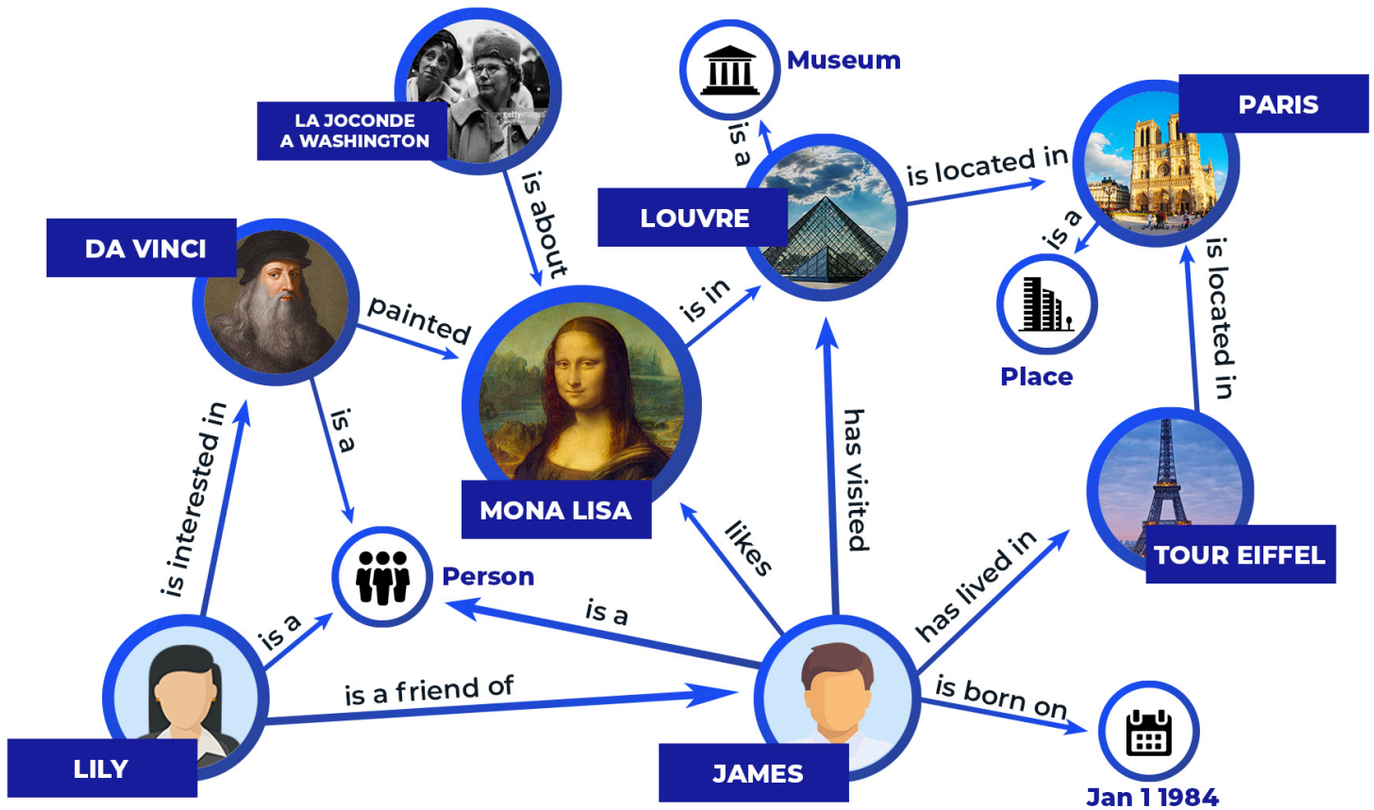
What is a graph?

- Labels don’t have to be numerical, they can be textual.



- **Labels don’t have to be unique**; it’s entirely possible and sometimes useful to give multiple nodes the same label.

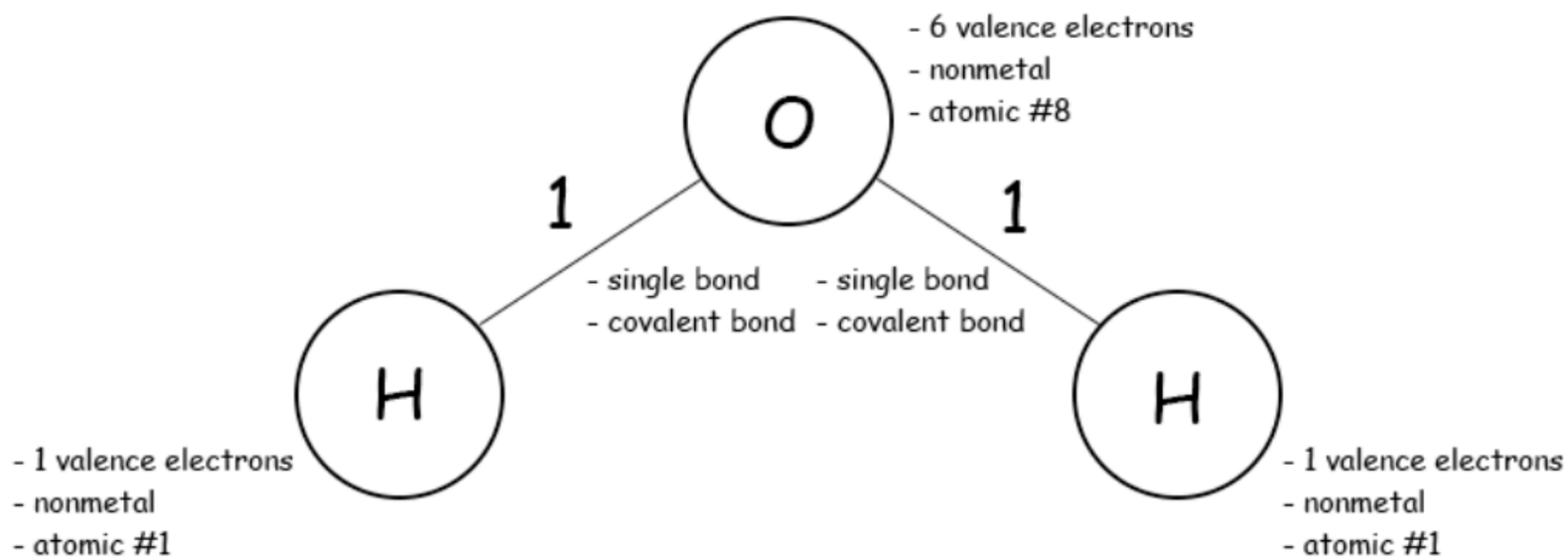
What is a graph?



Wikipedia
Google Knowledge graph
....

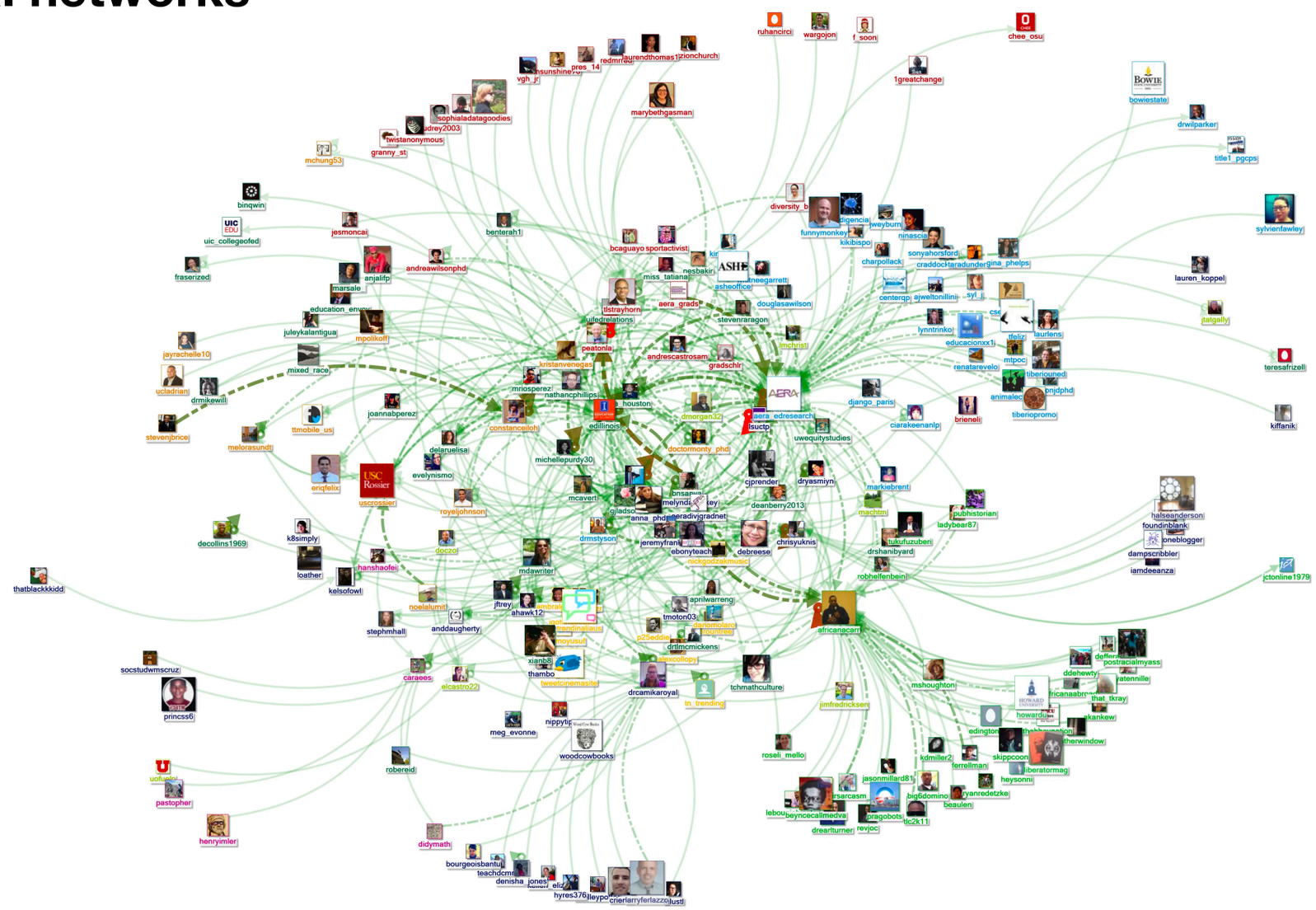
What is a graph?

- Graphs can have **features** (a.k.a attributes)**.**

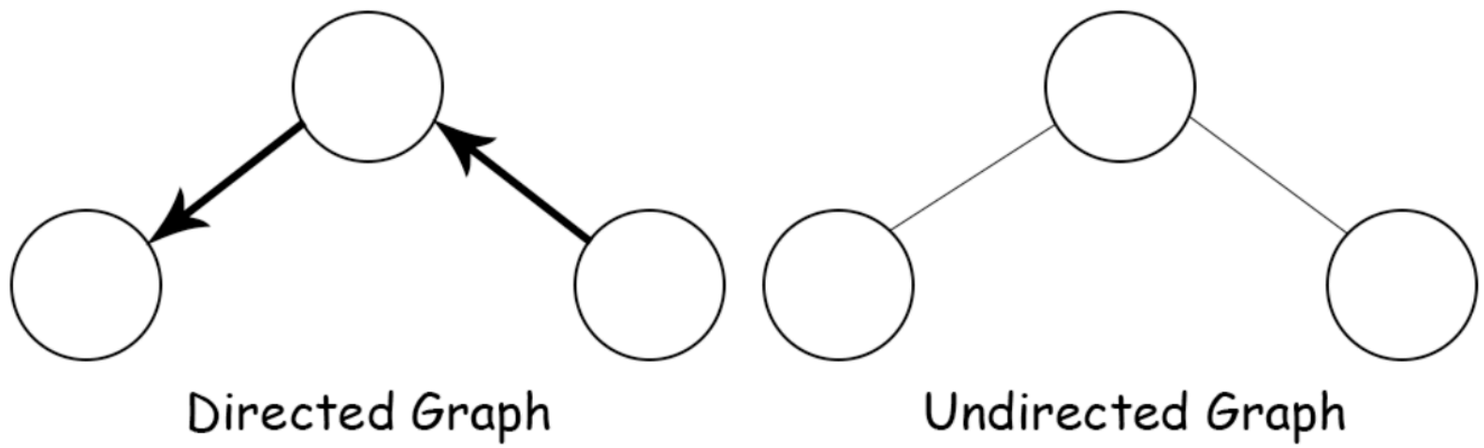


What is a graph?

Social networks



What is a graph?



Graphs can be either:

- **Heterogeneous** — composed of different types of nodes
- **Homogeneous** — composed of the same type of nodes

and are either:

- **Static** — nodes and edges do not change, nothing is added or taken away
- **Dynamic** — nodes and edges change, added, deleted, moved, etc.

Why are graphs useful?

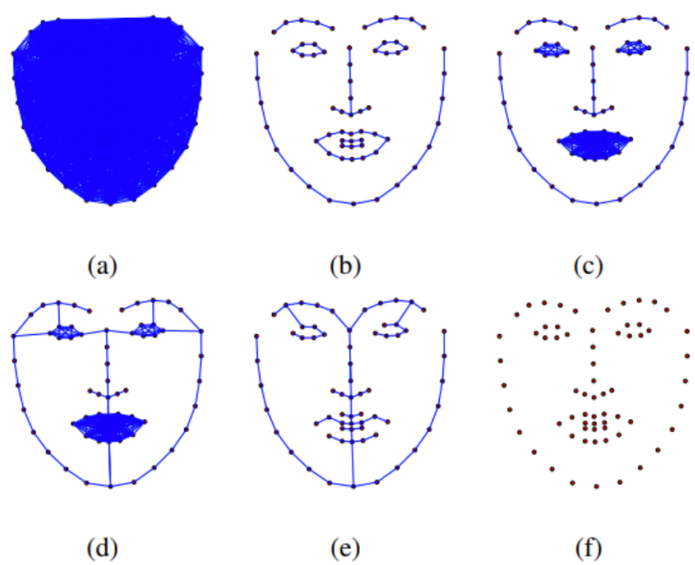
- This is a very **flexible data structure** that generalizes many other data structures. For example, if there are no edges, then it becomes a set; if there are only “vertical” edges and any two nodes are connected by exactly one path, then we have a tree.

- Nodes and edges typically come from some **expert knowledge** or intuition about the problem.

  e.g., Atoms in molecules, Users in a social network, Cities in a transportation system, Players in team sport, Neurons in the brain, Interacting objects in a dynamic physical system, and Pixels, bounding boxes or segmentation masks in images

Why are graphs useful?

- Most ML/CV problems can be viewed as graphs



from (Antonakos et al., CVPR, 2015)

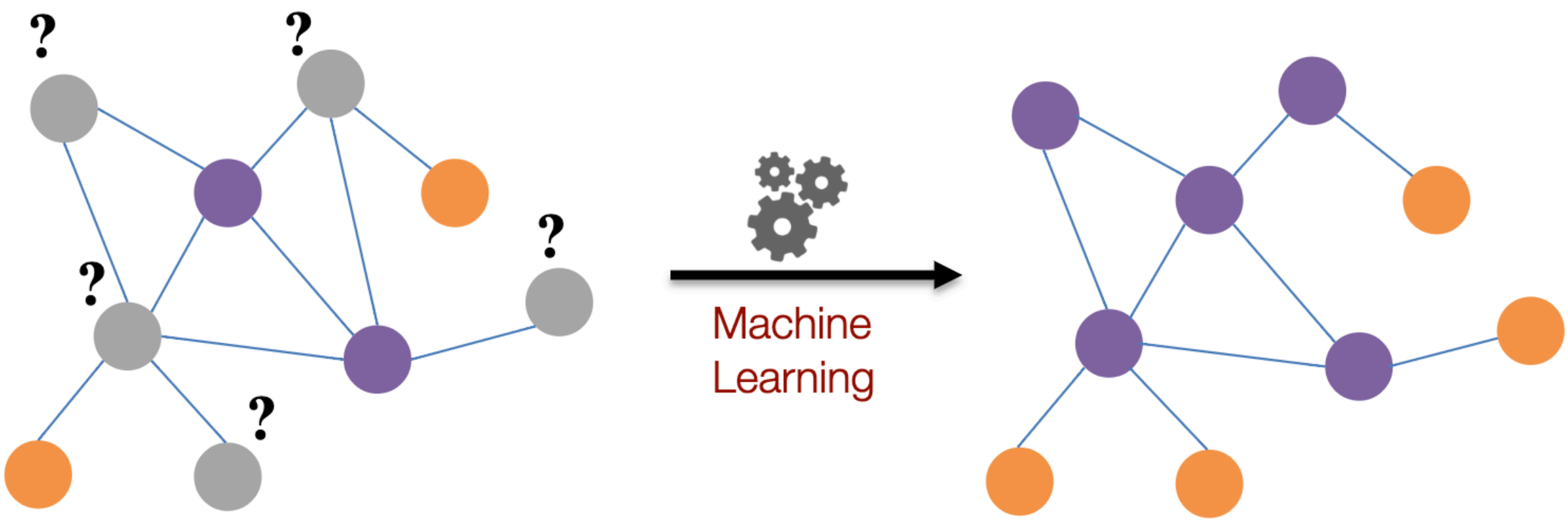
- Graph gives a lot of flexibility and can give a very different and interesting perspective on the problem

- Neural networks can be viewed as graph where nodes are neutrons and weights are edges

Graph mining tasks

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

Node Classification



Node Classification

Classifying the function of proteins in the interactome!

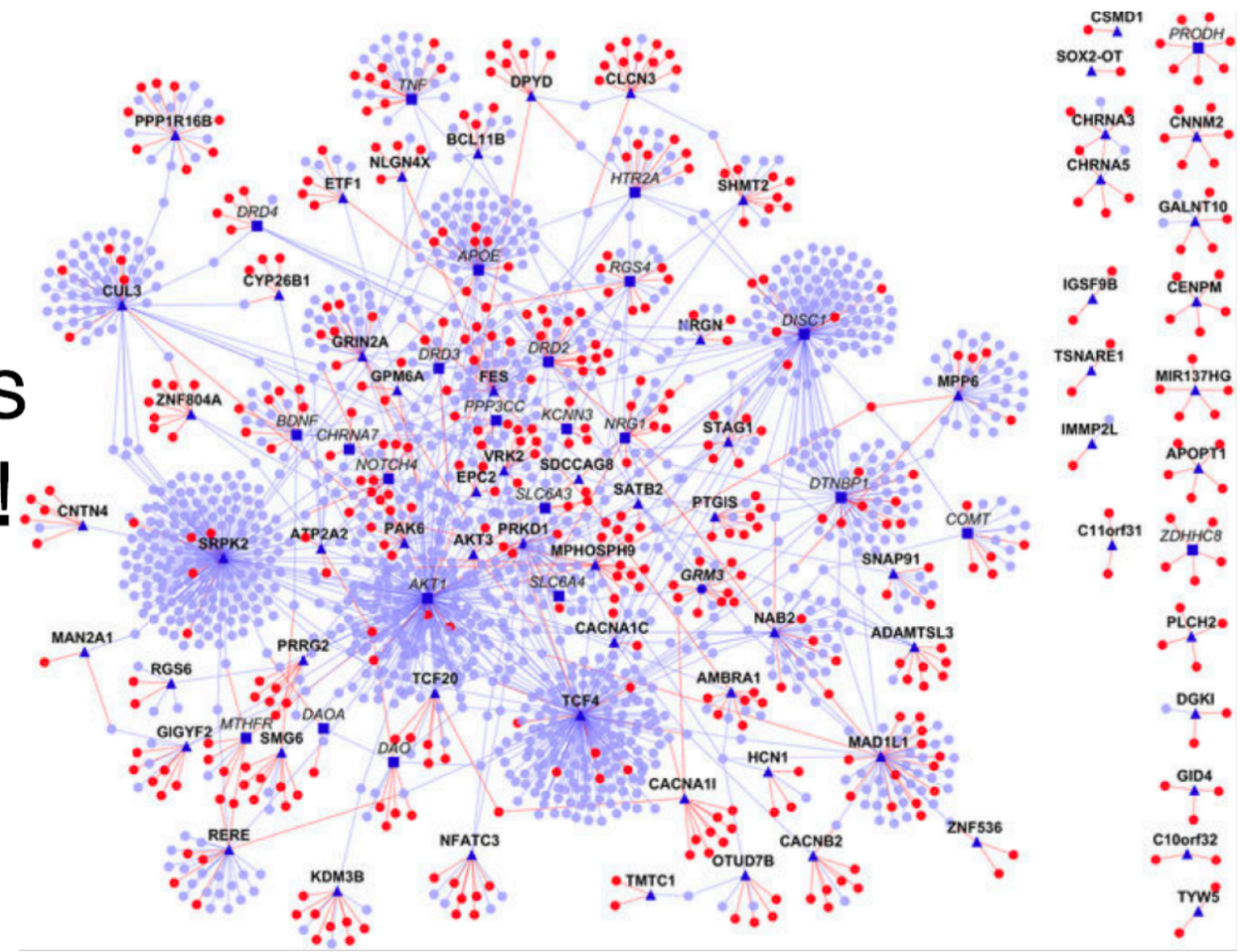
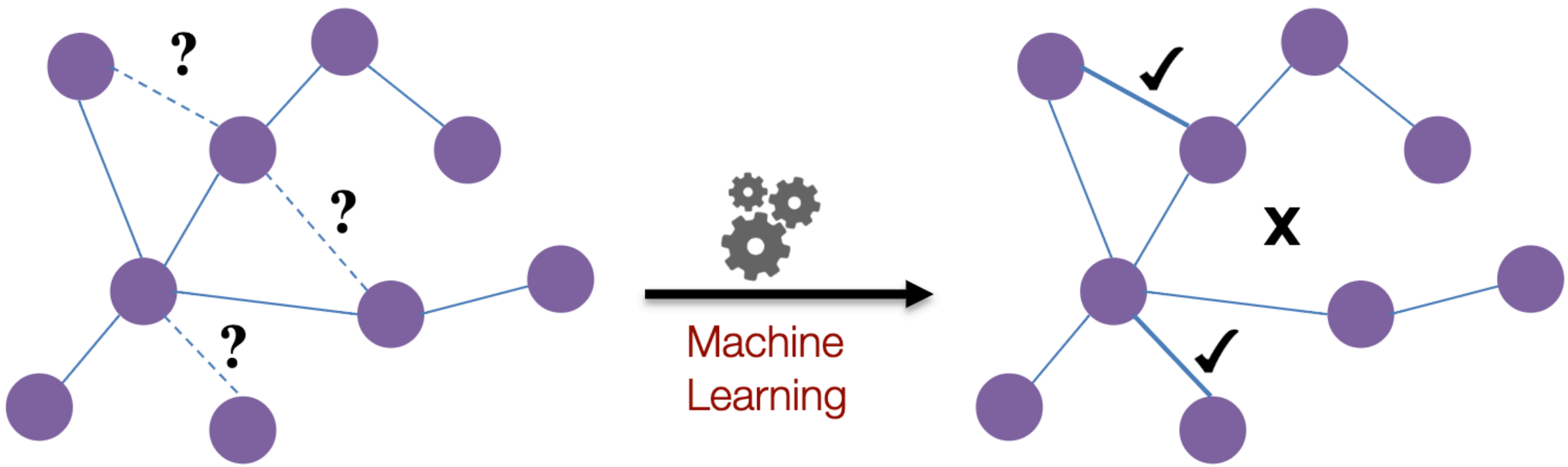


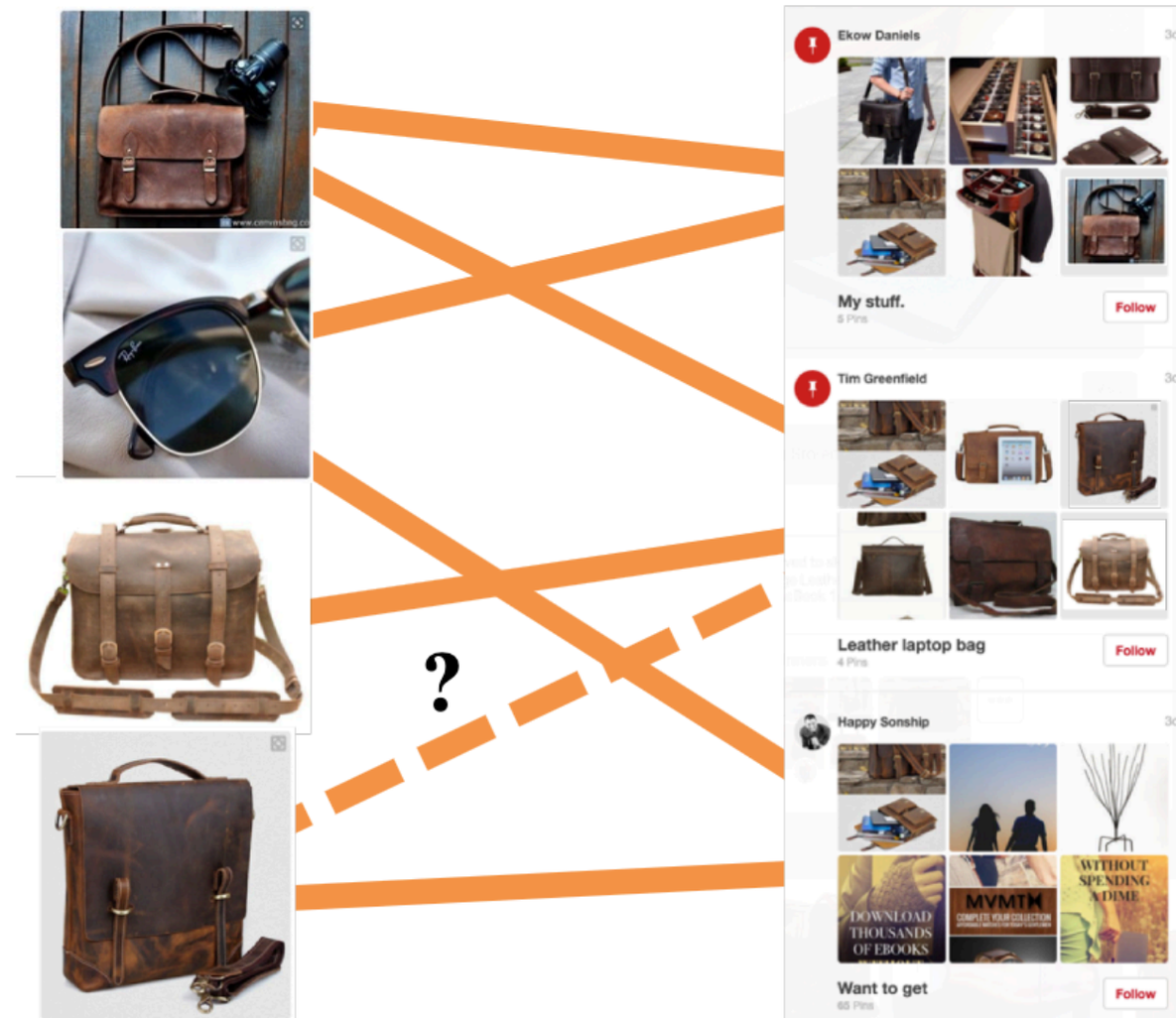
Image from: Ganapathiraju et al. 2016. Schizophrenia interactome with 504 novel protein–protein interactions. *Nature*.

Link Prediction



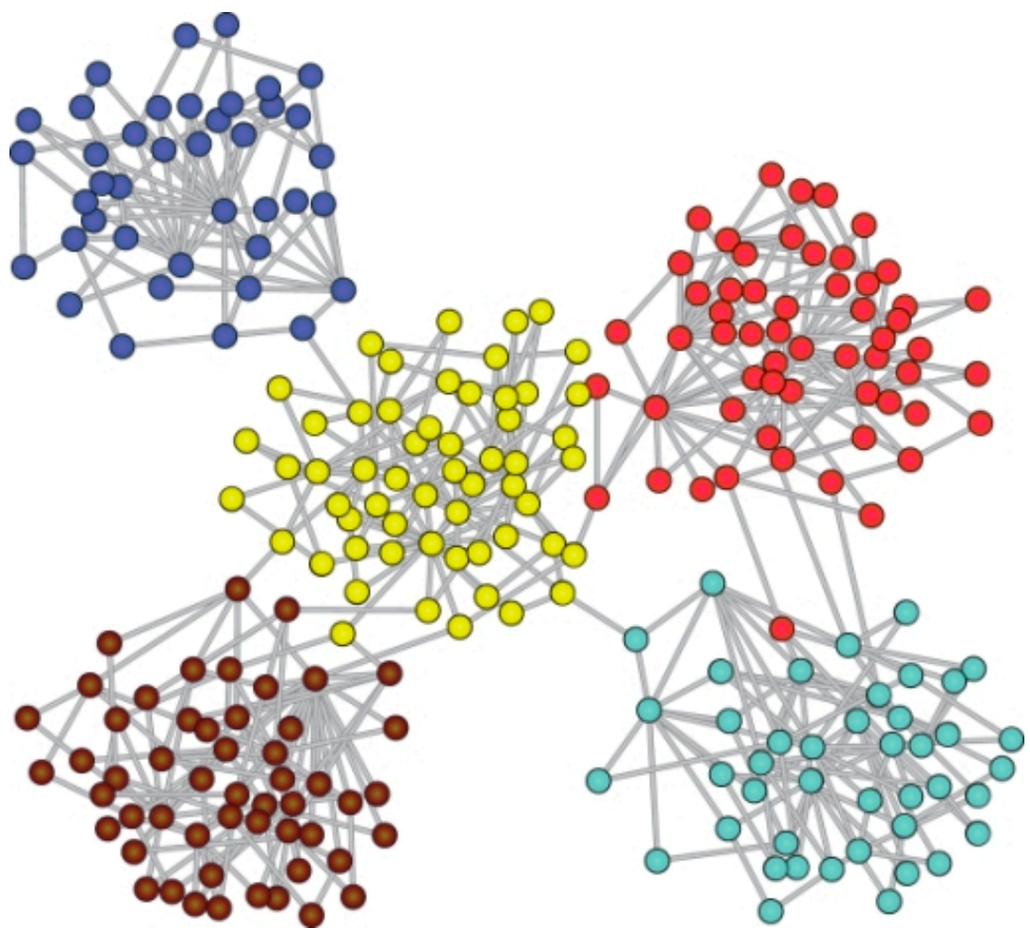
Link Prediction

Content recommendation is link prediction!

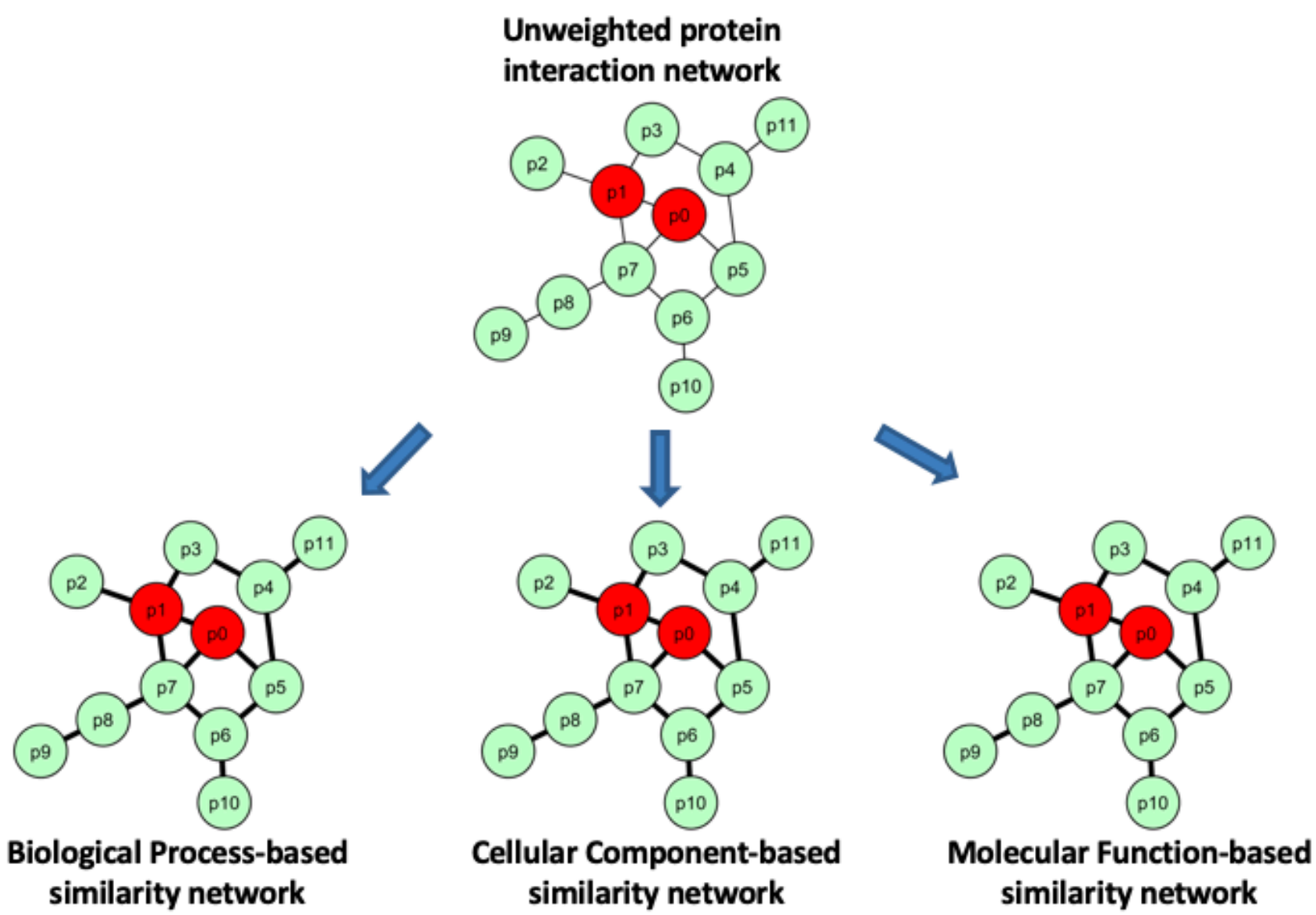


Community Detection

- The field of community detection aims to identify highly connected groups of individuals or objects inside these networks, these groups are called communities.

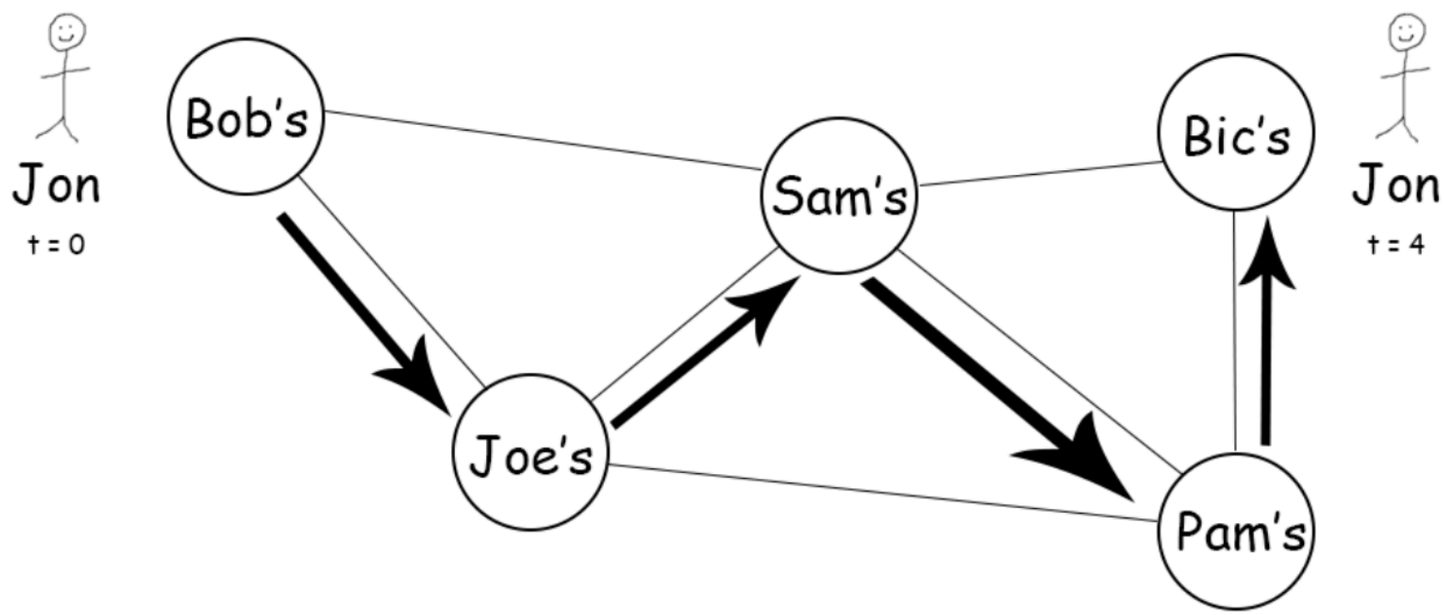


Network Similarity



Graph Basics

Traverse a graph



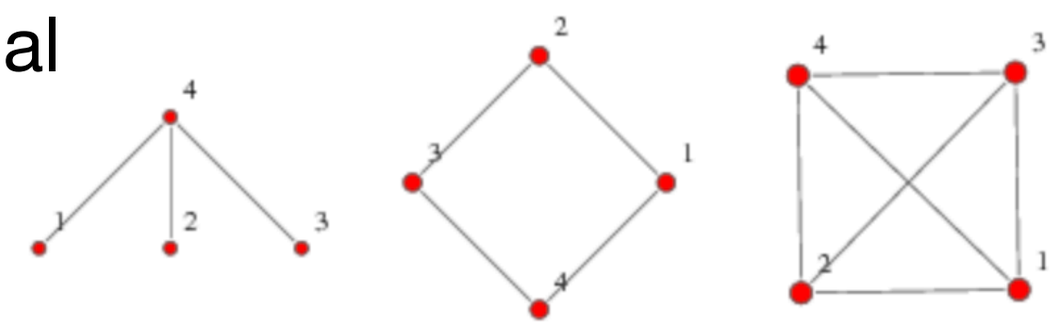
- **Walk:** A graph traversal — a **closed walk** is when the destination node is the same as the source node
- **Trail:** A walk with no repeated edges — a **circuit** is a closed trail
- **Path:** A walk with no repeated nodes — a c**ycle** is a closed path

Adjacency Matrix

- The Adjacency Matrix of a graph is be made of 1s and 0s **unless** it is otherwise weighted or labelled. ***A*** can be built by following this rule:

$$a_{ij} = \begin{cases} 1, \text{ if } (v_i, v_j) \in E \\ 0, \text{ otherwise} \end{cases}$$

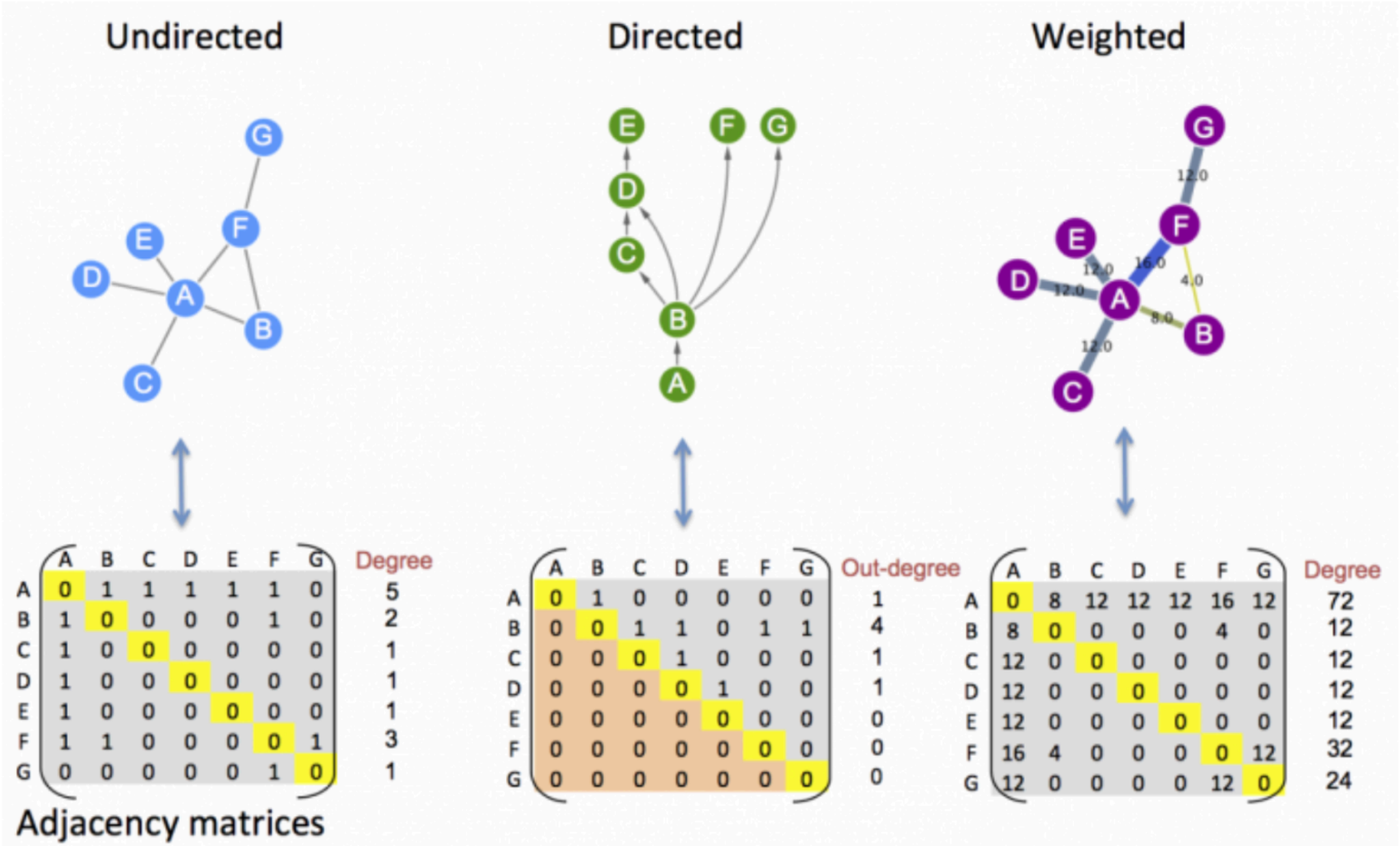
The Adjacency Matrix of a undirected graph is therefore symmetrical along its diagonal



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Degree Matrix

- The Degree Matrix ***D*** of a graph is essentially a diagonal matrix, where each value of the diagonal is the degree of its corresponding node.

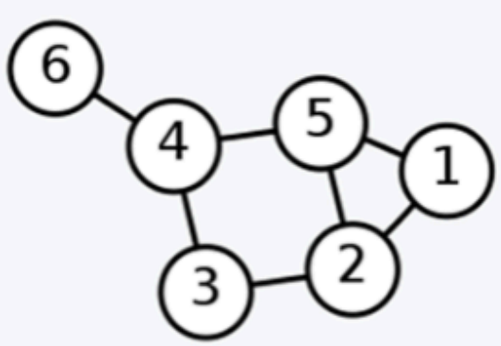


Laplacian Matrix

- The Laplacian Matrix of a graph is the result of subtracting the Adjacency Matrix from the Degree Matrix:

$$L = D - A$$

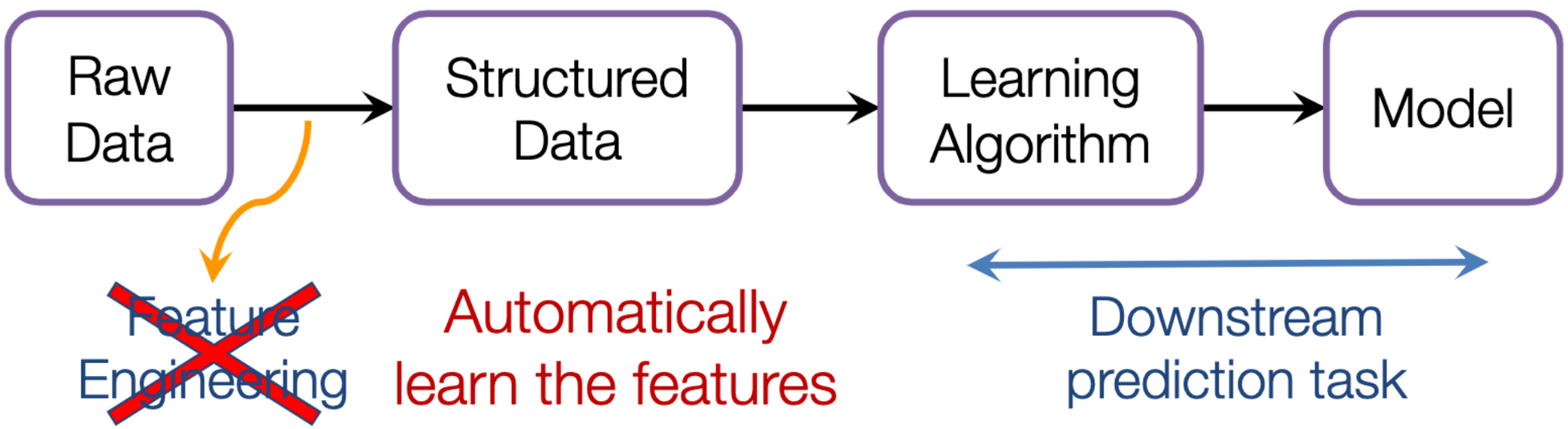
- Each value in the Degree Matrix is subtracted by its respective value in the Adjacency Matrix as such:

Labeled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

Graph Learning

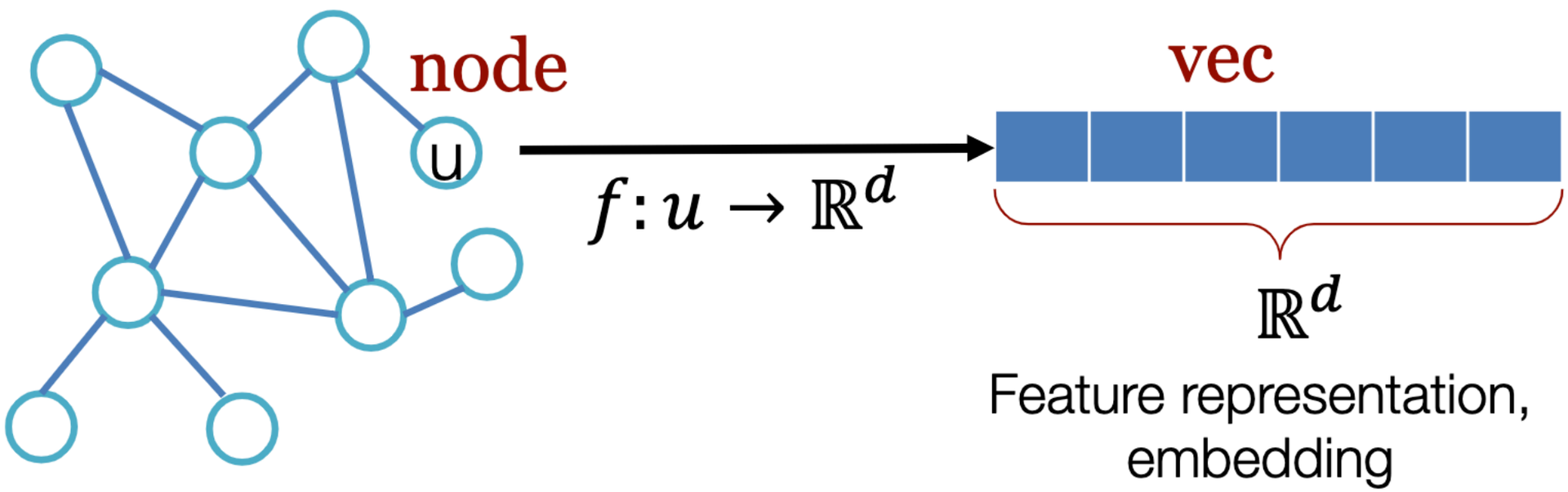
Traditional Machine Learning Pipeline

- (Supervised) Machine Learning Lifecycle



Feature Learning in Graphs

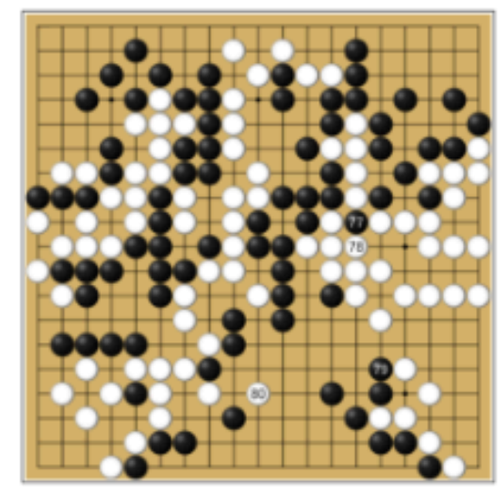
- **Goal**: Efficient task-independent feature learning for machine learning in graphs!



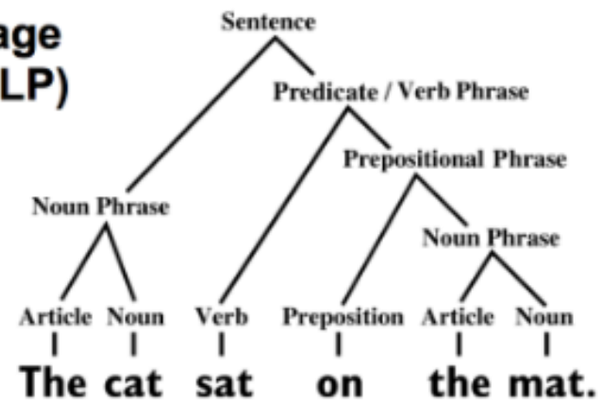
Representation

Speech data

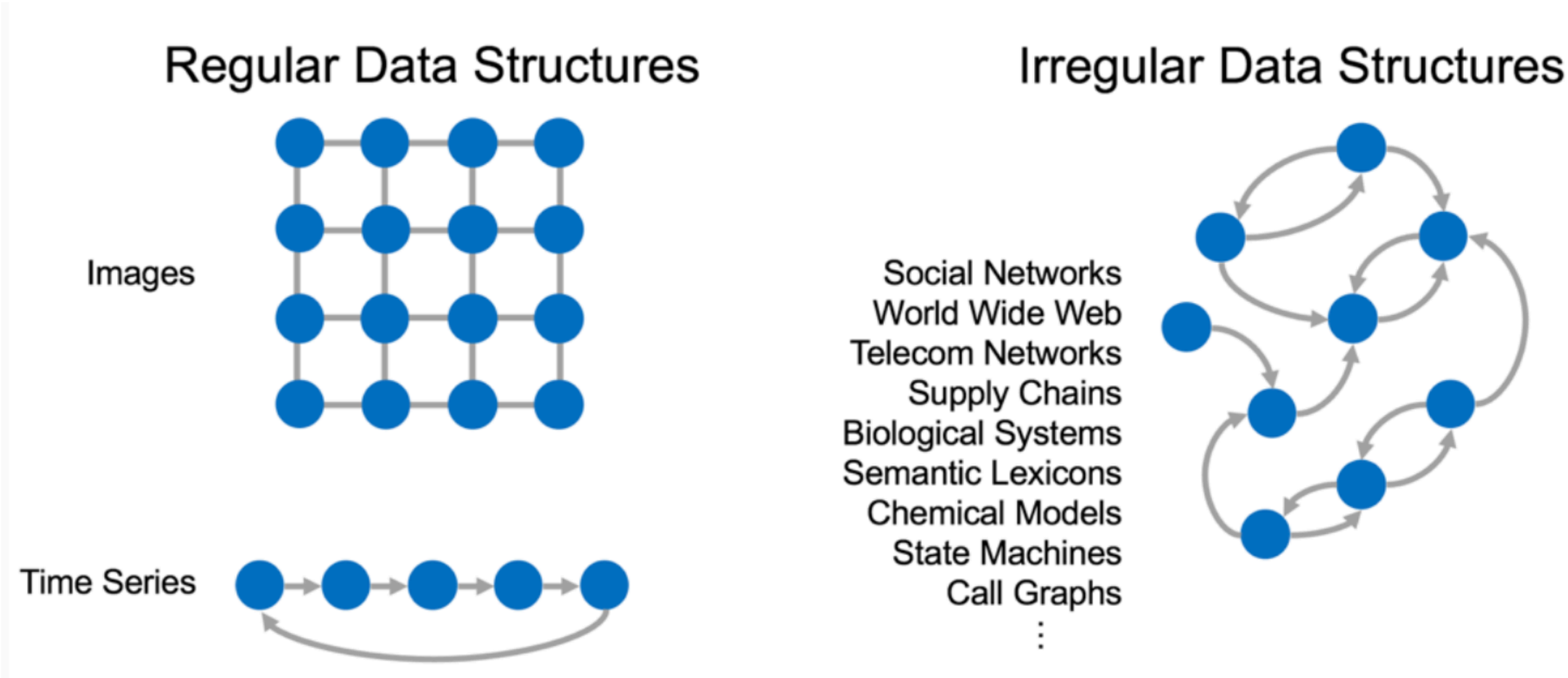
Grid games



Natural language processing (NLP)



Why graph learning is hard?

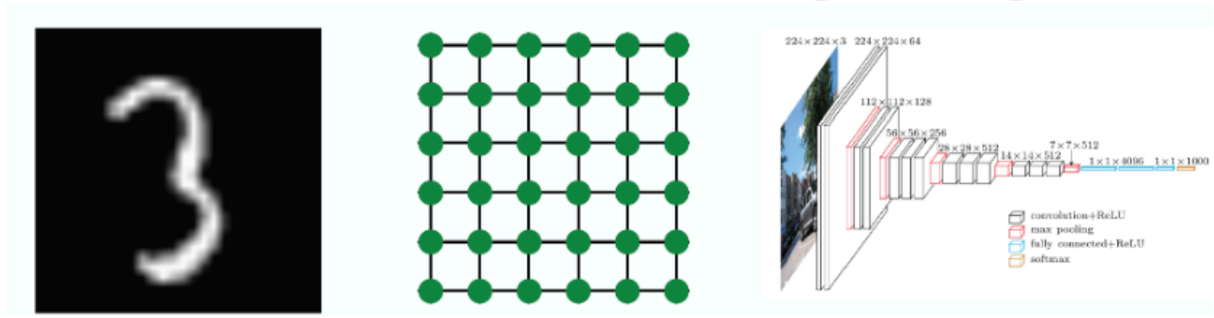


Standard machine learning/deep learning approaches don't work on this data!

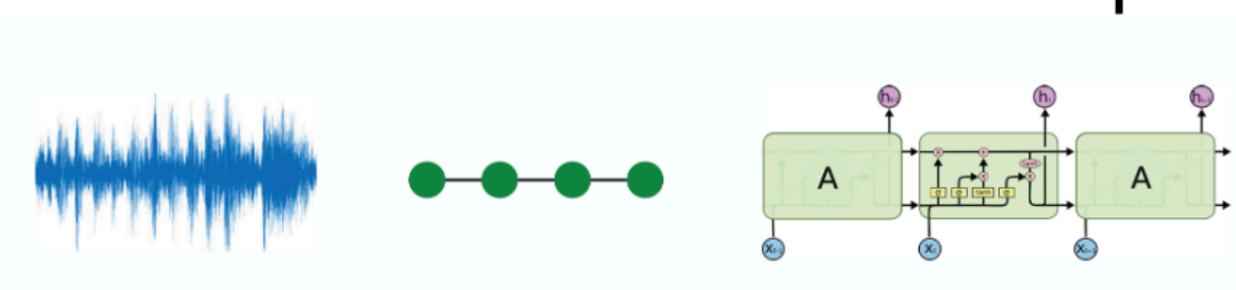
Why Graph learning is hard?

- Modern deep learning toolbox is designed for simple sequences or grids.

CNNs for fixed-size images/grids….

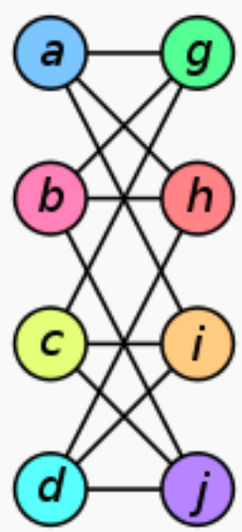
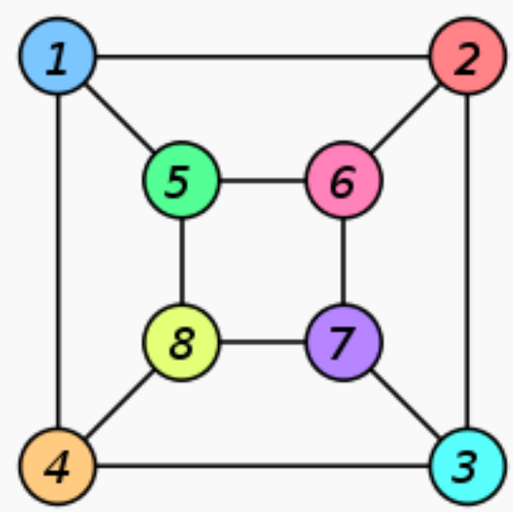


RNNs or word2vec for text/sequences…



recap: Isomorphism problem

- The **graph isomorphism problem** is the computational problem of determining whether two finite graphs are isomorphic.

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

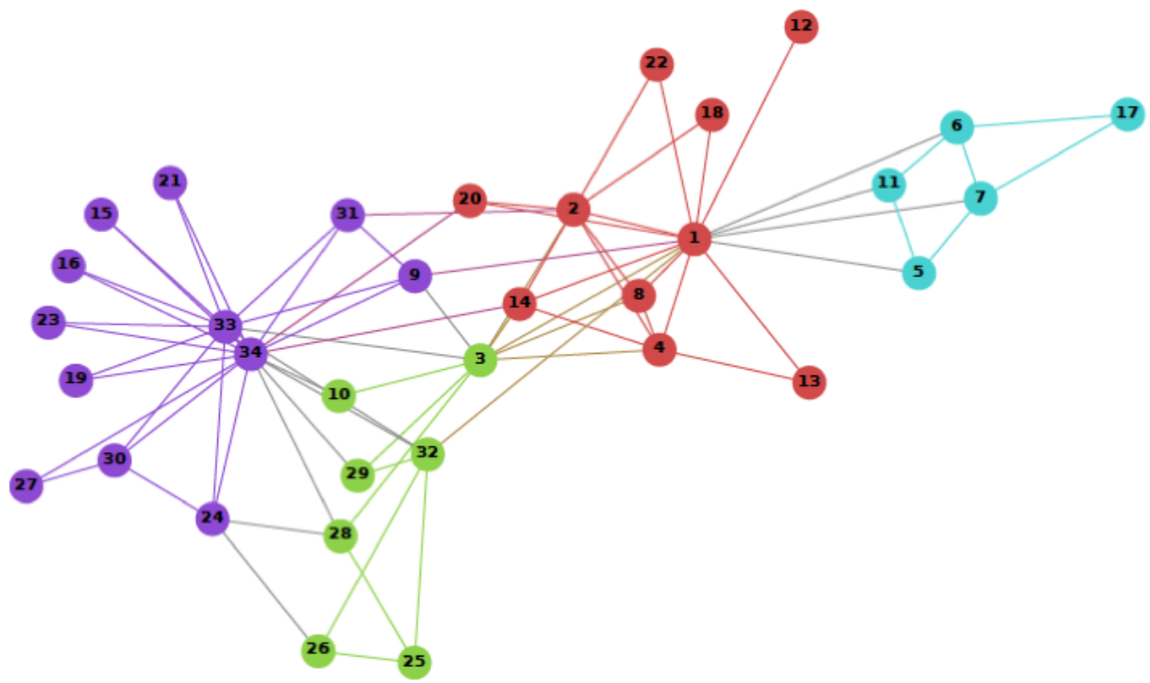
- The graph isomorphism problem is neither known to be NP-complete nor known to be tractable

Why Graph learning is hard?

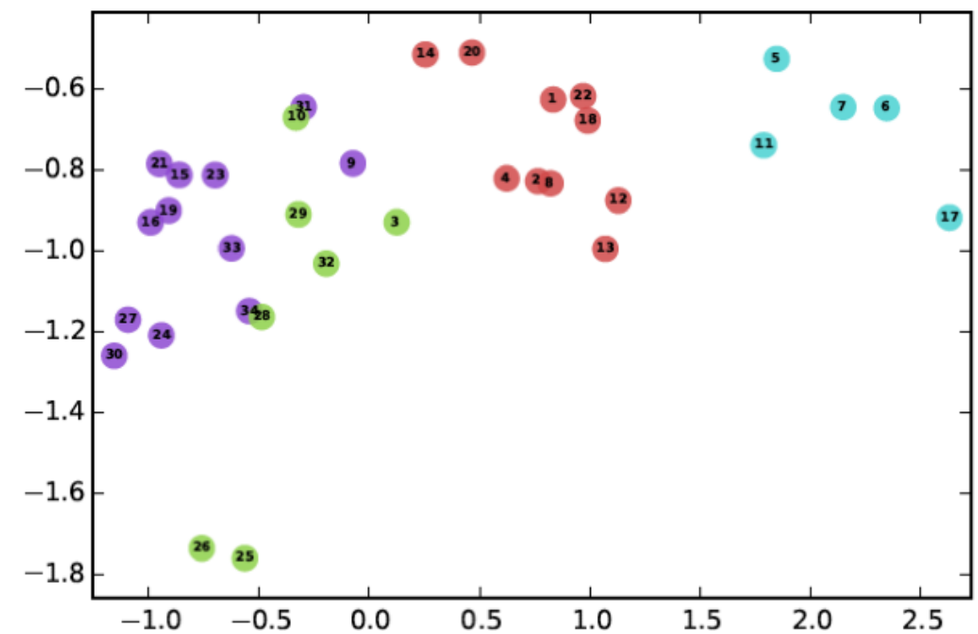
- Graphs are far **more complex** than text or visual data!
- Complex topographical structure (i.e., **no spatial locality** like grids)
- No fixed node ordering or reference point (i.e., the **isomorphism problem**)
- Often dynamic and have multimodal features.

Node Embedding

Node Embedding



Input



Output

- **Intuition**: Find embedding of nodes to **d-dimensions** so that **"similar" nodes** in the graph have embeddings that are **close together**.

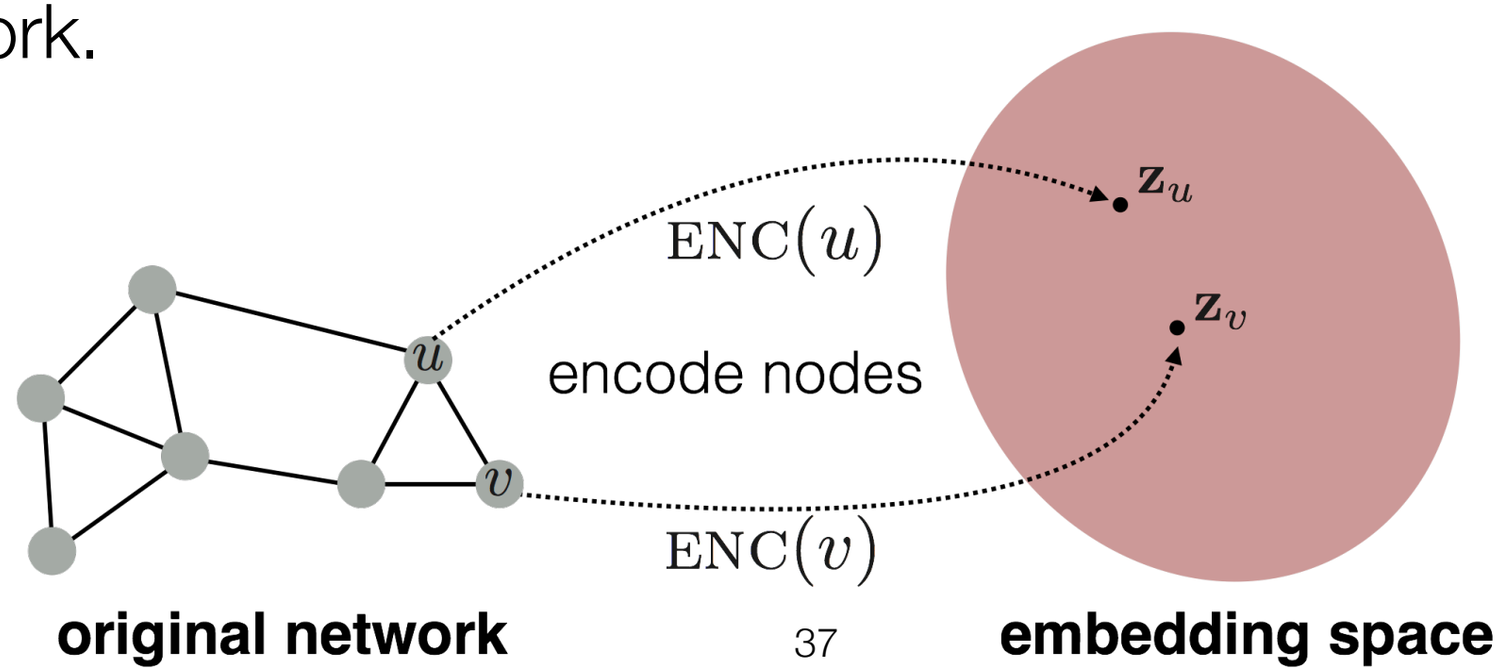
Node Embedding (Set up)

- Assume we have a graph G:
  - V is the vertex set.
  - A is the adjacency matrix (assume binary).
  - No node features or extra information is used!

Node Embedding (Set up)

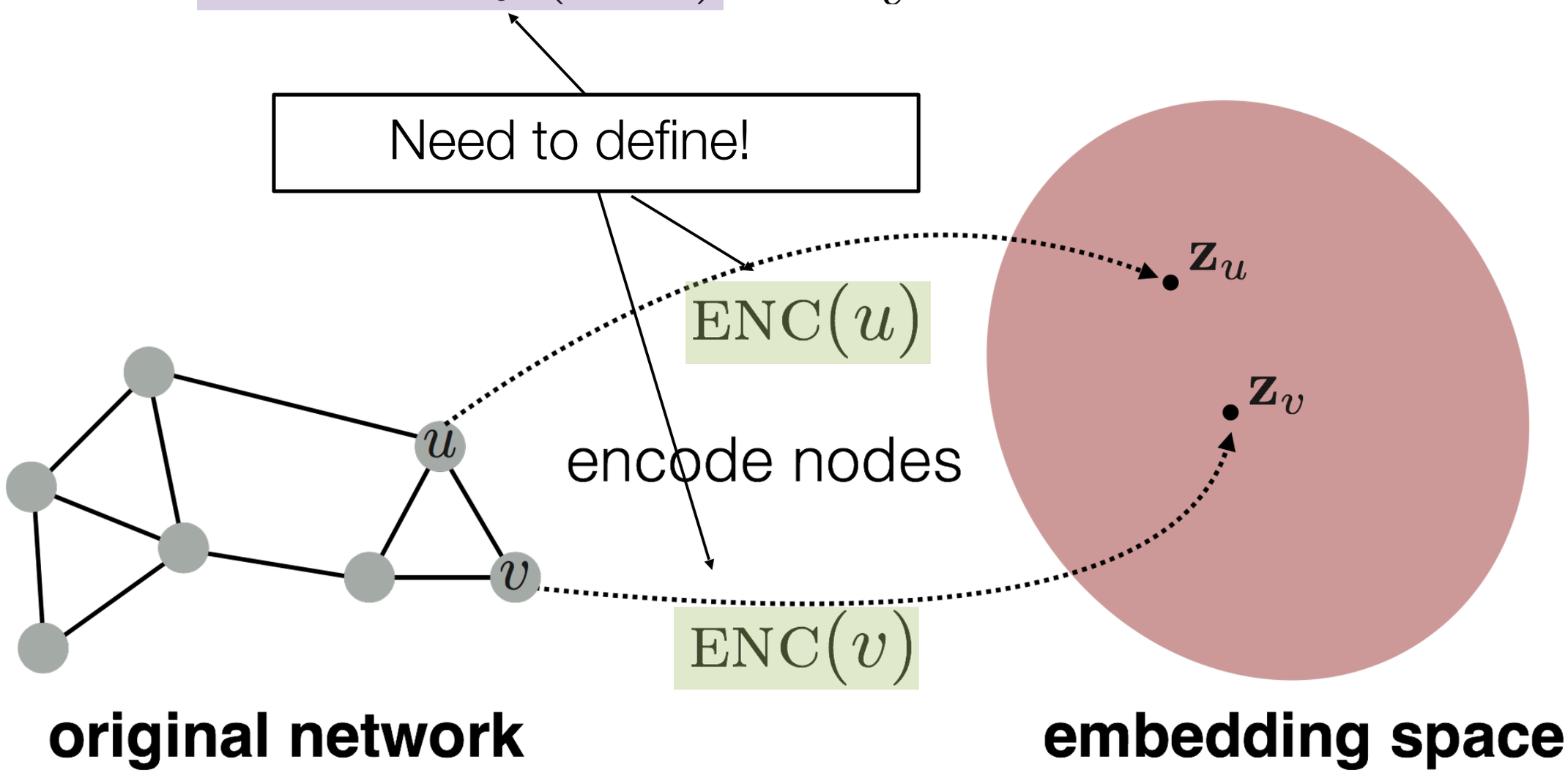
- Assume we have a graph G:
  - V is the vertex set.
  - A is the adjacency matrix (assume binary).
  - No node features or extra information is used!

- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.



Node Embedding (Set up)

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



Learning Node Embeddings

1. Define an encoder (i.e., a mapping from nodes to embeddings)

2. Define a node similarity function (i.e., a measure of similarity in the original network).

3. Optimize the parameters of the encoder so that:

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

Key Components

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

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

d-dimensional embedding

node in the input graph

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

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

Similarity of u and v in the original network

dot product between node embeddings

Shallow Encoding

- Simplest encoding approach: encoder is just an embedding-lookup

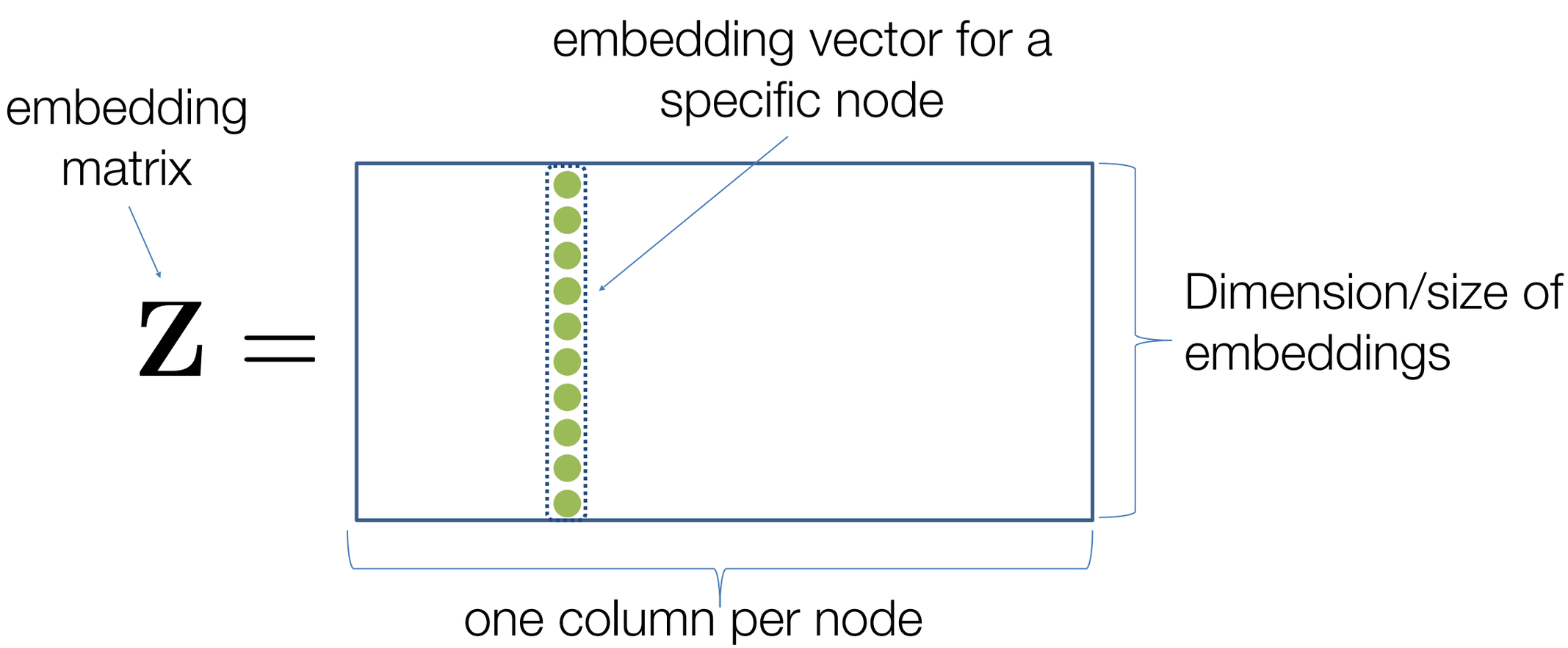
$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

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

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

Shallow Encoding

- Simplest encoding approach: encoder is just an embedding-lookup



From Shallow to Deep

- Limitations of shallow encoding:
  - O(|V|) parameters are needed: there **no parameter sharing** and every node has its own unique embedding vector.
  - Inherently "**transductive**": It is impossible to generate embeddings for nodes that were not seen during training.
  - Do not incorporate node features: Many graphs have features that we can and should leverage.

From Shallow to Deep

- We will now discuss “deeper” methods based on graph neural networks.

$$\mathrm{ENC}(v) = \text{complex function that depends on graph structure.}$$

- In general, all of encoders can be combined with the similarity functions that depends on graph structure.

How to Define Node Similarity?

- Key distinction between “shallow” methods is how they define node similarity.

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

How to Define Node Similarity?

1. Adjacency-based similarity
2. Multi-hop similarity
3. Random walk approaches

High-level structure and material from:

- Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.

Adjacency-based Similarity

Material based on:

- Ahmed et al. 2013. Distributed Natural Large Scale Graph Factorization. *WWW.*

Adjacency-based Similarity

- **Similarity function** is just the **edge weight** between u and v in the original network.

- Intuition: Dot products between node embeddings approximate edge existence.

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

loss (what we want to minimize)

sum over all node pairs

embedding similarity

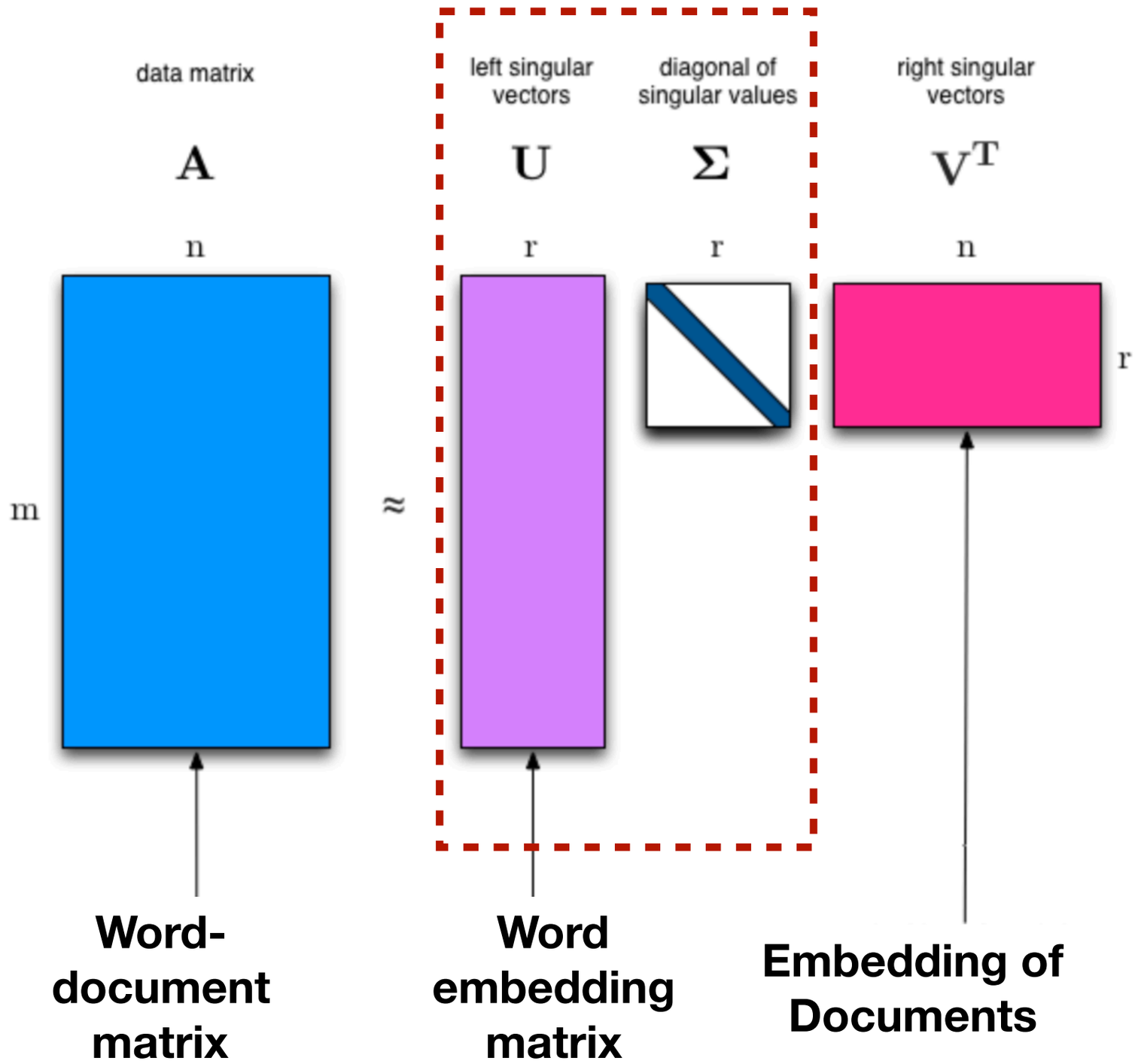
(weighted) adjacency matrix for the graph

Adjacency-based Similarity

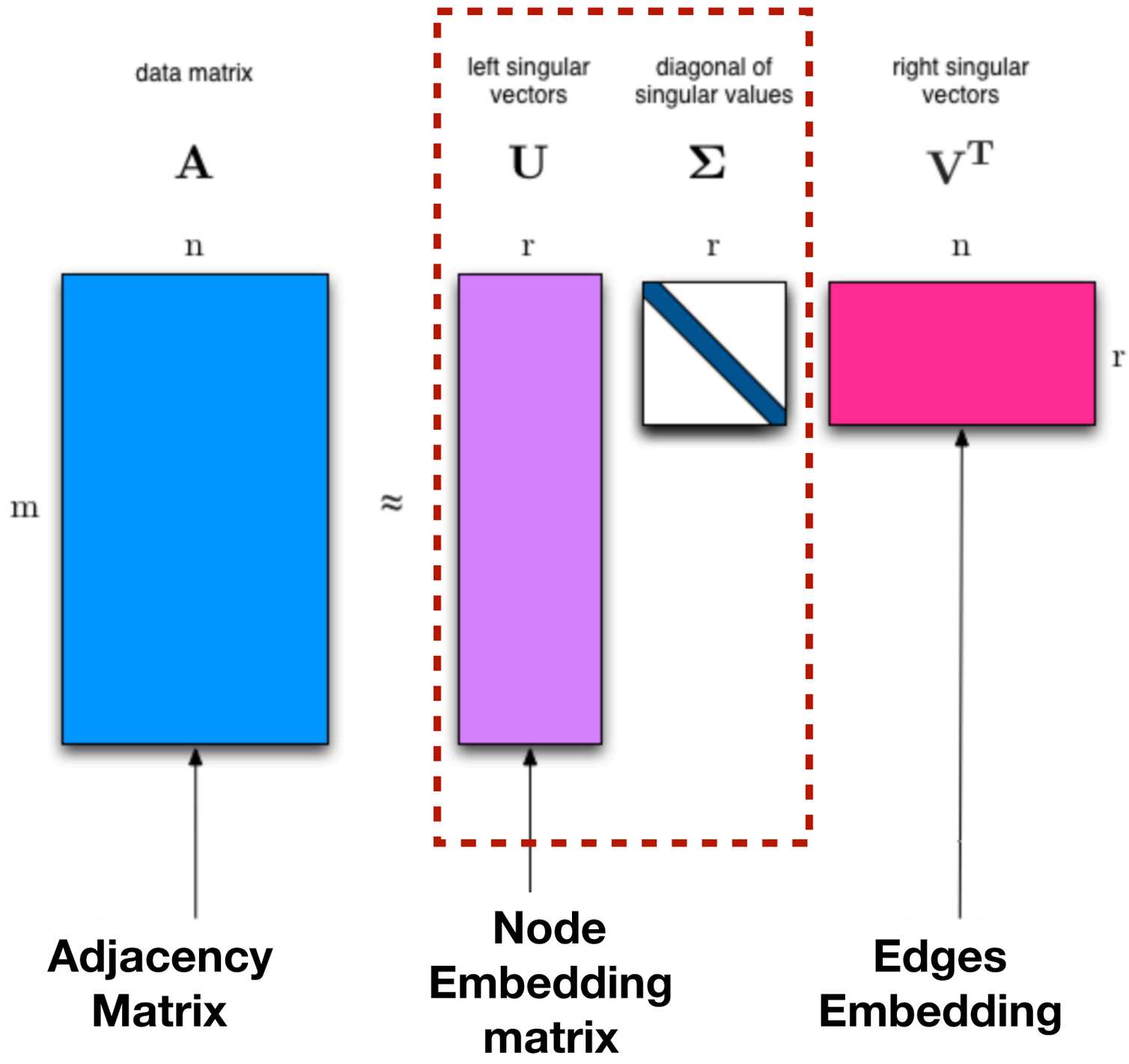
$$\mathcal{L} = \sum_{(u,v) \in V \times V} \| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v} \|^2$$

- Find embedding matrix that minimizes the loss
  - Option 1: Use **stochastic gradient descent (SGD)** as a general optimization method.
    - Highly scalable, general approach
  - Option 2: Solve **matrix decomposition solvers** (e.g., SVD).
    - Only works in limited cases.

Recap: SVD for Word embedding



Recap: SVD for Node embedding



Recap: Node Similarity is preserved

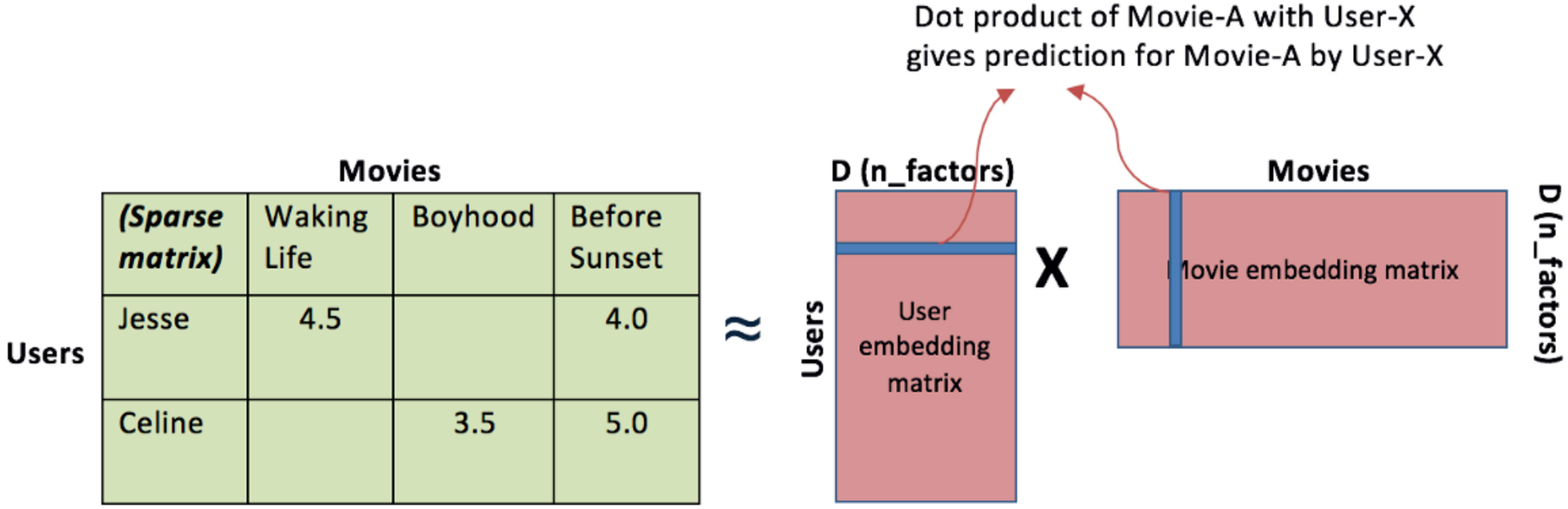
data matrix		left singular vectors	diagonal of singular values	right singular vectors
$\mathbf{C}$	$=$	$\mathbf{U}$	$\mathbf{\Sigma}$	$\mathbf{V}^{\mathbf{T}}$

- Given an Adjacency matrix C, we can get a decomposition C' from SVD.

$$
\begin{aligned}
\hat{X}\hat{X}^T &= (U\Sigma V^T)(U\Sigma V^T)^T \\
&= (U\Sigma V^T)(V\Sigma U^T) \\
&= U\Sigma\Sigma^T U^T \quad (\because V^T V = I) \\
&= U\Sigma(U\Sigma)^T
\end{aligned}
$$

Matrix factorization

- Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices.

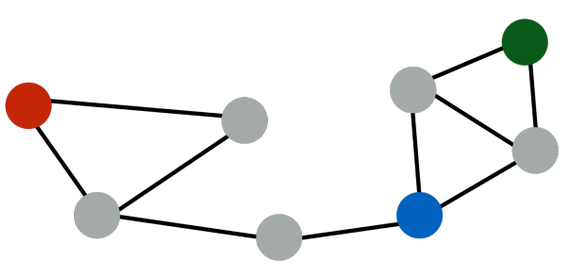


- This family of methods became widely known during the Netflix prize challenge 2006 due to its effectiveness in recommender systems..

Adjacency-based Similarity

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

- Drawbacks:
  - $O(|V|^2)$ runtime. (Must consider all node pairs.)
    - Can make $O(|E|)$ by only summing over non-zero edges and using regularization (e.g., Ahmed et al., 2013)
  - $O(|V|)$ parameters! (One learned vector per node).
  - Only considers direct, local connections.



e.g., the blue node is obviously more similar to green compared to red node, despite none having direct connections.

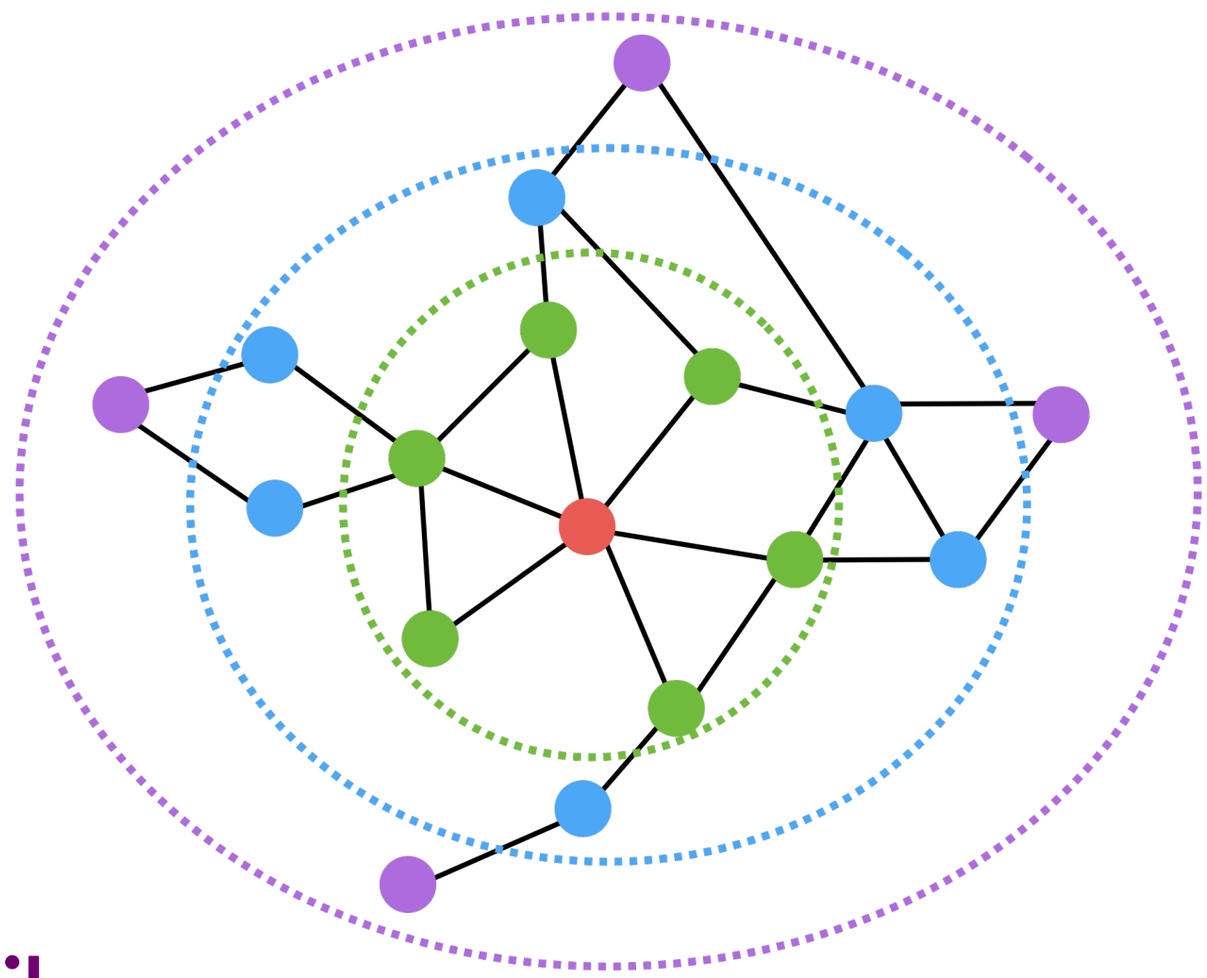
Multi-hop Similarity

Material based on:

- Cao et al. 2015. GraRep: Learning Graph Representations with Global Structural Information. *CIKM.*
- Ou et al. 2016. Asymmetric Transitivity Preserving Graph Embedding. *KDD.*
- Jian Tang, Meng Qu, Mingzhe Wang, Jun Yan, Ming Zhang and Qiaozhu Mei. LINE: Large-scale Information Network Embedding . WWW'15

Multi-hop Similarity

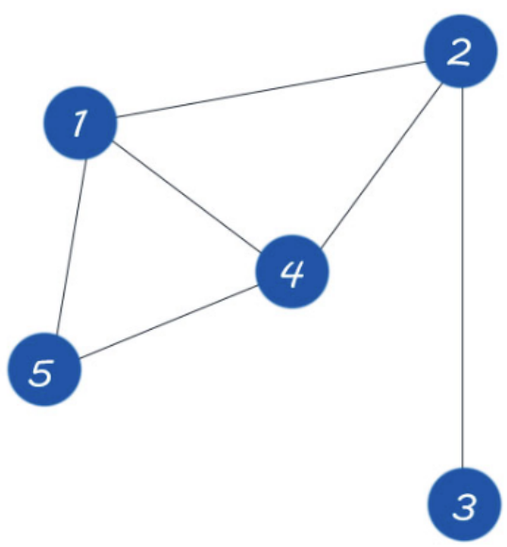
- Idea: Consider k-hop node neighbors.
- E.g., two or three-hop neighbors.



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

Power of djacency matrix

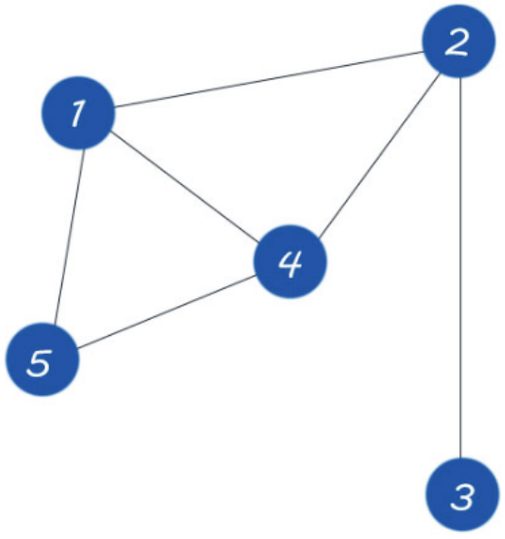
- If A^k gives us the number of walks from node i to node j after k steps.



$$A = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 1 & 1 & 0 \\ 3 & 0 & 1 & 0 & 0 & 0 \\ 4 & 1 & 1 & 0 & 0 & 1 \\ 5 & 1 & 0 & 0 & 1 & 0 \end{array}$$

Power of djacency matrix

- If A^k gives us the number of walks from node i to node j after k steps.



$$A^3 = \begin{pmatrix} 4 & 6 & 1 & 5 & 5 \\ 6 & 2 & 3 & 6 & 2 \\ 1 & 3 & 0 & 1 & 2 \\ 5 & 6 & 1 & 4 & 5 \\ 5 & 2 & 2 & 5 & 2 \end{pmatrix}$$

Multi-hop Similarity

- Basic idea: $$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}^k\|^2$$
- Train embeddings to predict k-hop neighbors.

Multi-hop Similarity

- Basic idea:

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

- In practice (GraRep from Cao et al, 2015):
  - Use log-transformed, **probabilistic adjacency matrix**:

$$\tilde{\mathbf{A}}_{i,j}^k = \max\left(\log\left(\frac{(\mathbf{A}_{i,j}/d_i)}{\sum_{l \in V}(\mathbf{A}_{l,j}/d_l)^k}\right)^k - \alpha, 0\right)$$

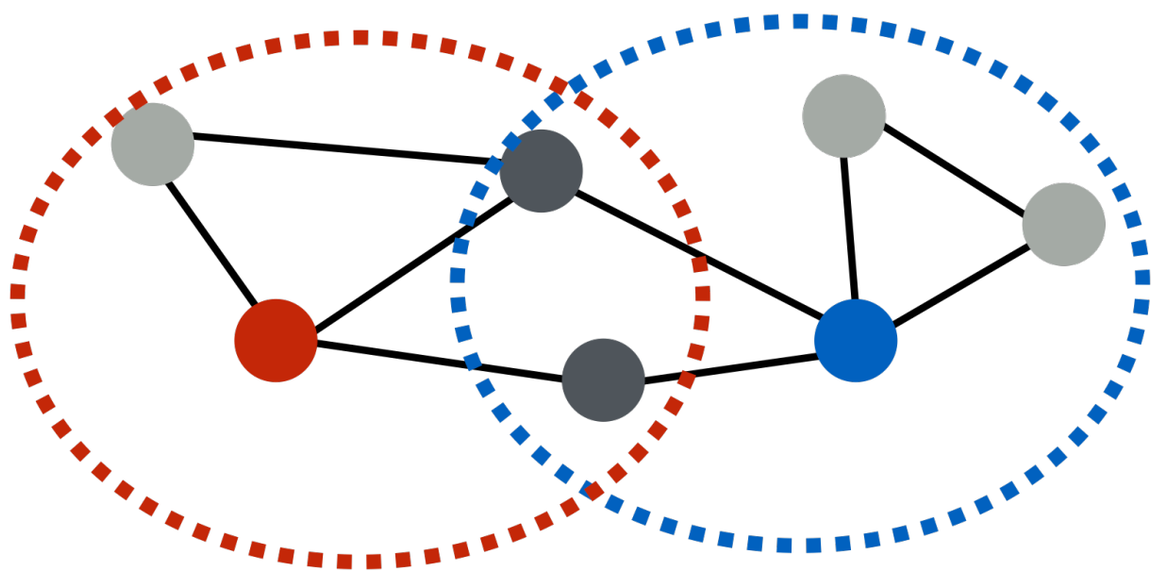
node degree

constant shift

  - Train multiple different hop lengths and concatenate output.

Multi-hop Similarity

- Another option: **Measure overlap between node neighborhoods.**



- Example overlap functions:
  - Jaccard similarity $$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Multi-hop Similarity

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

embedding similarity

multi-hop network similarity (i.e., any neighborhood overlap measure)

- $S_{u,v}$ is the neighborhood overlap between **u** and **v** (e.g., Jaccard overlap).
- This technique is known as HOPE (Yan et al., 2016).

Summary

- Basic idea so far:
    - 1) Define pairwise node similarities.
    - 2) Optimize low-dimensional embeddings to approximate these pairwise similarities.
- Issues:
    - Expensive: Generally $O(|V|^2)$, since we need to iterate over all pairs of nodes.
    - Brittle: Must hand-design deterministic node similarity measures.
    - Massive parameter space: $O(|V|)$ parameters

Random Walk Approaches

Material based on:

- Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations. *KDD.*
- Grover et al. 2016. node2vec: Scalable Feature Learning for Networks. *KDD.*

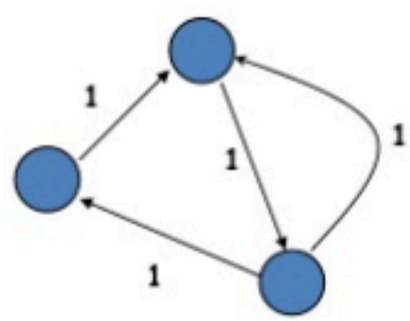
Transition Matrix

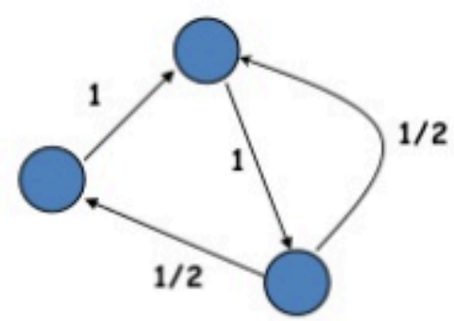
$$\begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{matrix}$$

Adjacency matrix A

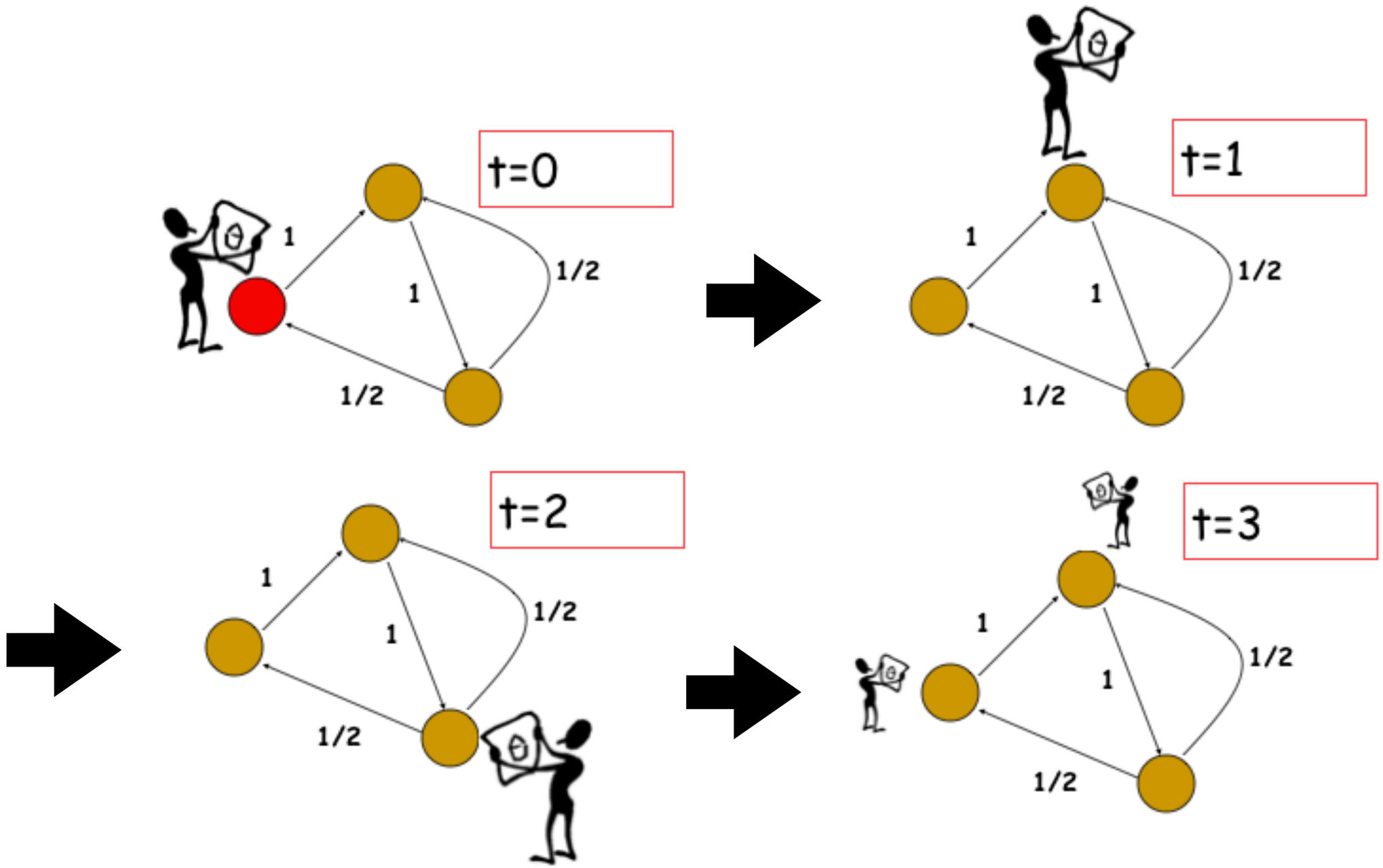
$$\begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1/2 & 1/2 & 0 \end{matrix}$$

Transition matrix P





Random Walk

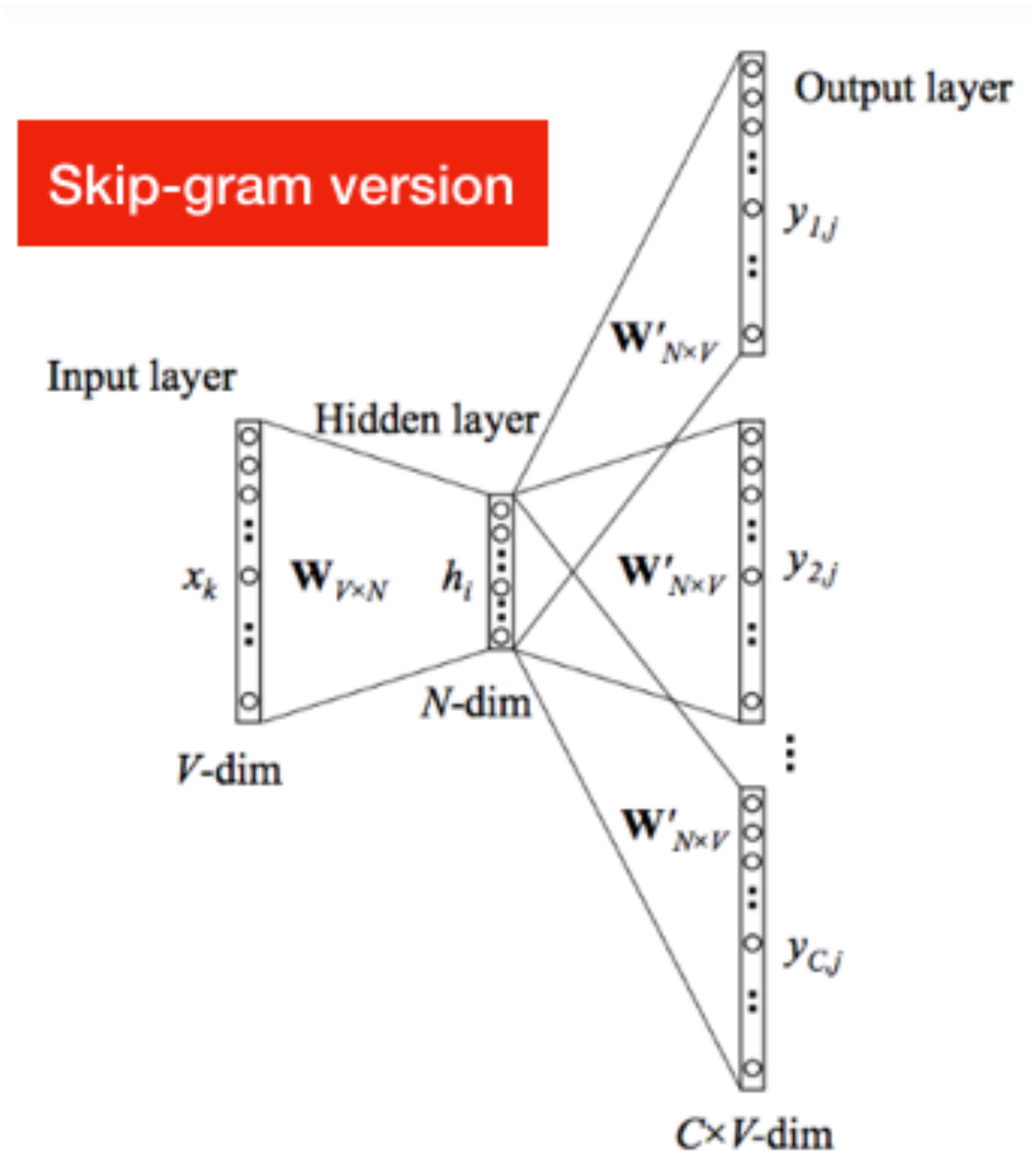


Random Walk Embeddings

$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$

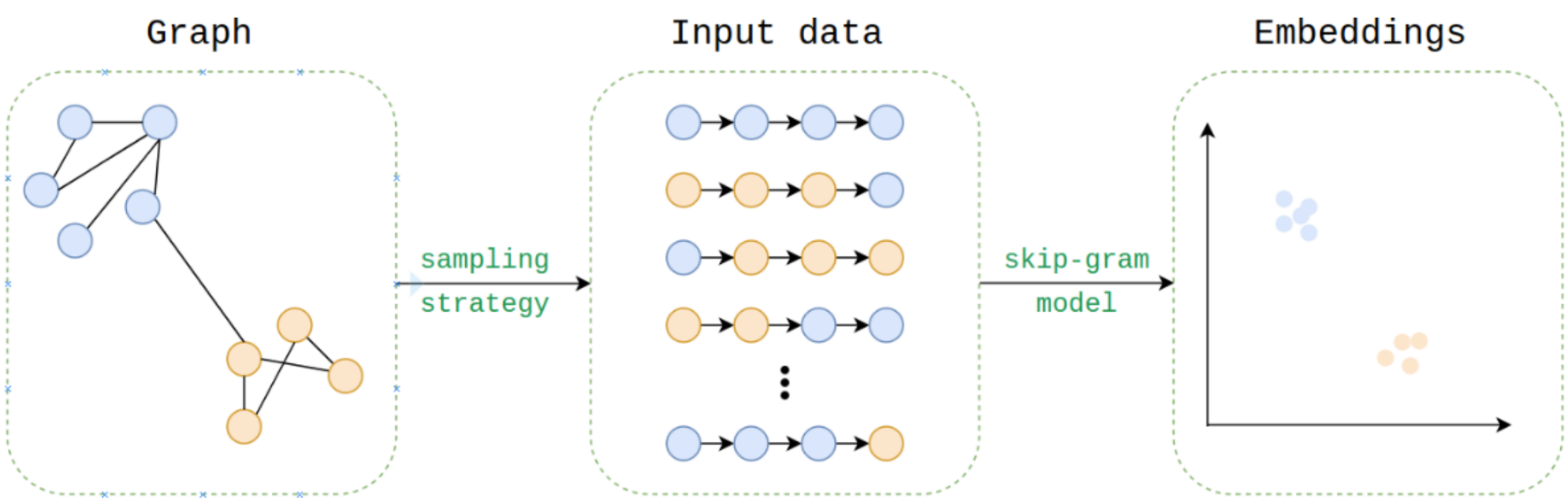
probability that $\mathbf{u}$ and $\mathbf{v}$ co-occur on a random walk over the network

recap: Skip-gram model

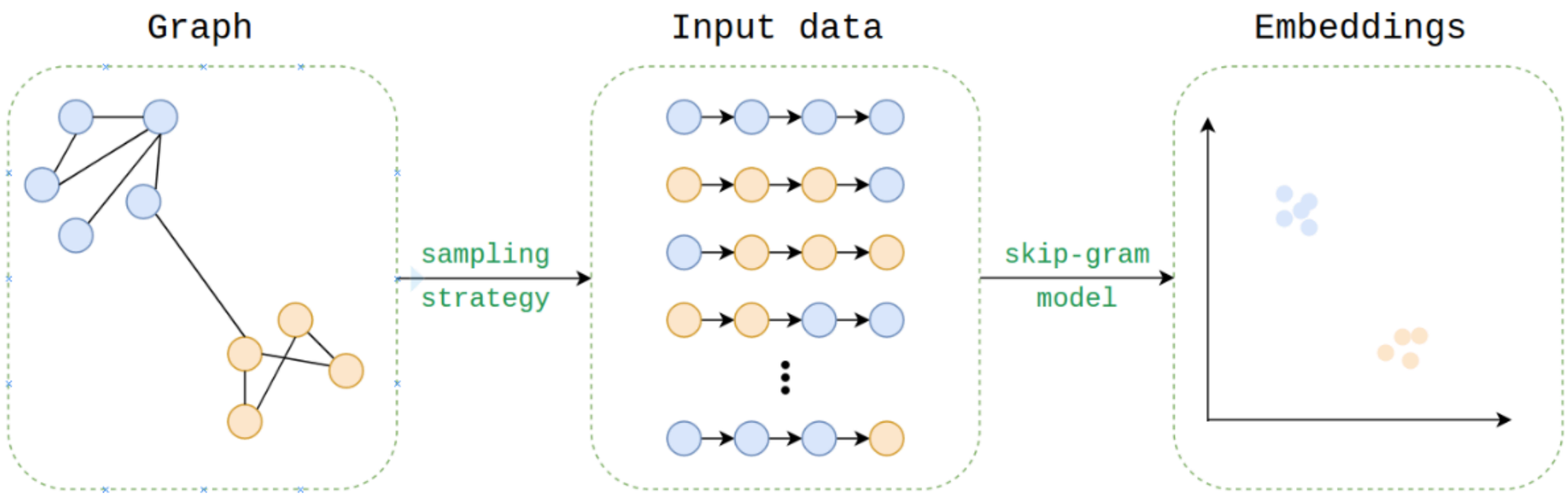


(recap) word2vec

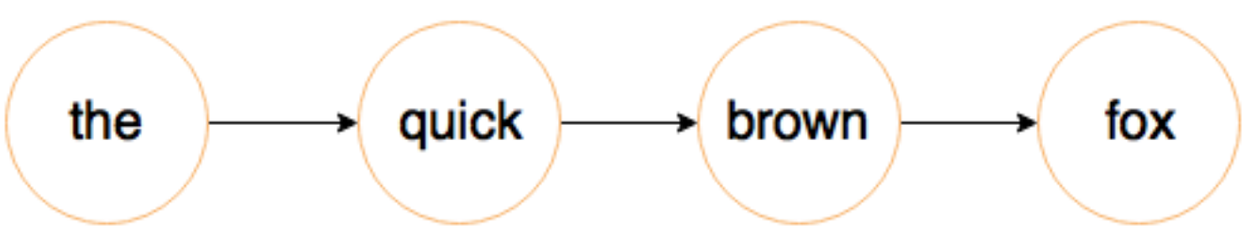
- Node2vec is similar to word2vec skip-gram model.
- The same way as a document is an ordered sequence of words, one could sample sequences of nodes from the underlying network and turn a network into a ordered sequence of nodes.



(recap) word2vec

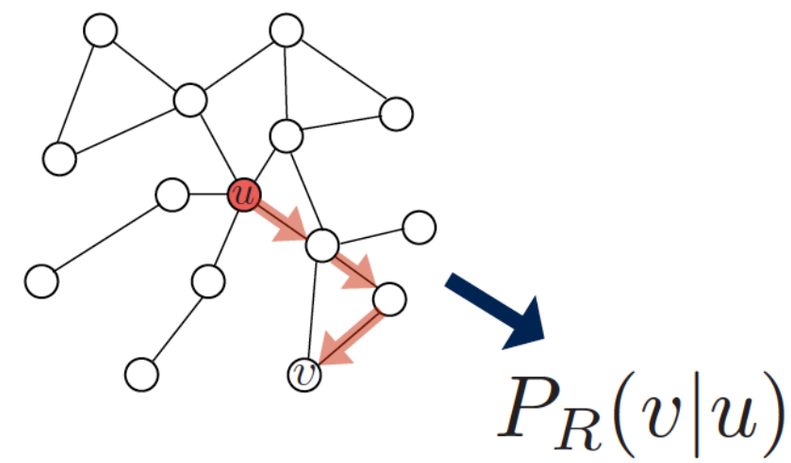


- word2vec can embed a very specific graphs:

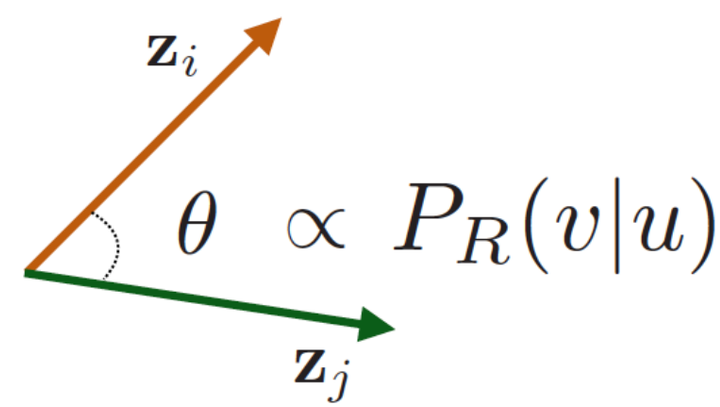


Random Walk Embeddings

1. Estimate **probability of visiting node v on a random walk starting from node u** using some random walk strategy R.



2. Optimize embeddings to encode these random walk statistics.



Why Random walks?

1. **Expressivity**: Flexible stochastic definition of node similarity that incorporates both **local** and **higher-order** neighborhood information.

2. **Efficiency**: Do not need to consider all node pairs when training; only need to consider **pairs that co-occur on random walks**.

Random Walk Optimization

1. Run short random walks starting from each node on the graph using some strategy R.

2. For each node u collect $N_R(u)$, the multiset* of nodes visited on random walks starting from u.

3. Optimize embeddings to according to:

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

* $\mathbf{N_R(u)}$ can have repeat elements since nodes can be visited multiple times on random walks.

Random Walk Optimization

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

- Intuition: Optimize embeddings to **maximize likelihood of random walk co-occurrences**.

Random Walk Optimization

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

- Intuition: Optimize embeddings to **maximize likelihood of random walk co-occurrences**.

- Parameterize $P(v \mid z_u)$ using softmax:

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

Random Walk Optimization

- Putting things together:

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

sum over all nodes $\mathbf{u}$

sum over nodes $\mathbf{v}$ seen on random walks starting from $\mathbf{u}$

predicted probability of $\mathbf{u}$ and $\mathbf{v}$ co-occuring on random walk

- Optimizing random walk embeddings = Finding embeddings z_u that minimize $\mathcal{L}$

Random Walk Optimization

- But doing this naively is too expensive!!

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

Nested sum over nodes
gives $O(|V|^2)$ complexity!!

Random Walk Optimization

- But doing this naively is too expensive!!

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

The normalization term from the softmax
can we approximate it?

Random Walk Optimization

- But doing this naively is too expensive!!

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

The normalization term from the softmax can we approximate it?

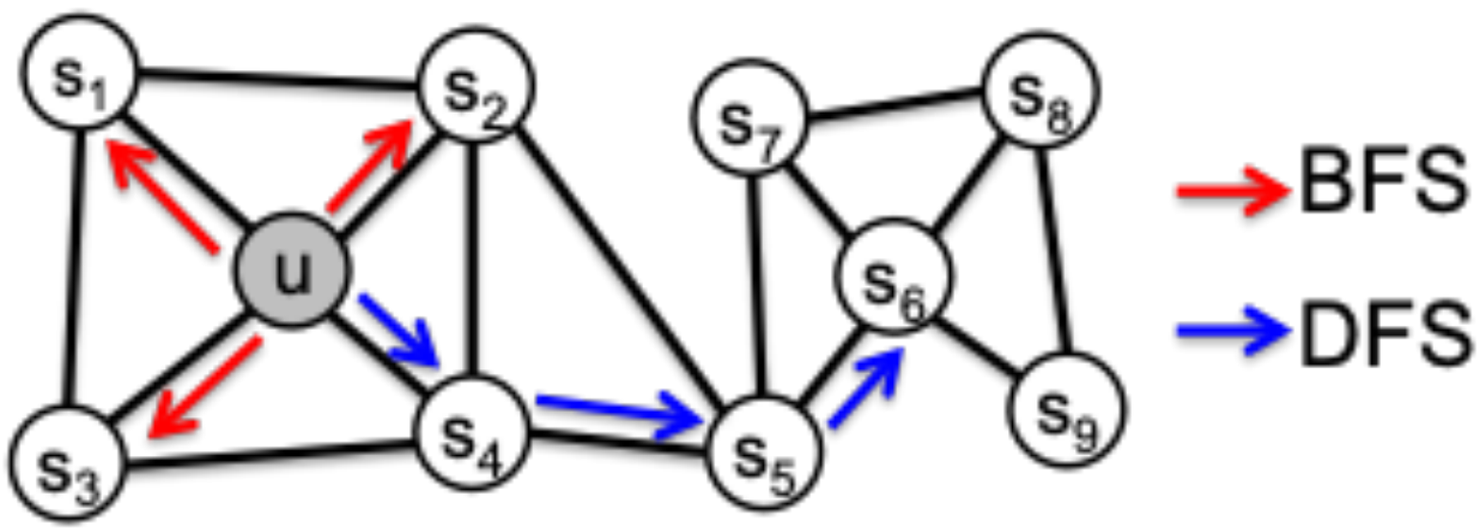
Negative Sampling

How we should randomly walk?

- So far we have described how to optimize embeddings given random walk statistics.
- What strategies should we use to run these random walks?
  - Simplest idea: Just run **fixed-length**, unbiased random walks starting from each node (i.e., DeepWalk from Perozzi et al., 2013).
  - But can we do better?

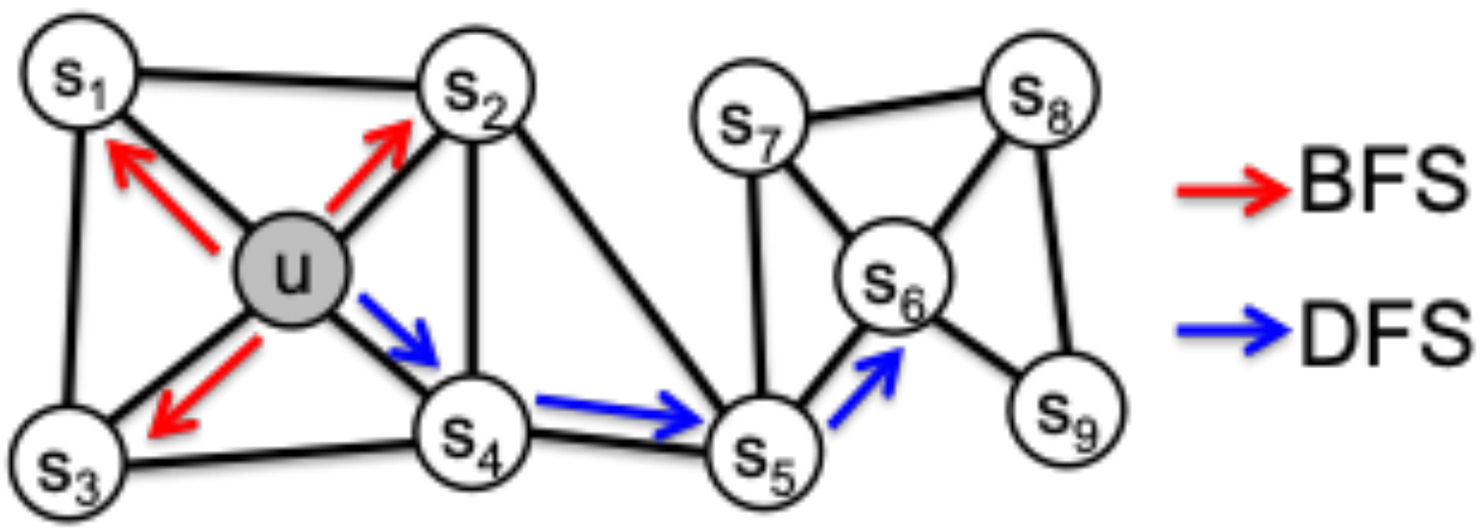
node2vec: Biased Walks

- Idea: use flexible, biased random walks that can trade off between **local** and **global** views of the network (Grover and Leskovec, 2016).



node2vec: Biased Walks

- Two classic strategies to define a neighborhood of a given node :



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

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

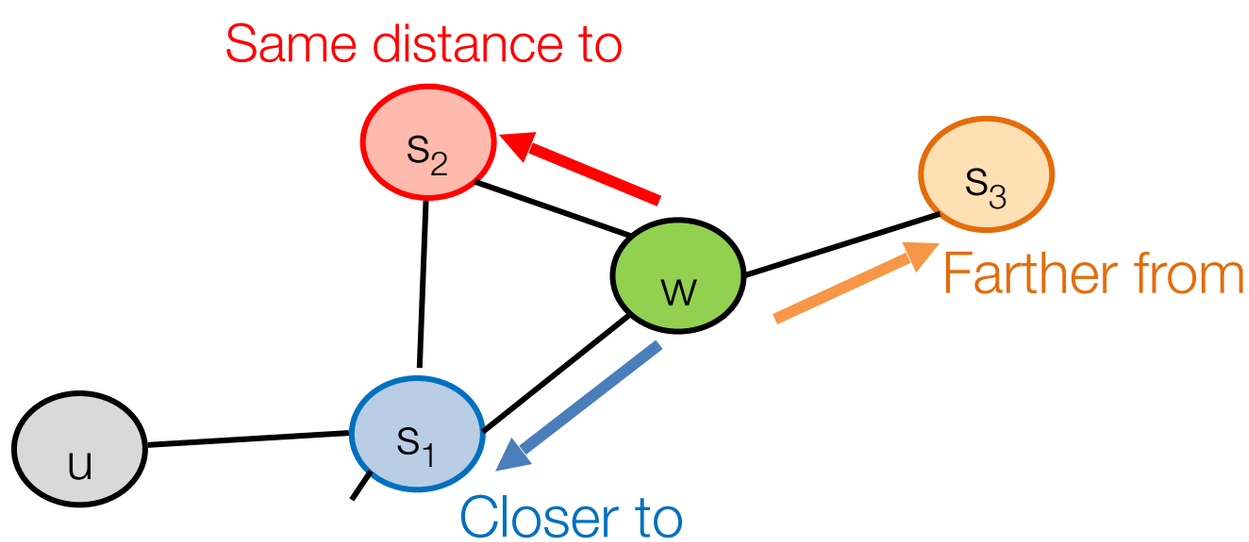
node2vec sampling strategy

- Node2vec's sampling strategy, accepts 4 arguments:
  - **Number of walks**: Number of random walks to be generated from each node in the graph
  - **Walk length**: How many nodes are in each random walk
  - **P**: Return hyperparameter
  - **q**: Inout hyperaprameter ("walk away" hyperaprameter)

  and also the standard skip-gram parameters (context window size, number of iterations etc.)

Biased Random Walks

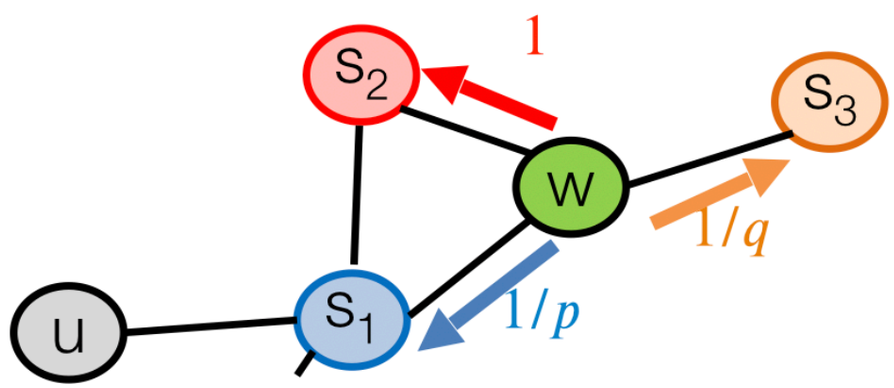
- Biased 2nd-order random walks explore network neighborhoods:
  - Random walk started at u and is now at w
  - Insight: Neighbors of w can only be:



Idea: Remember where that walk came from

Biased Random Walks

- Walker is at w. Where to go next?

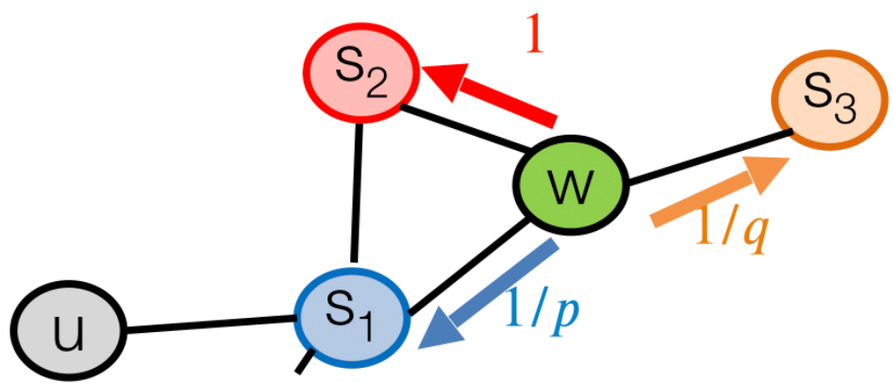


1/p, 1/q, 1 are unnormalized probabilities

- p, q model transition probabilities
  - p … return parameter
  - q … ”walk away” parameter

Biased Random Walks

- Walker is at w. Where to go next?



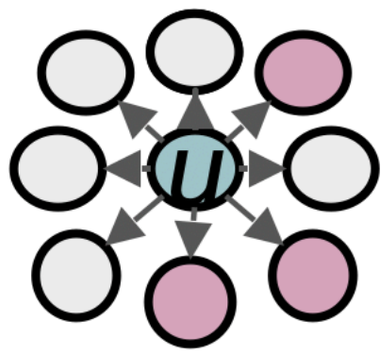
	s_1	$1/p$
w →	s_2	1
	s_3	$1/q$

Unnormalized transition prob.

- BFS-like walk: Low value of p
- DFS-like walk: Low value of q

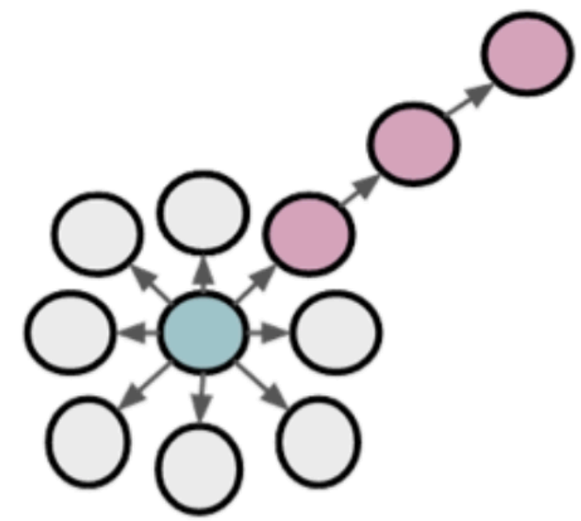
Ns(u) are the nodes visited by the walker

BFS vs. DFS



BFS:
Micro-view of neighbourhood

P

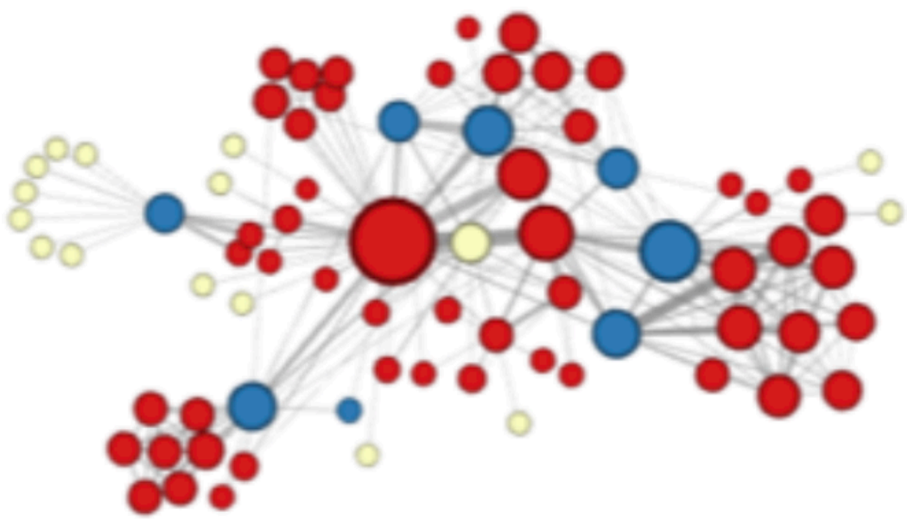


DFS:
Macro-view of neighbourhood

q

Experiments: Micro vs. Macro

- Interactions of characters in a novel:



p=1, q=2
Microscopic view of the network neighbourhood

p=1, q=0.5
Macroscopic view of the network neighbourhood

Other random walk ideas

- Different kinds of biased random walks:
  - Based on node attributes (Dong et al., 2017).
  - Based on a learned weights (Abu-El-Haija et al., 2017)
- Alternative optimization schemes:
  - Directly optimize based on 1-hop and 2-hop random walk probabilities (as in 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).

Summary

- **Basic idea**: Embed nodes so that distances in embedding space reflect node similarities in the original network.
  - Different notions of node similarity:
    - **Adjacency-based** (i.e., similar if connected)
    - **Multi-hop similarity definitions.**
    - **Random walk approaches**.

So what method should I use..?

- No one method wins in all cases….
  - e.g., node2vec performs better on node classification while multi-hop methods performs better on link prediction (Goyal and Ferrara, 2017 survey).
- Random walk approaches are generally more efficient (i.e., $O(|E|)$ vs. $O(|V|^2)$)
- In general: Must choose a node similarity that matches application!

Thursday!

Graph Neural Networks
&
Graph Convolutional Networks