LECTURE 13: GRAPH REPRESENTATION LEARNING

Prof. Pan Hui

CSIT 6000K: Social Networks and Social Computing: A Data Science Perspective

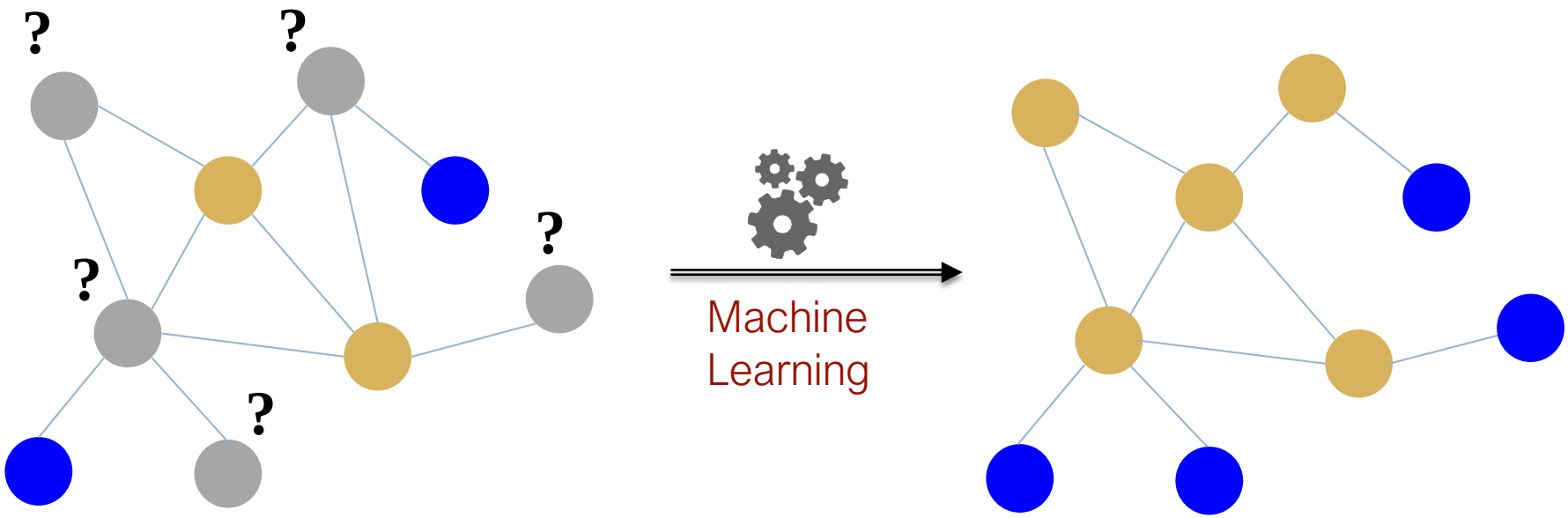
Thursdays 07:30 PM - 10:20 PM

Machine Learning with Networks

Classical ML tasks in networks:

- 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

Example: Node Classification



Example: Node Classification

Classifying the function of proteins in the interactome!

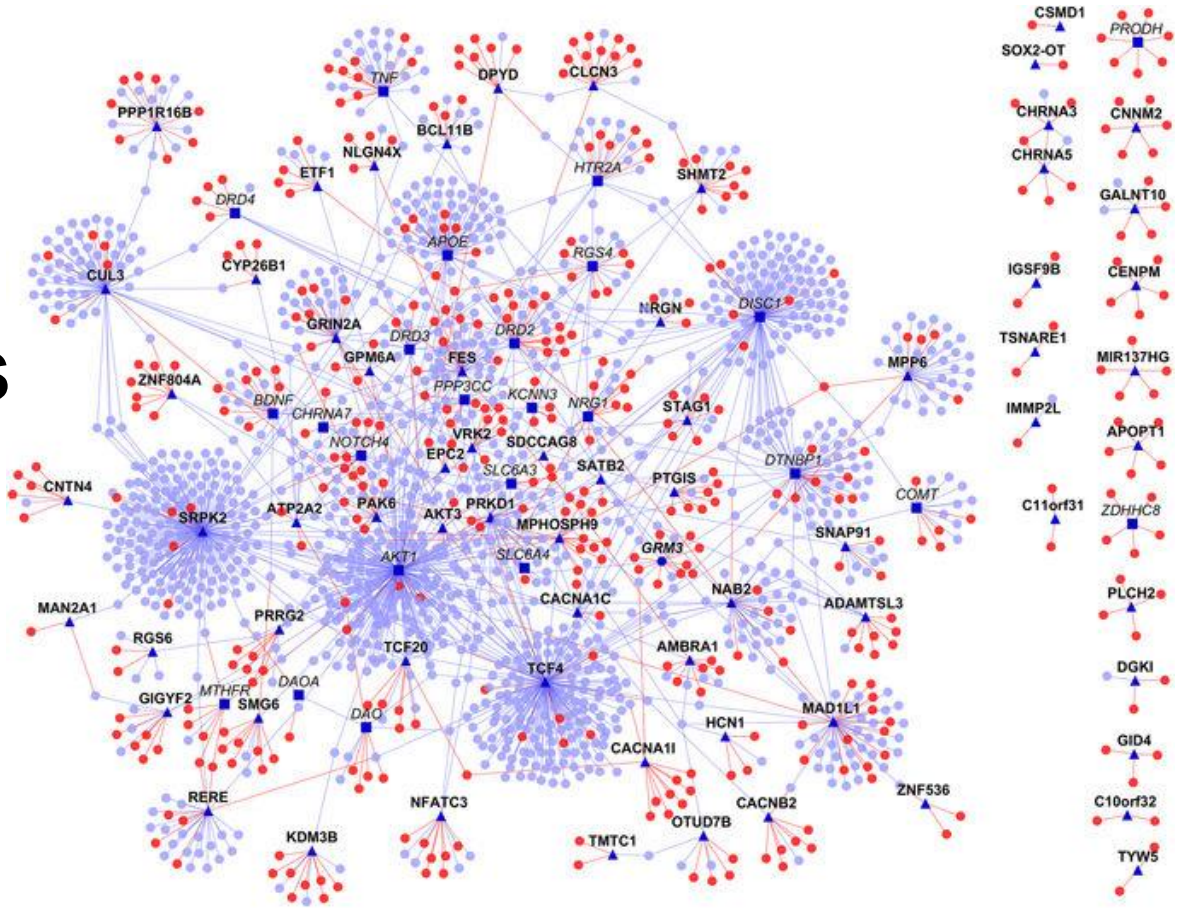
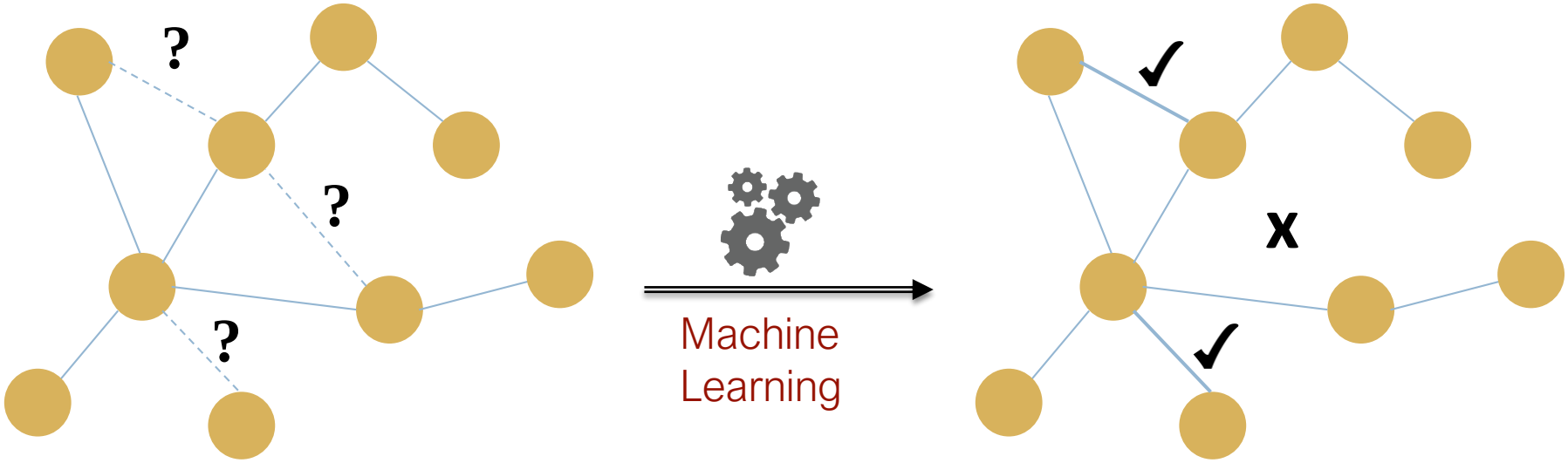


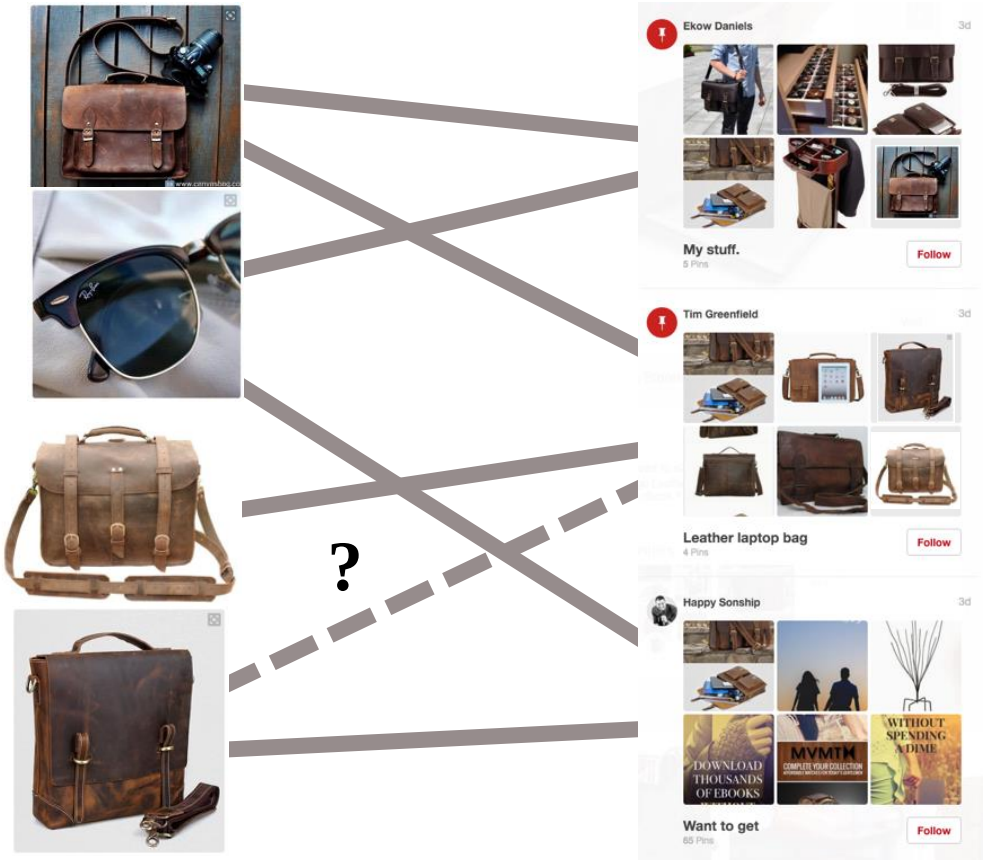
Image from: Ganapathiraju et al. 2016. Schizophrenia interactome with 504 novel protein–protein interactions. *Nature*.

Example: Link Prediction



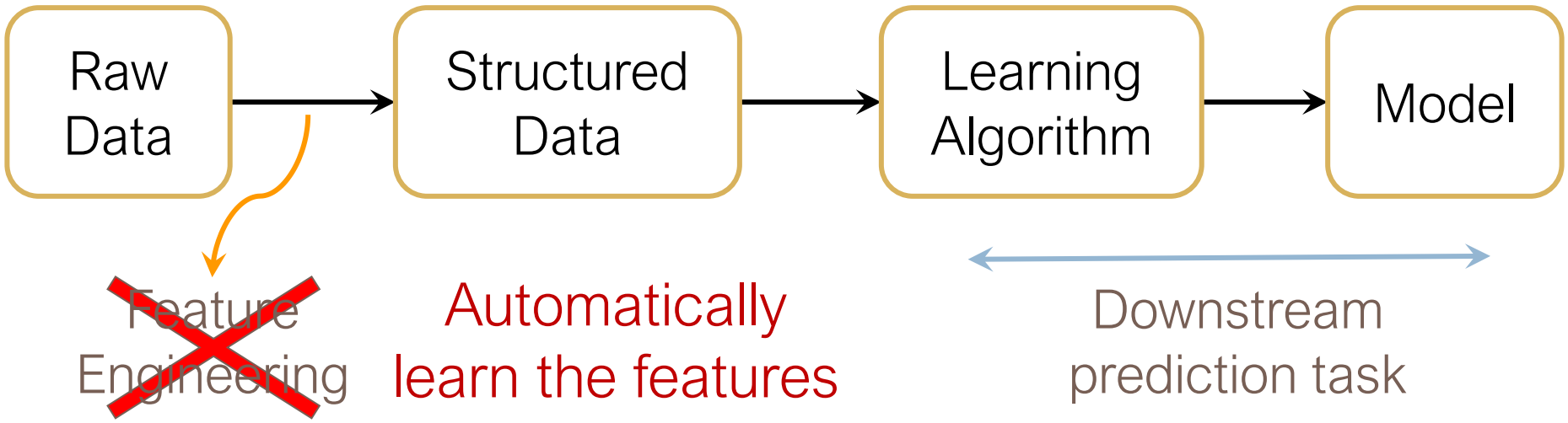
Example: Link Prediction

Content recommendation is link prediction!



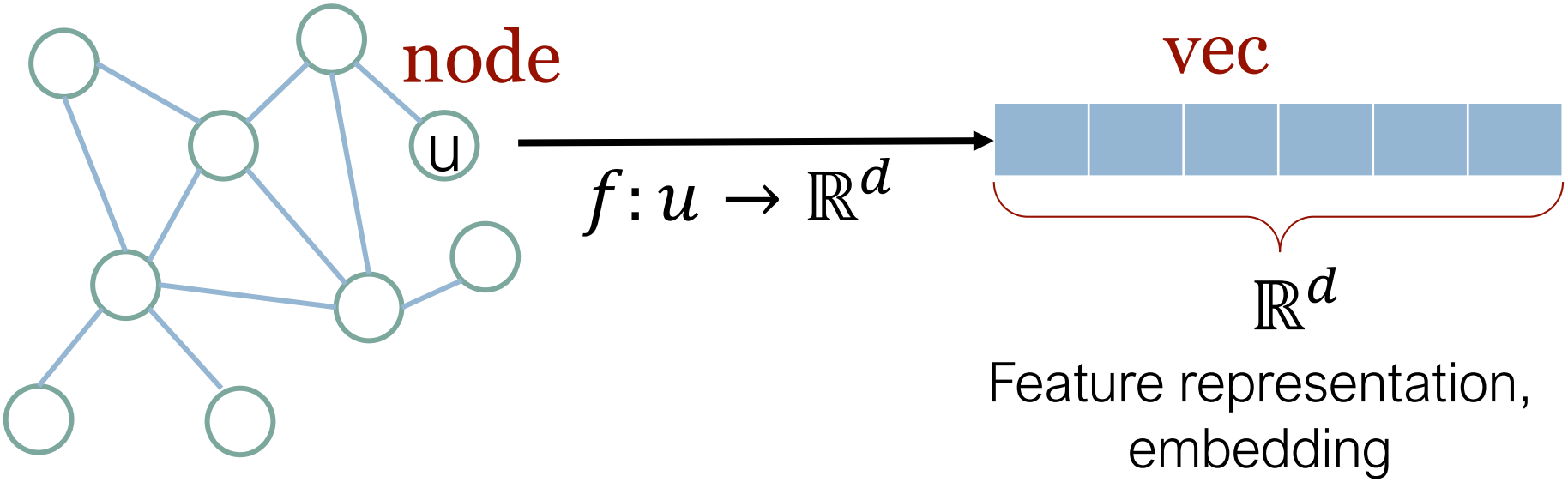
Machine Learning Lifecycle

- (Supervised) Machine Learning Lifecycle: This feature, that feature. Every single time!



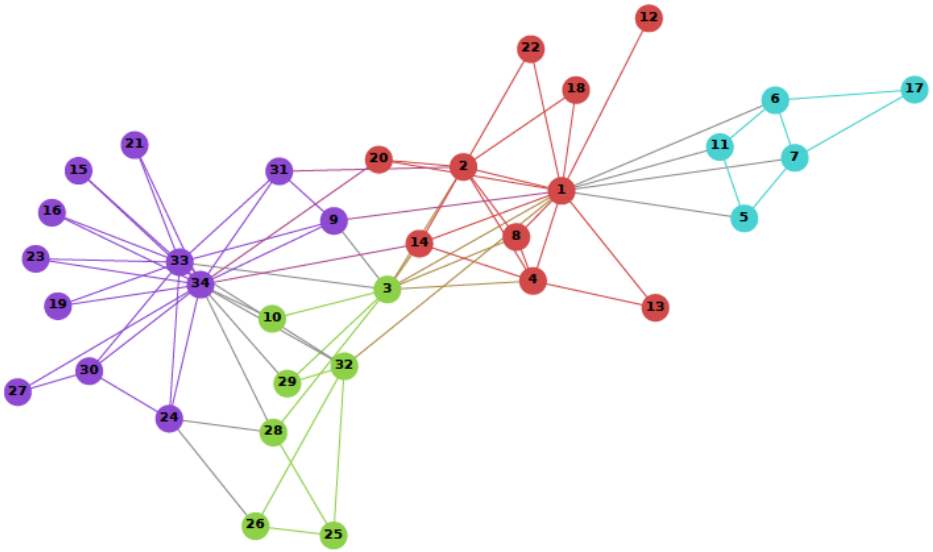
Feature Learning in Graphs

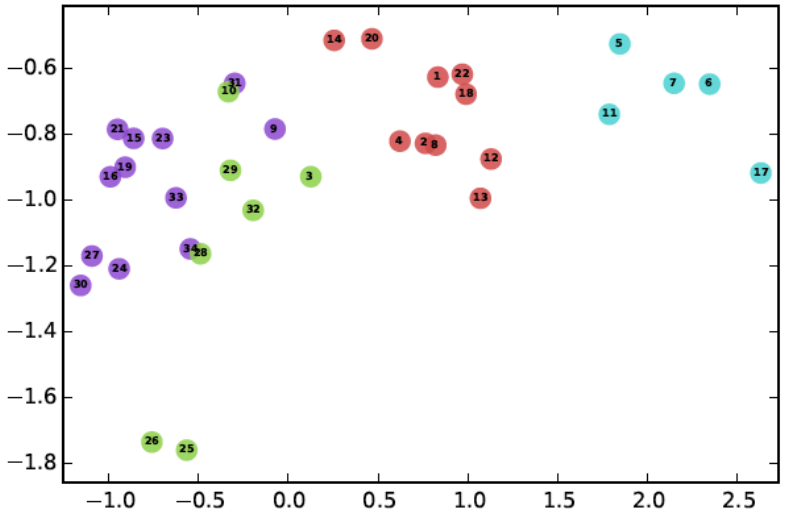
Goal: Efficient task-independent feature learning for machine learning in networks!



Example

- Zachary's Karate Club Network:





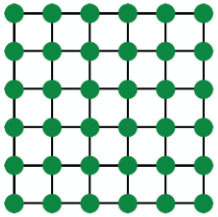
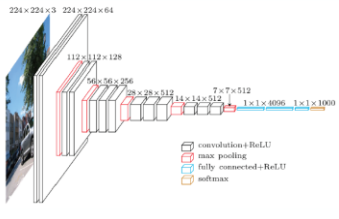
Input

Output

Image from: Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations. *KDD.*

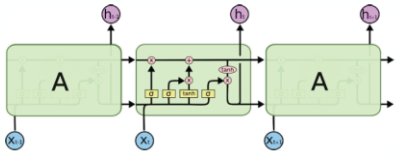
Why Is It Hard?

- Modern deep learning toolbox is designed for simple sequences or grids.
  - CNNs for fixed-size images/grids….

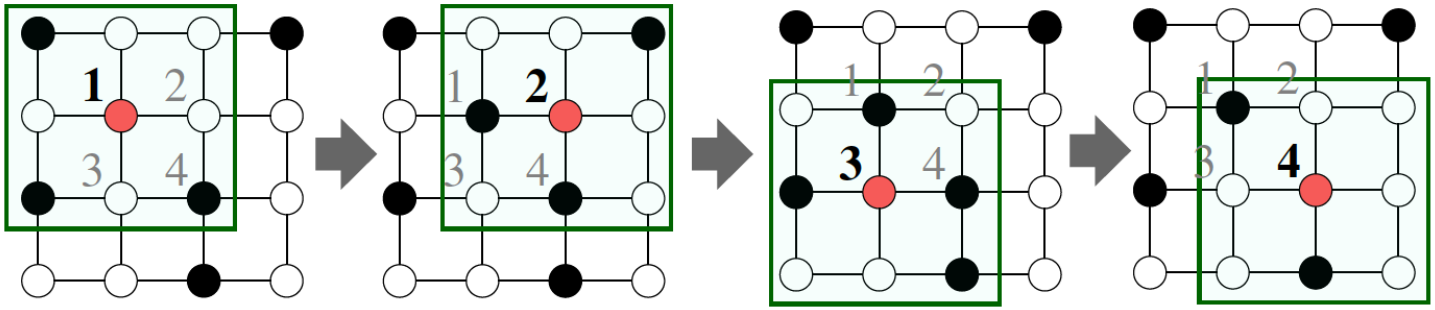
  - RNNs or word2vec for text/sequences…





Why Is It Hard?

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

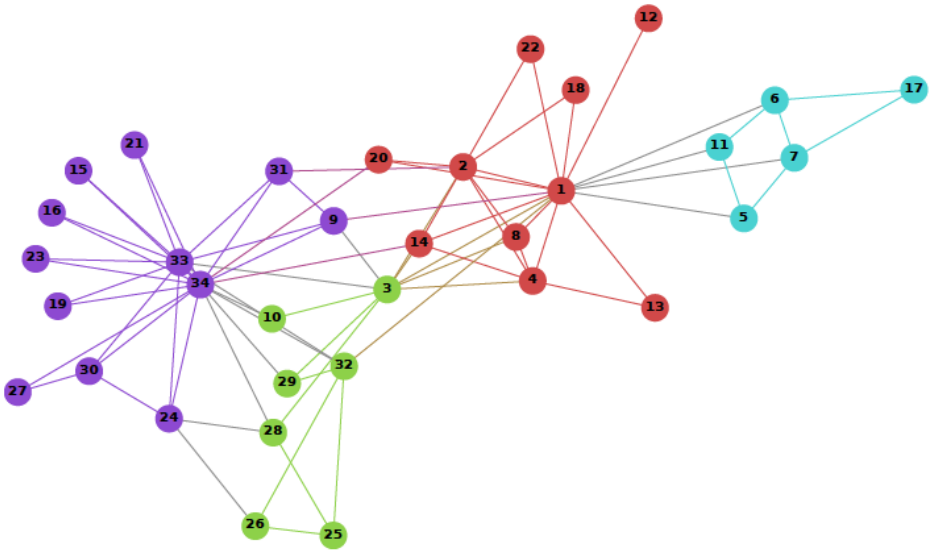


  - No fixed node ordering or reference point (i.e., the isomorphism problem)
  - Often dynamic and have multimodal features.

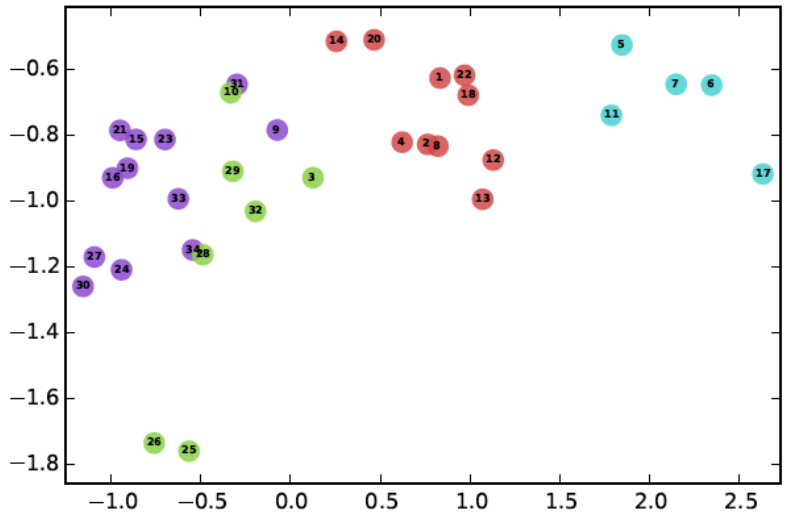
NODE EMBEDDINGS

4/21/2022

Embedding Nodes



Input



Output

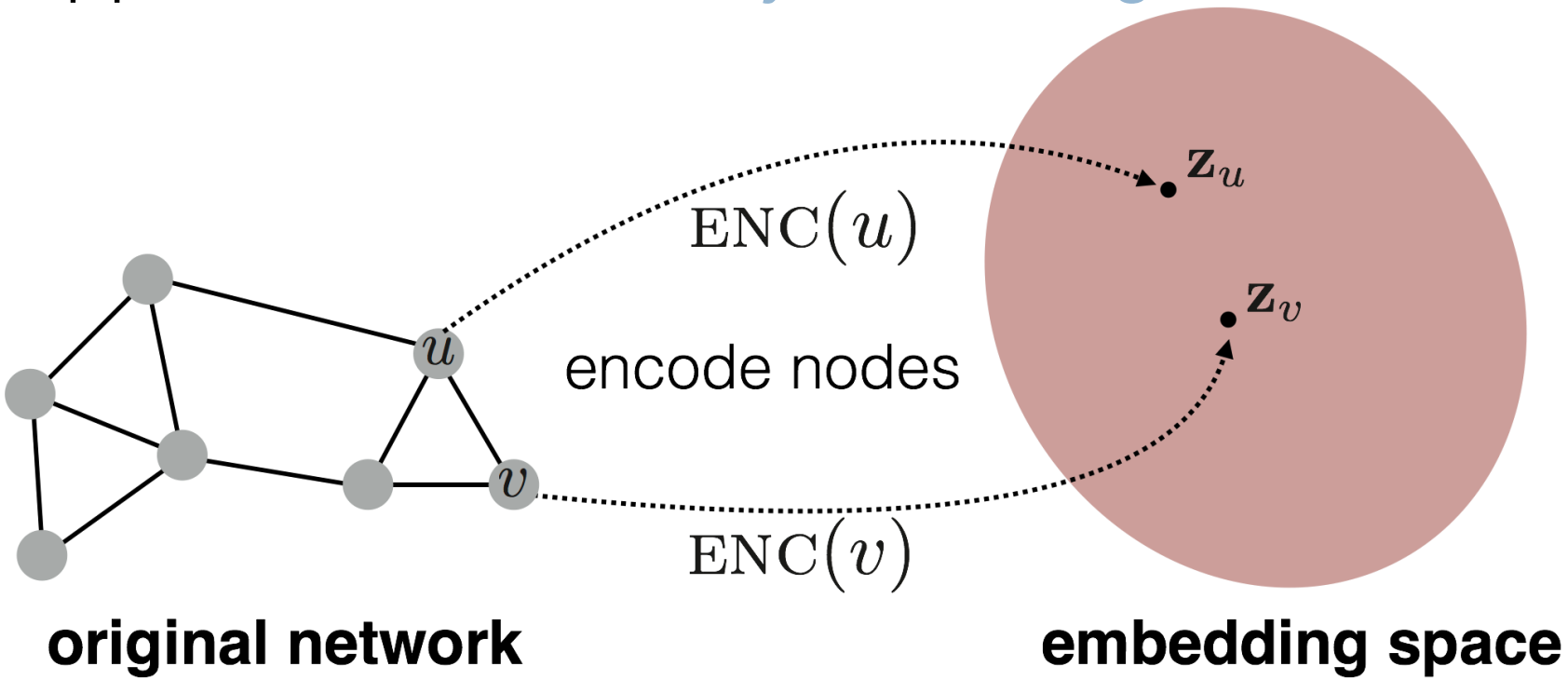
Intuition: Find embedding of nodes to d-dimensions so that "similar" nodes in the graph have embeddings that are close together.

Setup

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

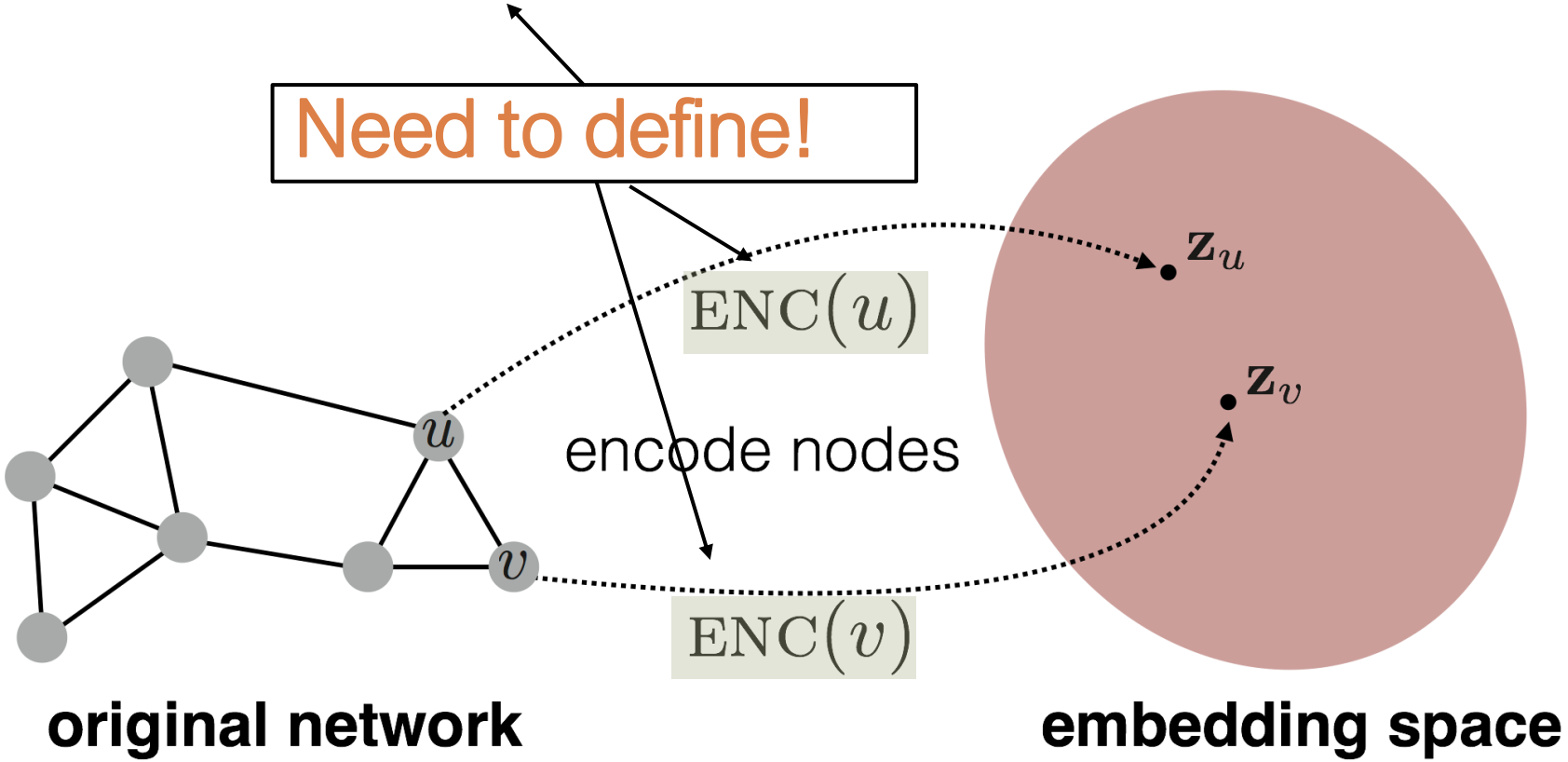
Embedding Nodes

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



Embedding Nodes

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:**

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

Two Key Components

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

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

d-dimensional embedding

node in the input graph

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

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

Similarity of u and v in the original network

dot product between node embeddings

"Shallow" Encoding

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

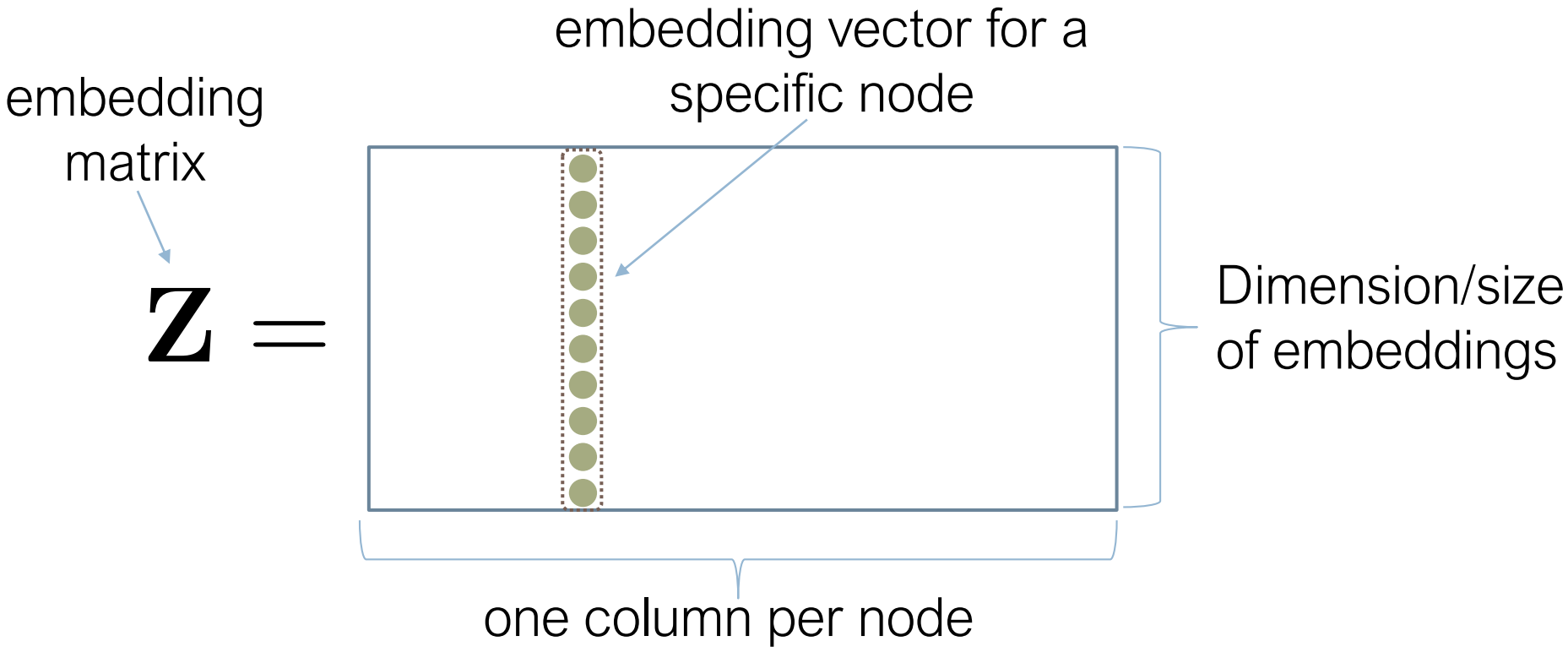
$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ 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**



“Shallow” Encoding

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

i.e., each node is assigned a unique embedding vector.

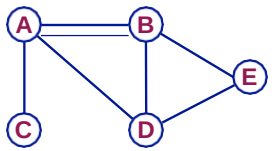
- E.g., node2vec, DeepWalk, LINE

How to Define Node Similarity?

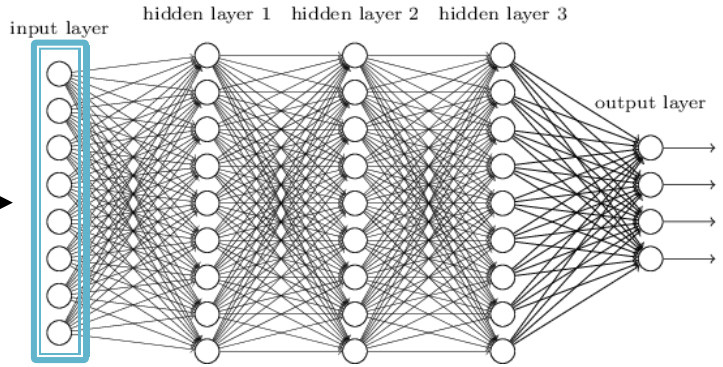
- Key distinction between "shallow" methods is **how they define node similarity.**
- E.g., should two nodes have similar embeddings if they….
  - are connected?
  - share neighbors?
  - have similar "structural roles"?
  - …?

A Näive Approach

- Join adjacency matrix and features¡
- Feed them into a deep neural net:



	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0



?

- Issues with this idea:
  - $O(N)$ parameters
  - Not applicable to graphs of different sizes
  - Not invariant to node ordering

Three Approaches

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

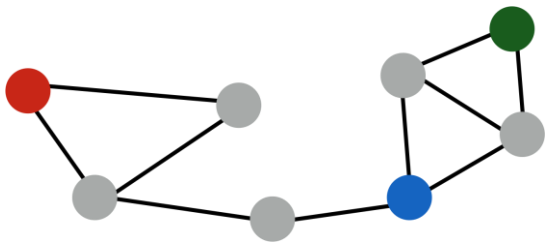
$$\mathcal{L} = \sum_{(u,v) \in V \times V} \left\| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v} \right\|^2$$

- Find embedding matrix $\mathbf{Z} \in \mathbb{R}^{\boldsymbol{d} \times |\boldsymbol{V}|}$ that minimizes the loss $\mathcal{L}$
  - E.g., Use stochastic gradient descent (SGD) as a general optimization method.
    - Highly scalable, general approach

Adjacency-based Similarity

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

- **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.

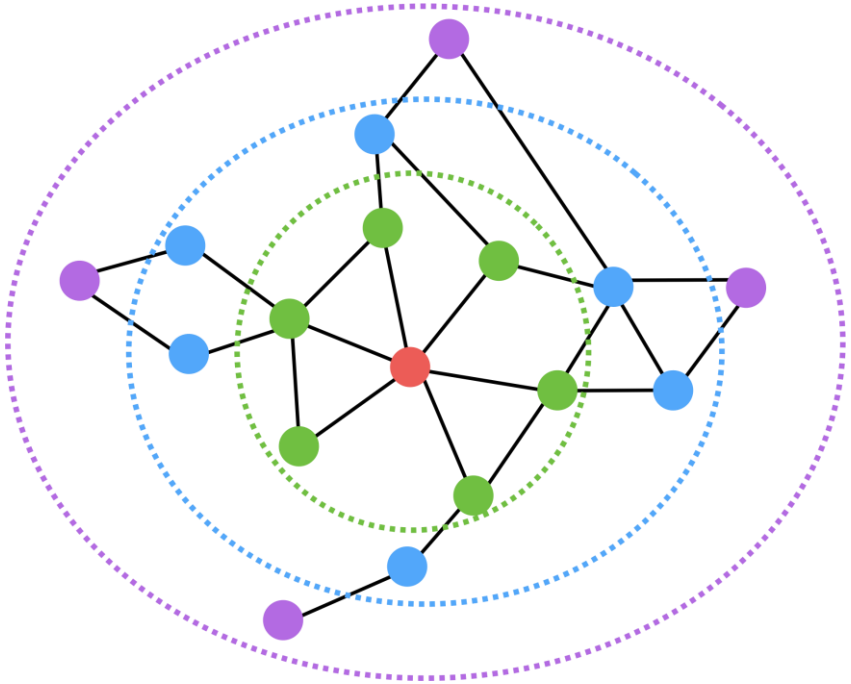
Multi-hop Similarity

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.*

Multi-hop Similarity

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



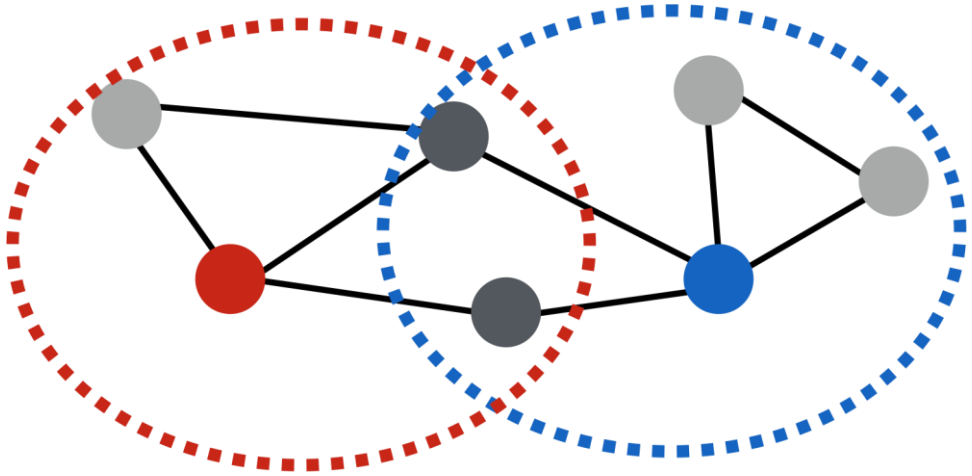
- Red: Target node
- Green: 1-hop neighbors
  - $\mathbf{A}$ (i.e., adjacency matrix)
- Blue: 2-hop neighbors
  - $\mathbf{A}^2$
- Purple: 3-hop neighbors
  - $\mathbf{A}^3$

Multi-hop Similarity

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

Multi-hop Similarity

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



- Example overlap functions:
  - Jaccard similarity

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).

Summary so far

- 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.*

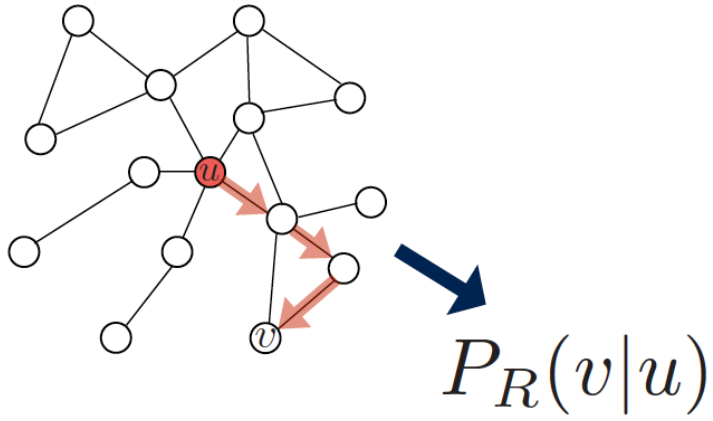
Random-walk Embeddings

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

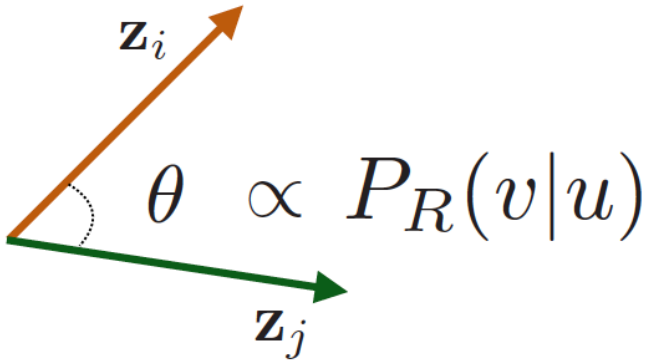
probability that u and v co-occur on a random walk over the network

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 enco 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 $\mathrm{N}_R(u)$, the multiset* of nodes visited on random walks starting from u.
3. Optimize embeddings to according to:

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

* $\mathrm{N}_R(u)$ 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.
- Parameterize $\mathrm{P}(v|\,\mathbf{z}_u)$ 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 u

sum over nodes v seen on random walks starting from u

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

Optimizing random walk embeddings =

Finding embeddings $\mathbf{z}_u$ that minimize $\mathbf{L}$

Random Walks: Summary

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

2. For each node u collect $\mathrm{N}_R(u)$, the multiset of nodes visited on random walks starting from u.

3. Optimize embeddings to according to:

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

Summary so far

- **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.

Summary so far

- **So what method should I use..?**
- No one method wins in all cases…. (Goyal and Ferrara, 2017 survey).
- Random walk approaches are generally more efficient (i.e., $O(|E|)$ vs. $O(|V|^2)$)
- **In general:** Must choose def'n of node similarity that matches application!