Data Mining and Analysis

Graphs (again)

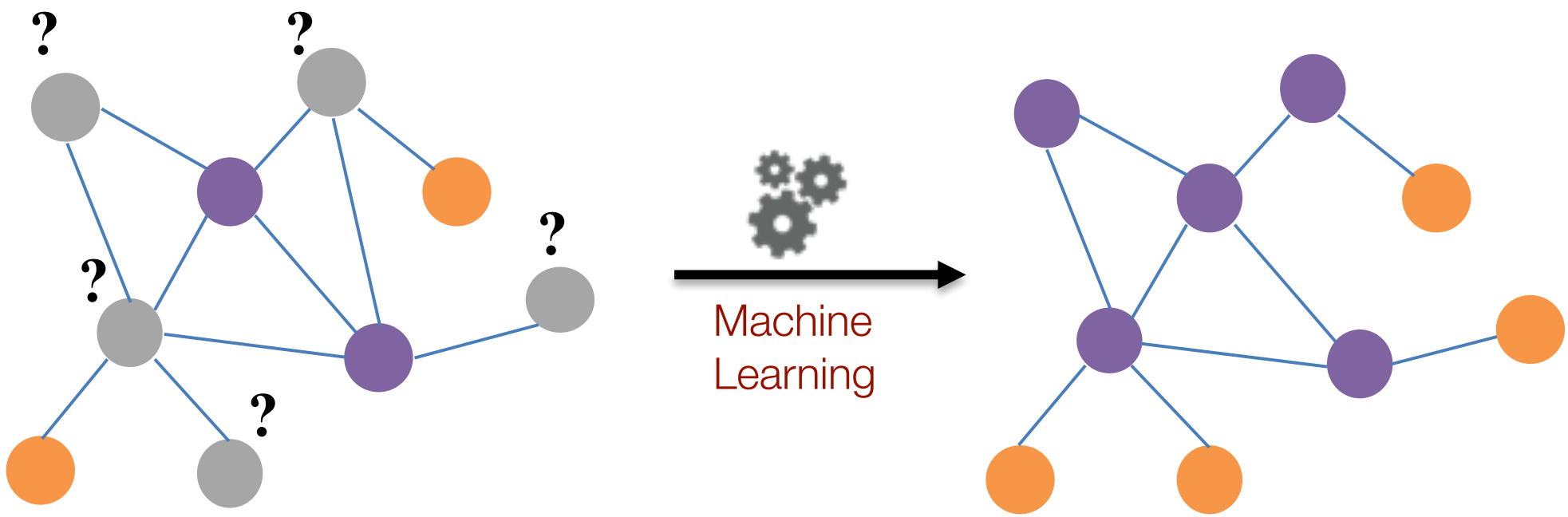
CSCE 676 :: Fall 2019
Texas A&M University
Department of Computer Science & Engineering
Prof. James Caverlee

Resources

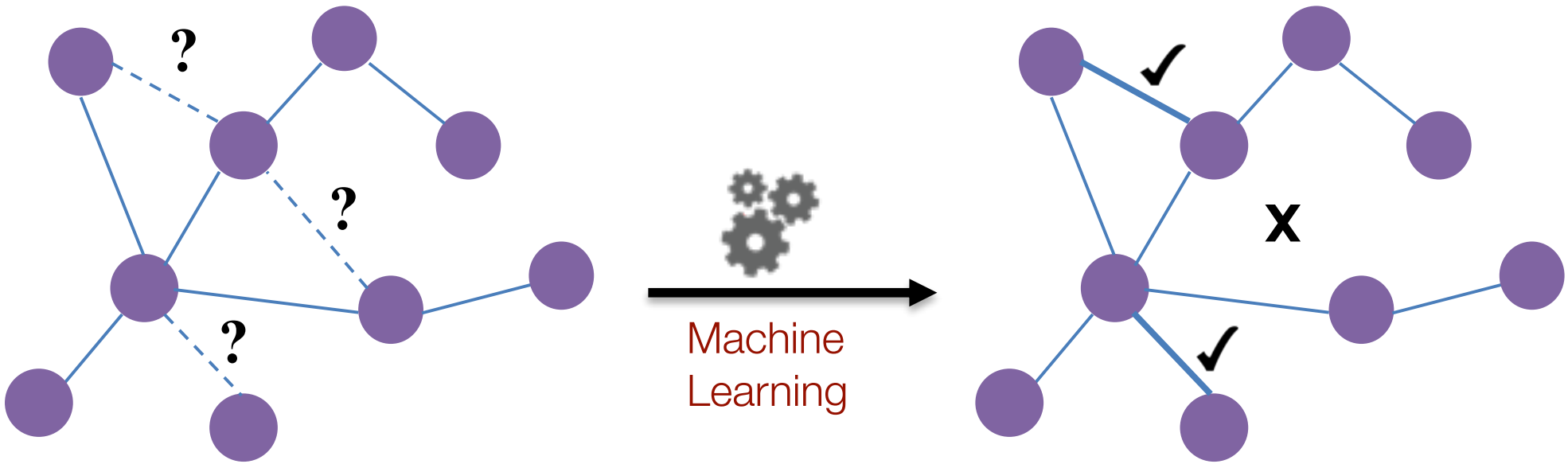
Hamilton et al. WWW Tutorial on Graph Representation Learning [http://snap.stanford.edu/proj/embeddings-www/]

Hamilton, Ying, Leskovec. Representation Learning on Graphs: Methods and Applications (IEEE Data Engineering Bulletin)

Node Classification

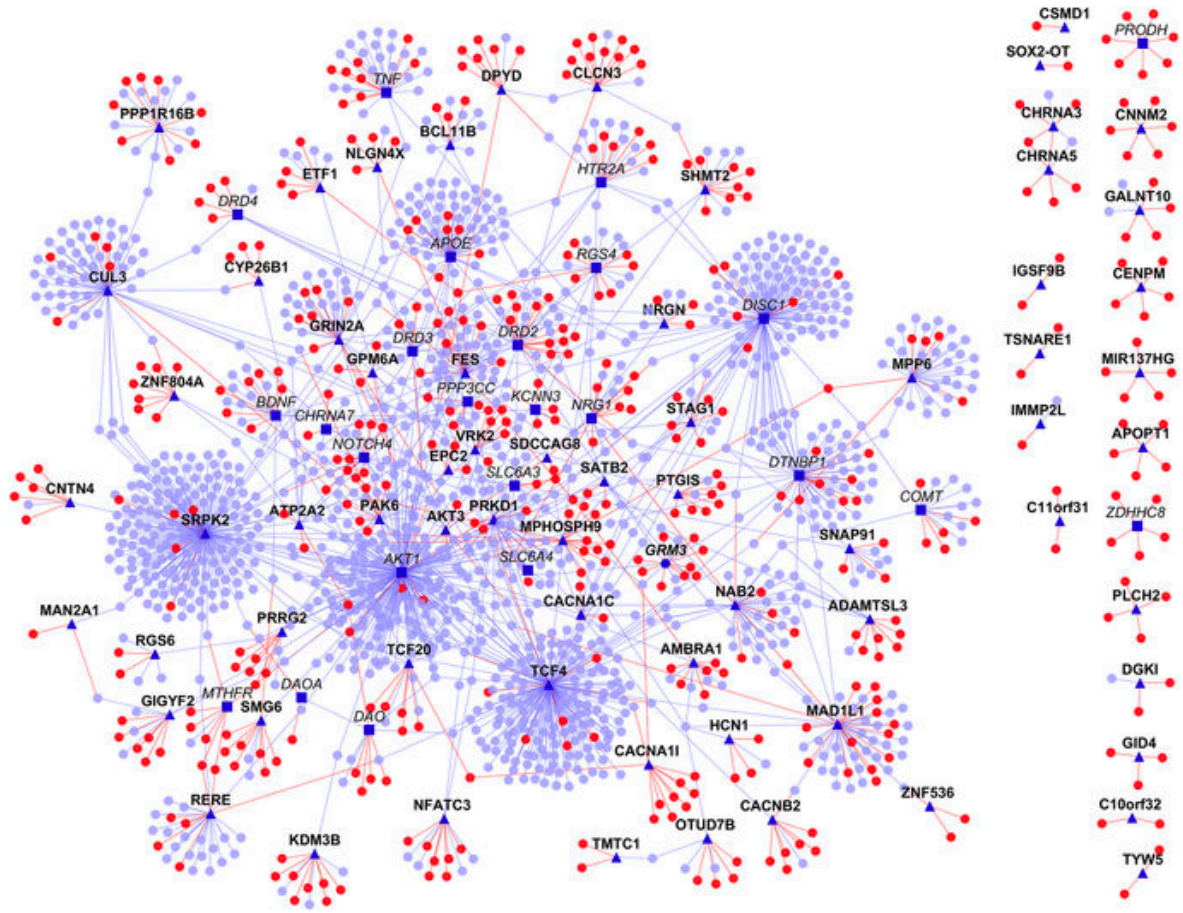


Example: Link Prediction



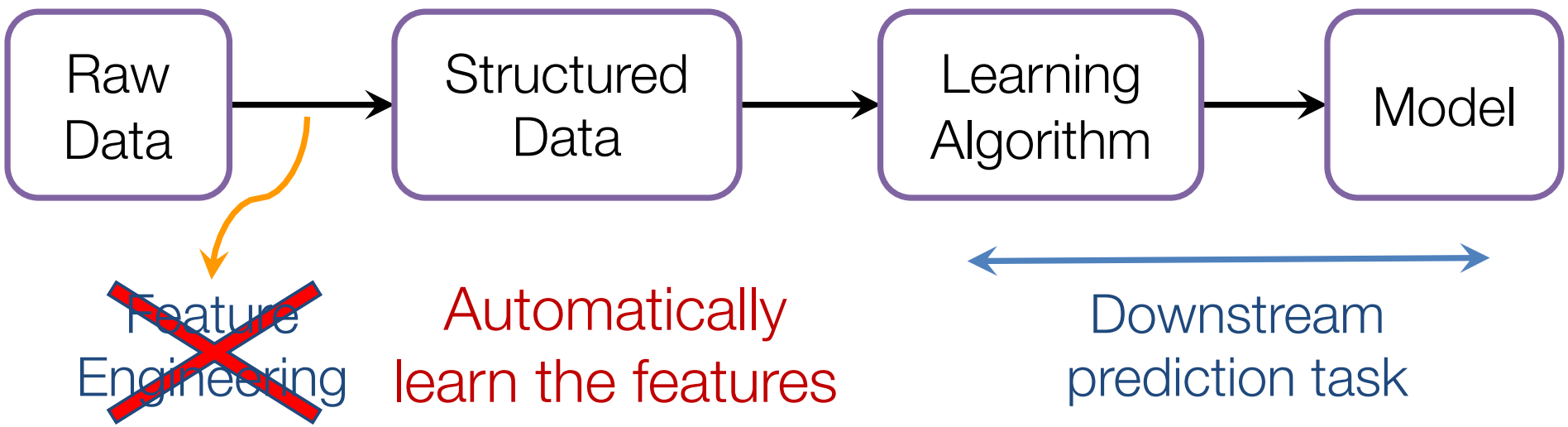
Example: Node Classification

Classifying the function of proteins in the interactome



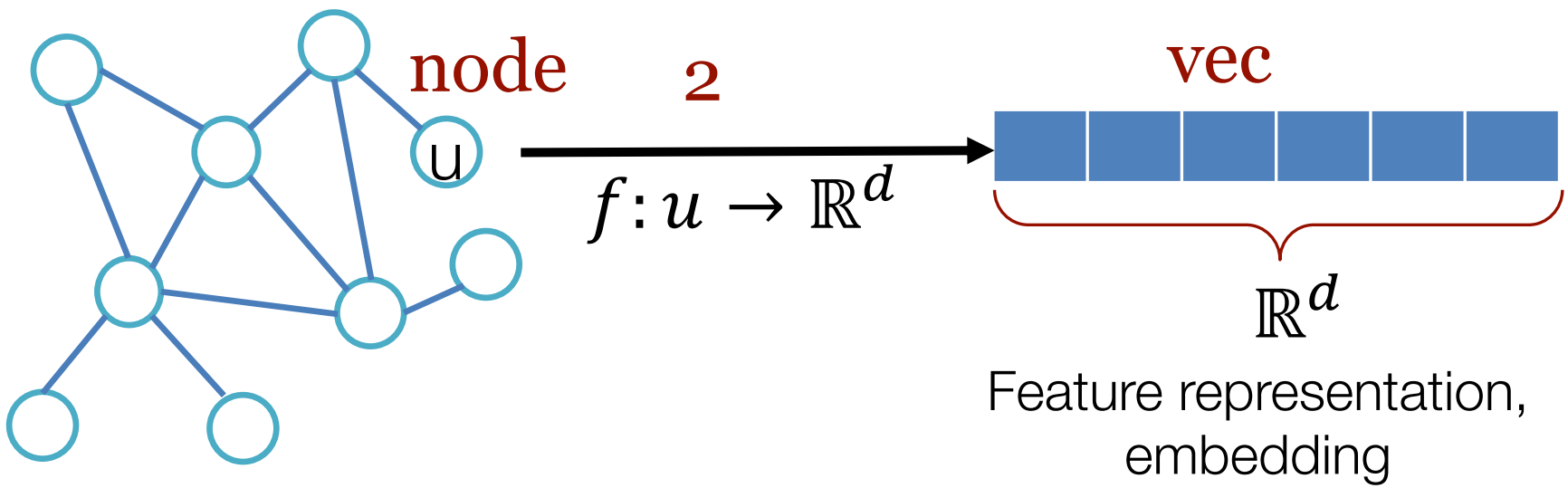
Machine Learning Lifecycle

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

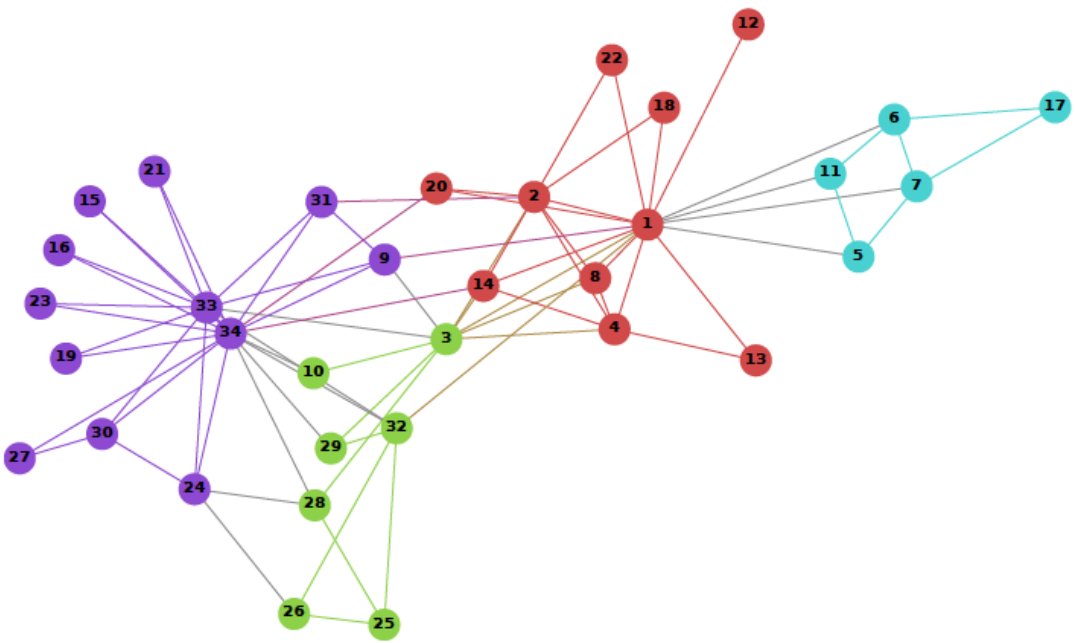


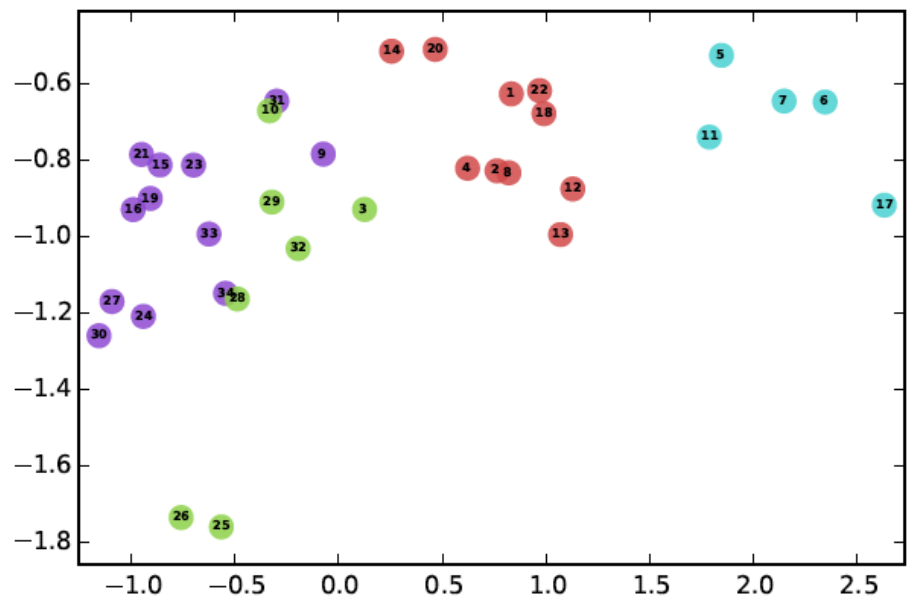
Feature Learning in Graphs

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



Example: Zachry's Karate Club





From DeepWalk (KDD 2014)

Network embedding

Task: We map each node in a network to a point in a low-dimensional space

- Distributed representation for nodes
- Similarity of embedding between nodes indicates their network similarity
- Encode network information and generate node representation

What can we do with embeddings?

Clustering/community detection: Cluster nodes/points based on embeddings

Node classification: Predict label of a node based on embeddings

Link prediction: Predict edge (i,j) based on pairwise embeddings

Embedding Nodes

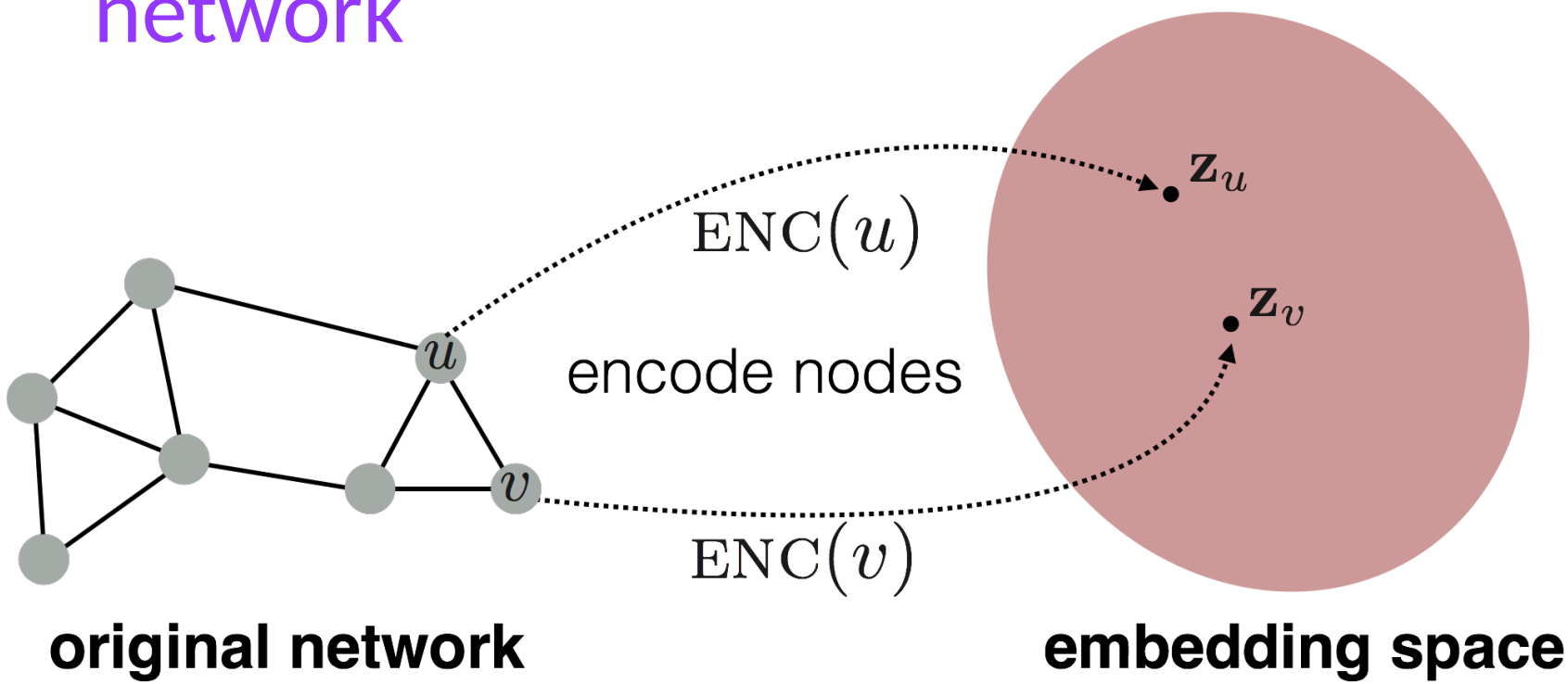
Setup

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

Embedding Nodes

Goal is to encode nodes so that similarity in the embedding space approximates similarity in the original network

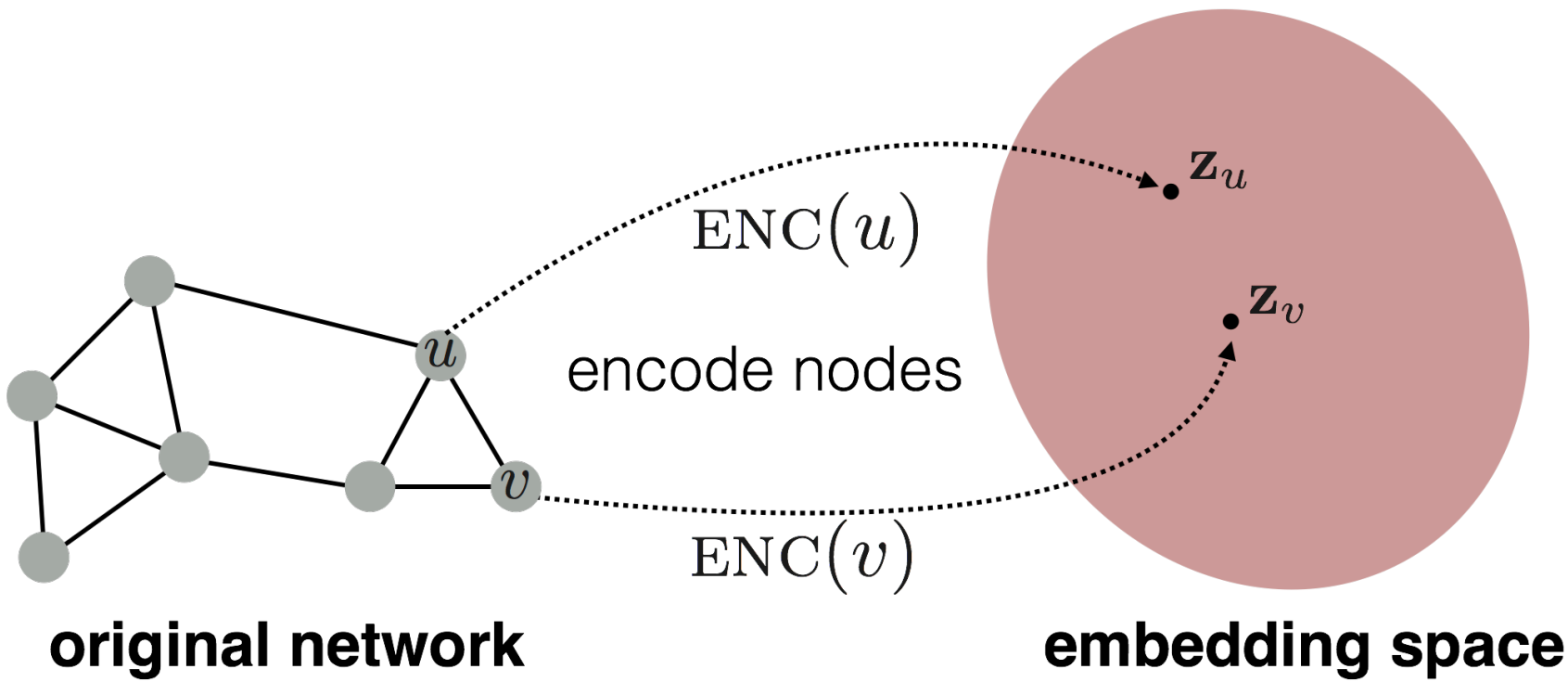


Embedding Nodes

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

Need to define:

Similarity and encodings



Learning Node Embeddings

Define an encoder

A mapping from nodes to embeddings

Define a node similarity function

A measure of similarity in the original network

Optimize the parameters of the encoder so that:

$$\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^\top \mathbf{z}_u}_{\text{Similarity of the embedding}}$$

Two Key Components

Encoder maps each node to a low-dimensional vector.

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

$\mathbf{d}$-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

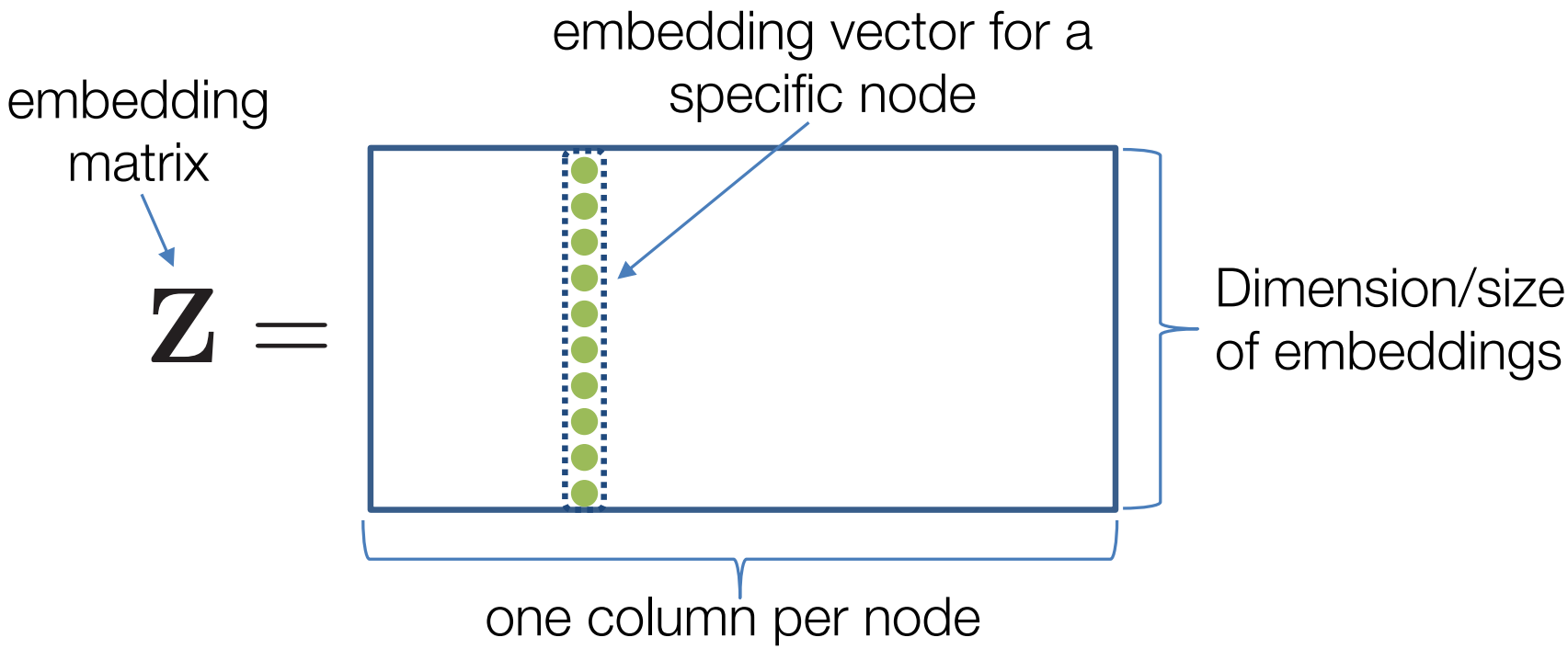
$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

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

Each node is assigned a unique embedding vector

Many methods: 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”?
- …?

Random Walk Approaches to Node Embeddings

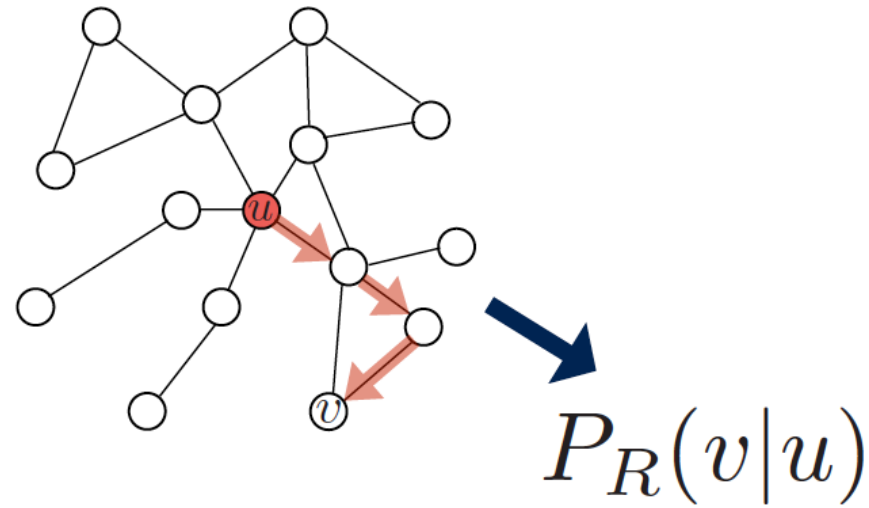
Random Walk Embeddings

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

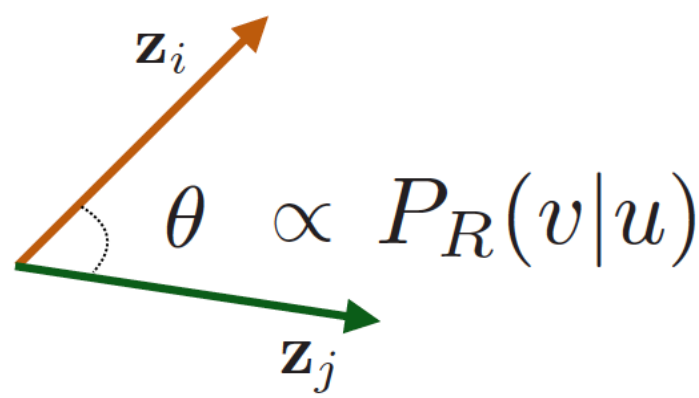
probability that u and v co-occur on a random walk over the network

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 encode these random walk statistics. (Cosine similarity encodes random walk "similarity")



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.

Unsupervised Feature Learning

Intuition: Find embedding of nodes to d-dimensional space so that node similarity is preserved

Idea: Learn node embedding such that nearby nodes are close together in the network

Given a node u, how do we define nearby nodes?

$N_R(u)$ ... neighborhood of u obtained by some strategy R

Feature Learning as Optimization

Given G= (V, E)

Our goal is to learn a mapping z:u $\rightarrow \mathbb{R}^d$

Log-likelihood objective:

$$\max_{z} \sum_{u \in V} \log P(N_R(u) | z_u)$$

where $N_R(u)$ is neighborhood of node u

Given node u, we want to learn feature representations predictive of nodes in its neighborhood $N_R(u)$

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: Given node u, predict its neighbors $N_R(u)$

$$\max_{\mathrm{z}} \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) \mid z_u)$$

Random Walk Optimization

$$\max_{\mathrm{z}} \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) |\, z_u)$$

Assumption: Conditional likelihood factorizes over the set of neighbors:

$$\log \mathrm{P}(N_{\mathrm{R}}(u) | z_u) = \sum_{v \in N_s(u)} \log \mathrm{P}(\mathrm{z}_v \mid z_u)$$

Softmax parametrization:

$$\Pr(\mathrm{z}_v | z_u) = \frac{\exp(z_v \cdot \mathrm{z_u})}{\sum_{\mathrm{n} \in V} \exp(z_n \cdot \mathrm{z_u})}$$

Putting it all 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 z_u that minimize 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!!

Solution: Negative Sampling

Instead of normalizing w.r.t. all nodes, just normalize against k random "negative samples"

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

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

sigmoid function

random distribution over all nodes

Negative Sampling

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

random distribution over all nodes

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

Sample negative nodes proportional to degree.

Two considerations for k (# negative samples):

1. Higher k gives more robust estimates.
2. Higher k corresponds to higher prior on negative events.

Random Walks: Stepping Back

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:

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

We can efficiently approximate this using negative sampling!

How should we 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

Goal: Embed nodes with similar network neighborhoods close in the feature space

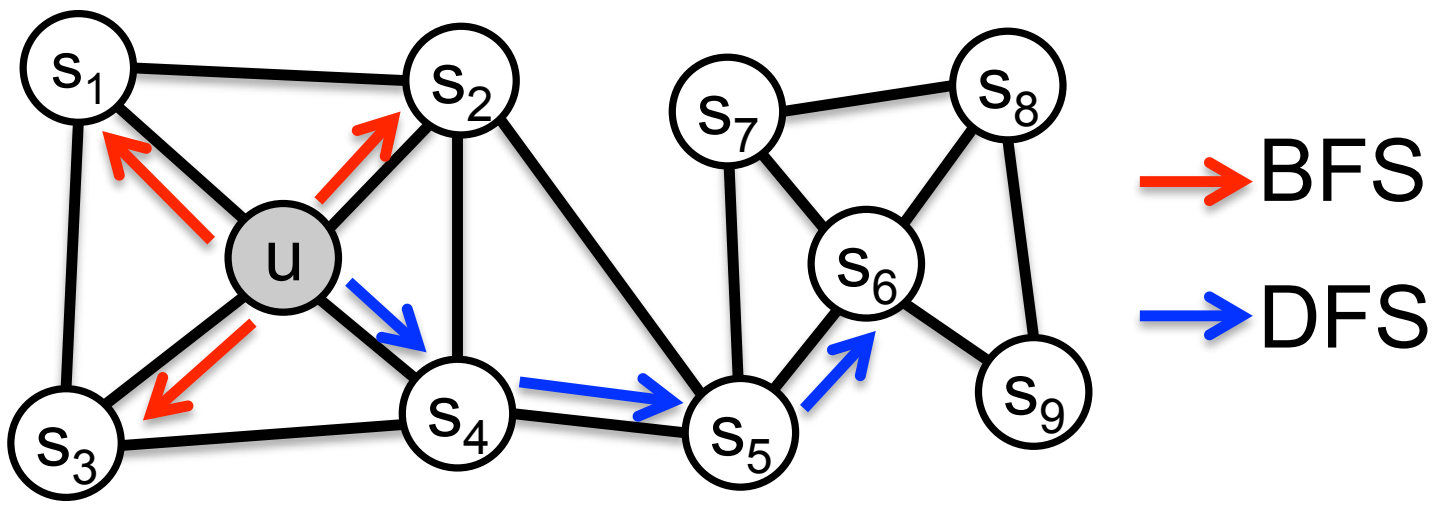
- We frame this goal as prediction-task independent maximum likelihood optimization problem

Key observation: Flexible notion of network neighborhood $N_R(u)$ of node u leads to rich node embeddings

- Develop biased 2nd order random walk R to generate network neighborhood $N_R(u)$ of node u

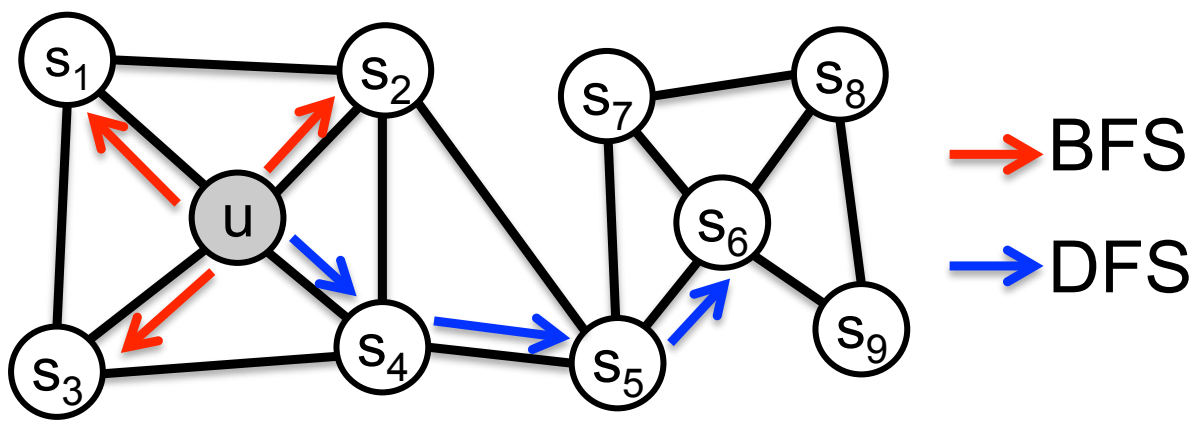
node2vec: Biased Walks

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 $N_R(u)$ of a given node u:



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

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

Interpolating BFS and DFS

Biased random walk R that given a node u generates neighborhood $N_R(u)$

Two parameters:

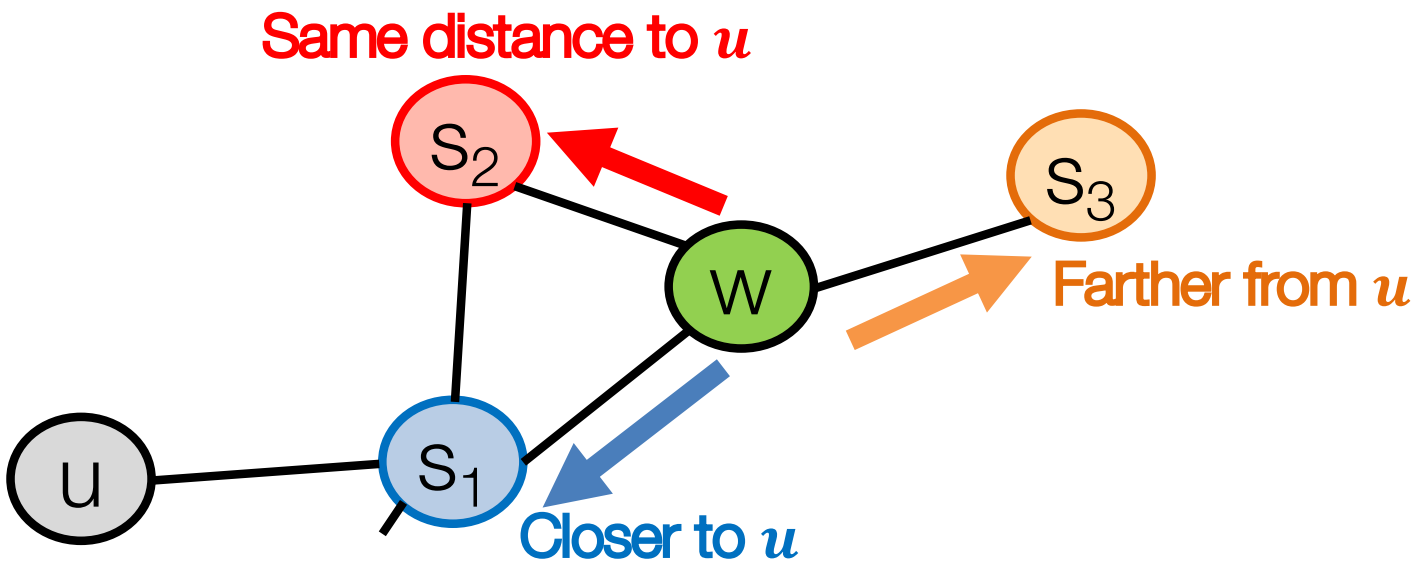
- Return parameter p:
  - Return back to the previous node
- In-out parameter q:
  - Moving outwards (DFS) vs. inwards (BFS)
  - Intuitively, q is the “ratio” of BFS vs. DFS

Biased Random Walks

Biased 2nd-order random walks explore network neighborhoods:

Random walk started at u and is now at w

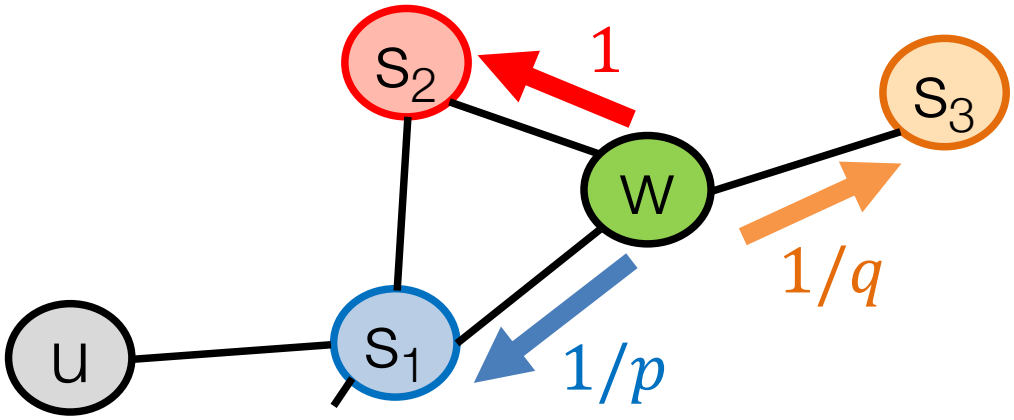
Insight: Neighbors of w can only be:



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

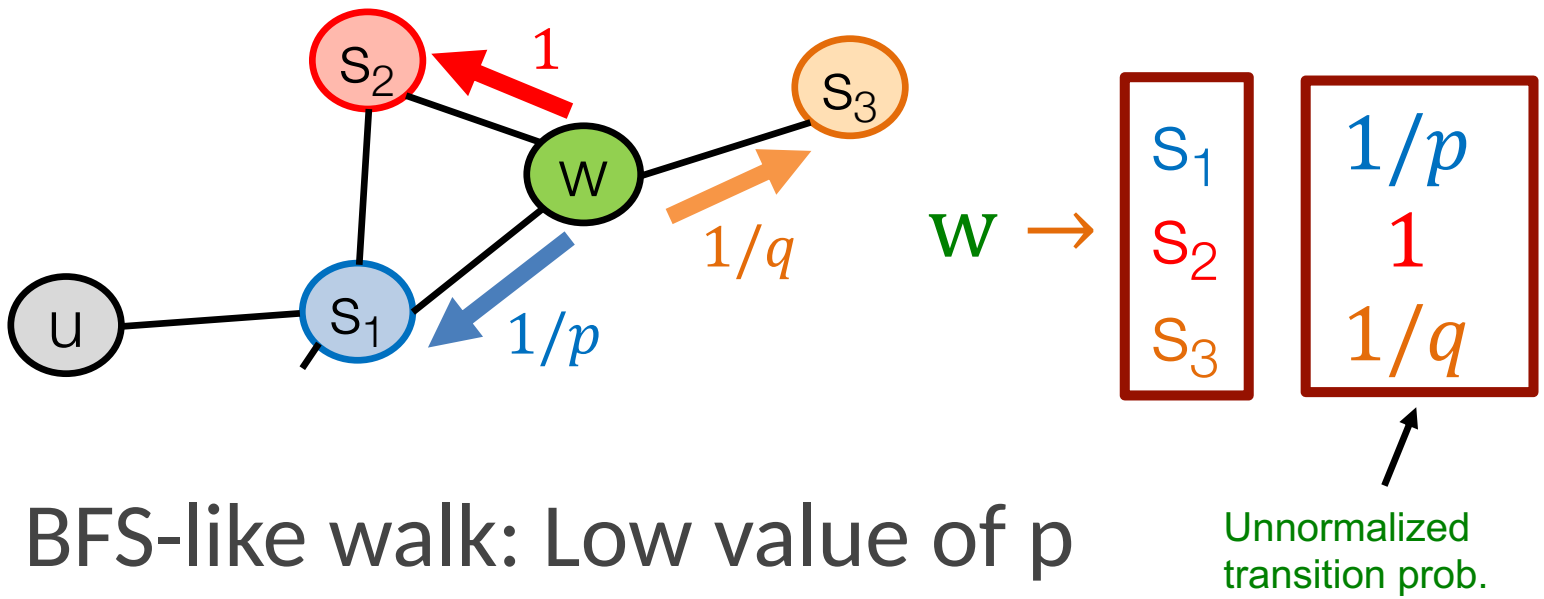
p,q model transition probabilities

p … return parameter

q … ”walk away” parameter

Biased Random Walks

Walker is at w. Where to go next?



BFS-like walk: Low value of p

DFS-like walk: Low value of q

$N_R(u)$ are the nodes visited by the walker

node2vec

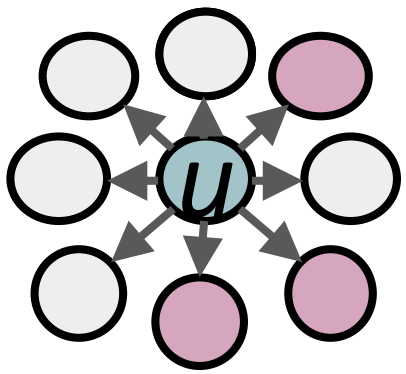
1) Compute random walk probabilities

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

BFS vs. DFS

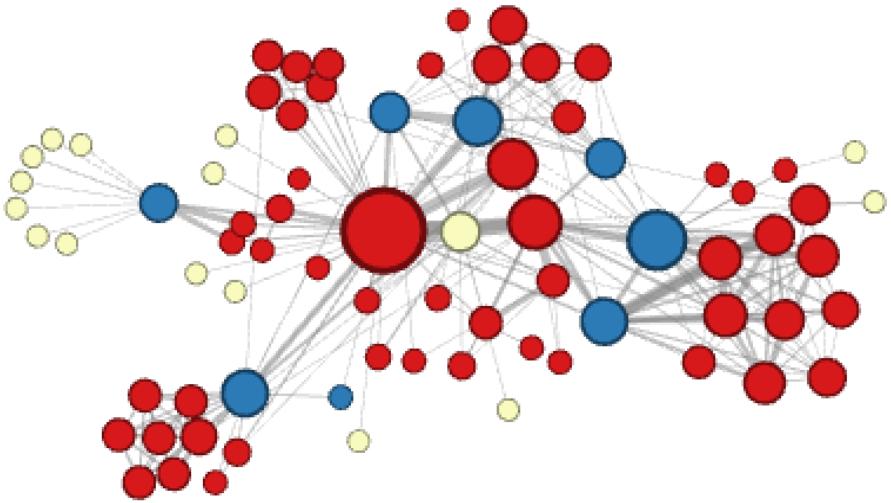


BFS:
Micro-view of neighbourhood

DFS:
Macro-view of neighbourhood

Micro vs. Macro

"structural" equivalence / roles



p=1, q=2

Microscopic view of the network neighbourhood

community structure / homophily

p=1, q=0.5

Macroscopic view of the network neighbourhood

Adjacency-Based Approaches to Node Embeddings

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 Z∈R^(d x |V|)that minimizes the loss L

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 or QR decomposition routines).

Only works in limited cases.

Adjacency-based similarity

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

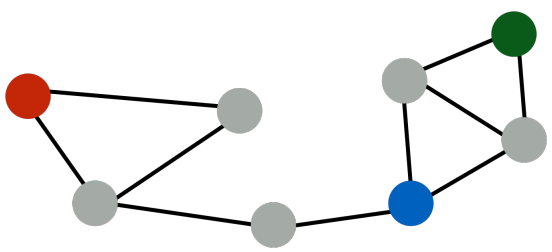
Drawbacks:

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.



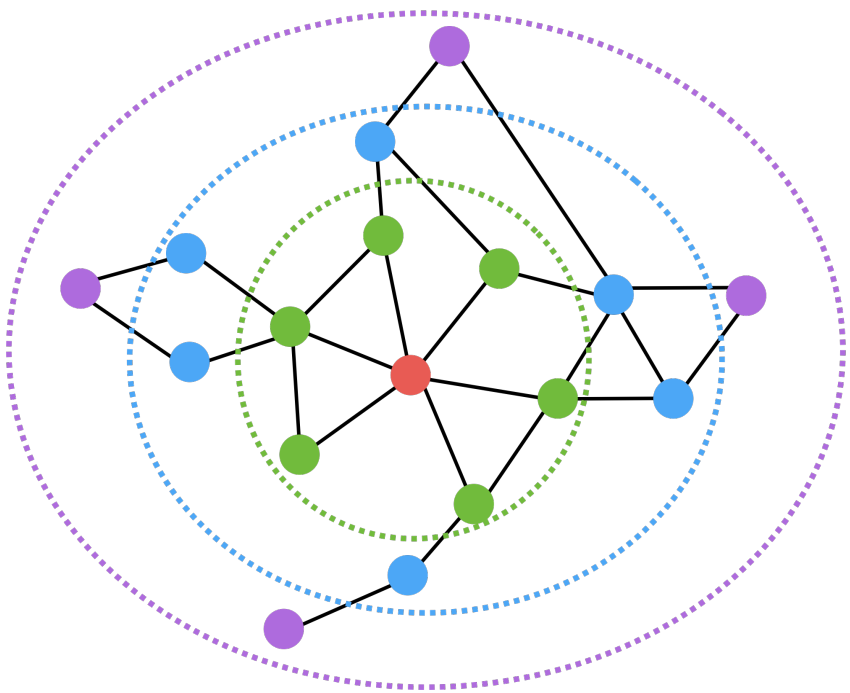
e.g., the blue node is obviously more similar to green compared to red node, despite none having direct connections.

Multi-hop similarity

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:

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

Train embeddings to predict k-hop neighbors.

In practice (GraRep from Cao et al, 2015):

Use log-transformed, probabilistic adjacency matrix:

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

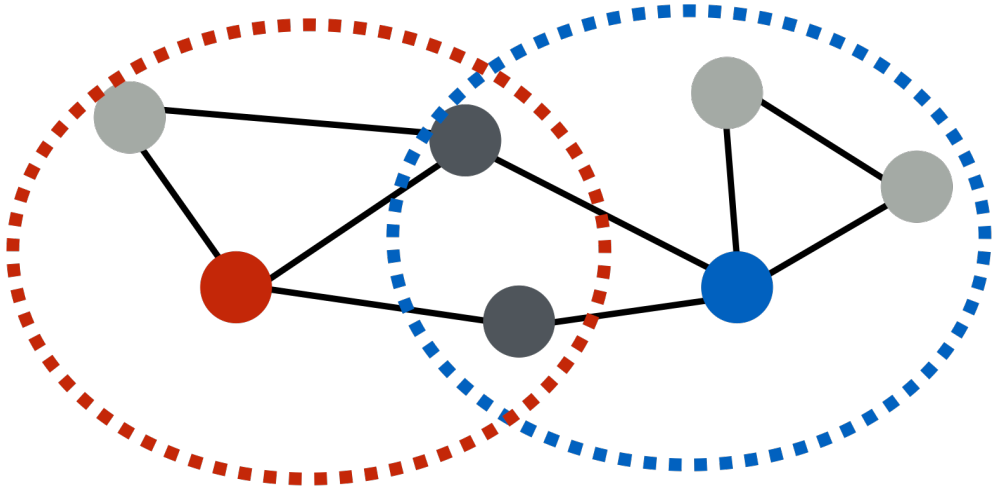
node degree

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

Adamic-Adar score

Multi-hop similarity

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

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 or Adamic-Adar score).

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

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).