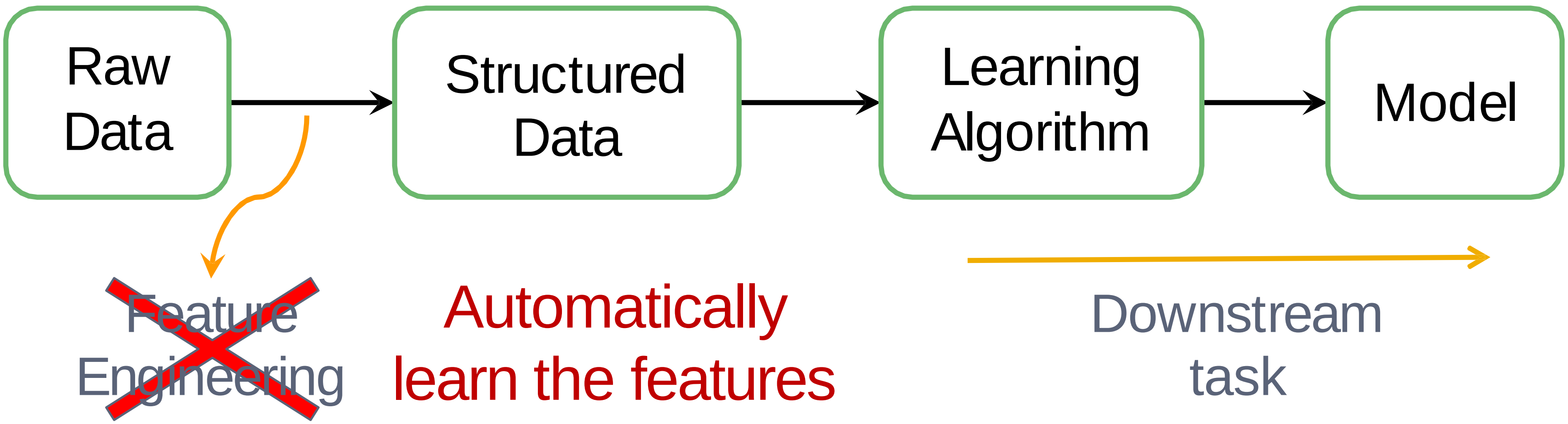
Graph Representation Learning

Deeksha Chandola

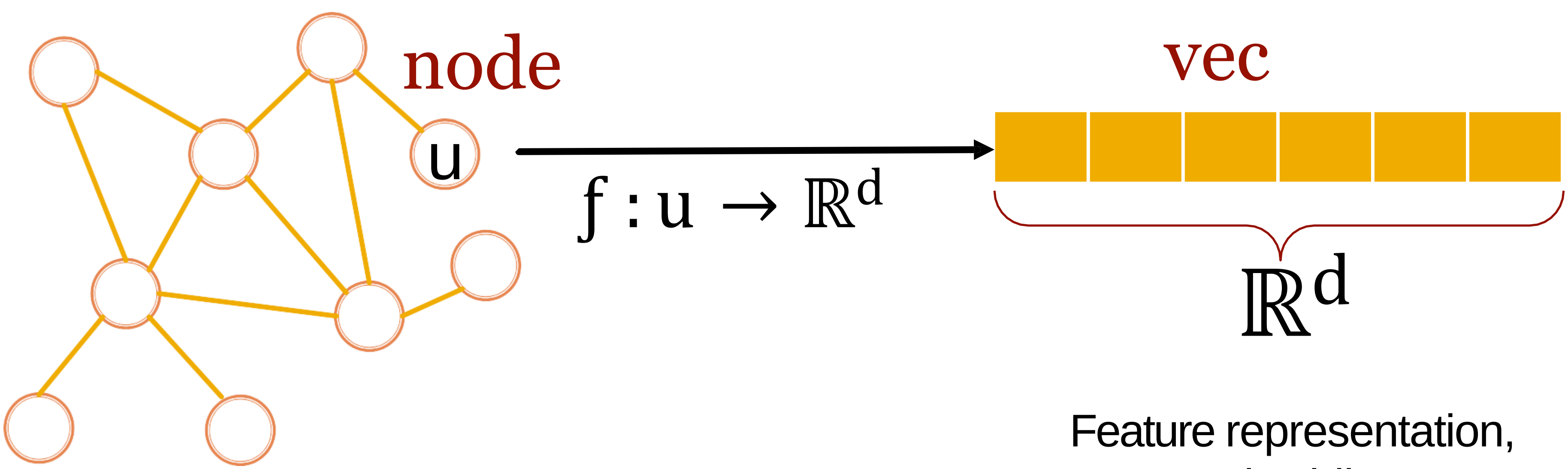
Machine Learning Lifecycle

- **(Supervised) Machine Learning Lifecycle requires feature engineering every single time!**



Feature Learning in Graphs

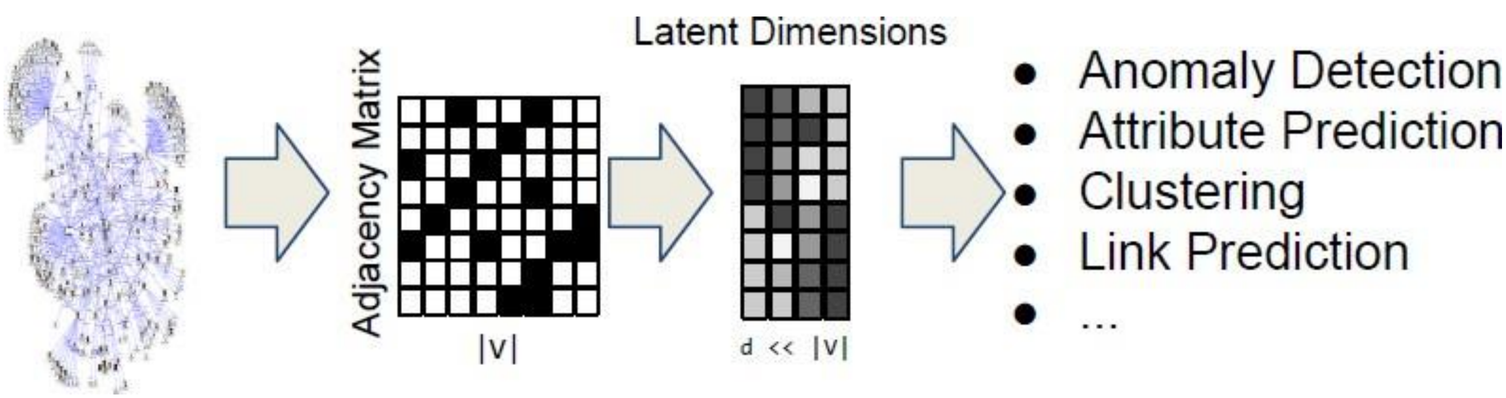
Goal: Efficient task-independent feature learning for machine learning in networks!



Feature representation, embedding

Why network embedding?

- **Task: We map each node in a network into a low-dimensional space**
  - Distributed representation for nodes
  - Similarity of embedding between nodes indicates their network similarity
  - Encode network information and generate node representation



Jure Leskovec, Stanford CS224W: Analysis of Networks, http://cs224w.stanford.edu

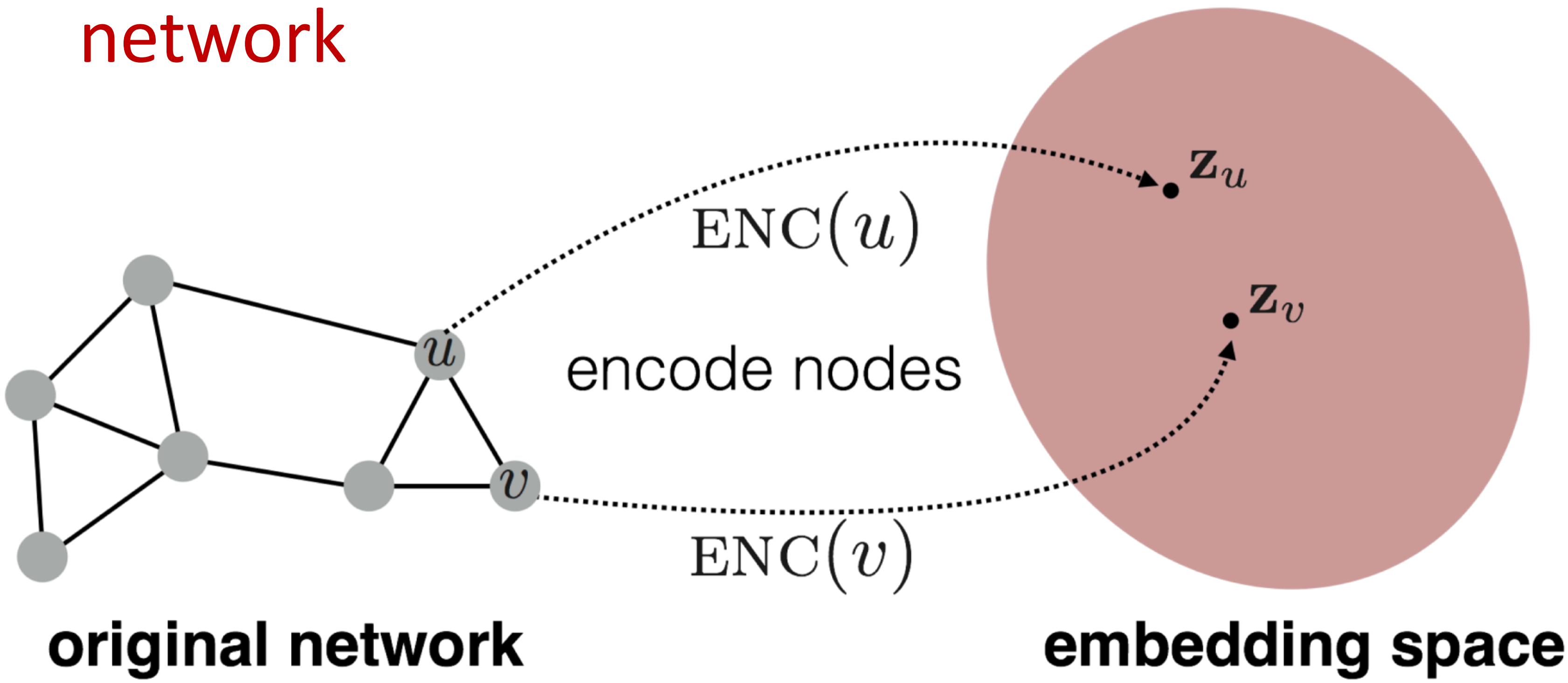
Node Embeddings

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



Two Key Components

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

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

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

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:**

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

in the original network

Similarity of the embedding

"Shallow" Encoding

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

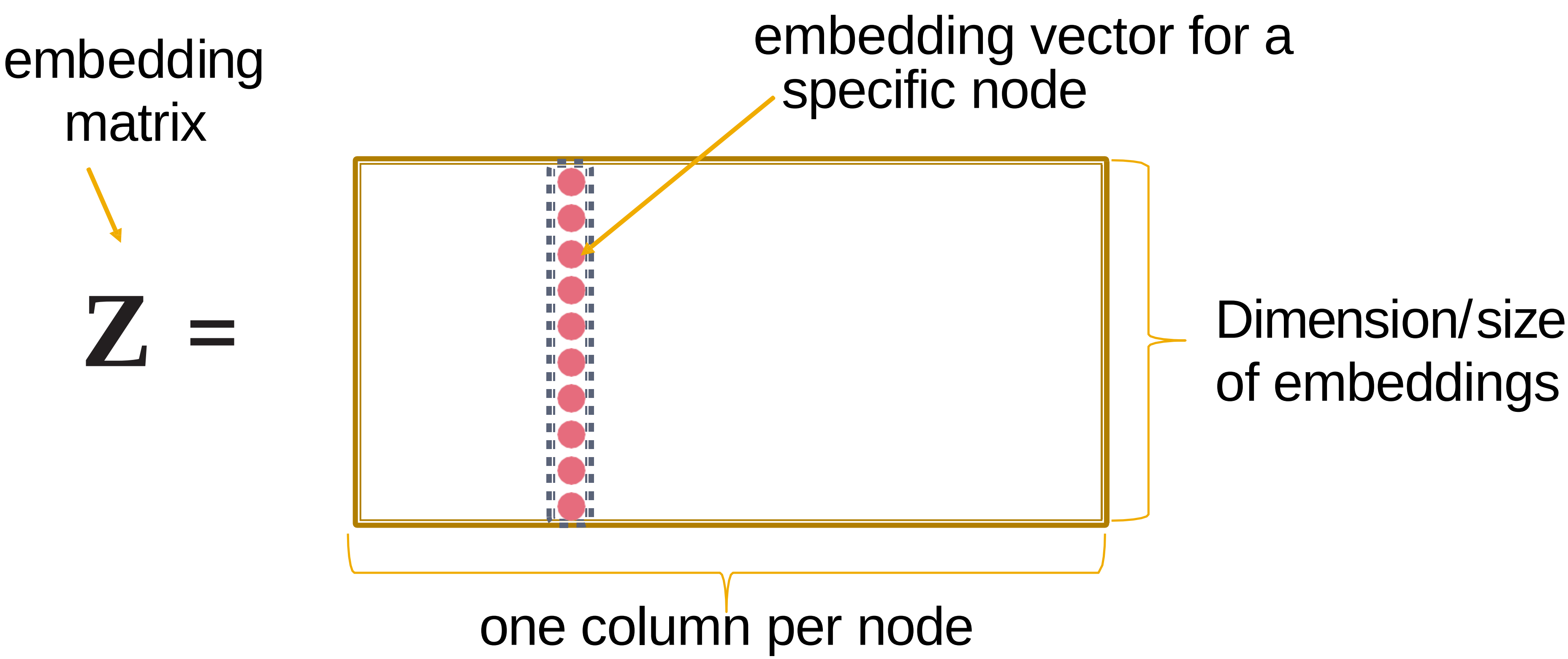
$$\mathrm{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ 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**
- **Each node is assigned a unique embedding vector**



How to Define Node Similarity?

- Key choice of 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"?
  - …?

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)

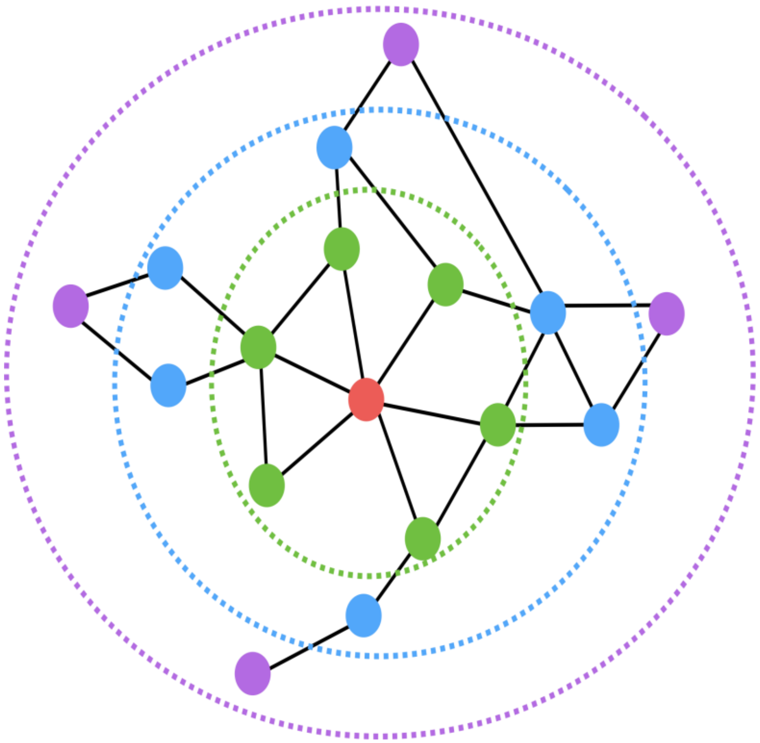
sum over all node pairs

embedding similarity

(weighted) adjacency matrix for the graph

Multi-hop Similarity

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



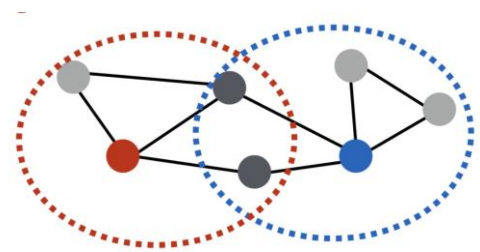
- **Red:** Target node
- **Green**: 1-hop neighbors
  - **A** (i.e., adjacency matrix)
- **Blue**: 2-hop neighbors
  - $\mathbf{A}^2$
- **Purple**: 3-hop neighbors
  - $\mathbf{A}^3$

Multi-hop Similarity

Train embeddings for different adjacency matrices.

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

… concatenate.



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

embedding similarity

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

- **Jacard**
- **Adamic-Adar**
- **…**

Discussion 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. Only considers direct, local connections.

Random Walk Approaches:

Expressivity: incorporates both local and higher-order neighborhood information

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

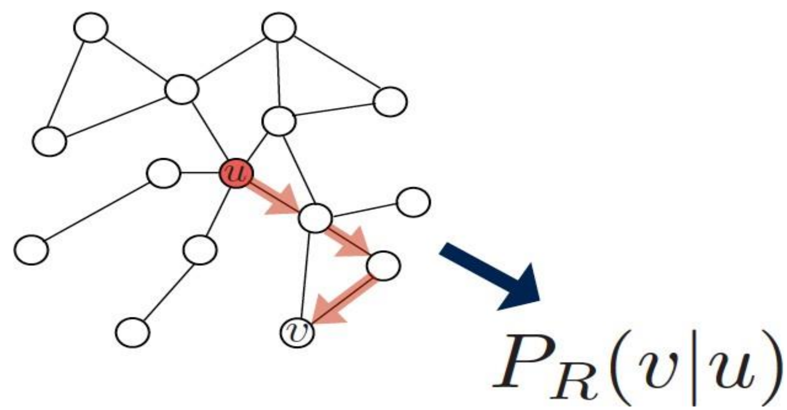
Random-walk Embeddings

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

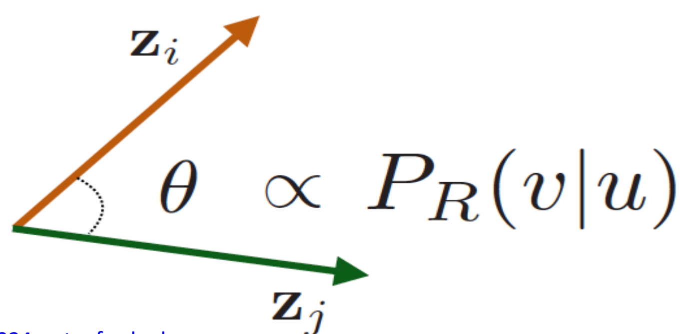
probability that $\boldsymbol{u}$ and $\boldsymbol{v}$ co-occur on a random walk over the network

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.**



How should we randomly walk?

- **What strategies should we use to run these random walks?**

  - DeepWalk [1]
  - Node2Vec [2]

Reference:
[1] Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations. *KDD.*
[2] Grover et al. 2016. node2vec: Scalable Feature Learning for Networks. *KDD.*

Unsupervised Feature Learning

Intuition: Find embedding of nodes to d-dimensions that preserves similarity

- 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

DEEP WALK

1. Run **short fixed-length 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: Given node u, predict its neighbors $\mathrm{n}_\mathrm{R}(\mathrm{u})$

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

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

Random Walk Optimization

Softmax Function $$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

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

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

Optimizing random walk embeddings = Finding embeddings z_u that minimize L

Overview of node2vec

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

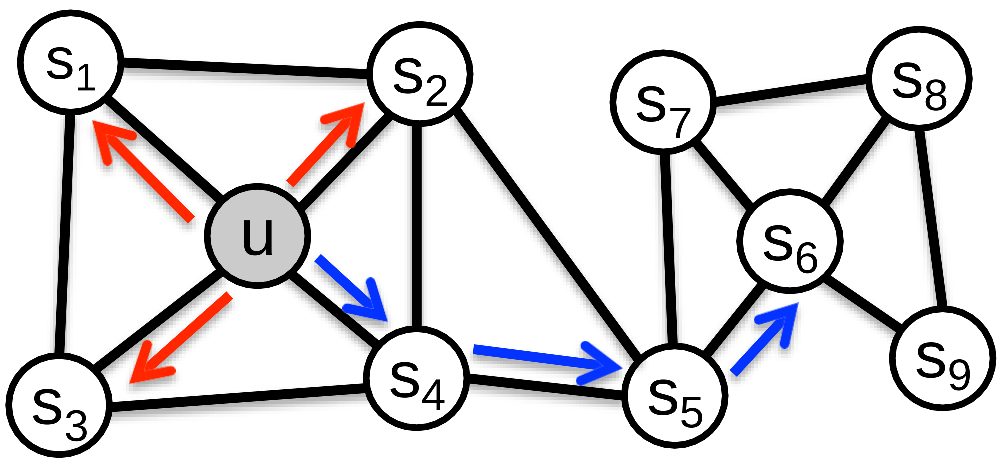
- We frame this goal as prediction-task independent maximum likelihood optimization problem

- **Key observation:** Flexible notion of network neighborhood $\mathrm{N_R(u)}$ of node u leads to rich node embeddings

- Develop biased 2nd order random walk $\&$ to generate network neighborhood $\mathrm{N_R(u)}$ of node u

node2vec: Biased Walks

Two classic strategies to define a neighborhood $N_R(u)$ of a given node u:



Local microscopic view

Global macroscopic view

BFS

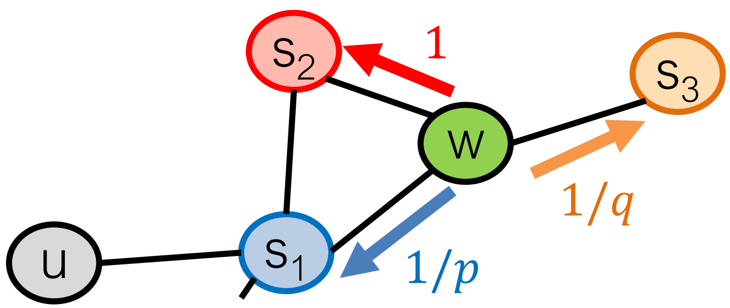
DFS

Interpolating BFS and DFS

1. Return parameter p: Return back to the previous node
2. In-out parameter q: Moving outwards (DFS) vs. inwards(BFS)

Biased Random Walks

- Walker is at w.
- Where to go next?

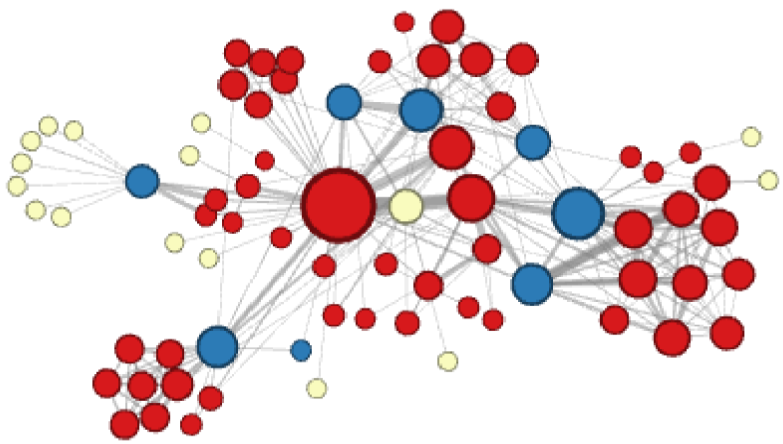


$1/p$, $1/q$, 1 are unnormalized probabilities

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

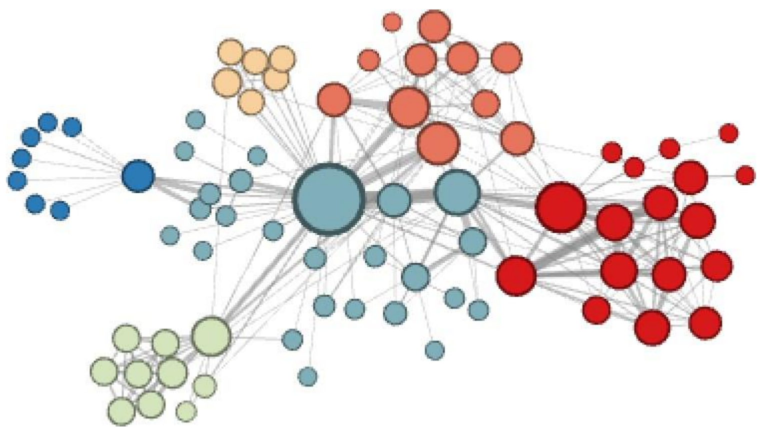
Experiments: Micro vs. Macro

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

Summary

- Feature learning in networks as a search based optimization problem
- **DeepWalk** proposes search using uniform random walks
- It gives us no control over the explored neighborhoods
- **node2vec** search strategy is both flexible and controllable exploring network neighborhoods through parameters p and q
- **node2vec** is scalable and robust to perturbations.
- ***node2vec*** can learn representations that organize nodes based on their **network roles** (structural equivalence) and **communities** (homophily) they belong to.

How to Use Embeddings

- **How to use embeddings z_i of nodes:**
  - **Clustering/community detection:** Cluster points z_i
  - **Node classification:** Predict label $f(z_i)$ of node i based on z_i
  - **Link prediction:** Predict edge $((i, j)$ based on $f(z_i, z_j)$
    - Vector operators: concatenate, avg, product, or take a difference between the embeddings:
      - Concatenate: $f(z_i, z_j) = g([z_i, z_j])$
      - Hadamard: $f(z_i, z_j) = g(z_i * z_j)$ (per coordinate product)
      - Sum/Avg: $f(z_i, z_j) = g(z_i + z_j)$
      - Distance: $f(z_i, z_j) = g(||z_i - z_j||_2)$