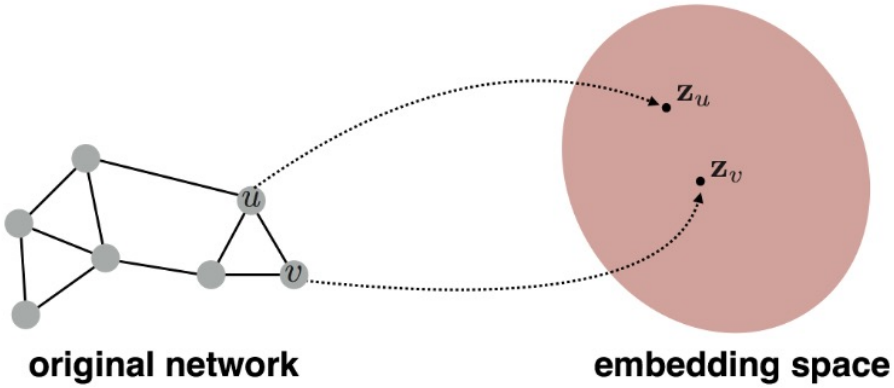
# Graph Representation Learning

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Amazon

http://cs246.stanford.edu

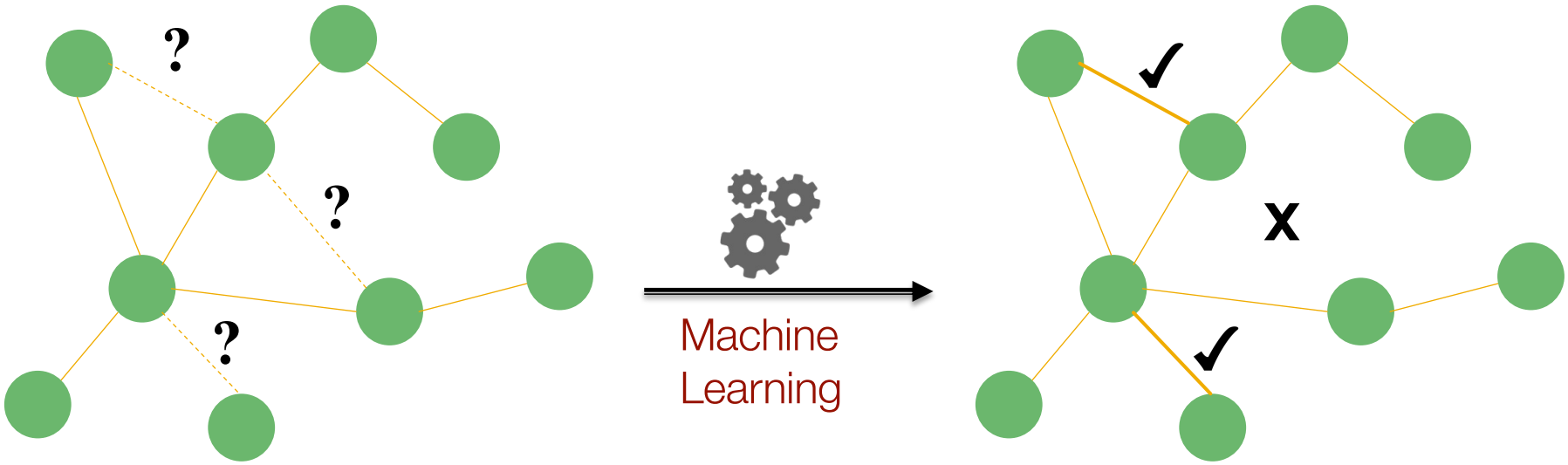
# Representation Learning in Graphs

It is to learn a mapping that embeds nodes, subgraphs, or the entire graphs, as points in a low-dimensional vector space.




Such that geometric relationships in the learned space reflect the structure of the original graph.

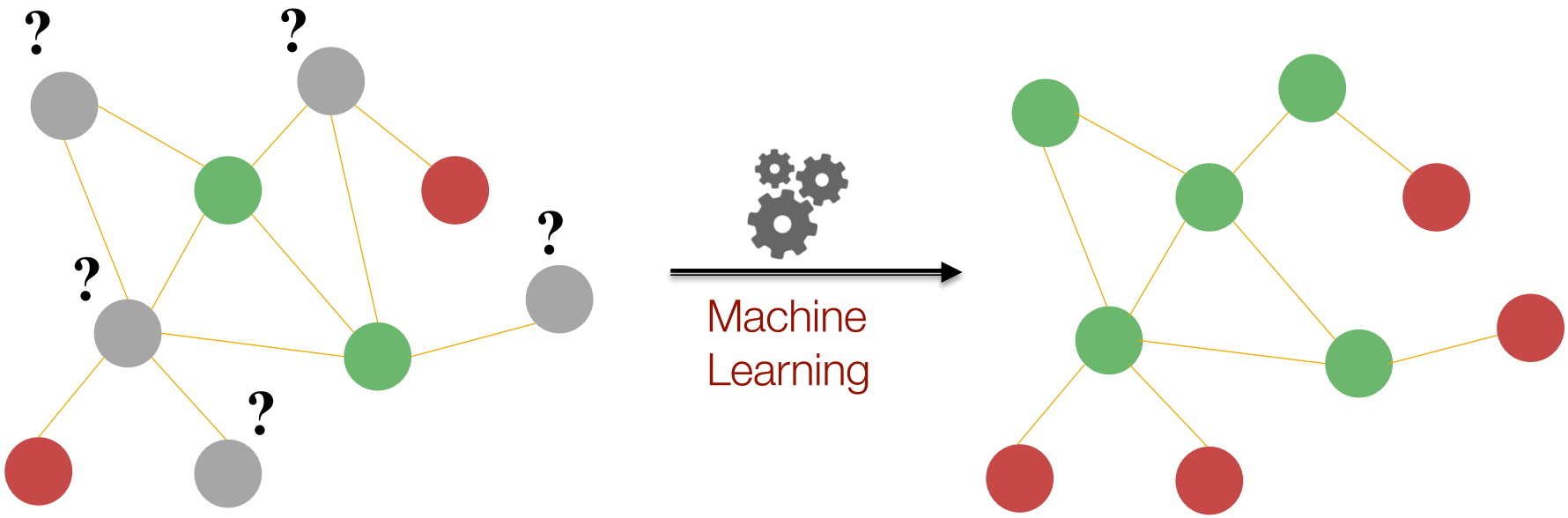
# Example: Link Prediction




**Examples of link prediction task:**

- Identifying real-world friends in social network
- Discovering novel interactions between genes in genomics
- Recommender systems

# Example: Node Classification



# Example: Node Classification

Classifying the function of proteins in the interactome

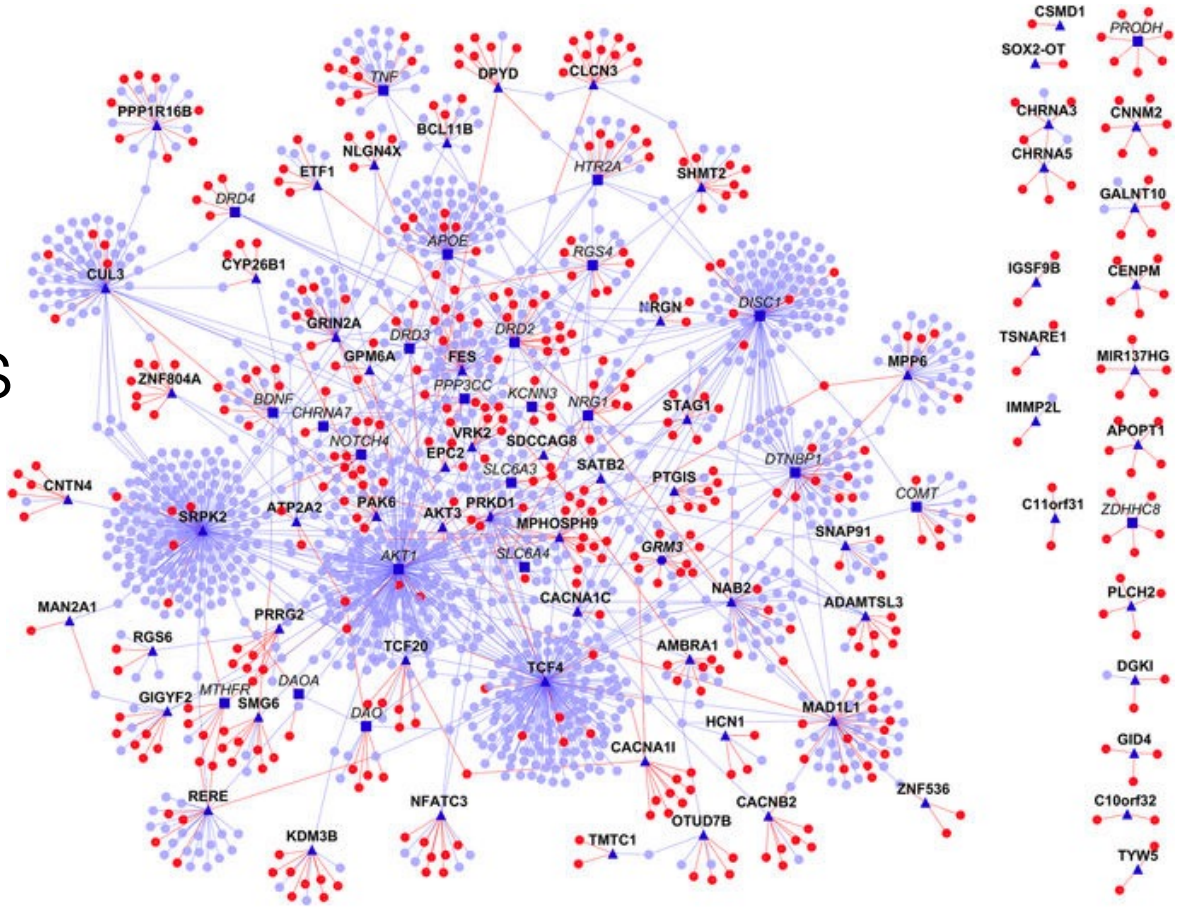
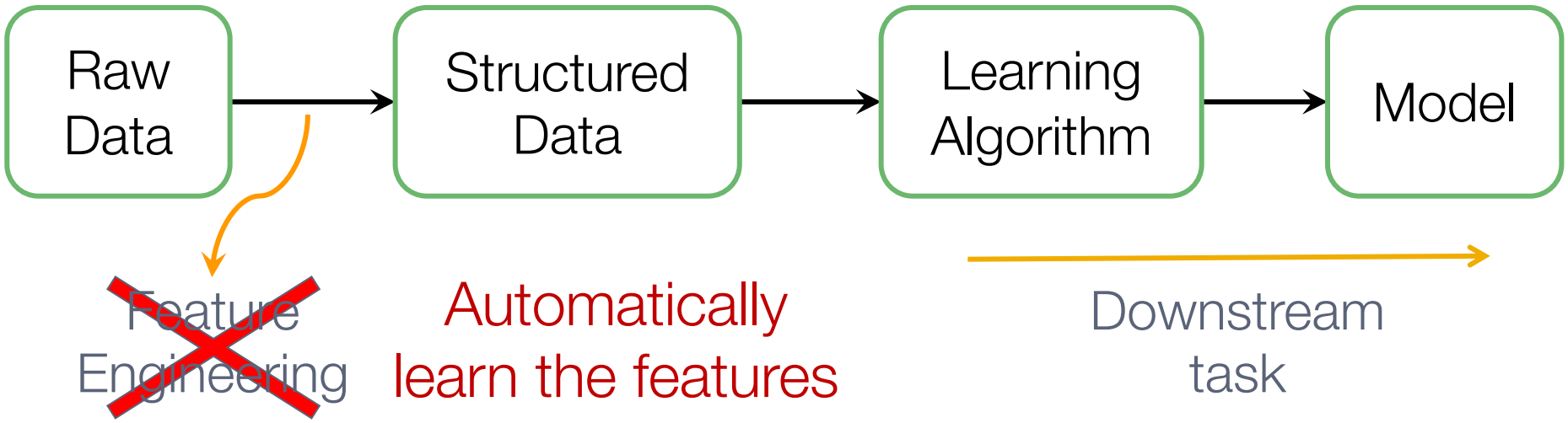



Image from: Ganapathiraju et al. 2016. Schizophrenia interactome with 504 novel protein–protein interactions. *Nature*.

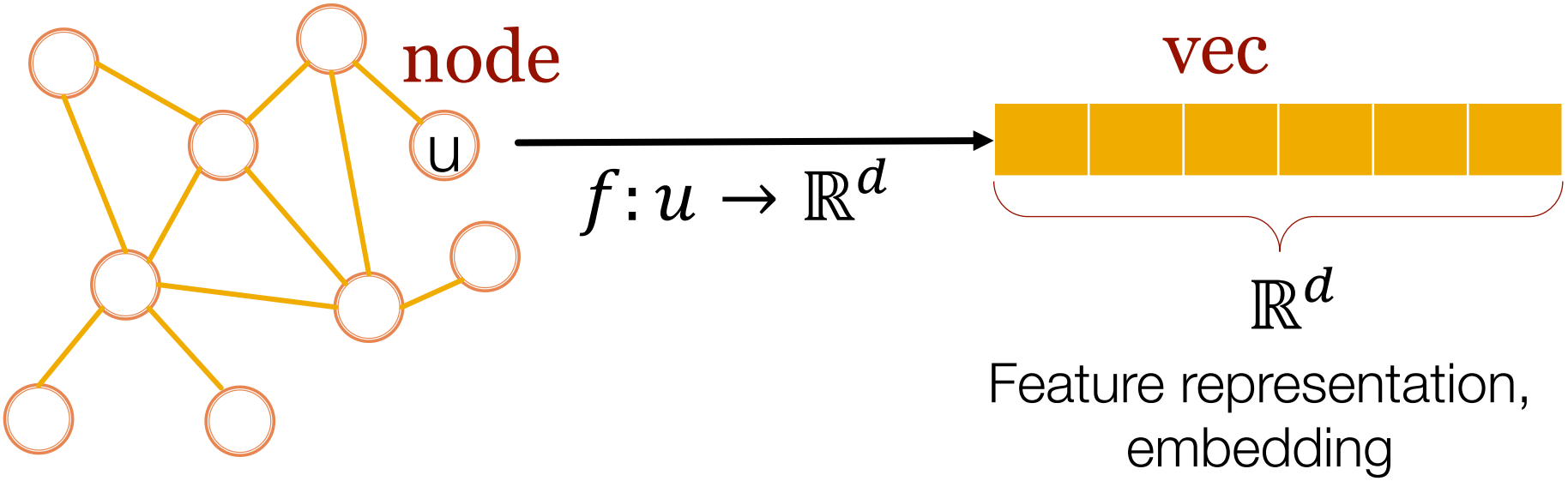
# Machine Learning Lifecycle

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



# Feature Learning in Graphs

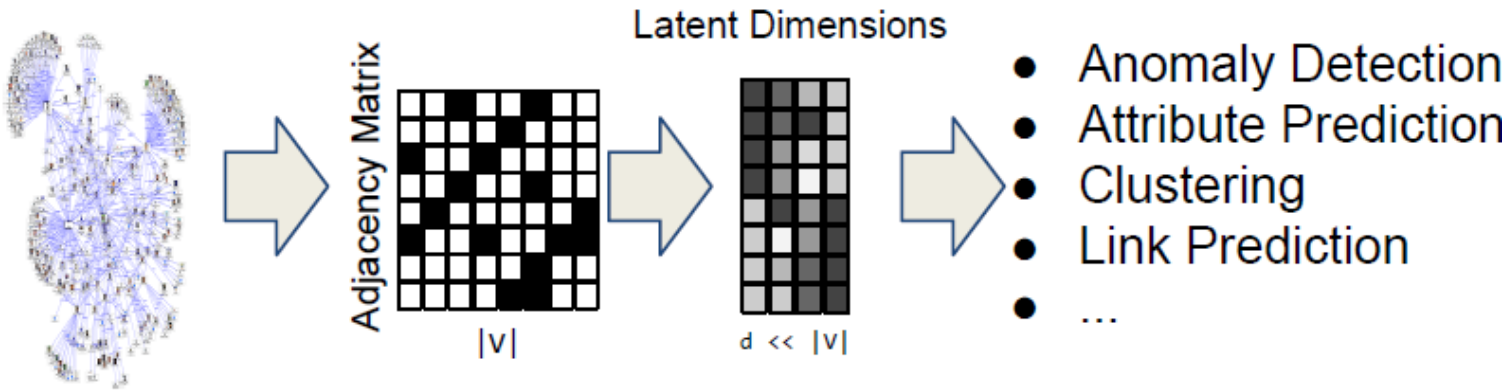
Goal: Efficient task-independent feature learning for machine learning in networks!



# Why 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



# Example Node Embedding

**2D embedding of nodes of the Zachary's Karate Club network:**

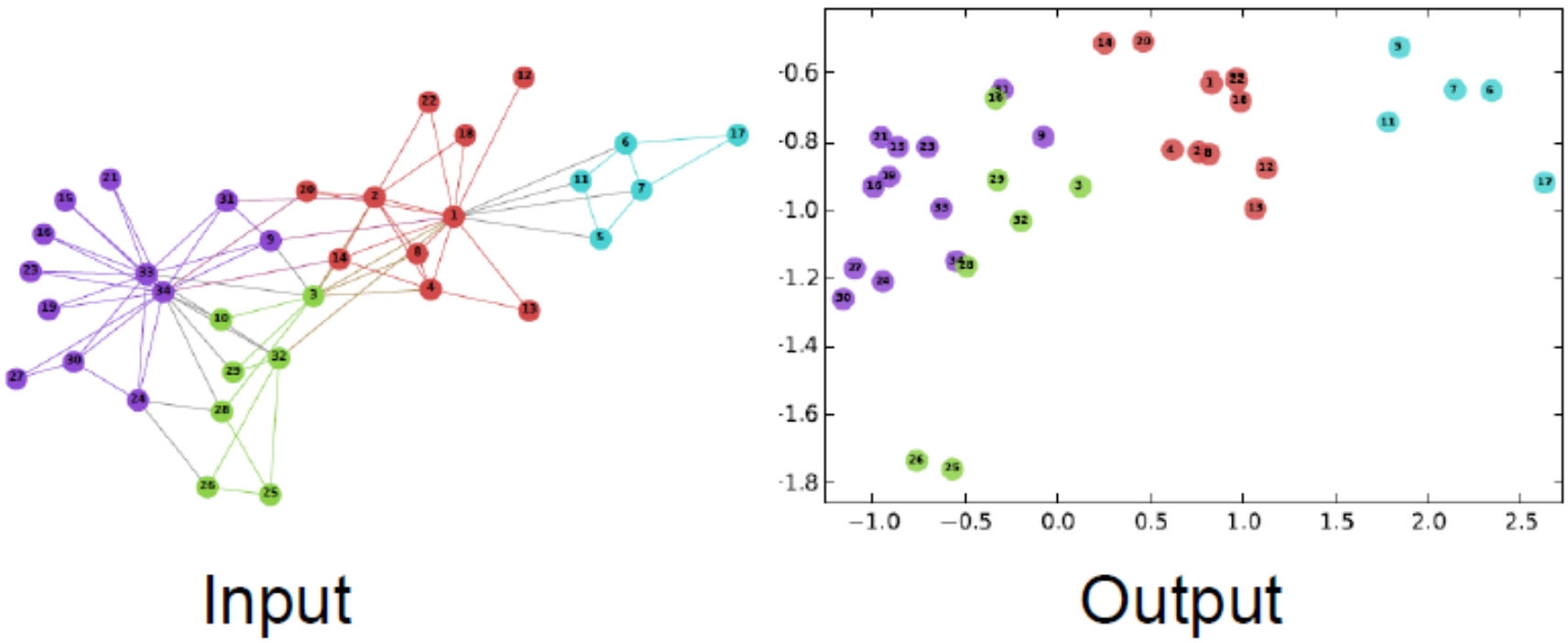



Image from: Perozzi et al. DeepWalk: Online Learning of Social Representations. *KDD 2014.*

# Embedding Nodes

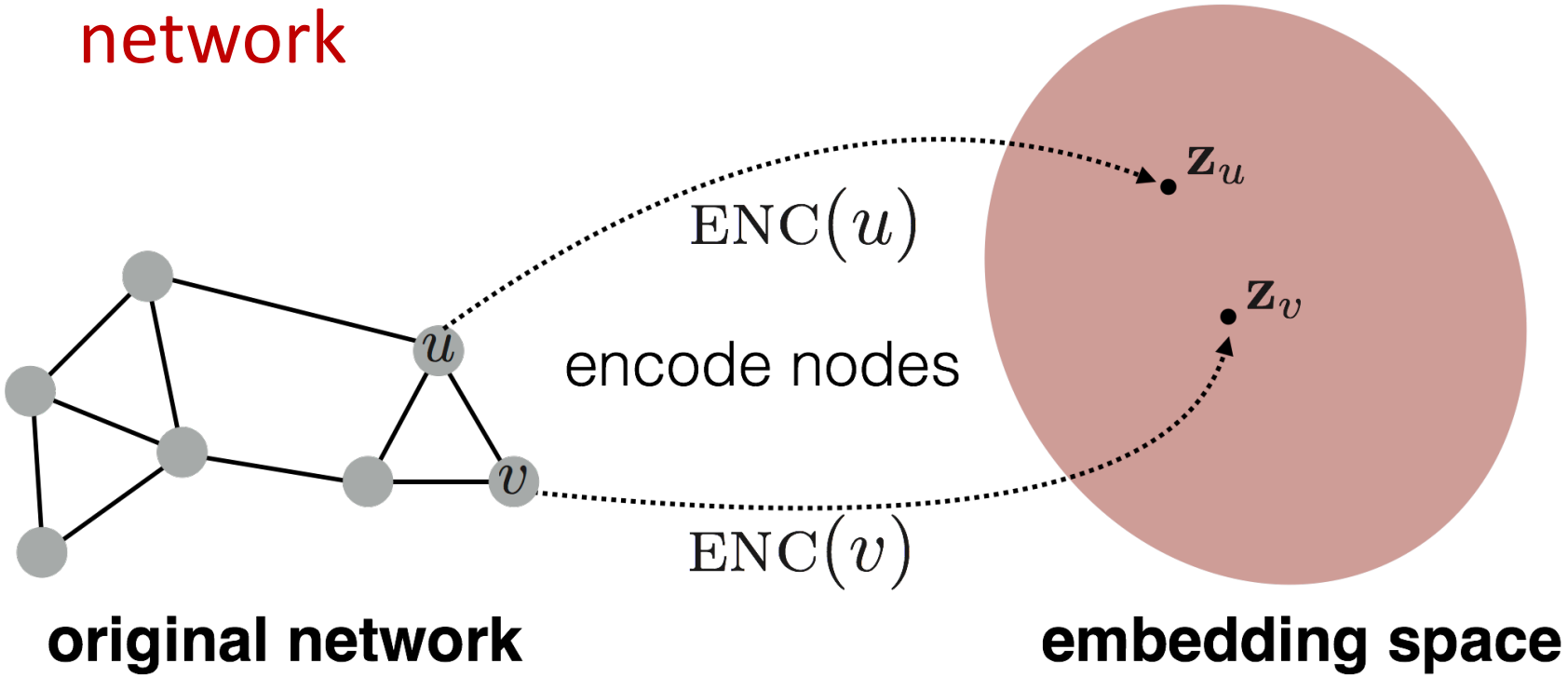
# Setup

**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 (e.g., dot product) approximates similarity in the original network



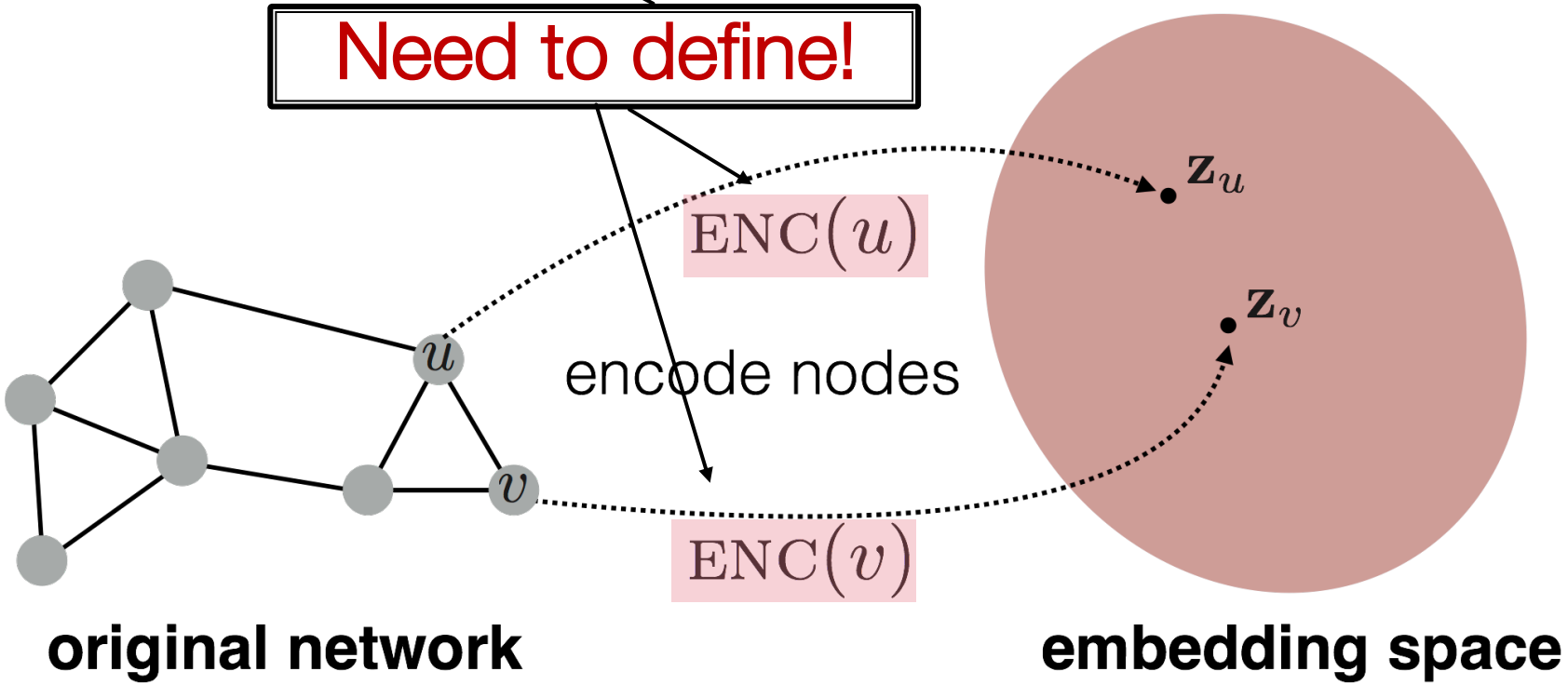
# Embedding Nodes

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

in the original network

Similarity of the embedding

Need to define!



# 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:**

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

# Two Key Components

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

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

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**

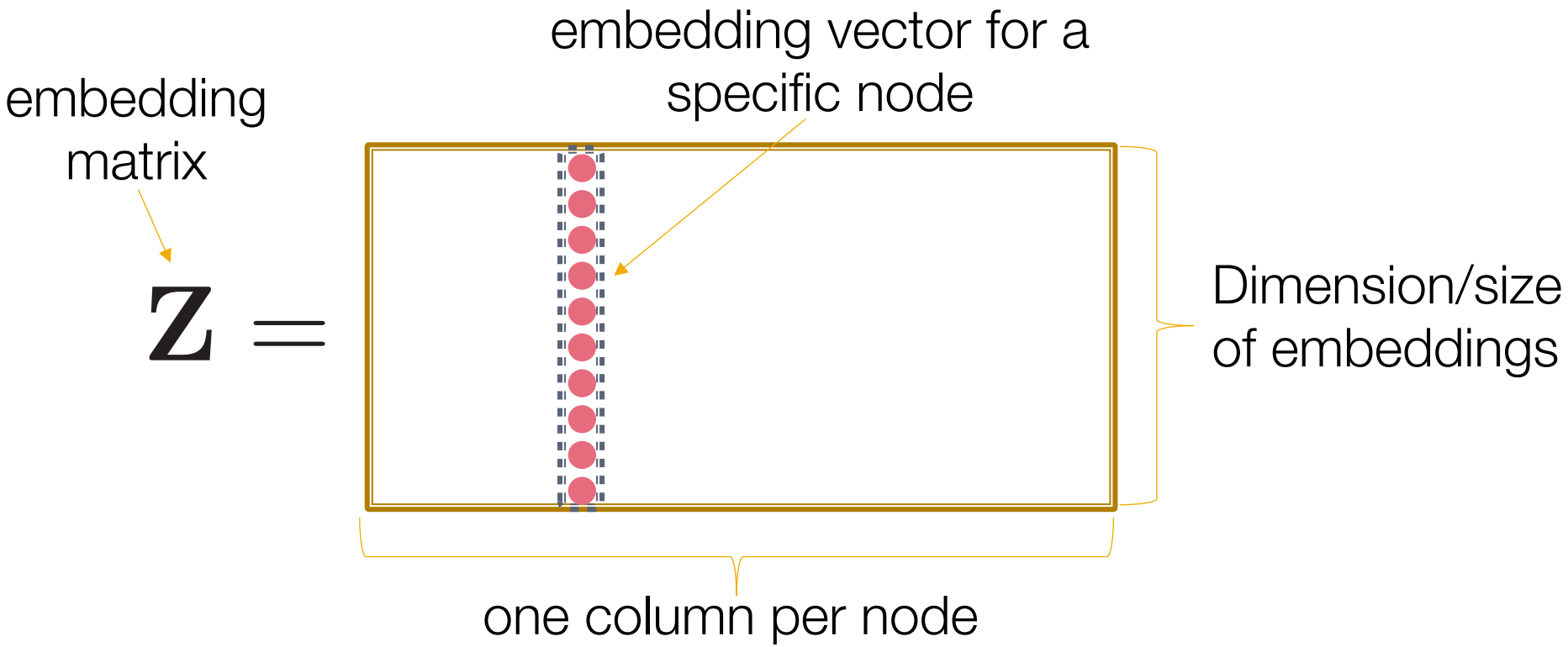
$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

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

$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ Indicator vector, all zeroes except for a "1" at the position that corresponds to 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**

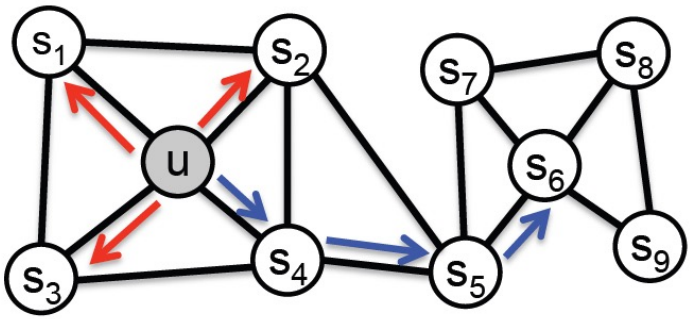
Many methods: node2vec, DeepWalk, LINE

# How to Define Node Similarity?

Key choice of methods is **how they define node similarity.**

E.g., should two nodes have similar embeddings if they…

- are connected?
- share neighbors?
- similar "structural roles"?
- …?



# Random Walk Approaches to Node Embeddings

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

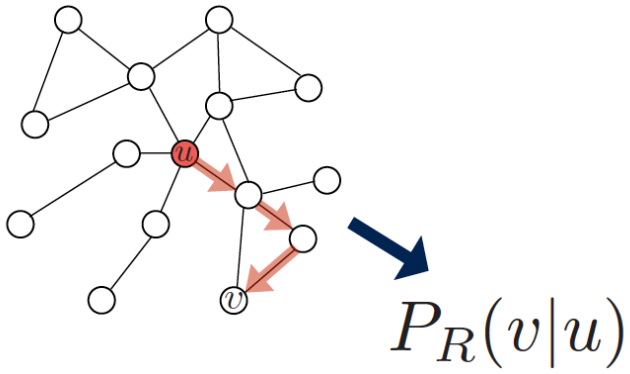
$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$

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

$z_u$ … embedding of node $u$

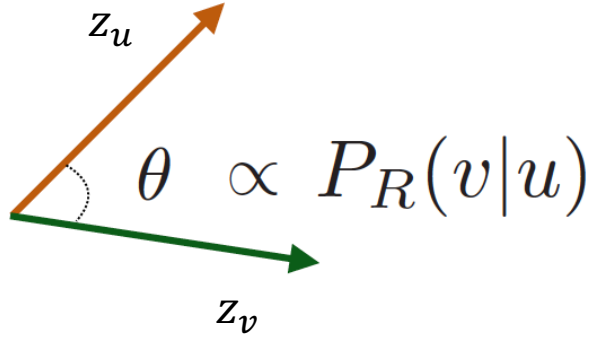
# Random-walk Embeddings

The probability of visiting node $v$ on a random walk starting from node $u$ using some random walk strategy $R$




$$\mathbf{z}_u^\top \mathbf{z}_v \approx \boldsymbol{P_R(v|u)}$$

Optimize the embeddings to capture node 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

$$\mathbf{z}_u^\top \mathbf{z}_v \approx \boldsymbol{P_R(v|u)}$$

- For every $u$, what $v$ to consider?
- **Idea:** Consider nearby nodes
- For a node $u$, consider all $v$ in neighborhood of u
  - $N_R(u)$ … neighbourhood of $u$ obtained by some strategy $R$

# Feature Learning as Optimization

- Given $G = (V, E)$. Our goal is to learn a mapping $z: u \rightarrow \mathbb{R}^d$
- Given node $u$, we want to learn feature representations predictive of nodes in its neighborhood $N_{\mathrm{R}}(u)$
- Maximum likelihood optimization problem:
  - Maximize Log-probability of observing neighborhood:

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

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

# Random Walk Optimization

1. Run **short fixed-length 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 according to: Given node $u$, predict its neighbors $N_R(u)$

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

*$N_R(u)$ can have repeat elements since nodes can be visited multiple times on random walks

# Random Walk Optimization

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

- **Conditional independence**
  - Likelihood of observing a neighborhood node is independent of observing any other neighborhood node

$$\mathrm{P}(N_{\mathrm{R}}(u)|z_u) = \prod_{v \in N_R(u)} \mathrm{P}(\mathrm{z}_v \mid z_u)$$

- **Symmetry in feature space**
  - **Softmax parametrization:**

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

**Why softmax?**
We want node $v$ to be most similar to node $u$ (out of all nodes $n$).
**Intuition:** $\sum_i \exp(x_i) \approx \max_i \exp(x_i)$

# Random Walk Optimization

**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 $v$ appearing in random walk starting from $u$

**Optimizing random walk embeddings =**

**Finding node embeddings $z$ 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 is the culprit… can we approximate it?**

# Negative Sampling

- **Solution:** Negative sampling

**Why is the approximation valid?**
Technically, this is a different objective. But Negative Sampling is a form of Noise Contrastive Estimation (NCE) which approx. maximizes the log probability of softmax.

New formulation corresponds to using a logistic regression (sigmoid func.) to distinguish the target node $v$ from nodes $n_i$ sampled from background distribution $P_v$.

More at https://arxiv.org/pdf/1402.3722.pdf

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

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

sigmoid function
(makes each term a "probability" between 0 and 1)

random distribution over all nodes

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

# 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 $k$ negative nodes proportional to degree
- Two considerations for $k$ (# negative samples):
  1. Higher $k$ gives more robust estimates
  2. Higher $k$ corresponds to higher computation cost

  In practice $k$ =5-20

# Random Walks: Stepping Back

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 using Stochastic Gradient Descent:

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

We can efficiently approximate this using negative sampling!

# How should we randomly walk?

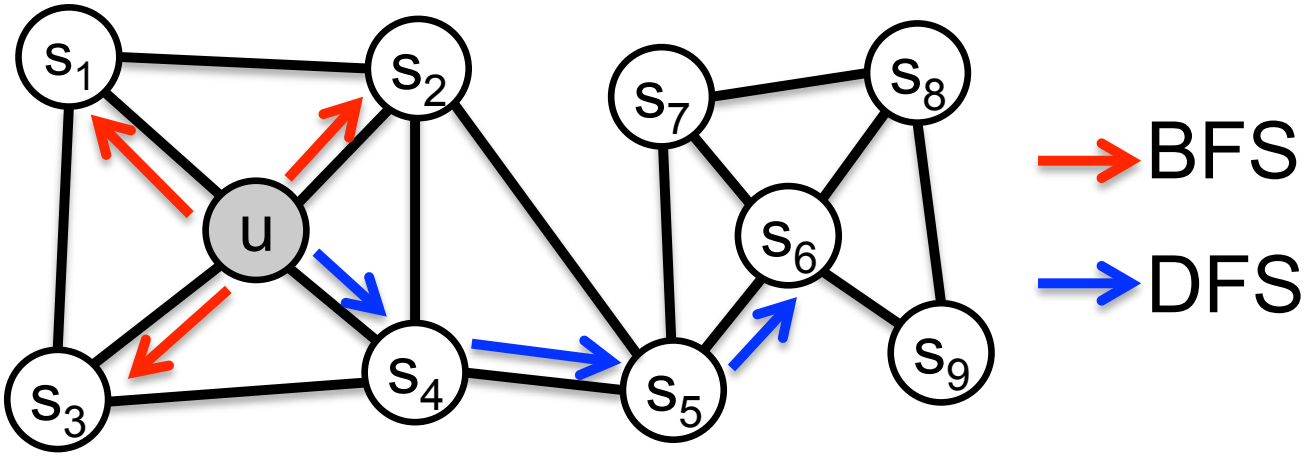
- So far we have described how to optimize embeddings given random walk statistics
- **What strategies should we use to run these random walks?**
  - Simplest idea: **Just run fixed-length, unbiased random walks starting from each node** (i.e., DeepWalk from Perozzi et al., 2013).
    - The issue is that such notion of similarity is too constrained
  - How can we generalize this?

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

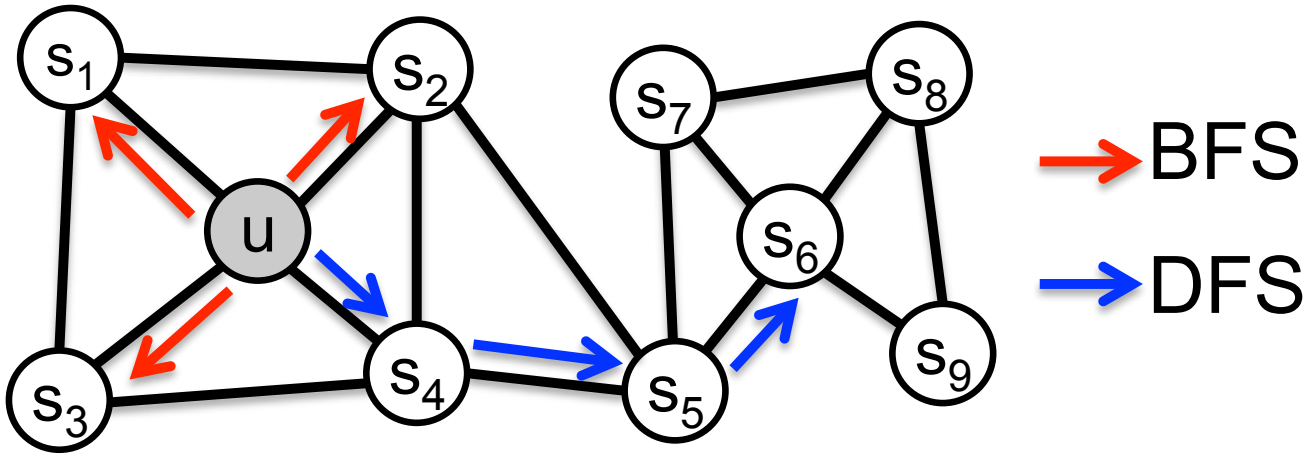
# node2vec: Biased Walks

**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 $u$:




**Walk of length 3** ($N_R(u)$ 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 macroscopic view

# Interpolating BFS and DFS

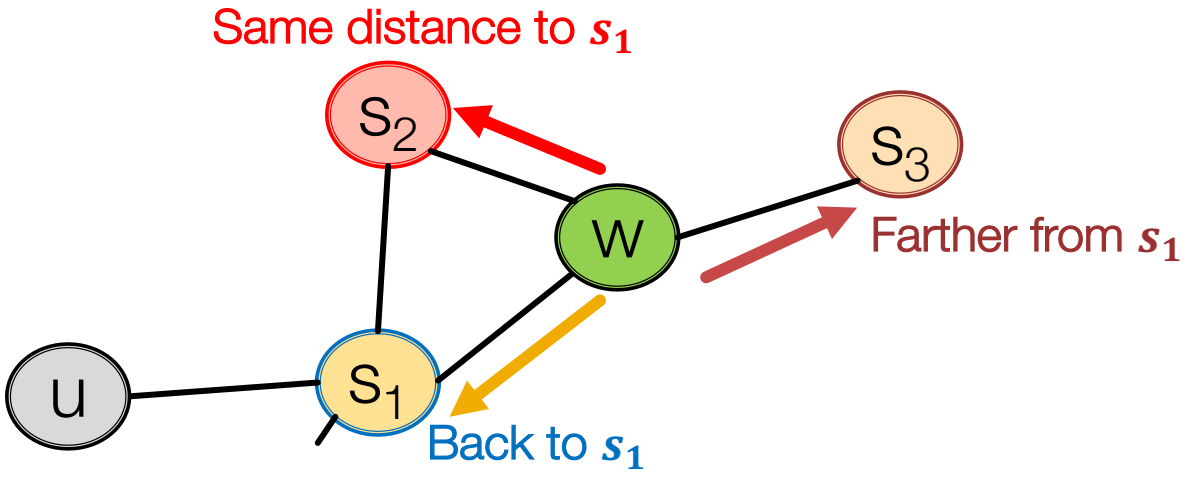
**Biased fixed-length random walk $R$ that given a node $u$ generates neighborhood $N_R(u)$**

- Two parameters:
  - **Return parameter $p$:**
    - Return back to the previous node
  - **In-out parameter $q$:**
    - Moving outwards (DFS) vs. spreading (BFS)
    - Intuitively, $q$ is the "ratio" of BFS vs. DFS

# Biased Random Walks

Biased 2nd-order random walks explore network neighborhoods:

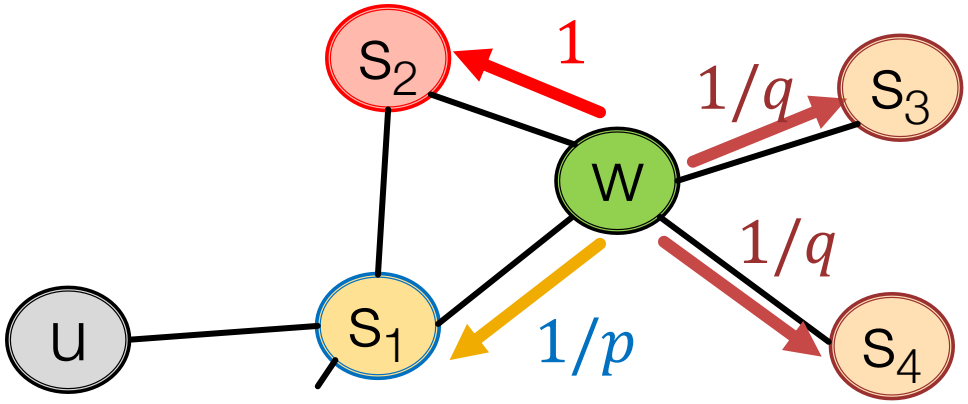
- Rnd. walk just traversed edge $(s_1, w)$ and is now at $w$
- **Insight:** Neighbors of $w$ can only be:




**Idea:** Remember where that walk came from

# Biased Random Walks

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

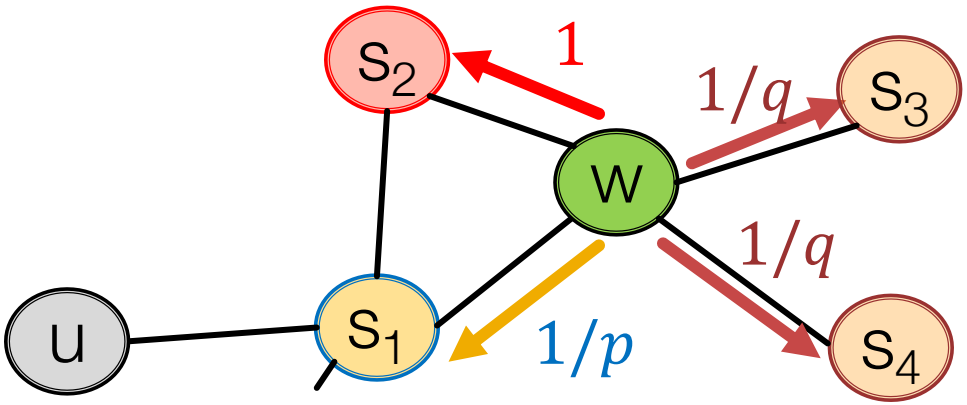



$1/p, 1/q, 1$ are unnormalized probabilities

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

# Biased Random Walks

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




$w \rightarrow$

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

Unnormalized transition prob. segmented based on distance from $s_1$

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

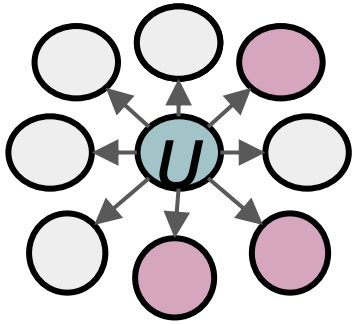
$N_R(u)$ are the nodes visited by the biased walk

# node2vec algorithm

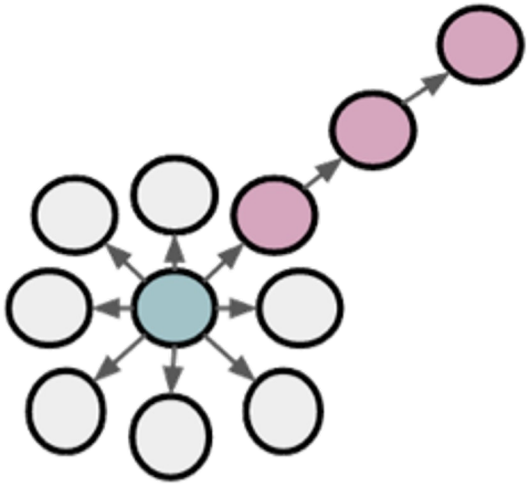
- 1) Compute random walk probabilities
- 2) Simulate $r$ random walks of length $l$ starting from each node $u$
- 3) Optimize the node2vec objective using Stochastic Gradient Descent

Linear-time complexity.
All 3 steps are individually parallelizable

# BFS vs. DFS



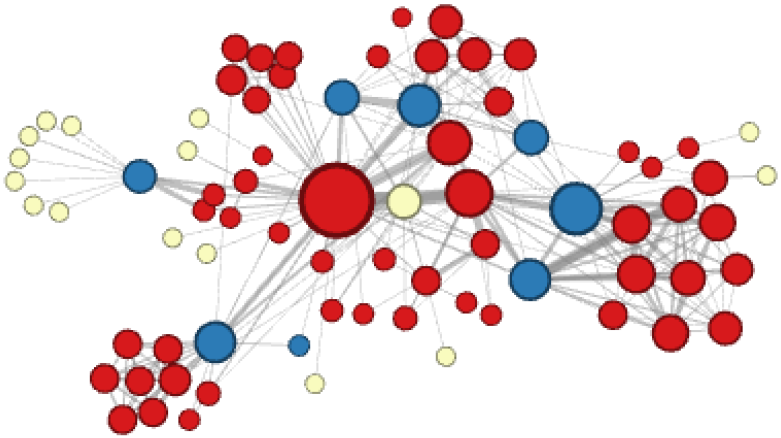

BFS:
Micro-view of neighbourhood



DFS:
Macro-view of neighbourhood

# Experiments: Micro vs. Macro

**Small network of interactions of characters in a novel:**



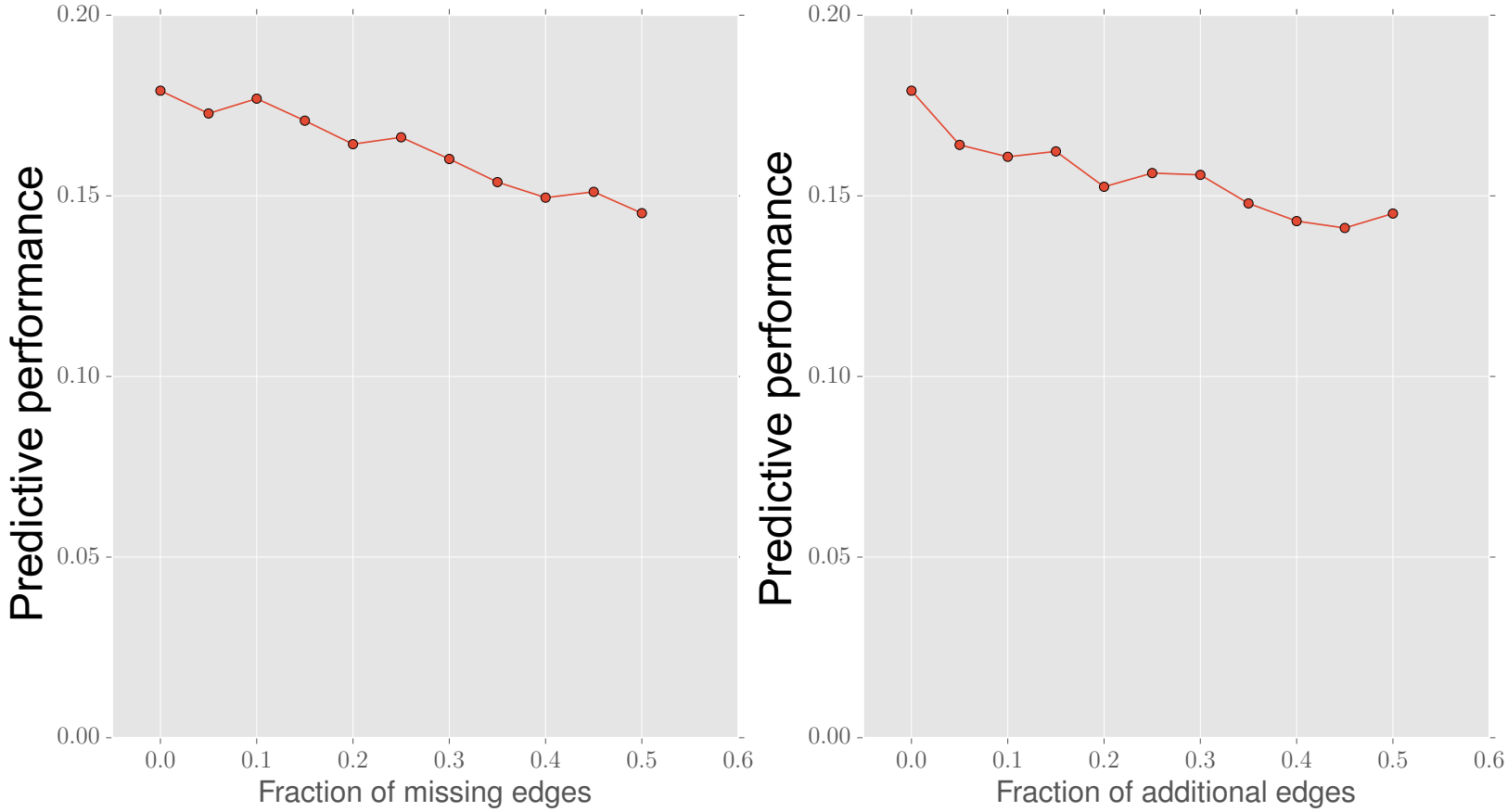
p=1, q=2

Microscopic view of the network neighbourhood

p=1, q=0.5

Macroscopic view of the network neighbourhood

# Node2vec: Incomplete Network




**How does predictive performance change as we**

- randomly remove a fraction of edges (left)
- randomly add a fraction of edges (right)

# Other random walk ideas

**(not covered in detailed here but for your reference)**

- **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).

# How to Use Embeddings

- **How to use embeddings $z_i$ of nodes:**
  - **Clustering/community detection:** Cluster nodes/points based on $z_i$
  - **Node classification:** Predict label $f(z_i)$ of node $i$ based on $z_i$
  - **Link prediction:** Predict edge $(i, j)$ based on $f(z_i, z_j)$
    - Where we can: 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: $f(z_i, z_j) = g(||z_i - z_j||_2)$

# Summary

- **Basic idea:** Embed nodes so that similarities 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 **(covered today)**

# Summary

- **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 perform better on link prediction (Goyal and Ferrara, 2017 survey)
- Random walk approaches are generally more efficient
- **In general:** Must choose def'n of node similarity that matches your application!