

Graph Representation Learning

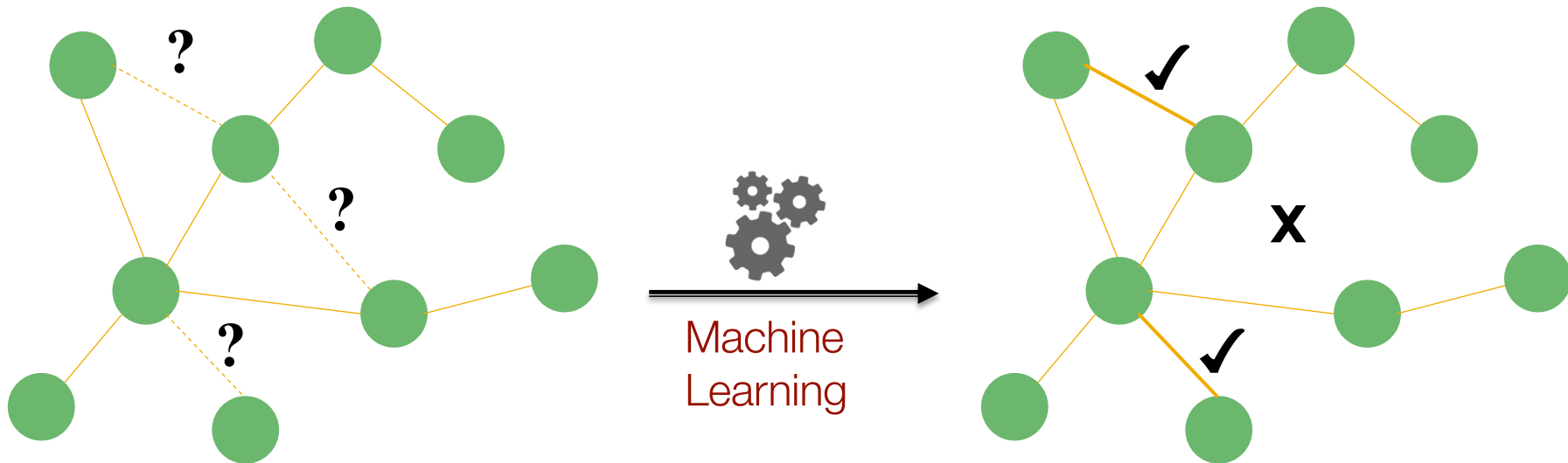
CSE547 Machine Learning for Big Data

Tim Althoff

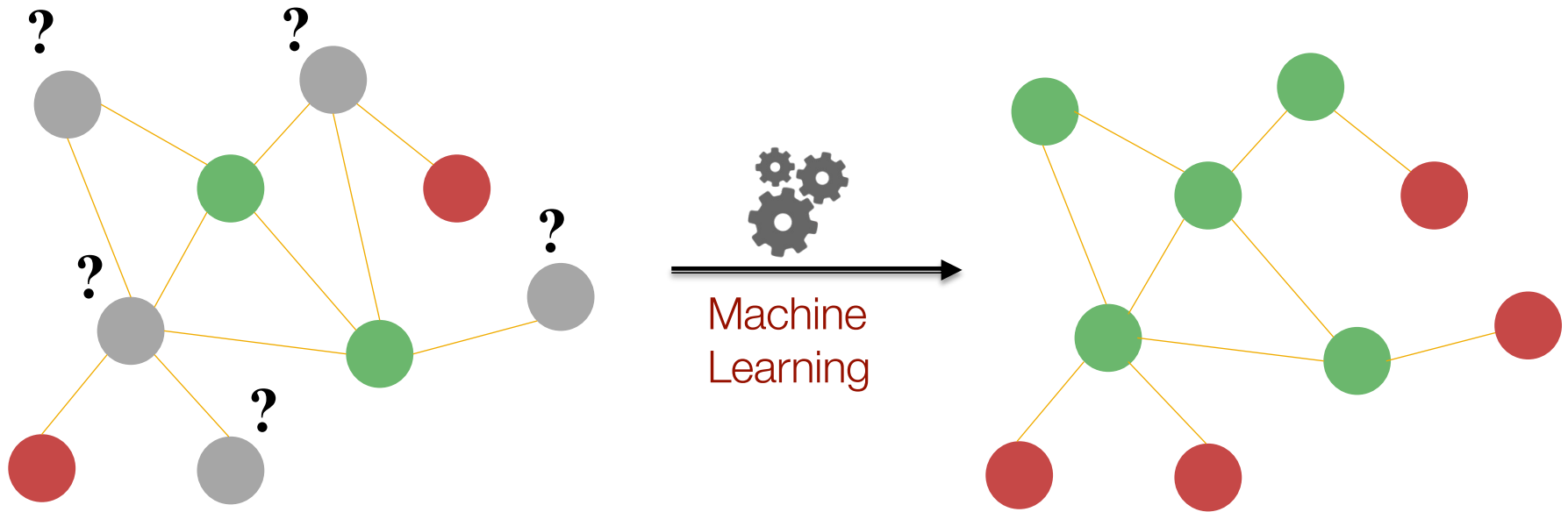


PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Example: Link Prediction



Machine Learning in Networks



Node classification

Example: Node Classification

Classifying the function of proteins in the interactome

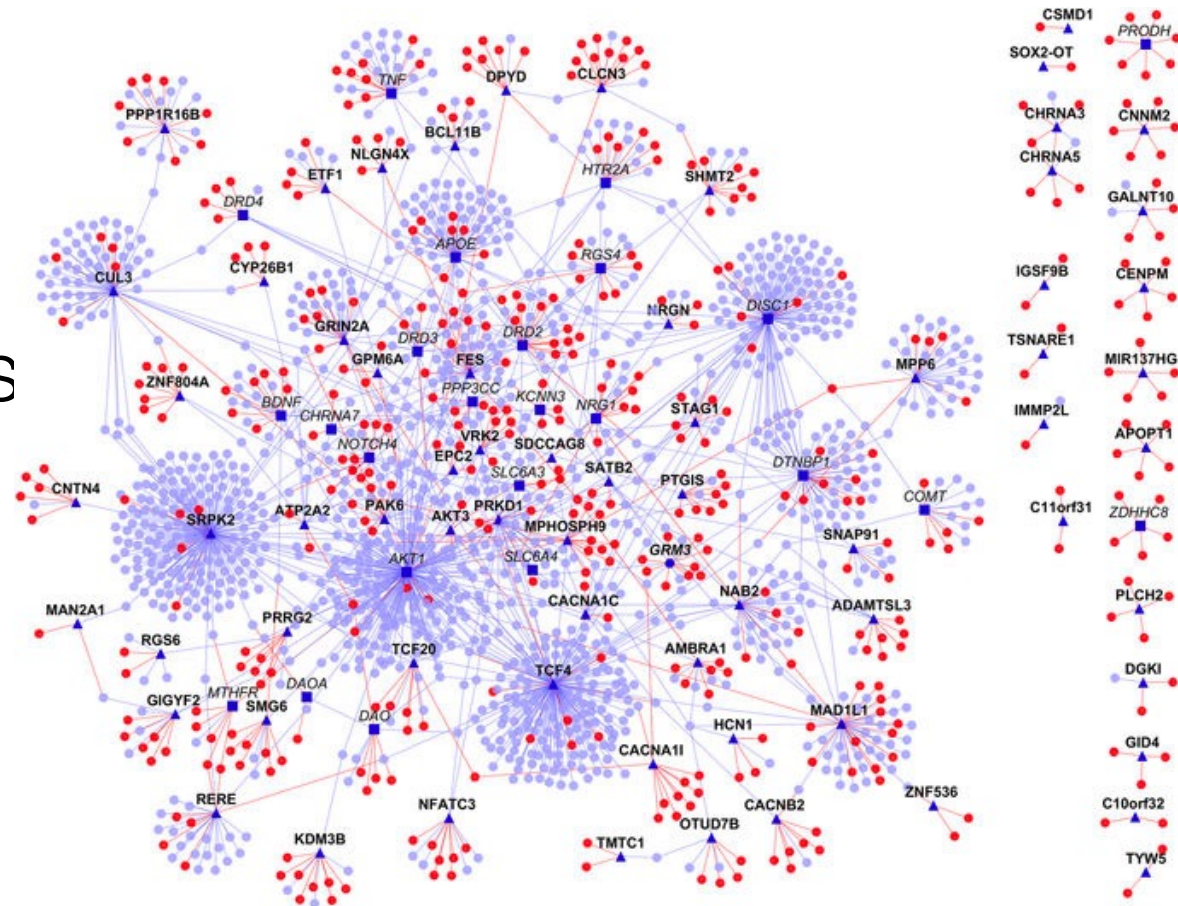
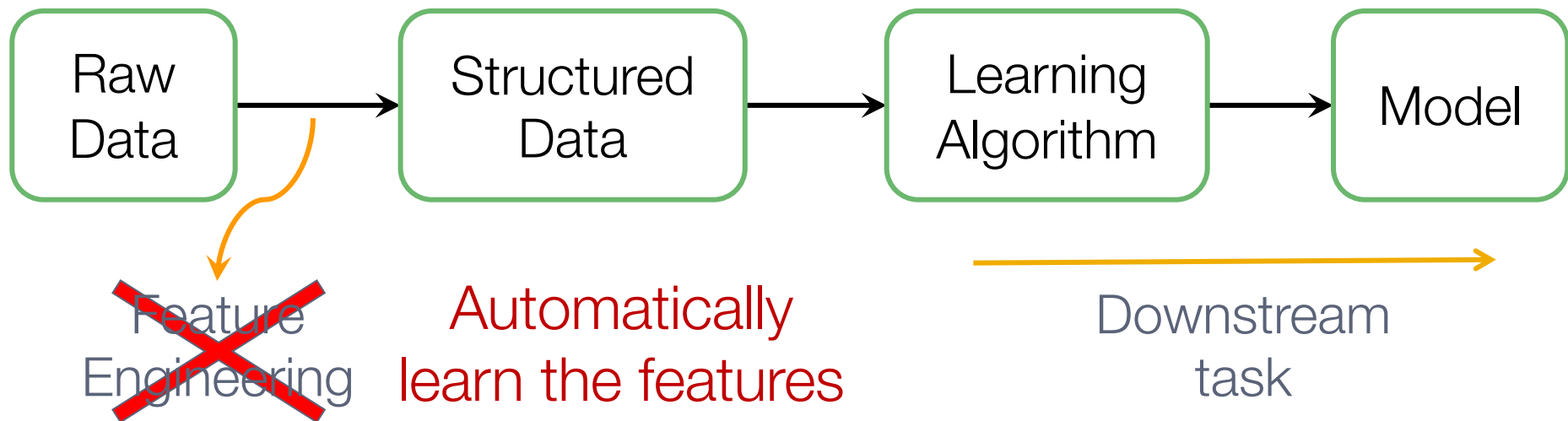


Image from: Ganapathiraju et al. 2016. [Schizophrenia interactome with 504 novel protein-protein interactions](#). *Nature*.

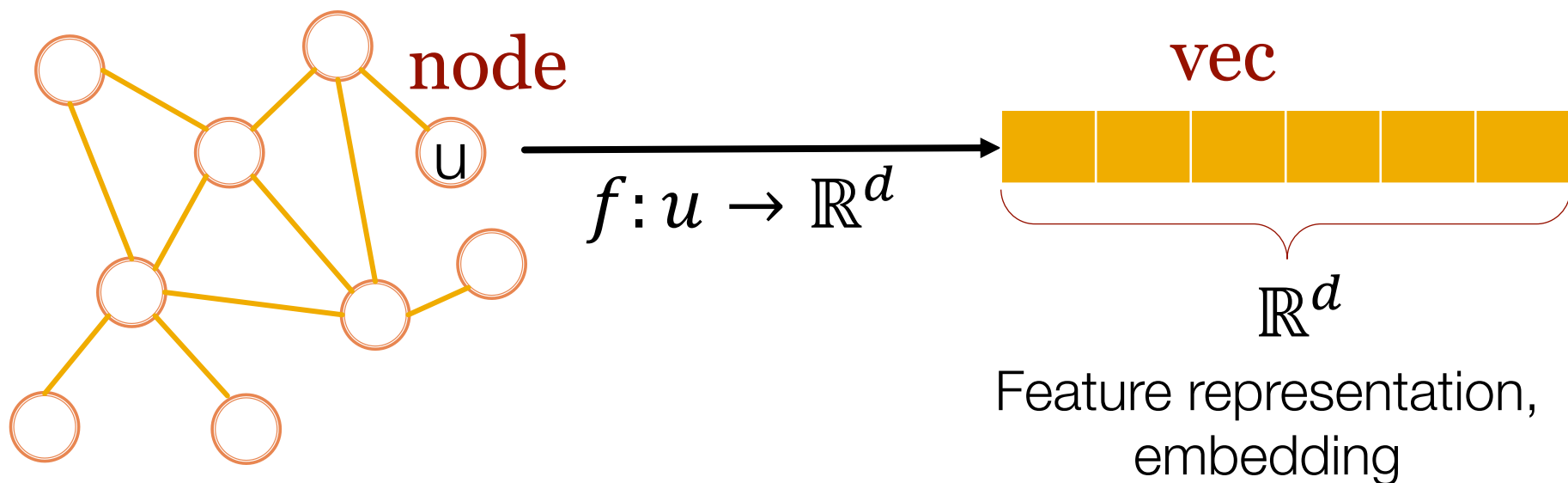
Machine Learning Lifecycle

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



Feature Learning in Graphs

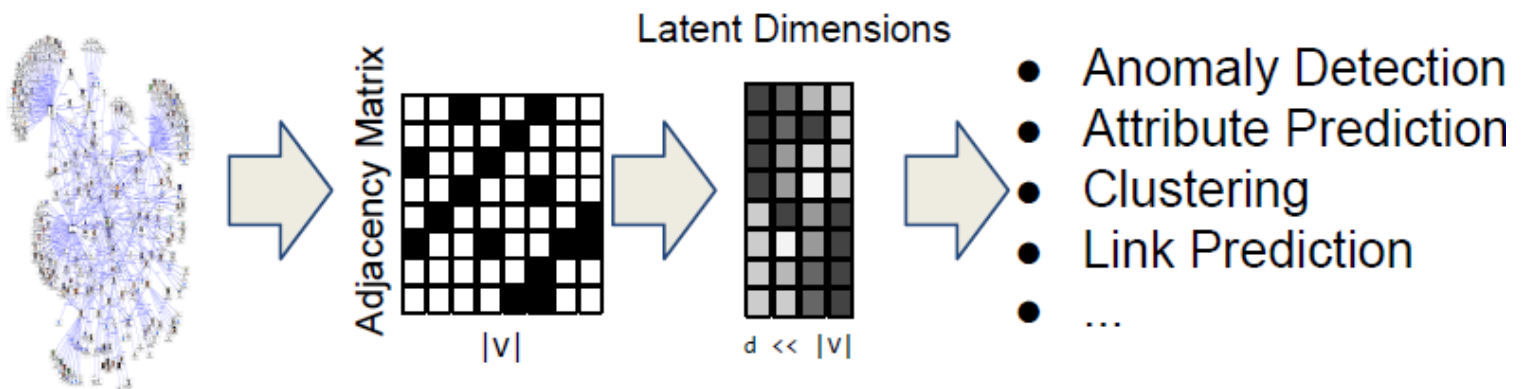
Goal: Efficient **task-independent** feature learning for machine learning in networks!



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:

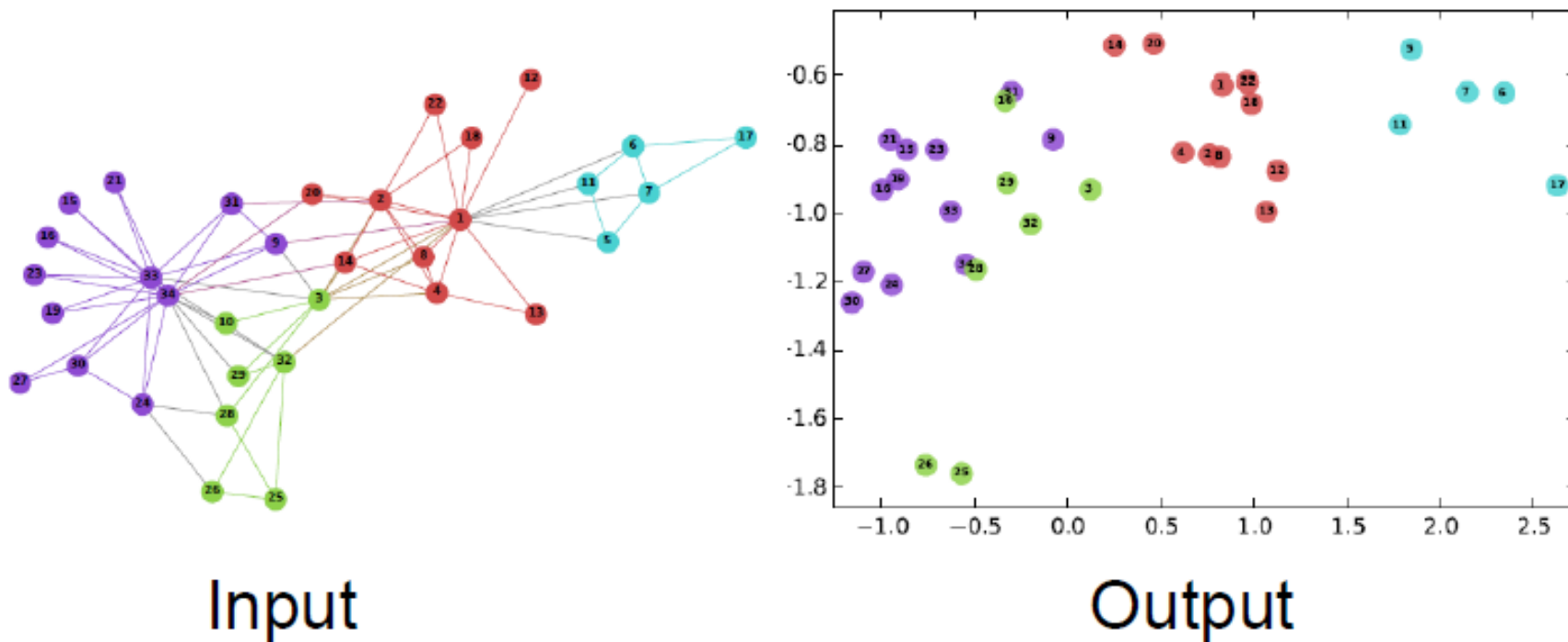
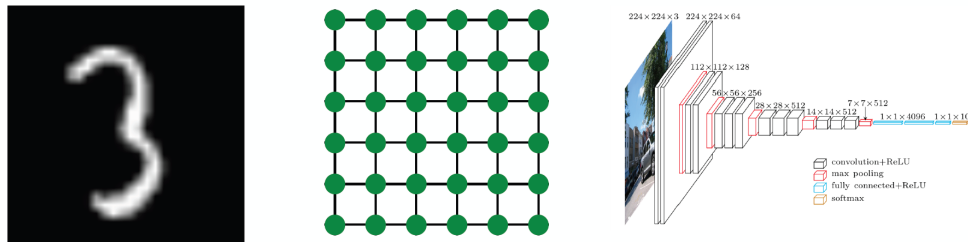


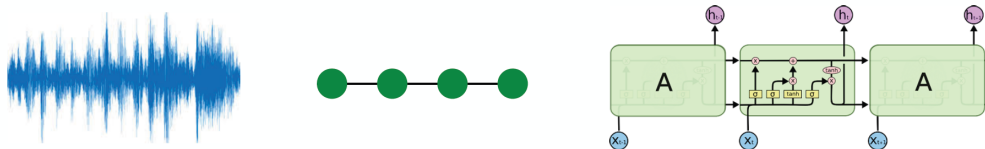
Image from: [Perozzi et al.](#) DeepWalk: Online Learning of Social Representations. *KDD 2014*.

Why Is It Hard?

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



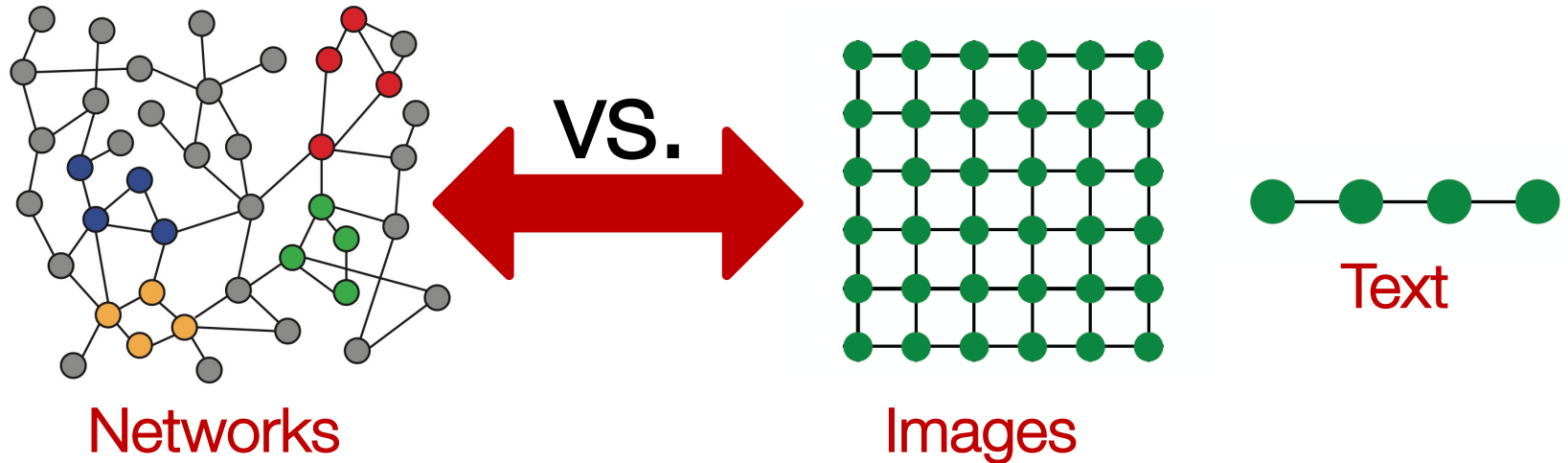
- Transformers, RNN, word2vec for text/sequences...



Why Is It Hard?

But networks are far more complex!

- Complex topographical structure (no spatial locality like grids)



- No fixed node ordering or reference point
- Often dynamic and have multimodal features.

Embedding Nodes

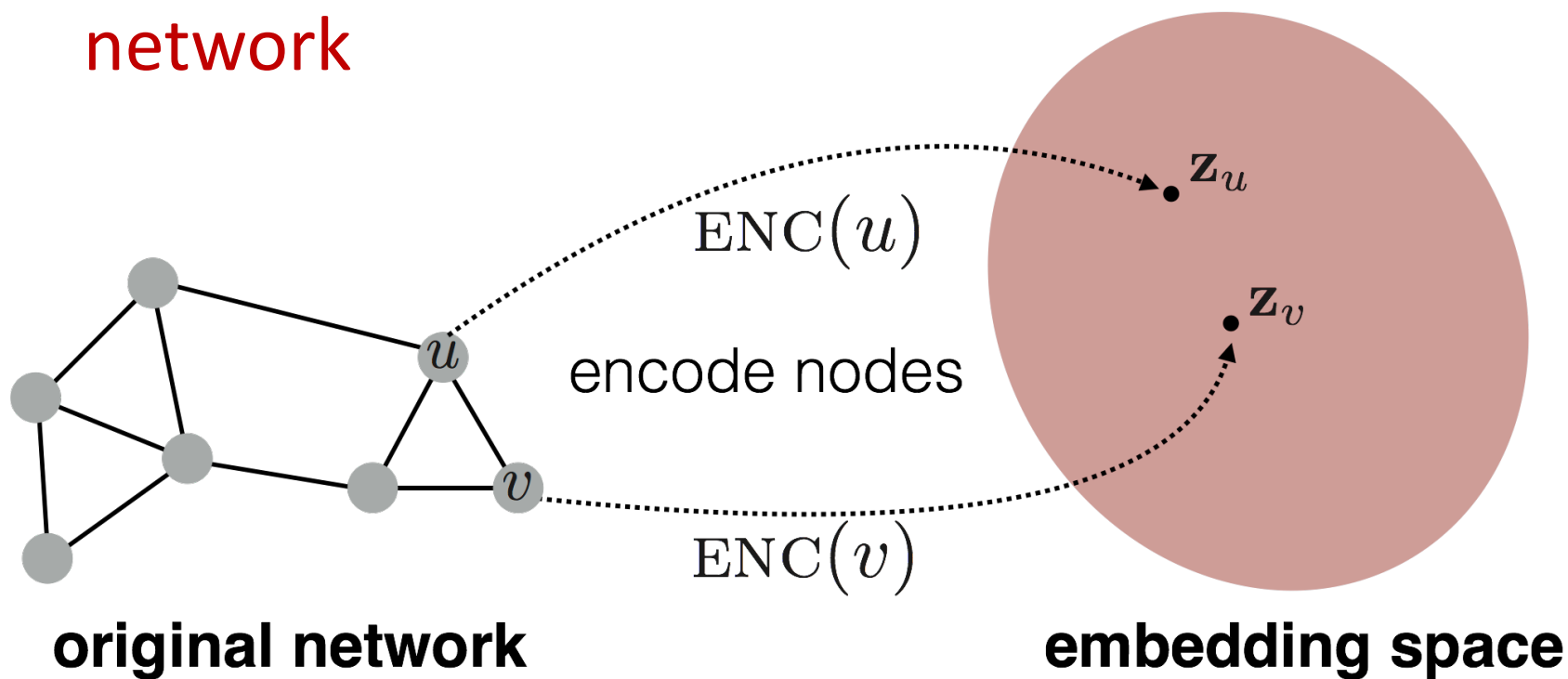
Setup

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

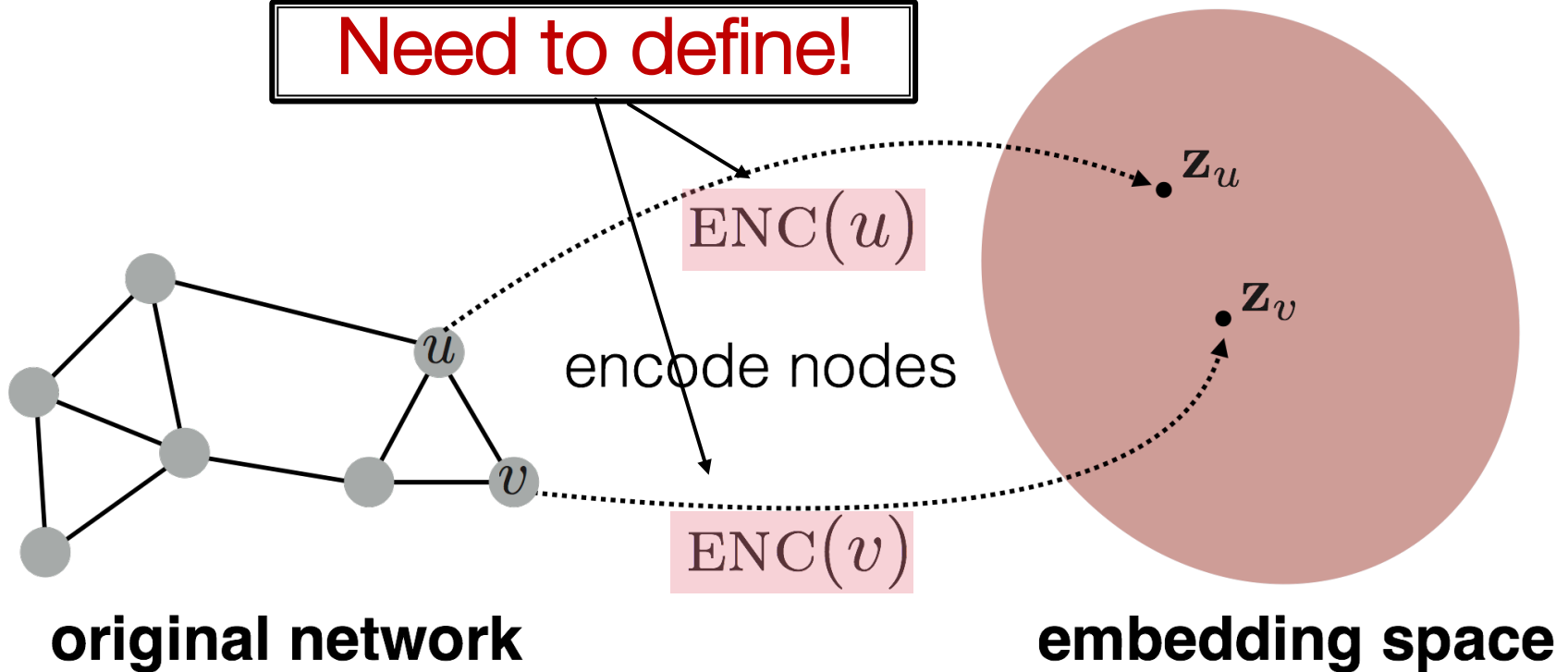
- Goal is to encode nodes so that **similarity in the embedding space** (e.g., dot product) approximates **similarity in the original network**



Embedding Nodes

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

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

in the original network Similarity of the embedding

Two Key Components

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

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

node in the input graph

d-dimensional embedding

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

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

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**

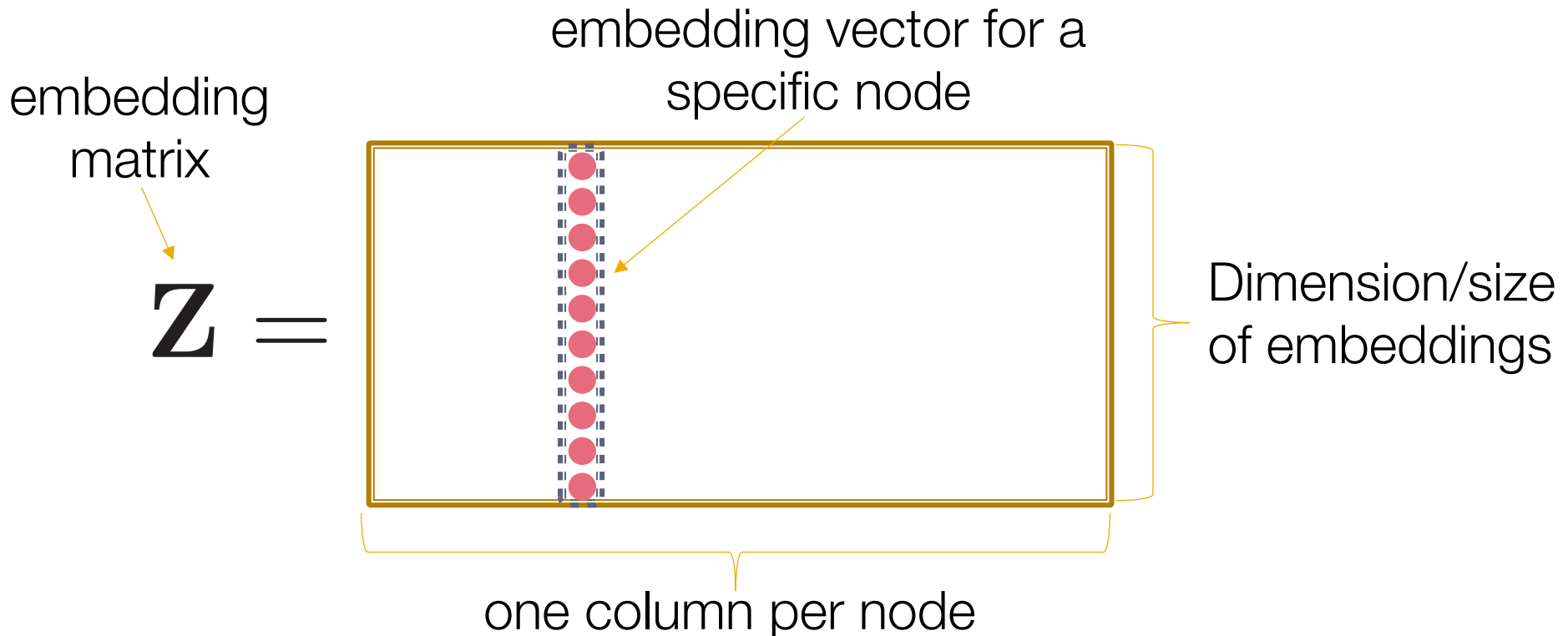
$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

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

$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ Indicator vector, all zeroes except 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 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”?
- ...?

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

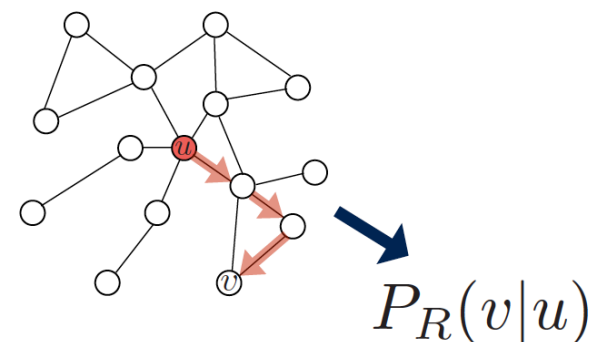
$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$

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

z_u ... embedding of node u

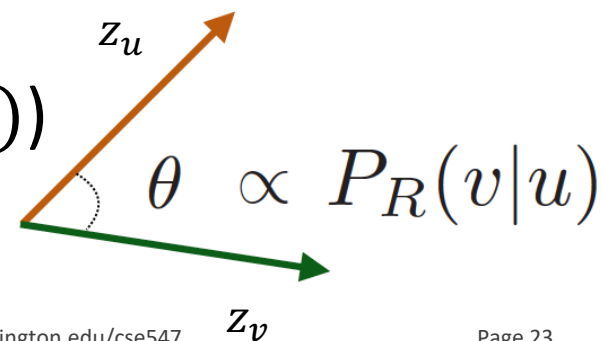
Random-walk Embeddings

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:

Similarity (here: dot product $\approx \cos(\theta)$) 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?**
 - $N_R(u)$... neighbourhood of u obtained by some strategy R

Feature learning as optimization

- Given $G = (V, E)$
- Our goal is to learn a mapping $z: u \rightarrow \mathbb{R}^d$
- Maximize 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 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 $N_R(u)$**

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

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

Random Walk Optimization

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

- **Assumption:** Conditional likelihood factorizes over the set of neighbors:

$$\log P(N_R(u) | z_u) = \sum_{v \in N_R(u)} \log P(z_v | z_u)$$

- **Softmax parametrization:**

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

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 u and v co-occurring on random walk

Optimizing random walk embeddings =

Finding node embeddings \mathbf{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**

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

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

sigmoid function

(makes each term a “probability”
between 0 and 1)

random distribution over
all nodes

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> and <https://arxiv.org/pdf/1410.8251.pdf>

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 prior on negative events

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 $N_R(u)$, 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?

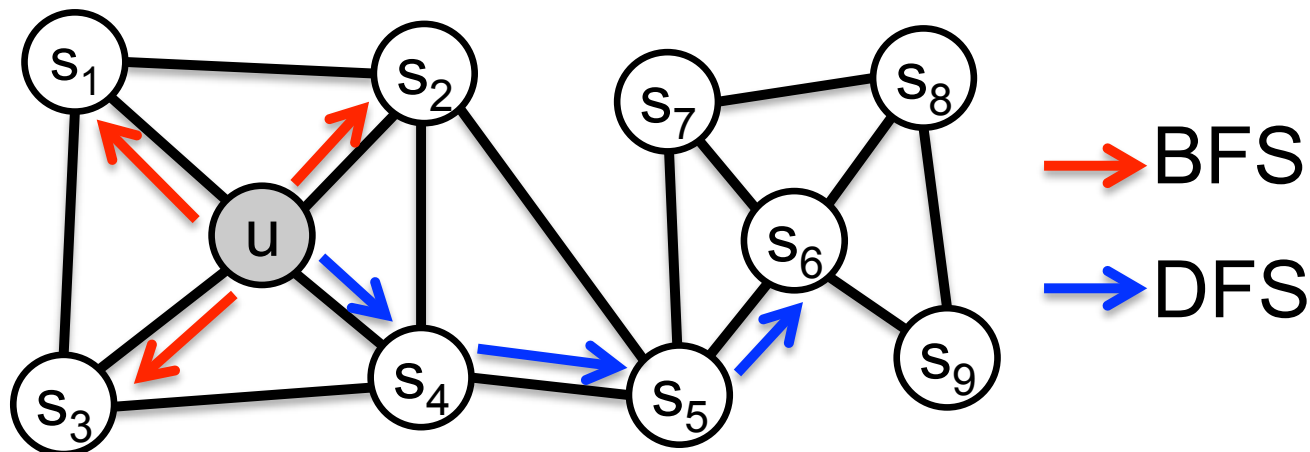
- So far we have described how to optimize embeddings given random walk statistics
- **What strategies should we use to run these random walks?**
 - Simplest idea: **Just run fixed-length, unbiased random walks starting from each node** (i.e., [DeepWalk from Perozzi et al., 2013](#)).
 - The issue is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

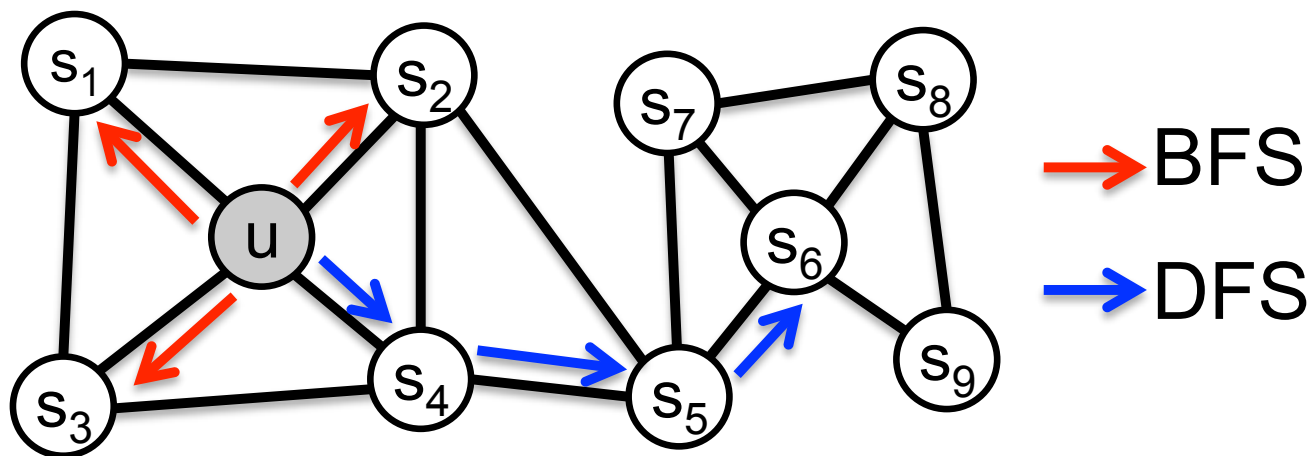
node2vec: Biased Walks

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

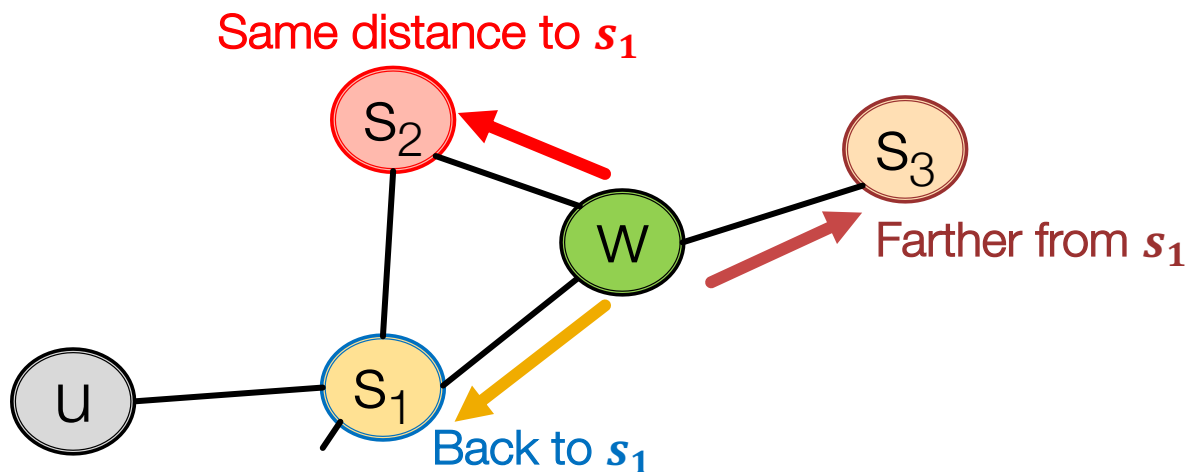
Biased fixed-length random walk R that given a node u generates neighborhood $N_R(u)$

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

Biased Random Walks

Biased 2nd-order random walks explore network neighborhoods:

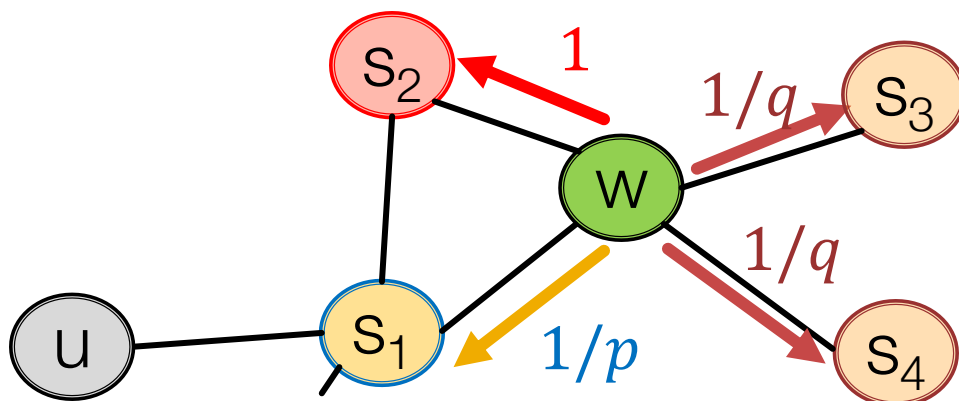
- Rnd. walk just traversed edge (s_1, w) and is now at w
- **Insight:** Neighbors of w can only be:



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?



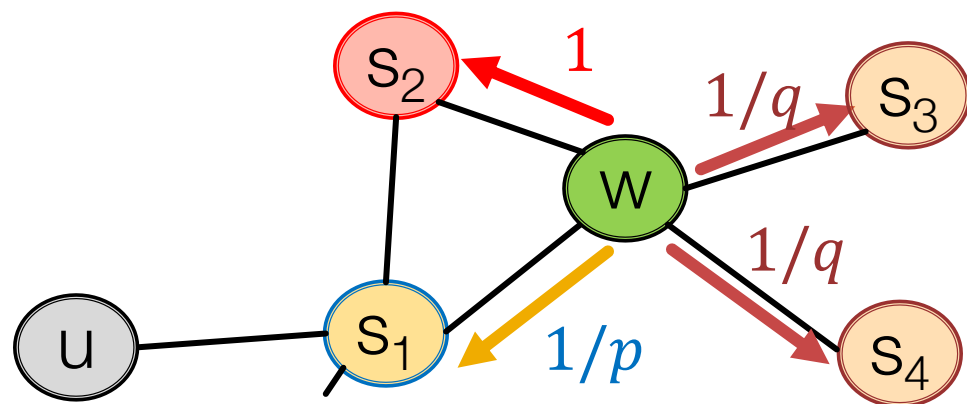
$1/p, 1/q, 1$ are
“unnormalized”
probabilities (weights
we later convert to
probability distribution)

- 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

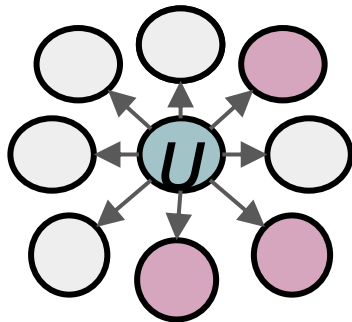
node2vec algorithm

- 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.

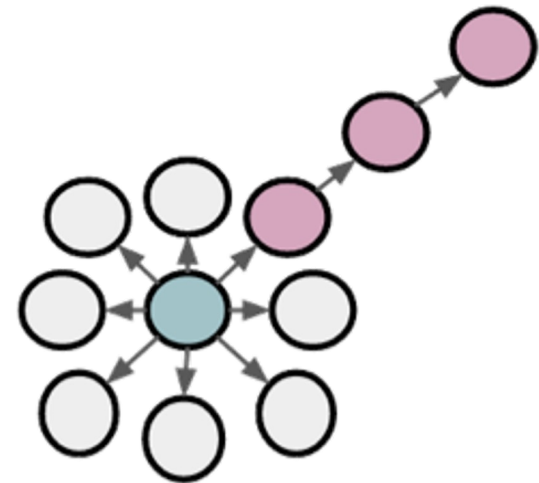
All 3 steps are individually parallelizable

BFS vs. DFS



BFS:

Micro-view of
neighbourhood

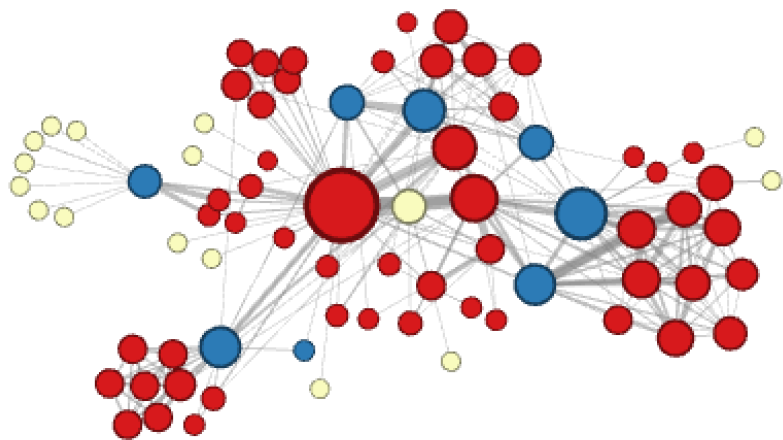


DFS:

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

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 so far

- **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
- **In general:** Must choose def'n of node similarity that matches your application!

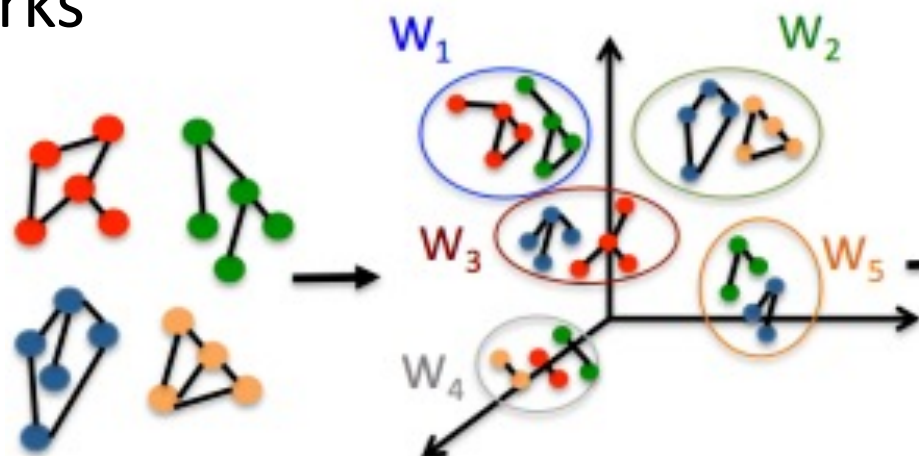
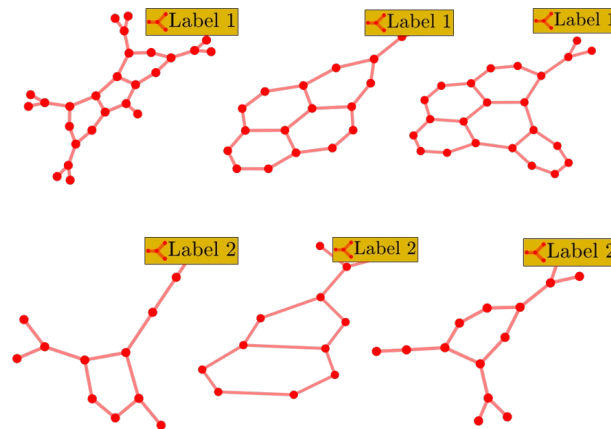
Embedding Entire Graphs

Graph Classification

■ Tasks:

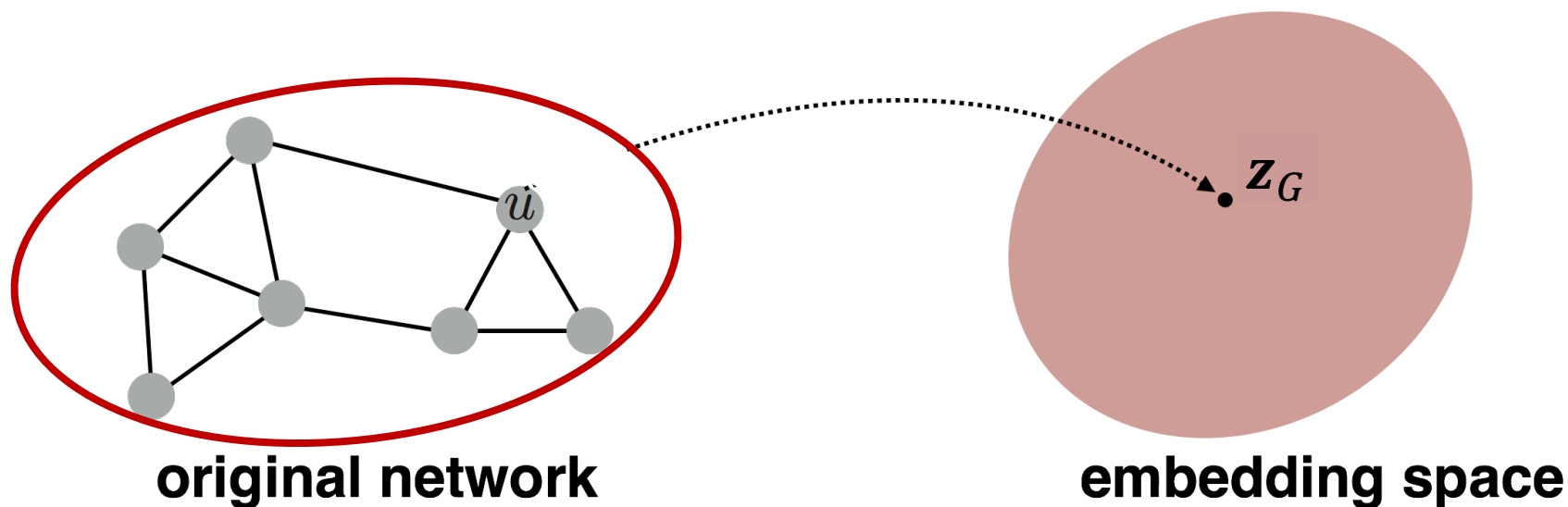
- Classifying toxic vs. non-toxic molecules
- Identifying cancerogenic molecules
- Graph anomaly detection
- Classifying social networks

Graph Classification



Embedding Entire Graphs

- **Goal:** Want to embed an entire graph G



Approach 1

Simple idea:

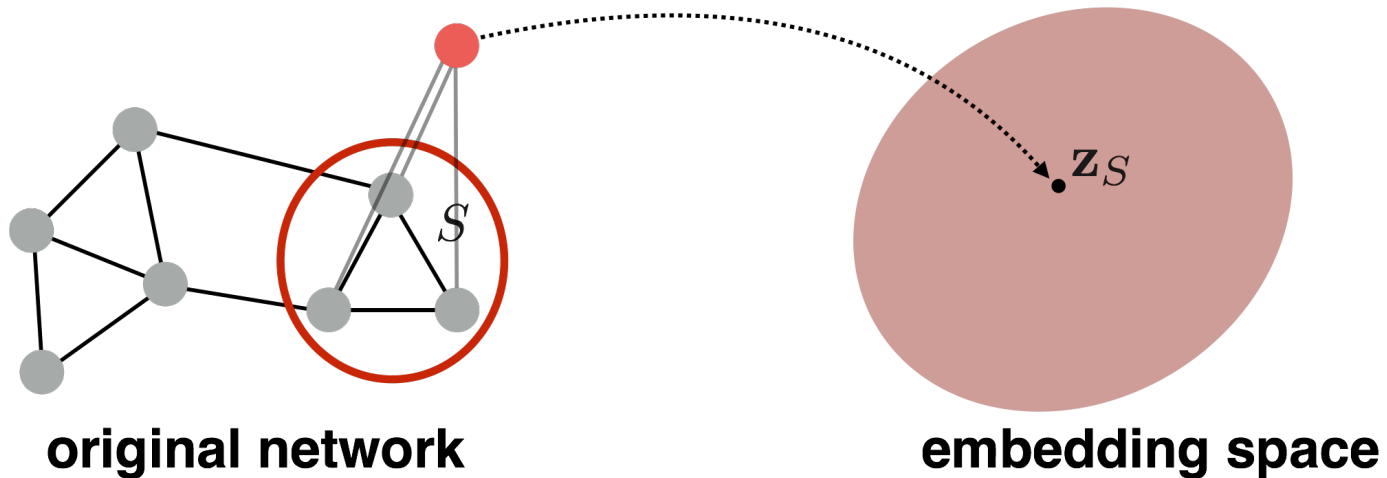
- Run a standard node embedding technique *on* the (sub)graph G
- Then just sum (or average) the node embeddings in the (sub)graph G

$$z_G = \sum_{v \in G} z_v$$

- Used by [Duvenaud et al., 2016](#) to classify molecules based on their graph structure

Approach 2

- **Idea:** Introduce a “**virtual node**” to represent the (sub)graph and run a standard graph embedding technique



- Proposed by [Li et al., 2016](#) as a general technique for subgraph embedding

Please give us feedback 😊
<https://bit.ly/CSE547feedback2024>