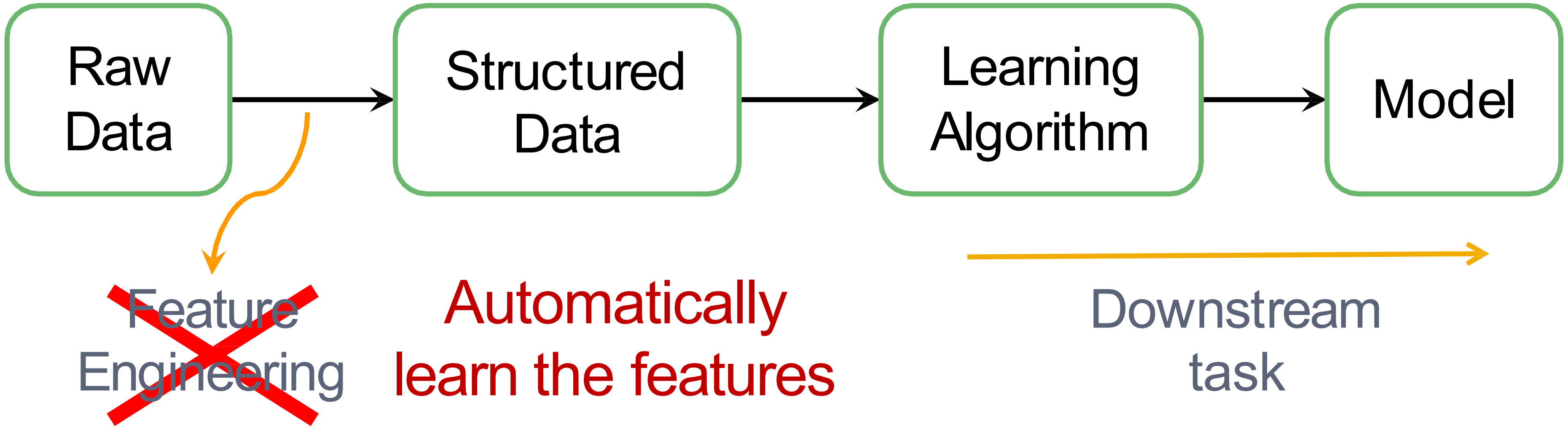
Graph Representation Learning

上海交通大学
约翰·霍普克罗夫特计算机科学中心
John Hopcroft Center for Computer Science

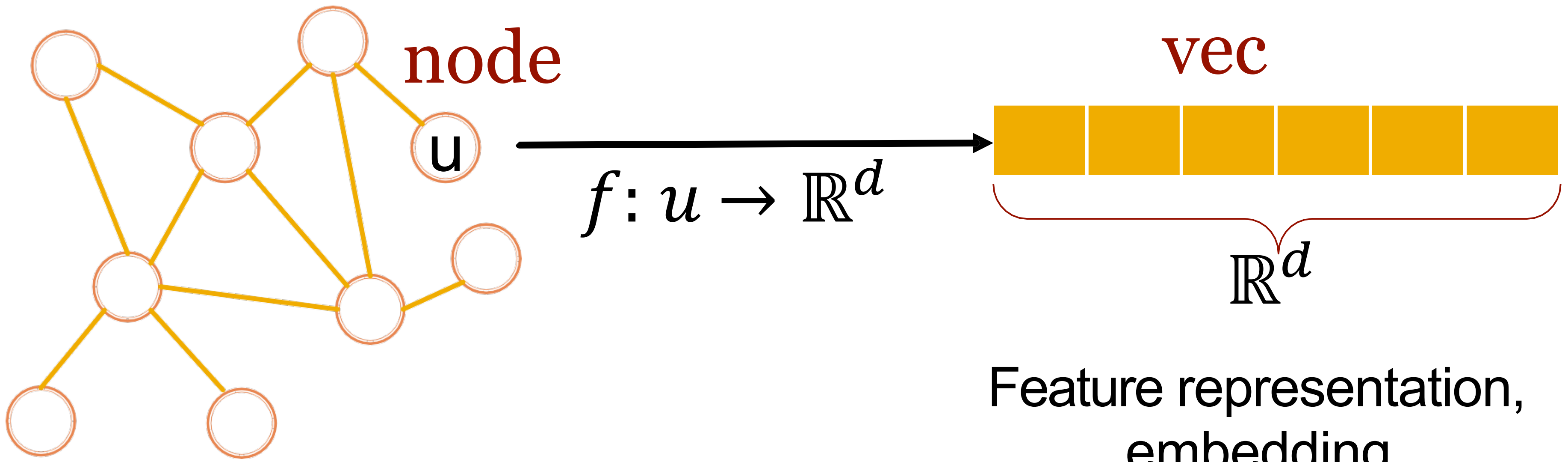
Machine Learning Lifecycle

Machine Learning Lifecycle requires feature engineering. Feature engineering is painful! How can we make it easy?



Feature Learning in Graphs

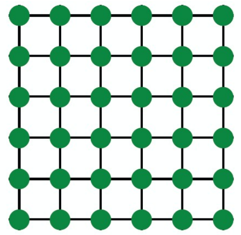
- **Goal:** Efficient task-independent feature learning for machine learning in graphs!

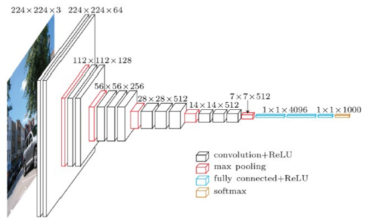


Feature representation, embedding

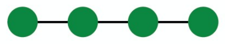
Why is it hard?

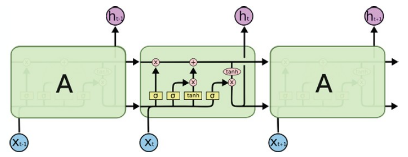
- **Modern deep learning toolbox is designed for simple sequences or grids**
  - CNNs for fixed-size images/grids….





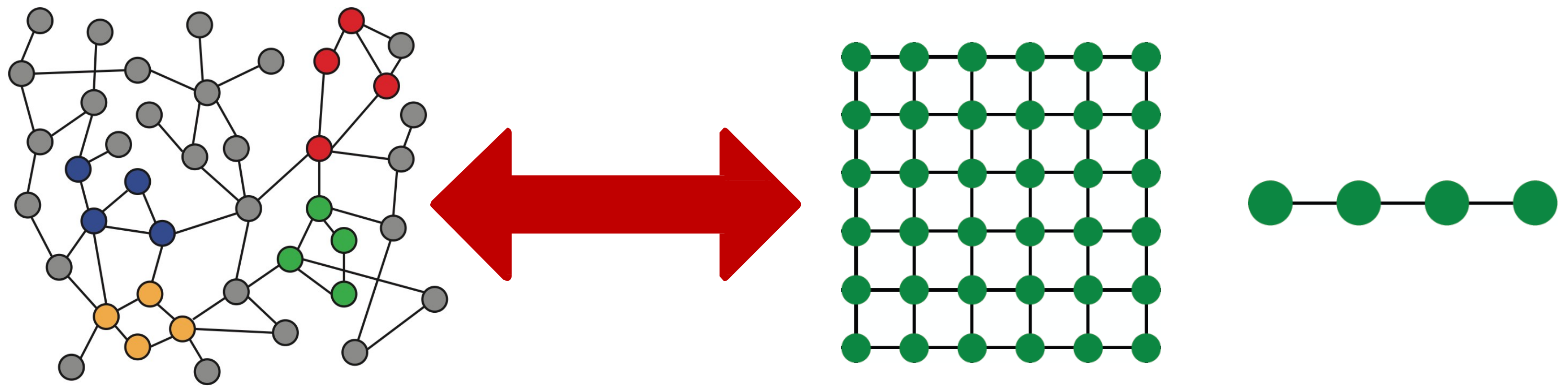
  - RNNs or word2vec for text/sequences…





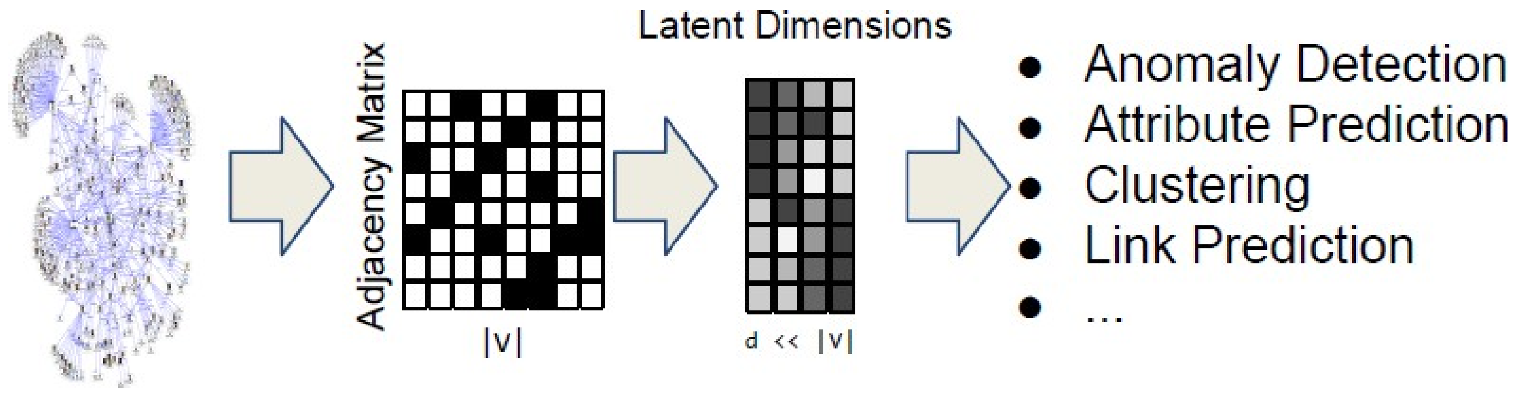
Why is it hard?

- **Networks are far more complex.**
  - Complex topographical structure
  - No fixed node ordering or reference point.



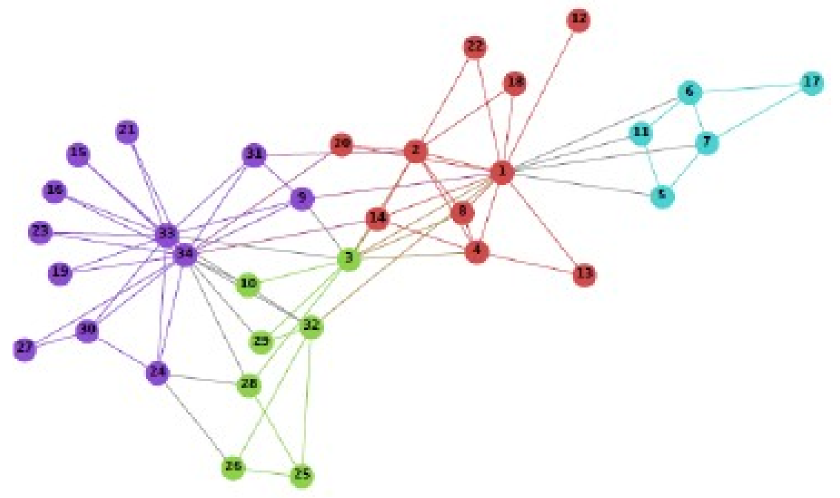
Node embedding

- **Task: Give a graph G, we map each node in a network to a point in a low-dimensional space**
  - Encode network information and generate node representation
  - Similarity of embedding between nodes approximates their original network similarity

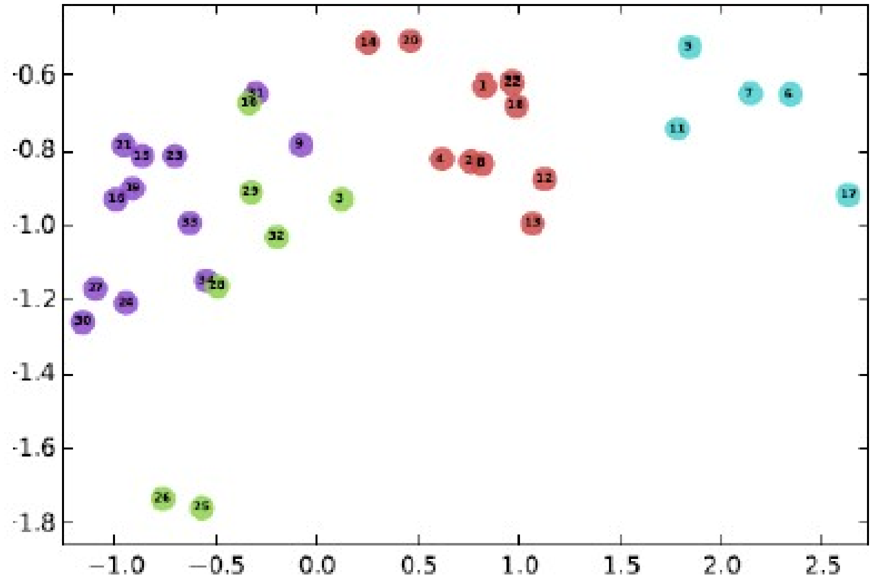


Example: Node Embedding

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



Input



Output

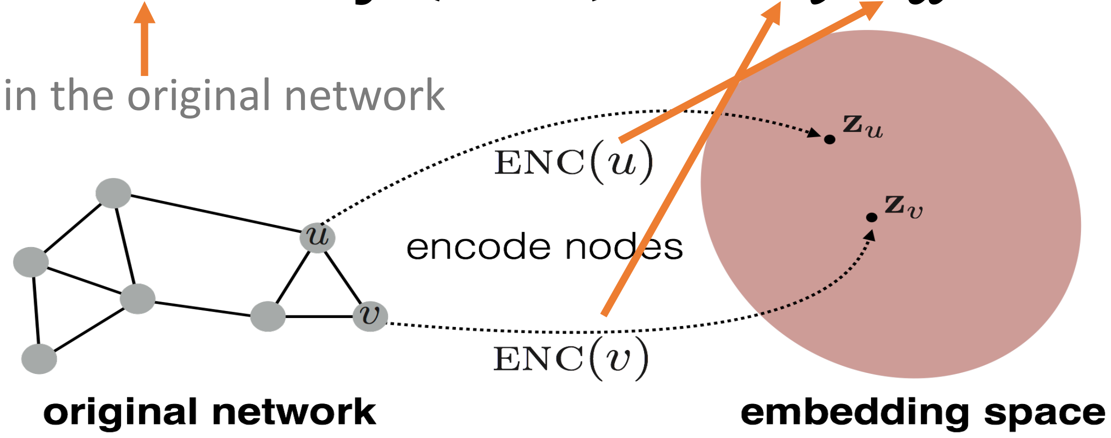
Image from: Perozzi et al. DeepWalk: Online Learning of Social Representations. *KDD 2014.*

Learning Node Embeddings

- Define a **node similarity function** $\text{similarity}(u, v)$
  - a measure of similarity in the original network
- **Encoder ENC** maps from nodes to embeddings $\text{ENC}(v) = z_v$
- **Decoder DEC** maps from embedding to the similarity score(e.g. dot product $\boldsymbol{z}_v^T \boldsymbol{z}_u$)
- **Optimize the parameters of the encoder so that:**

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

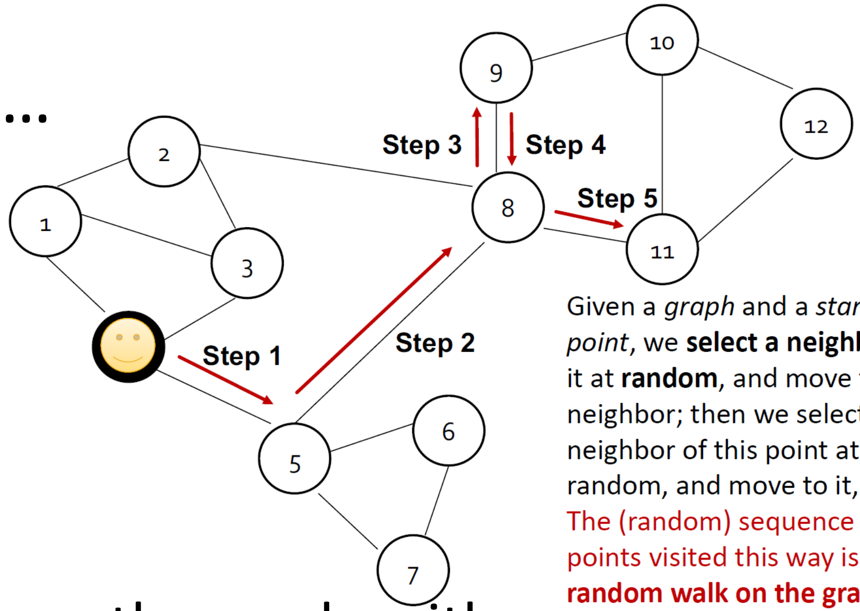
Similarity in the original network



How to define node similarity?

- Should two nodes have similar embeddings if they…
  - are connected?
  - share neighbors?
  - have similar "structural roles"?

- **Random walk approaches for node embedding!**
  - **Expressivity:** if random walk starting from a node visits another node with high probability, they are similar
  - **Efficiency:** only need to consider pairs that co-occur on random walks
- **We will introduce DeepWalk, node2vec.**

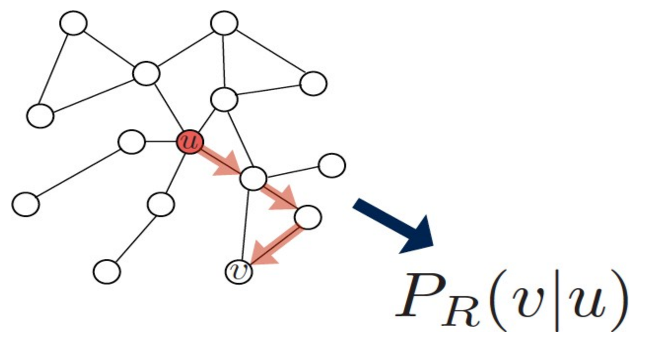


Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points visited this way is a **random walk on the graph**.

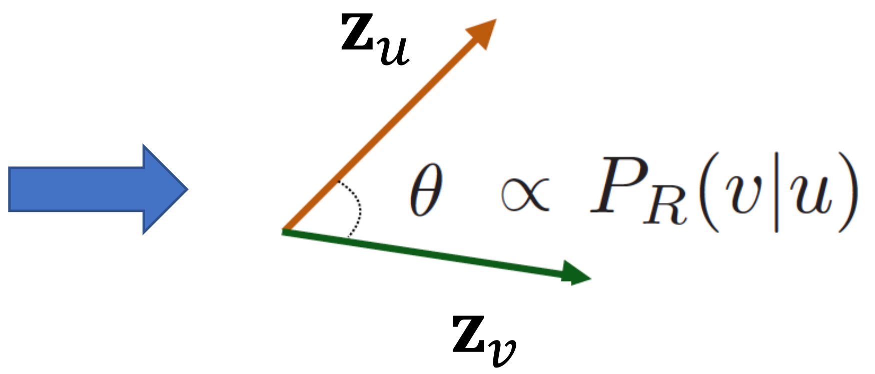
Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014.
Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016.

Random Walk Embeddings

$\mathbf{z}_u^T \mathbf{z}_v \approx$ Probability that u and v co-occur on a random walk over the graph



Estimate probability of visiting node $\boldsymbol{v}$ on a **random walk** starting from node $\boldsymbol{u}$ using some random walk **strategy** $\boldsymbol{R}$



Optimize embeddings to encode these random walk statistics

Notation

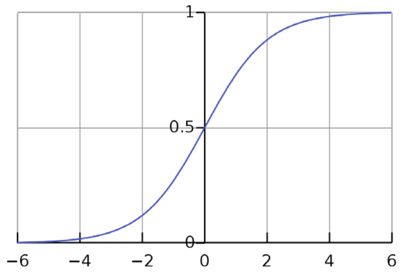
- **Vector** z_u: the embedding of node u
- **Probability** $P(v|z_u)$:
  - The predicted probability of visiting node v on random walks starting from node u (our model prediction based on z_u)

- **Softmax** function:

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

Turns vector of K real values (model predictions) into K probabilities that sum to 1

- **Sigmoid** function:



$$S(x) = \frac{1}{1+e^{-x}}$$

S-shaped function that turns real values into the range of (0, 1).

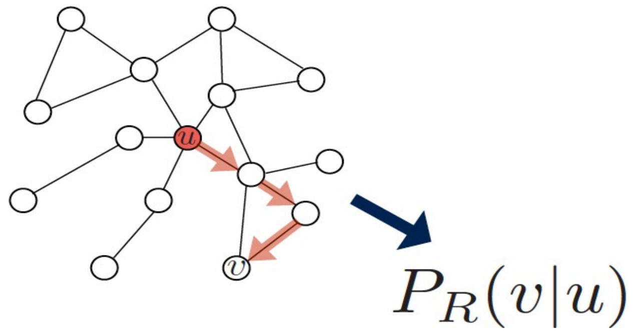
Feature Learning Objective

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

Such that:

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

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



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

Random Walk Optimization $N_R(u)$

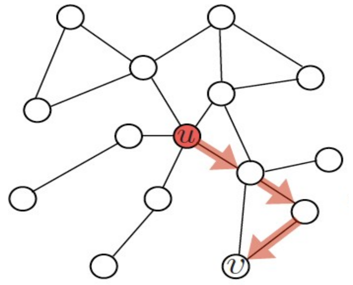
- Run short fixed-length random walks starting from each node on the graph using some strategy R
- For each node u collect $N_R(u)$, the multiset of nodes visited on random walks starting from u
- Optimize embeddings according to: Given node u, predict its neighbors $N_R(u)$

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

Random Walk Optimization $P(N_R(u)|z_u)$

- **Assumption:** Conditional likelihood factorizes over the set of neighbors $P(N_R(u)|z_u) = \Pi_{v \in N_R(u)} P(v|z_u)$
- The objective becomes:

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



- Softmax parametrization $P(v|z_u)$:

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}$$

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 together

Computation cost is high! $O(V^2)$

$$\mathcal{L} = \underbrace{\sum_{u \in V}}_{\text{sum over all nodes } u} \underbrace{\sum_{v \in N_R(u)}}_{\text{sum over nodes } v \text{ seen on random walks starting from } u} - \log \left(\underbrace{\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}}_{\text{predicted probability of } v \text{ appearing in random walk starting from } u} \right)$$

- Optimization random walk embedding is finding node embedding that minimize this loss function

Negative Sampling

- **Reduce Computation Cost: Negative sampling**

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

分母很复杂，就不全算，算一个很小的子集

Detailed proof is here

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

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

- $n_i \sim P_v$
- Sample k negative nodes proportional to **degree**
- Two considerations for k (# negative samples):
  - Higher k gives more **robust** estimates
  - Higher k corresponds to higher **prior** on **negative events**. In practice, $k = 5 \sim 20$

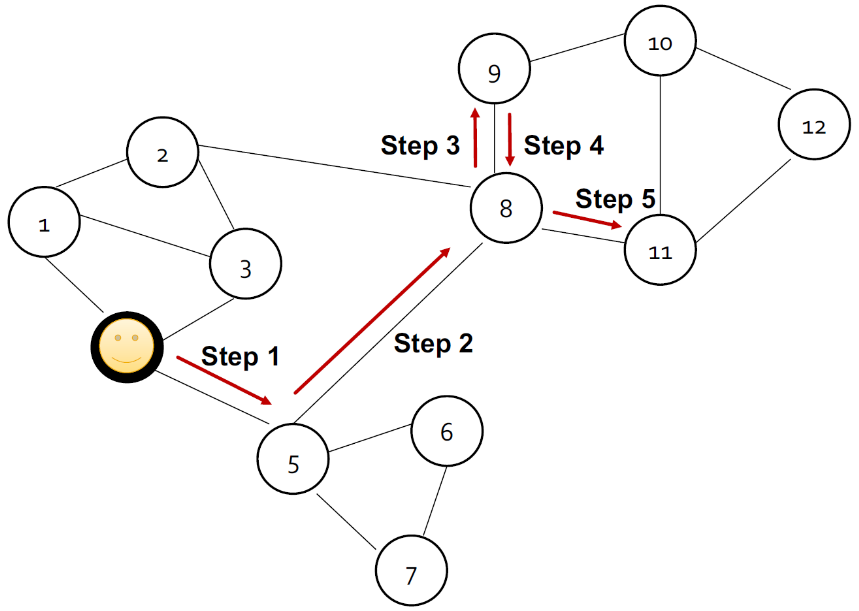
Random Walk Embedding Framework

- Run **short fixed-length** random walks starting from each node on the graph using some strategy R.
- For each node u collect $N_R(u)$, the multiset of nodes visited on random walks starting from u
- Optimize embeddings using **stochastic gradient descent** with loss function

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

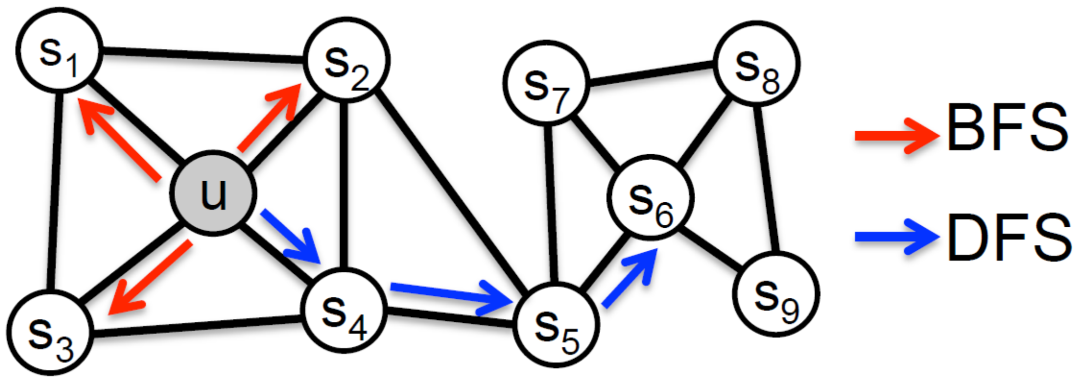
How should we randomly walk?

- **What strategies should we use to run these random walks?**
- Simplest idea: **Just run fixed-length, unbiased random walks starting from each node** （**DeepWalk** from Perozzi et al., 2013）
  - The issue is that such notion of similarity is too constrained
- How can we generalize this?



Biased Random Walk

- Second order random walk R: remember the previous step.
    - Richer node structures
- Use flexible, biased random walks that can trade off between local and global views of the network (**Node2Vec**, 2016).



Two strategies to define neighborhoods:
Breadth First Search(BFS)
Depth First Search(DFS).

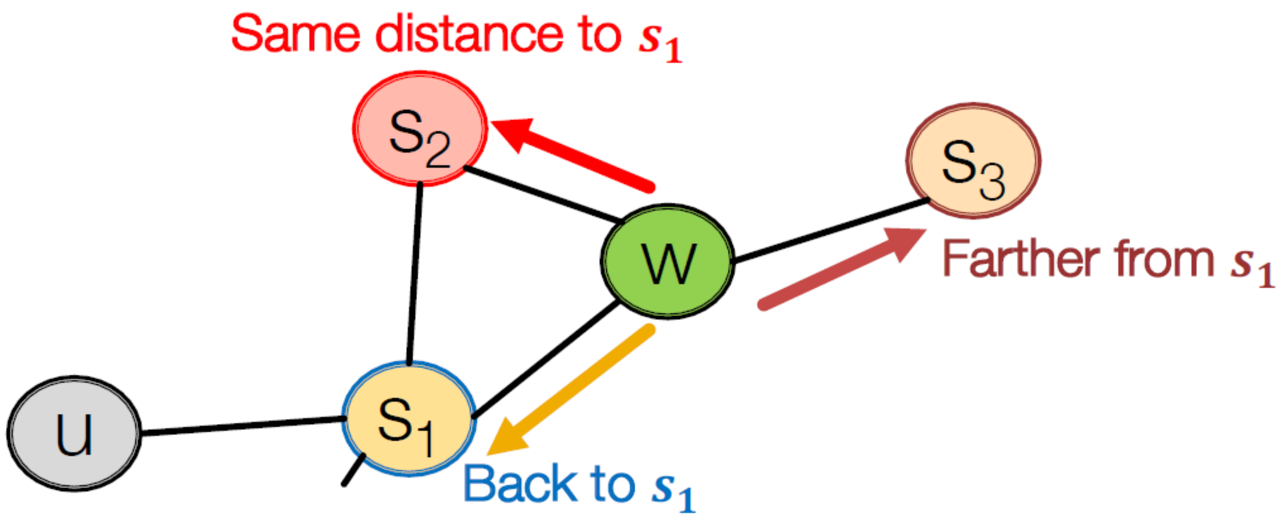
Walk of length 3 ($N_R(u)$ of size 3):

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

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

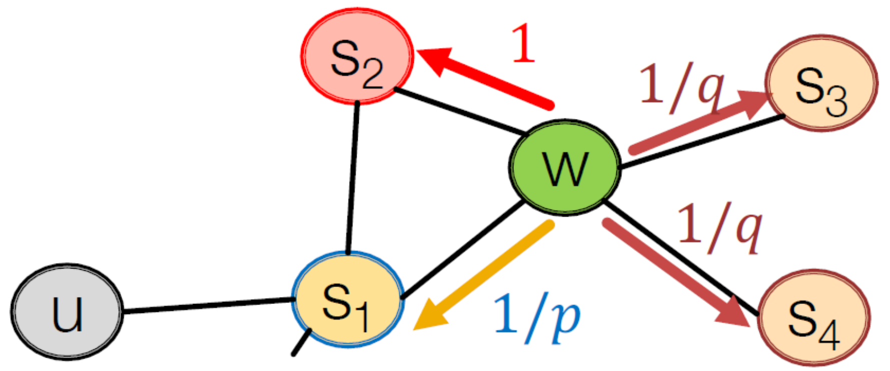
Biased Random Walks

- Biased second order random walks to explore neighborhoods:
  - Random walk just traversed edge (s_1, w) and is now at w
  - Neighbors of w can only be:



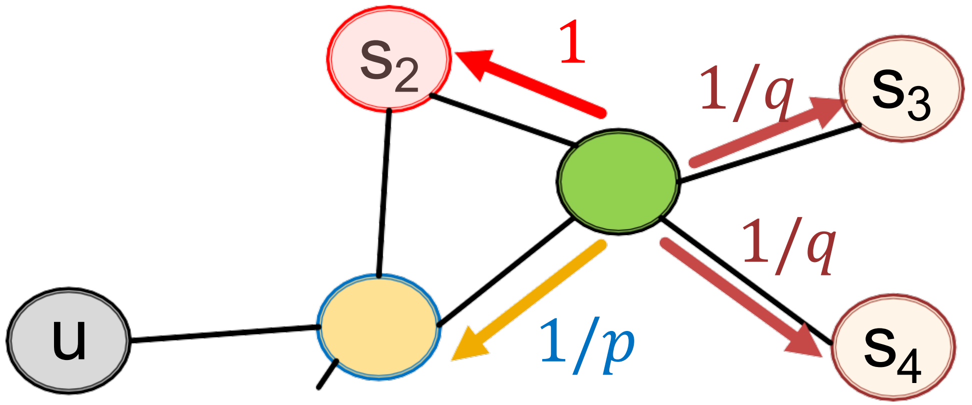
Tadeoff between BFS and DFS

- Two parameters to tune:
  - Return parameter p: the ratio between **BFS** and **return** to the previous node
  - In-out parameter q: the ratio between **BFS** and **DFS**



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

Biased Random Walk



$w \rightarrow$

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

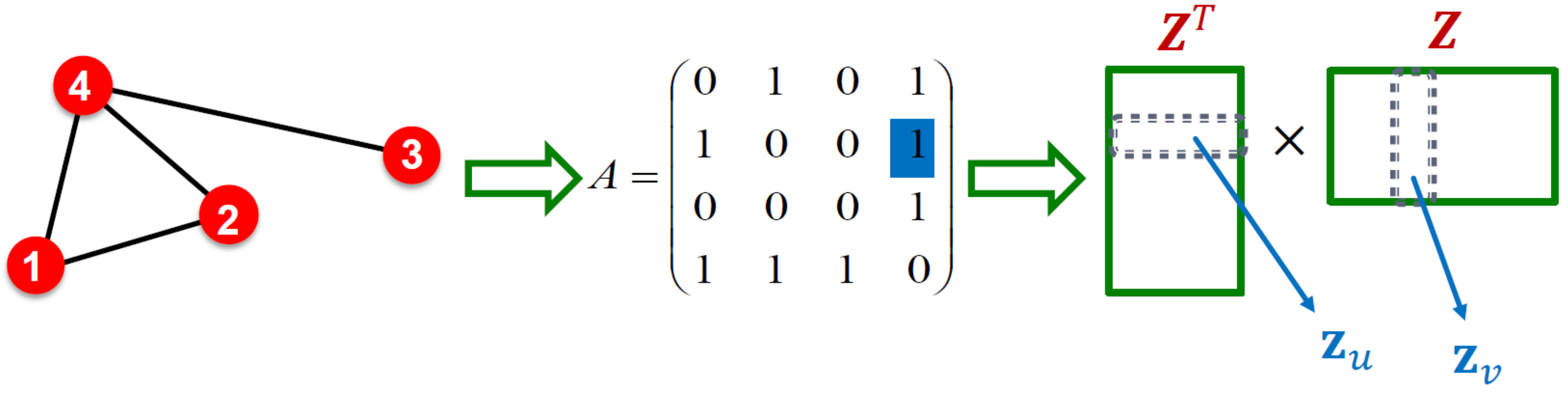
Node2vec Algorithm

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

Network Embedding as Matrix Factorization

- Inner product decoder with node similarity defined by edge connectivity is equivalent to matrix factorization of adjacency matrix A
  - Exact factorization $A = Z^T Z$ is generally not possible
  - We can approximate by

$$\min_{Z} \left\| A - Z^T Z \right\|_2$$



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

What does DeepWalk do?

- **DeepWalk** have a more complex **node similarity** definition based on random walks, equivalent to matrix factorization of:

Volume of graph

$$vol(G) = \sum_i \sum_j A_{i,j}$$

Diagonal matrix D

$$D_{u,u} = \deg(u)$$

$$\log\left(vol(G)\left(\frac{1}{T}\sum_{r=1}^{T}(D^{-1}A)^r\right)D^{-1}\right) - \log b$$

context window size

Power of normalized adjacency matrix

Number of negative samples

$$T = |N_R(u)|$$

Qiu, Jiezhong, et al. "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec." *WSDM 2018*

How to use node embedding

- Node classification: predict label based on embedding
- Link prediction: predict edge (i, j) based on $f(z_i, z_j)$, where we can concatenate, avg, product, or take a difference between 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)

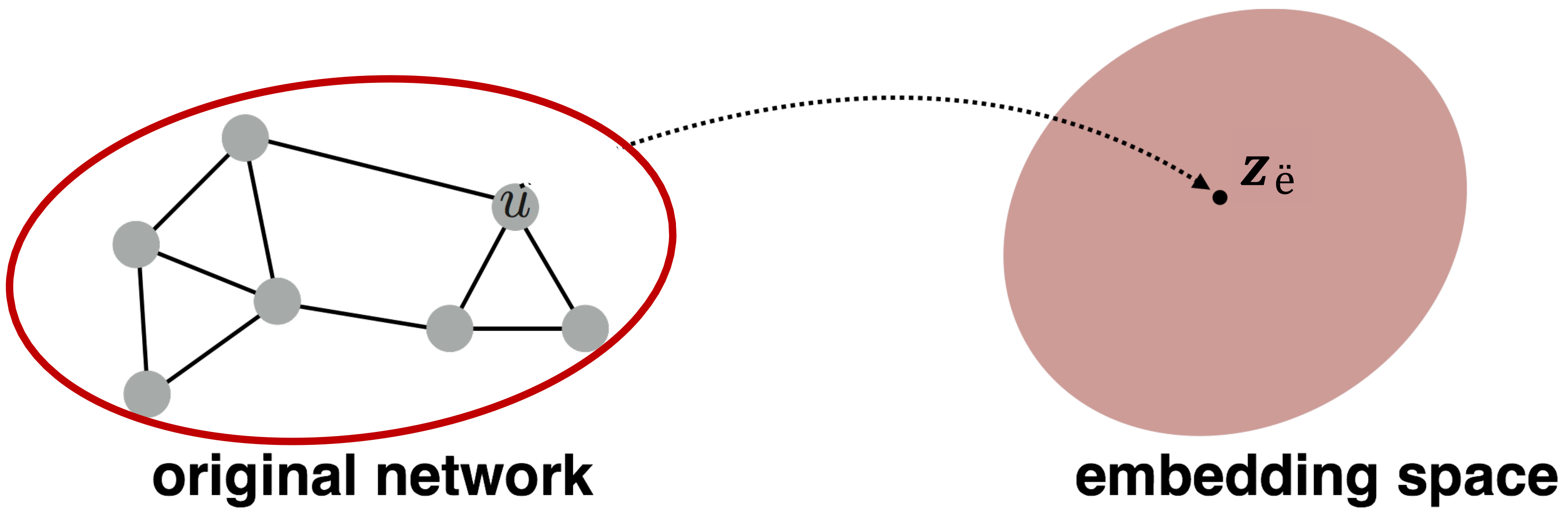
  Sum/Avg: $f(z_i, z_j) = g(z_i + z_j)$

  Distance: $f(z_i, z_j) = g(||z_i - z_j||_2)$

- No one method wins in all cases!
- Must choose node similarity that matches your application.

Embedding Entire Graphs

- **Goal:** Embed an entire graph G



Approach

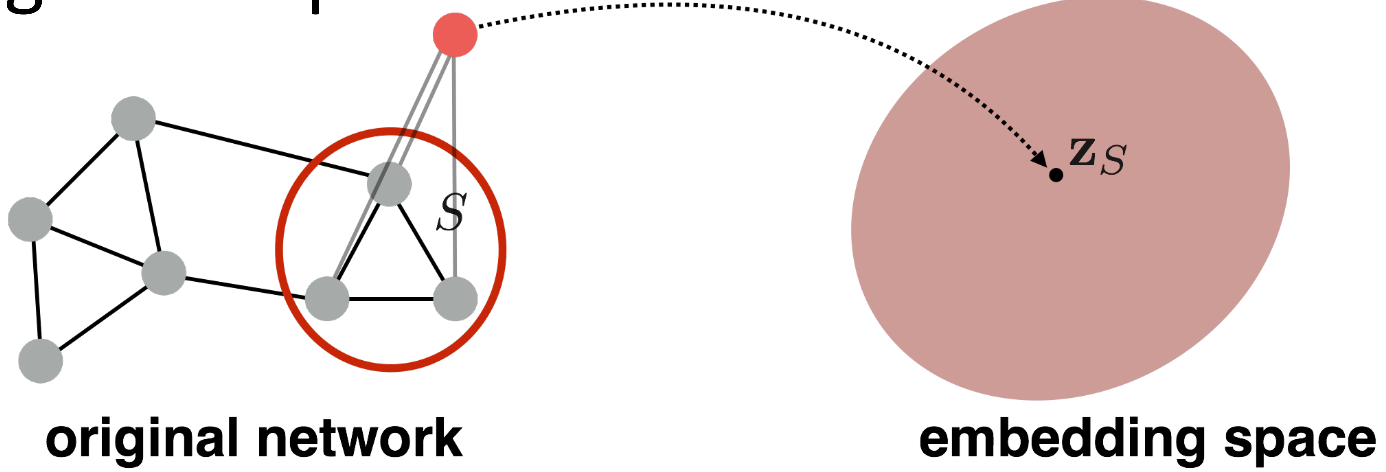
- Run a standard graph embedding
- Then just sum the node embeddings in the subgraph

$$Z_G = \sum_{v \in G} Z_v$$

- Used by Duvenaud et al., 2016 to classify molecules based on their graph structure

Approach

- **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

Recommended Readings

- 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).

Note on Random Walk Embeddings

- This is **unsupervised** way of learning node embeddings
  - We are **not** utilizing node labels or features
  - The goal is to directly estimate a set of coordinates of a node so that some aspect of the network structure is preserved
- These embeddings are **task independent**
  - They are not trained for a specific task but can be used for **any task**

Limitations of Random Walk Embedding

- **Limitations** of random walk based embedding methods:
  - $\boldsymbol{O}(|\boldsymbol{V}|)$ parameters are needed:
    - No sharing of parameters between nodes
    - Every node has its own unique embedding
  - Inherently **"transductive"**:
    - Cannot generate embeddings for nodes that are not seen during training
  - Do **not** incorporate node **features:**
    - Many graphs have features that we can and should leverage