Node Embeddings

Prof. Dr. Gunnar Rätsch, Dr. Rita Kuznetsova
Institute for Machine Learning, Department of Computer Science

Outline for Today

1st slot (45 min):

1. Goals
2. Feature Engineering: Brief Recap
3. Node Embeddings
   a. Deep Walk
   b. Node2Vec
   c. Limitations
   d. Examples
4. Summary & Take-Home Messages

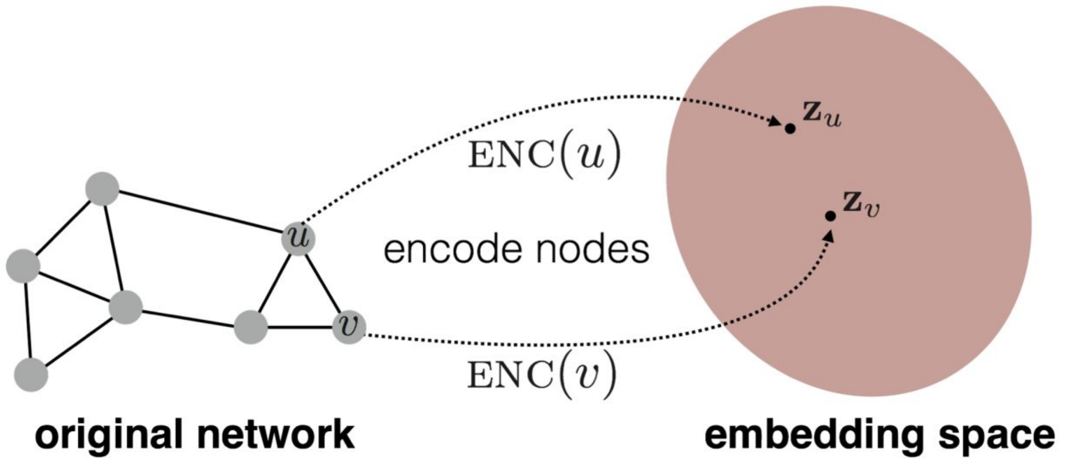
2nd slot (45 min):

Coding Tutorial: PyTorch Geometric

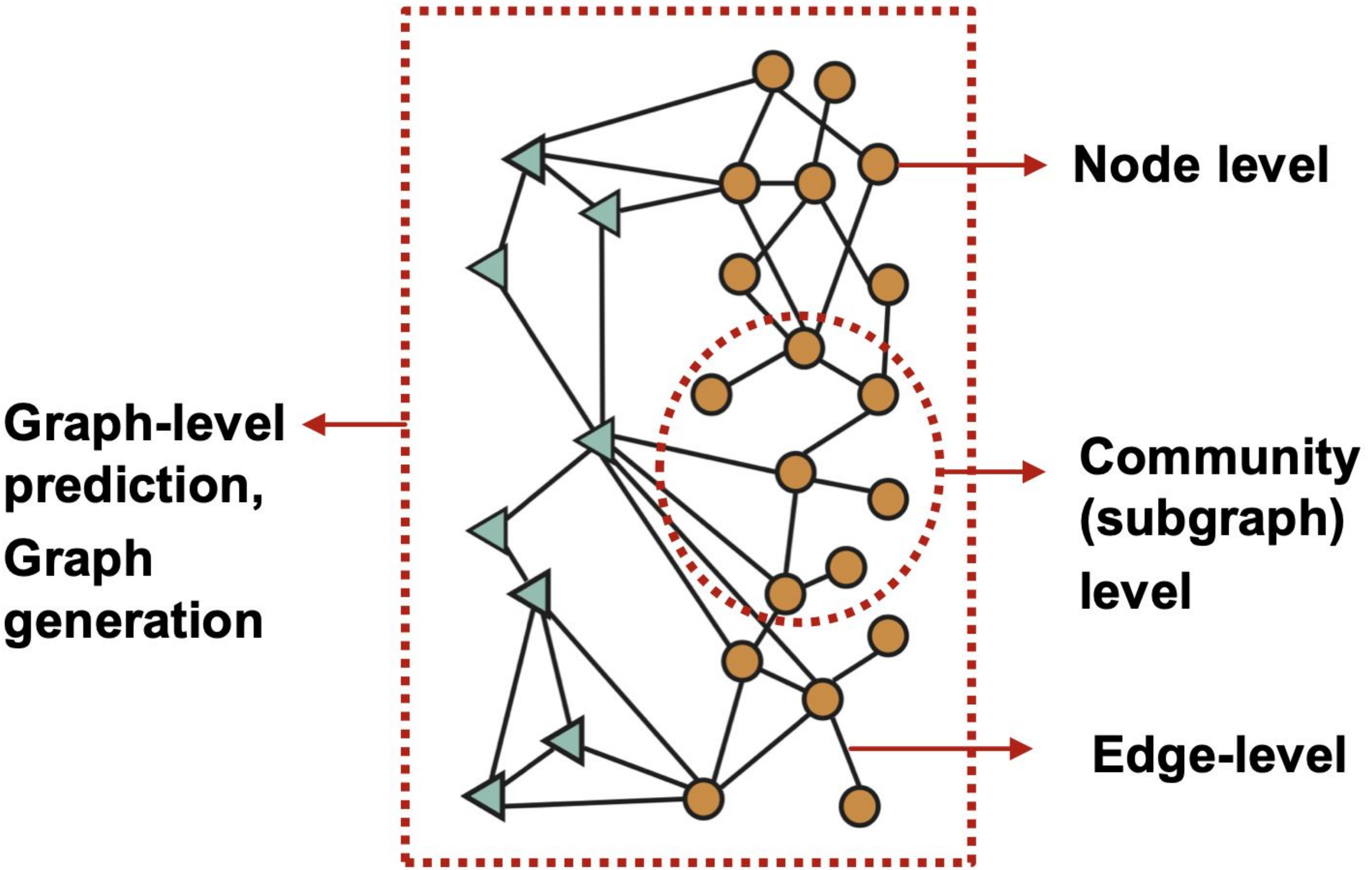
Goals for Today's Lecture

Objective for Today's Lecture:

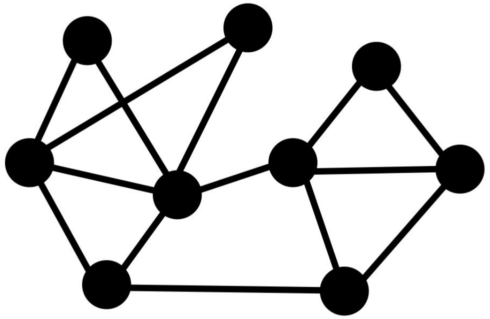
1. Gain an understanding with various methods for encoding nodes as low-dimensional vectors that effectively summarizing their positions within the graph and the structure of their local graph neighborhood.
2. Delve into learning node embeddings through unsupervised approaches.



Different Types of Tasks



Traditional ML Pipeline



Input Graph

Structured Features → Train ML Model → Prediction

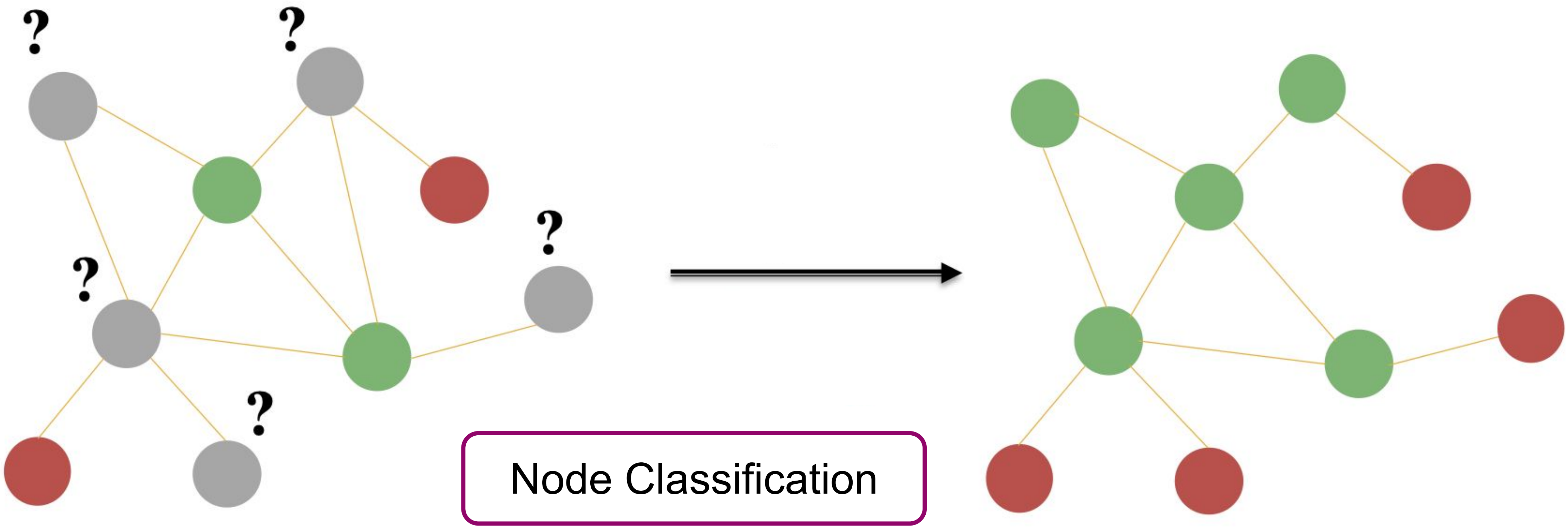
Hand-crafted features: nodes, links, graphs

Traditional ML pipeline uses feature engineering and node attributes.

We give a brief overview of the hand-crafted features for:

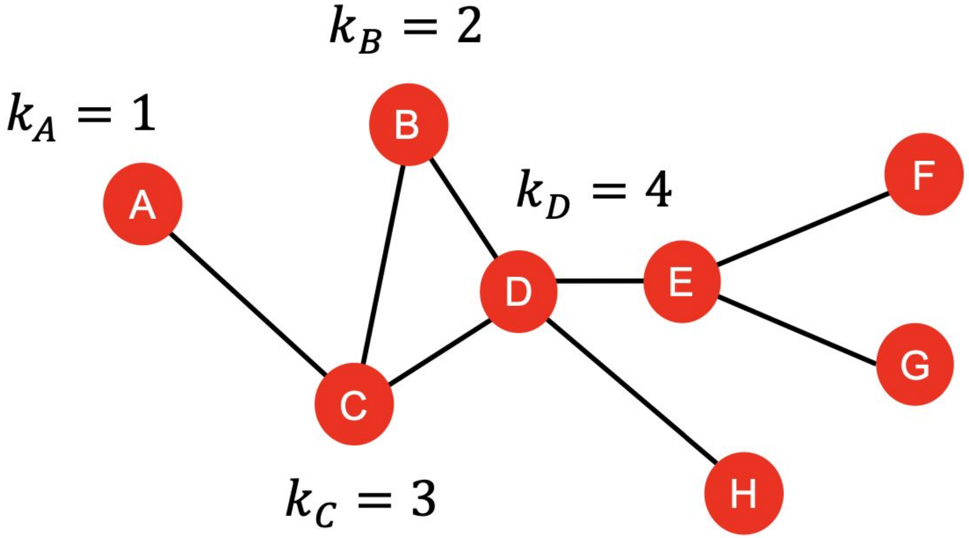
- Nodes,
- Links.

Node-level Tasks



Traditional Features: Node Level Features

Node Degree: the degree k_v of node v is the number of edges (neighboring nodes) the node has.

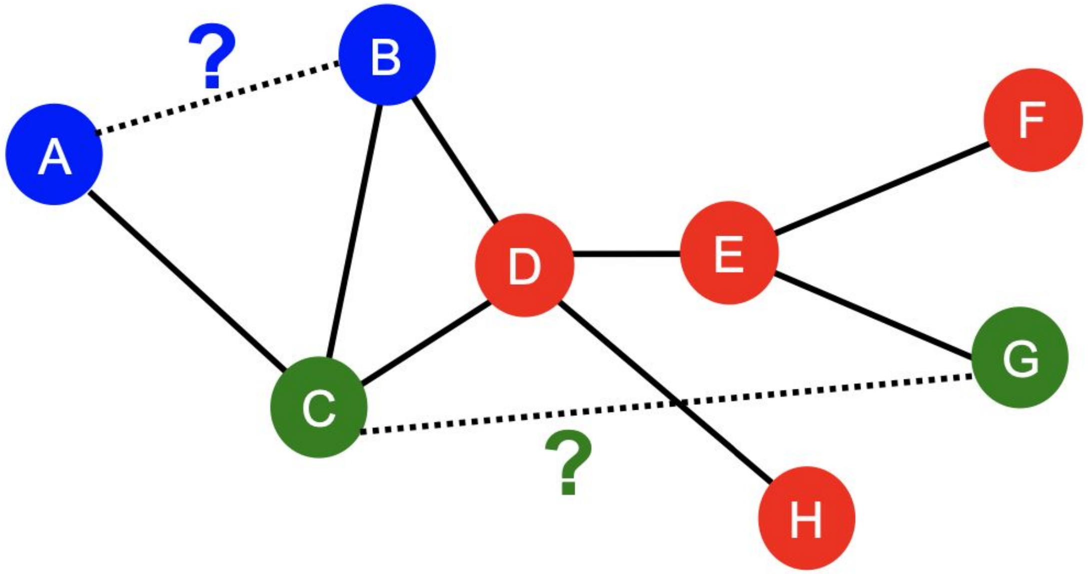


Also:

- Node centrality – takes the node importance in a graph into account,
- Clustering coefficient – how connected v's neighbouring nodes,
- Graphlets – how many pre-defined subgraphs are in the graph.

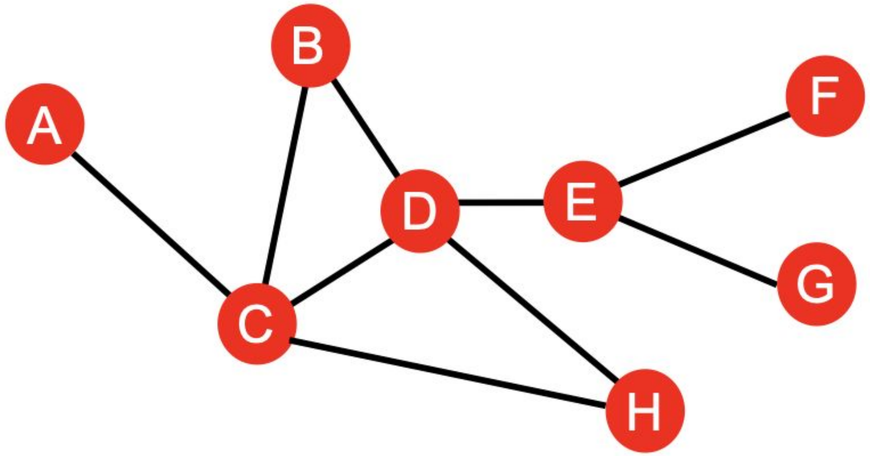
Edge-Level Tasks

- Predict **new links** based on the existing links.
- The key is to design features for **pairs of nodes**.



Traditional Features: Edge Level Features

- Distance-based feature: shortest-path distance between two nodes

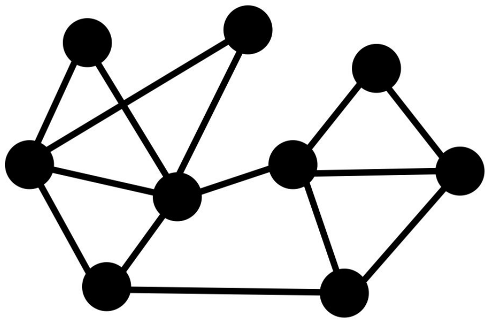


$$S_{BH} = S_{BE} = S_{AB} = 2$$
$$S_{BG} = S_{BF} = 3$$

Also:

- Local neighborhood overlap - # of neighbouring nodes,
- Global neighborhood overlap.

Graph Representation Learning



Input Graph

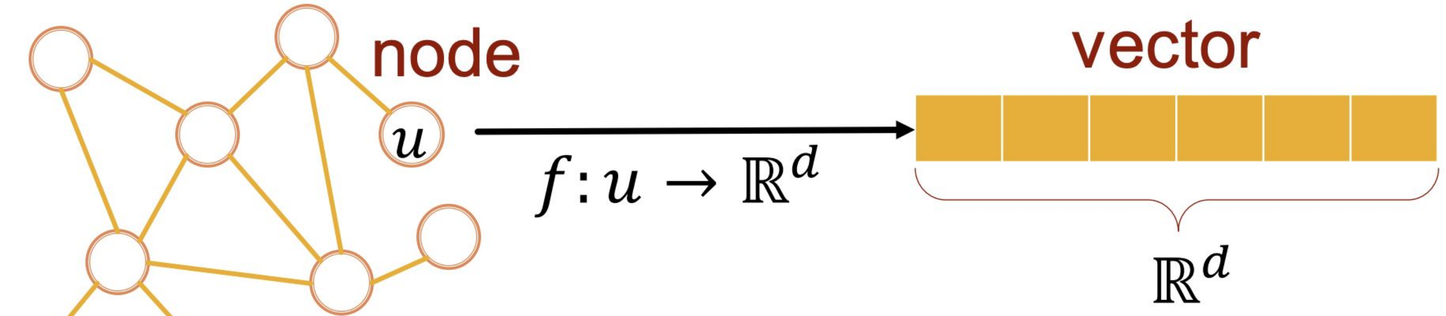
Structured Features → Train ML Model → Prediction

~~Feature Engineering~~

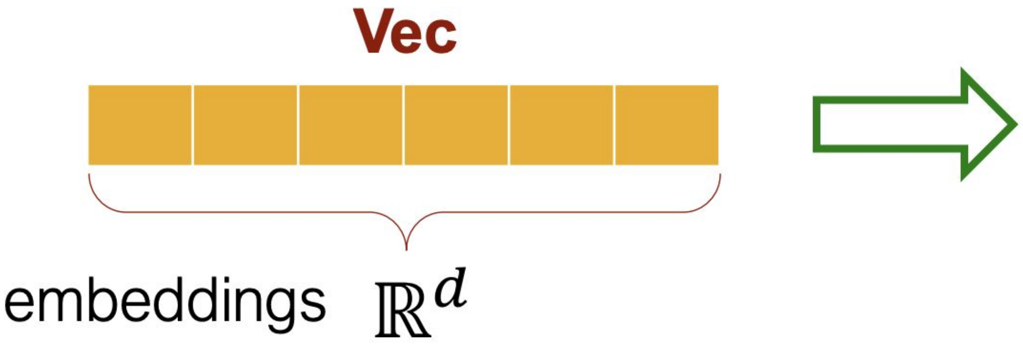
Representation Learning - Automatically learn the features

Graph Representation Learning

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



Feature representation, embedding



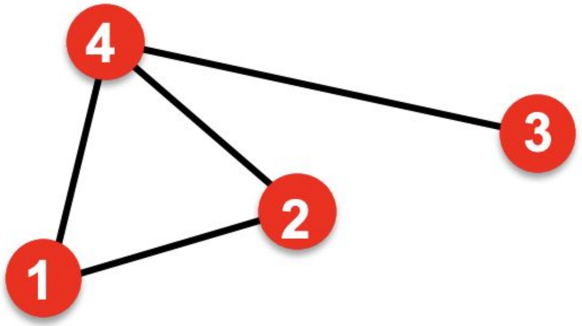
Tasks

- Node classification
- Link prediction
- Graph classification
- Anomalous node detection
- Clustering
- ….

Node Embeddings

Given graph G = (V, E)**:**

- V - nodes,
- A - the adjacency matrix (assume binary),
- E - edges.



V: {1, 2, 3, 4}

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

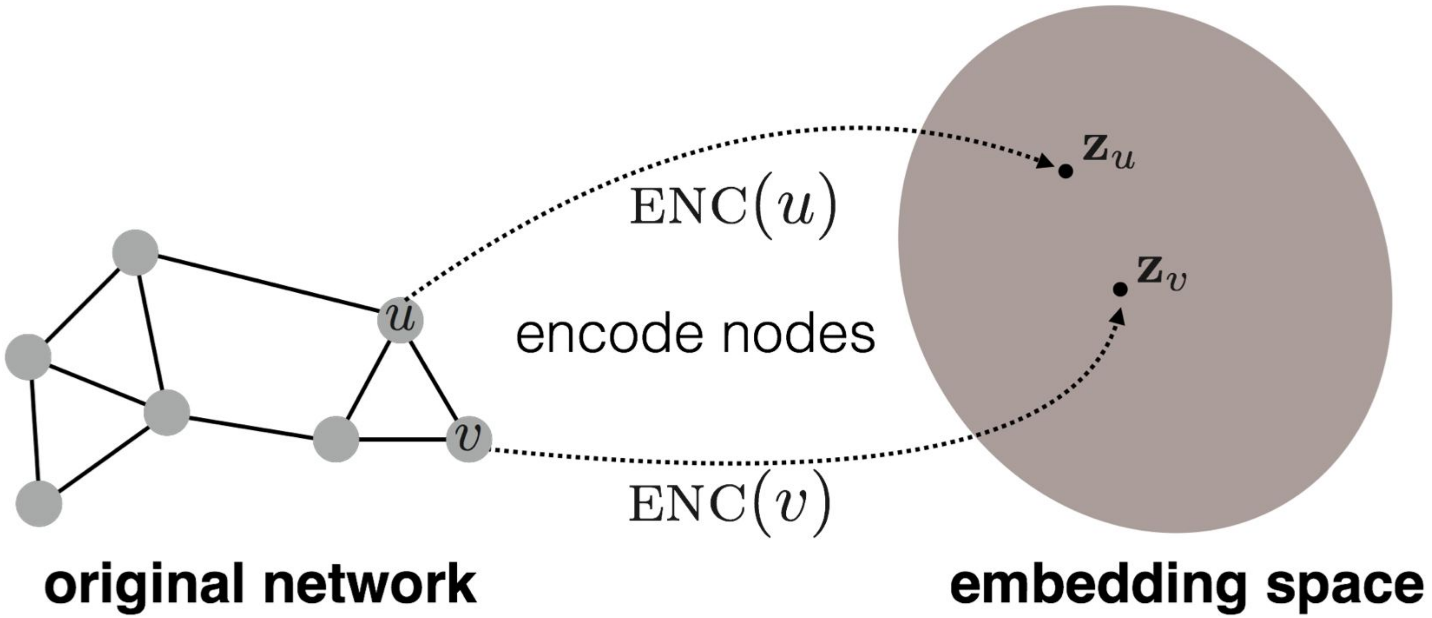
Node Embeddings

Given graph G = (V, E)**:**

- V - nodes,
- E - edges.

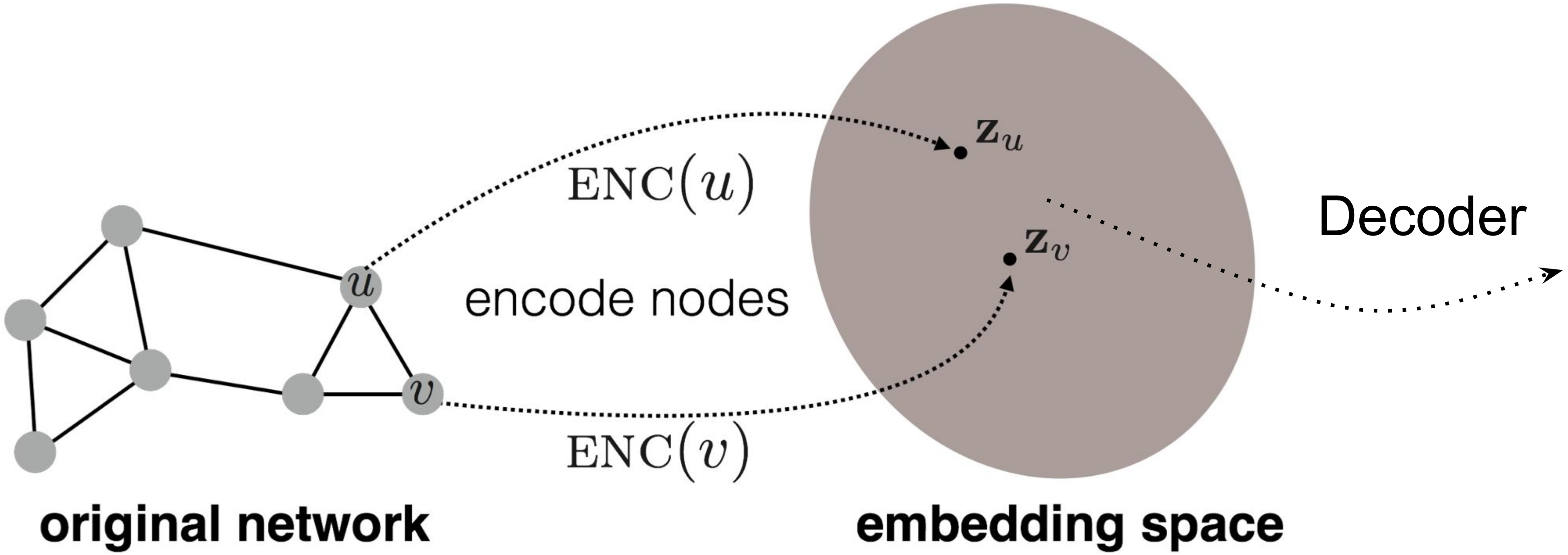
Goal is to encode nodes as low-dimensional vectors that similarity in the latent space correspond to relationships in the graph.

$v,\ u \in V$



Credit: [1,2]

Node Embeddings



Decoder

Similarity score

$\approx \text{similarity}(u, v)$

An Encoder-Decoder Perspective

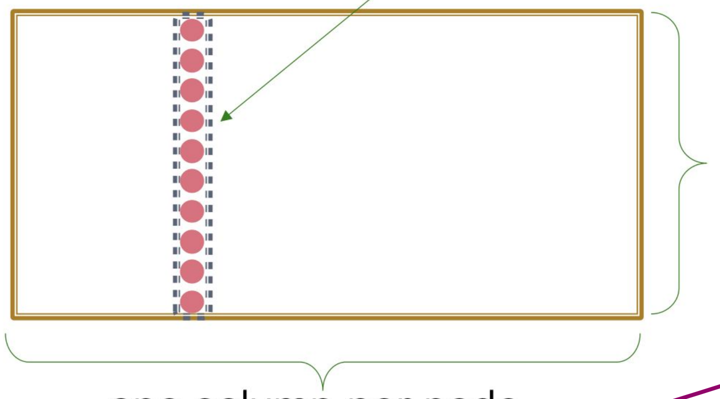
Encoder maps nodes $v \in V$ to vector embeddings:

$$\boxed{ENC(v) = z_v \in \mathbb{R}^d}$$

Simplest encoding approach: Shallow Encoding

$$ENC(v) = z_v = \underbrace{Z}_{\in \mathbb{R}^{d\times|V|}} \cdot \underbrace{v}_{\in \mathbb{I}^{|\mathbb{V}|}}$$

embedding matrix

$\mathbf{Z} =$ 

What we are learning

Indicator vector, all zeros except a one in column indicating node v

An Encoder-Decoder Perspective

Decoder (or pairwise decoder) maps from embeddings to the similarity score:

$$\boxed{\mathrm{DEC} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+}$$

Applying the **decoder** to a pair of embeddings:

$$DEC(ENC(u),\, ENC(v)) \;=\; DEC(z_u,\, z_v) \;\approx\; \text{similarity}(u,\, v)$$

Similarity function: is a graph-based similarity measure between nodes.

Loss function: to be defined.

An Encoder-Decoder Perspective

Method	Decoder	Similarity measure	Loss function
Lap. Eigenmaps	$\|\mathbf{z}_u - \mathbf{z}_v\|_2^2$	general	$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u,v]$
Graph Fact.	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u,v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u,v]\|_2^2$
GraRep	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u,v], ..., \mathbf{A}^k[u,v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u,v]\|_2^2$
HOPE	$\mathbf{z}_u^\top \mathbf{z}_v$	general	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u,v]\|_2^2$
DeepWalk			
node2vec		**In today's lecture**	

Objective: maximize $\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$ for node pairs (u, v) that are **similar.**

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

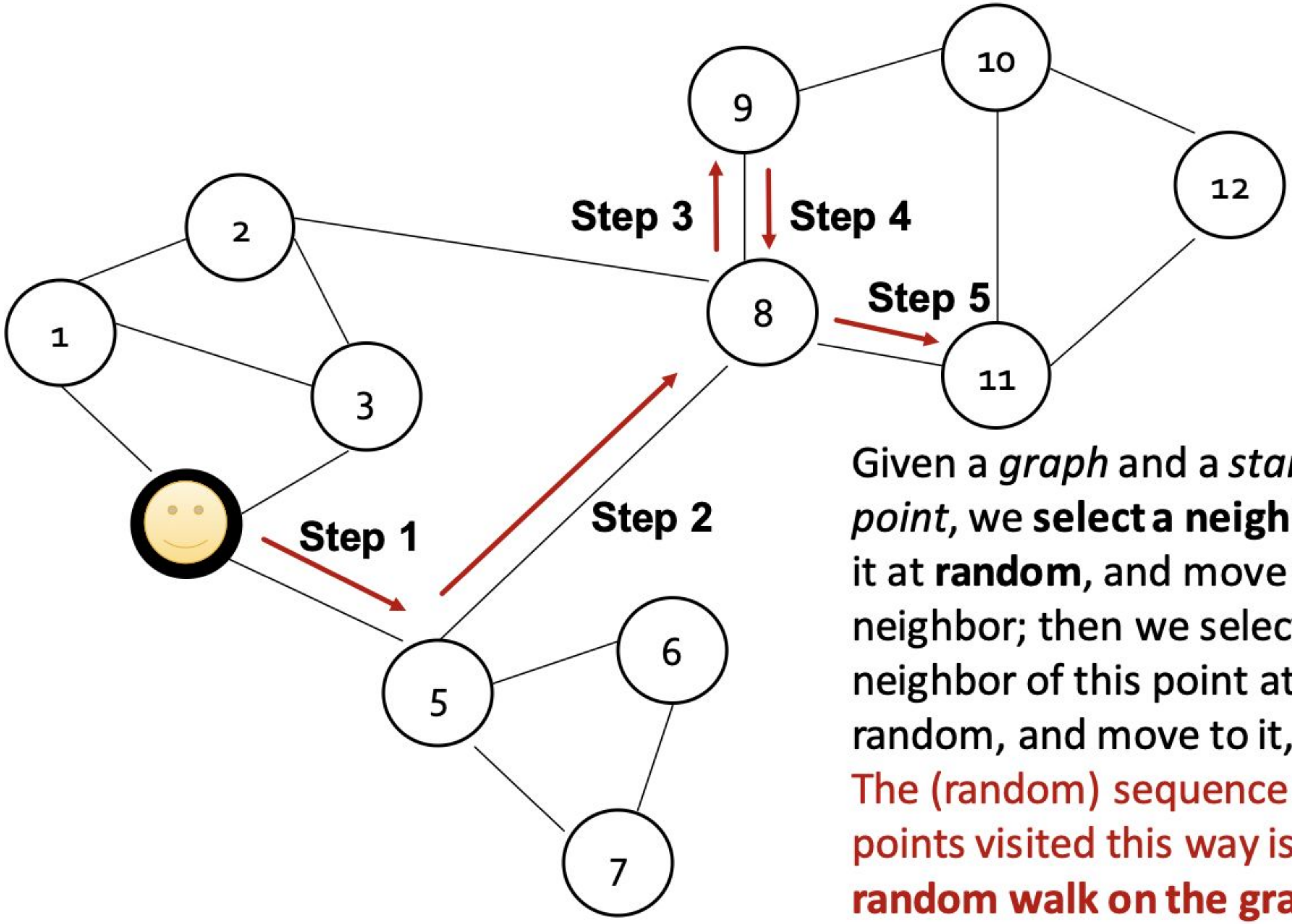
What is “Similar Nodes”?

Should two nodes have a similar embedding if they…

- are linked?
- share neighbors?
- have similar “structural roles”?

In today’s lecture: we learn node similarity definition that uses **random walks**, and how to optimize embeddings for such a similarity measure.

Random Walk Approaches



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**.

Framework Summary

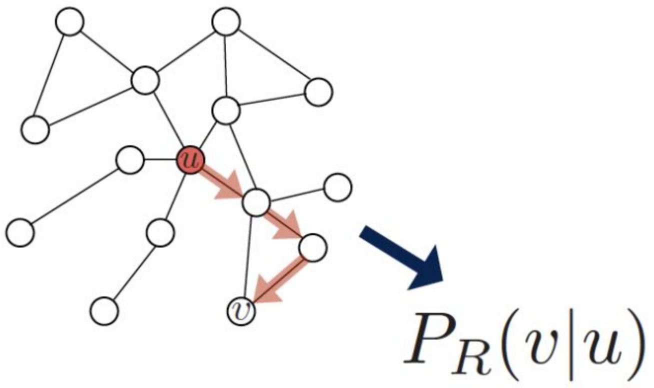
This is **unsupervised/self-supervised** way of learning node embeddings:

- We are not utilizing node labels,
- We are not utilizing node features,
- The goal is to directly estimate a set of coordinates (the embedding) of a node so that some aspect of the network structure is preserved.

Random Walk Embeddings

Two nodes have similar embeddings if they tend to co-occur on random walks over the graph.

We need to estimate probability of visiting node v on a random walk starting from node u using some random walk strategy ***R.***



Why Random Walk?

1. **Expressivity**: Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information.

   Idea: if random walk starting from node u visits v, u and v are similar (high-order multi-hop information).

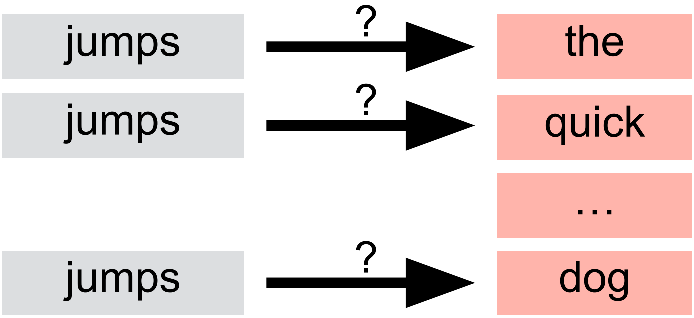
2. **Efficiency**: Do not need to consider all node pairs when training; only need to consider pairs that co-occur on random walks.

Connection with word2vec

The basic idea is to **predict** a missing word from a <u>context</u>. What is **context**?

Example: "the quick red fox jumps over the lazy brown dog"

Skip-gram - the 'context' is *each* word from the surrounding *central* word. Based on the *central* word we predict each word from this surrounding.



Positive pairs: (context word, central word).

Random Walk Embeddings

Our goal is to learn a mapping $f(u) = z_u \in \mathbb{R}^d$ that preserves similarity, i.e. embeddings of the nodes which are nearby in the network are close.

Nearby nodes: $N_R(u)$ neighbourhood of u obtained by some random walk strategy R.

Log-likelihood objective:

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_\mathrm{R}(u) \mid \mathbf{z}_u)$$

Given node u, we want to learn feature representations that are predictive of the nodes in its random walk neighborhood.

Random Walk Optimization

Run short fixed-length random walks starting from each node $u \in V$ using strategy R.

For each node u collect $N_R(u)$.

Optimize: Given node u, predict its neighbors.

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$

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

Optimize embeddings to maximize the likelihood of random walk co-occurrences.

Random Walk Optimization

Run short fixed-length random walks starting from each node $u \in V$ using strategy R.

For each node u collect $N_R(u)$.

Optimize:
Given node u, predict its neighbors.

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$

Optimize embeddings to maximize the likelihood of random walk co-occurrences.

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

Parameterize using softmax: $P(v|\mathbf{z}_u) = \dfrac{\exp(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_n)}$

Random Walk Optimization

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

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.

Random Walk Optimization

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

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 embeddings that minimize $\mathcal{L}$.

Negative Sampling

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

Very expensive!

Solution: Negative sampling

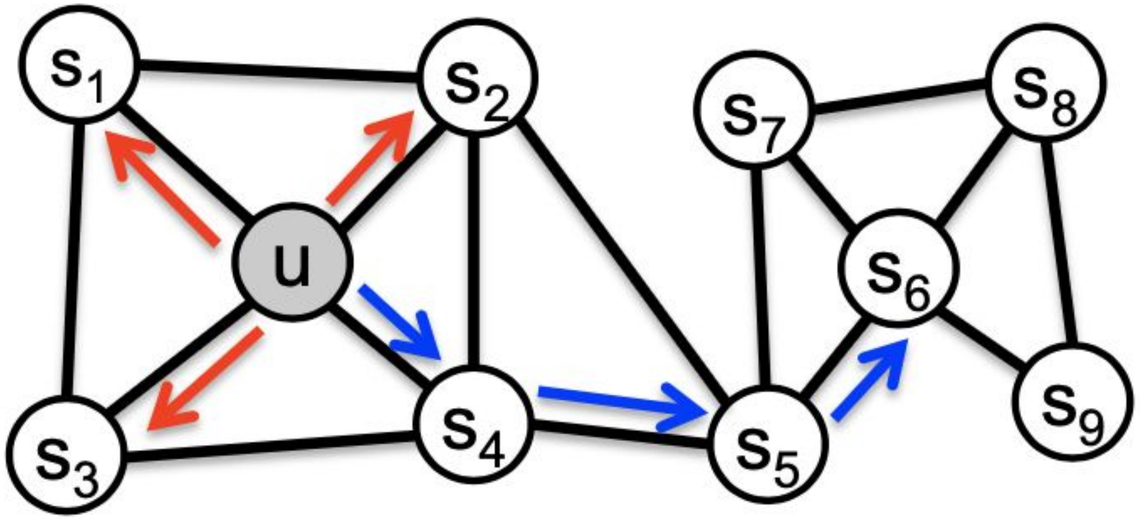
$$-\log \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)} \approx \log\Big(\sigma\big(\mathbf{z}_u^\mathrm{T} \mathbf{z}_v\big)\Big) - \sum_{i=1}^{k} \log\Big(\sigma\big(\mathbf{z}_u^\mathrm{T} \mathbf{z}_{n_i}\big)\Big)$$

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

In practice k = 5 - 20.

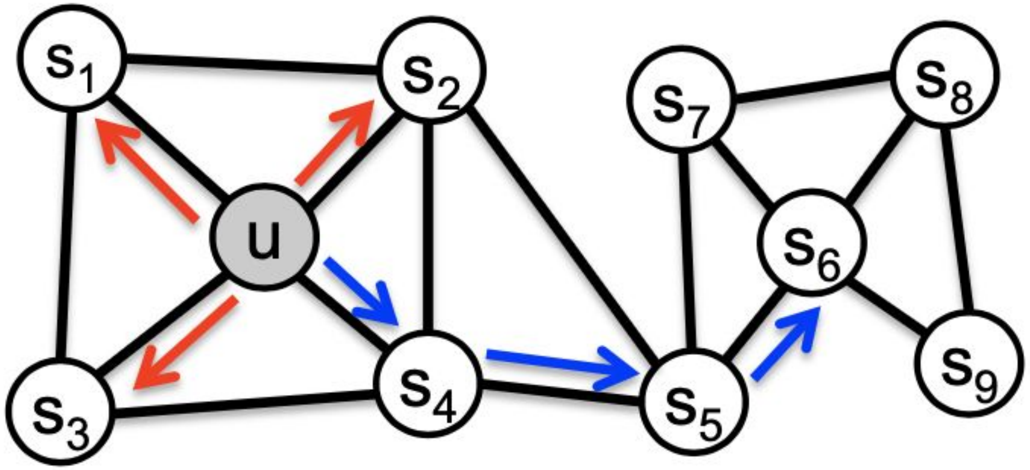
How to Define the Random Walk Strategy *R*?

DeepWalk approach: run fixed-length, unbiased random walks starting from each node.



Biased Walks: node2vec

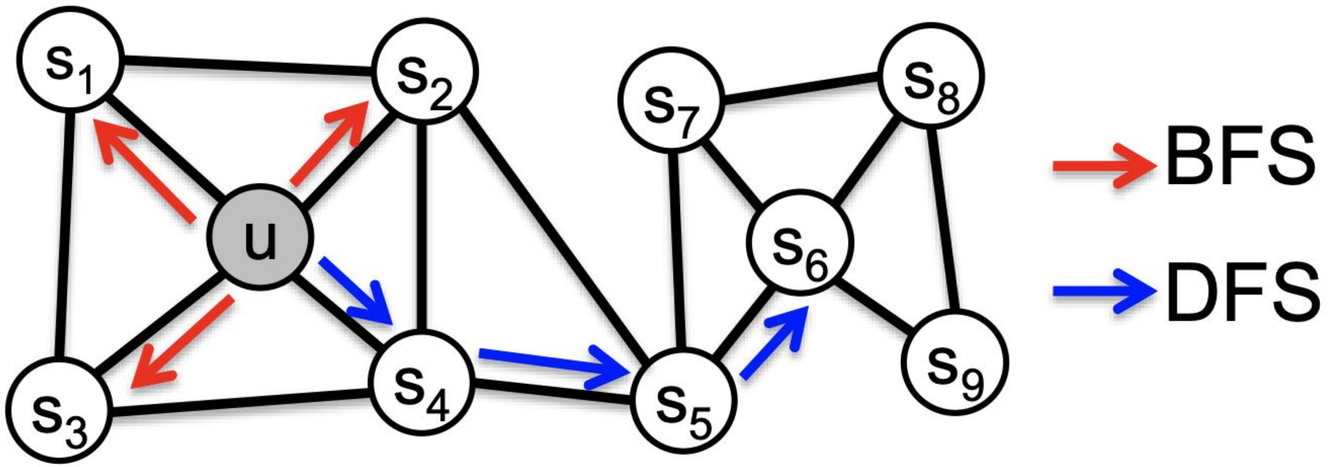
Goal: Develop biased 2nd order random walk R to generate network neighborhood $N_R(u)$ of node u.



Idea: use flexible, biased random walks that can trade off between **local** and **global** views of network.

Biased Walks: node2vec

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



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 & DFS

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

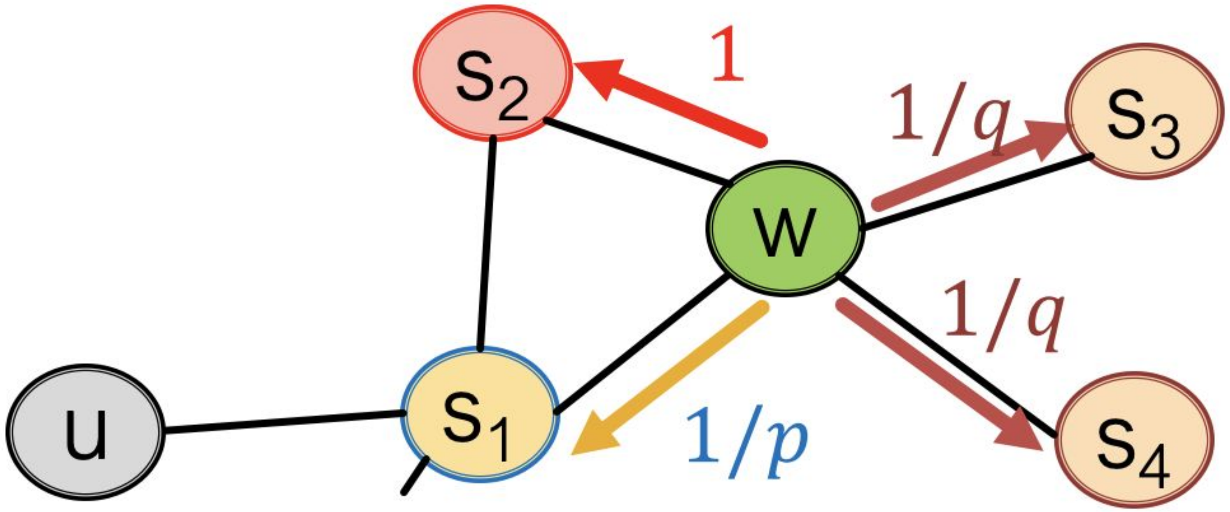
Two hyperparameters:

1. Return parameter ***p***:
   a. return back to the previous node
2. In-out parameter $\boldsymbol{q}$:
   a. Moving outwards (DFS) vs. inwards (BFS),
   b. Intuitively, q is the “ratio” of BFS vs. DFS.

p and $\boldsymbol{q}$ ranges: positive real numbers

Biased Walks: node2vec

Walker came over edge $(\mathbf{s1}, \mathbf{w})$ and is at $\mathbf{w}$. Where to go next?

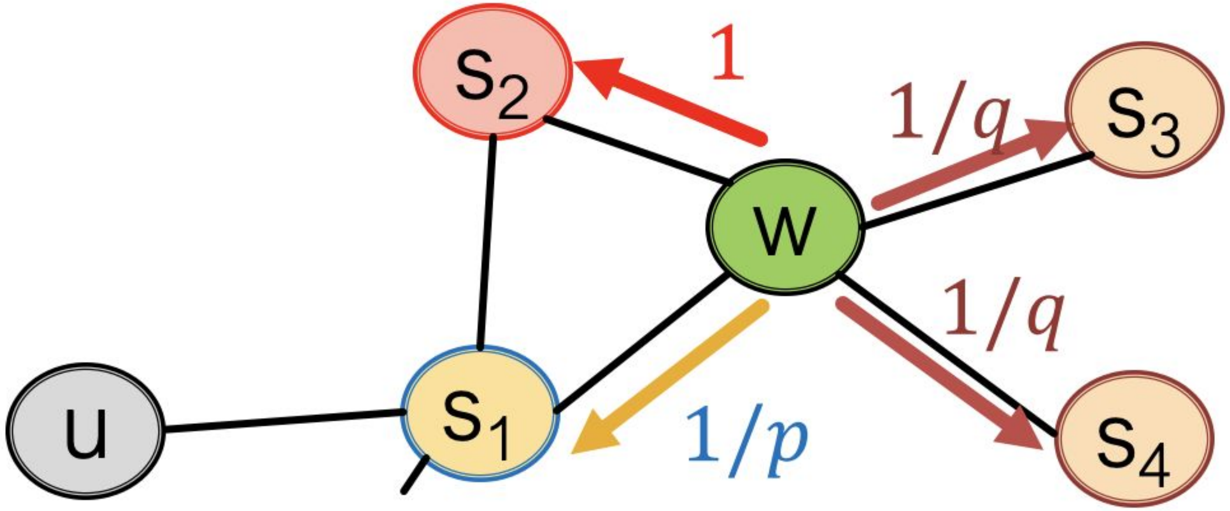


$1/p$, $1/q$, 1 are unnormalized probabilities

- p – return parameter
- q – "walk away" parameter

Biased Walks: node2vec

Walker came over edge $(\mathbf{s1}, \mathbf{w})$ and is at $\mathbf{w}$. 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

- BFS-like walk:
  - Low value of p (keep the walk "local" close to the starting point);
  - High value of q
- DFS-like walk:
  - Low value of q
  - High value of p

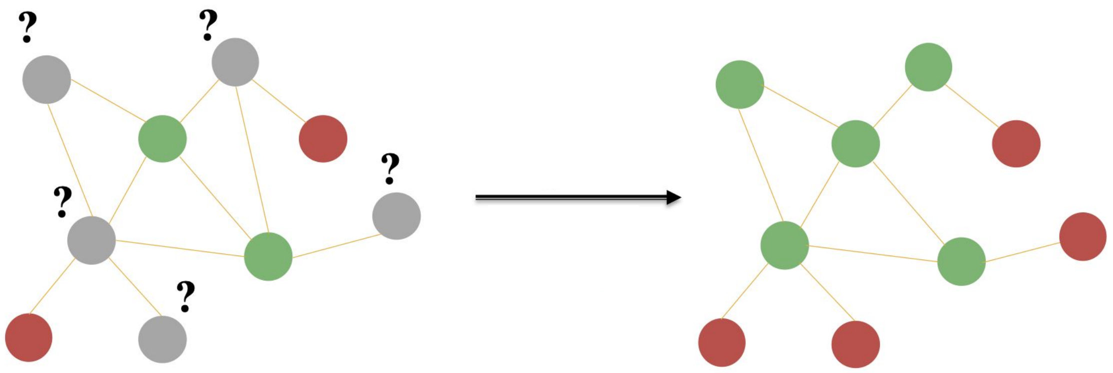
Training: node2vec & Deep Walk

1. Compute random walk probabilities (node2vec)
2. Simulate r random walks of length l starting from each node u (node2vec/Deep Walk)
3. Optimize the node2vec objective using SGD (node2vec/Deep Walk)

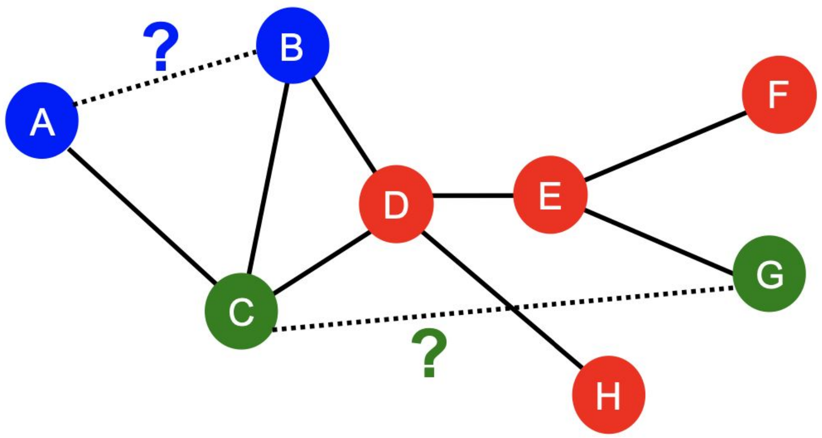
Summary

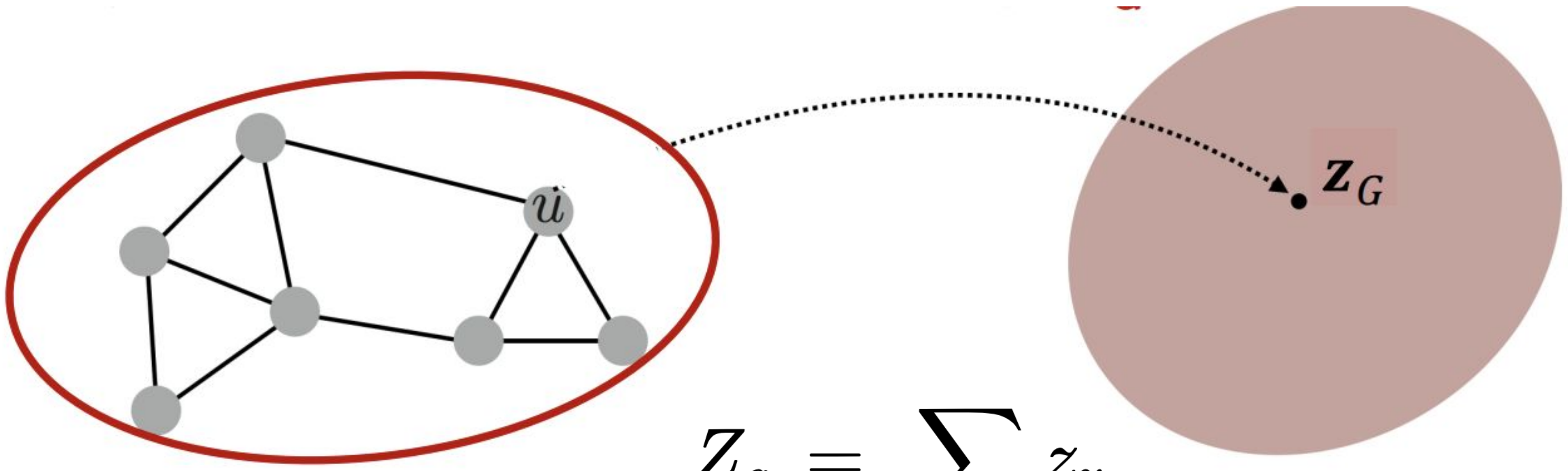
1. Node2vec / Deep Walk methods: embed nodes so that distances in embedding space reflect node similarities in the original network.
2. Notion of node similarity: Random walk methods.
3. Random walk methods serve a solid baseline for node classification / link prediction.
4. No one method wins in all cases.

How we use node embeddings



For each pair (u,v): $z_u \circ z_v$





$$Z_g = \sum_{v \in V} z_v$$

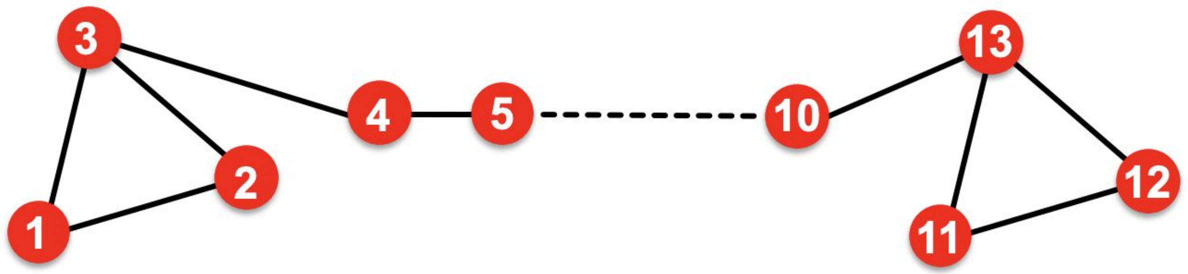
(or average)

Credit: [1]

Limitations of node2vec / Deep Walk

1. Can not obtain embeddings for unseed nodes;

2. Can not capture well the structural similarity;



3. Can not utilize directly node, edge and graph features;

4. Sensitive to the walk length, p/q choice.

Other Random Walk Ideas

1. metapath2vec
2. Watch Your Step: Learning Node Embeddings via Graph Attention
3. LINE
4. struc2vec
5. HARP

Take-Home Messages

1. We discussed graph representation learning - how to learn node embeddings for downstream tasks, without feature engineering.
2. We can use these embeddings for all level tasks: node/link/graph
3. Encoder-decoder framework - powerful setup:
   a. Encoder: embedding lookup
   b. Decoder: predict score based on embedding to match node similarity
4. Node similarity measure: (biased) random walk (node2vec/Deep Walk)

BIOMEDICAL
INFORMATICS

Slides & Image Credits

1. CS224W: Machine Learning with Graphs
2. Graph Representation Learning Book
3. DeepWalk: Online Learning of Social Representations
4. node2vec: Scalable Feature Learning for Networks
5. Efficient Estimation of Word Representations in Vector Space
6. Distributed Representations of Words and Phrases and their Compositionality
7. word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method

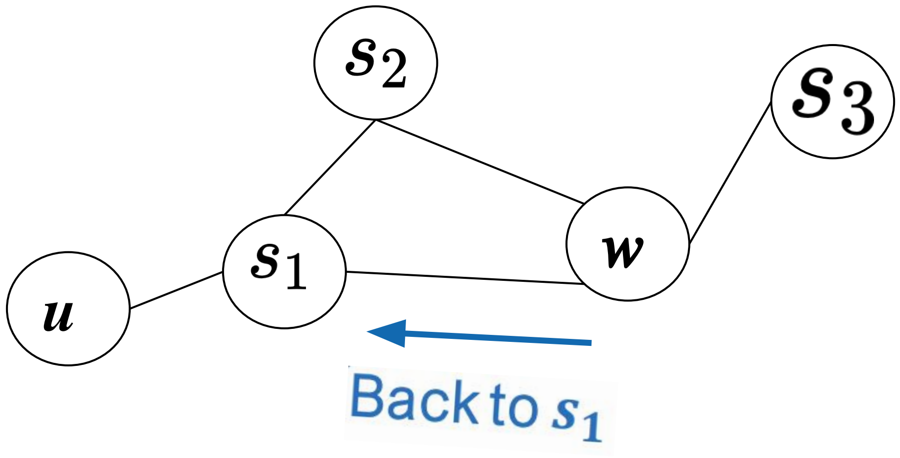
Interpolating BFS & DFS

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

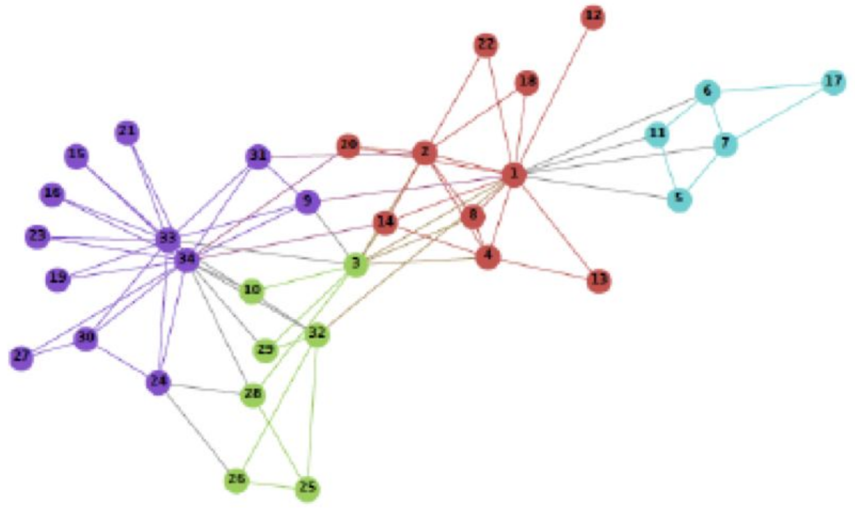
Two parameters:

1. Return parameter ***p***: return back to the previous node

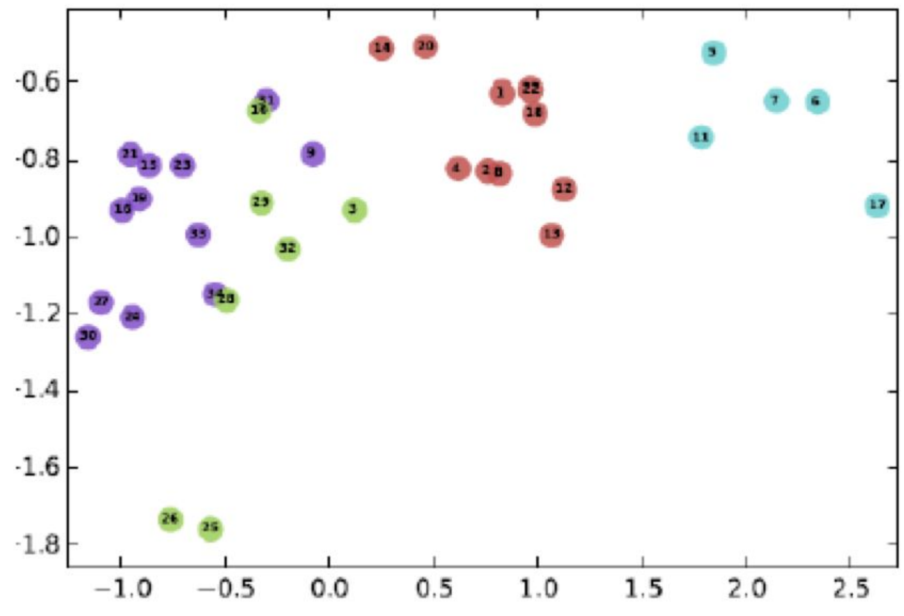
   RW just traversed edge (s_1, w) and is now at w:



Node Embeddings: Zachary’s Karate Club network



Input



Output

Various node classification tasks

- To learn **social representations** of graph nodes – latent features of the vertices that capture neighborhood similarity and community membership;
- These **latent representations** encode social relations in a continuous vector space.