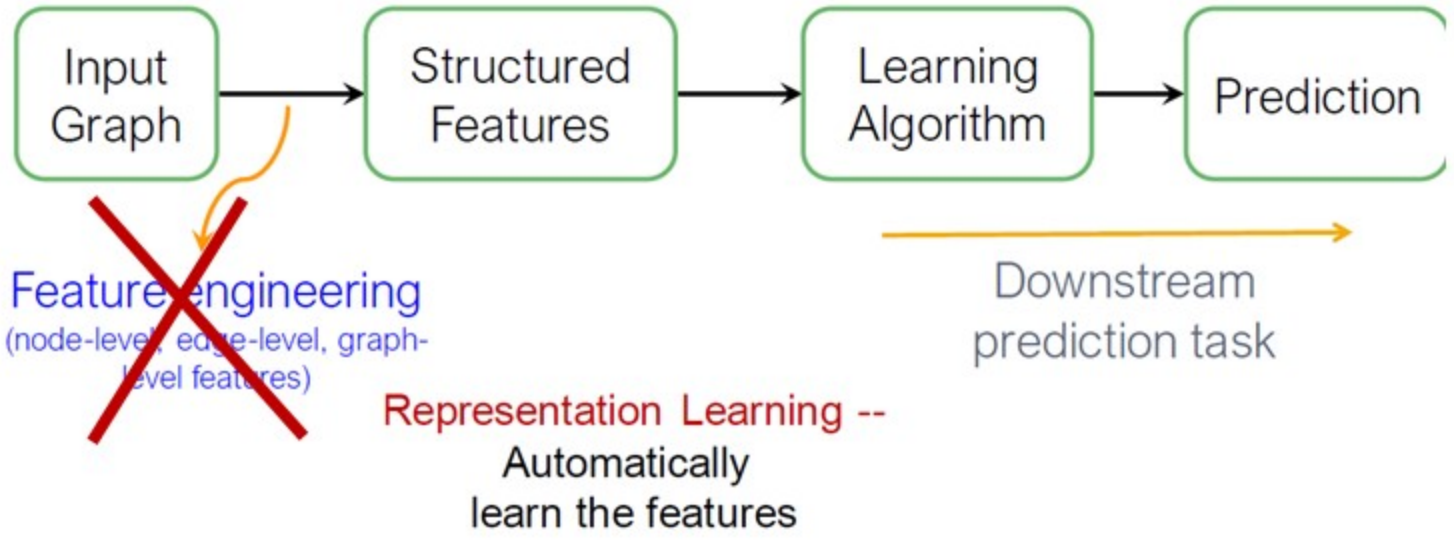
Node Embeddings

Node embeddings

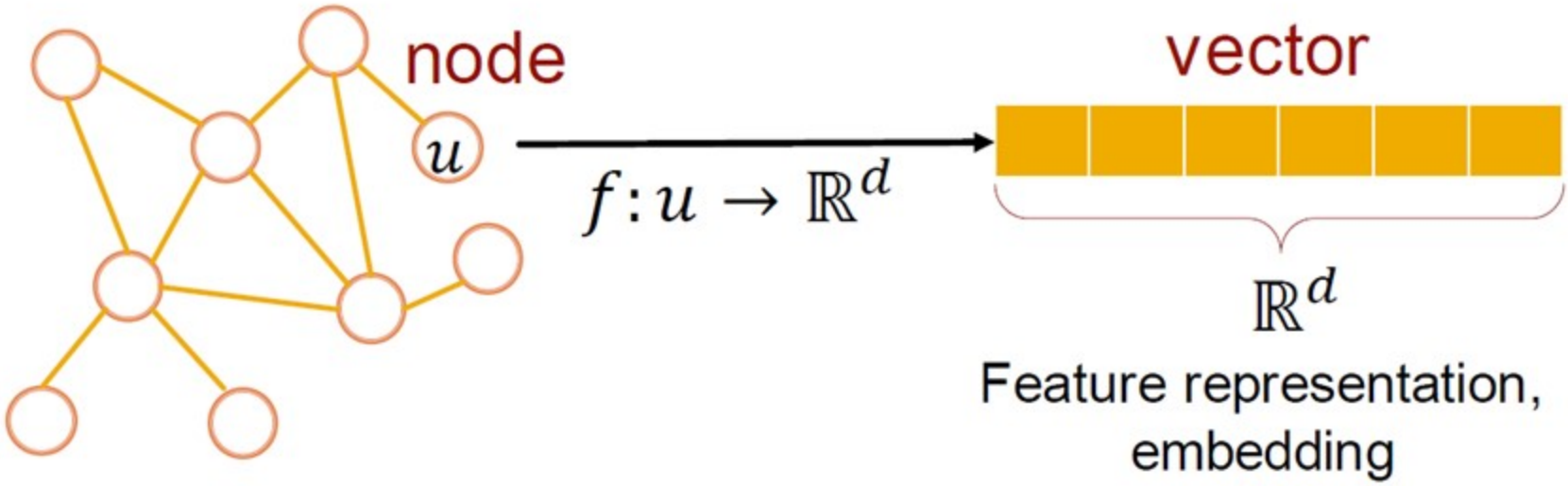
- Traditional ML for Graphs

  Given an input graph, extract node, link and graph-level features, learn a model (SVM, neural network, etc.) that maps features to labels.



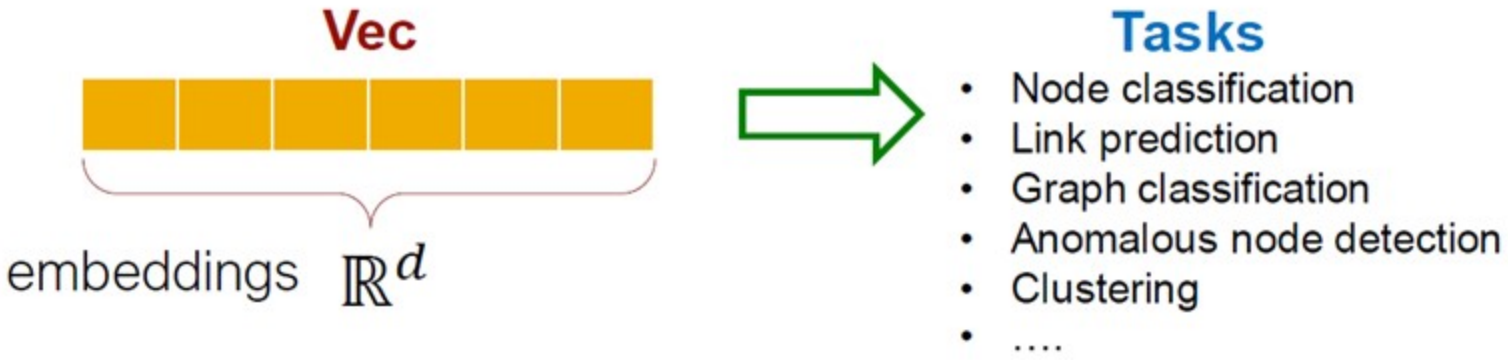
Node embeddings

- Graph representation learning
  - Efficient task-independent feature learning for machine learning with graphs



Node embeddings

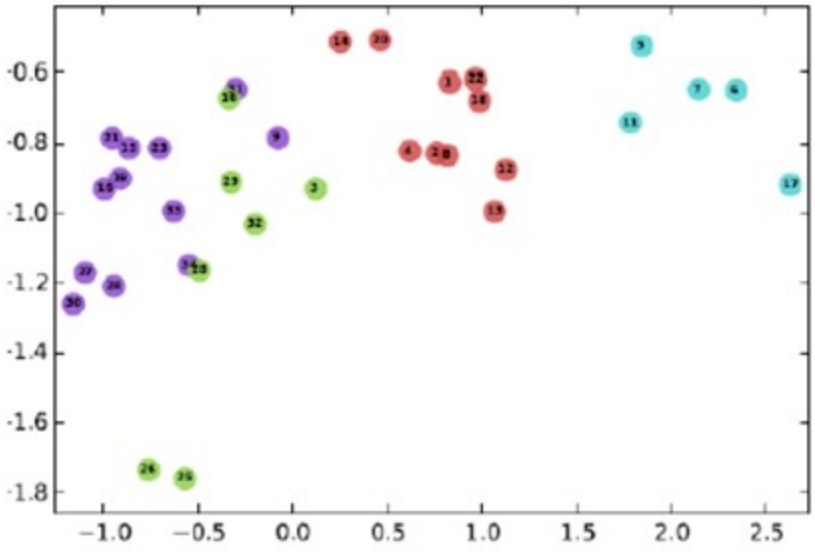
- Why embedding?
  - **Task: Map nodes into an embedding space**
    - Similarity of embeddings between nodes indicates their similarity in the network. For example:
      - Both nodes are close to each other (connected by an edge)
    - Encode network information
    - Potentially used for many downstream predictions



Node embeddings

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

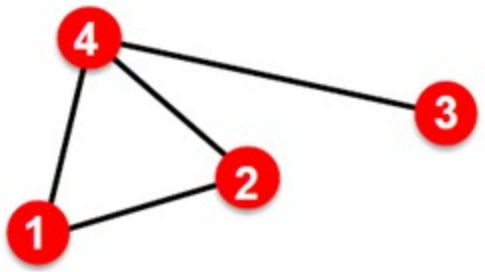
Input



Output

Node embeddings

- Setup
  - **Assume we have a graph G:**
    - V is the vertex set.
    - A is the adjacency matrix (assume binary).
    - **For simplicity: No node features or extra information is used**



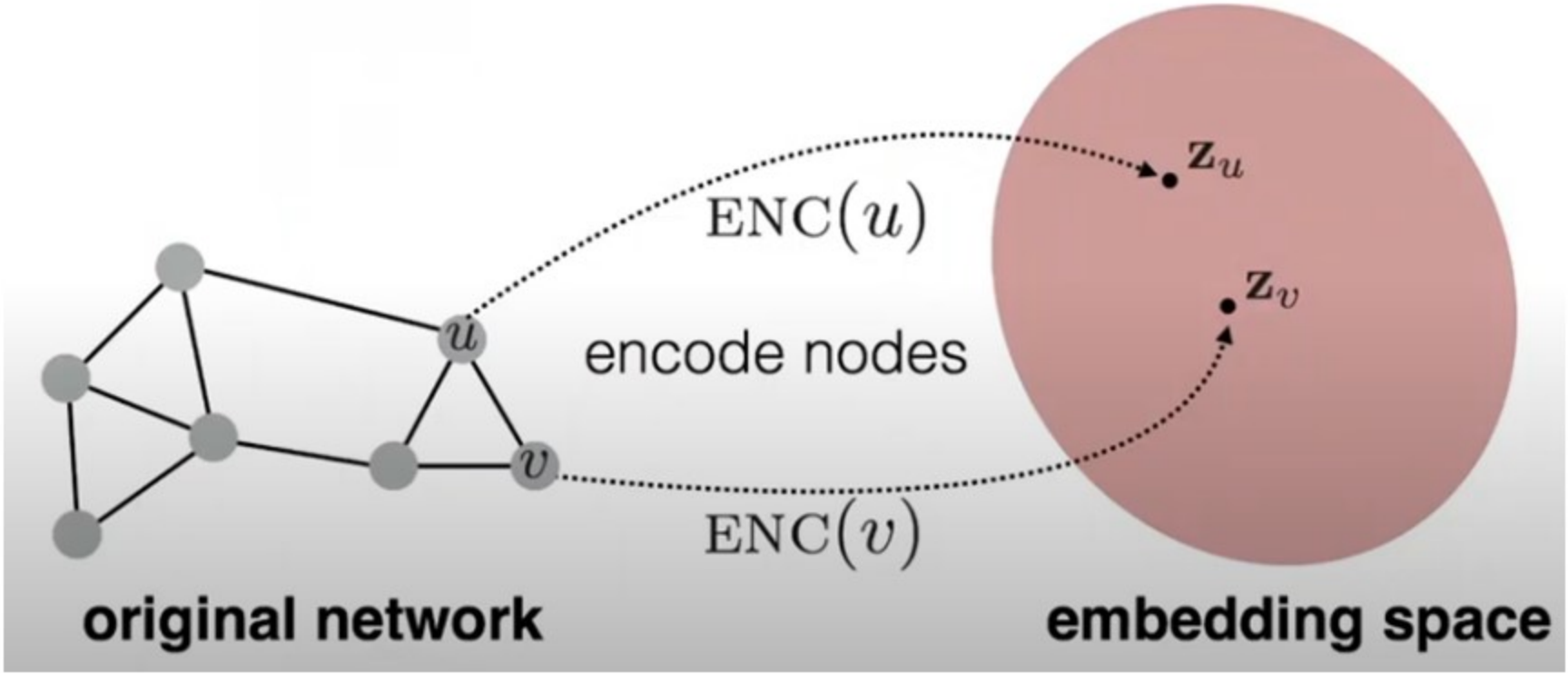
V: {1, 2, 3, 4}

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

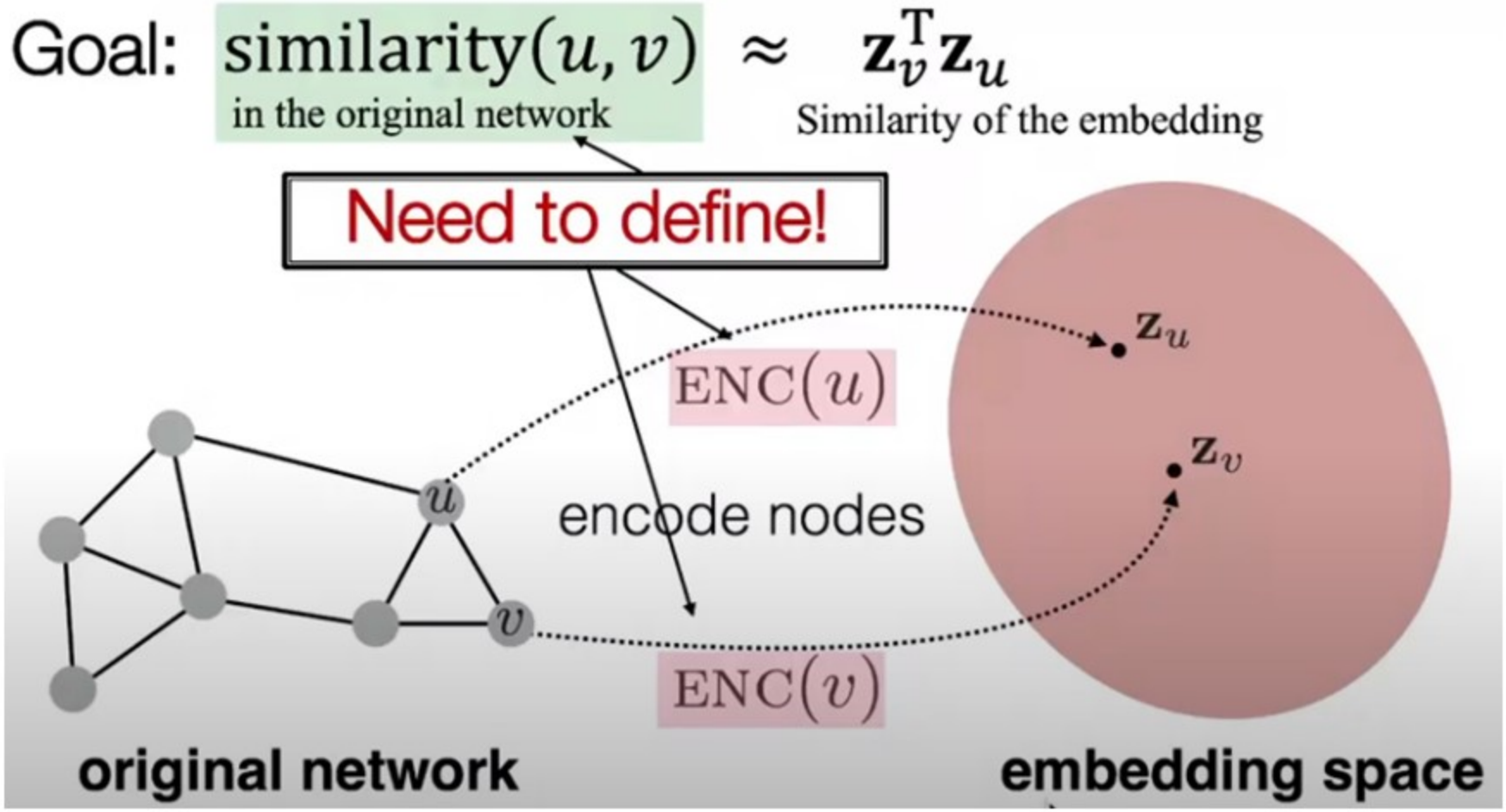
Node embeddings

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

in the original network — Similarity of the embedding



Node embeddings



Node embeddings

- Encoder/Decoder

1. **Encoder** maps from nodes to embeddings
2. **Define a node similarity function** (i.e., a measure of similarity in the original network)
3. **Decoder DEC** maps from embeddings to the similarity score
4. **Optimize the parameters of the encoder so that:**

$\mathrm{DEC}(\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u)$

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

Node embeddings

- Encoder/Decoder
  - **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 the relationships in vector space map to the relationships in the original network

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

Decoder

Similarity of u and v in the original network

dot product between node embeddings

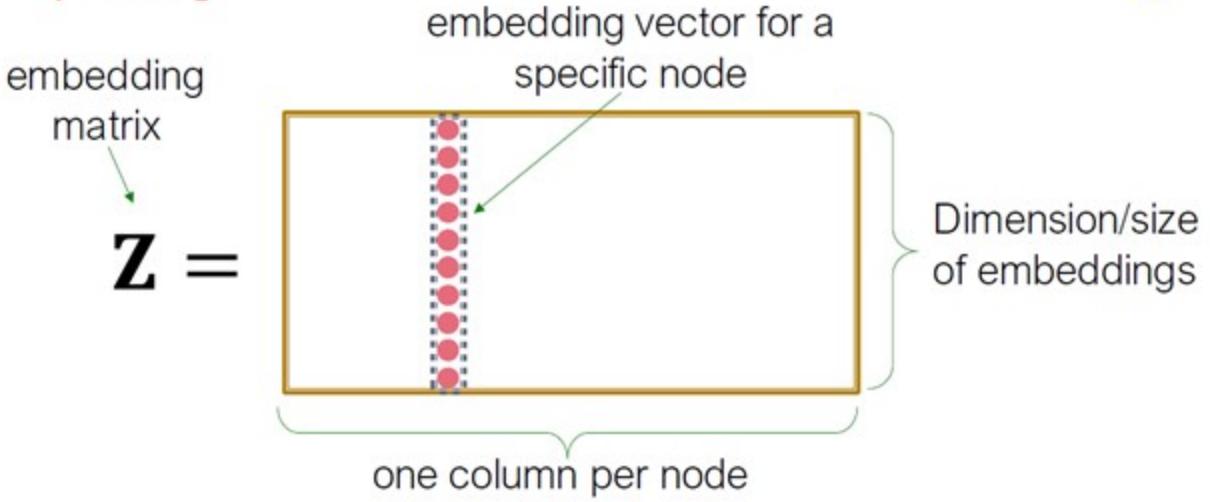
Node embeddings

- Shallow Encoding

  Simplest encoding approach: **Encoder is just an embedding-lookup** $\mathrm{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is a node embedding [what we learn / optimize]

$v \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node *v*



Node embeddings

- Shallow Encoding
  - **Limitations** of shallow embedding methods:
    - **$O(|V|)$ 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**:
      - Nodes in many graphs have features that we can and should leverage

Node embeddings

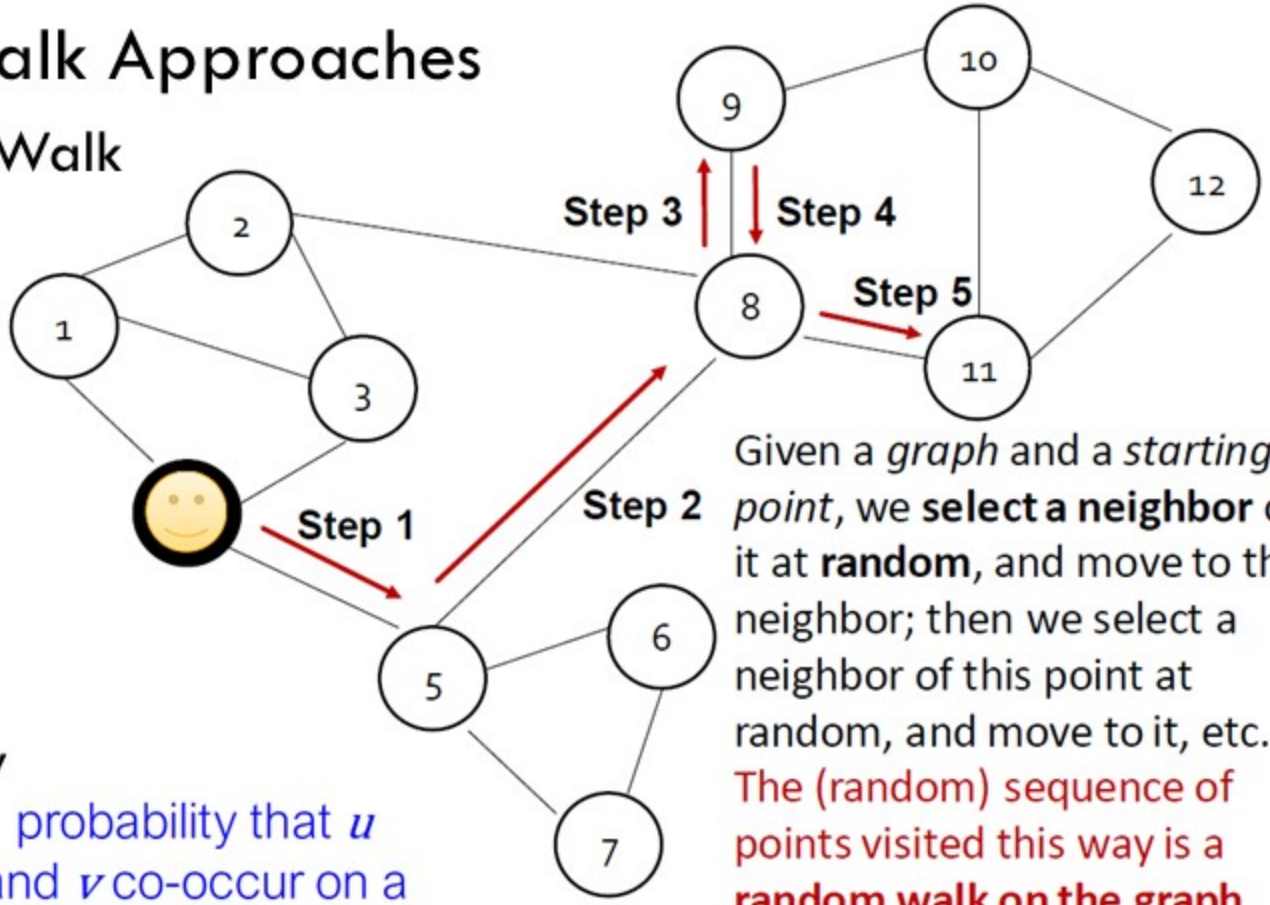
- Node similarity
  - Key choice of methods is **how they define node similarity.**
  - Should two nodes have a similar embedding if they…
    - are linked?
    - share neighbors?
    - have similar "structural roles"?
  - We will now learn node similarity definition that uses **random walks**, and how to optimize embeddings for such a similarity measure.

Node embeddings

- 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 (i.e., the embedding) of a node so that some aspect of the network structure (captured by DEC) is preserved.
- These embeddings are **task independent**
  - They are not trained for a specific task but can be used for any task.

Node embeddings

- Random Walk Approaches
  - Random Walk



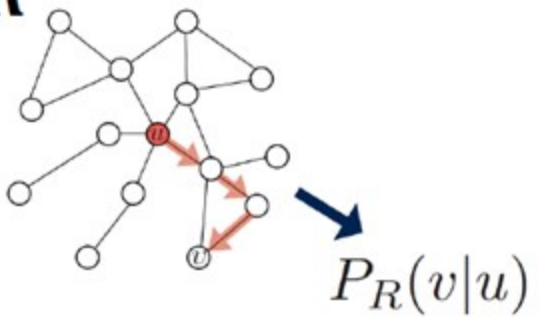
Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points visited this way is a **random walk on the graph**.

  - Similarity

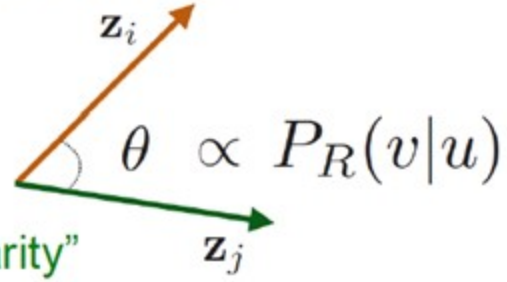
$\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v \approx$ probability that u and v co-occur on a random walk over the graph

Node embeddings

- Random Walk Approaches
  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 in embedding space (Here: dot product=$\cos(\theta)$) encodes random walk "similarity"

Node embeddings

- Why Random Walks?
  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 with high probability, 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

Node embeddings

- Unsupervised Feature Learning
  - Intuition: Find embedding of nodes in d-dimensional space 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?
    - $N_R(u)$ ... neighbourhood of u obtained by some random walk strategy R

Node embeddings

- Feature Learning as Optimization
  - Given $G = (V, E)$,
  - Our goal is to learn a mapping $f: u \rightarrow \mathbb{R}^d$:
    $f(u) = \mathbf{z}_u$
  - Log-likelihood objective:

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

    - $N_R(u)$ is the neighborhood of node u by strategy R
  - Given node u, we want to learn feature representations that are predictive of the nodes in its random walk neighborhood $N_R(u)$.

Node embeddings

- Random Walk Optimization
  1. Run **short fixed-length random walks** starting from each node u in the graph using some random walk 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 $N_R(u)$.

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_R(u) | \mathbf{z}_u) \Rightarrow$$ Maximum likelihood objective

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

Node embeddings

- Random Walk Optimization

  Equivalently,

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

  - **Intuition:** Optimize embeddings $\mathbf{z}_u$ to maximize the likelihood of random walk co-occurrences.
  - **Parameterize $P(v|\mathbf{z}_u)$ using softmax:**

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

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

Node embeddings

- Random Walk Optimization

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

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 $\mathbf{z}_u$ that minimize $\mathcal{L}$

But doing this naively is too expensive! $O(|V|^2)$ complexity!

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

- Negative sampling allows for quick likelihood calculation.

Node embeddings

- Random Walk Optimization

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

random distribution over nodes

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

  - Sample k negative nodes each with prob. proportional to its degree
  - Two considerations for k (# negative samples):
    1. Higher k gives more robust estimates
    2. Higher k corresponds to higher bias on negative events

  In practice k =5-20.

Can negative sample be any node or only the nodes not on the walk? People often use any nodes (for efficiency). However, the most "correct" way is to use nodes not on the walk.

Node embeddings

- Random Walk Optimization
  - After we obtained the objective function, how do we optimize (minimize) it?

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

  - **Gradient Descent**: a simple way to minimize $\mathcal{L}$
  - **Stochastic Gradient Descent**: Instead of evaluating gradients over all examples, evaluate it for each **individual** training example.

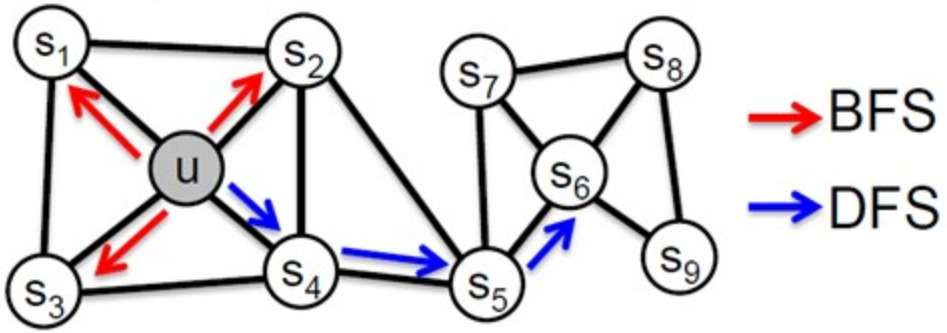
Node embeddings

- How should we randomly walk ?
  - **DeepWalk** : just run fixed-length, unbiased random walks starting from each node
    - The issue is that such notion of similarity is too constrained
  - **node2vec** : develop biased 2nd order random walk R to generate network neighborhood of the node
    - Idea : use flexible, biased random walks that can trade off between local and global views of the network

Node embeddings

- Biased random walks
  - Strategies

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

Node embeddings

- Biased random walks
  - Interpolating BFS et DFS

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

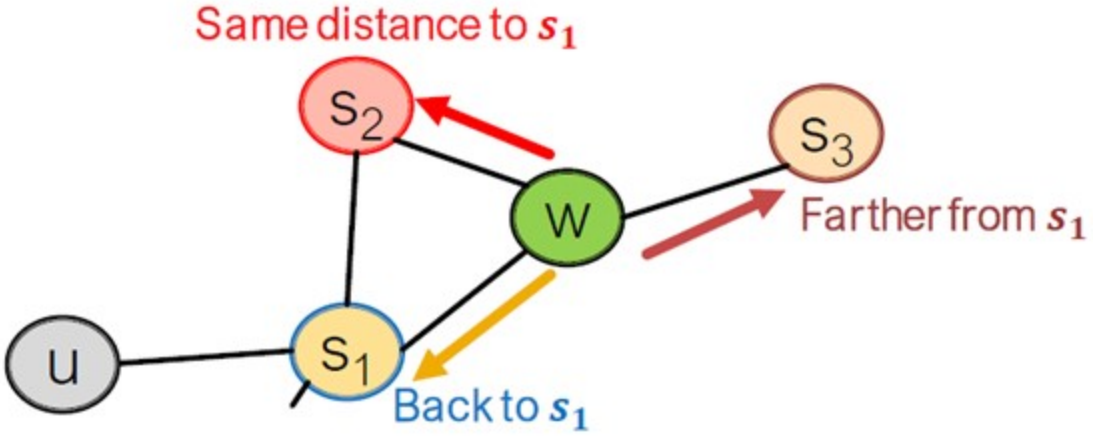
- Two parameters:
  - **Return parameter p:**
    - Return back to the previous node
  - **In-out parameter q:**
    - Moving outwards (DFS) vs. inwards (BFS)
    - Intuitively, q is the "ratio" of BFS vs. DFS

Node embeddings

- Biased random walks

Biased 2nd-order random walks explore network neighborhoods:

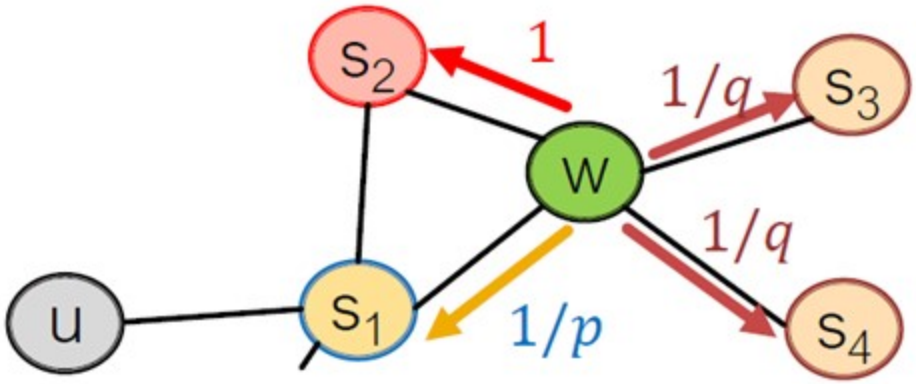
- Rnd. walk just traversed edge (s_1, w) and is now at w
- **Insight:** Neighbors of w can only be:



Idea: Remember where the walk came from

Node embeddings

- Biased random walks
  - **Walker came over edge (s_1, w) and is at w. Where to go next?**

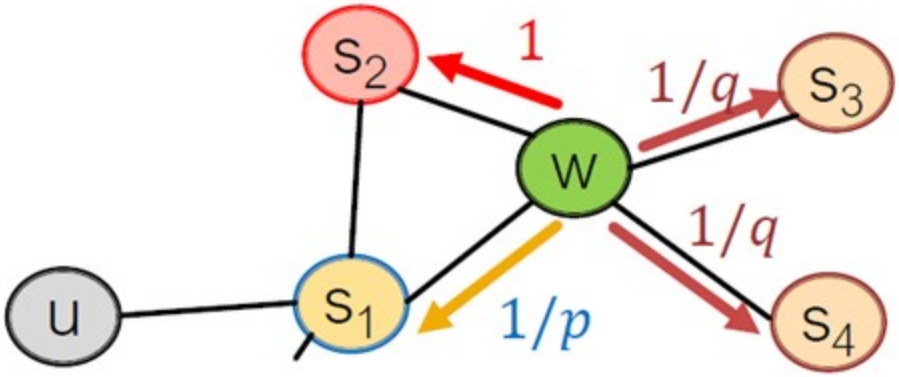


$1/p, 1/q, 1$ are unnormalized probabilities

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

Node embeddings

- 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

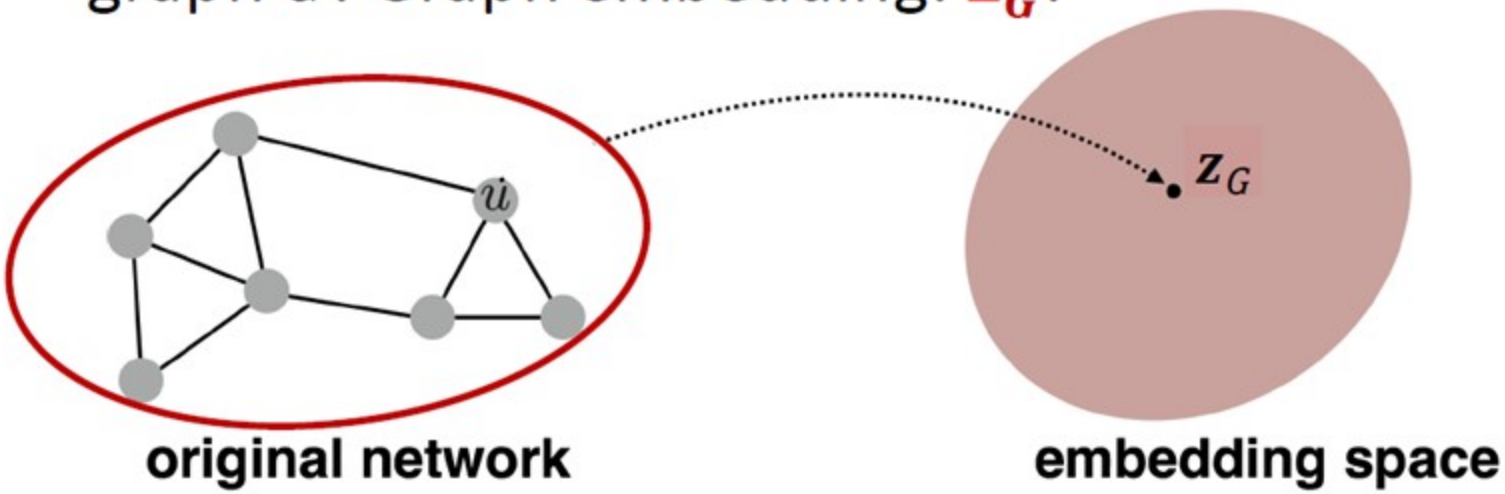
$N_R(u)$ are the nodes visited by the biased walk

Node embeddings

- Biased random walks
  - 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
  - Advantages
    - **Linear-time** complexity
    - All 3 steps are individually parallelizable

Node embeddings

- Embedding Entire Graphs
  - **Goal:** Want to embed a subgraph or an entire graph G. Graph embedding: $\mathbf{z}_G$.



  - **Tasks:**
    - Classifying toxic vs. non-toxic molecules
    - Identifying anomalous graphs

Node embeddings

- Embedding Entire Graphs
  - **Approach 1**

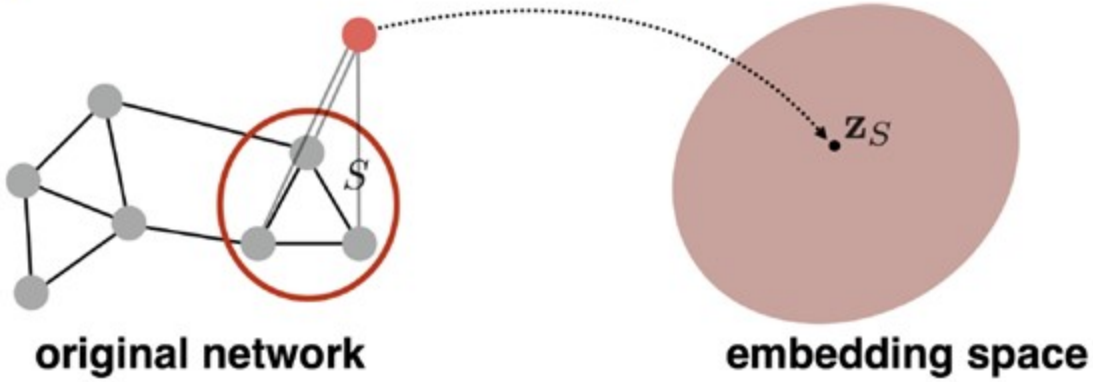
    Simple (but effective) approach 1:
    - Run a standard graph embedding technique *on* the (sub)graph G.
    - Then just sum (or average) the node embeddings in the (sub)graph G.

$$\mathbf{z}_G = \sum_{v \in G} \mathbf{z}_v$$

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

Node embeddings

- Embedding Entire Graphs
  - **Approach 2:** Introduce a **"virtual node"** to represent the (sub)graph and run a standard graph embedding technique

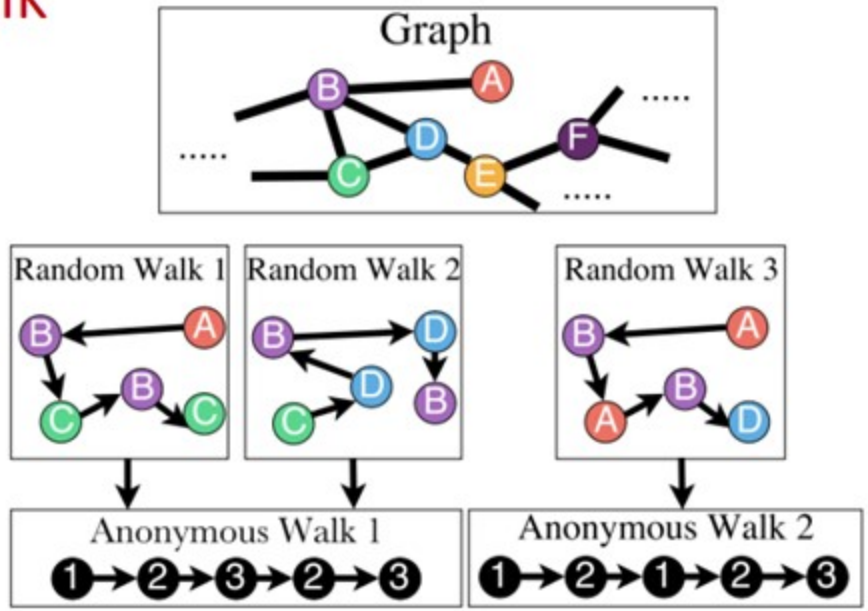


  - Proposed by Li et al., 2016 as a general technique for subgraph embedding

Node embeddings

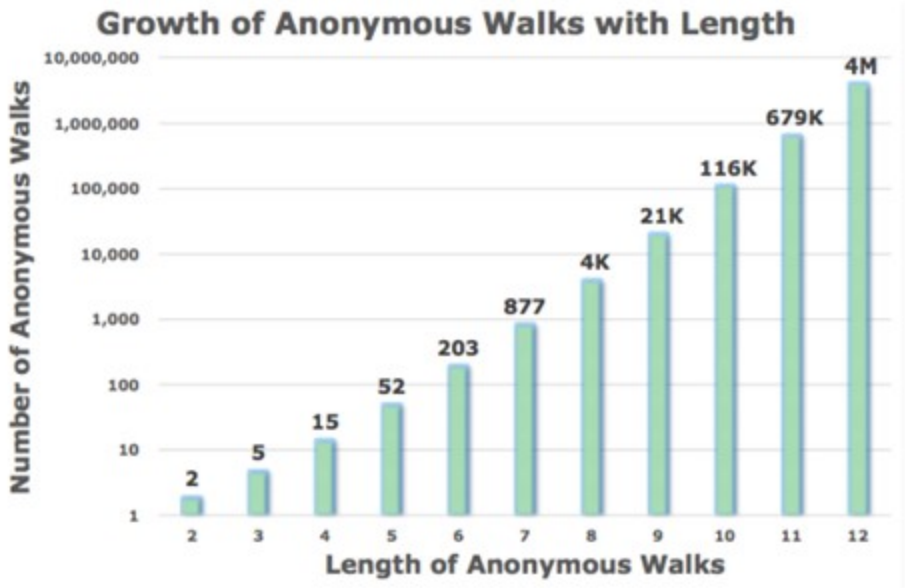
- Embedding Entire Graphs
  - **Approach 3 : Anonymous Walk Embeddings**

States in **anonymous walks** correspond to the index of the **first time** we visited the node in a random walk



Node embeddings

- Embedding Entire Graphs
  - **Approach 3 : Anonymous Walk Embeddings**



Number of anonymous walks grows exponentially:

- There are 5 anon. walks w_i of length 3:
  w_1=111, w_2=112, w_3= 121, w_4= 122, w_5= 123

Node embeddings

- Embedding Entire Graphs
  - **Approach 3 : Anonymous Walk Embeddings**
  - Simulate anonymous walks w_i of l steps and record their counts.
  - **Represent the graph as a probability distribution over these walks.**
  - **For example:**
    - Set $l = 3$
    - Then we can represent the graph as a 5-dim vector
      - Since there are 5 anonymous walks w_i of length 3: 111, 112, 121, 122, 123
    - $\boldsymbol{z}_G[i]$ = probability of anonymous walk w_i in graph G.

Node embeddings

- Embedding Entire Graphs
  - **Approach 3 : Anonymous Walk Embeddings**
    - **Sampling anonymous walks:** Generate independently a set of m random walks.
    - Represent the graph as a probability distribution over these walks.
    - How many random walks m do we need?
      - We want the distribution to have error of more than ε with prob. less than δ:

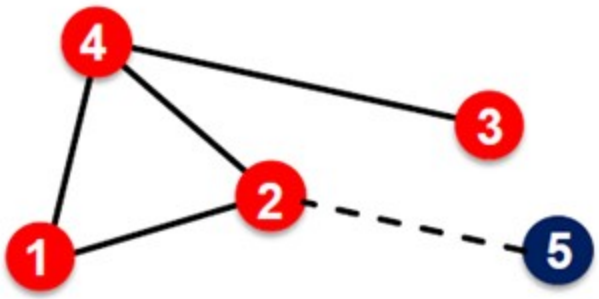
$$m = \left\lceil \frac{2}{\varepsilon^2} \left(\log(2^\eta - 2) - \log(\delta) \right) \right\rceil$$

where: η is the total number of anon. walks of length l.

For example:
There are $\eta = 877$ anonymous walks of length $l = 7$. If we set $\varepsilon = 0.1$ and $\delta = 0.01$ then we need to generate m=122,500 random walks

Node embeddings

- Limitations
  - Cannot obtain embeddings for nodes not in the training set



Training set

A newly added node 5 at test time (e.g., new user in a social network)

Cannot compute its embedding with DeepWalk / node2vec. Need to recompute all node embeddings.

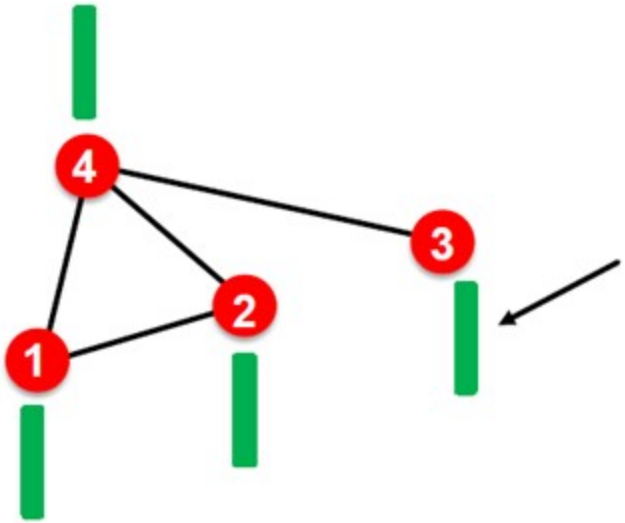
Node embeddings

- Limitations
  - Cannot capture **structural similarity**:

  - Node 1 and 11 are **structurally similar** – part of one triangle, degree 2, …
  - However, they have very **different** embeddings.
    - It's unlikely that a random walk will reach node 11 from node 1.
  - **DeepWalk and node2vec do not capture structural similarity.**

Node embeddings

- Limitations
  - Cannot utilize node, edge and graph features



Feature vector
(e.g. protein properties in a protein-protein interaction graph)

DeepWalk / node2vec embeddings do not incorporate such node features

Solution to these limitations: Deep Representation Learning and Graph Neural Networks