

COMP4222 Machine Learning with Structured Data

Network Representation Learning

Instructor: Yangqiu Song

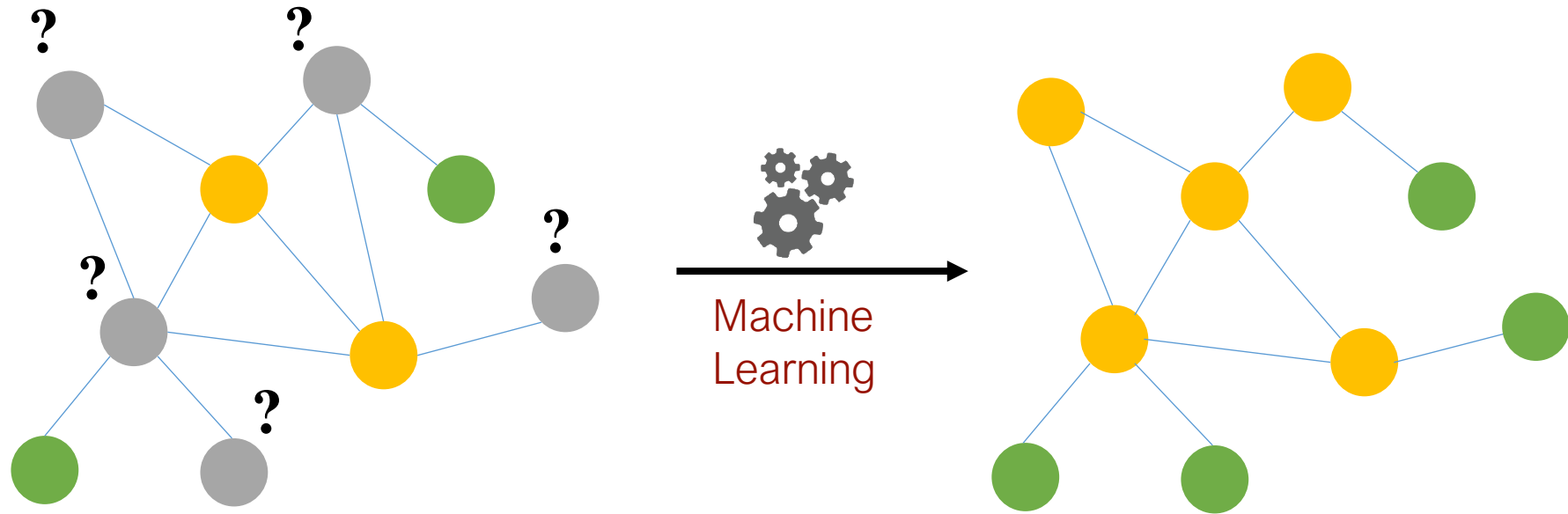
Slides credits: Jure Leskovec, William L. Hamilton, Rex Ying, Rok Sosic

Machine Learning with Graphs

Classical ML tasks in networks:

- Node classification
 - Predict a type of a given node
- Link prediction
 - Predict whether two nodes are linked
- Community detection
 - Identify densely linked clusters of nodes
- Network similarity
 - How similar are two (sub)networks

Example: Node Classification



Example: Node Classification

Classifying the function of proteins in the interactome!

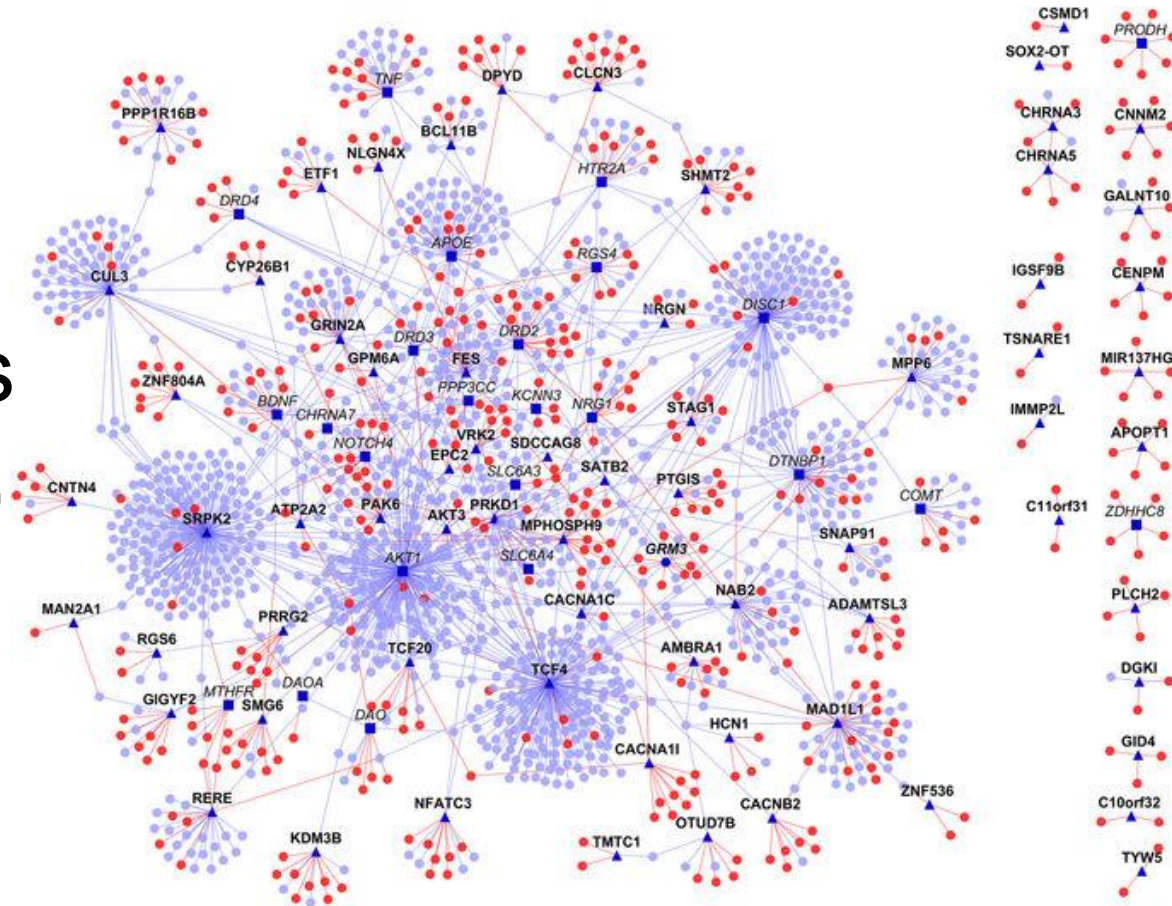
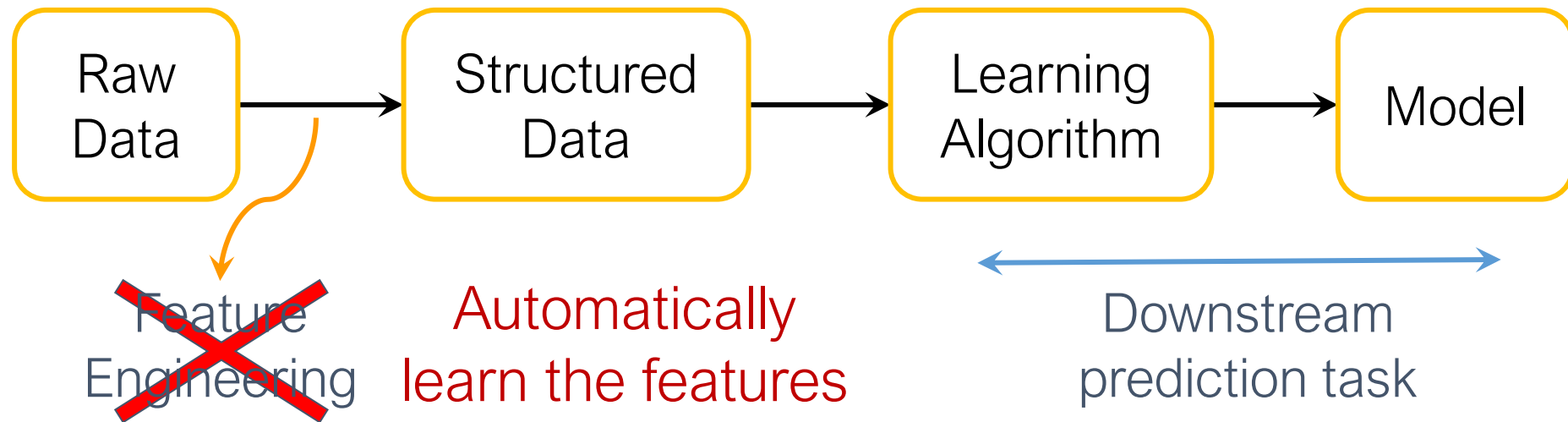


Image from: Ganapathiraju et al. 2016. [Schizophrenia interactome with 504 novel protein–protein interactions](#). *Nature*.

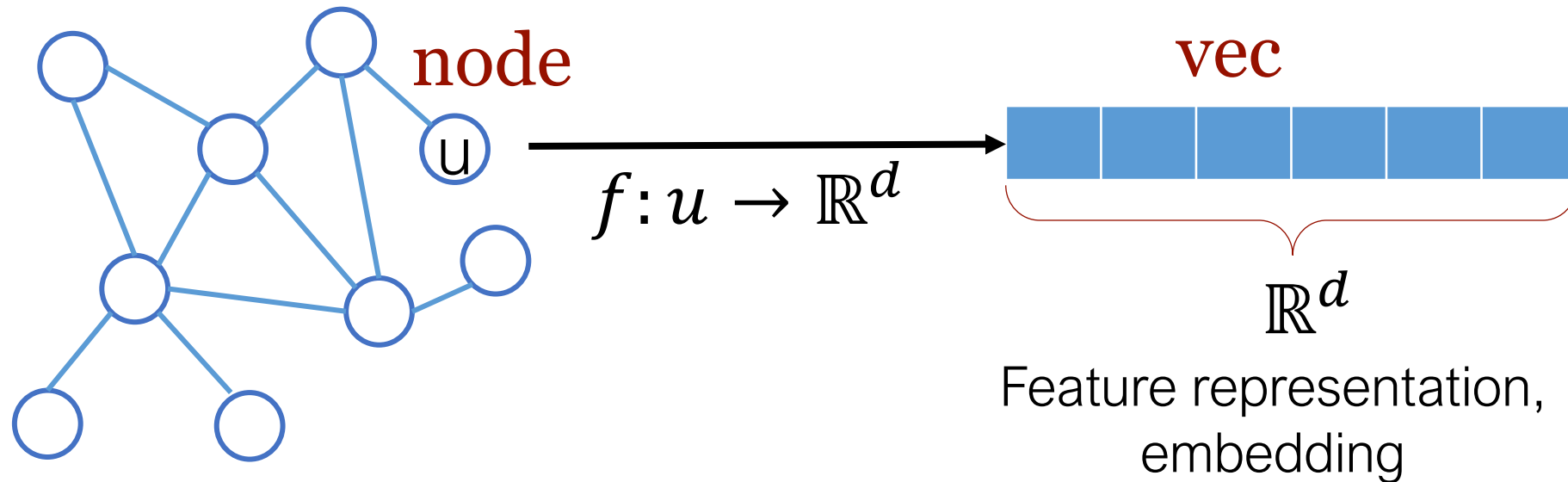
Machine Learning Lifecycle

- (Supervised) Machine Learning Lifecycle: This feature, that feature.
Every single time!



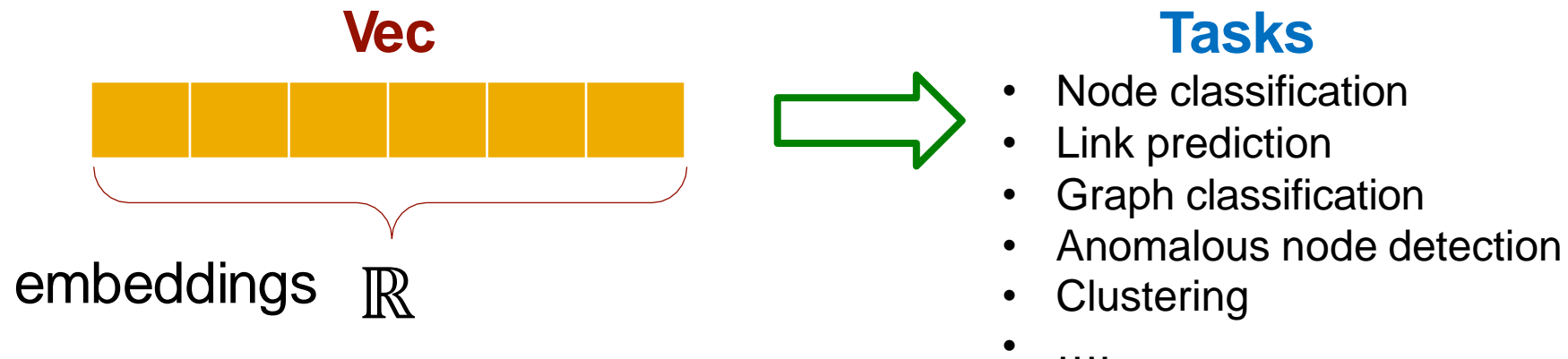
Feature Learning in Graphs

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



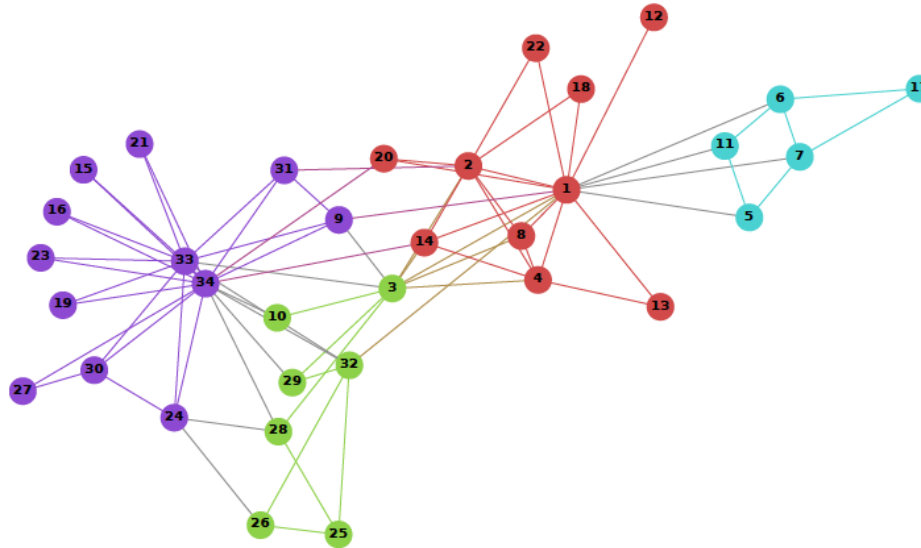
Why Embedding?

- **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

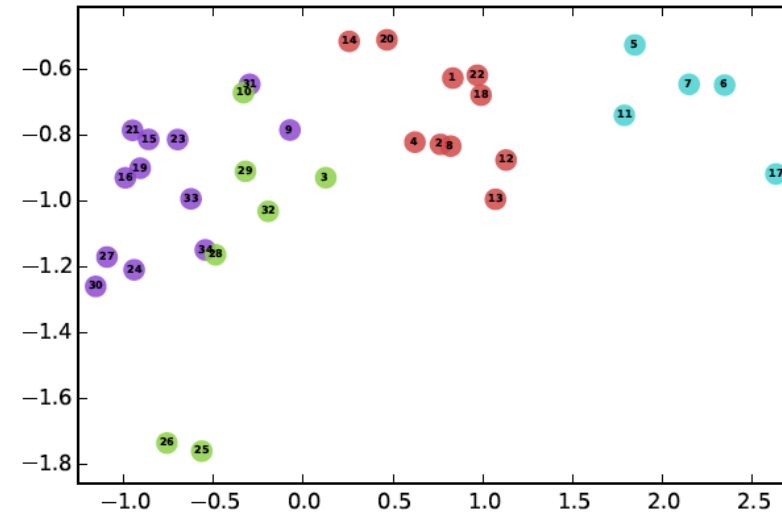


Example

- Zachary's Karate Club Network:



Input

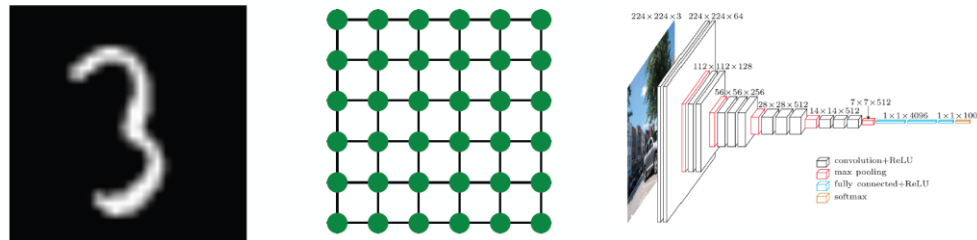


Output

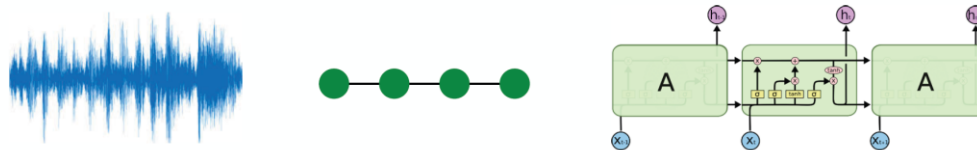
Image from: [Perozzi et al. 2014](#). DeepWalk: Online Learning of Social Representations. *KDD*.

Why Is It Hard?

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

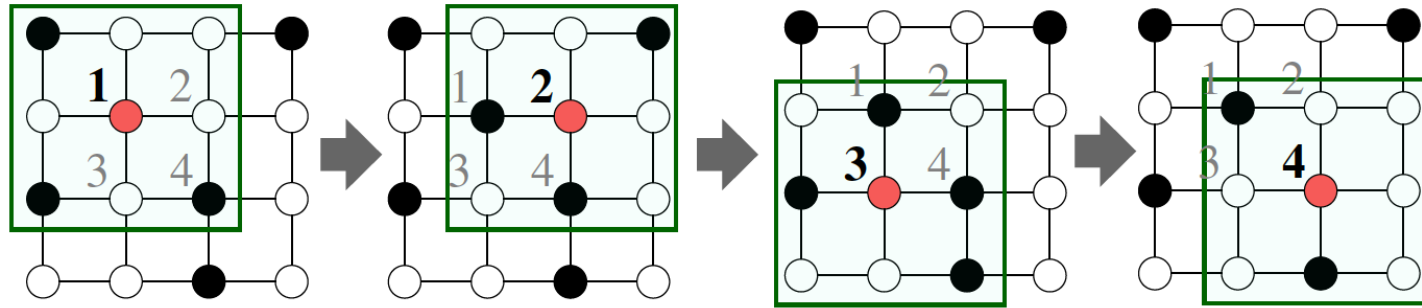


- RNNs or word2vec for text/sequences...



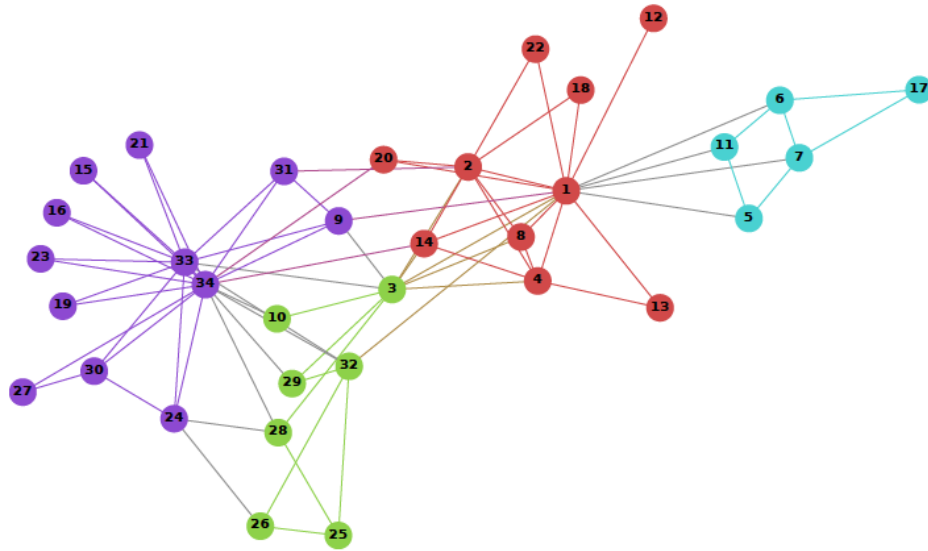
Why Is It Hard?

- But networks are far more complex!
 - Complex topographical structure (i.e., no spatial locality like grids)

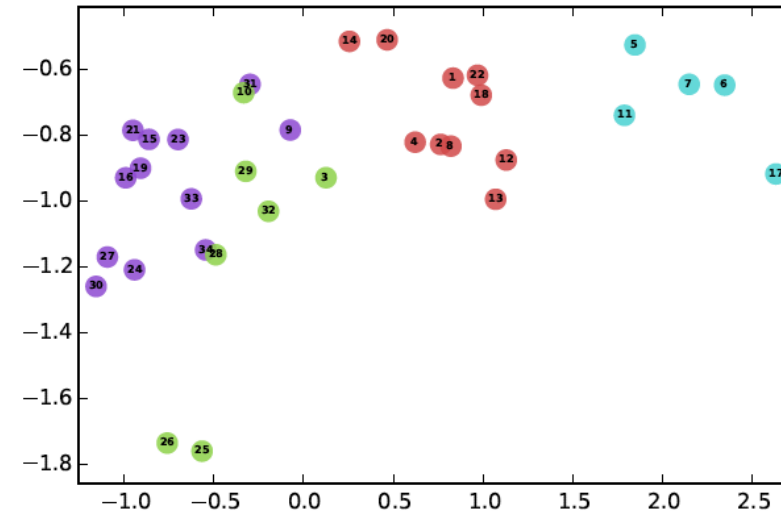


- No fixed node ordering or reference point (i.e., the isomorphism problem)
- Often dynamic and have multimodal features.

Embedding Nodes



Input



Output

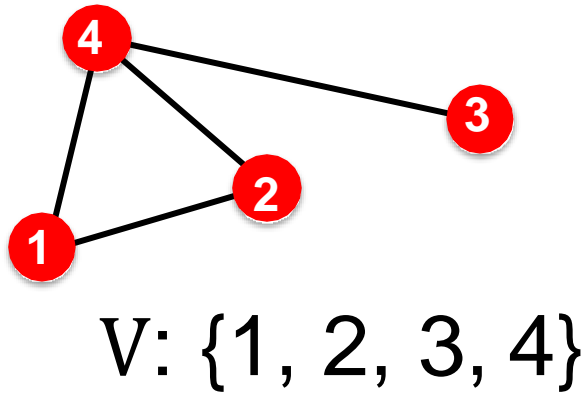
Intuition: Find embedding of nodes to d-dimensions so that “similar” nodes in the graph have embeddings that are close together.

Important Works in the History

- Graph/Network Embedding
 - ISOMap [Tenenbaum et al., Science'00]
 - LLE [Roweis and Saul, Science'00]
 - Laplacian EigenMap [Belkin et al., NIPS'01]
 - (t)-SNE [Maaten and Hinton, JMLR'08]
 - Deepwalk [Perozzi et al., KDD'14]
 - LINE [Tang et al., WWW'15]
 - Node2vec [Grover and Leskovec, KDD'16]

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!**

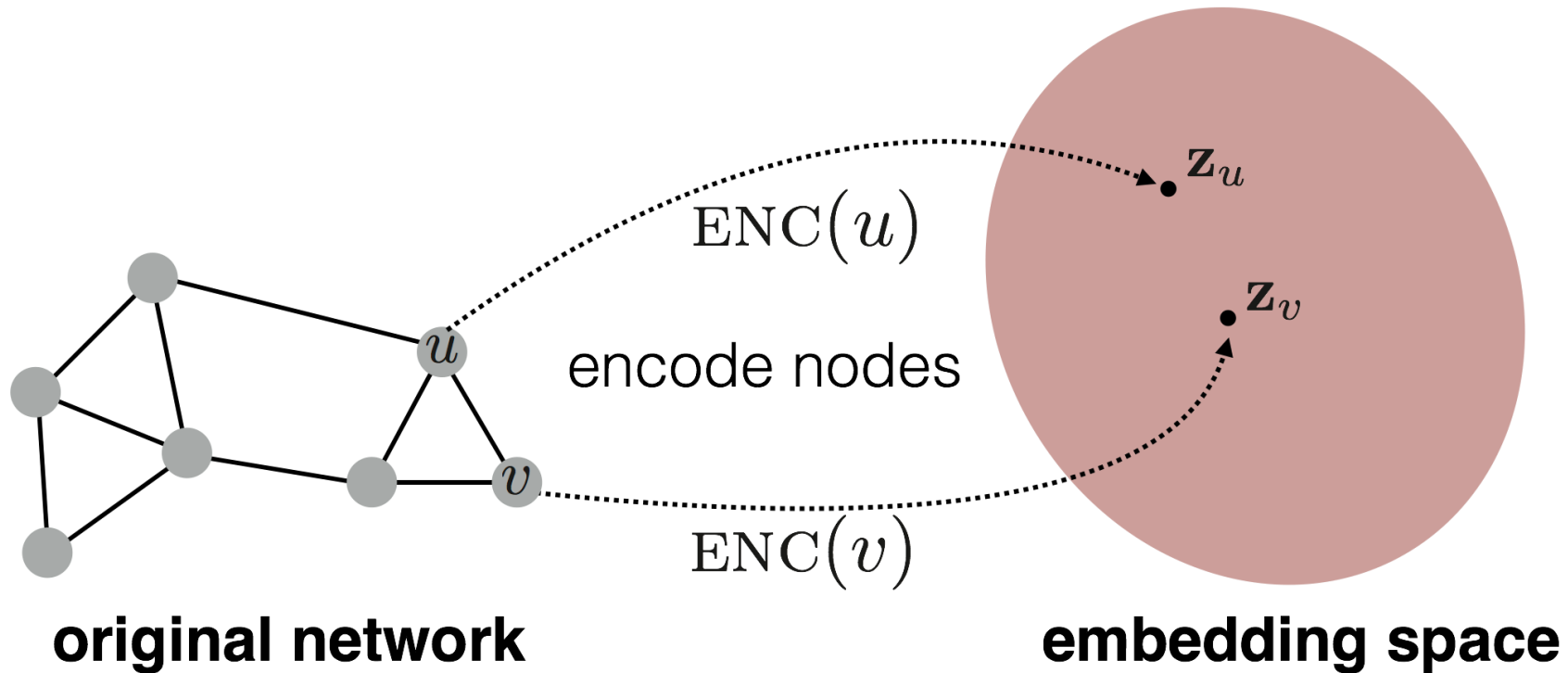


$A_{ij} = 1$ if there is a link from node i to node j
 $A_{ij} = 0$ otherwise

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

Embedding Nodes

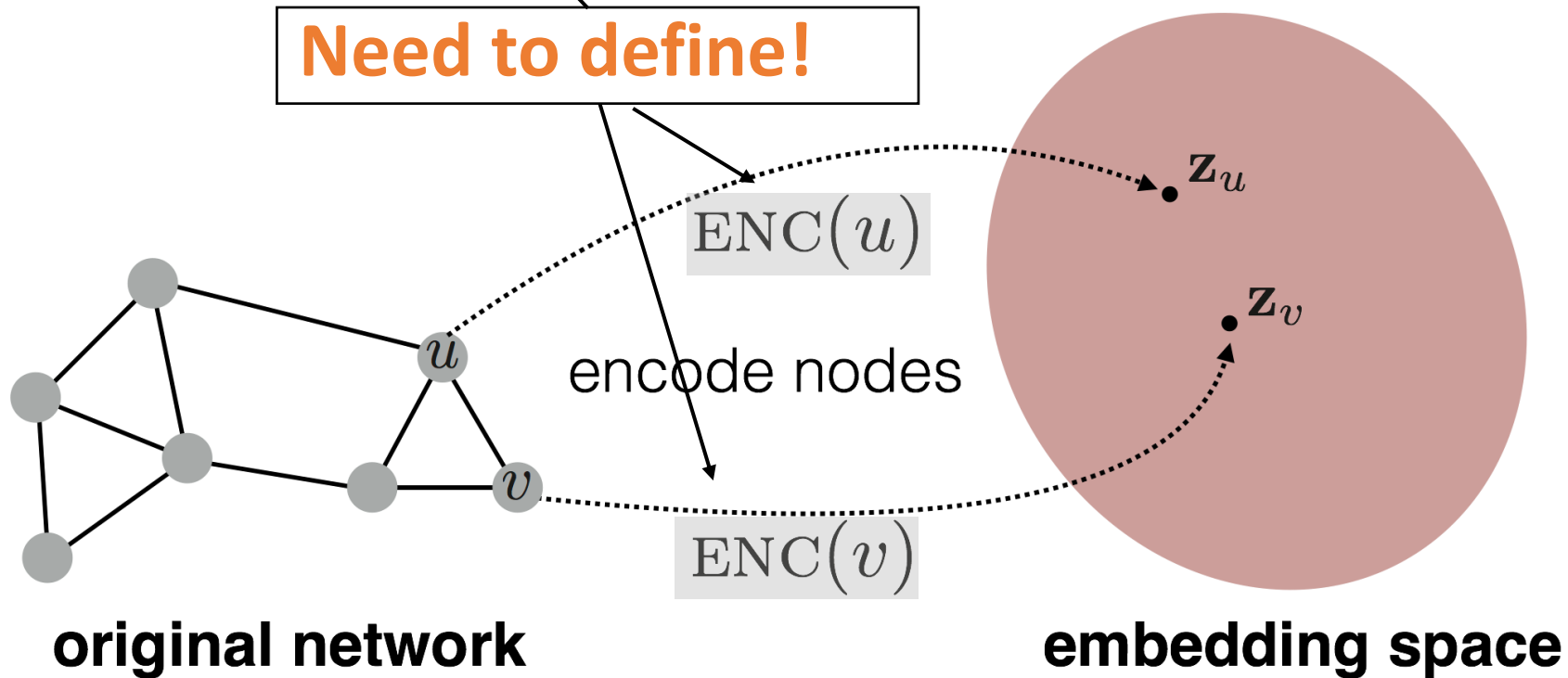
- Goal is to encode nodes so that **similarity in the embedding space (e.g., dot product)** approximates **similarity in the original network**.



Embedding Nodes

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

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. **Decoder** maps from embeddings to the similarity score
4. **Optimize the parameters of the encoder so that:**

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

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

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.

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

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**

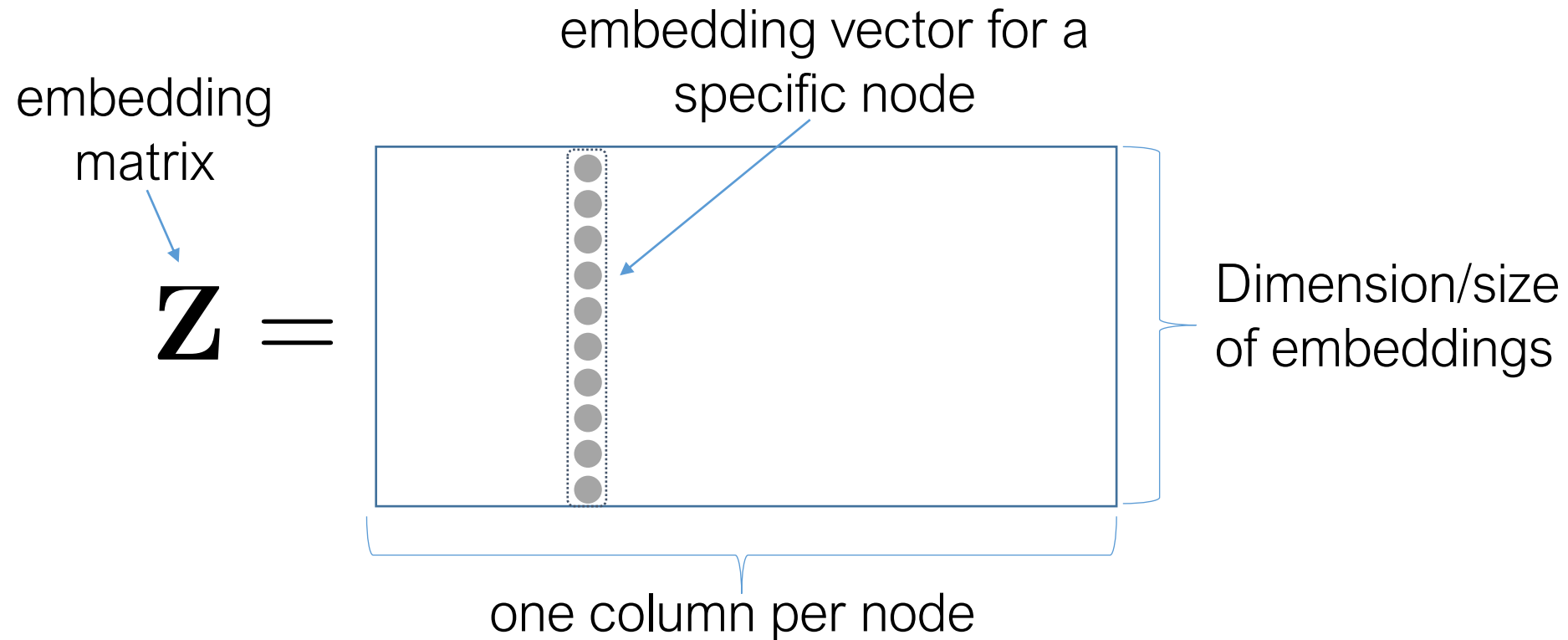
$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is 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.**

i.e., each node is assigned a unique embedding vector.
(i.e., we directly optimize the embedding of each node)

Many methods: DeepWalk, node2vec

Framework Summary

- **Encoder + Decoder Framework**
 - Shallow encoder: embedding lookup
 - Parameters to optimize: \mathbf{Z} which contains node embeddings \mathbf{z}_u for all nodes $u \in V$
 - We will cover deep encoders (GNNs) later
 - **Decoder:** based on node similarity.
- **Objective:** maximize $\mathbf{z}^T \mathbf{z}$ for node pairs (u, v) that are **similar**

Note on 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.

How to Define Node Similarity?

- Key distinction between “shallow” 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”?
 - ...?
- We will also learn node similarity definition that uses **random walks**, and how to optimize embeddings for such a similarity measure.

We introduce two ways

1. Adjacency-based similarity

2. Random walk approaches

High-level structure and material from:

- [Hamilton et al. 2017](#). Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.

Adjacency-based Similarity

Material based on:

- Ahmed et al. 2013. [Distributed Natural Large Scale Graph Factorization](#). WWW.

Adjacency-based Similarity

- **Similarity function** is just the edge weight between u and v in the original network.
- **Intuition:** Dot products between node embeddings approximate edge existence.

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v} \|^2$$

loss (what we want to minimize)

sum over all node pairs

embedding similarity

(weighted) adjacency matrix for the graph

Adjacency-based Similarity

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}\|^2$$

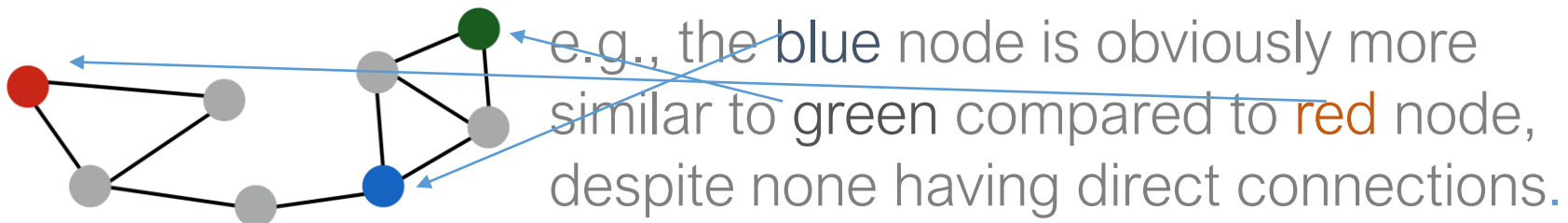
- Find embedding matrix $\mathbf{Z} \in \mathbb{R}^{d \times |V|}$ that minimizes the loss \mathcal{L}
 - Option 1: Use stochastic gradient descent (SGD) as a general optimization method.
 - Highly scalable, general approach
 - Option 2: Solve matrix decomposition solvers (e.g., SVD or QR decomposition routines).
 - Only works in limited cases.

Adjacency-based Similarity

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}\|^2$$

- **Drawbacks:**

- $O(|V|^2)$ runtime. (Must consider all node pairs.)
 - Can make $O(|E|)$ by only summing over non-zero edges and using regularization (e.g., [Ahmed et al., 2013](#))
- Only considers direct, local connections.




Random Walk Approaches

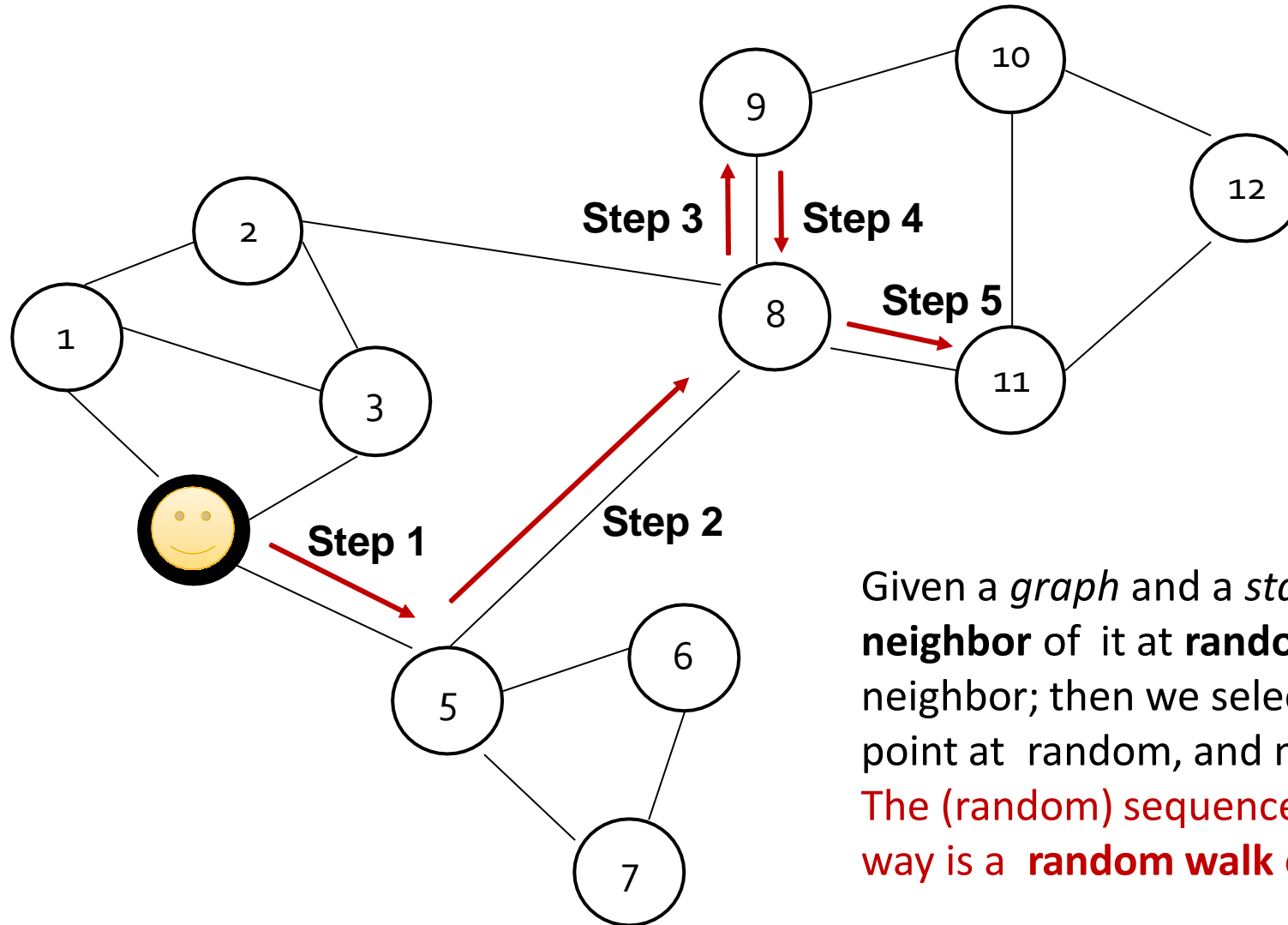
Material based on:

- Perozzi et al. 2014. [DeepWalk: Online Learning of Social Representations](#). *KDD*.
- Grover et al. 2016. [node2vec: Scalable Feature Learning for Networks](#). *KDD*.

Notation

- **Vector** \mathbf{z}_u :
 - The embedding of node u (what we aim to find).
- **Probability** $P(v|\mathbf{z}_u)$:  Our model prediction based on \mathbf{z}_u
 - The **(predicted) probability** of visiting node v on random walks starting from node u .
- **Non-linear functions used to produce predicted probabilities**
- **Softmax** function:
 - Turns vector of K real values (model predictions) into K probabilities that sum to 1: $\sigma(\mathbf{z})[i] = \frac{e^{\mathbf{z}[i]}}{\sum_{j=1}^K e^{\mathbf{z}[j]}}$
- **Sigmoid** function:
 - S-shaped function that turns real values into the range of (0, 1).
 - Written as $S(x) = \frac{1}{1+e^{-x}}$.

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**.

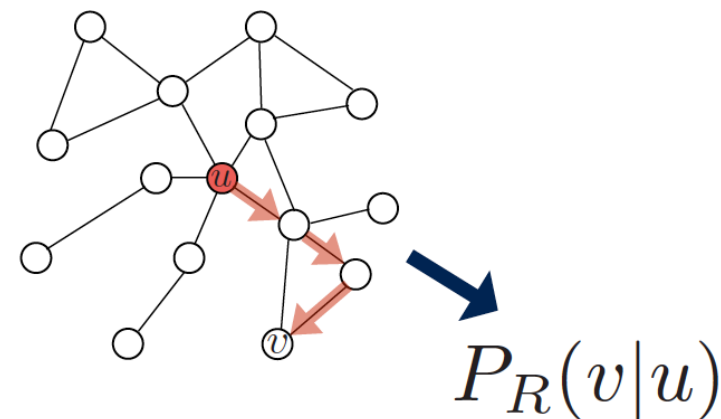
Random-walk Embeddings

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

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 in embedding space (Here: dot product= $\cos(\theta)$) encodes random walk “similarity”



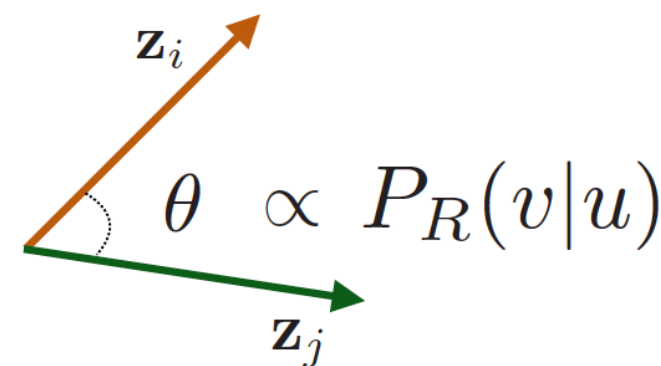
One hop: $D^{-1}A$

Two hop: $(D^{-1}A)^2$

Three hop: $(D^{-1}A)^3$

Random Walk Window Size 3:

$$D^{-1}A + (D^{-1}A)^2 + (D^{-1}A)^3$$



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.

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?**
 - $N_R(u)$... neighborhood of u obtained by some **random walk strategy R**

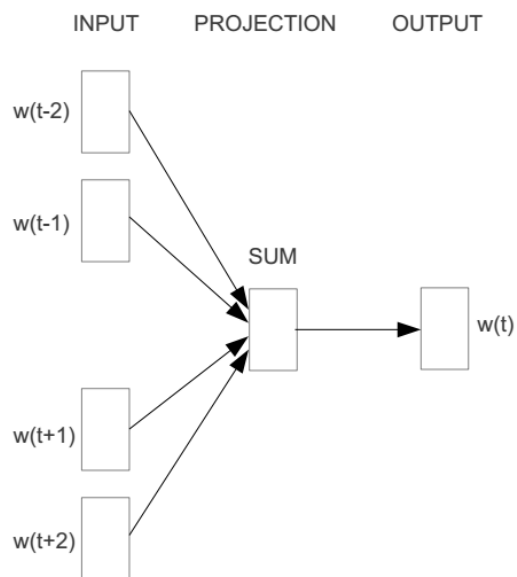
DeepWalk: The Big Idea

- DeepWalk **learns** a latent representation of adjacency matrices using deep learning techniques developed for language modeling

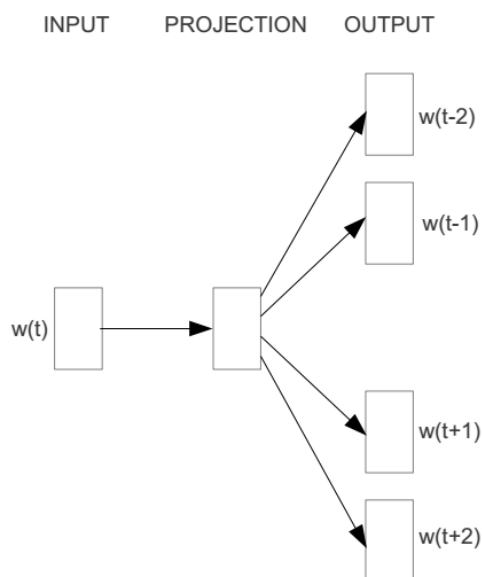
Language Modeling

stains open and the moon shining in on the
 and the cold , close moon " . And neither o
 the night with the moon shining so bright
 in the light of the moon . It all boils do
 ly under a crescent moon , thrilled by ice
 the seasons of the moon ? Home , alone ,
 dazzling snow , the moon has risen full an
 d the temple of the moon , driving out of

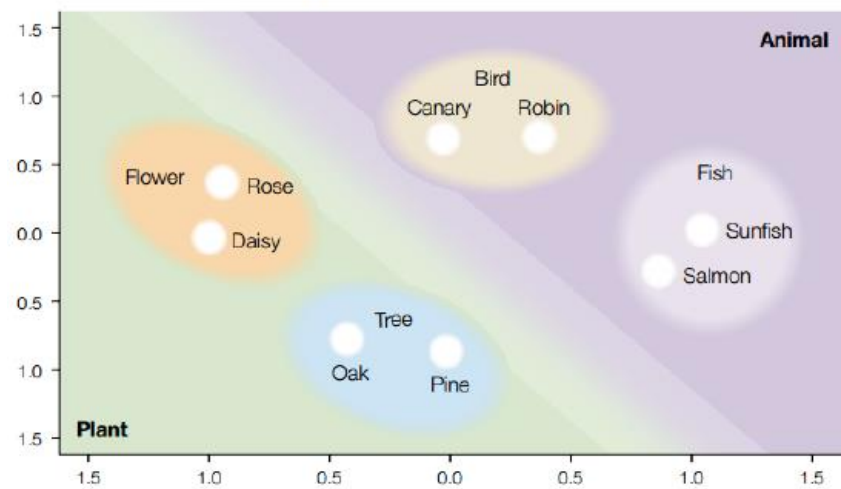
[Baroni et al, 2009]



CBOW

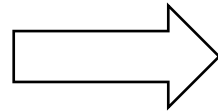
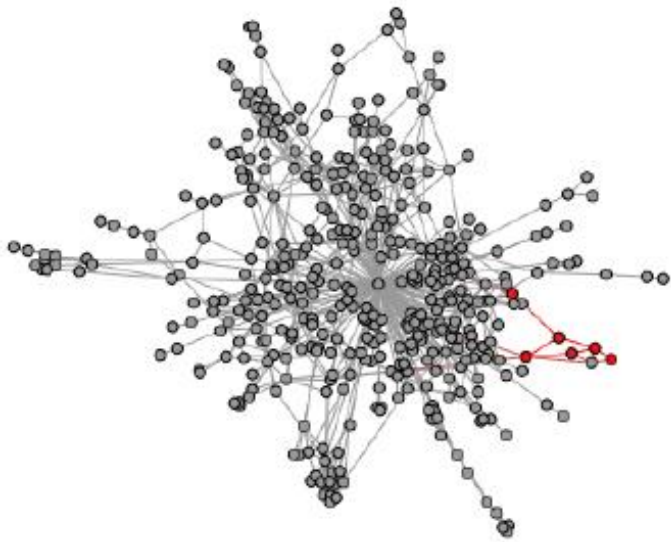


Skip-gram



[Rumelhart+, 2003]

Random Walk Paths as Documents



$v_{71} \rightarrow v_{24} \rightarrow v_5 \rightarrow \textcolor{red}{v_1} \rightarrow v_{17} \rightarrow v_{80} \rightarrow$
 $v_{92} \rightarrow v_2 \rightarrow v_3 \rightarrow \textcolor{red}{v_1} \rightarrow v_{12} \rightarrow v_{73} \rightarrow$
 $v_{37} \rightarrow v_{34} \rightarrow v_9 \rightarrow \textcolor{red}{v_1} \rightarrow v_{10} \rightarrow v_{94} \rightarrow$
 $v_{73} \rightarrow v_{64} \rightarrow v_5 \rightarrow \textcolor{red}{v_1} \rightarrow v_{12} \rightarrow v_1 \rightarrow$
 $v_{75} \rightarrow v_{14} \rightarrow v_6 \rightarrow \textcolor{red}{v_1} \rightarrow v_{13} \rightarrow v_{61} \rightarrow$

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: $\max_f \sum_{u \in V} \log P(N_R(u) | \mathbf{z}_u)$
- $N_R(u)$ 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 $N_R(u)$.

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 $N_R(u)$.

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

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

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:**

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

Random Walk Optimization

- Putting it all together:

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

sum over all nodes u

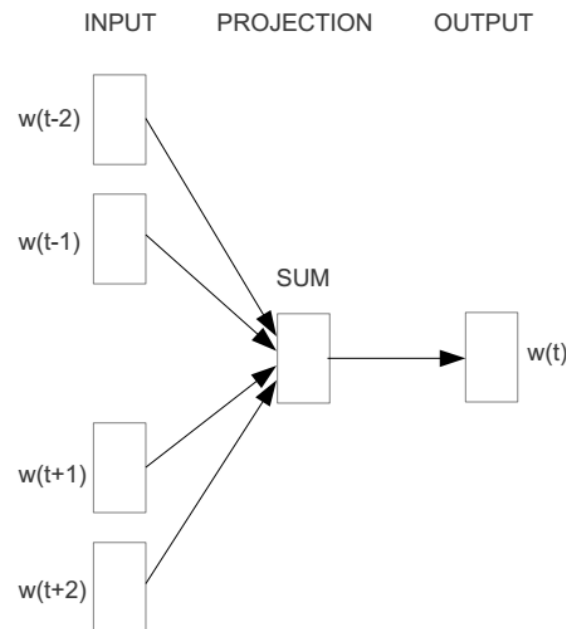
sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occurring on random walk

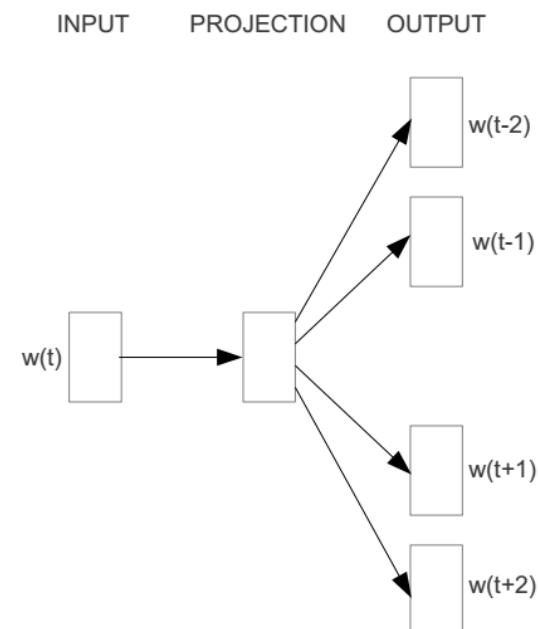
Optimizing random walk embeddings =
Finding embeddings \mathbf{z}_u that minimize \mathcal{L}

Main Idea from word2vec

- 2 basic neural network models:
 - Continuous Bag of Word (CBOW): use a window of word to predict the middle word
 - Skip-gram (SG): use a word to predict the surrounding ones in window.



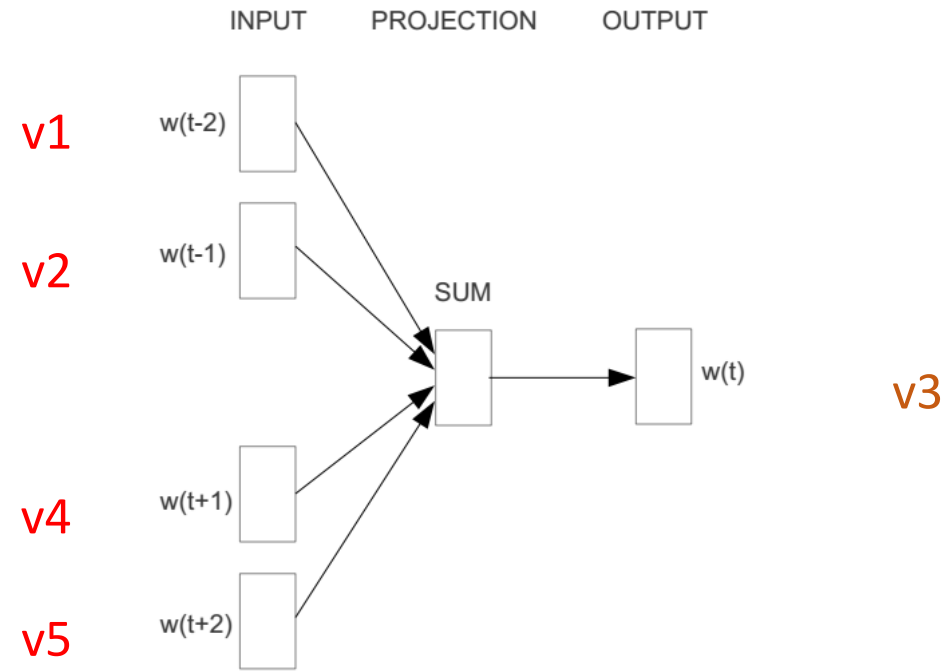
CBOW



Skip-gram

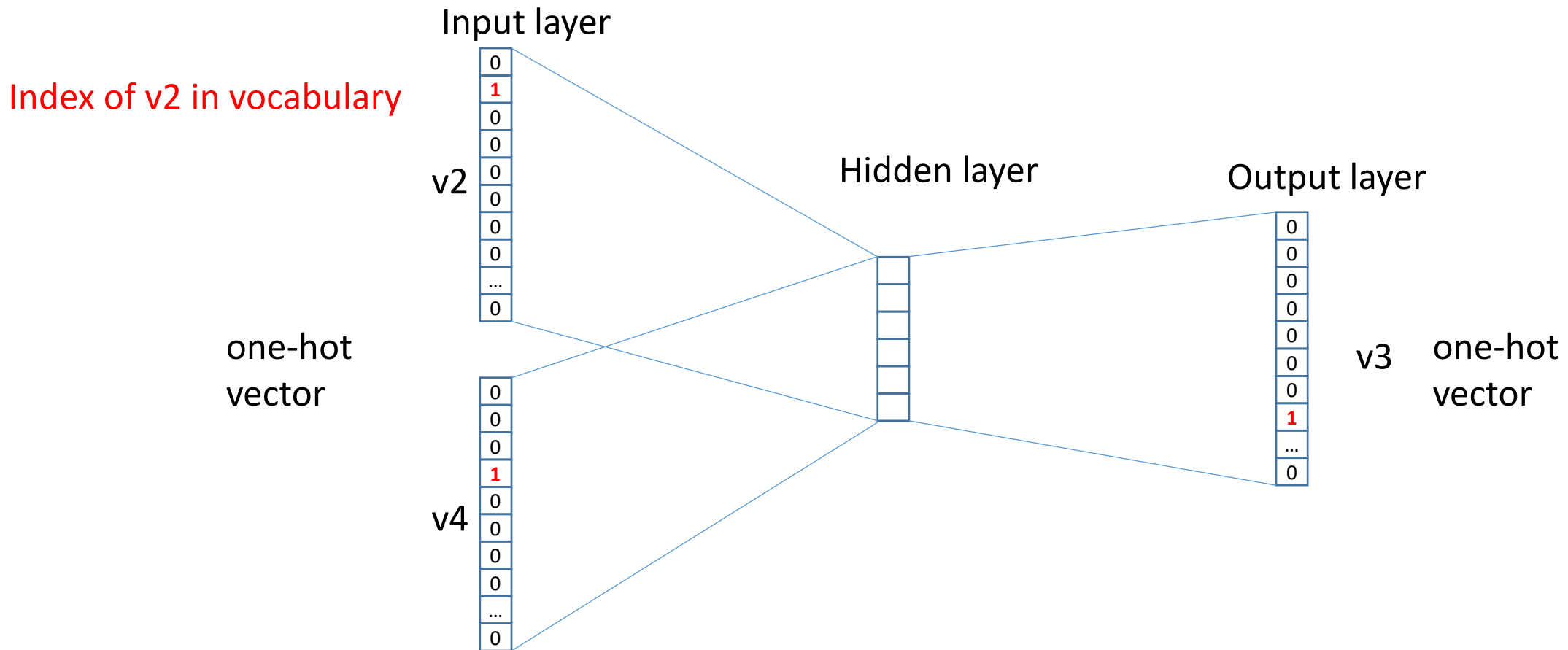
Continuous Bag of Word

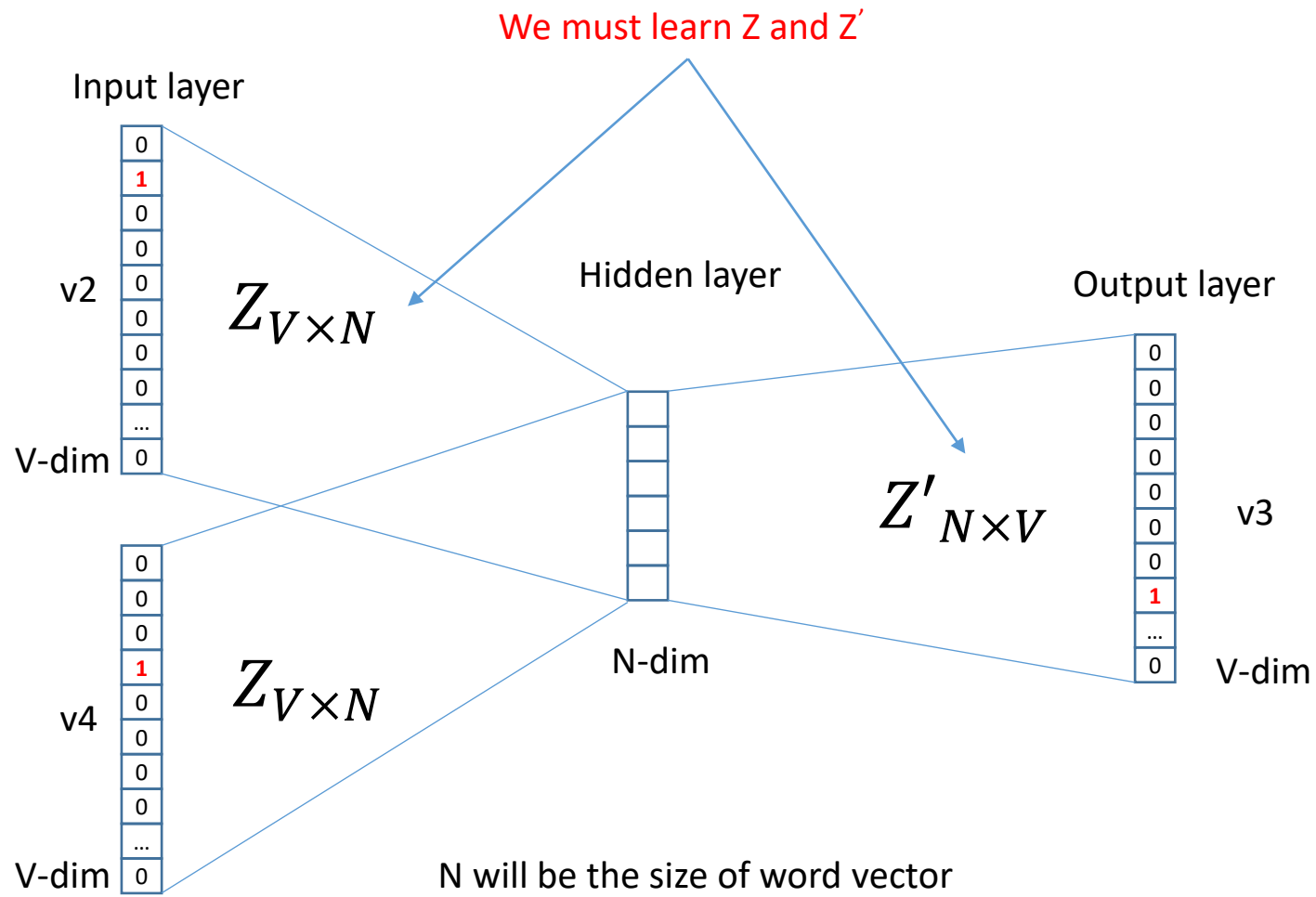
- A random walk contains “v1 v2 v3 v4 v5”
 - Window size = 2



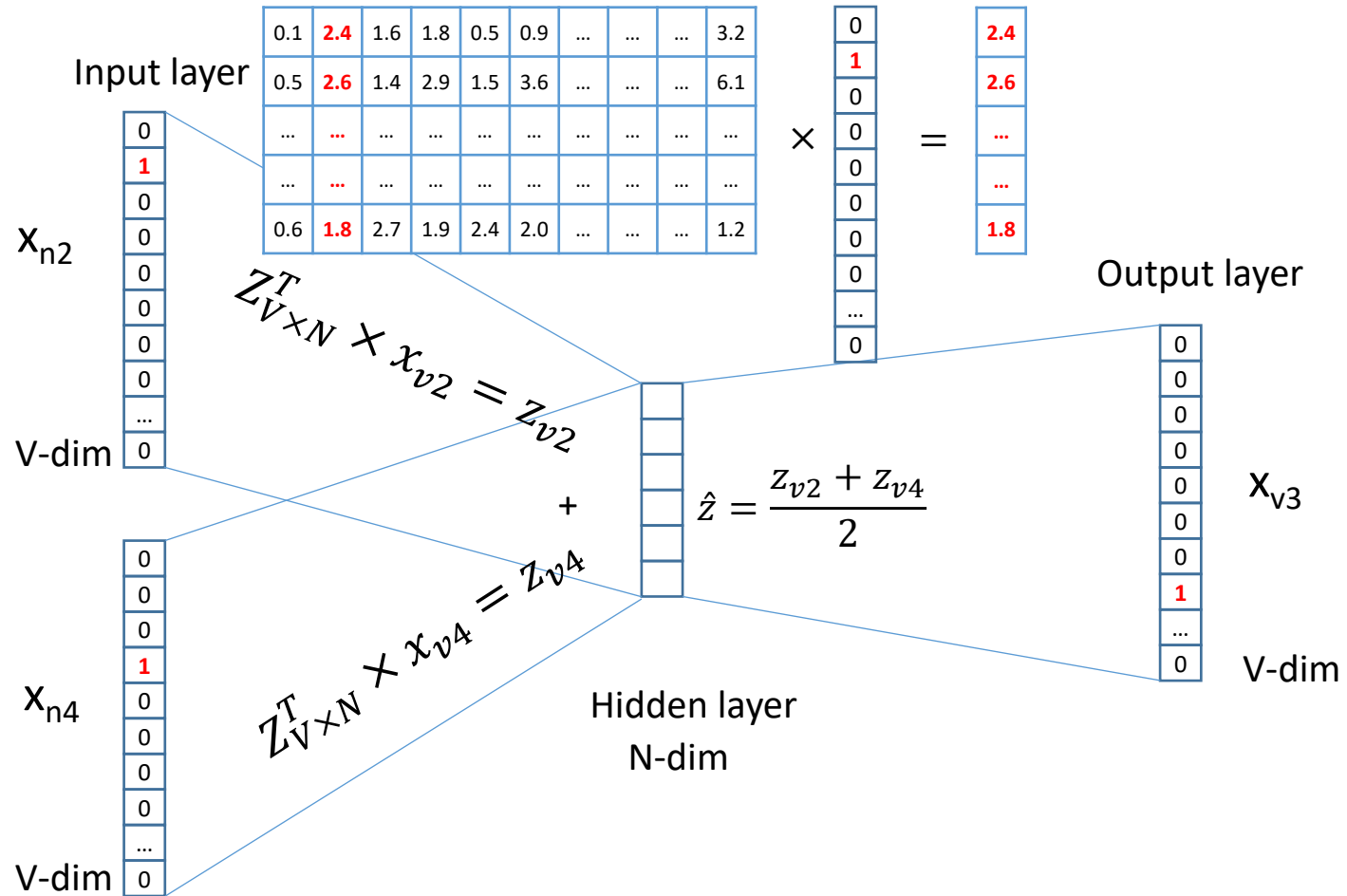
A random walk contains “v1 v2 v3 v4 v5”

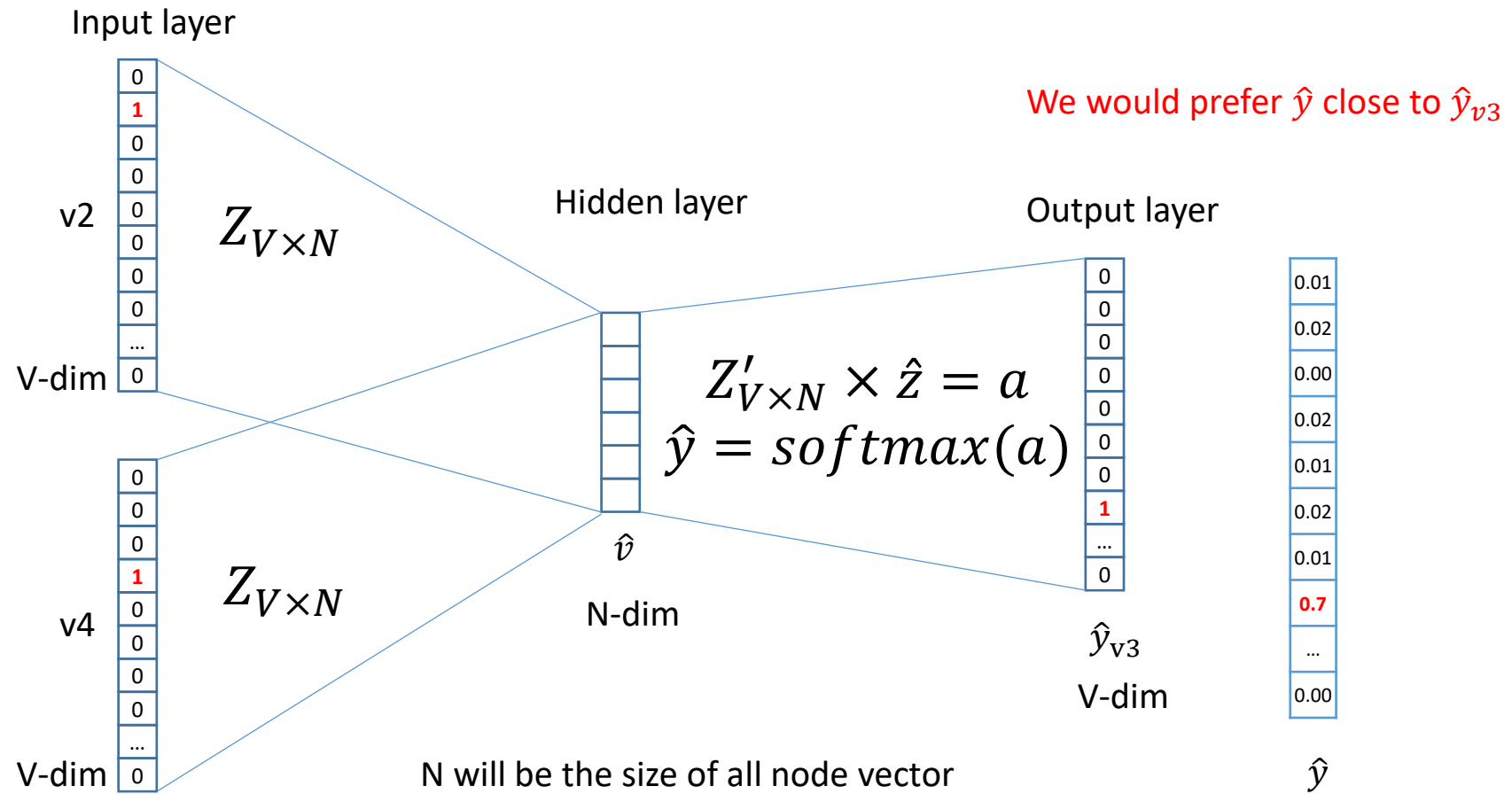
Window size = 1





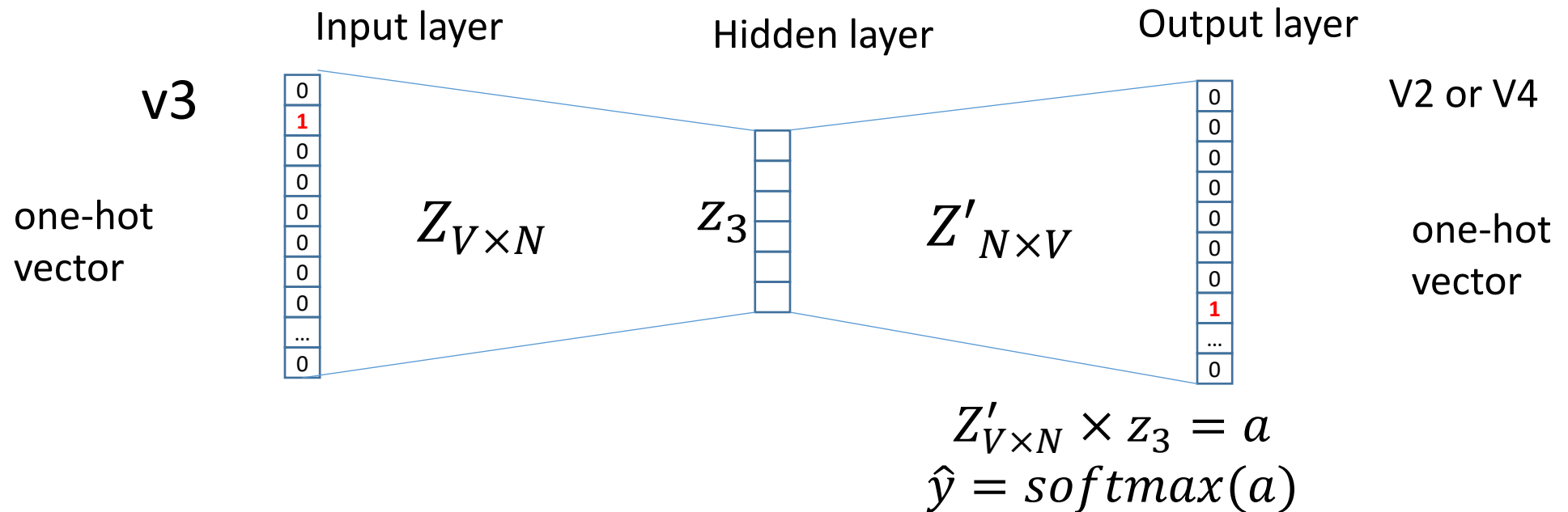
$$Z_{V \times N}^T \times x_{v2} = z_{v2}$$





Skip-gram Model

- A random walk contains “v1 v2 v3 v4 v5”
- Window size = 1



Random Walk Optimization

- But doing this naively is too expensive!

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

Nested sum over nodes gives
 $O(|V|^2)$ complexity!

Random Walk Optimization

- But doing this naively is too expensive!

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

The normalization term from the softmax is too costly
(averaging over all the nodes on a graph)

Can we approximate it?


Negative Sampling

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.

- Sample k negative nodes each with prob. proportional to its degree
- Two considerations for k (# negative samples):
 - Higher k gives more robust estimates
 - Higher k corresponds to higher bias on negative events
 - In practice $k = 5-20$.

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

random distribution over nodes


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

Random Walk based Learning Summary

1. Run short fixed-length random walks starting from each node on the graph
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

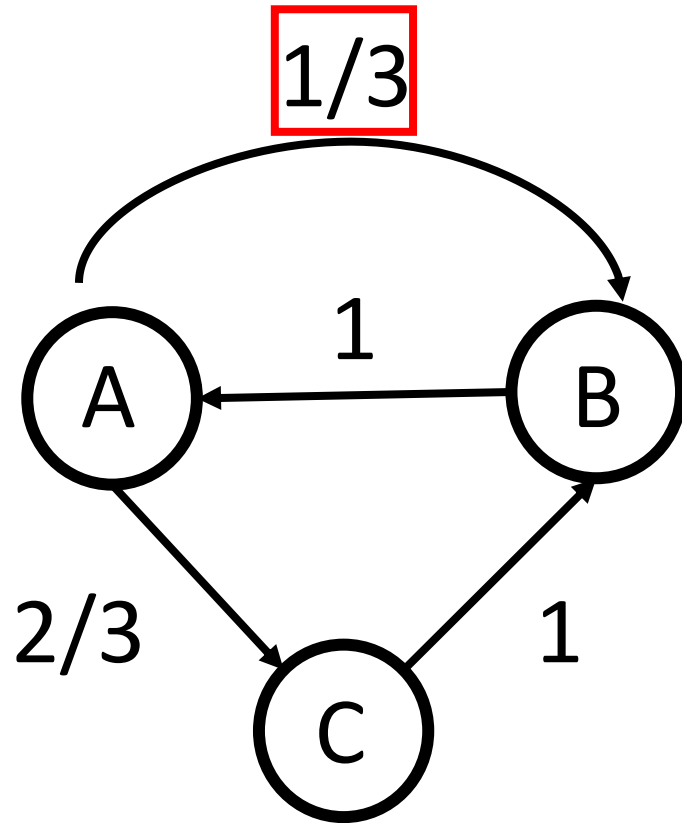
$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

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](#)).
 - But can we do better?

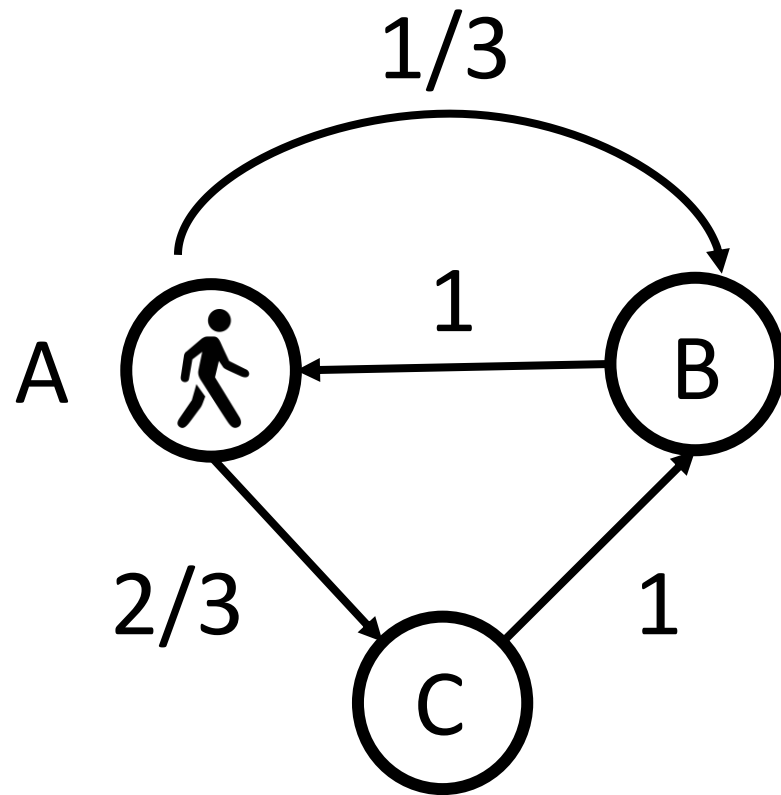
Random Walk



Transition Probability

	A	B	C
A	0	$\frac{1}{3}$	$\frac{2}{3}$
B	1	0	0
C	0	1	0

Random Walk

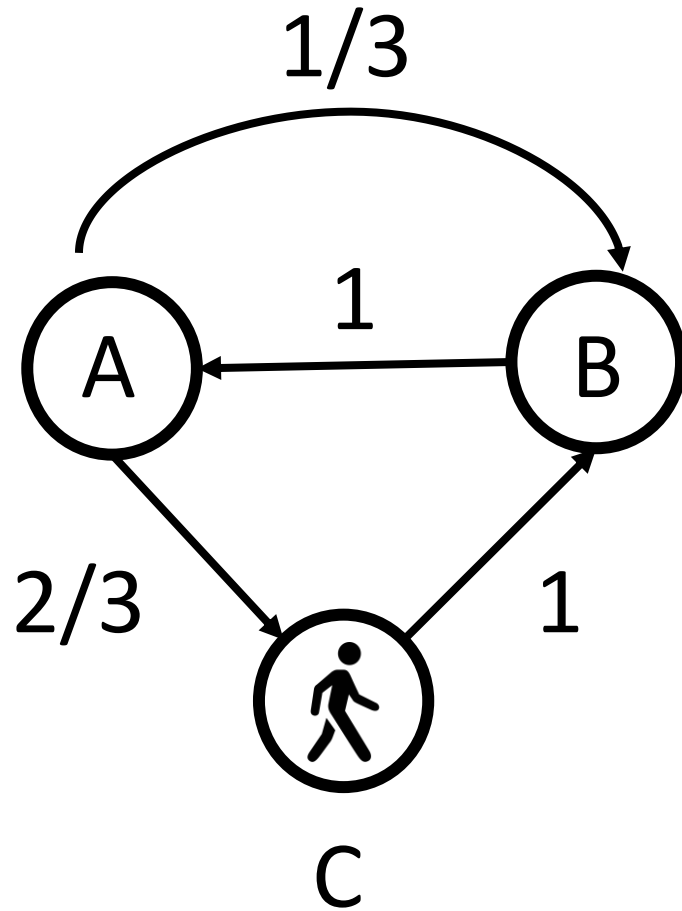


Transition Probability

	A	B	C
A	0	$1/3$	$2/3$
B	1	0	0
C	0	1	0

Path: A -

Random Walk

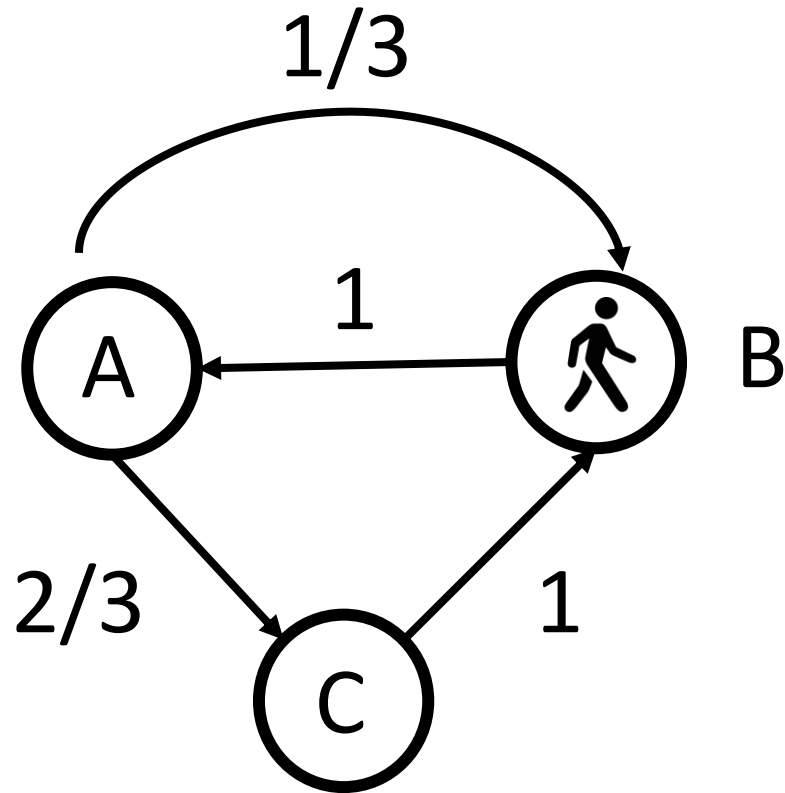


Path: A - C

Transition Probability

	A	B	C
A	0	$1/3$	$2/3$
B	1	0	0
C	0	1	0

Random Walk

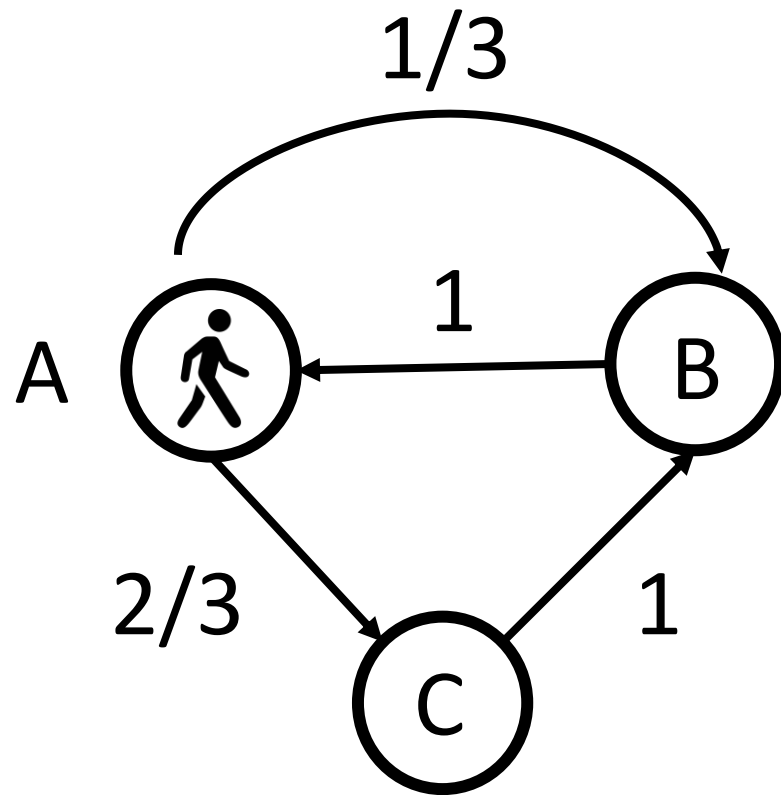


Transition Probability

	A	B	C
A	0	$1/3$	$2/3$
B	1	0	0
C	0	1	0

Path: A - C - B

Random Walk



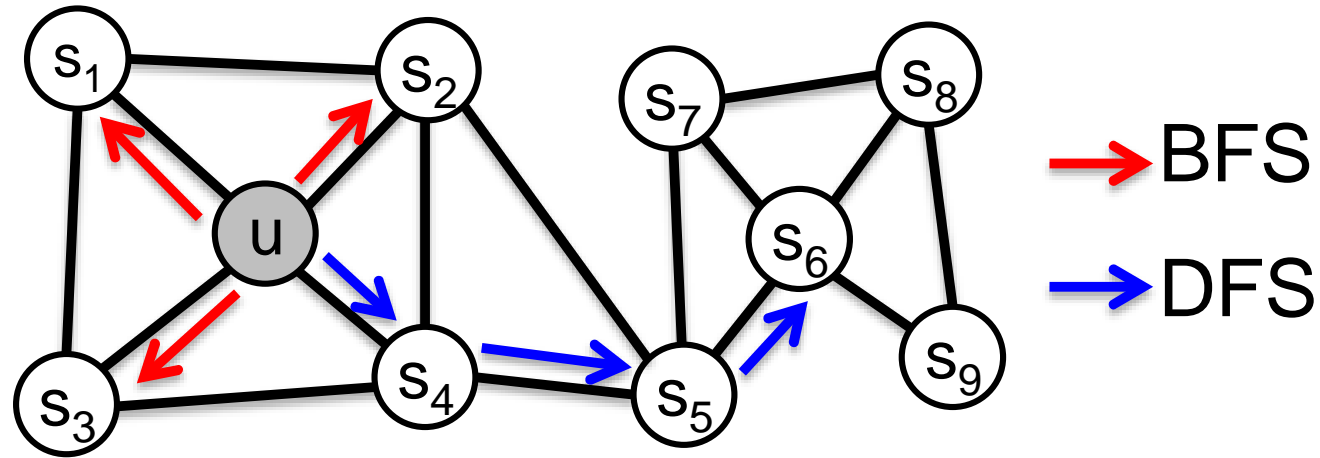
Transition Probability

	A	B	C
A	0	$1/3$	$2/3$
B	1	0	0
C	0	1	0

Path: A - C - B - A

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](#)).

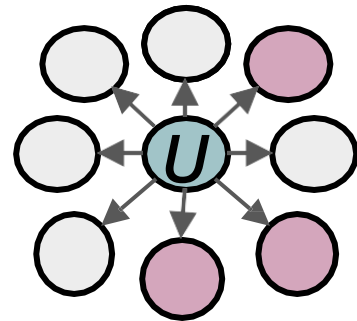


Two classic strategies to define a neighborhood $N_R(u)$ of a given node u :

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

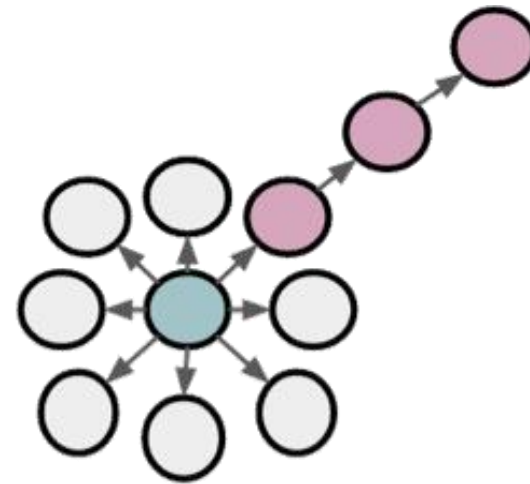
$N_{DFS}(u) = \{s_4, s_5, s_6\}$ **Global** macroscopic view

BFS vs. DFS



BFS:

Micro-view of
neighbourhood



DFS:

Macro-view of
neighbourhood

Interpolating BFS and DFS

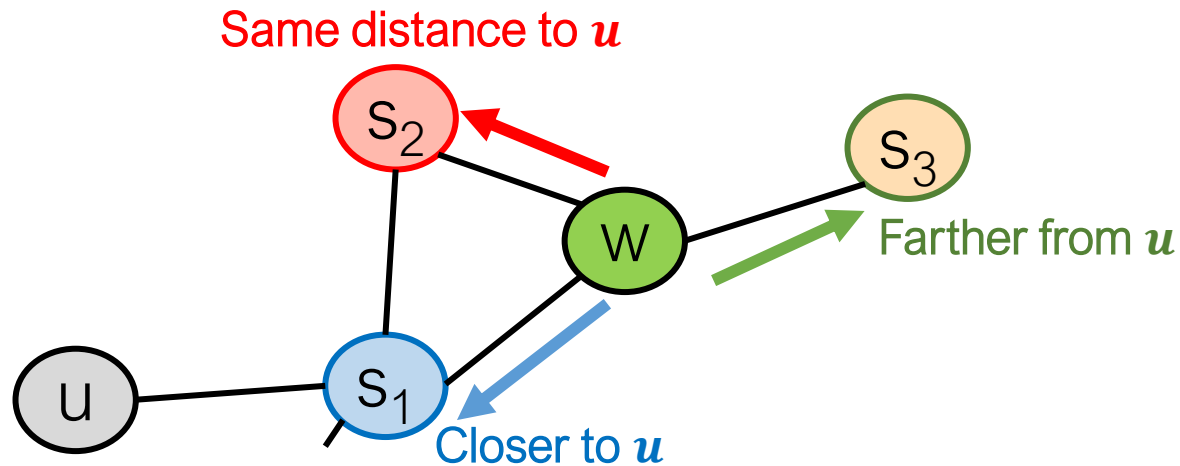
Biased random walk R that given a node u generates neighborhood $N_R(u)$

- Two hyper-parameters:
 - Return hyper-parameter p :
 - Return back to the previous node
 - In-out hyper-parameter q :
 - Moving outwards (DFS) vs. inwards (BFS)

Biased Random Walks

Biased 2nd-order random walks explore network neighborhoods:

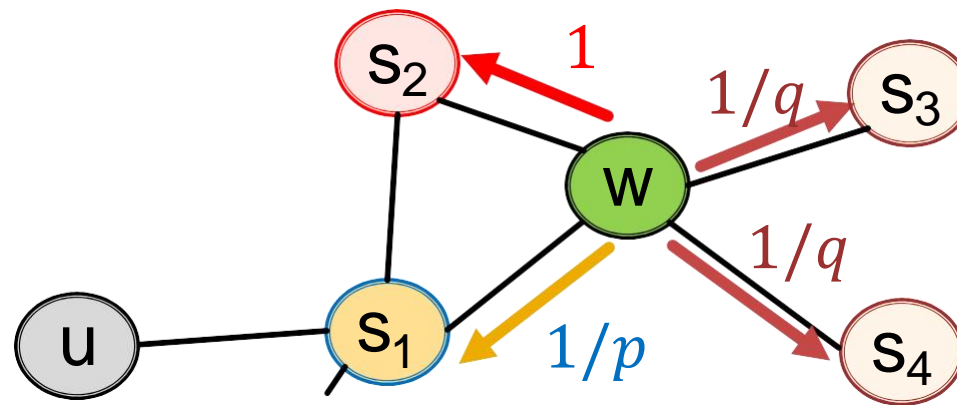
- Random walk started at u and is now at w
- **Insight:** Neighbors of w can only be:



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?



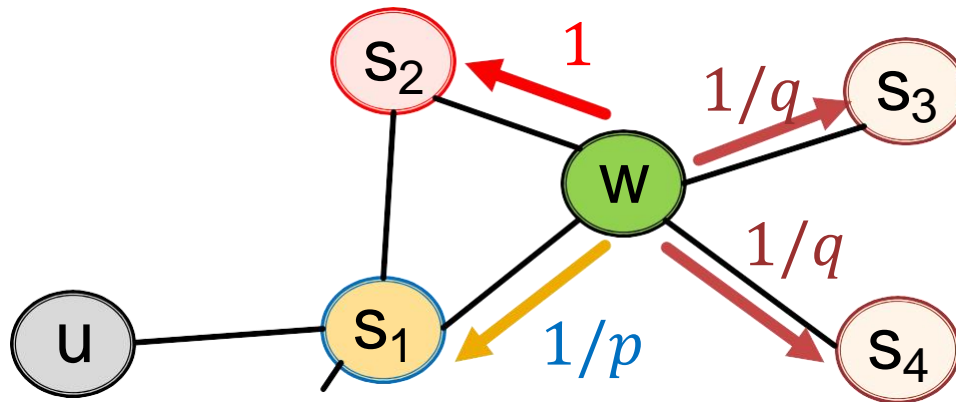
$1/p, 1/q, 1$ are
unnormalized
probabilities

○ 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

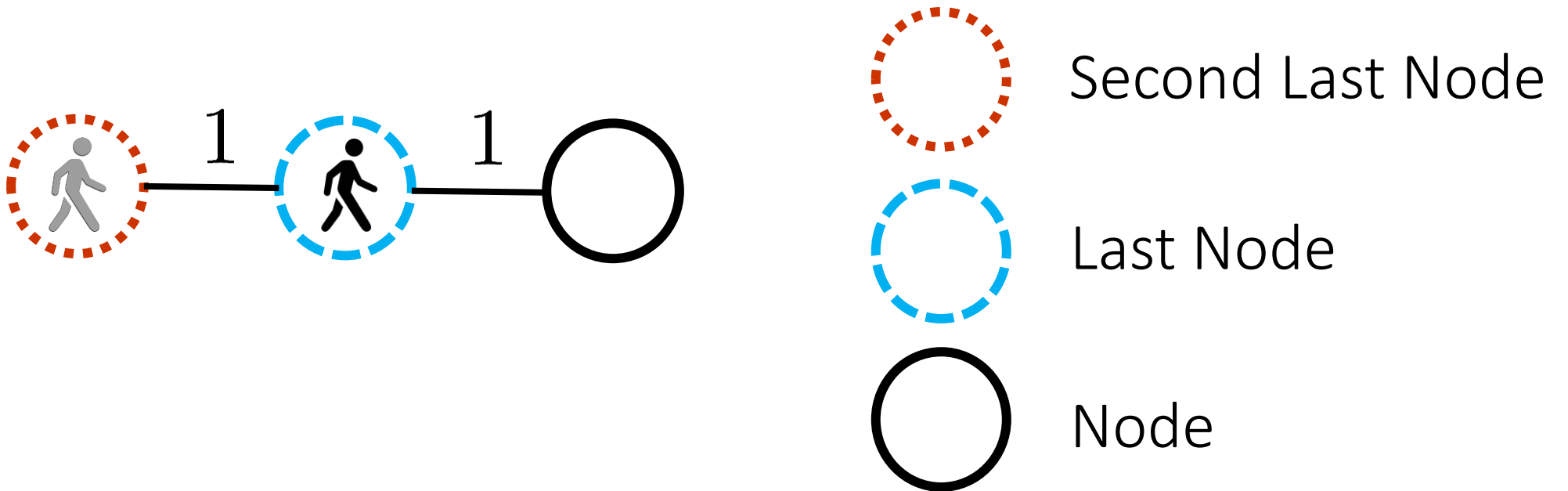
- **BFS-like** walk: Low value of p
- **DFS-like** walk: Low value of q

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

First-order Random Walk

A random walker moves to next node based on the last node.

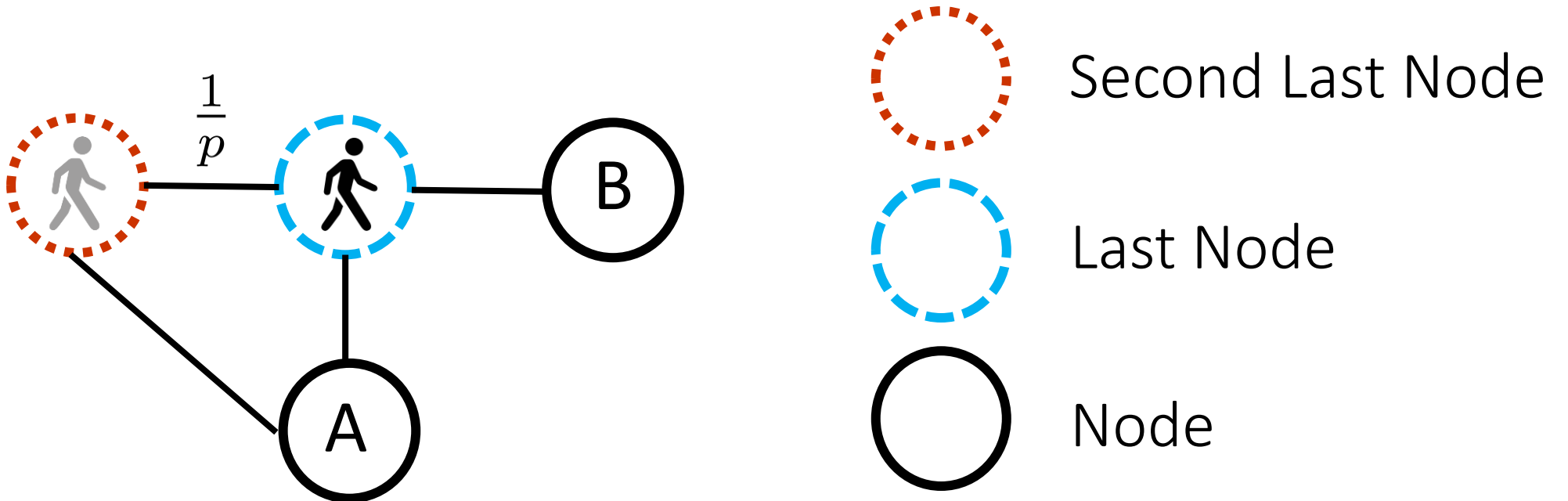
$$\pi_{c_i|c_{i-1},c_{i-2}} = \pi_{c_i|c_{i-1}} = \begin{cases} 1, & \text{if } (c_{i-1}, c_i) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}$$



Second-order Random Walk

A random walker moves to next node based on the last two nodes.

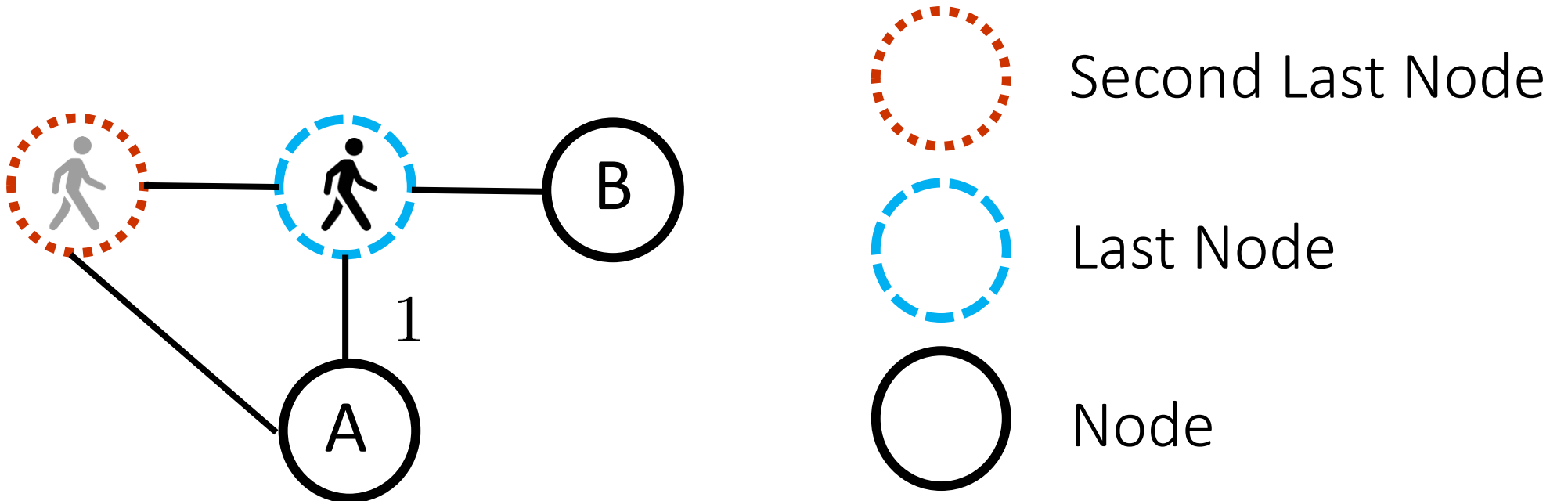
$$\pi_{c_i | c_{i-1}, c_{i-2}} = \begin{cases} \frac{1}{p}, & \text{if } d_{c_{i-2}c_i} = 0 \\ 1, & \text{if } d_{c_{i-2}c_i} = 1 \\ \frac{1}{q}, & \text{if } d_{c_{i-2}c_i} = 2 \end{cases}$$



Second-order Random Walk

A random walker moves to next node based on the last two nodes.

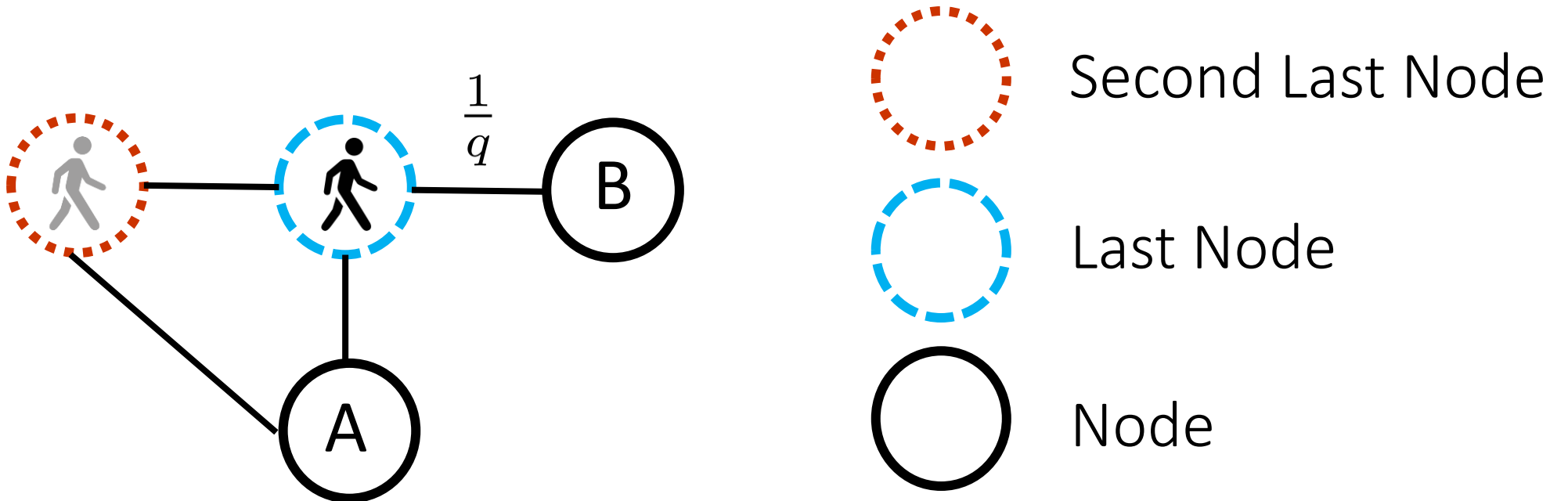
$$\pi_{c_i | c_{i-1}, c_{i-2}} = \begin{cases} \frac{1}{p}, & \text{if } d_{c_{i-2}c_i} = 0 \\ 1, & \text{if } d_{c_{i-2}c_i} = 1 \\ \frac{1}{q}, & \text{if } d_{c_{i-2}c_i} = 2 \end{cases}$$



Second-order Random Walk

A random walker moves to next node based on the last two nodes.

$$\pi_{c_i | c_{i-1}, c_{i-2}} = \begin{cases} \frac{1}{p}, & \text{if } d_{c_{i-2}c_i} = 0 \\ 1, & \text{if } d_{c_{i-2}c_i} = 1 \\ \frac{1}{q}, & \text{if } d_{c_{i-2}c_i} = 2 \end{cases}$$



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**

Summary So Far

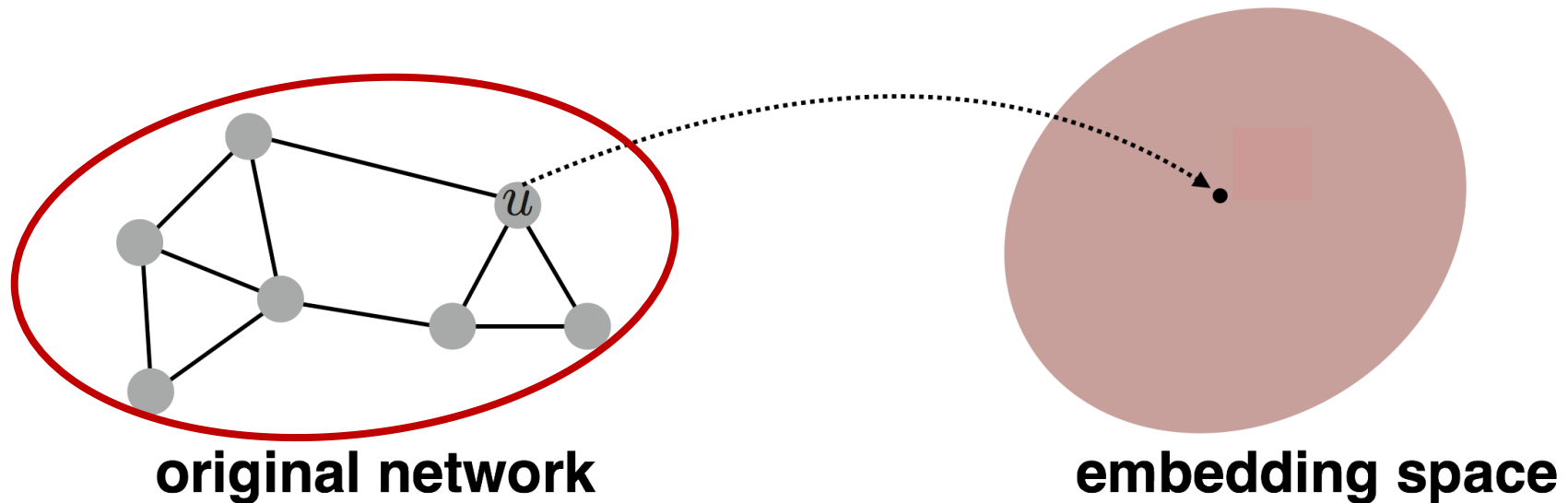
- **Core idea:** Embed nodes so that distances in embedding space reflect nodesimilarities in the original network.
- **Different notions of node similarity:**
 - Naïve: similar if two nodes are connected
 - Neighborhood overlap (covered in Lecture 2)
 - Random walk approaches (**covered today**)

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 alternative methods perform better on link prediction ([Goyal and Ferrara, 2017 survey](#)).
- Random walk approaches are generally more efficient.
- **In general:** Must choose definition of node similarity that matches your application.

Embedding the Entire Graph

- **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

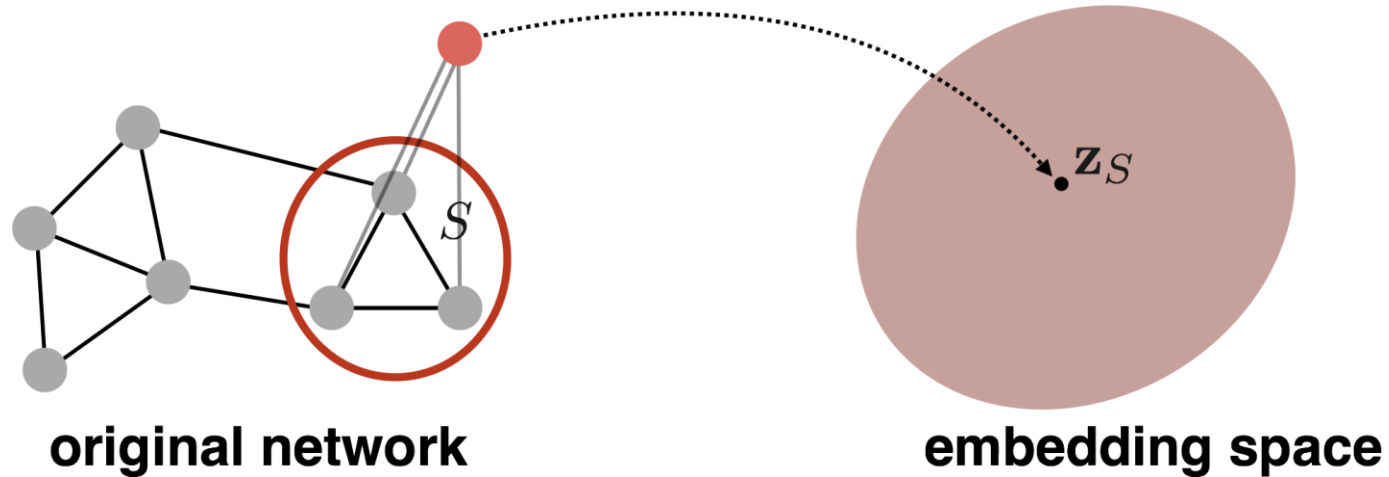
Approach 1

- **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 .

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

Approach 2

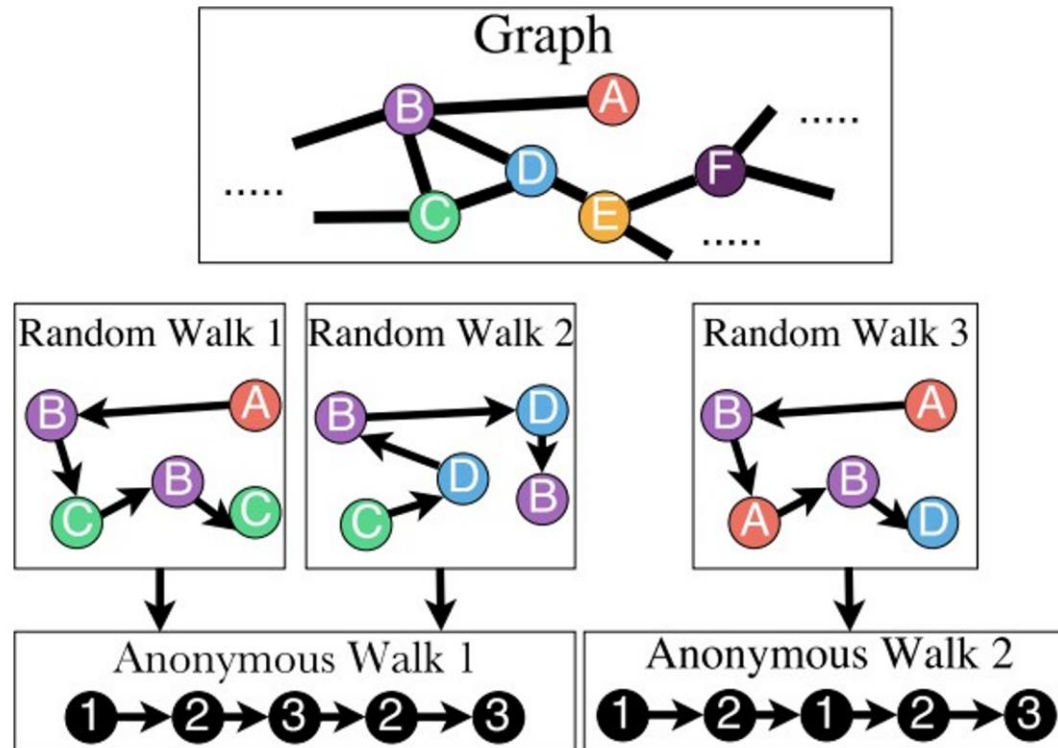
- **Approach 2:** Introduce a “**virtual node**” to represent the (sub)graph and run a standard graph embedding technique



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

Approach 3: Anonymous Walk Embeddings

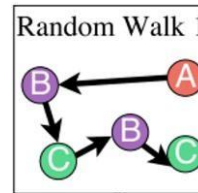
- States in anonymous walks correspond to the index of the first time we visited the node in a random walk



Approach 3: Anonymous Walk Embeddings

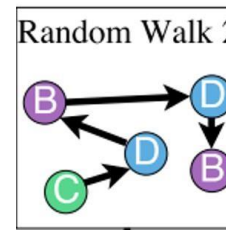
- Agnostic to the identity of the nodes visited (hence anonymous)

- **Example:** Random walk w_1 :

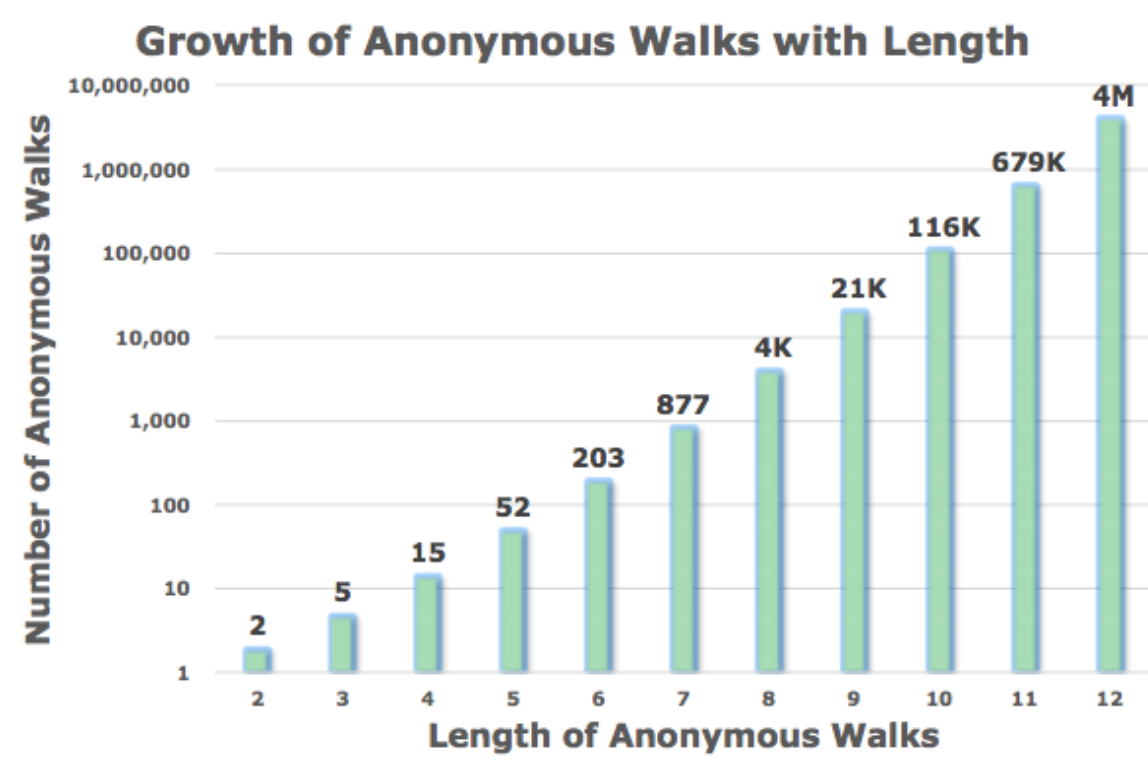


- Step 1: node A → node 1
- Step 2: node B → node 2 (different from node 1)
- Step 3: node C → node 3 (different from node 1, 2)
- Step 4: node B → node 2 (same as the node in step 2)
- Step 5: node C → node 3 (same as the node in step 3)

Note: Random walk w_2 gives the same anonymous walk:



Number of Anonymous Walks Grows



Number of anonymous walks grows exponentially:

- There are 5 anon. walks w_i of length 3:

$$w_1=111, w_2=112, w_3=121, w_4=122, w_5=123$$

Simple Use of Anonymous Walks

- 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
 - $\mathbf{z}_G[i] =$ probability of anonymous walk w_i in graph G .

Sampling Anonymous 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 :

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

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

Summary

- We discussed 3 ideas to graph embeddings:
 - **Approach 1:** Embed nodes and sum/avg them
 - **Approach 2:** Create super-node that spans the (sub) graph and then embed that node.
 - **Approach 3:** Anonymous Walk Embeddings
 - **Idea 1:** Sample the anon. walks and represent the graph as fraction of times each anon walk occurs.
 - **(Idea 2: Learn graph embedding together with anonymous walk embeddings.)**

How to Use Embeddings \mathbf{z}_i of nodes

- **Clustering/community detection:** Cluster points \mathbf{z}_i
- **Node classification:** Predict label of node i based on \mathbf{z}_i
- **Link prediction:** Predict edge (i, j) based on $(\mathbf{z}_i, \mathbf{z}_j)$
 - Where we can: concatenate, avg, product, or take a difference between the embeddings:
 - Concatenate: $f(\mathbf{z}_i, \mathbf{z}_j) = g([\mathbf{z}_i, \mathbf{z}_j])$
 - Hadamard: $f(\mathbf{z}_i, \mathbf{z}_j) = g(\mathbf{z}_i * \mathbf{z}_j)$ (per coordinate product)
 - Sum/Avg: $f(\mathbf{z}_i, \mathbf{z}_j) = g(\mathbf{z}_i + \mathbf{z}_j)$
 - Distance: $f(\mathbf{z}_i, \mathbf{z}_j) = g(\|\mathbf{z}_i - \mathbf{z}_j\|_2)$
- **Graph classification:** Graph embedding \mathbf{z}_G via aggregating node embeddings or anonymous random walks.
- Predict label based on graph embedding \mathbf{z}_G .

Summary

- We discussed graph representation learning, a way to learn node and graph embeddings for downstream tasks, without feature engineering.
- **Encoder-decoder framework:**
 - Encoder: embedding lookup
 - Decoder: predict score based on embedding to match node similarity
- **Node similarity measure: (biased) random walk**
 - Examples: DeepWalk, Node2Vec
- **Extension to Graph embedding: Node embedding aggregation and Anonymous Walk Embeddings**