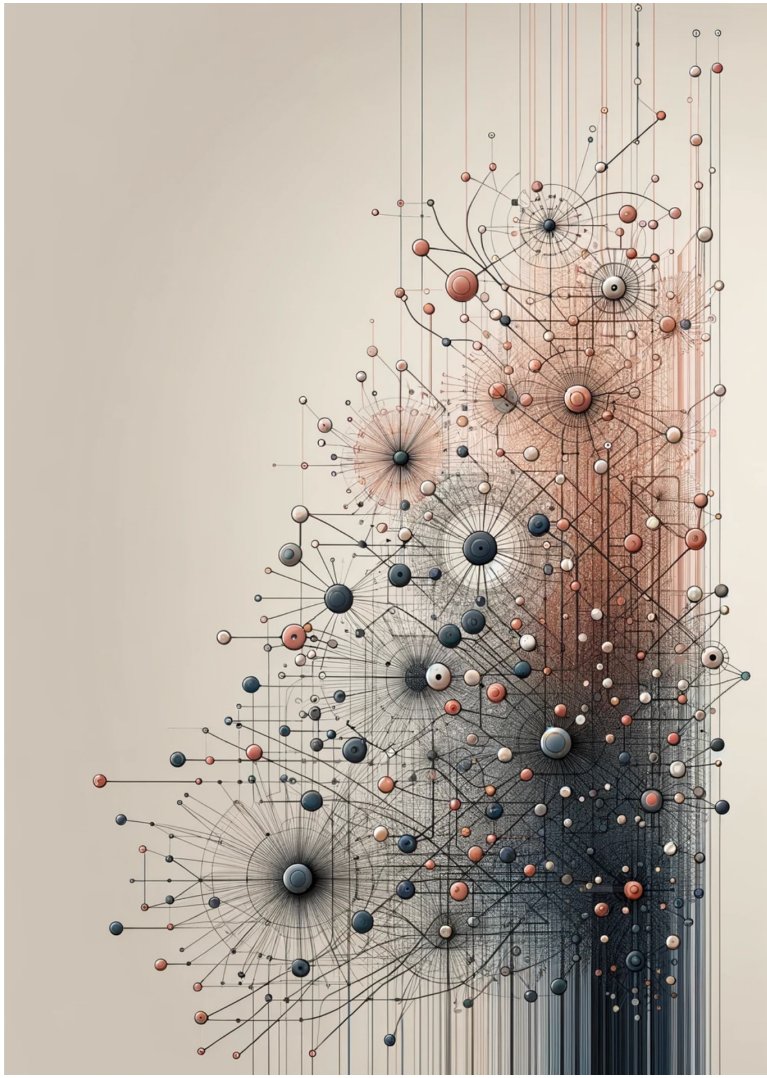
UNIVERSITÀ DI TORINO

Analisi e Visualizzazione delle Reti Complesse

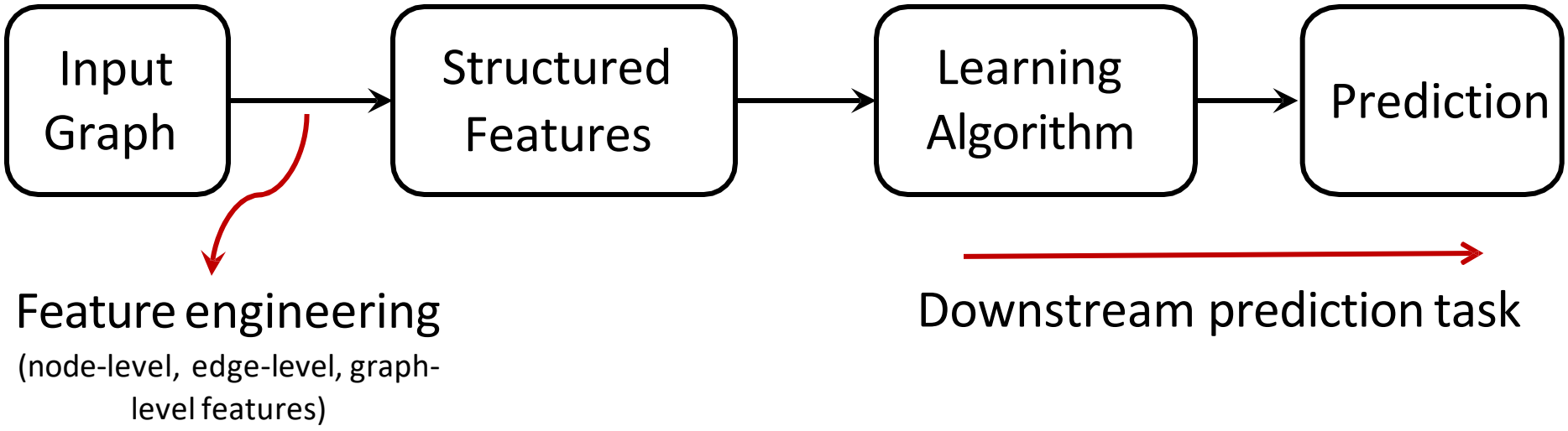
NS23 - Representation Learning on Graphs

Prof. Rossano Schifanella



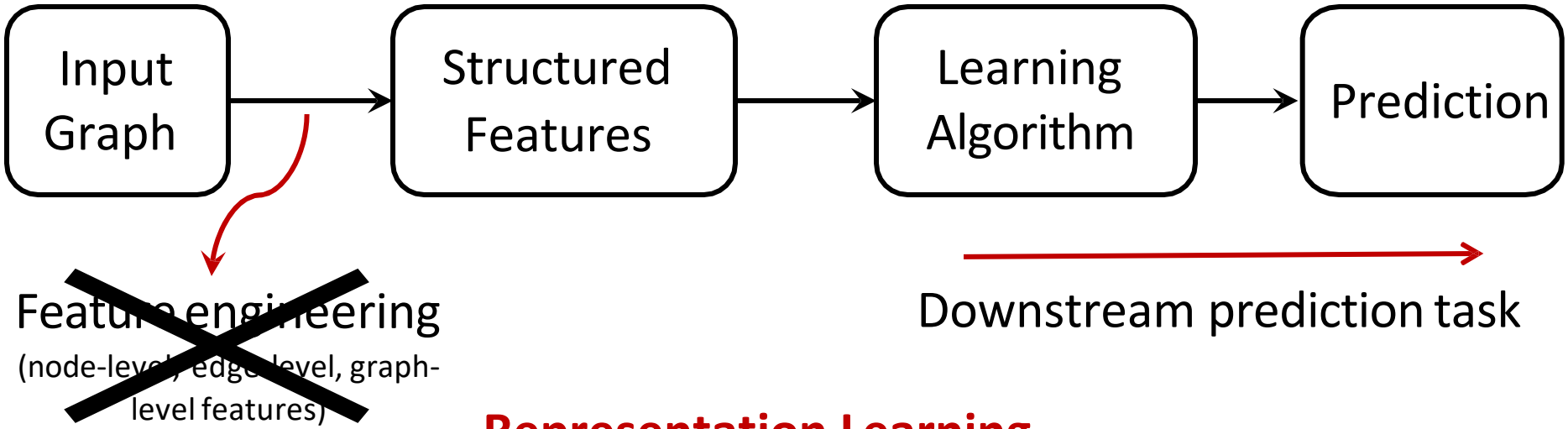
Recap: Traditional ML for Graphs

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



Graph Representation Learning

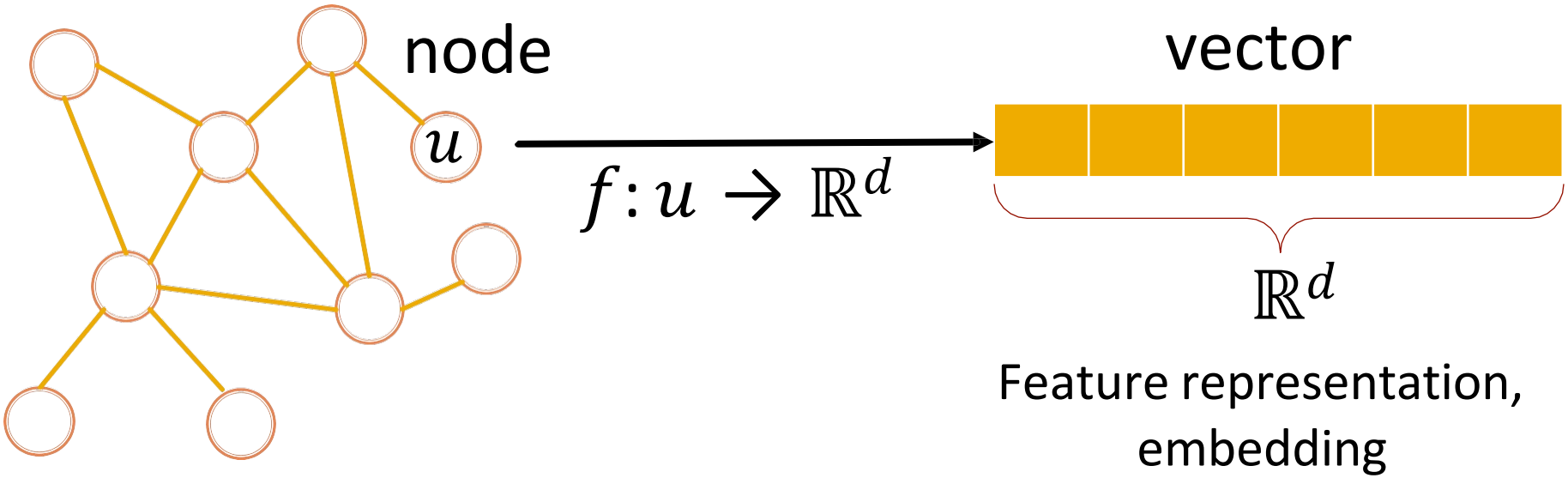
Graph Representation Learning alleviates the need to do feature engineering every single time.



Representation Learning automatically learn the features

Graph Representation Learning

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

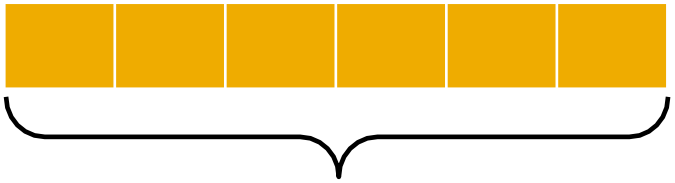


Why embeddings?

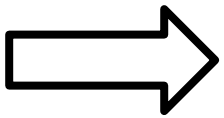
Task: Map nodes into an embedding space

- The **similarity of embeddings between nodes indicates their similarity in the network**.
- For example, two nodes are close to each other if they are connected by an edge.
- Encoded network information is potentially used for many downstream predictions.

Vec



embeddings $\mathbb{R}^d$



Tasks

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

Example Node Embedding

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

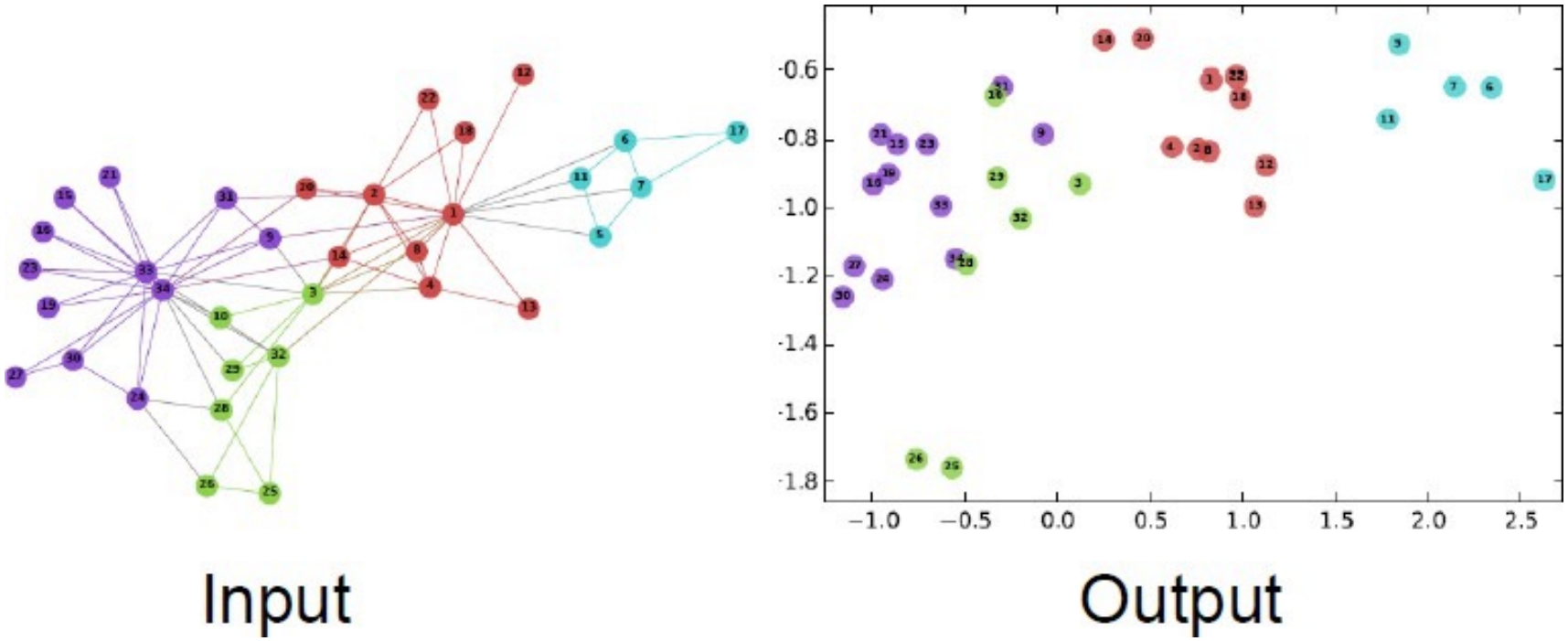
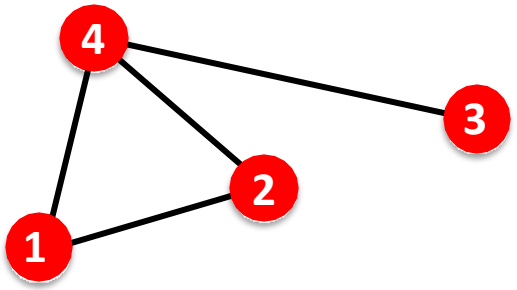


Image from: Perozzi et al. DeepWalk: Online Learning of Social Representations. *KDD 2014.*

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.**

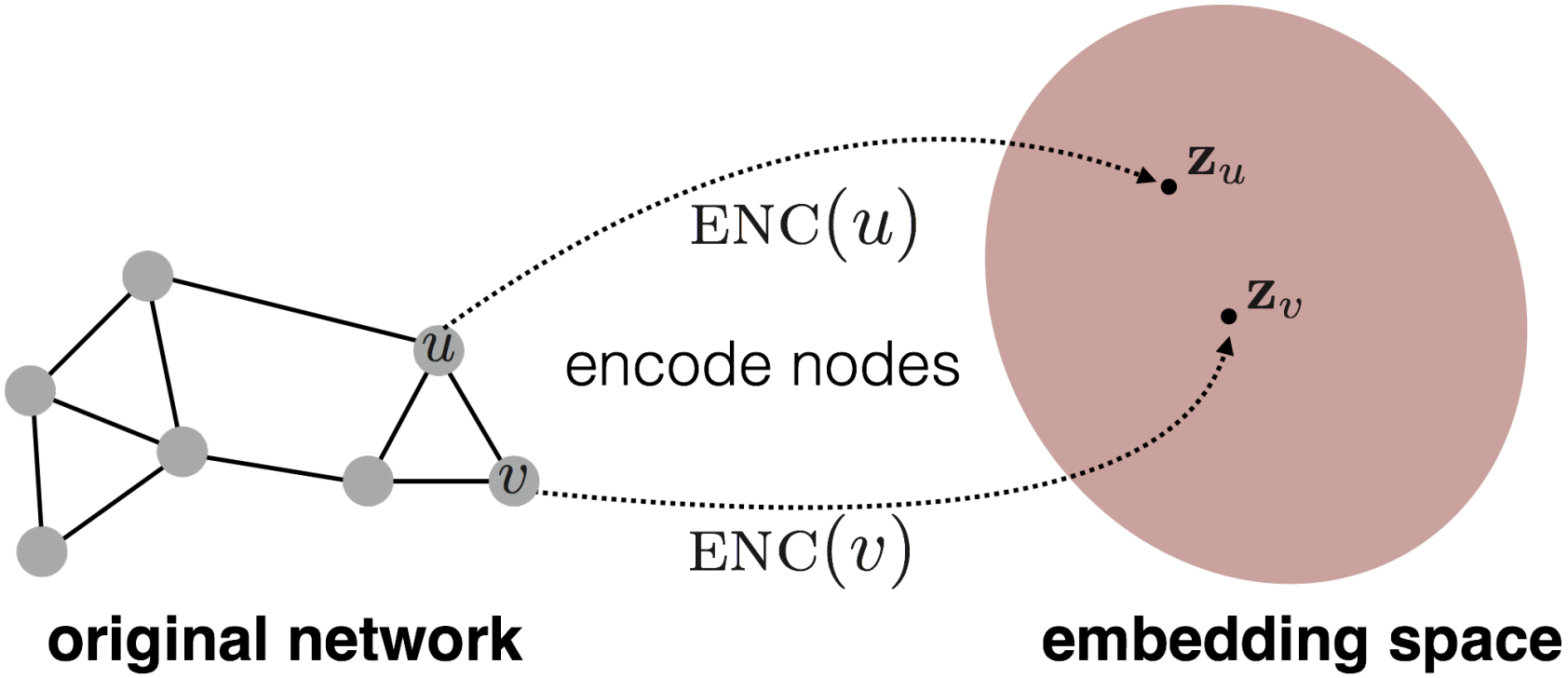


V: {1, 2, 3, 4}

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

Embedding Nodes

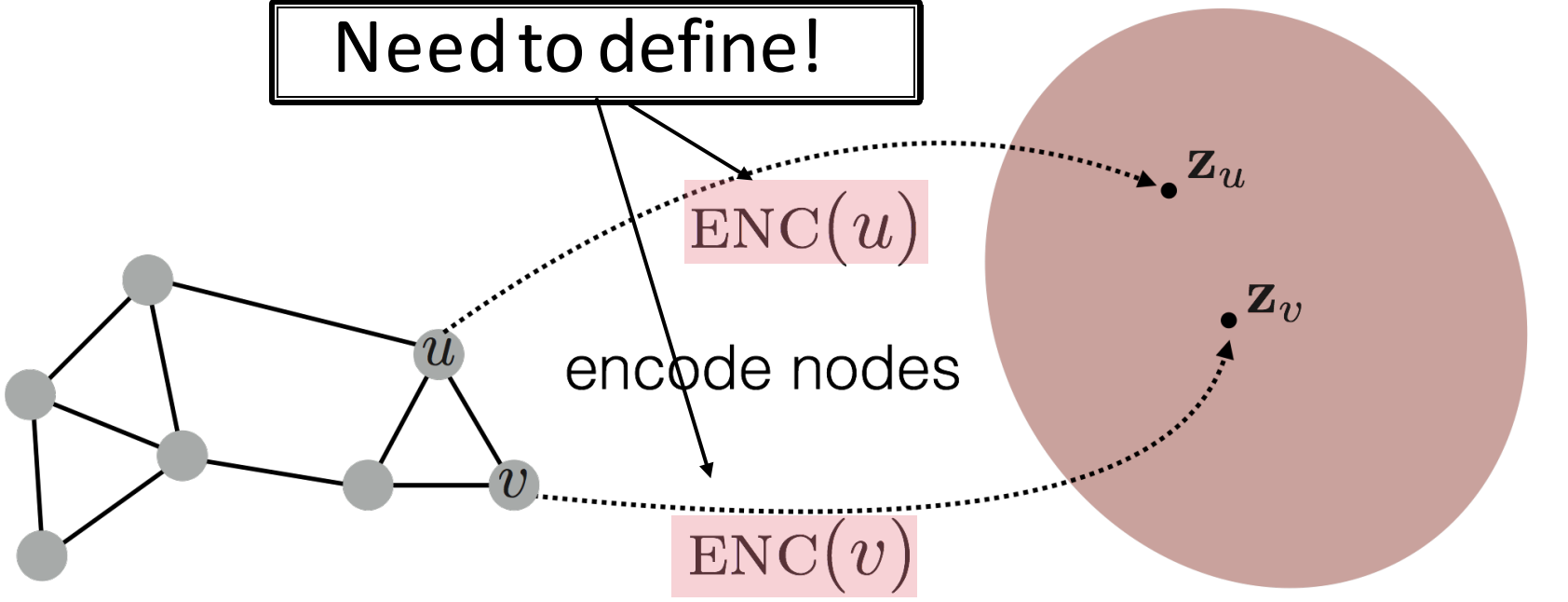
The goal is to **encode nodes so that similarity in the embedding space** (e.g., dot product) **approximates similarity in the graph.**



Embedding Nodes

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

Need to define!



Learning Node Embeddings

1. **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

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

in the original network — Similarity of the embedding

Two keys components

Encoder: maps each node to a low-dimensional vector

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

$\mathbf{z}_v$: ***d*-dimensional embedding**

v: **node in the input graph**

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

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

Decoder

$\text{similarity}(u, v)$: **Similarity of u and v in the original network**

$\mathbf{z}_v^\top \mathbf{z}_u$: **dot product between node embeddings**

Shallow encoding

Simplest encoding approach: **Encoder is just an embedding-lookup**

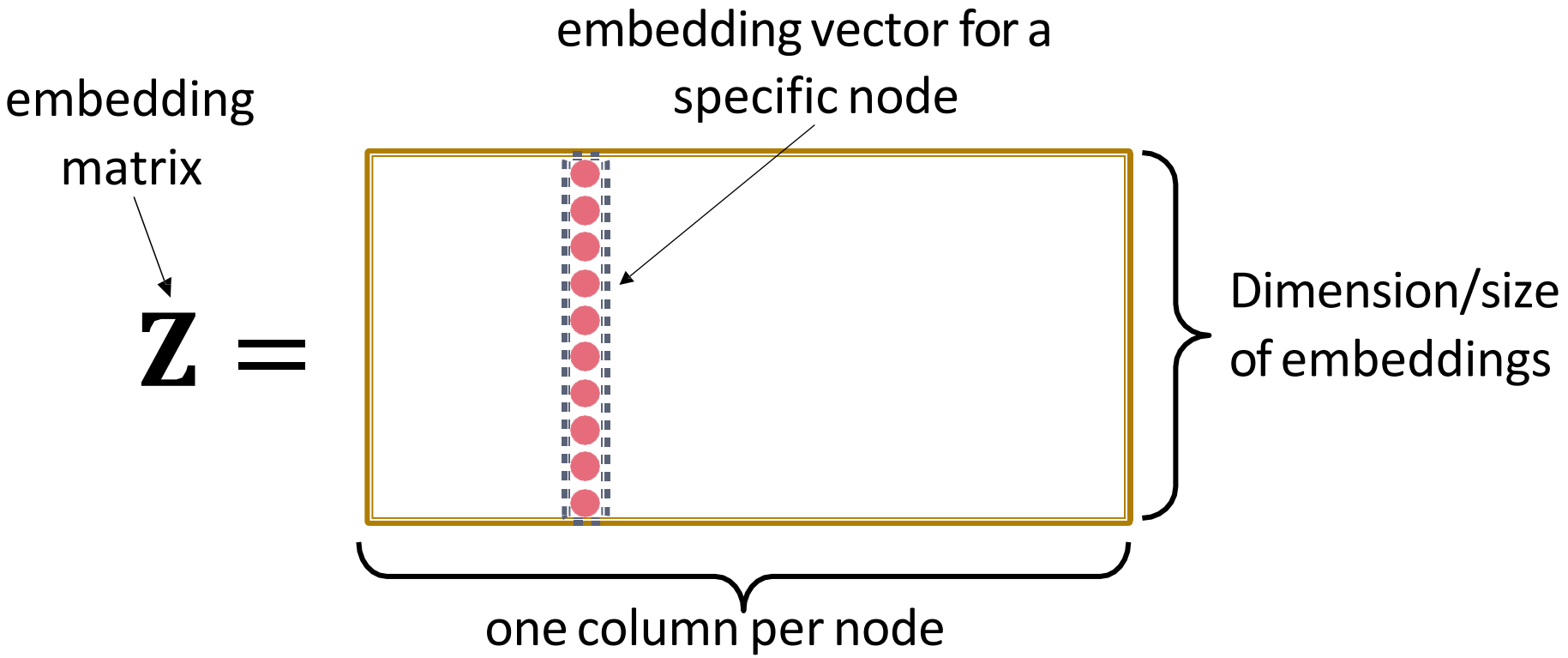
$$\text{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ 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 the column indicating node v

Shallow encoding

Simplest encoding approach: **encoder is just an embedding-lookup**



Shallow encoding

Simplest encoding approach: **Encoder is just an embedding-lookup**

Each node is assigned a unique embedding vector (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 $u \in V$
- Deep encoders: GNNs (another class!)

- **Decoder:** based on node similarity.
- **Objective:** maximize $\mathbf{z}_v^\top \mathbf{z}_u$ for node pairs (u, v) that are **similar**

How to define node similarity

- The 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 use a node similarity definition that uses **random walks** and discuss how to optimize embeddings for such a similarity measure.

Note on Node Embeddings

This is an **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 the DEC) is preserved.

These embeddings are **task-independent**

- They are not trained for a specific task but can be used for any task.

Notation

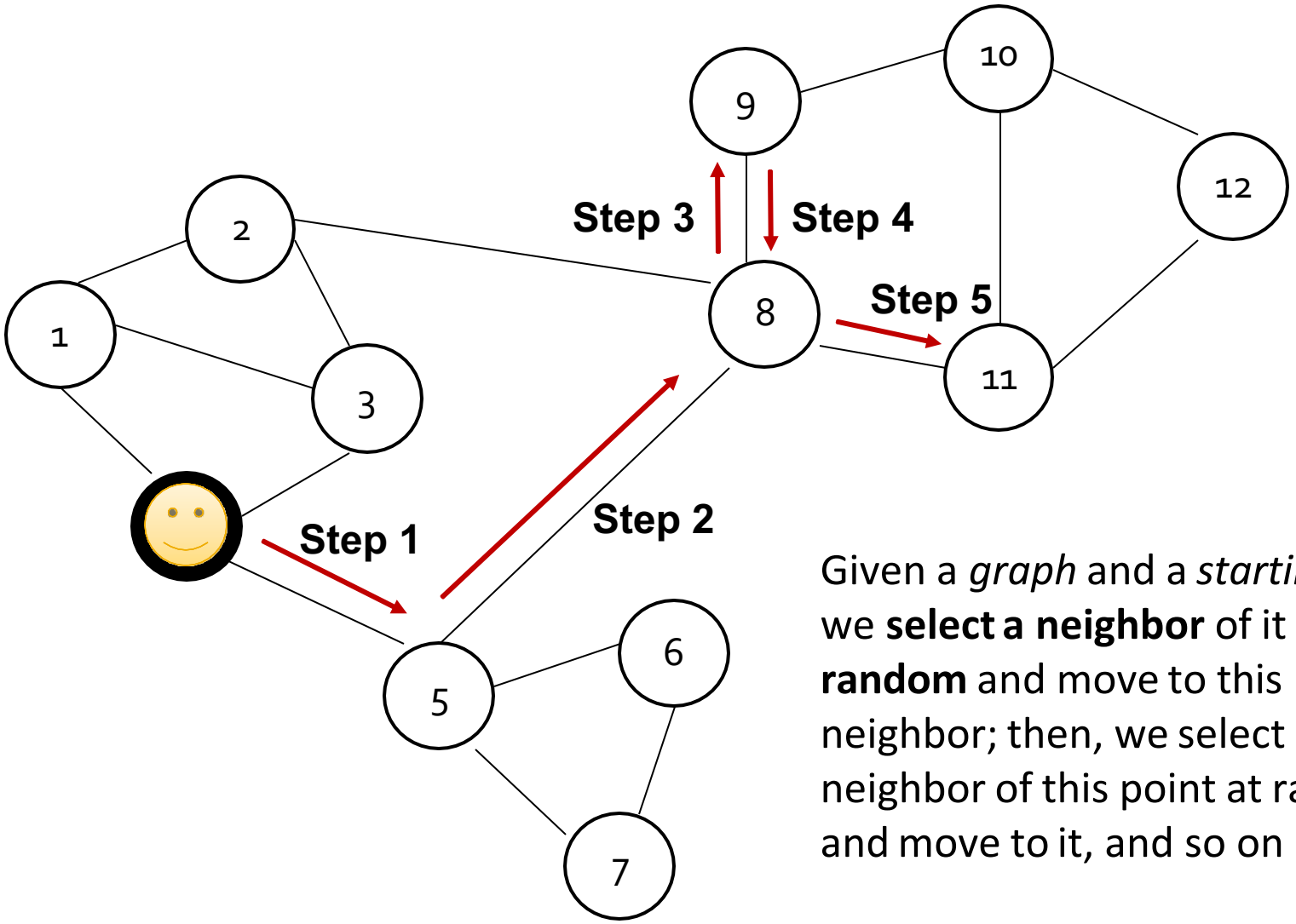
Vector $\mathbf{z}_u$:

- The embedding of node u (what we aim to find).

Probability $P(\ v \mid \mathbf{z}_u\ )$: ⟸ Our model prediction based on $\mathbf{z}_u$

- The **(predicted) probability** of visiting node v on random walks starting from node u.

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, and so on

The sequence of points visited this way is a **random walk on the graph**.

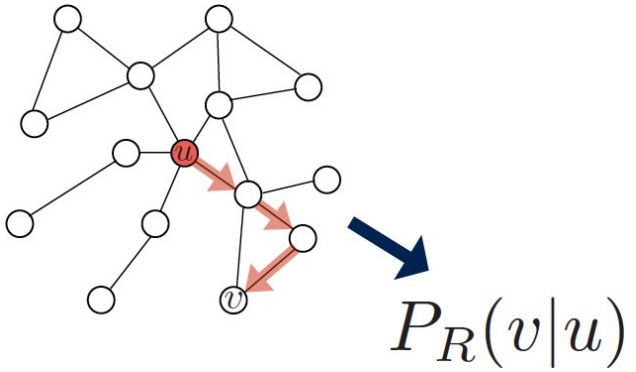
Random-Walk Embeddings

$$\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v \approx$$

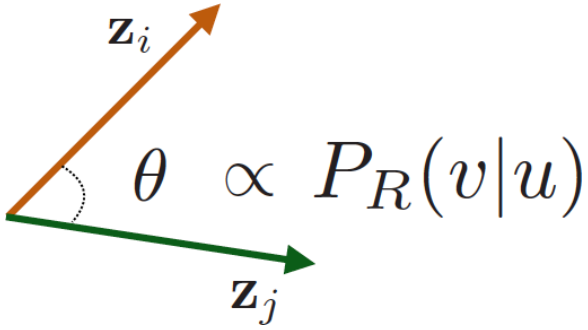
probability that u and v co-occur on a random walk over the graph

Random-Walk Embeddings

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



2. Optimize embeddings to encode these random walk statistics:



Similarity in embedding space (Here: dot product=cos(θ)) encodes random walk "similarity"

Why Random Walks?

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

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 $N_R(u)$.

Randon 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 $N_{\mathrm{R}}(u)$.

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

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

Random Walks: 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 random walk?

So far we have described how to optimize embeddings given a random walk strategy *R*

What strategies should we use to run these random walks?

- Simplest idea: **Just run fixed-length, unbiased random walks starting from each node** (i.e., DeepWalk from Perozzi et al., 2013)
  - The issue is that such notion of similarity is too constrained

How can we generalize this?

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

Overview of node2vec

Goal: Embed nodes with similar network neighborhoods close in the feature space.

We frame this goal as a maximum likelihood optimization problem, independent of the downstream prediction task.

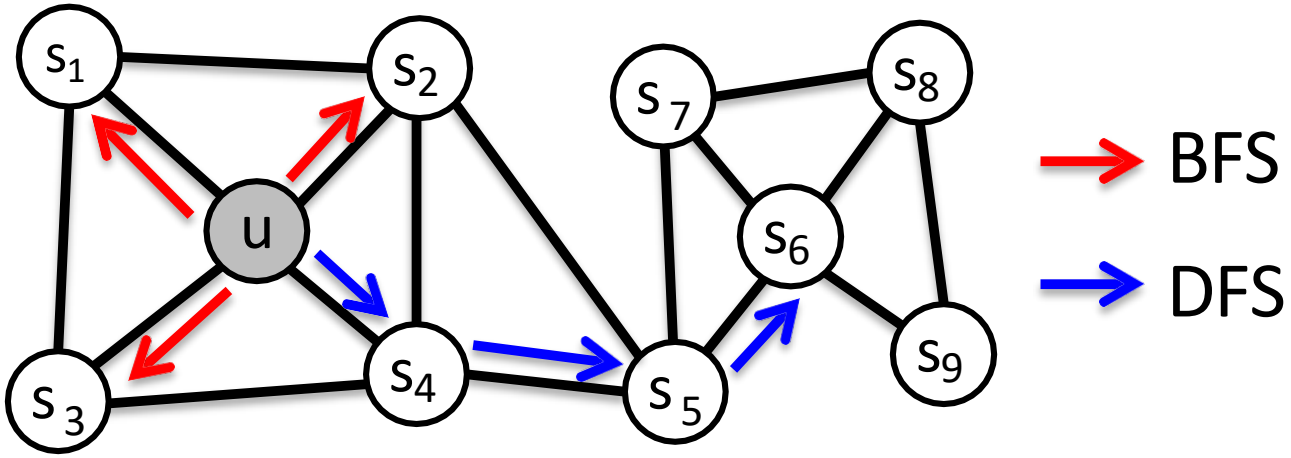
Key observation: Flexible notion of network neighborhood $N_R(u)$ of node u leads to rich node embeddings

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

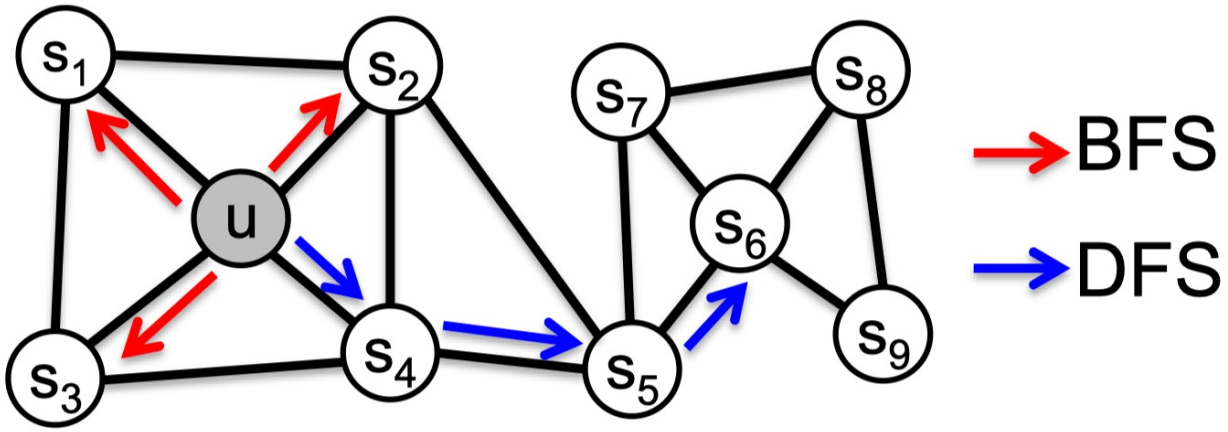
Reference: Grover et al. 2016. node2vec: Scalable Feature Learning for Networks. *KDD.*

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:

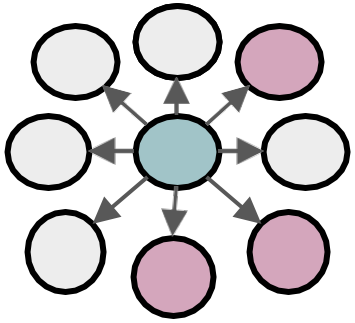


Walk of length 3 ($N_R(u)$ of size 3):

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

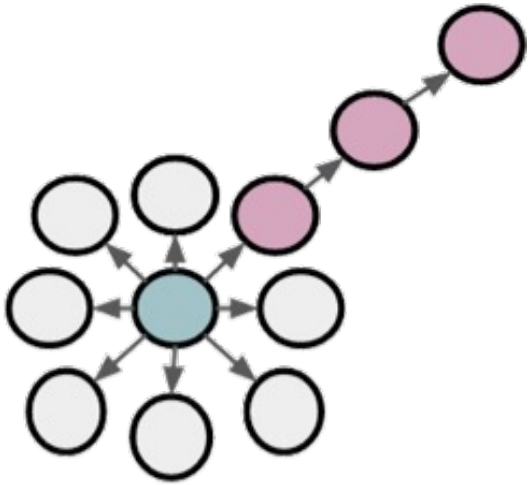
$N_{DFS}(u) = \{ s_4, s_5, s_6 \}$ **Global** macroscopic view

BFS vs DFS



BFS:

Micro-view of neighbourhood



DFS:

Macro-view of neighbourhood

Interpolating BFS and DFS

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

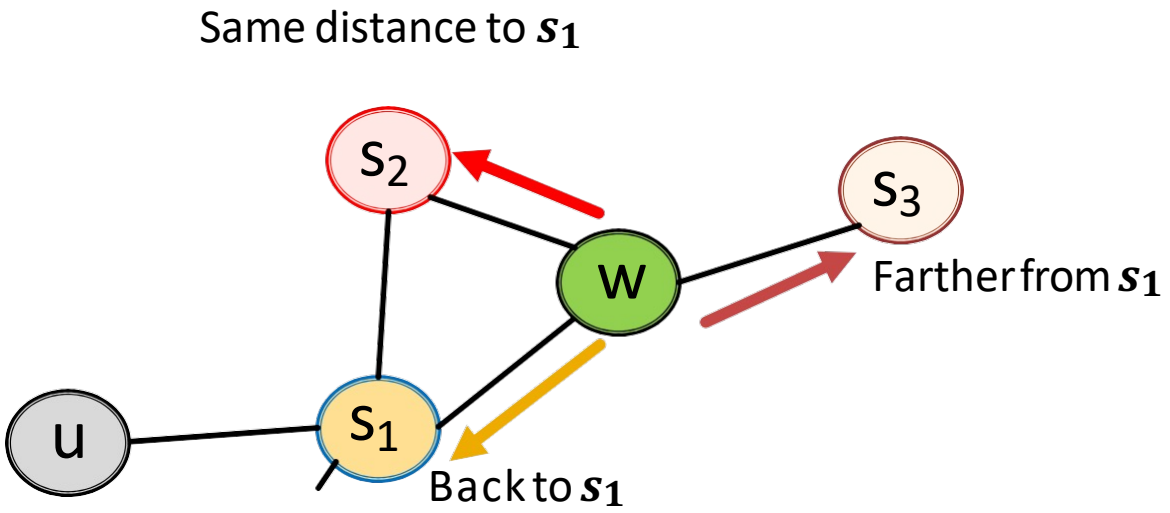
Two parameters:

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

Biased Random Walks

Biased 2nd-order random walks explore network neighborhoods:

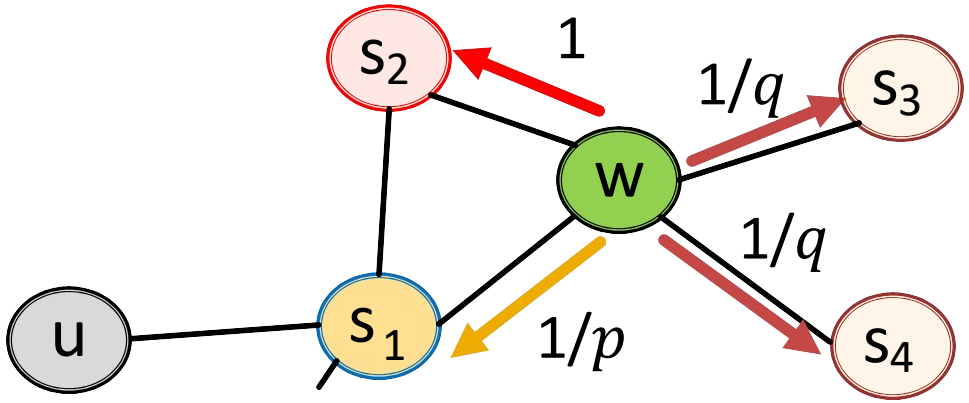
- Random walk just traversed edge (s_1, w) and is now at w
- **Insight:** Neighbors of w can only be:



Idea: Remember where the walk came from

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

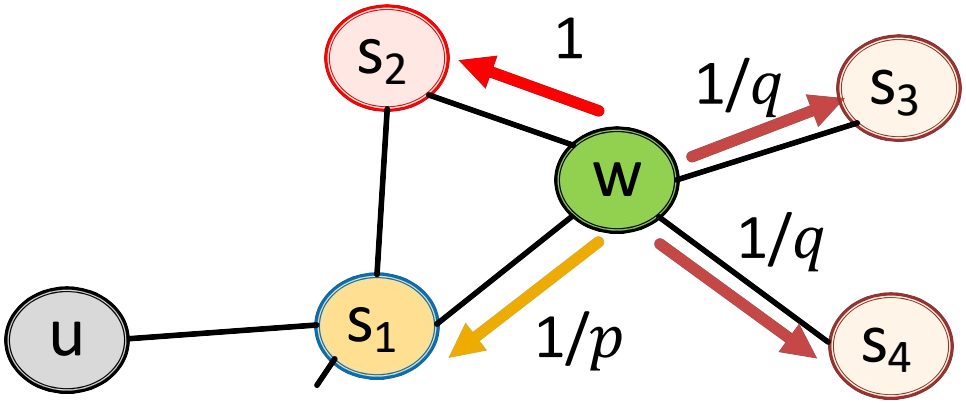
p, q model transition probabilities

- 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 →

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

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

Other Random Walk Ideas

Different kinds of biased random walks:

- Based on node attributes (Dong et al., 2017).
- Based on 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).

Summary so far

- **Core idea:** embed nodes so that distances in embedding space reflect node similarities in the original network.

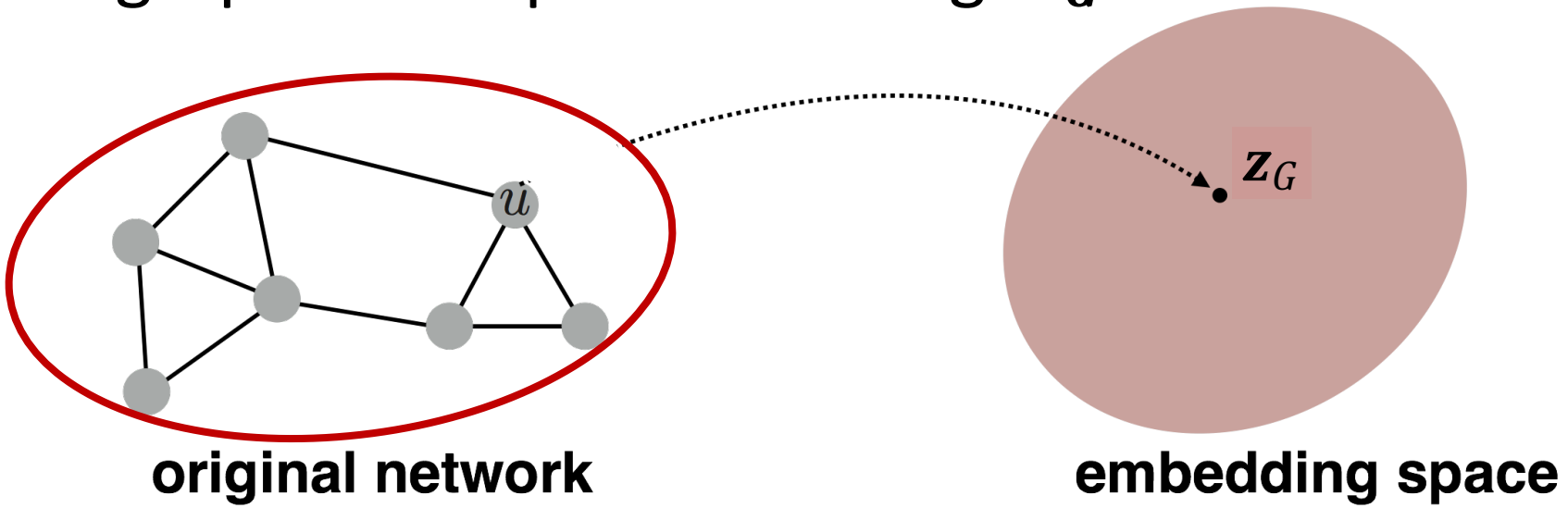
- **Different notions of node similarity:**
  - Naïve: Similar if two nodes are connected
  - Neighborhood overlap
  - Random walk approaches

Summary so far

- **So what method should I use..?**
- No one method wins in all cases….
  - For example, 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:** You must choose the definition of node similarity that matches your application.

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

Approach 1

Simple (but effective) approach 1:

Run a standard graph embedding technique *on* the (sub)graph G.

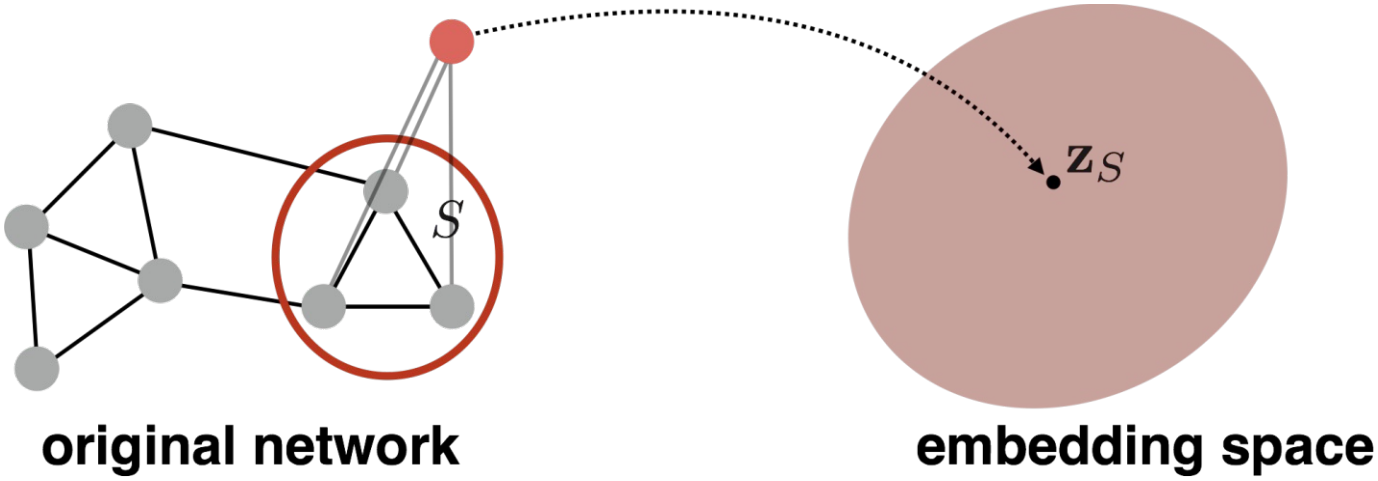
Then just sum (or average) the node embeddings in the (sub)graph G.

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

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

Approach 2

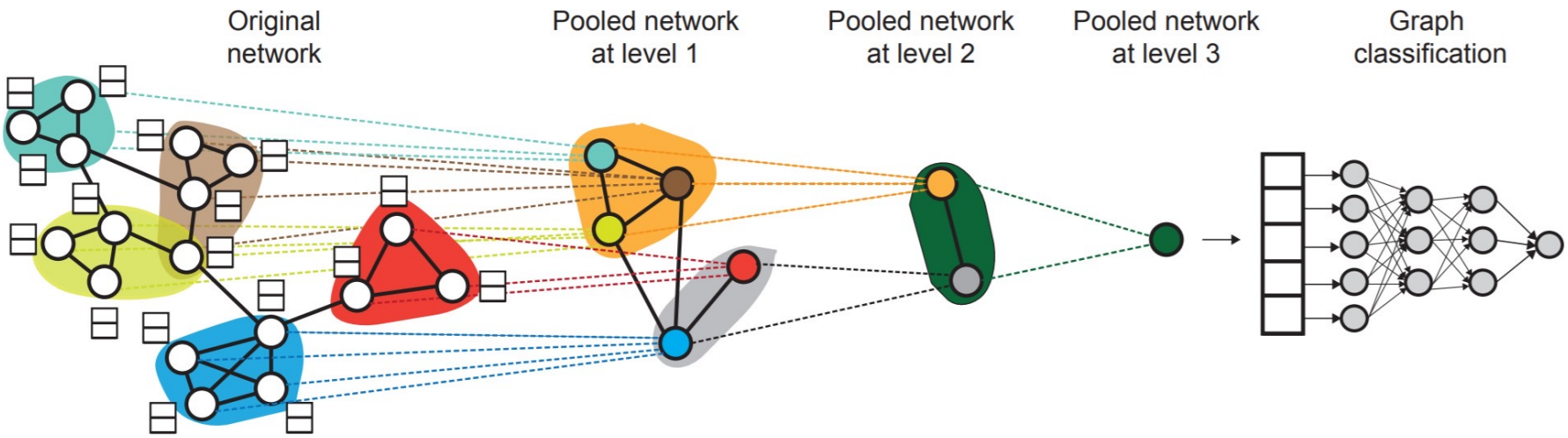
Approach 2: Introduce a **"virtual node"** to represent the (sub)graph and run a standard graph embedding technique



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

Preview: Hierarchical Embeddings

We can **hierarchically** cluster nodes in graphs and sum/average the node embeddings according to these clusters.



How to use embeddings

How to use embeddings z_i of nodes:

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

Today's 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

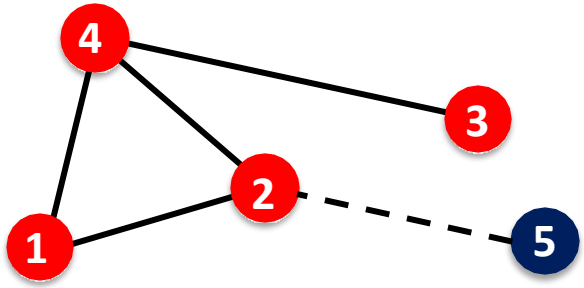
- Examples: DeepWalk, Node2Vec

Extension to Graph embedding: Node embedding aggregation

Limitation (1)

Limitations of node embeddings via matrix factorization and random walks

- Cannot obtain embeddings for nodes not in the training set



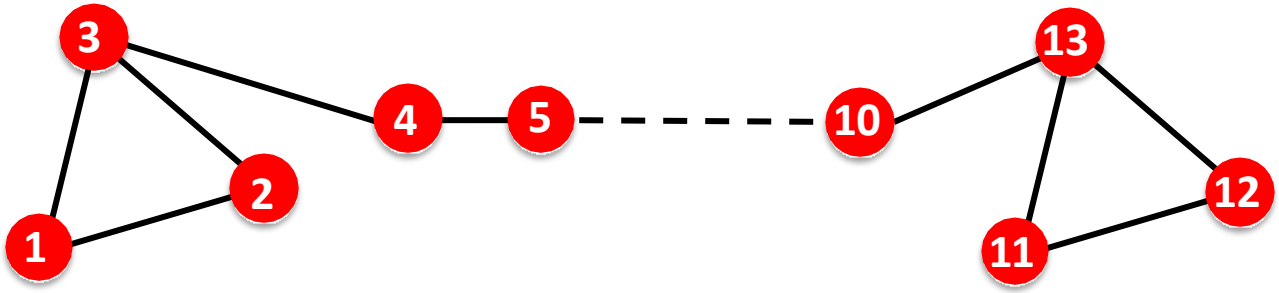
Training set

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.

Limitation (2)

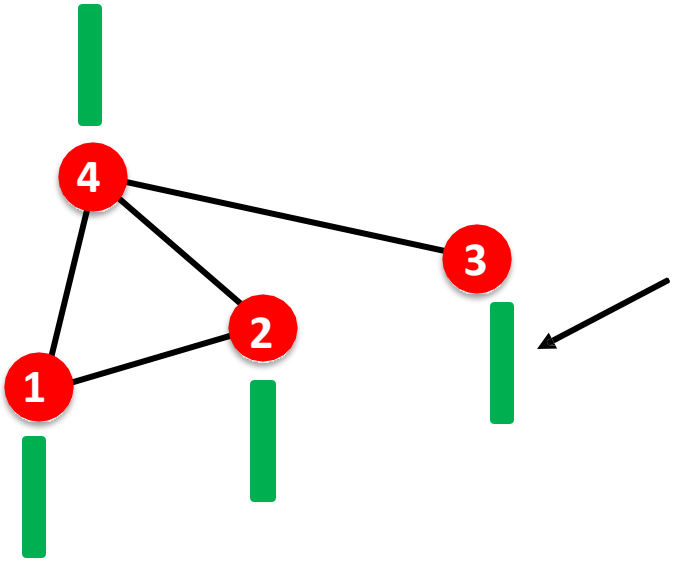
Cannot capture **structural similarity**:



- Nodes 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.**

Limitation (3)

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

UNIVERSITÀ DI TORINO

Q&A