Yale

Distributed Node Embeddings

CPSC483: Deep Learning on Graph-Structured Data

Rex Ying

Outline of Today's Lecture

1. **Distributed Node Embeddings**

2. **Random Walk Approaches for Node Embeddings**

3. **Embedding Entire Graphs**

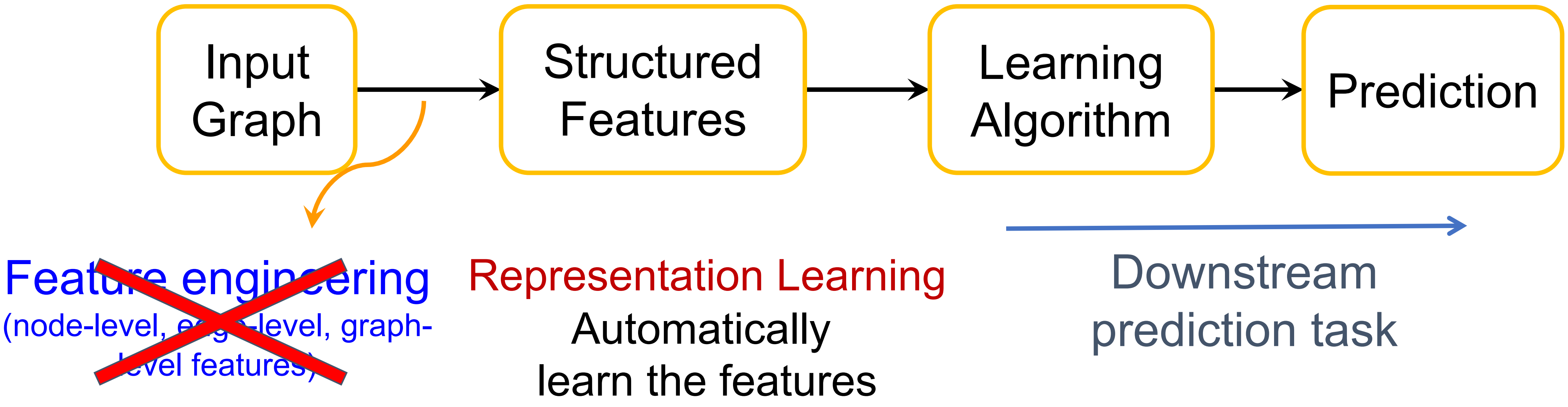
Outline of Today's Lecture

1. **Distributed Node Embeddings**
2. **Random Walk Approaches for Node Embeddings**
3. **Embedding Entire Graphs**

Recap: Graph Representation Learning

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

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



Recap: Learning Node Embeddings

An **unsupervised** setting for learning node embeddings

1. **Encoder** **ENC** 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:**

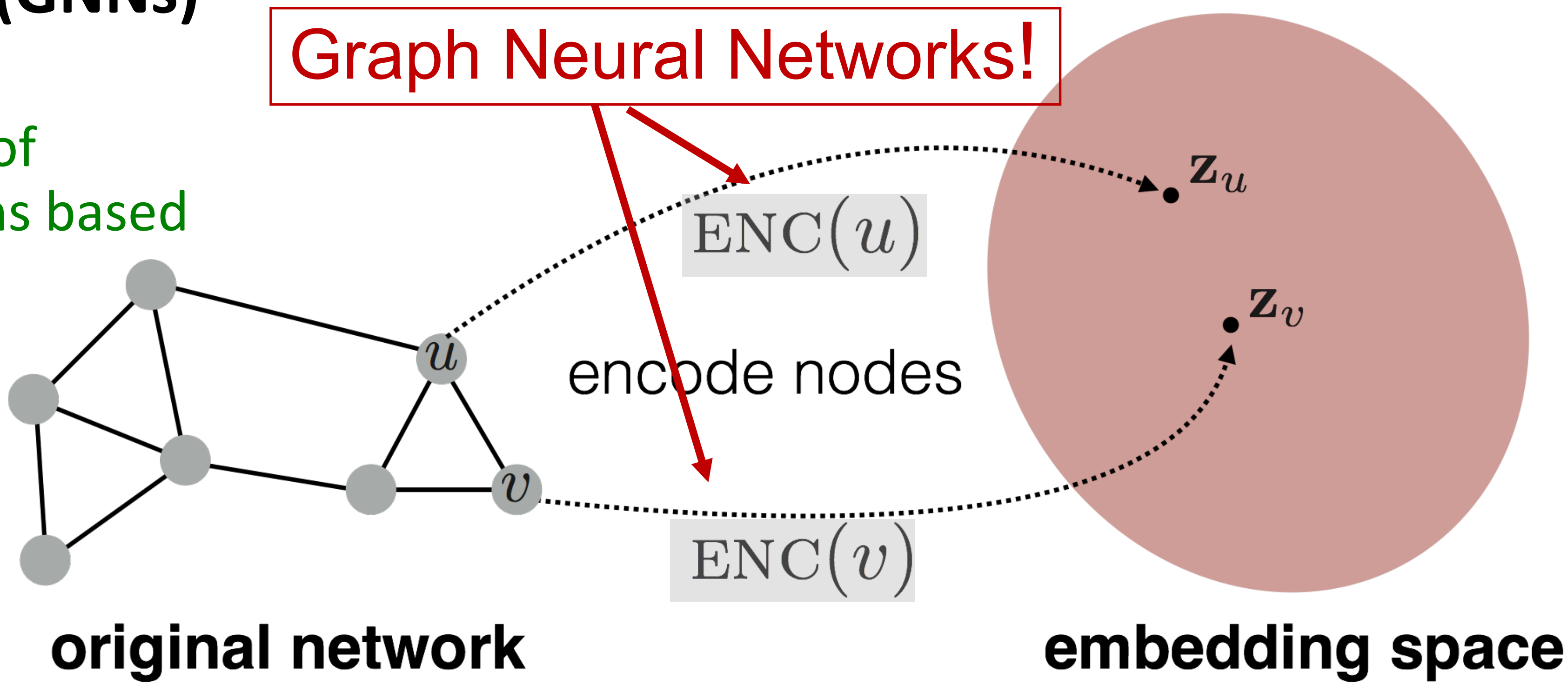
$$\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$$

in the original network   Similarity of the embedding

Recap: GNN for Node Embeddings

- **graph neural networks (GNNs) encoder:**

  $\mathrm{ENC}(\cdot) =$ multiple layers of non−linear transformations based on graph structures



Today: **“Shallow” Encoding!**

“Shallow” Encoding (1)

- Consider a node v in a graph’s nodes set $\mathcal{V}$
- Simplest encoding approach: **Encoder is just an embedding-lookup**

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

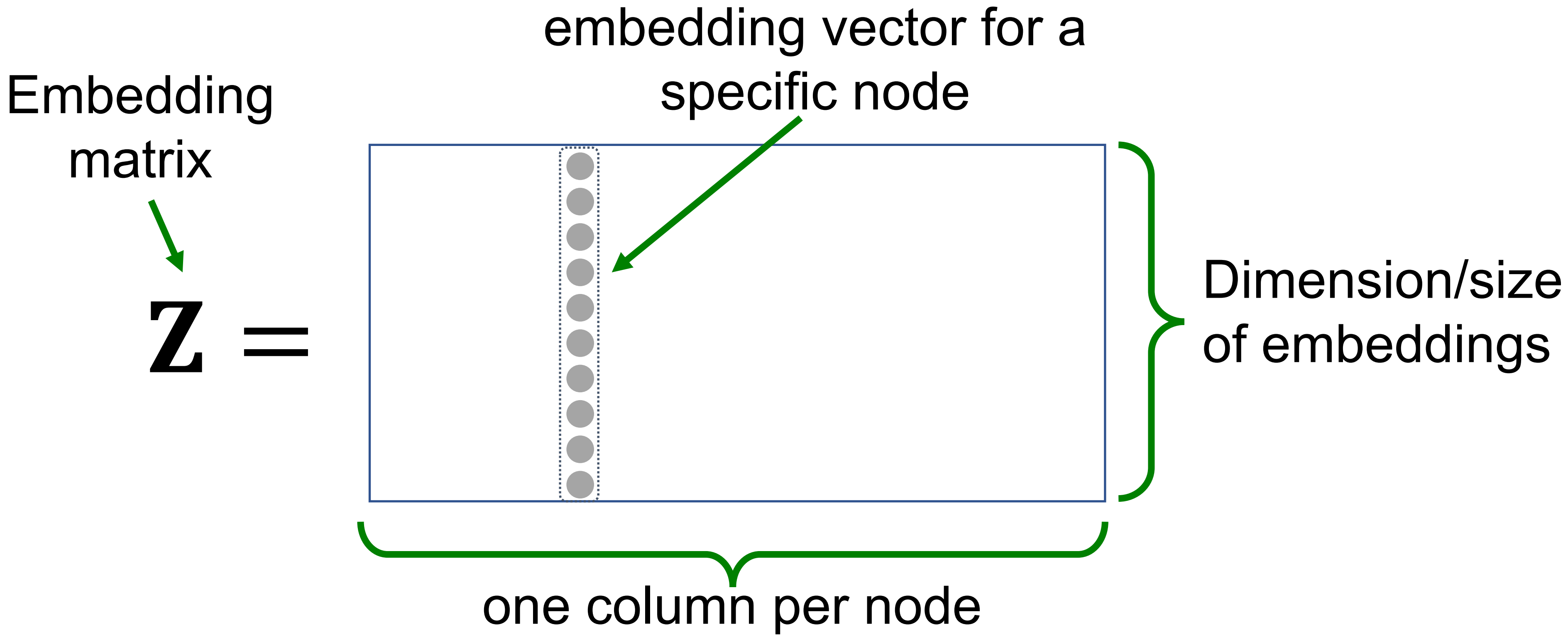
Encoder function

embedding dimension

- $\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$
  - 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

"Shallow" Encoding (2)

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



"Shallow" Encoding (3)

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**

Encoder + Decoder 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 **not** cover deep encoders today.

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

How to Define 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.

Note on Distributed 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 learn the embeddings of nodes so that some aspects of the network structure (captured by decoder) are preserved
- These embeddings are **task independent**
  - They are not trained for a specific task but can be used for any task.

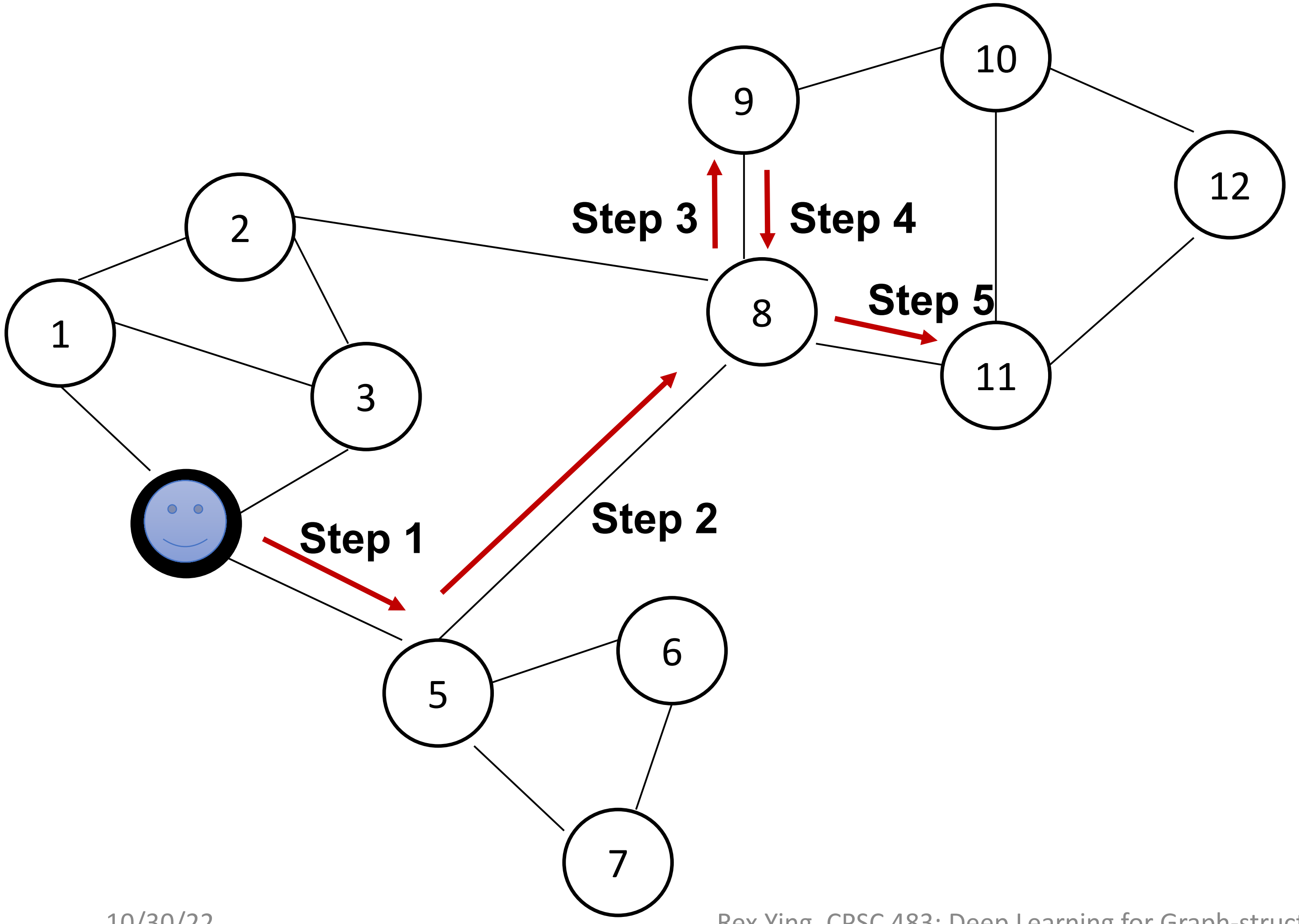
Outline of Today's Lecture

1. Non-GNN Node Embeddings

2. **Random Walk Approaches for Node Embeddings**

3. Embedding Entire Graphs

Random Walk



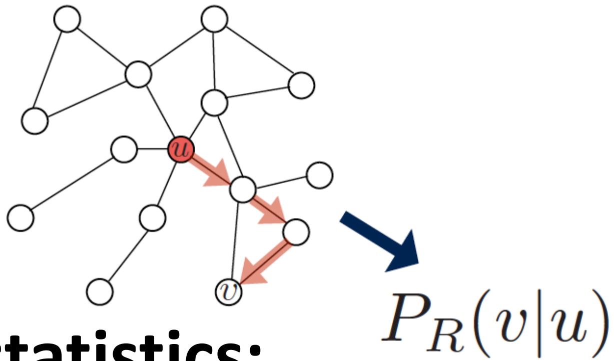
Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points visited this way is a **random walk on the graph**.

Random Walk Embeddings (1)

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

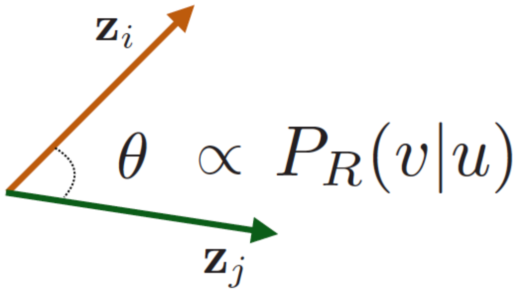
Random Walk Embeddings (2)

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”



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 $\boldsymbol{u}$ visits $\boldsymbol{v}$ with high probability, $\boldsymbol{u}$ and $\boldsymbol{v}$ 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?
  - $N_R(u)$ … neighbourhood of u obtained by some random walk strategy R

Feature Learning as Optimization

- Given $G = (V, E)$,
  - Our goal is to learn a mapping $f: u \to \mathbb{R}^d$:

$$f(u) = \mathbf{z}_u$$

- 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 $N_{\mathrm{R}}(u)$

Random Walk Optimization (1)

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

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

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

Random Walk Optimization (2)

- Equivalently,

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

- **Intuition:** Optimize embeddings 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: $\sum_i \exp(x_i) \approx \max_i \exp(x_i)$

Random Walk Optimization (3)

- Putting it all together

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

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

Random Walk Optimization (4)

- **But doing this naively is too expensive!**

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

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

Random Walk Optimization (5)

- **But doing this naively is too expensive!**

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

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

The normalization term from the softmax is the culprit… can we approximate it?

Negative Sampling (1)

- **Solution**: Negative Sampling

Why is the approximation valid?
Technically, this is a different objective. But Negative Sampling is a form of Noise Contrastive Estimation (NCE) which approx. maximizes the log probability of softmax.
New formulation corresponds to using a logistic regression (sigmoid func.) to distinguish the target node v from nodes n_i sampled from background distribution P_v.
More at https://arxiv.org/pdf/1402.3722.pdf

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

sigmoid function

Random distribution over nodes

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

Negative Sampling (2)

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

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

  In practive, $k = 5{\sim}20$

Random Walk Optimization (6)

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

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

- **Solution**: Gradient Descent
  - $z_i \leftarrow z_i - \eta \frac{\partial \mathcal{L}}{\partial z_i}, i \in \mathcal{V}$

Random Walk: 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?

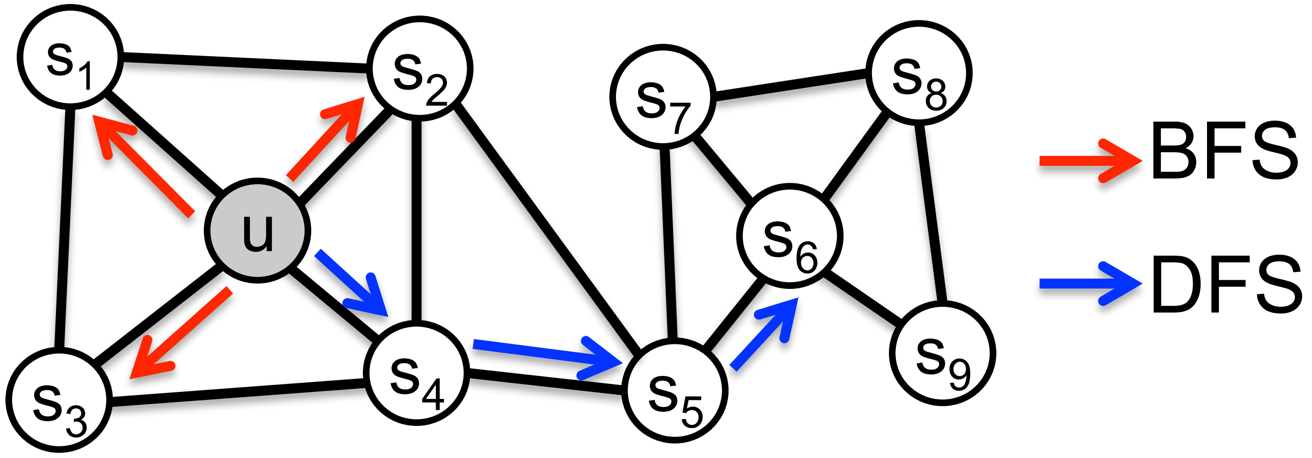
- So far we have described how to optimize embeddings given a random walk strategy R
- **What strategies should we use to run these random walks?**
  - Simplest idea: **Just run fixed-length, unbiased random walks starting from each node** (i.e., DeepWalk from Perozzi et al., 2013)
    - The issue is that such notion of similarity is too constrained
- **How can we generalize this?**

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 to the downstream prediction task.
- **Key observation:** Flexible notion of network neighborhood $N_R(u)$ of node u leads to rich node embeddings
- Develop biased 2^{nd} order random walk R to generate network neighborhood $N_R(u)$ of node u

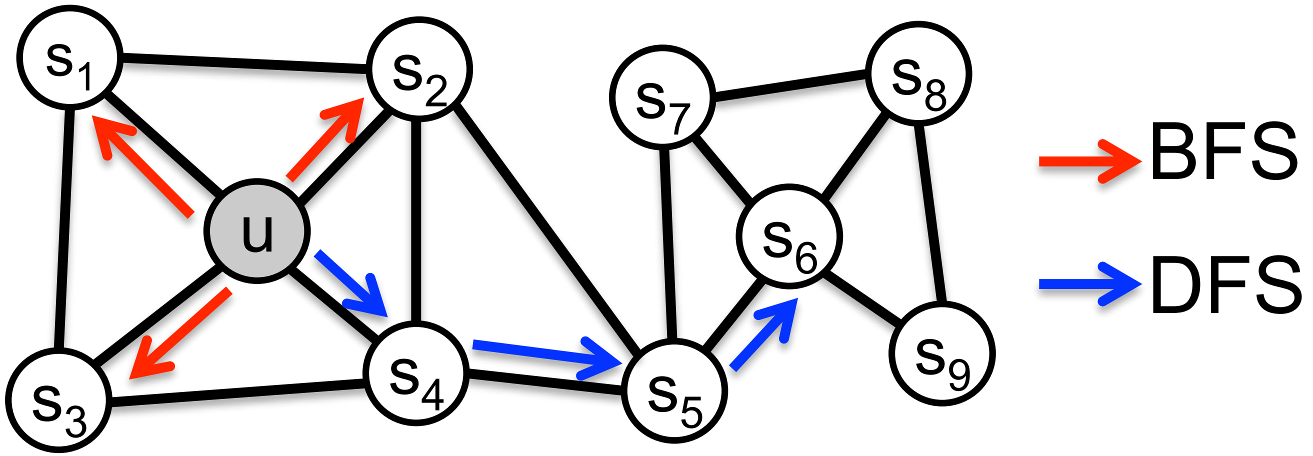
Node2Vec: Biased Walks (1)

- **Idea:** use flexible, biased random walks that can trade off between local and global views of the network (Grover and Leskovec, 2016).



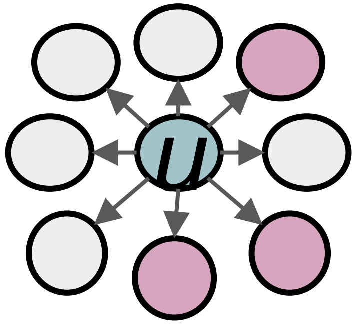
Node2Vec: Biased Walks (2)

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

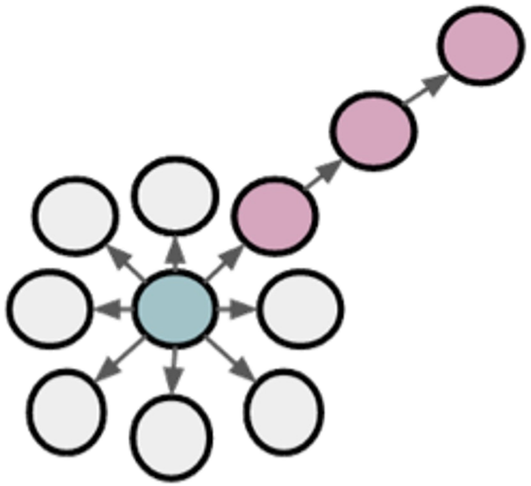


- Walk of length 3 ($N_R(u)$ of size 3):
  - $N_{BFS}(u) = \{s_1, s_2, s_3\}$ Local microscopic view
  - $N_{DFS}(u) = \{\, s_4, s_5, s_6\}$ Global macroscopic view

BFS vs. DFS



BFS:
Micro-view of neighbourhood



DFS:
Macro-view of neighbourhood

Interpolating BFS and DFS

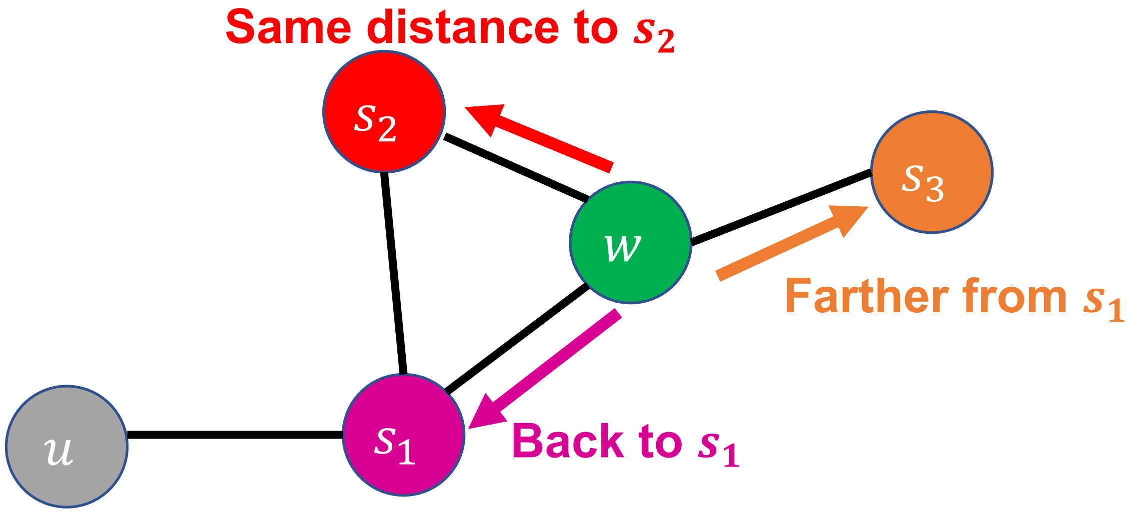
Biased fixed-length random walk R that given a node u generates neighborhood $N_R(u)$

- 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 (1)

Biased 2nd-order random walks explore network neighborhoods:

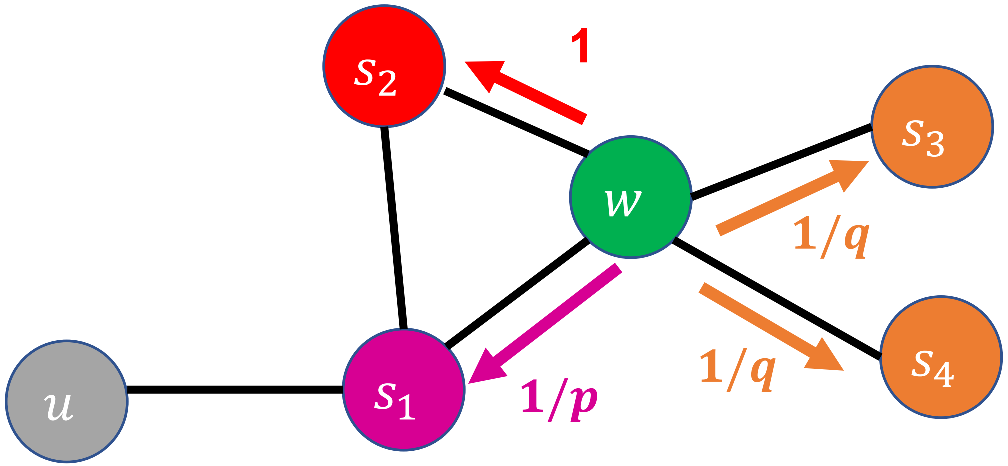
- Rnd. walk just traversed edge (s_1, w) and is now at w
- **Insight:** Neighbors of w can only be:



Idea: Remember where the walk came from

Biased Random Walks (2)

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

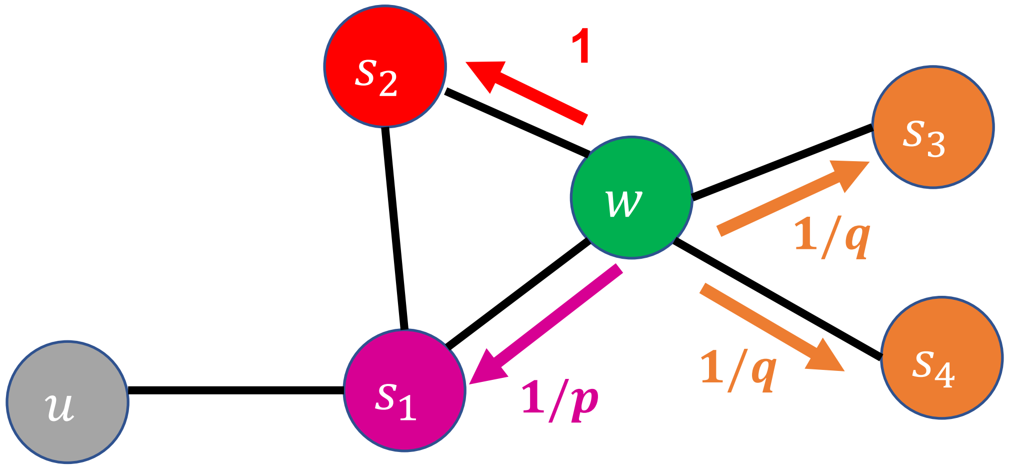


$1/p, 1/q, 1$ are unnormalized probabilities

- p, q model transition probabilities
  - p … return parameter
  - q … ”walk away” parameter

Biased Random Walks (3)

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



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

Unnormalized transition prob. segmented based on distance from s_1

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 decent

- Linear-time complexity
- All 3 steps are individually parallelizable

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 of Part 2

- **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 2 nodes are connected
  - Neighborhood overlap (covered in Lecture 2)
  - Random walk approaches **(covered today)**
- **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!

Outline of Today's Lecture

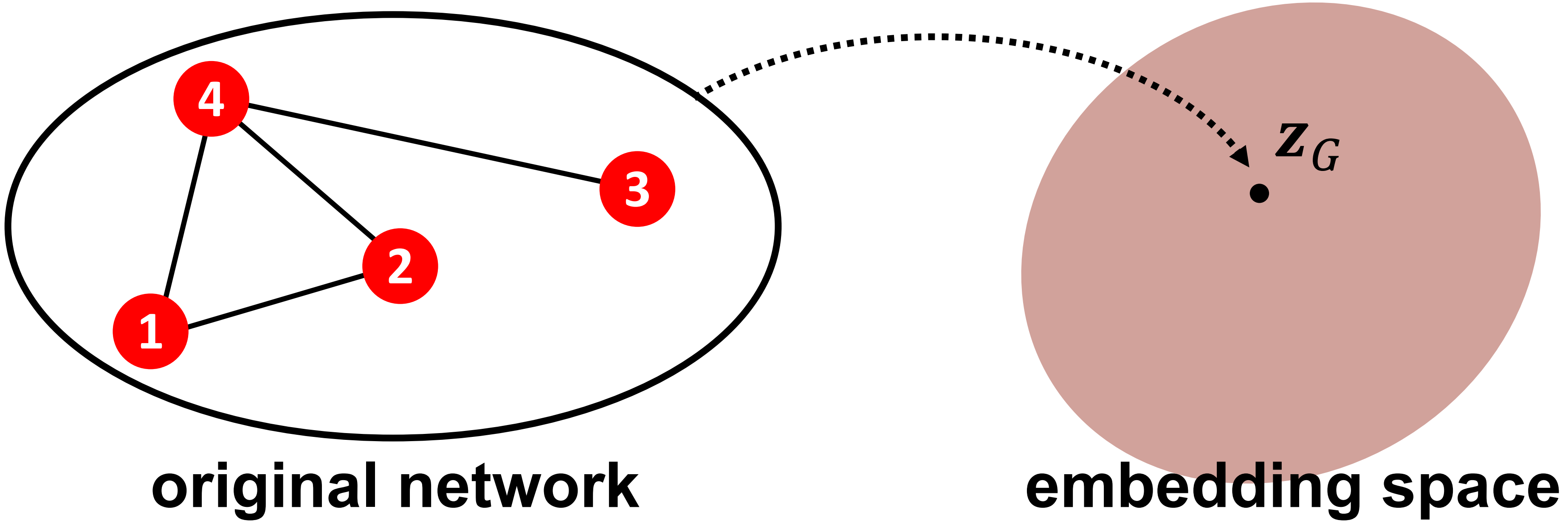
1. Non-GNN Node Embeddings

2. Random Walk Approaches for Node Embeddings

3. **Embedding Entire Graphs**

Embedding Entire Graphs

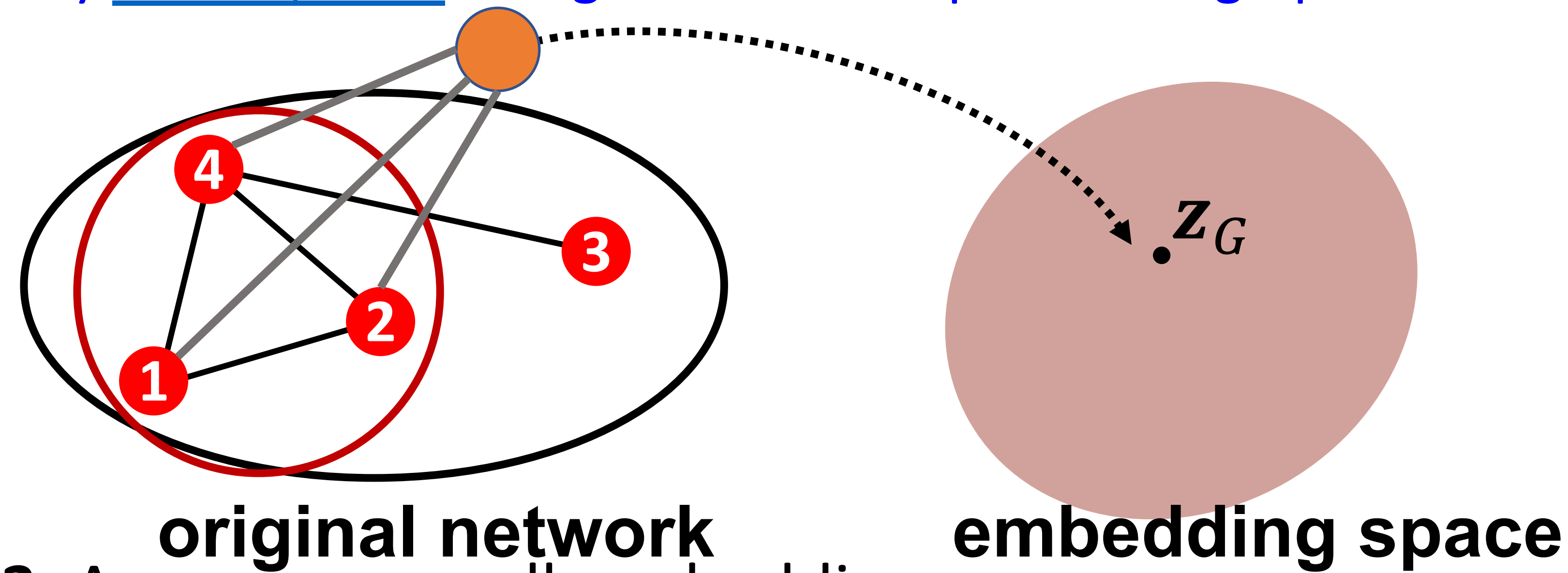
- **Goal:** Want to embed a subgraph or an entire graph G. Graph embedding: $\mathbf{z}_G$.



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

Embedding Entire Graphs: Approaches

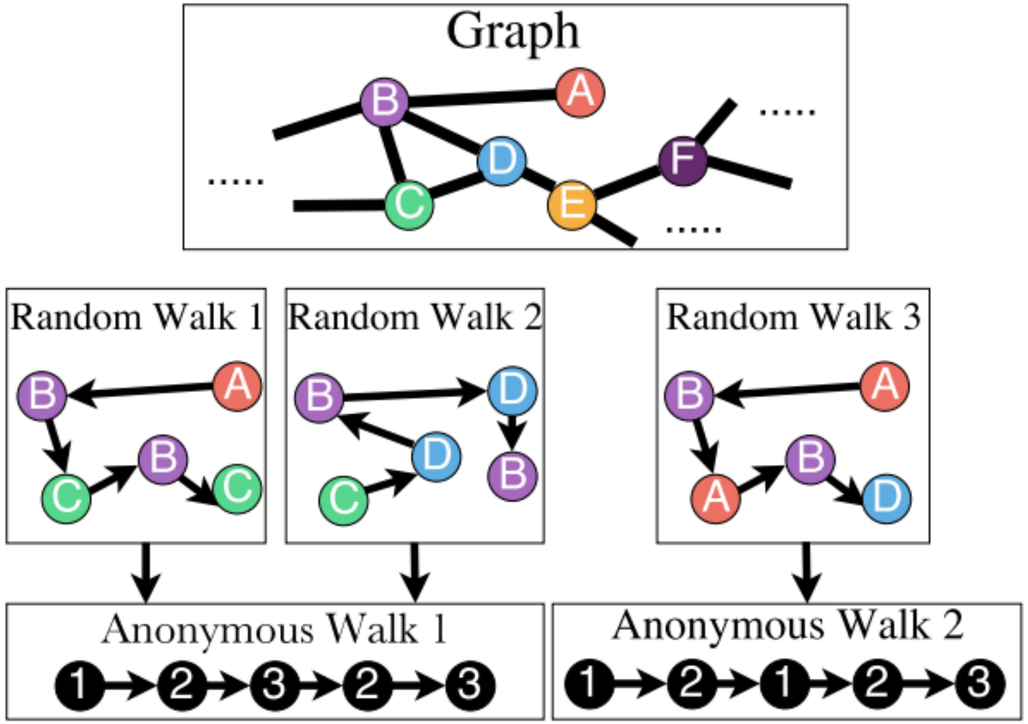
- **Approach 1**: (Recall in lecture 5) sum/mean/max/hierarchical pooling
- **Approach 2**: add **virtual node**
  - Proposed by Li et al., 2016 as a general technique for subgraph embedding



- **Approach 3**: Anonymous walk embeddings

Anonymous Walk Embeddings (1)

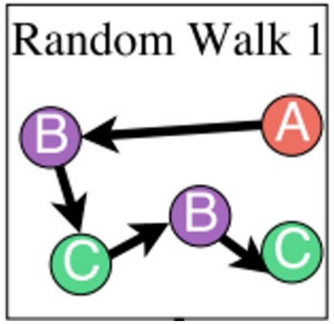
- States in **anonymous walks** correspond to the index of the **first time** we visited the node in a random walk

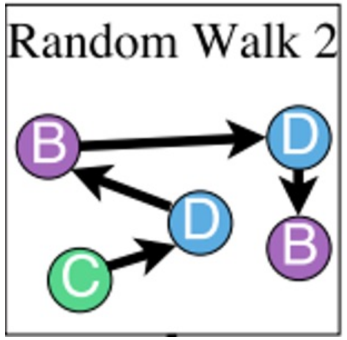


Anonymous Walk Embeddings, ICML 2018 https://arxiv.org/pdf/1805.11921.pdf

Anonymous Walk Embeddings (2)

- Agnostic to the identity of the nodes visited (hence anonymous)
- Example RW1 (Random Walk 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)



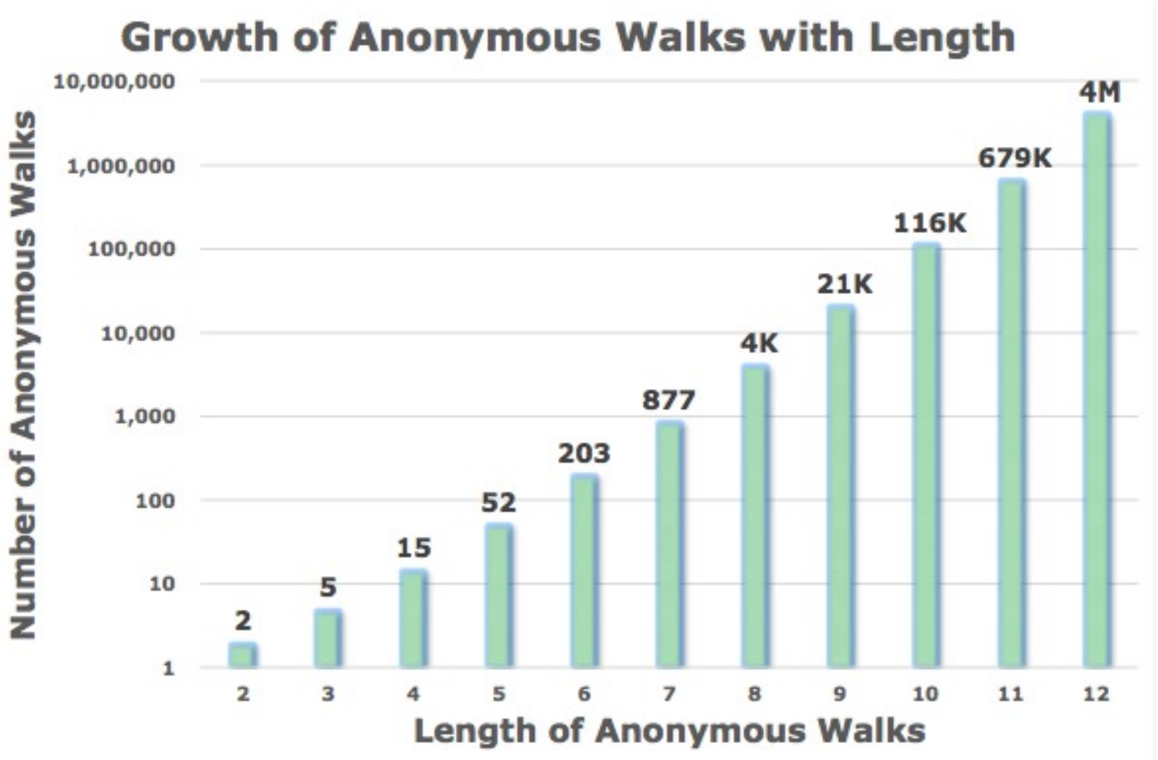


- Note: RW2 gives the same anonymous walk

Number of 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
  - $\boldsymbol{Z_G}[i]$ = probability of anonymous walk w_i in G

Sampling Anonymous Walks

- **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

New Idea: Learn Walk Embeddings

Rather than simply represent each walk by the fraction of times it occurs, we **learn embedding $\boldsymbol{z_i}$ of anonymous walk $\boldsymbol{w_i}$**

- **Learn a graph embedding $\boldsymbol{z_G}$ together with all the anonymous walk embeddings $\boldsymbol{z_i}$**
  $Z = \{z_i : i = 1 \ldots \eta\}$, where η is the number of sampled anonymous walks.
- Note that z_i are embeddings of anonymous walk now instead of embeddings of nodes.

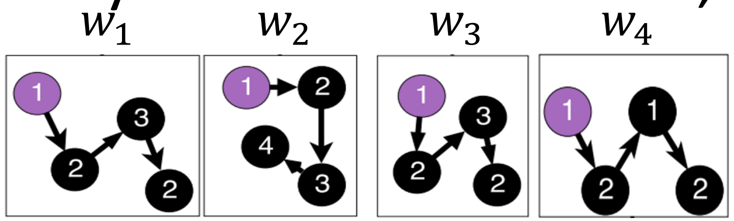
How to embed walks?

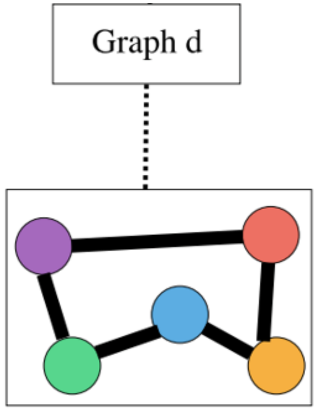
- **Idea:** Embed walks s.t. the next walk can be predicted

Anonymous Walk Embeddings, ICML 2018 https://arxiv.org/pdf/1805.11921.pdf

Learn Walk Embeddings (1)

- A vector parameter $\boldsymbol{z_G}$ for input graph
  - The embedding of entire graph to be learned
- Starting from **node 1**: Sample anonymous random walks, e.g.





- **Learn to predict walks that co-occur in Δ-size window** (e.g. predict w_2 given w_1, w_3 if $\Delta = 1$)
- Objective:

$$\max \sum_{t=\Delta}^{T-\Delta} \log P(w_t | w_{t-\Delta}, \ldots, w_{t+\Delta}, \boldsymbol{z_G})$$

- Sum the objective over all nodes in the graph

Learn Walk Embeddings (2)

- **Run T different random walks from u each of length l:**

$$N_R(u) = \{w_1^u, w_2^u \dots w_T^u\}$$

- **Learn to predict walks that co-occur in Δ-size window**
- Estimate embedding z_i of anonymous walk w_i

Objective: $\max \frac{1}{T} \sum_{t=\Delta}^{T-\Delta} \log P(w_t | \{w_{t-\Delta}, \dots, w_{t+\Delta}, \mathbf{z}_G\})$

Learn Walk Embeddings (3)

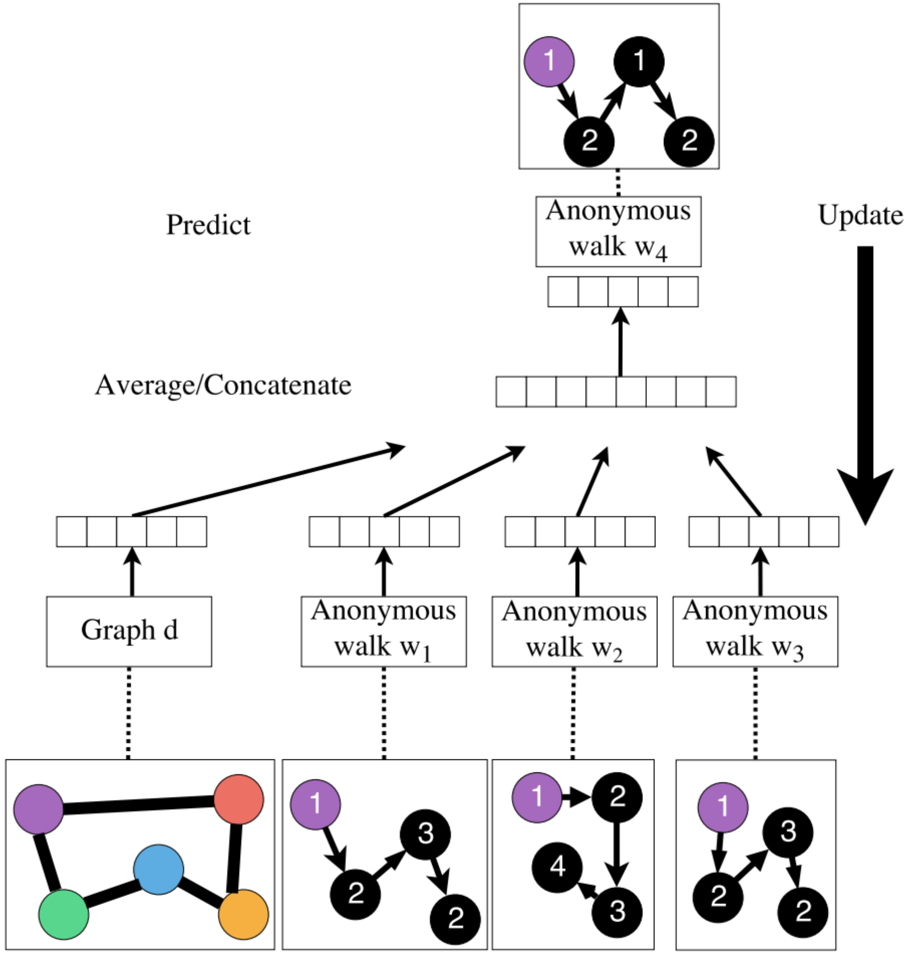
Objective: $\max \frac{1}{T}\sum_{t=\Delta}^{T-\Delta} \log P(w_t | \{w_{t-\Delta}, \dots, w_{t+\Delta}, \boldsymbol{z_G}\})$

- $P(w_t | \{w_{t-\Delta}, \dots, w_{t+\Delta}, \boldsymbol{z_G}\}) = \frac{\exp(y(w_t))}{\sum_{i=1}^{\eta} \exp(y(w_i))}$
- $y(w_t) = b + U \cdot \left(\mathrm{cat}(\frac{1}{2\Delta}\sum_{i=-\Delta, i\neq 0}^{\Delta} \boldsymbol{z_i}, \boldsymbol{z_G})\right)$
  - $\mathrm{cat}(\frac{1}{2\Delta}\sum_{i=-\Delta}^{\Delta} z_i, \boldsymbol{z_G})$ means an average of anonymous walk embeddings in window, concatenated with the graph embedding $\boldsymbol{z_G}$
  - $b \in \mathbb{R}$, $U \in \mathbb{R}^D$ are learnable parameters. This represents a linear layer.
  - $\boldsymbol{z_i}, \boldsymbol{z_G}$ are learnable.

All possible walks
(η be number of all
possible walk embeddings)

Learn Walk Embeddings (4)

- We obtain the graph embedding $\boldsymbol{z}_G$ (learnable parameter) after optimization
- Use $\boldsymbol{z}_G$ to make predictions (e.g. graph classification)



Overall Architecture

Summary of Part 3

We discussed 3 ideas to graph embeddings

- **Approach 1:** sum/mean/max/hierarchical pooling
- **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: Jointly learn anonymous walks' embeddings and graph embedding

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 and "shallow" encoding:**
  - **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**