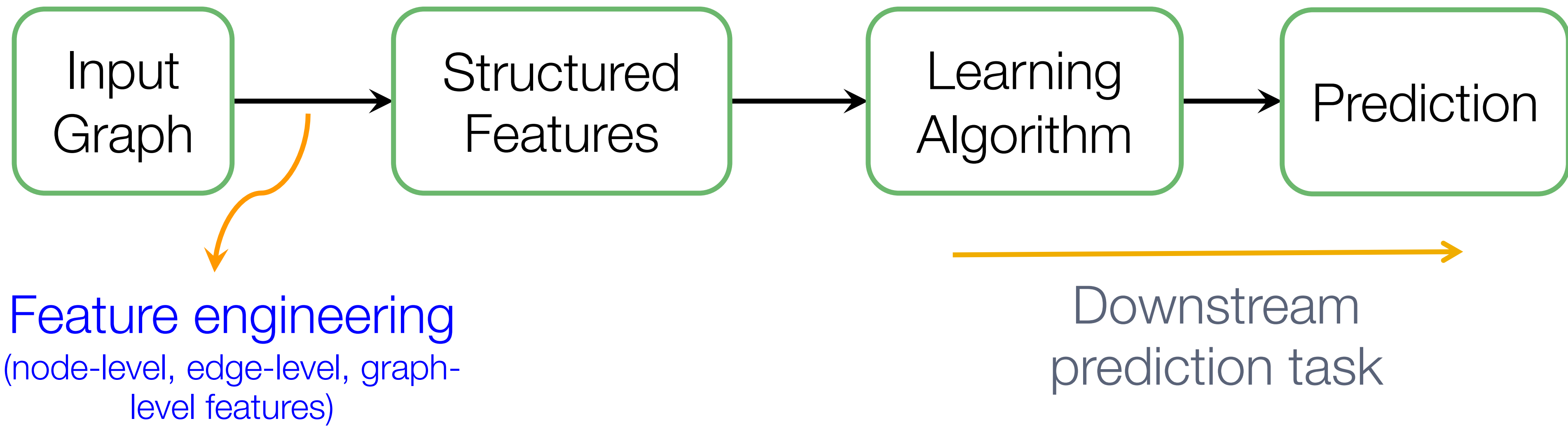
Stanford CS224W: Node Embeddings

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

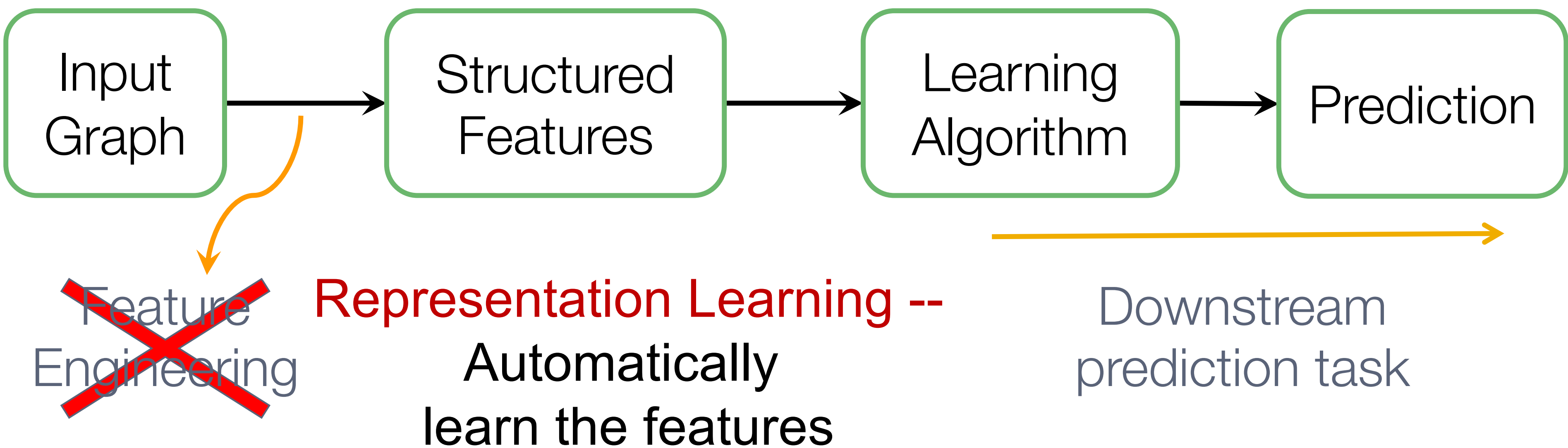
Recap: Traditional ML for Graphs

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



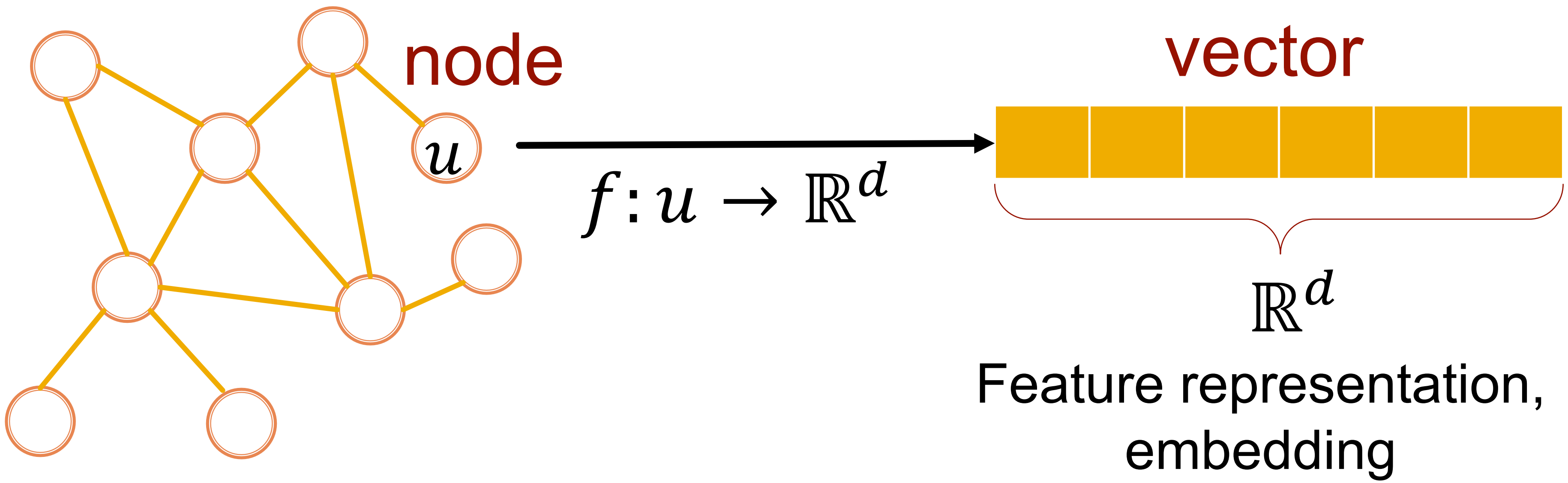
Graph Representation Learning

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



Graph Representation Learning

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



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

Vec

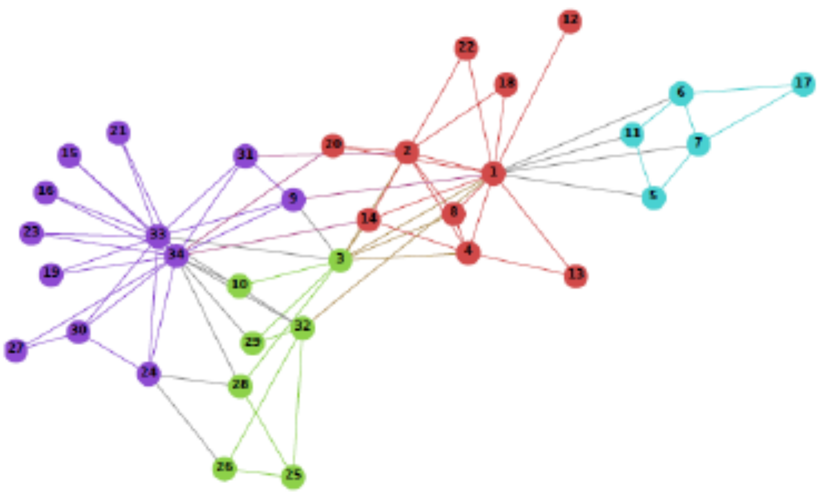
embeddings $\mathbb{R}^d$

Tasks

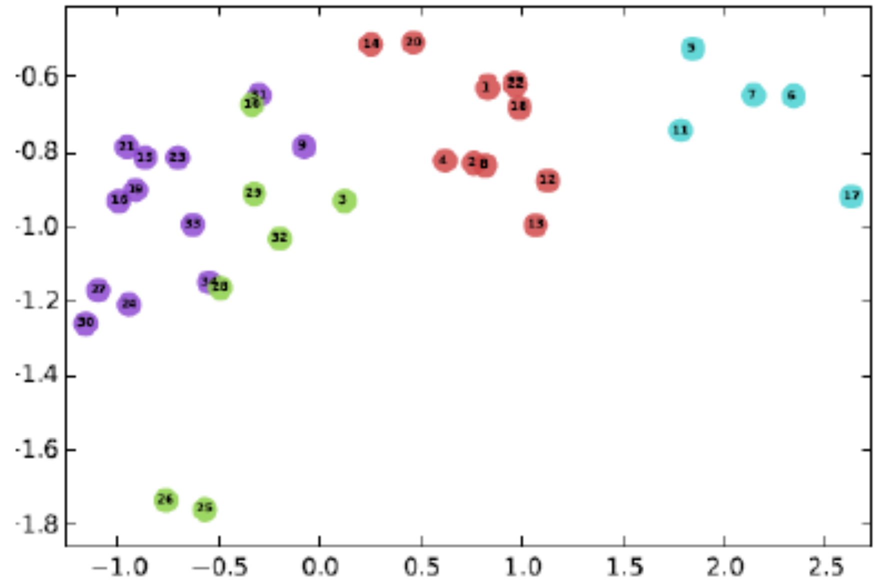
- Node classification
- Link prediction
- Graph classification
- Anomalous node detection
- Clustering
- ….

Example Node Embedding

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



Input



Output

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

Stanford CS224W: Node Embeddings: Encoder and Decoder

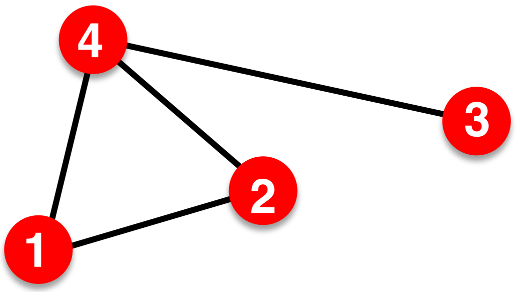
CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

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**

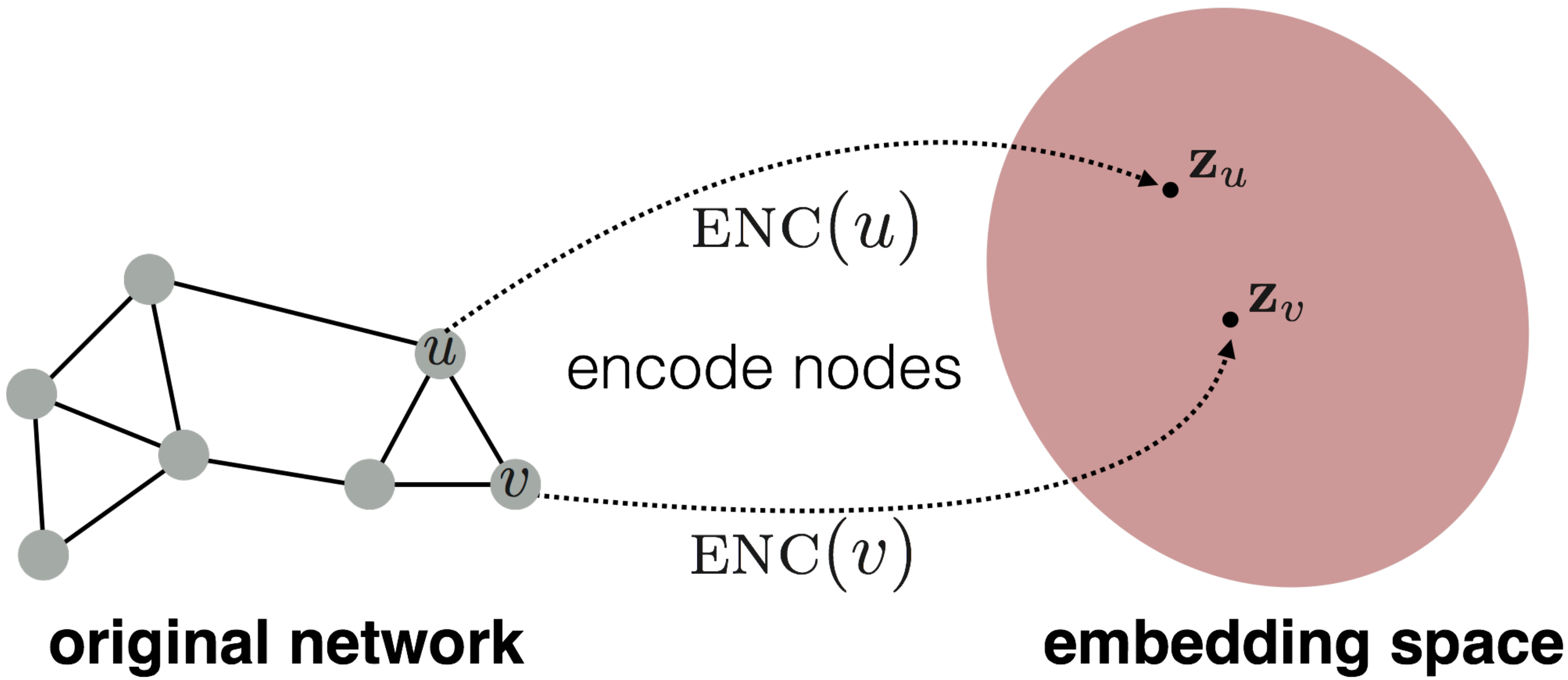


V: {1, 2, 3, 4}

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

Embedding Nodes

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

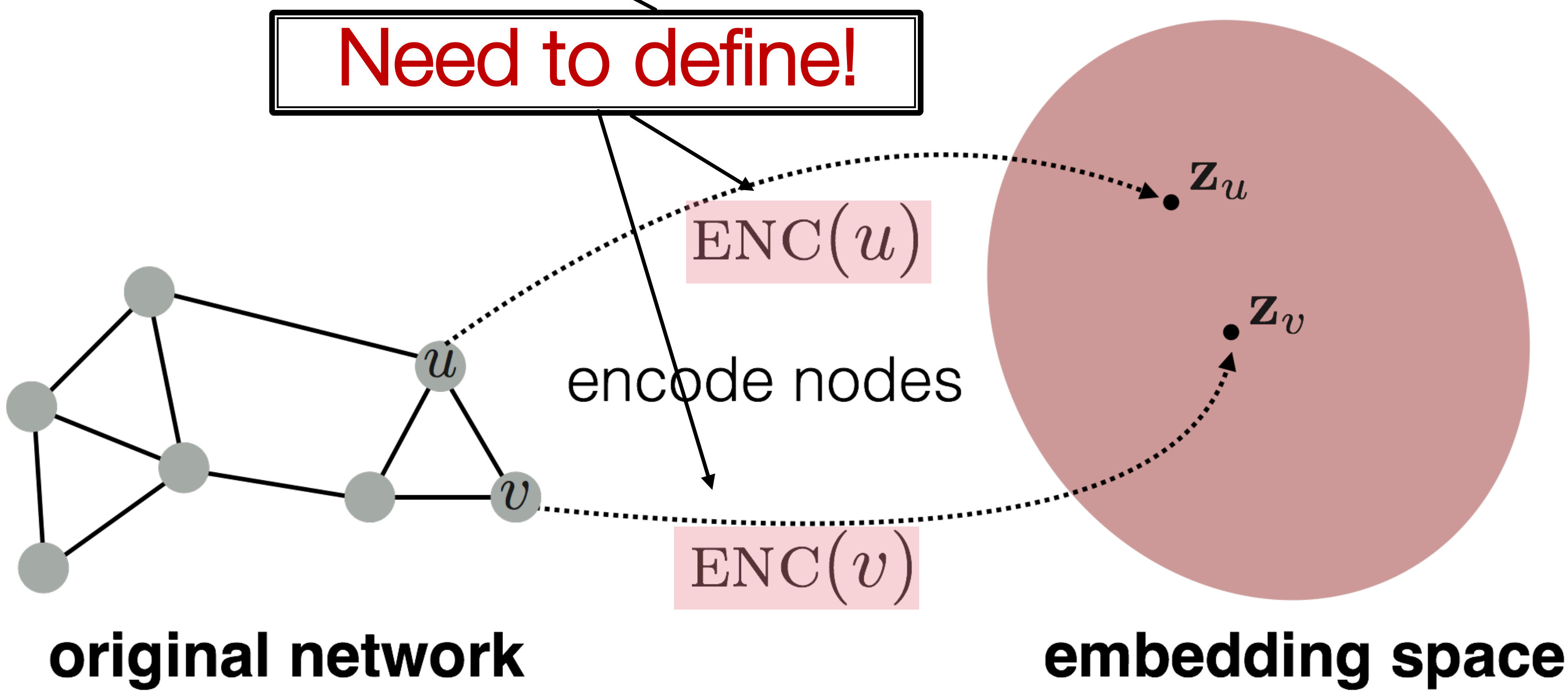


Embedding Nodes

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

in the original network

Similarity of the embedding



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** $\mathbf{DEC}$ maps from embeddings to the similarity score
4. **Optimize the parameters of the encoder so that:**

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

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

in the original network　　Similarity of the embedding

Two Key Components

- **Encoder:** maps each node to a low-dimensional vector

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

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

"Shallow" Encoding

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

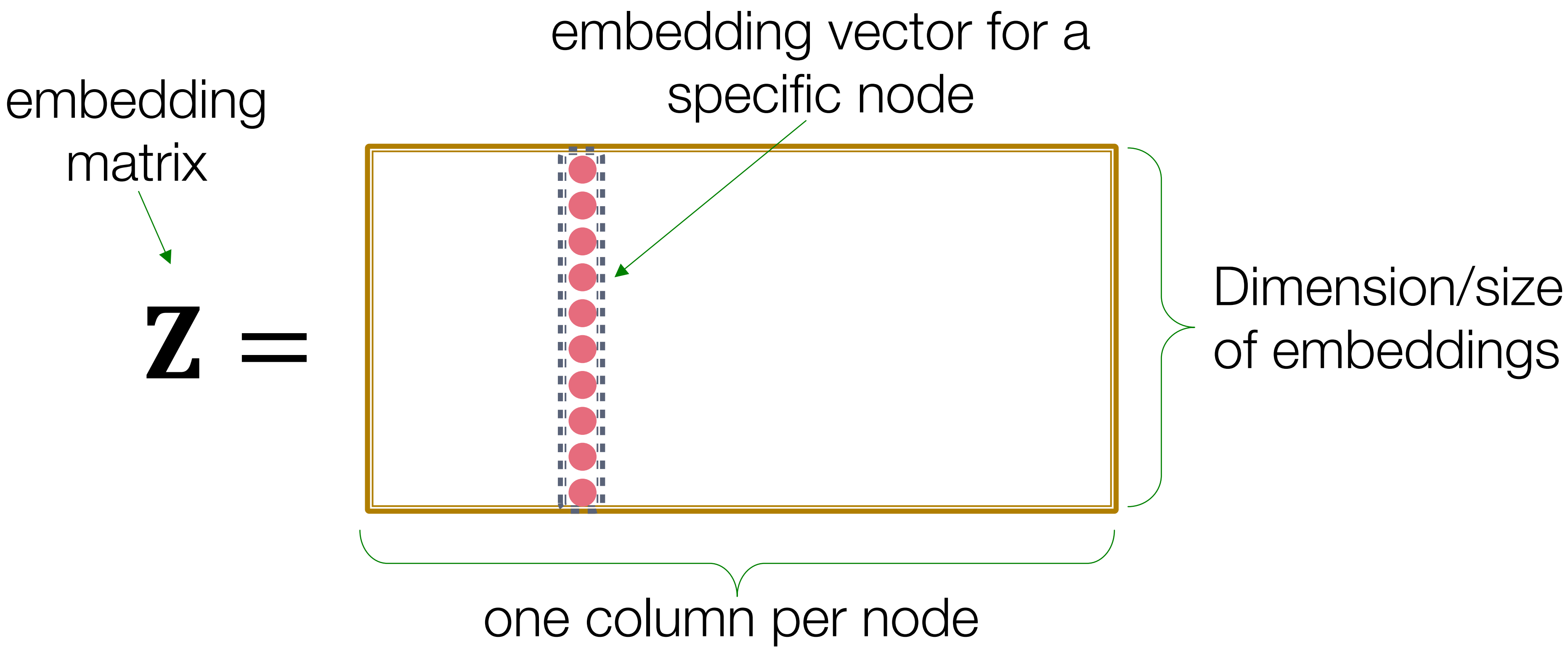
$$\text{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ 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 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 $u \in V$
  - We will cover deep encoders (GNNs) in Lecture 6

  - **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 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.

Stanford CS224W: Random Walk Approaches for Node Embeddings

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

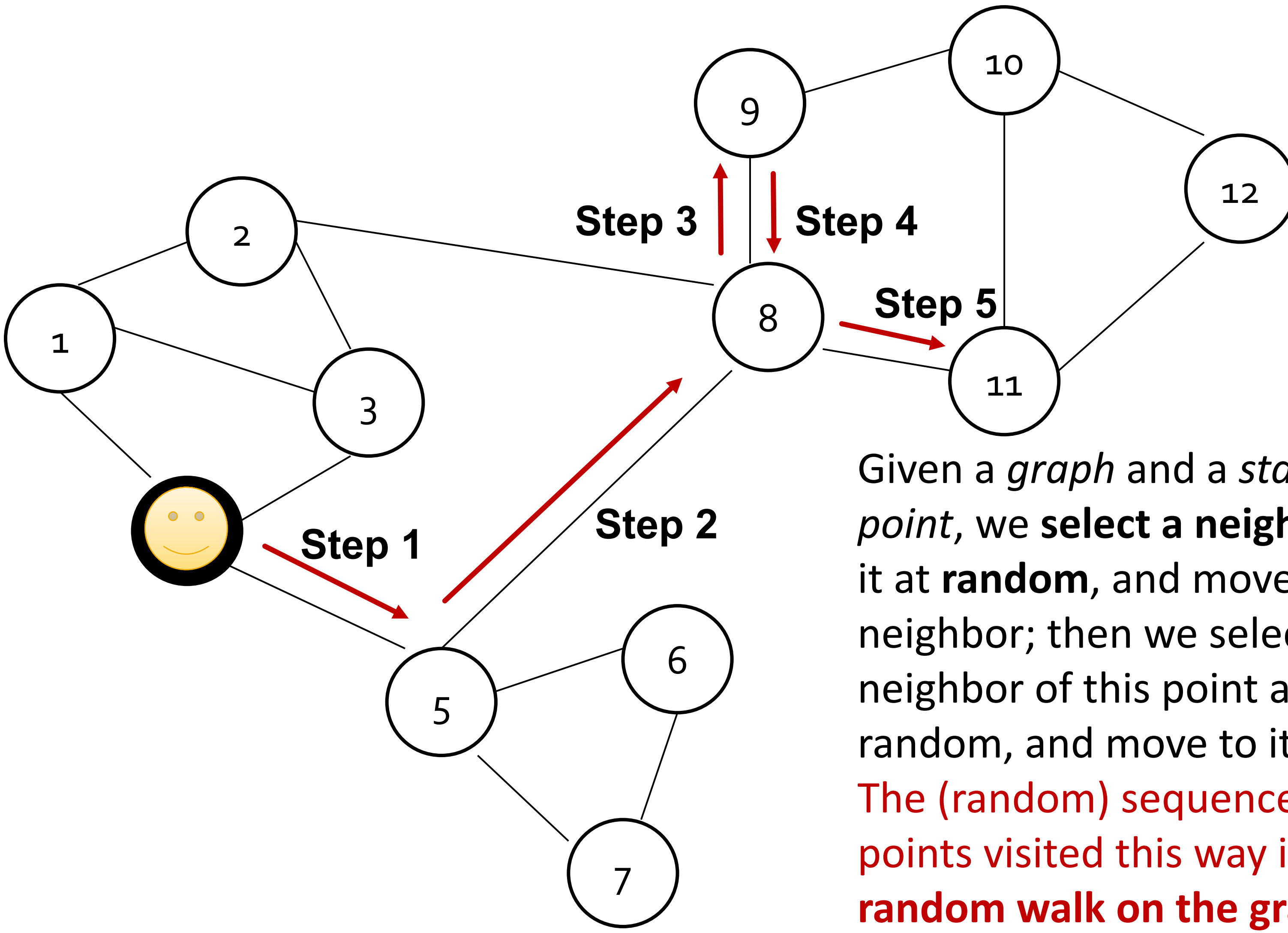
Notation

- **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: $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$.
- **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



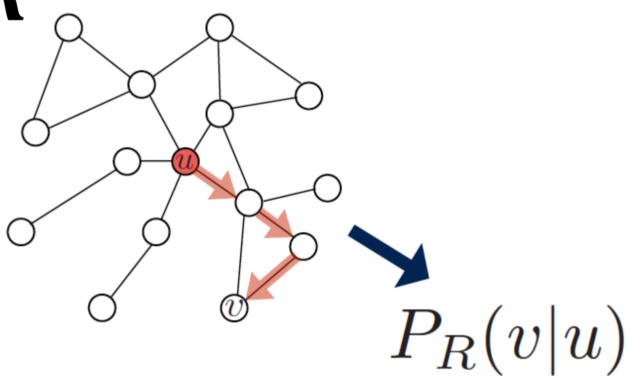
Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points visited this way is a **random walk on the graph**.

Random-Walk Embeddings

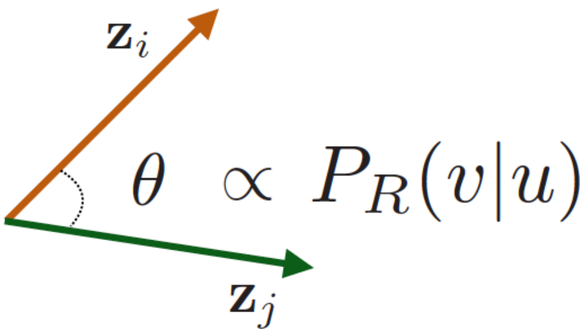
$$\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v \approx$$ probability that u and v co-occur on a random walk over the graph

Random-Walk Embeddings

1. **Estimate 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. 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 $N_{\mathrm{R}}(u)$

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

Putting it all together:

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

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 $\mathcal{L}$

Random Walk Optimization

But doing this naively is too expensive!

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

Nested sum over nodes gives
$\mathrm{O}(|\mathrm{V}|^2)$ complexity!

Random Walk Optimization

But doing this naively is too expensive!

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

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

Negative Sampling

- **Solution:** Negative sampling

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

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

sigmoid function
(makes each term a "probability" between 0 and 1)

random distribution over nodes

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

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

Negative Sampling

random distribution over nodes

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

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

- Sample k negative nodes each with prob. proportional to its degree
- Two 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

Stochastic Gradient Descent

- 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 $\mathcal{L}$:
  - Initialize z_i at some randomized value for all i.
  - Iterate until convergence.
    - For all i, compute the derivative $\frac{\partial \mathcal{L}}{\partial z_i}$.
    - For all i, make a step towards the direction of derivative: $z_i \leftarrow z_i - \eta \frac{\partial \mathcal{L}}{\partial z_i}$.

η: learning rate

Stochastic Gradient Descent

- **Stochastic Gradient Descent**: Instead of evaluating gradients over all examples, evaluate it for each **individual** training example.

  - Initialize z_i at some randomized value for all i.
  - Iterate until convergence: $\mathcal{L}^{(u)} = \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$
    - Sample a node i, for all j calculate the derivative $\frac{\partial \mathcal{L}^{(i)}}{\partial z_j}$.
    - For all j, update: $z_j \leftarrow z_j - \eta \frac{\partial \mathcal{L}^{(i)}}{\partial z_j}$.

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 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?**

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

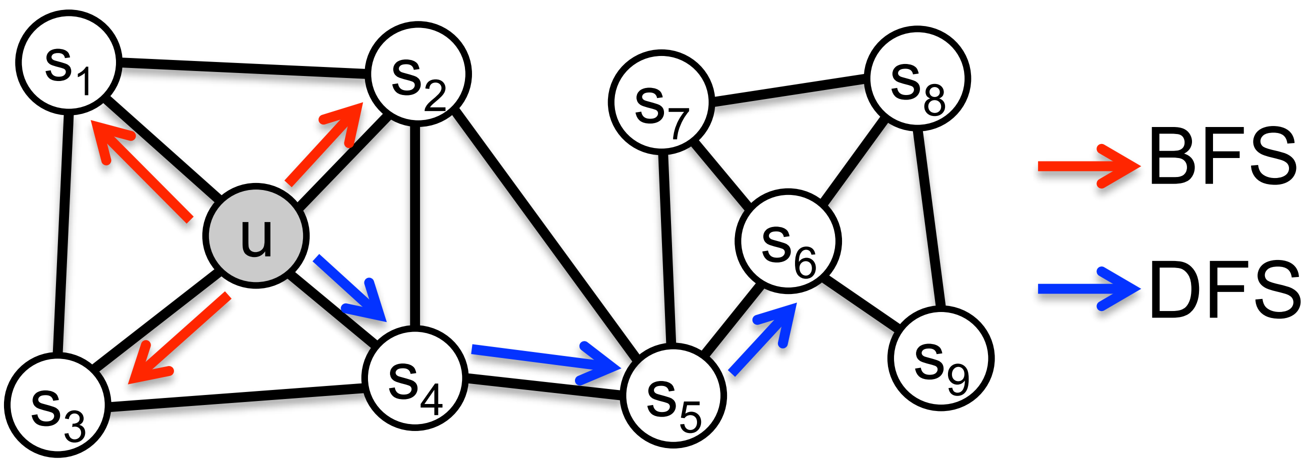
Overview of node2vec

- **Goal:** Embed nodes with similar network neighborhoods close in the feature space.
- We frame this goal as a maximum likelihood optimization problem, independent 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 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.*

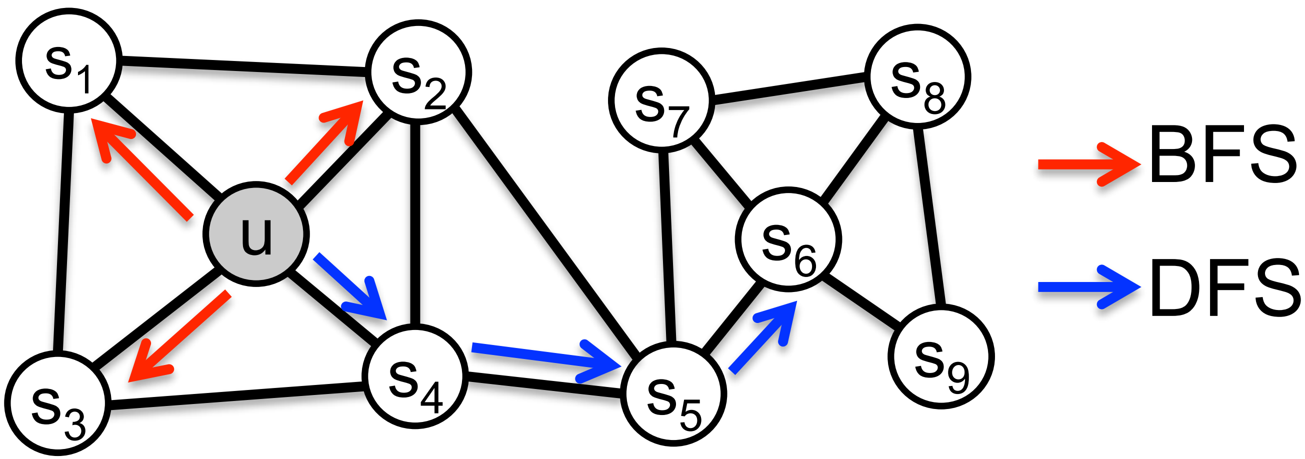
node2vec: Biased Walks

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



node2vec: Biased Walks

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

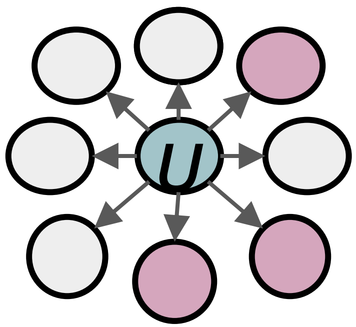


Walk of length 3 ($N_R(u)$ of size 3):

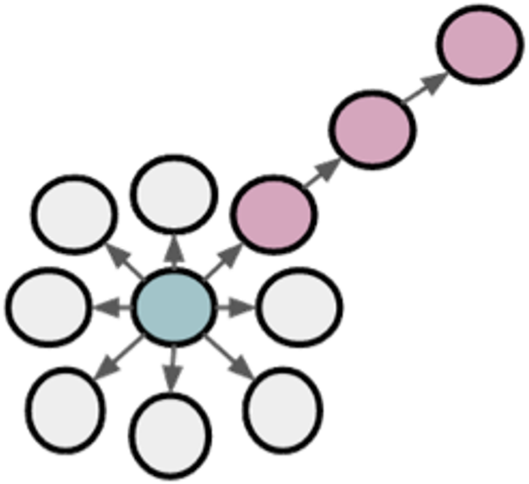
$N_{BFS}(u) = \{ s_1, s_2, s_3 \}$ Local microscopic view

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

BFS vs. DFS



BFS:
Micro-view of neighbourhood



DFS:
Macro-view of neighbourhood

Interpolating BFS and DFS

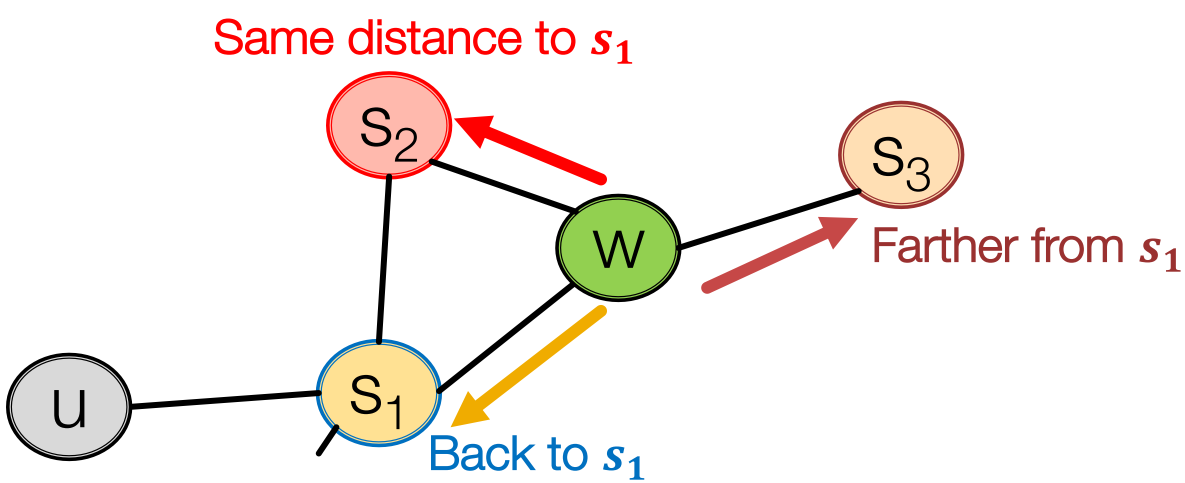
Biased fixed-length random walk R that given a node u generates neighborhood $N_R(u)$

- 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:

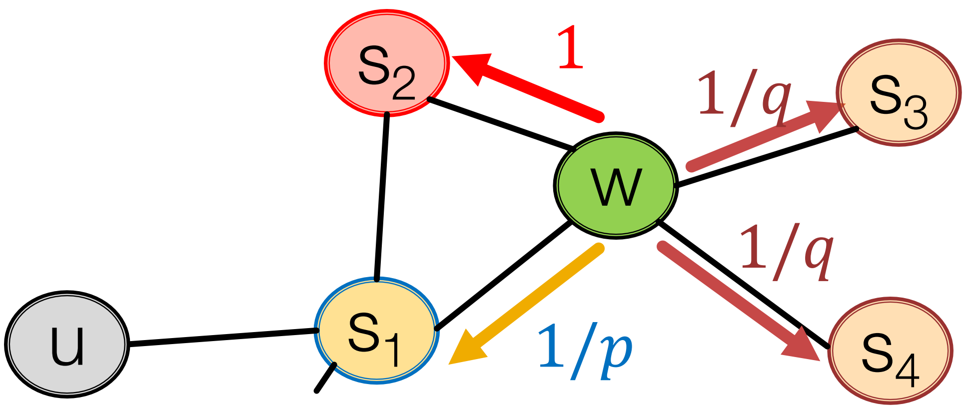
- Rnd. walk just traversed edge (s_1, w) and is now at w
- **Insight:** Neighbors of w can only be:



Idea: Remember where the walk came from

Biased Random Walks

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

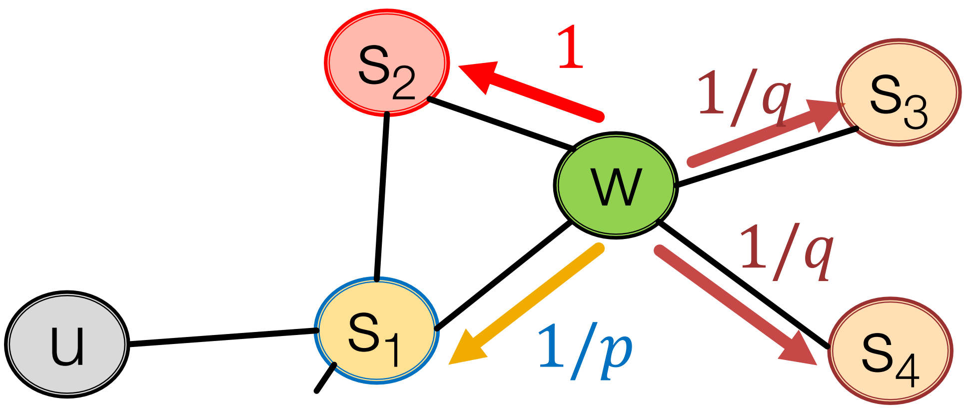


$1/p, 1/q, 1$ are unnormalized probabilities

- 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

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

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 2 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!

Stanford CS224W: Embedding Entire Graphs

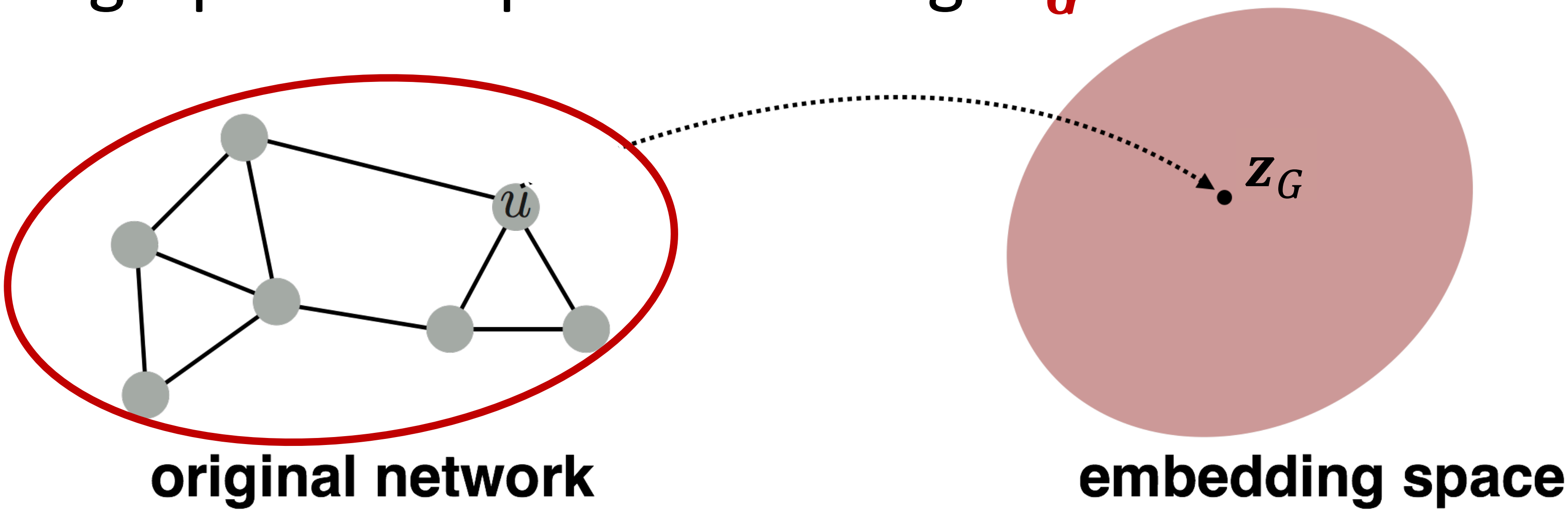
CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

Embedding Entire Graphs

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



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

Approach 1

Simple idea 1:

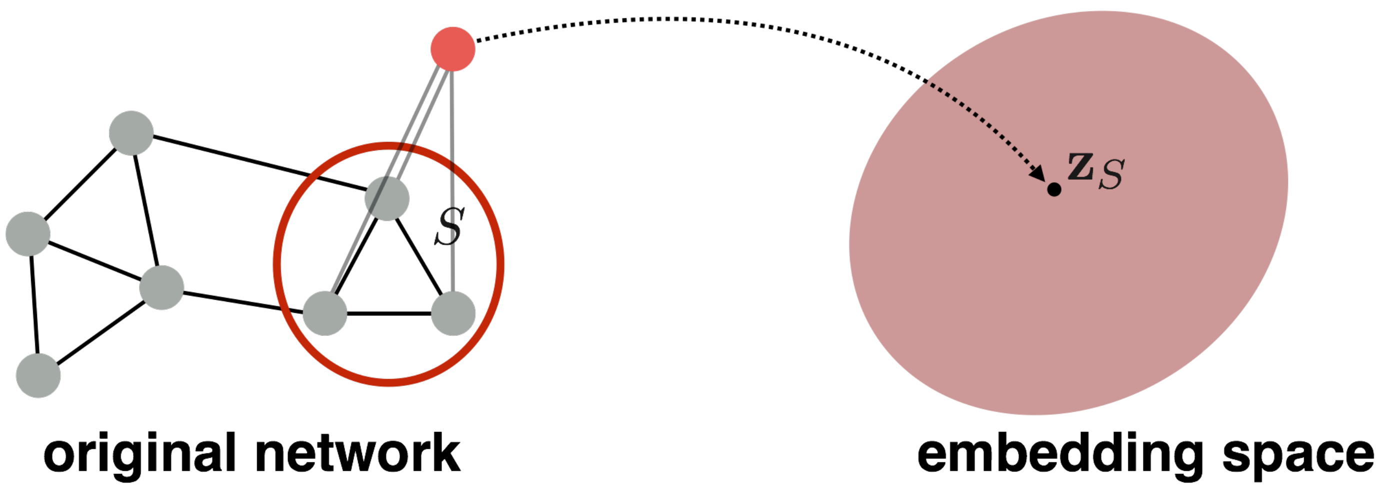
- Run a standard graph embedding technique *on* the (sub)graph G
- Then just sum (or average) the node embeddings in the (sub)graph G

$$\mathbf{z}_{\boldsymbol{G}} = \sum_{v \in G} z_v$$

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

Approach 2

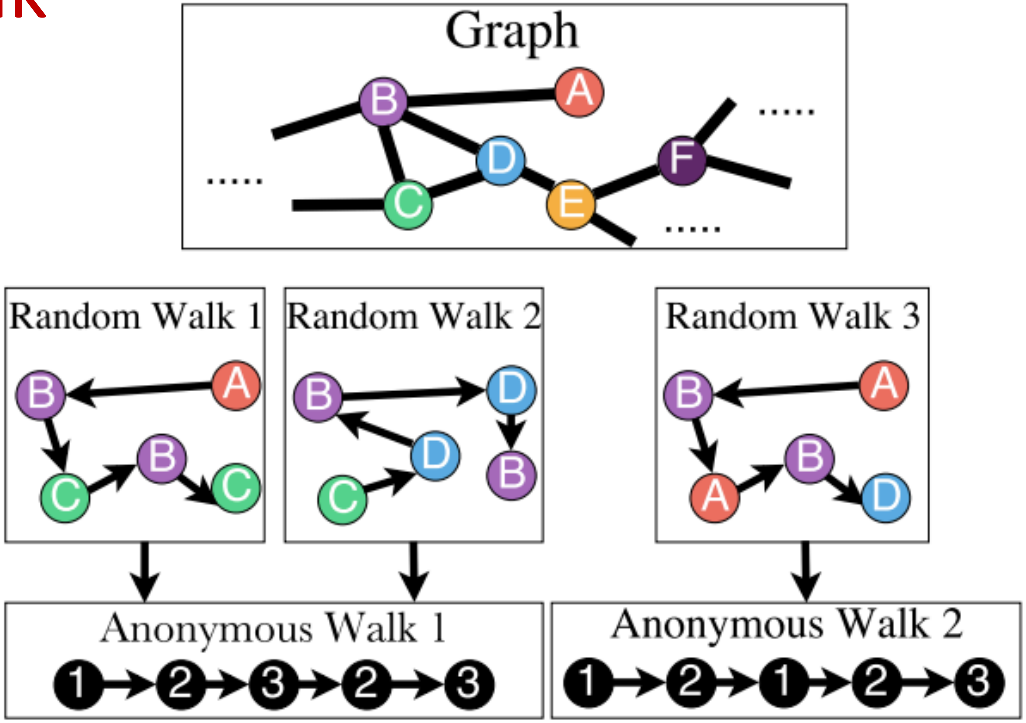
- **Idea 2:** Introduce a **"virtual node"** to represent the (sub)graph and run a standard graph embedding technique



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

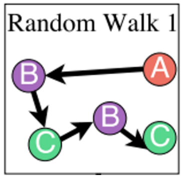
Approach 3: Anonymous Walk Embeddings

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

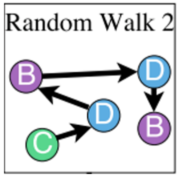


Anonymous Walk Embeddings, ICML 2018 https://arxiv.org/pdf/1805.11921.pdf

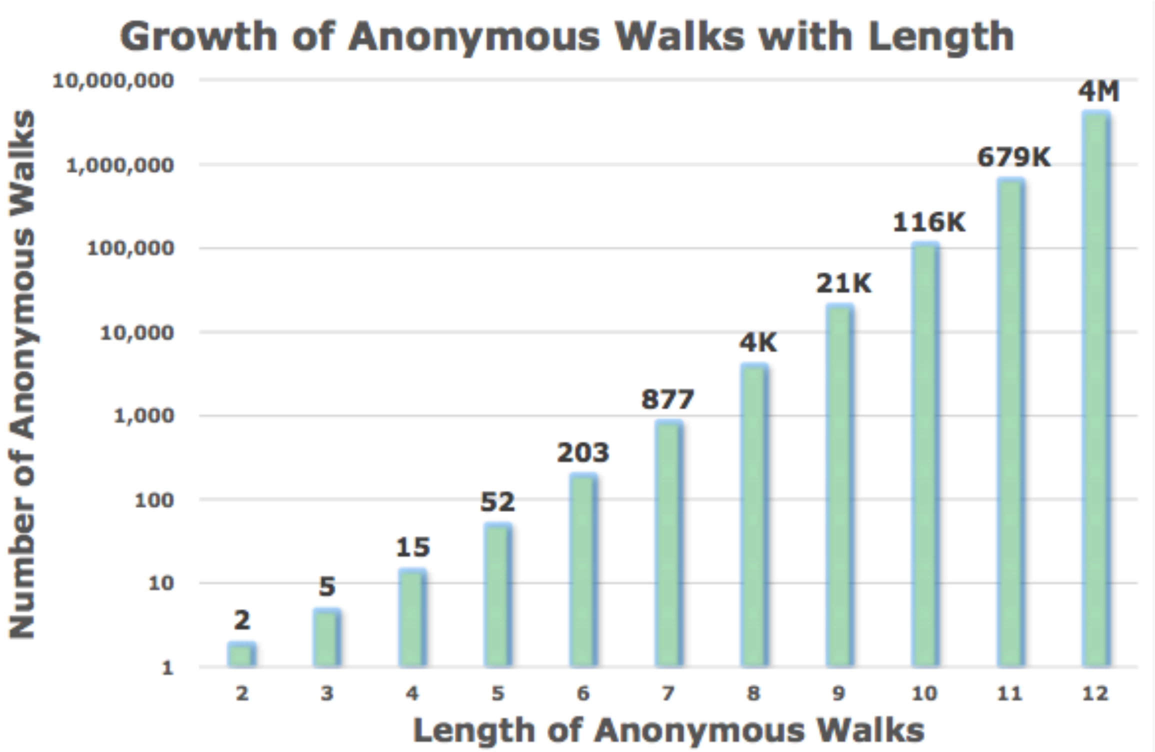
Approach 3: Anonymous Walk Embeddings

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

- 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

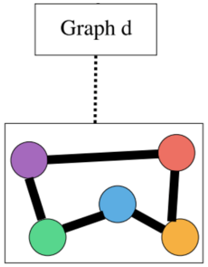
Rather than simply represent each walk by the fraction of times it occurs, we **learn embedding z_i of anonymous walk w_i**

- **Learn a graph embedding Z_G together with all the anonymous walk embeddings z_i**
  $Z = \{z_i: i = 1 \ldots \eta\}$, where η is the number of sampled anonymous walks.

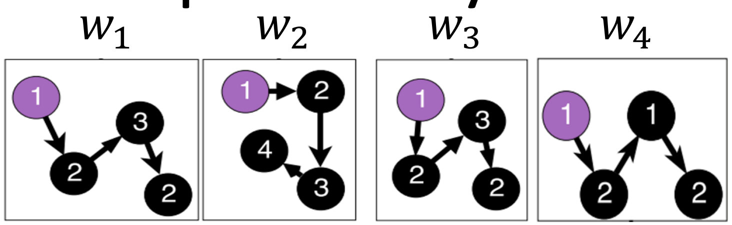
How to embed walks?

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

Learn Walk Embeddings



- 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:

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

- Sum the objective over all nodes in the graph

Learn Walk Embeddings

- **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
  Let η be number of all possible walk embeddings

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

  - $P(w_t | \{w_{t-\Delta}, \dots, w_{t+\Delta}, \boldsymbol{z_G}\}) = \frac{\exp(y(w_t))}{\sum_{i=1}^{\eta} \exp(y(w_i))}$

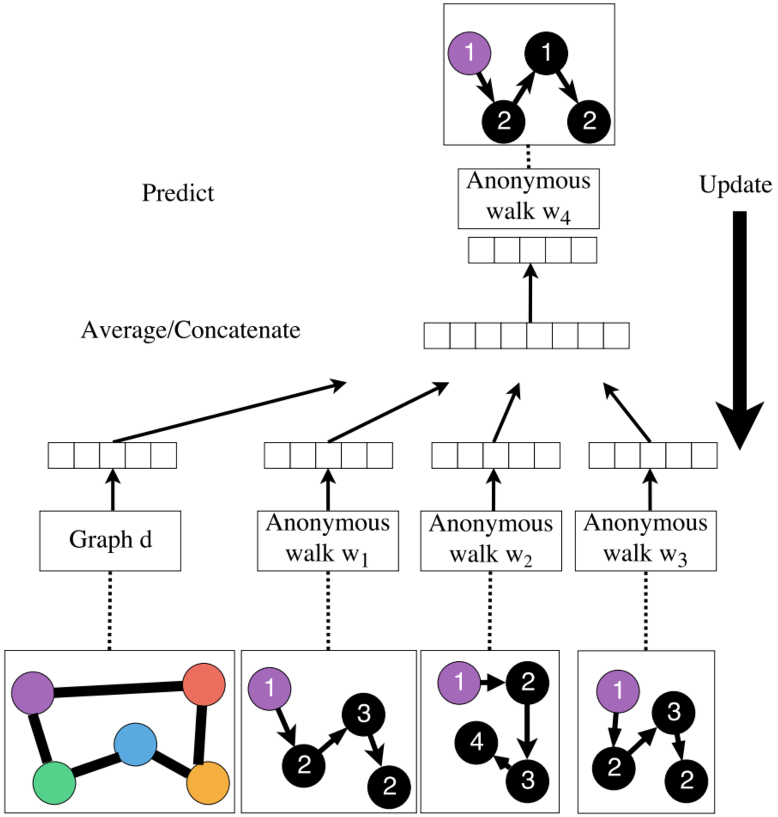
    All possible walks (require negative sampling)
  - $y(w_t) = b + U \cdot \left(cat(\frac{1}{2\Delta} \sum_{i=-\Delta}^{\Delta} z_i, \boldsymbol{z_G}) \right)$
    - $cat(\frac{1}{2\Delta} \sum_{i=-\Delta}^{\Delta} z_i, \boldsymbol{z_G})$ means an average of anonymous walk embeddings in window, concatenated with the graph embedding $\boldsymbol{z_G}$
    - $b \in \mathbb{R}$, $U \in \mathbb{R}^D$ are learnable parameters. This represents a linear layer.

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

Learn Walk Embeddings

- We obtain the graph embedding z_G (learnable parameter) after optimization
- Use z_G to make predictions (e.g. graph classification)
  - **Option1**: Inner product Kernel $z_{G_1}^T z_{G_2}$ (Lecture 2)
  - **Option2**: Use a neural network that takes z_G as input to classify

Overall Architecture



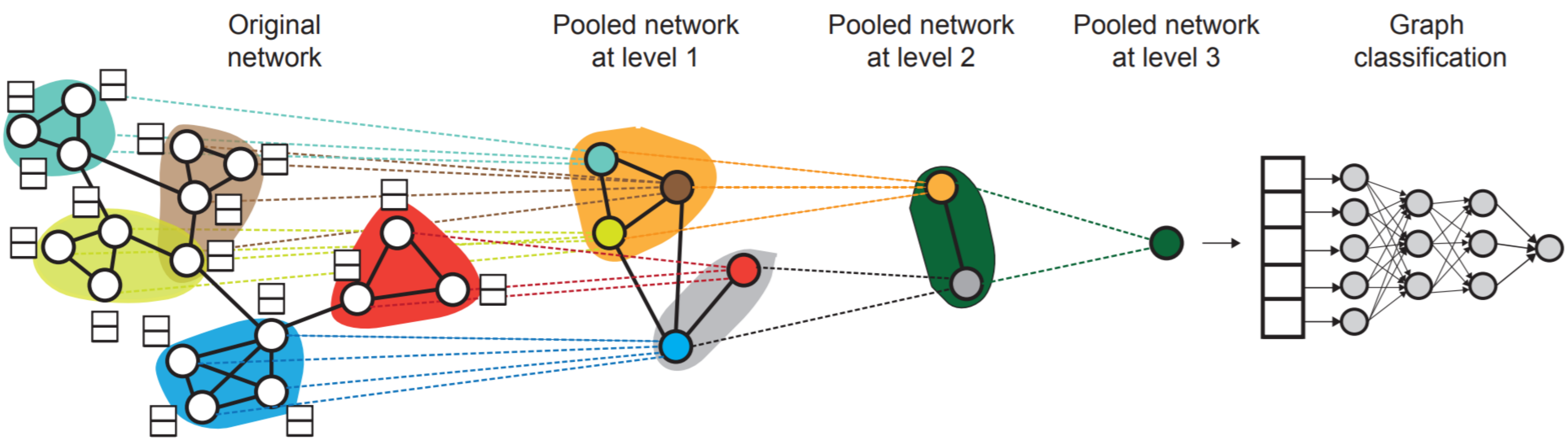
Summary

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: Embed anonymous walks, concatenate their embeddings to get a graph embedding

Preview: Hierarchical Embeddings

- We will discuss more advanced ways to obtain graph embeddings in Lecture 8.
- We can **hierarchically** cluster nodes in graphs, and **sum/avg** 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: $f(z_i, z_j)$= $g([z_i, z_j])$
      - Hadamard: $f(z_i, z_j)$= $g(z_i * z_j)$ (per coordinate product)
      - Sum/Avg: $f(z_i, z_j)$= $g(z_i + z_j)$
      - Distance: $f(z_i, z_j)$= $g(||z_i - z_j||_2)$
  - **Graph classification**: graph embedding $\boldsymbol{z_G}$ via aggregating node embeddings or anonymous random walks. Predict label based on graph embedding 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 and Anonymous Walk Embeddings**