Node Embedding (optional)

COMP9312_23T2

UNSW SYDNEY

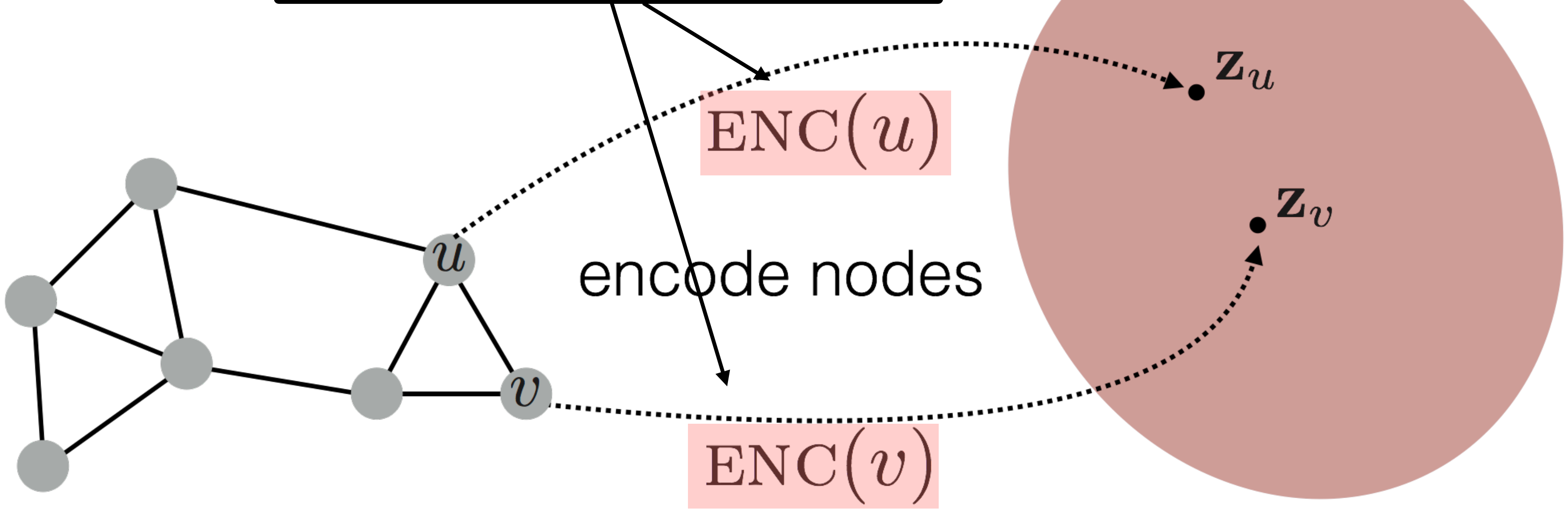
Embedding Nodes

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

in the original network

Similarity of the embedding

Need to define!



original network

embedding space

Decoder: Node Similarity

- Key choice of methods is **how they define node similarity.**

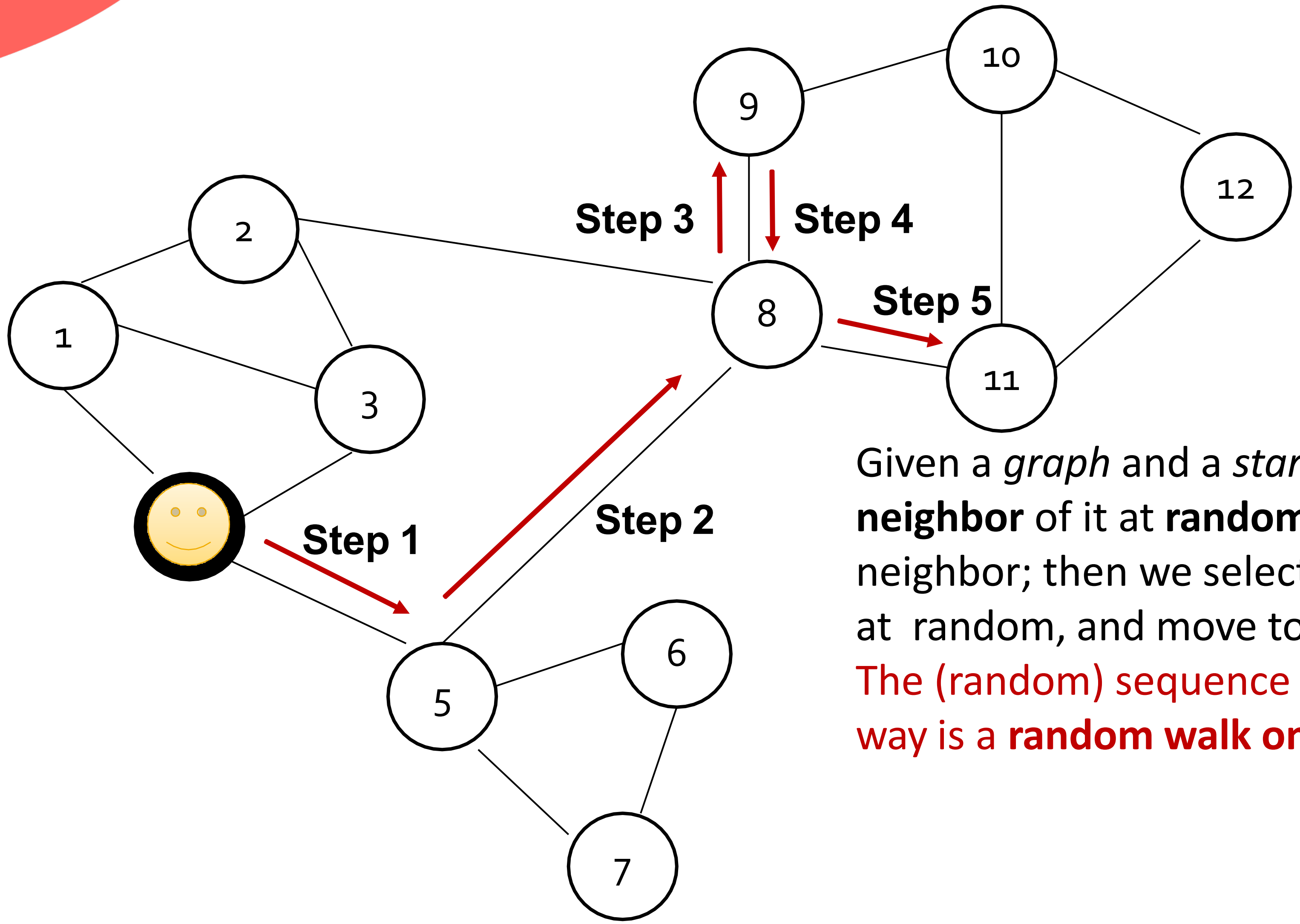
- Should two nodes have a similar embedding if they…
  - are linked?
  - share neighbors?
  - have similar “structural roles”?
- We will now learn node similarity definition that uses **random walks**, and how to optimize embeddings for such a similarity measure.

Train/Optimize Node Embeddings via Random Walks

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

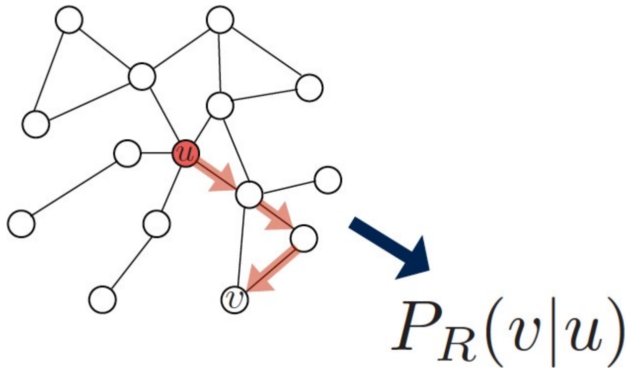
The (random) sequence of points visited this way is a **random walk on the graph**.

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

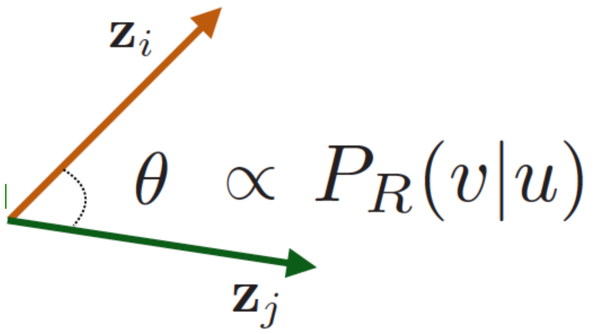
Random-Walk Embeddings

1. Estimate probability of visiting node $\boldsymbol{v}$ 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 ra 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: Loss

- 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 $N_R(u)$

Feature Learning: Loss (cont)

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 $NR(u)$, the multiset* of nodes visited on random walks starting from u
3. Optimize embeddings according to: Given a node u, predict its neighbors $N_R(u)$

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

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

Feature Learning: Loss (cont)

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: $\sum_i \exp(x_i) \approx \max_i \exp(x_i)$

Feature Learning: Loss (cont)

Putting it all together:

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

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
$O(|V|^2)$ complexity!

Negative Sampling

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

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

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

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

random distribution over nodes

Instead of normalizing w.r.t. all nodes, just normalize against k random "**negative samples**" n_i. In practice k =5-20

Training: 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

SGD (cont)

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!

Node2Vec

How to 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.*

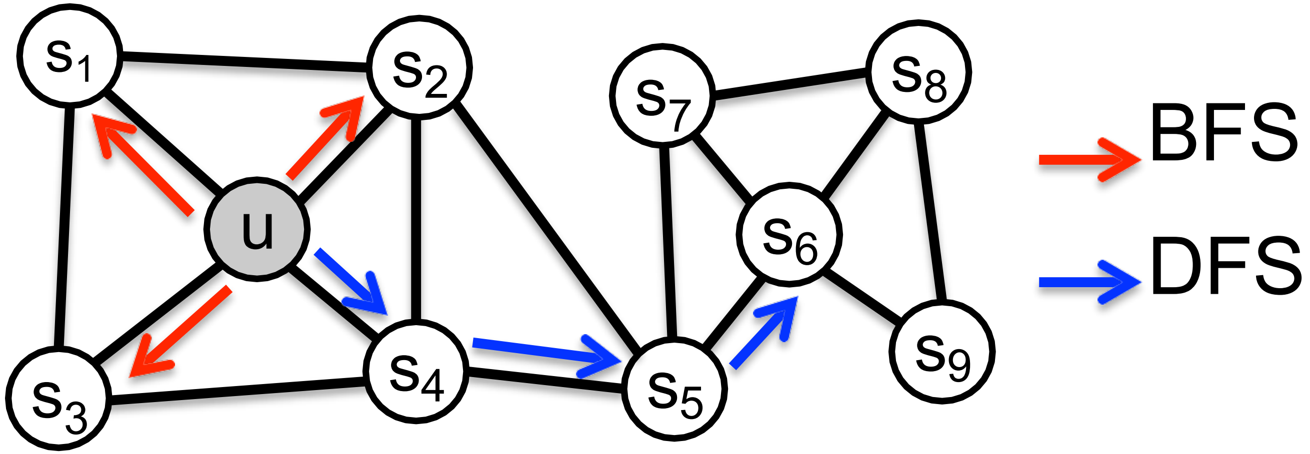
Overview of node2vec

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

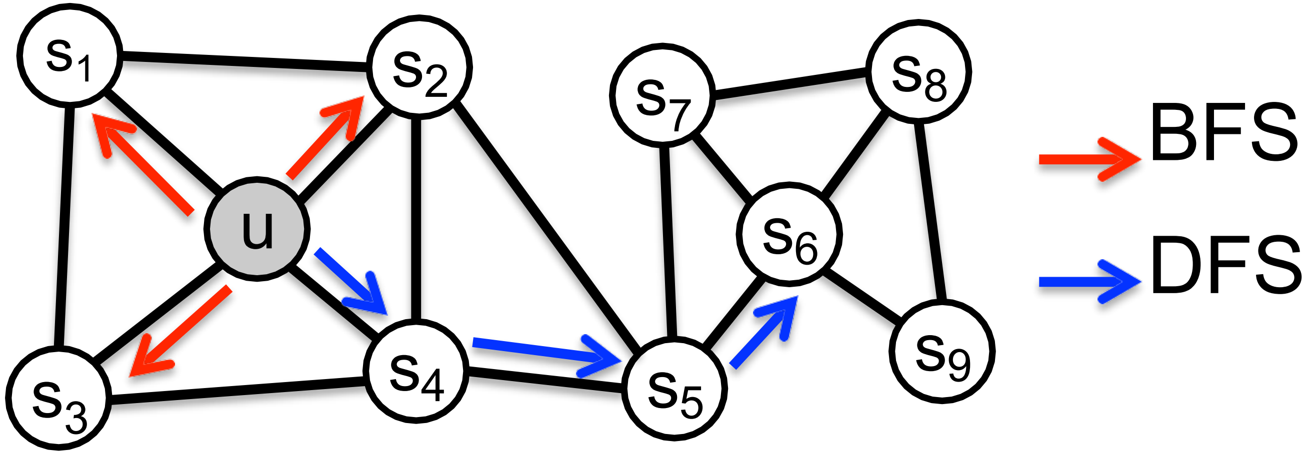
node2vec: biased walks

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 node u:

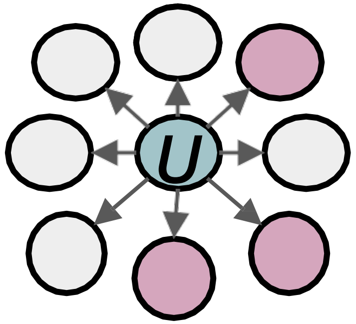


Walk of length 3 ($N_R(u)$ of size 3):

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

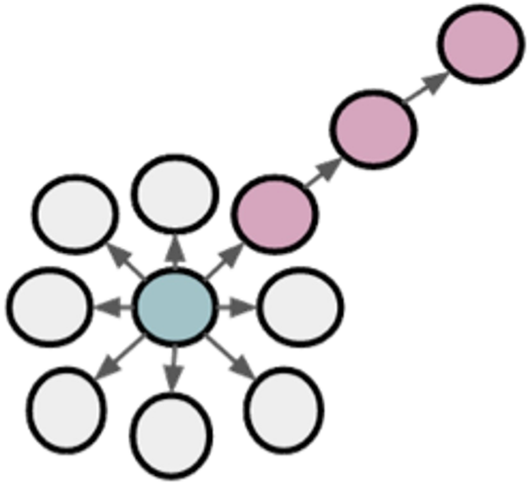
$N_{DFS}(u) = \{ s_4, s_5, s_6\}$ Global macroscopic view

BFS vs DFS



BFS:

Micro-view of neighbourhood



DFS:

Macro-view of neighbourhood

Interpolating BFS and DFS

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

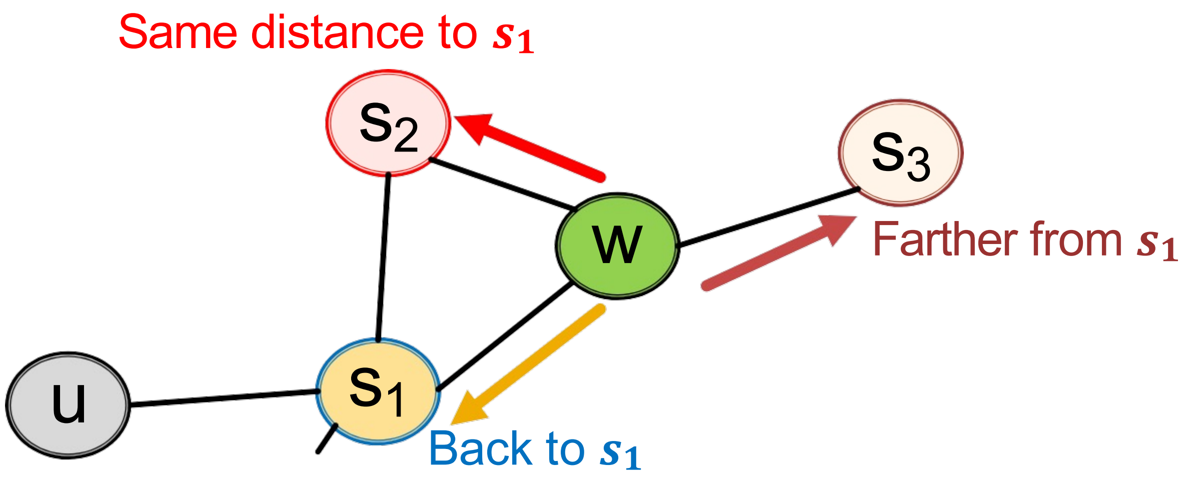
Two parameters:

- **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

node2vec: biased walks

Biased 2nd-order random walks explore network neighborhoods:

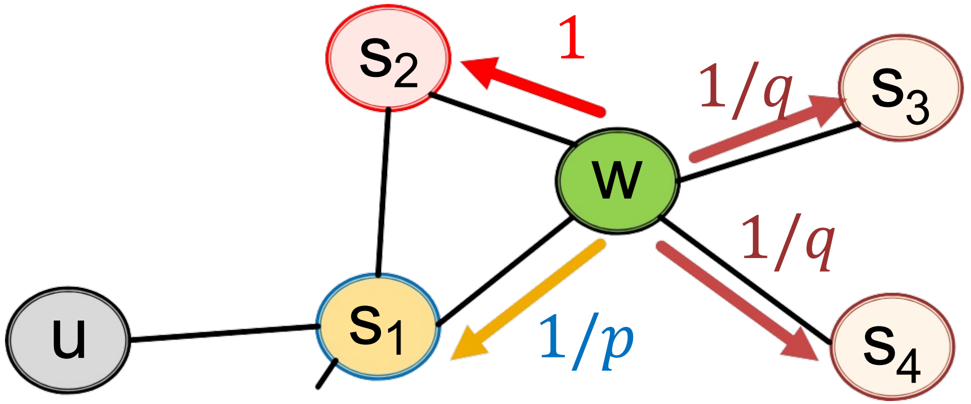
- Random walk just traversed edge (s_1, w) and is now at w
- **Insight:** Neighbors of w can only be:



Idea: Remember where the walk came from

node2vec: biased walks

Walker came over edge $(\mathbf{s_1}, \mathbf{w})$ and is at $\mathbf{w}$. Where to go next?



$\mathrm{w} \rightarrow$

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 Methods

- **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 Node Embedding

- **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 the former topic)
  - Random walk approaches **(covered today)**

Summary of Node Embedding (cont)

- **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!