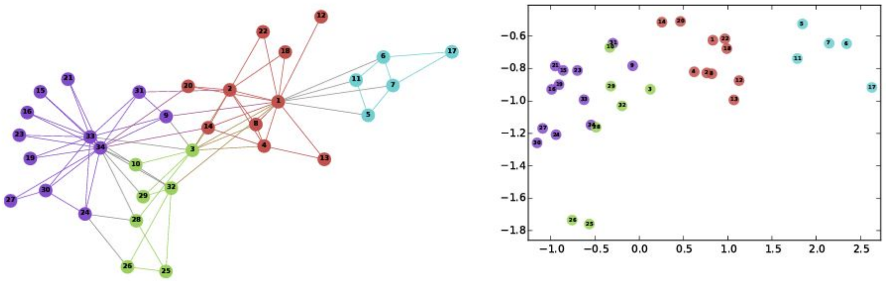


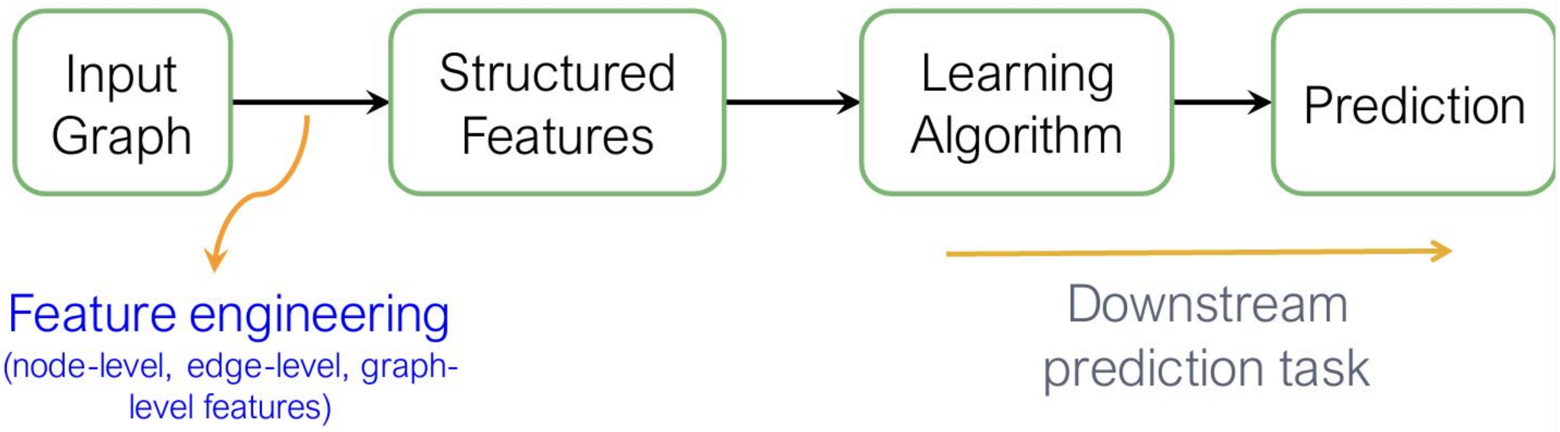
(a) Input: Karate Graph (b) Output: Representation

Node Embeddings
DeepWalk, Node2Vec

Jay Urbain, Ph.D.
Electrical Engineering and Computer Science Department
Milwaukee School of Engineering
Credits: Perozzi et al . DeepWalk : Online Learning of Social Representations. KDD 2014.

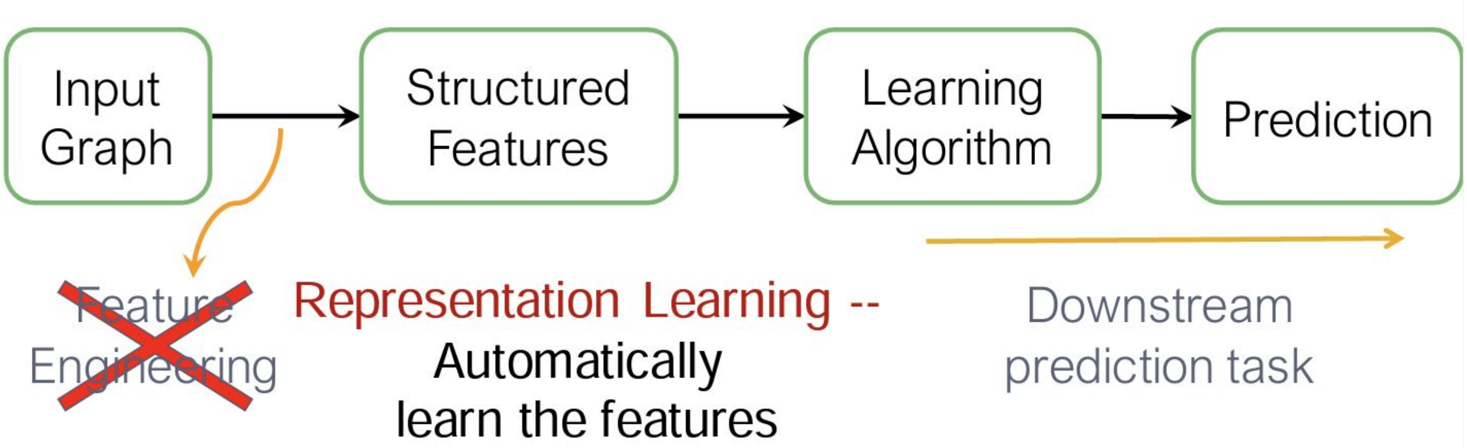
Traditional Machine Learning

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



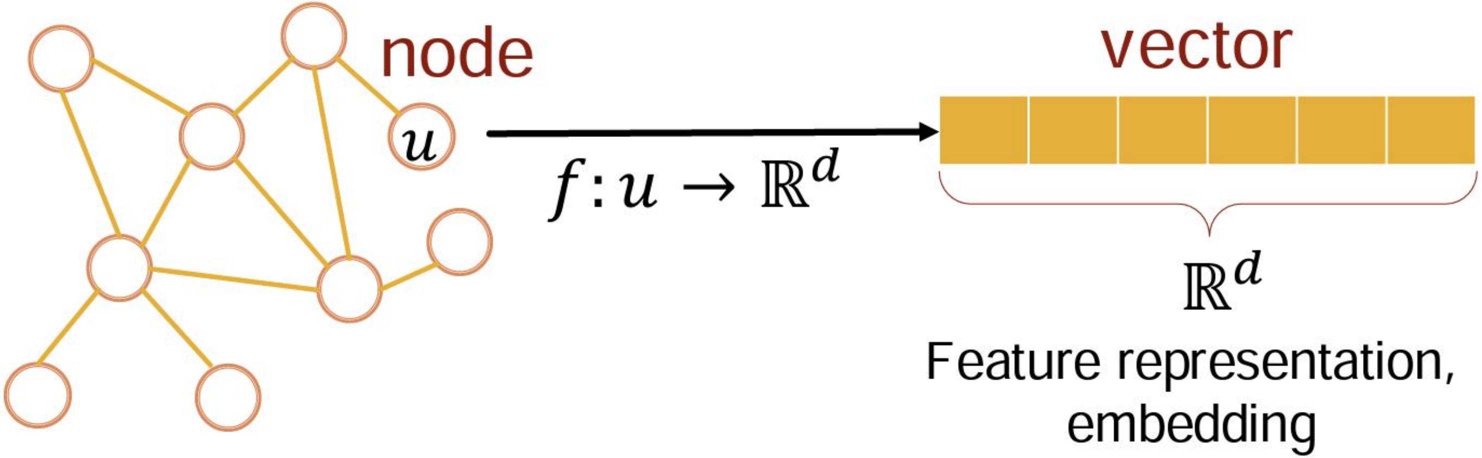
Graph Representation Learning

Graph representation learning alleviates the need to do feature engineering every single time.



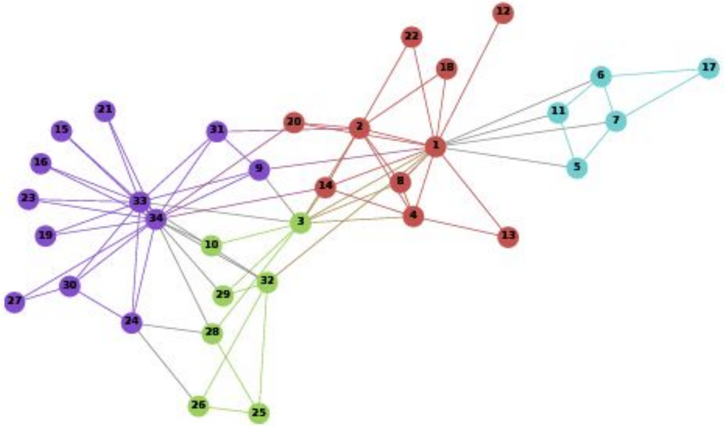
Graph Representation Learning

Goal: efficient task-independent feature learning for ML on graphs.

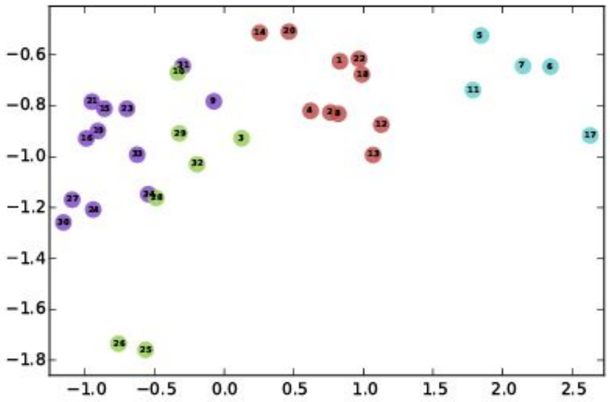


Learn function ***f*** that maps u to an embedding space

Example Node Embedding



(a) Input: Karate Graph

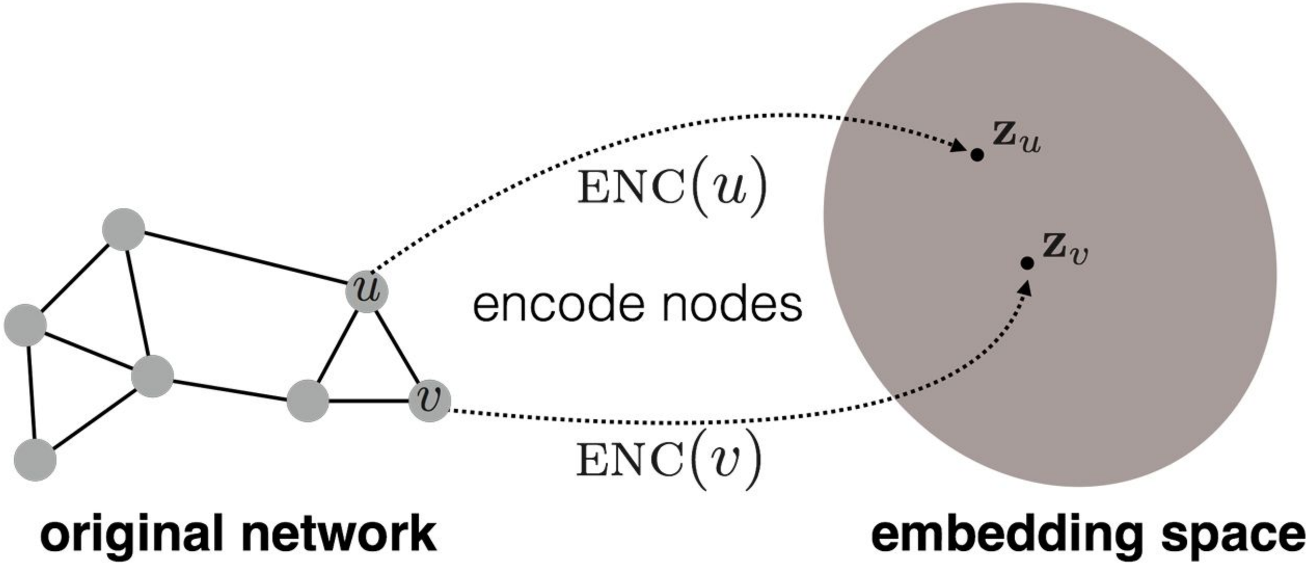


(b) Output: Representation

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

Embedding Nodes

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

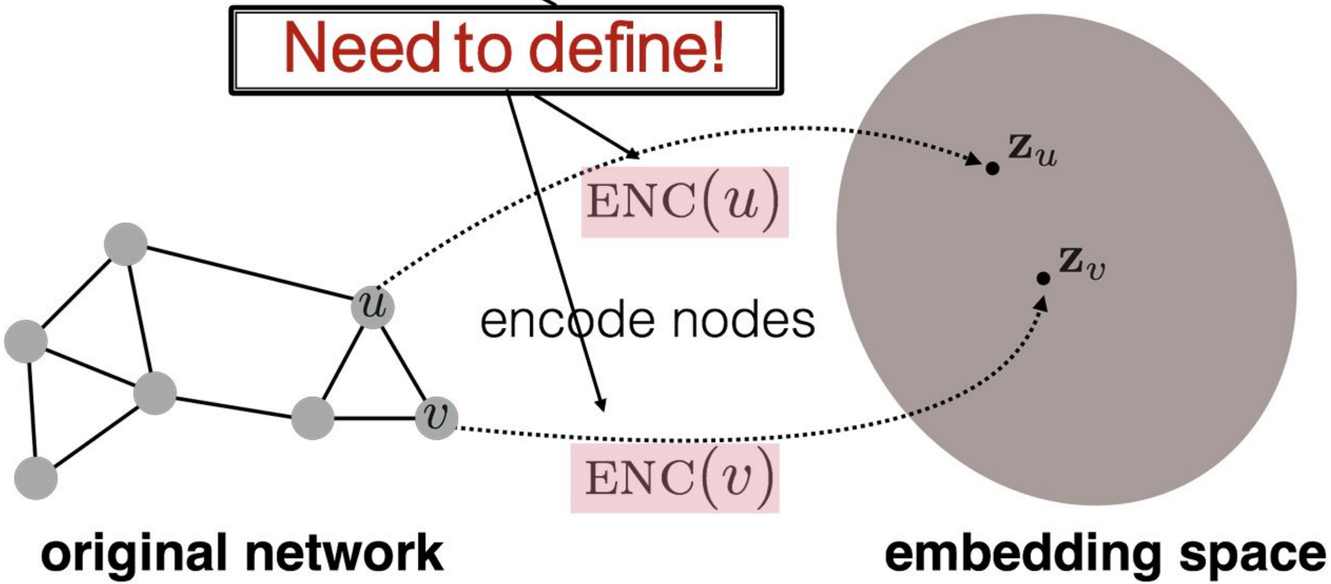


Embedding Nodes

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

- Encoder maps from nodes to embeddings
- Define a node similarity function (i.e., a measure of similarity in the original network)
- Decoder **DEC** maps from embeddings to the similarity score
- Optimize the parameters of the encoder so that:

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

$$\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u}_{\text{Similarity of the embedding}}$$

Two Components

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

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

d-dimensional embedding

- 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

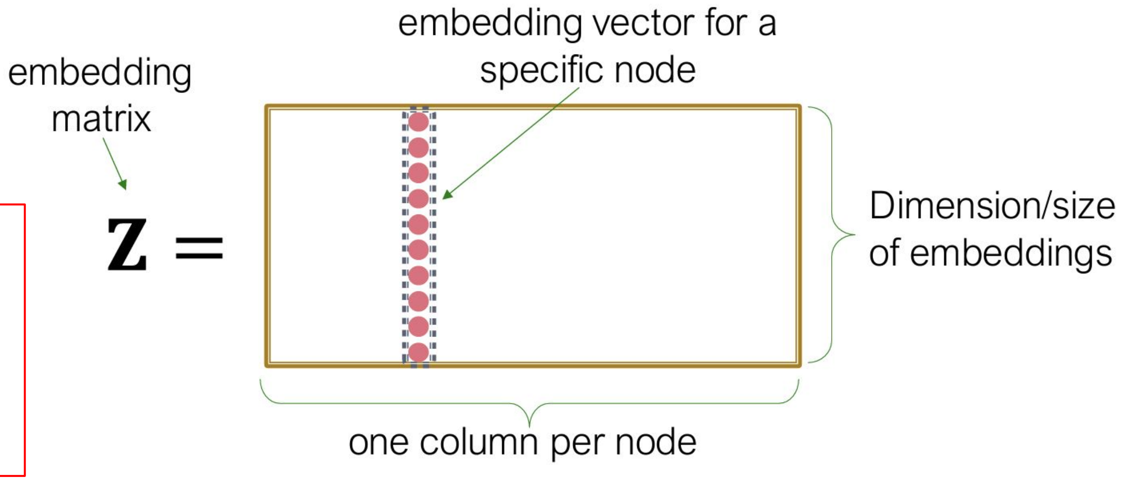
dot product between node embeddings

“Shallow” Encoding

Simple encoding approach: Encoder is just an embedding lookup.

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

Z - matrix,
v - indicator vector



Problems using representations that do not have reduced dimensionality?

“Shallow” Encoding

Simple encoding approach: Encoder is just an embedding lookup.

- Each node is assigned a unique embedding vector
- Need to optimize the embedding of each node
- Methods: **DeepWalk**, **node2vec**

Shallow Encoding Summary

Encoder-Decoder Framework

Shallow encoder: embedding lookup

Parameters to optimize: embeddings $\mathbf{z}_u$, $\mathbf{Z}$ which contains embeddings for all nodes $u \in V$

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 similarity?

- Key choice of methods is how they define similarity.
- Should two nodes have a similar embedding if they:
  - are linked?
  - share neighbors?
  - have similar “structural roles”?

Random Walk Node Embeddings

- Unsupervised/self-supervised learning of node embeddings
- Does not utilize node labels
- Does not utilize node features
- Goal: 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

- Vect $\boldsymbol{z_u}$ - embedding of node ***u*** (want we need to find).
- Probability $\boldsymbol{P(v|z_u)}$ - model prediction based on $\boldsymbol{Z_u}$
  - I.e., the predicted probability of visiting node ***v*** on random walks starting from node ***u***.
- Non-linear functions to produce proper probabilities:
  - **Softmax** - turn ***k*** real values (model predictions) into ***k*** probabilities that sum to 1

$$\sigma(\mathbf{z})[i] = \frac{e^{\mathbf{z}[i]}}{\sum_{j=1}^{K} e^{\mathbf{z}[j]}}$$

  - **Sigmoid** - turn real values into range (0,1)

$$S(x) = \frac{1}{1+e^{-x}}$$

Sigmoid

2 classes

$$\text{out} = P(Y=\text{class1}|X)$$

SoftMax

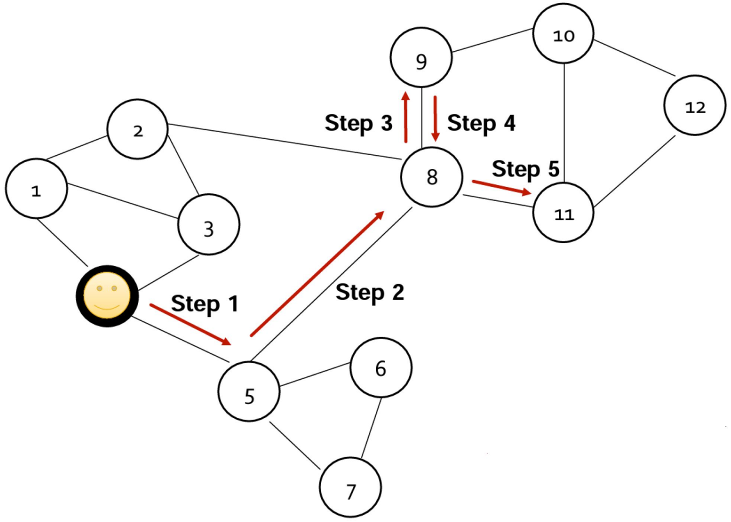
k>2 classes

$$\text{out} = \begin{bmatrix} P(Y=\text{class1}|X) \\ P(Y=\text{class2}|X) \\ P(Y=\text{class3}|X) \\ \vdots \\ P(Y=\text{classk}|X) \end{bmatrix}$$

Random Walk

Given a point graph, and a starting point:

- select a neighbor at random
- move to this neighbor
- then select a neighbor of this point at random, and move to it, etc.



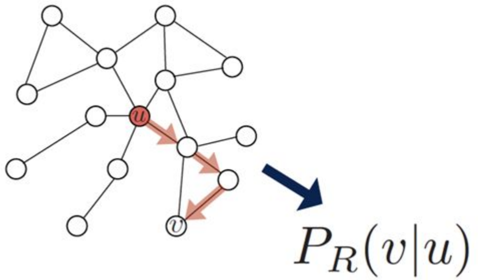
The (random) sequence of points visited this way is a random walk on the graph .

RandomWalk Embeddings

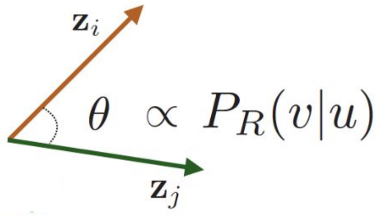
$$\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v \approx$$ probability that ***u*** and ***v*** co-occur on a random walk over the graph

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 random walk "similarity"

Why Random Walks?

- Flexible definition of node similarity incorporates both local and higher order neighborhood information.
- Idea: if random walk starting from node visits $\boldsymbol{v}$ with high probability, $\boldsymbol{u}$ and $\boldsymbol{v}$ are similar (**homophily, influence**).
- Do not need to consider all node pairs when training; only need to consider pairs that co-occur on random walks. **Why?**

Unsupervised Feature Learning

- Intuition: Find embedding of nodes in dimensional space that preserves similarity idea.
- How do we define nearby neighbourhood of random walk strategy R?

Feature Learning as Optimization

- Given node $\boldsymbol{u}$, we want to learn feature representations that are predictive of the nodes in its random walk neighborhood $\boldsymbol{N_R(u)}$.
- Learn mapping function ***f(u) = z_u*** that maximizes the log-likelihood that $\boldsymbol{N_R(u)}$ is ***u*'s** neighborhood**.**

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

Feature Learning as Optimization

- Run short fixed-length random walks starting from each node u according to some random walk strategy ***R***.
- For each node ***u*** collect $\boldsymbol{N_R(u)}$, the multiset of nodes visited on random walks starting from node ***u***.
- Optimize embeddings according to given node u to predict its neighbors $\boldsymbol{N_R(u)}$.

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

Feature Learning as Optimization

- Equivalent to minimizing the following loss:

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

Parameterize P***(v/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).

Random Walk: Summary

1. Run short fixed-length random walks starting from each node on the graph.
2. For each node $\boldsymbol{u}$ collect $\boldsymbol{N_R(u)}$, the multiset of nodes visited on random walks starting from $\boldsymbol{u}$.
3. Optimize embeddings using Stochastic Gradient Descent:

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

Efficiently approximate this using negative sampling.

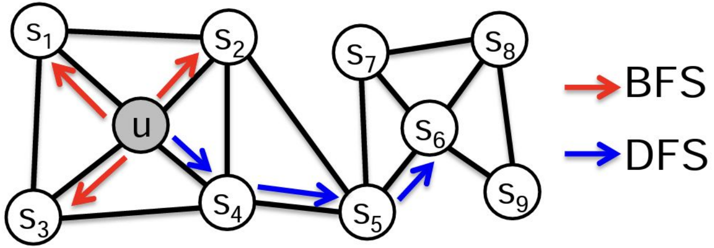
How should we randomly walk?

- Simplest idea: Just run fixed length, unbiased random walks starting from each node.
  - **DeepWalk** from Perozzi et al., 2013

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

Node2Vec: Biased Walks

- Goal: Embed nodes with similar network neighborhoods close in the feature space.
- Key observation: neighborhood embeddings.
- Idea: Use flexible, biased random walks that can trade off between local and global network view.



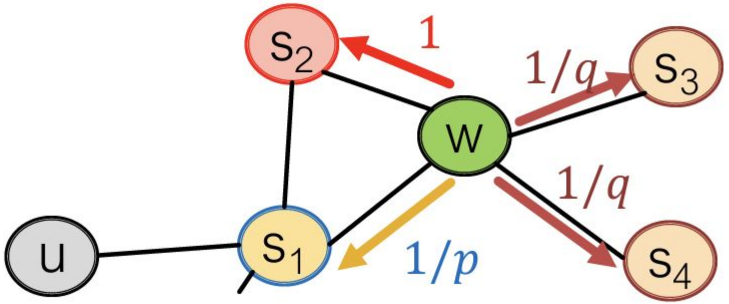
$N_{BFS}(u) = \{ s_1, s_2, s_3 \}$ Local microscopic view

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

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

Node2Vec: Biased Walks

- Return parameter $\boldsymbol{p}$:
  - Return back to the previous node
  - Intuitively, random restart
- In-out parameter $\boldsymbol{q}$:
  - Moving outwards (DFS) vs. inwards (BFS)
  - Intuitively, q is the "ratio" of BFS vs. DFS

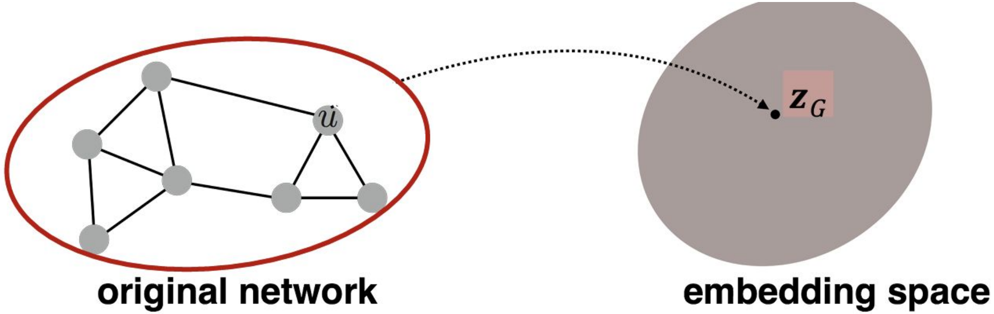


Other Random Walk Ideas

- Different kinds of biased random walks:
  - Based on node attributes (Dong et al., 2017).
  - Based on learned weights (AbuEl-Haija et al., 2017).
- Alternative optimization schemes:
  - Directly optimize based on 1-hop and 2-hop random walk probabilities (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 2016's HARP).

Embedding Entire Graphs

- Can generalize to graph using summation over all node embeddings, learn super nodes, etc.
- Toxic vs. non-toxic molecule.
- Identify anomalous graphs



Summary

- Core idea: Embed nodes so that distances in embedding space reflect node similarities in the original network.
- Different notions of node similarity. 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).
- Must choose definition of node similarity that matches your application.