Graph Embeddings

Natalia Semenova

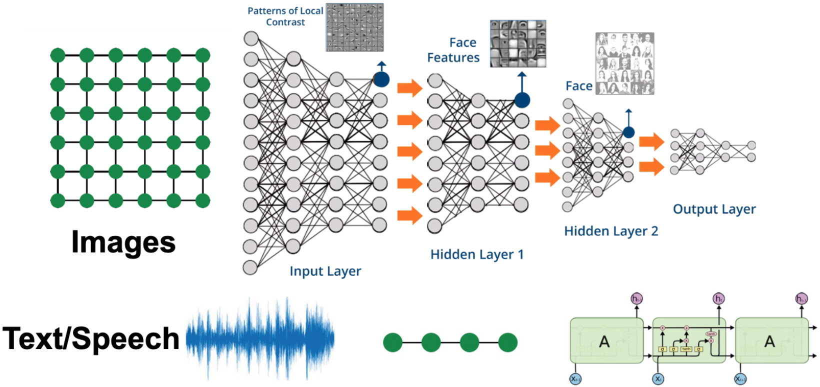
Higher School of Economics

nasemenova_2@edu.hse.ru

Structural Analysis and Visualization of Networks
19.03.2024

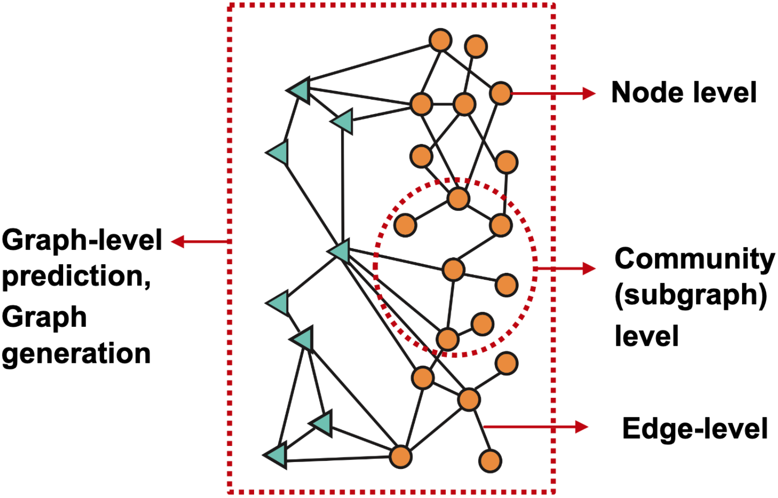
Presentation Overview

ML toolbox



Modern deep learning toolbox is designed for simple sequences & grids

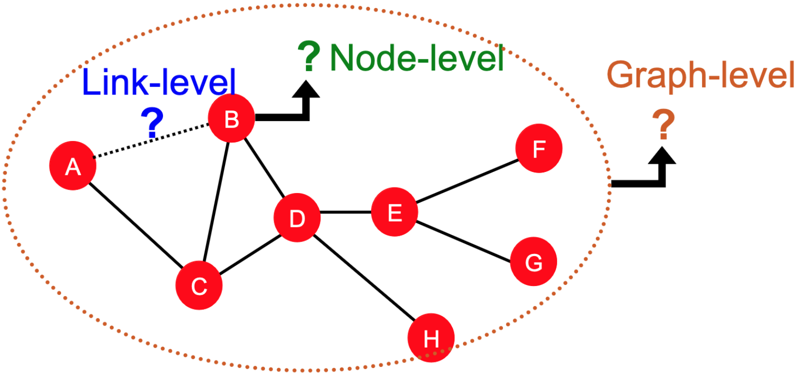
Task Levels



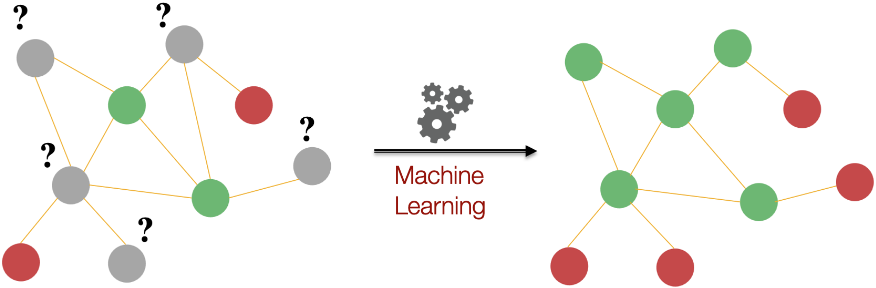
Node level
Graph-level prediction, Graph generation
Community (subgraph) level
Edge-level

Task Levels

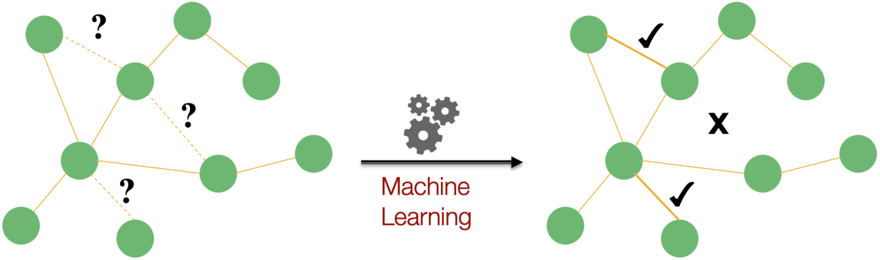
- Node-level prediction
- Link-level prediction
- Graph-level prediction



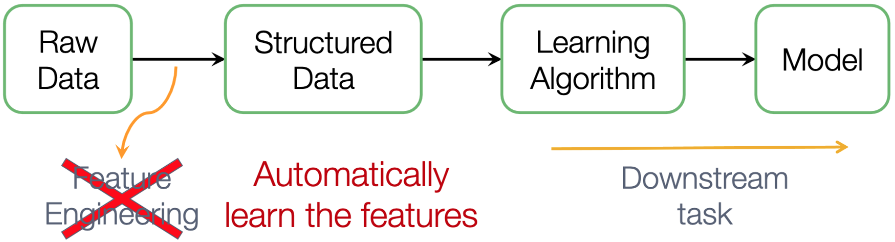
Node Classification



Link Prediction

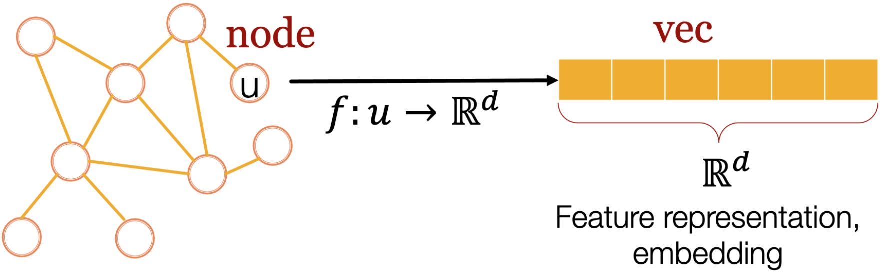


ML pipeline



(Supervised) Machine Learning Lifecycle requires feature engineering every single time!

Feature learning in Graphs



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

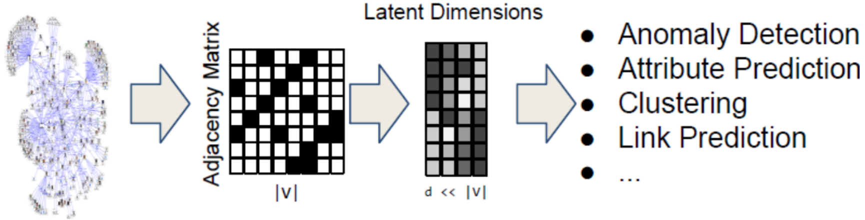
Why network embedding?

Task

We map each node in a network into a low-dimensional space

- Distributed representations for nodes
- Similarity of embeddings between nodes indicates their network similarity
- Encode network information and generate node representation

-



Example node embedding

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

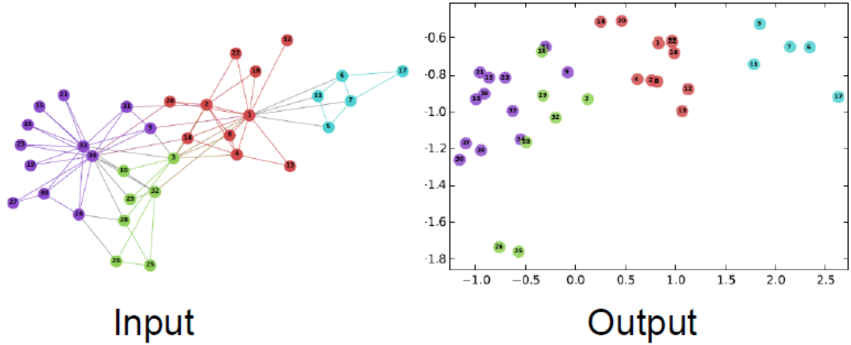


Image from: Perozzi et al. **DeepWalk**: Online Learning of Social Representations. KDD 2014

Why is it hard?

networks are far more complex

- Complex topographical structure (i.e., no spatial locality like grids)

No fixed node ordering or reference point (i.e., the isomorphism problem)

Often dynamic and have multimodal features.

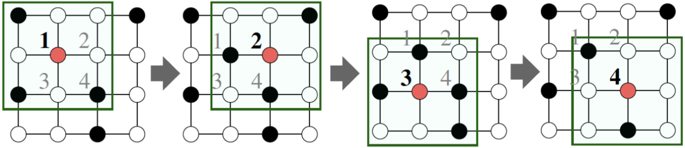


Image from: Perozzi et al. **DeepWalk**: Online Learning of Social Representations. KDD 2014

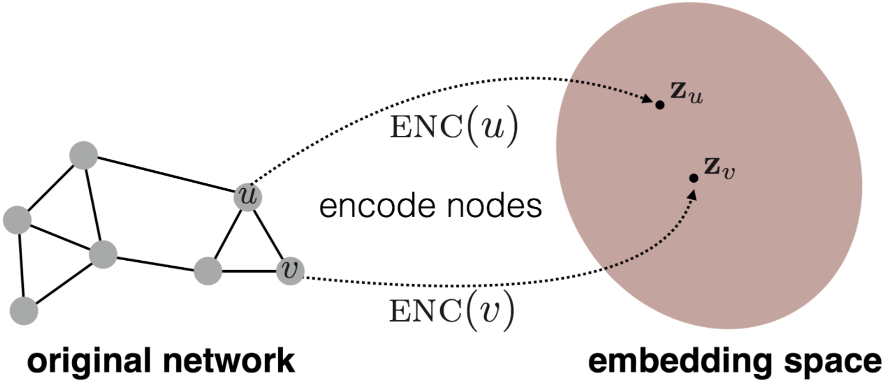
Setup

Assume we have a graph G:

- V is the vertex set.
- A is the adjacency matrix (assume binary).
- No node features or extra information is used!

Embedding nodes

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

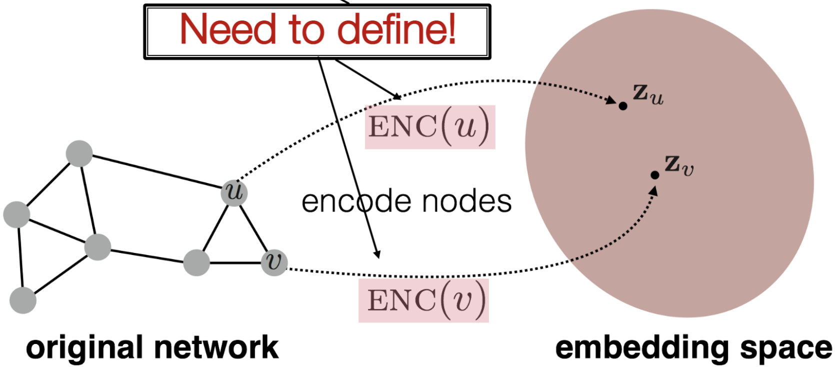


Embedding nodes

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

in the original network

Similarity of the embedding



Learning embedding nodes

1. Define an encoder (i.e., a mapping from nodes to embeddings)
2. Define a node similarity function (i.e., a measure of similarity in the original network)
3. Optimize the parameters of the encoder so that:

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

Key Components

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

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

d-dimensional embedding

node in the input graph

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

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

Similarity of u and v in the original network

dot product between node embeddings

Shallow embedding

Simplest encoding approach: encoder is just an embedding-lookup

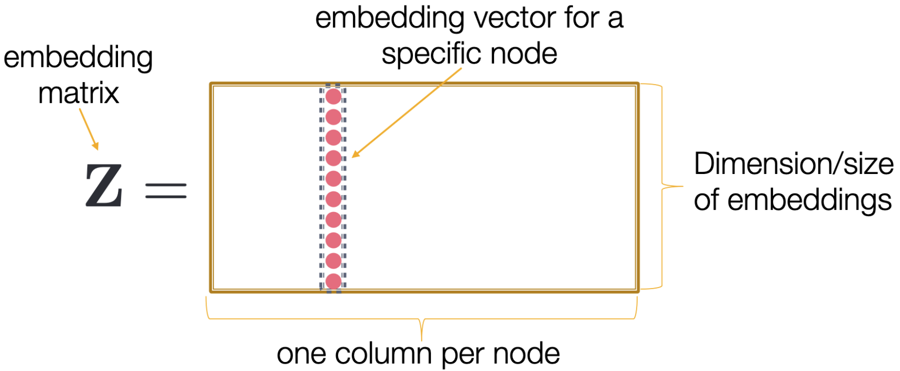
$$\mathrm{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is a node embedding [what we learn!]

$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ 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

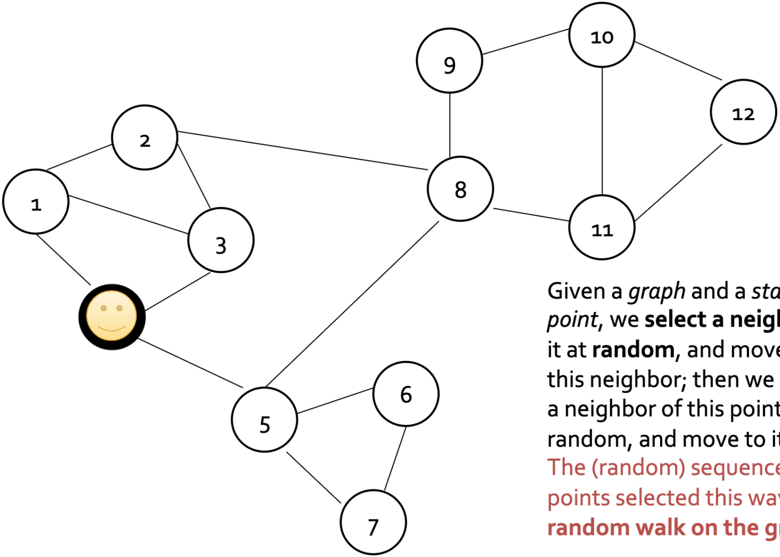
Simplest encoding approach: encoder is just an embedding-lookup

1. Each node is assigned to a unique embedding vector
2. Many methods: DeepWalk, node2vec, TransE

How to define similarity?

1. Key choice of methods is how they **define node similarity**.
2. E.g., should two nodes have similar embeddings if they...
   - are connected?
   - share neighbors?
   - have similar “structural roles”?
   - ...?

Random Walks

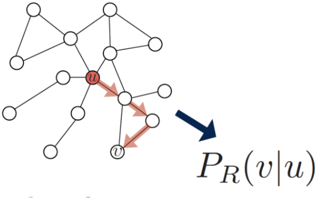


Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points selected this way is a **random walk on the graph**.

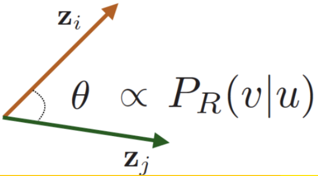
$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$ probability that u and v co-occur on a random walk over the network

Random Walks Embeddings

- Estimate probability of visiting node *v* on a random walk starting from node *u* using some random walk strategy *R*



- Optimize embeddings to encode these random walk statistics: Similarity (here: dot product=cos(θ)) encodes random walk "similarity"



Why Random Walks?

- Expressivity: Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information
- 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 to d-dimensions 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 strategy R

Feature learning as optimization

- Given $G = (V, E)$
- Our goal is to learn a mapping $z : u \rightarrow \mathrm{R}^d$
- Log-likelihood objective:

$$\max_{z} \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | z_u)$$

- where $N_R(u)$ is neighborhood of node u by strategy R
- Given node u, we want to learn feature representations that are predictive of the nodes in its neighborhood $N_R(u)$

Random Walk Optimization

1. Run short fixed-length random walks starting from each node on the graph using some strategy R
2. For each node u collect $N_R(u)$, the multiset[1] of nodes visited on random walks starting from u
3. Optimize embeddings according to: Given node u, predict its neighbors $N_R(u)$

$$\max_{z} \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | z_u)$$

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

Random Walk Optimization

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

1. Intuition: Optimize embeddings to maximize likelihood of random walk co-occurrences
2. Parameterize $P(v|z_u)$ using softmax:

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}$$

Random Walk Optimization

Putting it all together:

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

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 z_u that minimize L

Random Walk Optimization

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

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

Doing this naively is too expensive! We can approximate the normalization term from the softmax

Solution: Negative Sampling

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

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

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

random distribution over all nodes

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

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.

Negative Sampling

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

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

random distribution over all nodes

Sample *k* negative nodes proportional to degree Two considerations for *k* (# negative samples):

1. Higher *k* gives more robust estimates
2. Higher *k* corresponds to higher bias on negative events
3. In practice *k* =5-20

Random Walk by steps

1. Run short fixed-length random walks starting from each node on the graph using some 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 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!

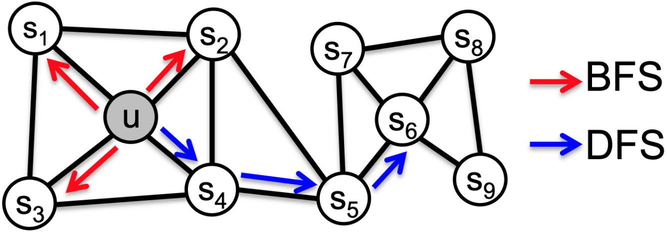
Random Walk: overview

1. we have described how to optimize embeddings given random walk statistics
2. 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?

node2vec: overview

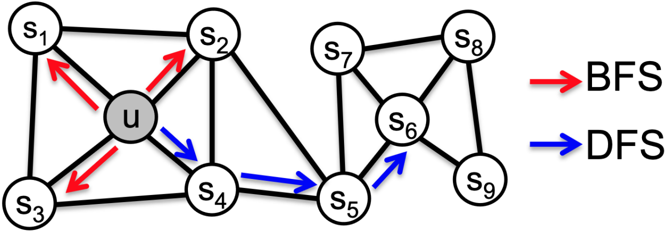
1. Goal: Embed nodes with similar network neighborhoods close in the feature space
2. We frame this goal as a maximum likelihood optimization problem, independent to the downstream prediction task
3. Key observation: Flexible notion of network neighborhood $N_R(u)$ of node u leads to rich node embeddings
4. Develop biased 2nd order random walk R to generate network neighborhood $N_R(u)$ of node u

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

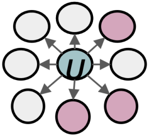
Two classic strategies to define a neighborhood $N_R(u)$ of a given node u:



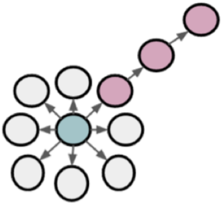
Walk of length 3 ($N_R(u)$ of size 3):

- **Local** microscopic view $N_{BFS}(u) = \{s_1, s_2, s_3\}$
- **Global** macroscopic view $N_{DFS}(u) = \{s_4, s_5, s_6\}$

BFS vs DFS



BFS:
Micro-view of neighbourhood



DFS:
Macro-view of neighbourhood

Interpolating BFS & DFS

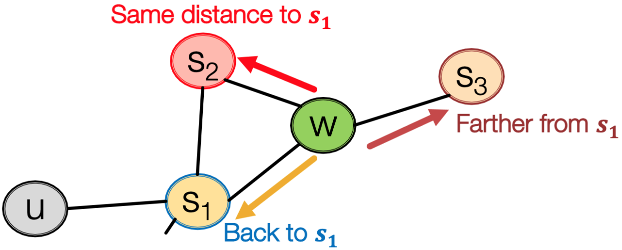
Biased fixed-length random walk R that given a node u generates neighborhood $N_R(u)$

- 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:

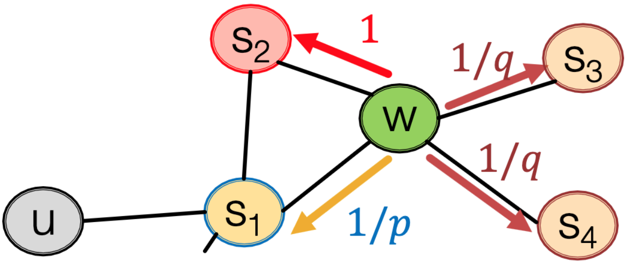


Idea: Remember where that walk came from

Biased Random Walks

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

- p and q model transition probabilities
- $p\ldots$ return parameter
- q walk away" parameter
- $1/p$, $1/q$, 1 are unnormalized probabilities

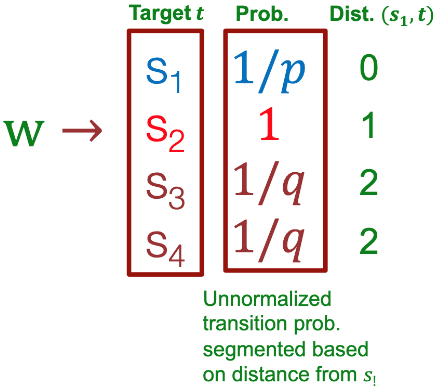


Idea: Remember where that walk came from

Biased Random Walks

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

- 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

- Compute random walk probabilities
- Simulate r random walks of length l starting from each node u
- Optimize the node2vec objective using Stochastic Gradient Descent
- All 3 steps are individually parallelizable

How to use embeddings

- Clustering/community detection: Cluster points
- Node classification: Predict label $f(z_i)$ of node i based on z_i
- Link prediction: Predict edge (i, j) based on $f(z_i, z_j)$
  - 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: $f(z_i, z_j) = g(||z_i - z_j||_2)$

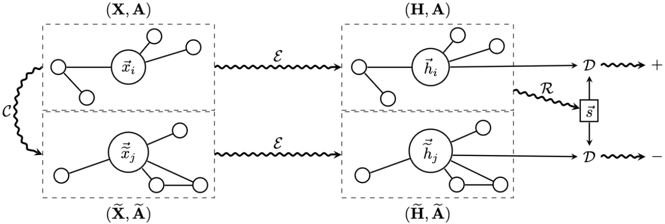
Summary

Basic idea: Embed nodes so that distances in embedding space reflect node similarities in the original network. Different notions of node similarity:

- Adjacency-based (i.e., similar if connected)
- Multi-hop similarity definitions
- Random walk approaches

Deep Graph Infomax

Basic idea: DGI relies on maximizing mutual information between patch representations and corresponding high-level summaries of graphs



$$\mathcal{R}(\mathbf{H}) = \sigma\left(\frac{1}{N}\sum_{i=1}^{N}\vec{h}_i\right)$$

$$\mathcal{D}(\vec{h}_i, \vec{s}) = \sigma\left(\vec{h}_i^T \mathbf{W} \vec{s}\right)$$