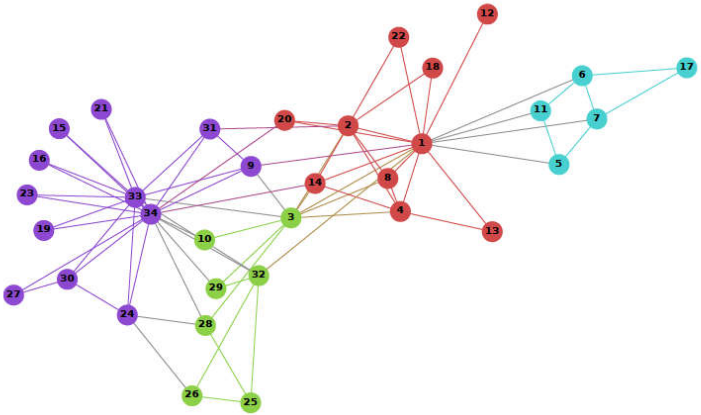
# Node Embeddings

# Content

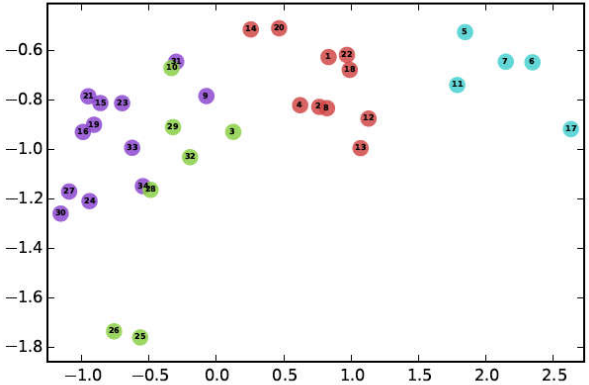
- 1) Node embeddings
  - Map nodes to low-dimensional embeddings.
- 2) Graph neural networks
  - Deep learning architectures for graph-structured data
- 3) Applications

Using talk "Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018"

# Embedding Nodes



Input




Output

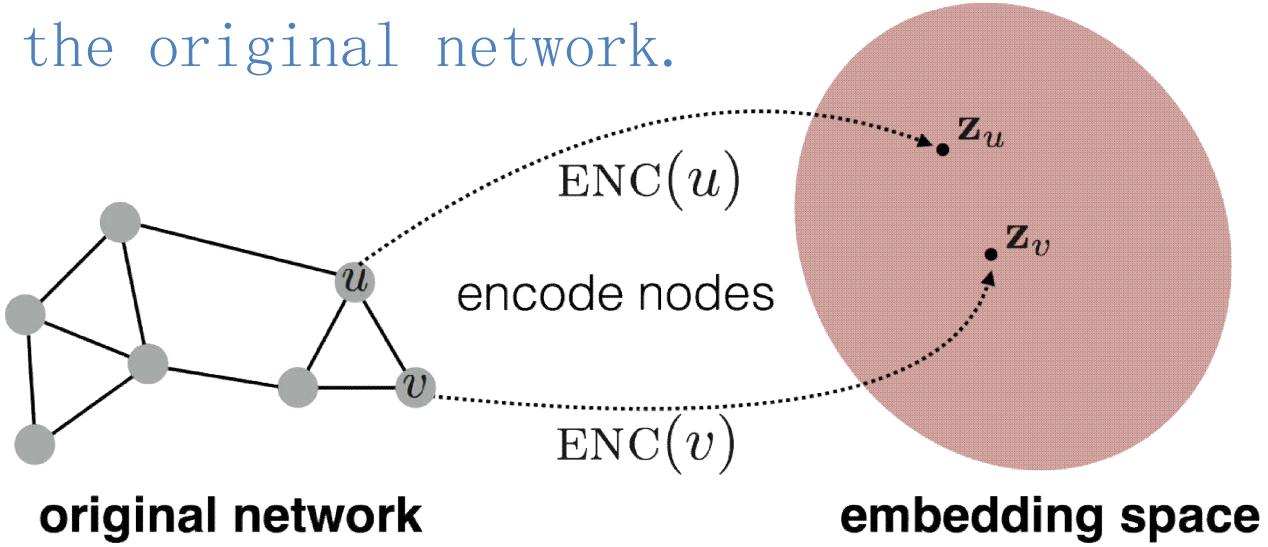
Intuition: Find embedding of nodes to d-dimensions so that "similar" nodes in the graph have embeddings that are close together.

# Setup

- Assume we have a graph $\mathrm{G}$:
  - $\mathrm{V}$ is the vertex set.
  - $\mathbf{A}$ is the adjacency matrix (assume binary).
  - **No node features or extra information is used!**

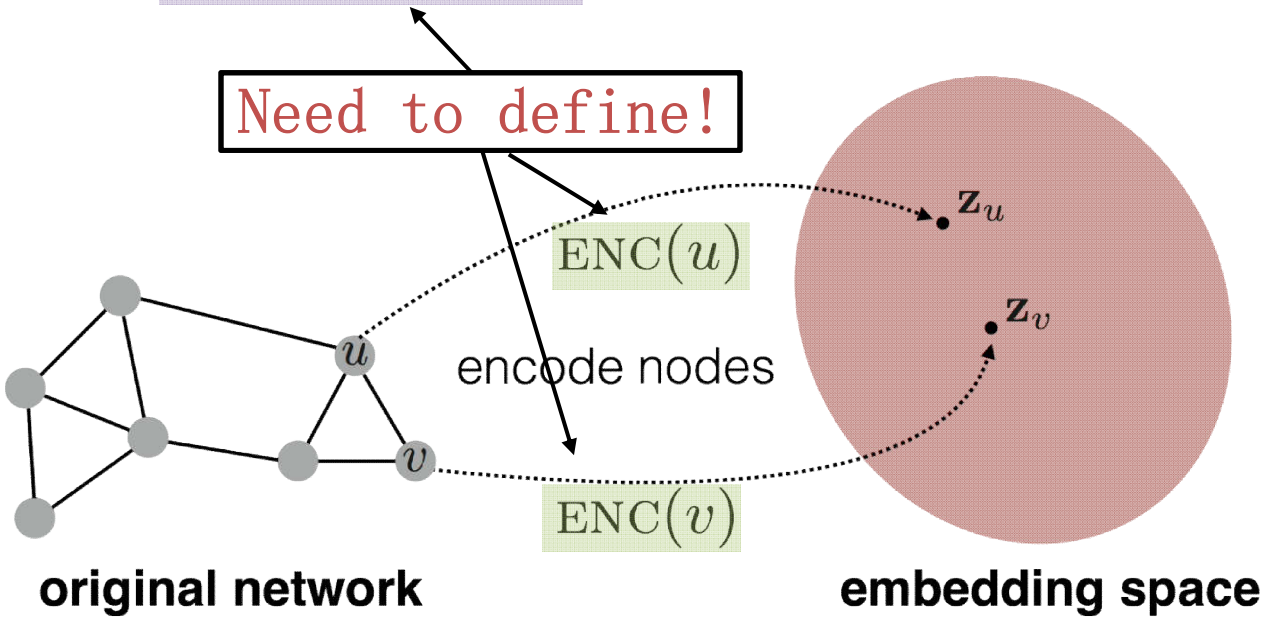
# Embedding Nodes

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



# Learning Node Embeddings

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:**

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

# Two Key Components

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

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

d-dimensional embedding

node in the input graph

- **Similarity function** specifies how relationships in vector space map to 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" Encoding

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

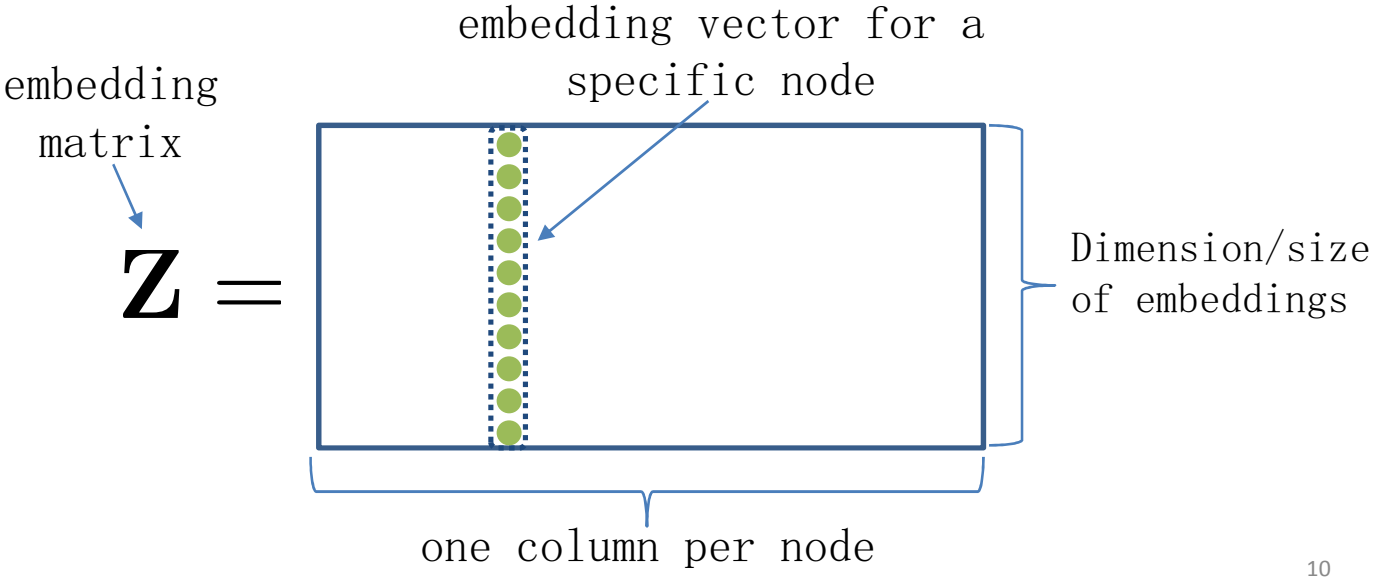
$$\mathrm{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is 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.**

**i.e., each node is assigned a unique embedding vector.**

- E.g., node2vec, DeepWalk, LINE

# “Shallow” Encoding

- Simplest encoding approach: **encoder is just an embedding-lookup.**
- **We will focus on shallow encoding in this section…**
- After that we will discuss more encoders based on deep neural networks.

# How to Define Node Similarity?

- Key distinction between "shallow" methods is **how they define node similarity.**
- E.g., should two nodes have similar embeddings if they….
  - are connected?
  - share neighbors?
  - have similar "structural roles"?
  - …?

# Outline of This Section

1. Adjacency-based similarity
2. Multi-hop similarity
3. Random walk approaches

High-level structure and material from:

- Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.

# Adjacency-based Similarity

Material based on:

- Ahmed et al. 2013. Distributed Natural Large Scale Graph Factorization. *WWW.*

# Adjacency-based Similarity

- Similarity function is just the edge weight between $u$ and $v$ in the original network.
- Intuition: Dot products between node embeddings approximate edge existence.

$$\mathcal{L} = \sum_{(u,v)\in V\times V} \| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v} \|^2$$

loss (what we want to minimize)

sum over all node pairs

embedding similarity

(weighted) adjacency matrix for the graph

# Adjacency-based Similarity

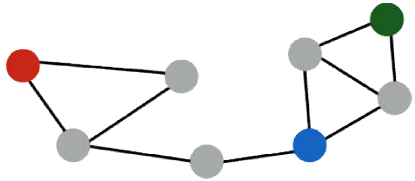
$$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}\|^2$$

- Find embedding matrix $\mathbf{Z} \in \mathbb{R}^{d \times |V|}$ that minimizes the loss $\mathcal{L}$
  - Option 1: Use stochastic gradient descent (SGD) as a general optimization method.
    - Highly scalable, general approach
  - Option 2: Solve matrix decomposition solvers (e.g., SVD or QR decomposition routines).
    - Only works in limited cases.

# Adjacency-based Similarity

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \left\| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v} \right\|^2$$

- **Drawbacks**:
  - O(|V|$^2$) runtime. (Must consider all node pairs.)
    - Can make O([E|) by only summing over non-zero edges and using regularization (e.g., Ahmed et al., 2013)
  - O(|V|) parameters! (One learned vector per node).
  - Only considers direct, local connections.



e.g., the blue node is obviously more similar to green compared to red node, despite none having direct connections.

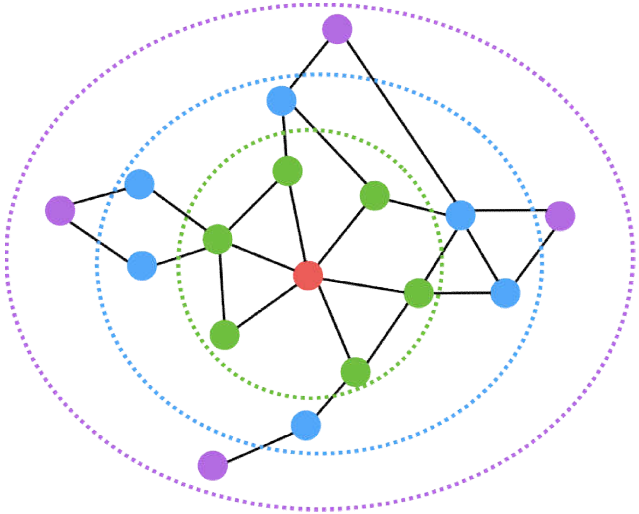
# Multi-hop Similarity

Material based on:

- Cao et al. 2015. GraRep: Learning Graph Representations with Global Structural Information. *CIKM.*
- Ou et al. 2016. Asymmetric Transitivity Preserving Graph Embedding. *KDD.*

# Multi-hop Similarity

- **Idea:** Consider k-hop node neighbors.
  - E.g., two or three-hop neighbors.



- Red: Target node
- Green: 1-hop neighbors
  - $\mathbf{A}$ (i.e., adjacency matrix)
- Blue: 2-hop neighbors
  - $\mathbf{A}^2$
- Purple: 3-hop neighbors
  - $\mathbf{A}^3$

# Multi-hop Similarity

- **Basic idea:** $$\mathcal{L} = \sum_{(u,v)\in V\times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}^k_{u,v}\|^2$$
- Train embeddings to predict k-hop neighbors.
- In practice (GraRep from Cao et al, 2015):
  - Use log-transformed, probabilistic adjacency matrix:

$$\tilde{\mathbf{A}}^k_{i,j} = \max\left(\log\left(\frac{(\mathbf{A}_{i,j}/d_i)}{\sum_{l\in V}(\mathbf{A}_{l,j}/d_l)^k}\right)^k - \alpha, 0\right)$$

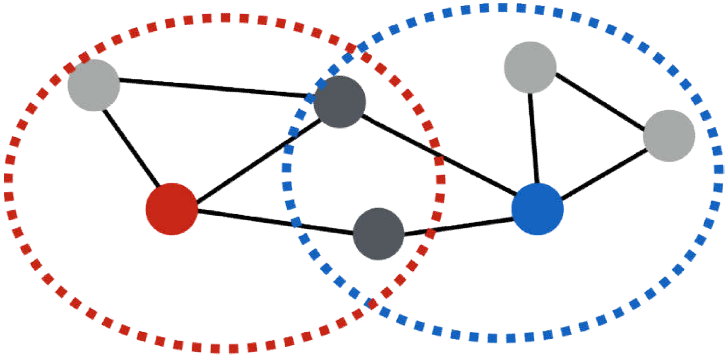
node degree

constant shift

  - Train multiple different hop lengths and concatenate output.

# Multi-hop Similarity

- **Another option: Measure overlap between node neighborhoods.**



- Example overlap functions:
  - Jaccard similarity
  - Adamic-Adar score

# Multi-hop Similarity

$$\mathcal{L} = \sum_{(u,v)\in V\times V} \| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{S}_{u,v} \|^2$$

embedding similarity

multi-hop network similarity (i.e., any neighborhood overlap measure)

- $\mathbf{S}_{u,v}$ is the neighborhood overlap between $u$ and $v$ (e.g., Jaccard overlap or Adamic-Adar score).
- This technique is known as [HOPE (Yan et al., 2016).]

# Summary so far

- Basic idea so far:
  - 1) Define pairwise node similarities.
  - 2) Optimize low-dimensional embeddings to approximate these pairwise similarities.
- Issues:
  - **Expensive:** Generally $O(|V|^2)$, since we need to iterate over all pairs of nodes.
  - **Brittle**: Must hand-design deterministic node similarity measures.
  - **Massive parameter space:** $O(|V|)$ parameters

# Random Walk Approaches

Material based on:

- Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations. *KDD.*
- Grover et al. 2016. node2vec: Scalable Feature Learning for Networks. *KDD.*

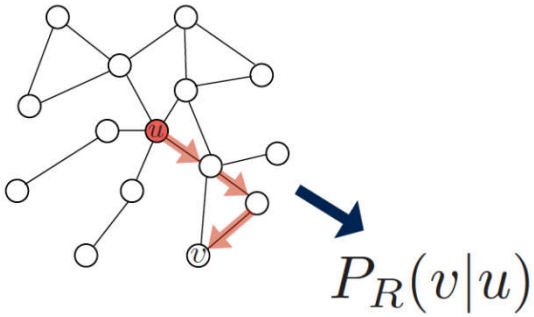
# Random-walk Embeddings

$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$

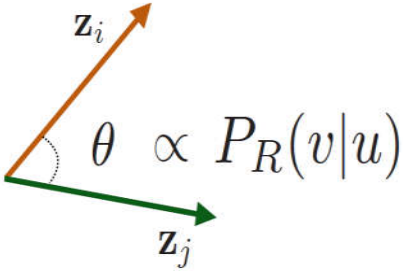
probability that $u$ and $v$ co-occur on a random walk over the network

# 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.



# Why Random Walks?

1. **Expressivity:** Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information.
2. **Efficiency:** Do not need to consider all node pairs when training; only need to consider pairs that co-occur on random walks.

# Random Walk Optimization

1. Run short random walks starting from each node on the graph using some strategy $R$.
2. For each node $u$ collect $\mathrm{N}_R(u)$, the multiset* of nodes visited on random walks starting from $u$.
3. Optimize embeddings to according to:

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

* $\mathrm{N}_R(u)$ can have repeat elements since nodes can be visited multiple times on random walks.

# Random Walk Optimization

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

- Intuition: Optimize embeddings to maximize likelihood of random walk co-occurrences.
- Parameterize $P(v|\mathbf{z}_u)$ 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 things together:

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

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 $\mathbf{z}_u$ that minimize L

# Random Walk Optimization

But doing this naively is too expensive!!

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

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

# Random Walk Optimization

But doing this naively is too expensive!!

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

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

# Negative Sampling

Solution: Negative sampling

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

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

sigmoid function

random distribution over all nodes

i.e., instead of normalizing w.r.t. all nodes, just normalize against **k** random "**negative samples**"

# Negative Sampling

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

random distribution over all nodes

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

- Sample negative nodes proportional to degree.
- Two considerations for **k** (# negative samples):
  1. Higher k gives more robust estimates.
  2. Higher k corresponds to higher prior on negative events.

# Random Walks: Stepping Back

1. Run short random walks starting from each node on the graph using some strategy $R$.

2. For each node $u$ collect $\mathrm{N}_R(u)$, the multiset of nodes visited on random walks starting from $u$.

3. Optimize embeddings to according to:

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

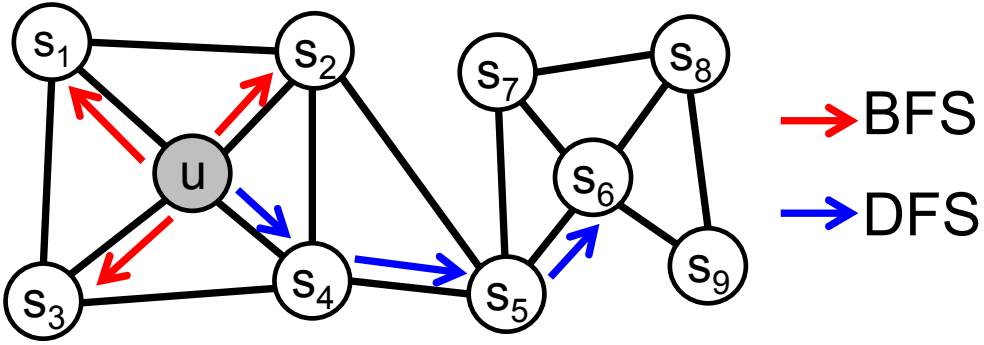
We can efficiently approximate this using negative sampling!

# How should we randomly walk?

- So far we have described how to optimize embeddings given random walk statistics.
- **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).
  - But can we do better?

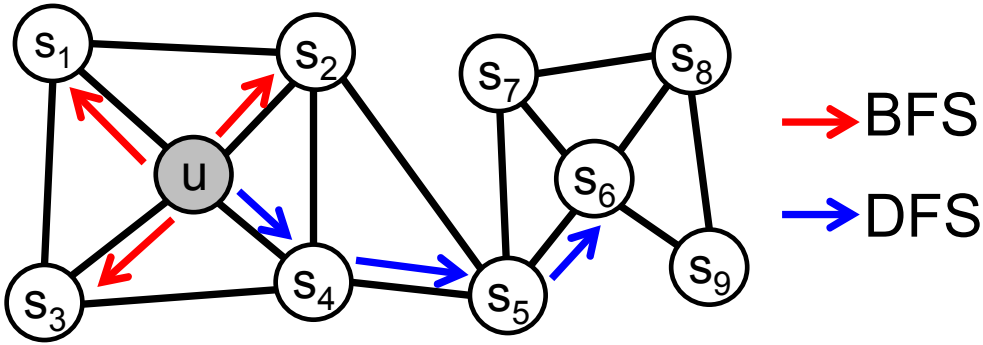
# node2vec: Biased Walks

**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 $u$:




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

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

# Interpolating BFS and DFS

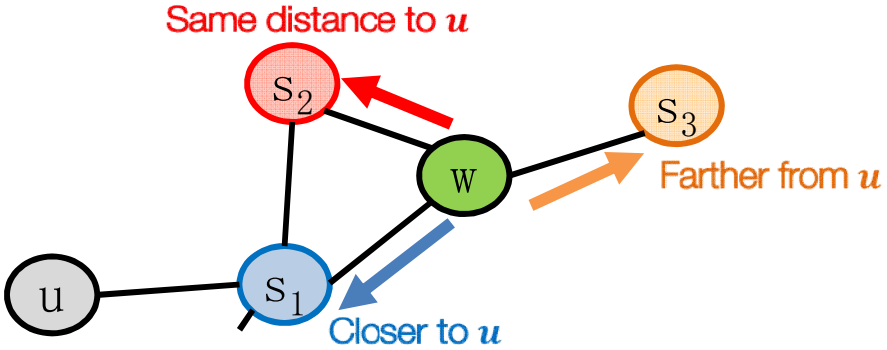
Biased random walk $R$ that given a node $u$ generates neighborhood $N_R(u)$

- Two parameters:
  - Return parameter $p$:
    - Return back to the previous node
  - In-out parameter $q$:
    - Moving outwards (DFS) vs. inwards (BFS)

# Biased Random Walks

Biased 2nd-order random walks explore network neighborhoods:

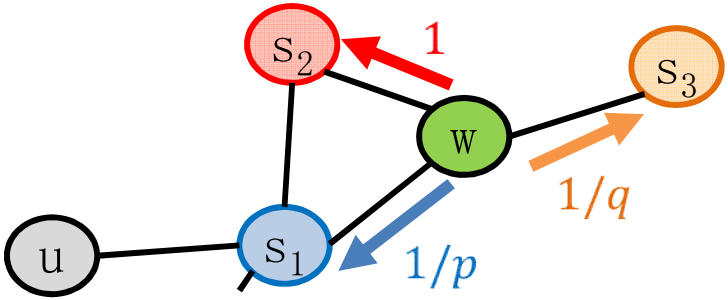
- Rnd. walk started at $u$ and is now at $w$
- **Insight:** Neighbors of $w$ can only be:




**Idea:** Remember where that walk came from

# Biased Random Walks

- Walker is at **w**. Where to go next?

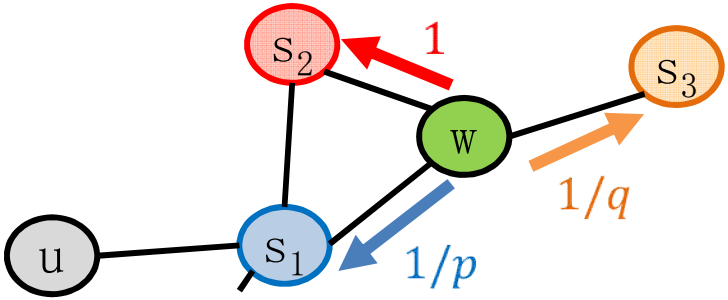



$1/p, 1/q, 1$ are unnormalized probabilities

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

# Biased Random Walks

- Walker is at **w**. Where to go next?



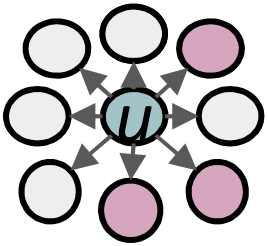

| | | |
|---|---|---|
| **w** → | $s_1$ | $1/p$ |
| | $s_2$ | $1$ |
| | $s_3$ | $1/q$ |

Unnormalized transition prob.

- **BFS-like** walk: Low value of $p$
- **DFS-like** walk: Low value of $q$

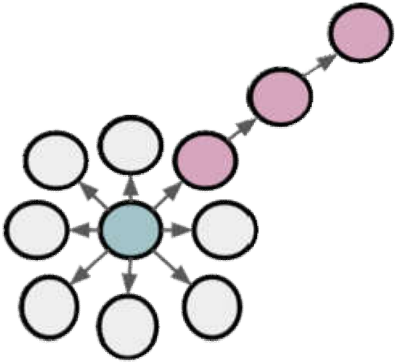
$N_S(u)$ are the nodes visited by the walker

# BFS vs. DFS




BFS:

Micro-view of neighbourhood

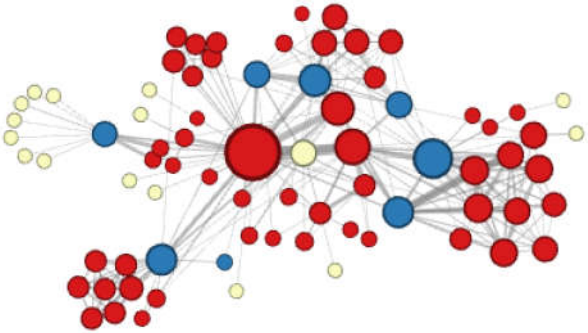


DFS:

Macro-view of neighbourhood

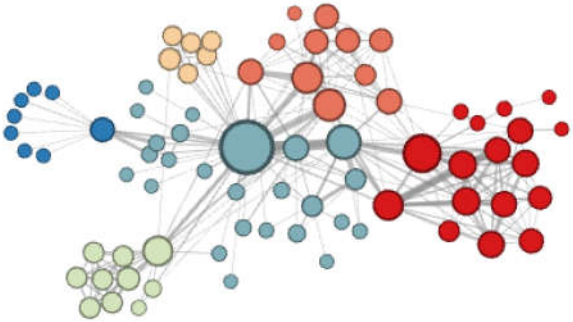
# Experiments: Micro vs. Macro

Interactions of characters in a novel:



p=1, q=2

Microscopic view of the network neighbourhood



p=1, q=0.5

Macroscopic view of the network neighbourhood

# Other random walk ideas (not covered in detail here)

- **Different kinds of biased random walks:**
  - Based on node attributes (Dong et al., 2017).
  - Based on a 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

- **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.

# Summary so far

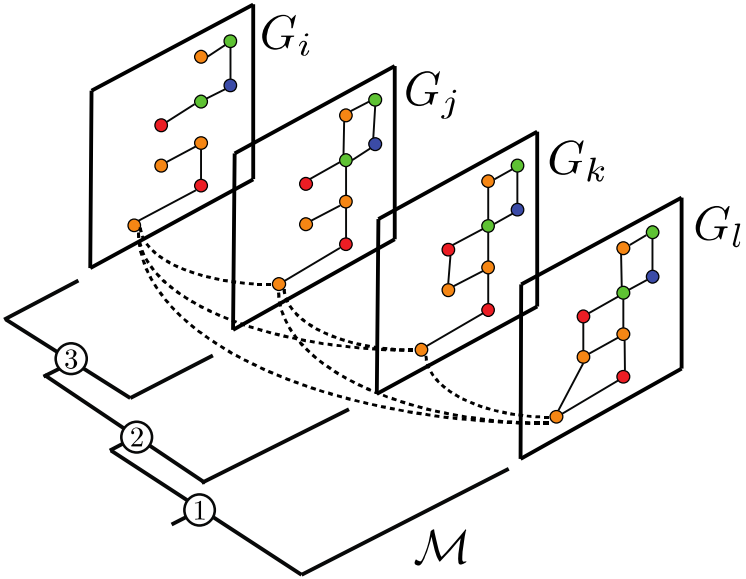
- **So what method should I use..?**
- No one method wins in all cases….
  - e.g., node2vec performs better on node classification while multi-hop methods performs better on link prediction (Goyal and Ferrara, 2017 survey).
- Random walk approaches are generally more efficient (i.e., $O(|E|)$ vs. $O(|V|^2)$)
- **In general:** Must choose def'n of node similarity that matches application!

# Multilayer Networks

Material based on:

- Zitnik and Leskovec. 2017. Predicting Multicellular Function through Multilayer Tissue Networks. *ISMB.*

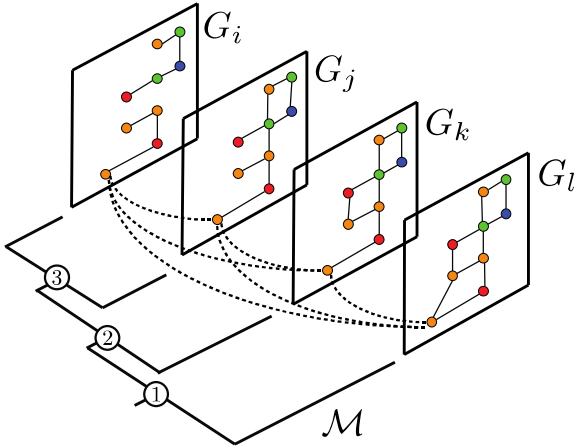
# Multilayer Networks




Let's generalize to multilayer networks!

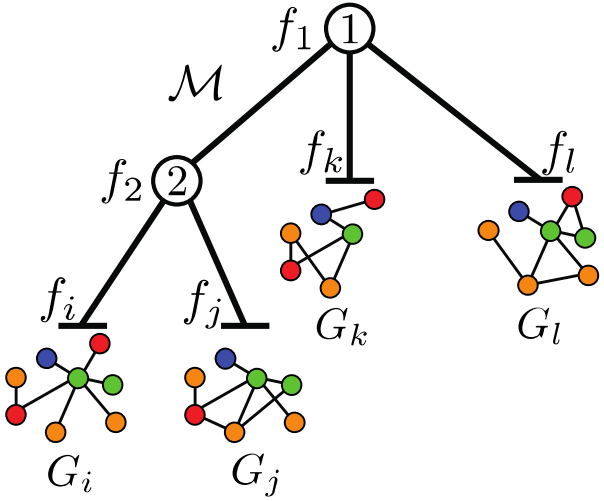
# Multilayer Networks

- Each network is a layer $G_i = (V_i, E_i)$
- Similarities between layers are given in hierarchy $\mathcal{M}$, map $\pi$ encodes parent-child relationships



# Multilayer Network Embeddings

- Given: Layers $\{G_i\}$ , hierarchy $\mathcal{M}$
  - Layers $\{G_i\}_{i=1..T}$ are in leaves of $\mathcal{M}$
- Goal: Learn functions: $f_i: V_i \rightarrow \mathbb{R}^d$




Nodes have different embeddings in different layers, but we want these embeddings to be related!

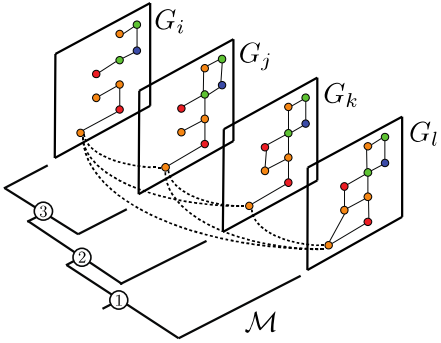
# Multilayer Network Embeddings

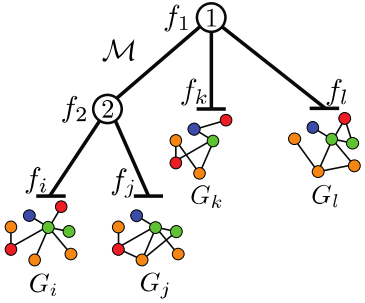
- Approach has two components:
  - Per-layer objectives: Standard node embedding objective (e.g., node2vec).
  - Hierarchical dependency objectives: Nodes in nearby layers in hierarchy are encouraged to share similar features

# Interdependent Layers

- How do we incorporate the hierarchy $\mathcal{M}$
- Per-layer node2vec objectives are learned independently of each other

How to model dependencies between layers when learning node features?



# Interdependent Layers

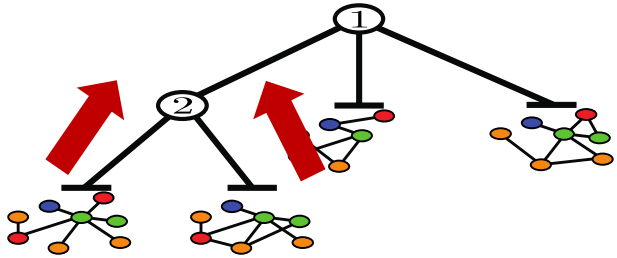
- Let $T \subset M$ be the set of all leaves in the hierarchy.
- Let Ti be the set of all leaves in the sub-hierarchy rooted at i. We assume that each layer Gi is attached to one leaf in the hierarchy. As a result, the hierarchy M has exactly K leaves.
- Let Ci denote the set of all children of element i in the hierarchy.

# Interdependent Layers

$$c_i(u) = \frac{1}{2}\|f_i(u) - f_{\pi(i)}(u)\|_2^2,$$

$$C_i = \sum_{u \in L_i} c_i(u)$$

where $L_i = \cup_{j \in T_i} V_j$




$L_i$ has all layers appearing in sub-hierarchy rooted at $i$

# *OhmNet*: Final Model

Learning node features in multi-layer networks

Solve maximum likelihood problem:

$$\max_{f_1, f_2, \ldots, f_{|M|}} \boxed{\sum_{i \in \mathcal{T}} \Omega_i} - \lambda \boxed{\sum_{j \in \mathcal{M}} C_j},$$

Per-layer network objectives

Hierarchical dependency objectives

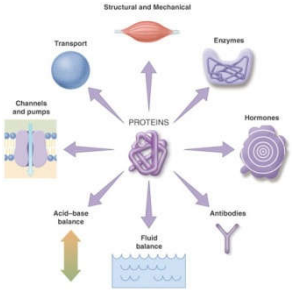
**Algorithm 1.** The *ObmNet* algorithm

**Input:** Multi-layer network, $(G_1, G_2, \ldots, G_K)$ with $G_i = (V_i, E_i)$, Hierarchy, $\mathcal{M}$, Feature representation size, $d$, Network neighborhood strategy, $S$, Regularization strength, $\lambda$

```
for i ∈ T do
    for u ∈ V_i do
        N_i(u) = Node2vecWalk(G_i, u, S) (Grover and Leskovec, 2016)
    end for
end for
while f_1, f_2, ..., f_|M| not converged do
    for i ∈ M do
        if i ∈ T then
            for u ∈ V_i do
                f_i(u) = SGD1(N_i(u), d, λ) by Equation (5)
            end for
        else
            for u ∈ ∪_{i∈T_i} V_i do
                f_i(u) = 1/(|C_i|+1) ( f_{π(i)}(u) + Σ_{c∈C_i} f_c(u) )
            end for
        end if
    end for
end while
return f_1, f_2, ..., f_|M|
```

# Experiments: Biological Nets

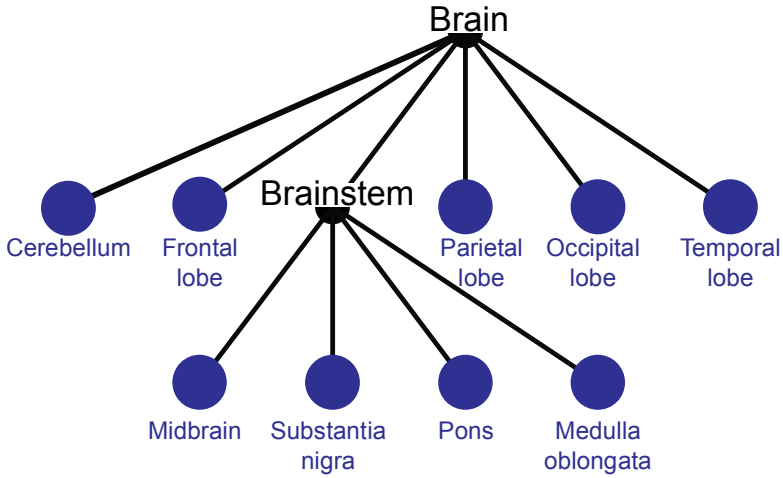
- Proteins are worker molecules
- Understanding protein function has great biomedical and pharmaceutical implications

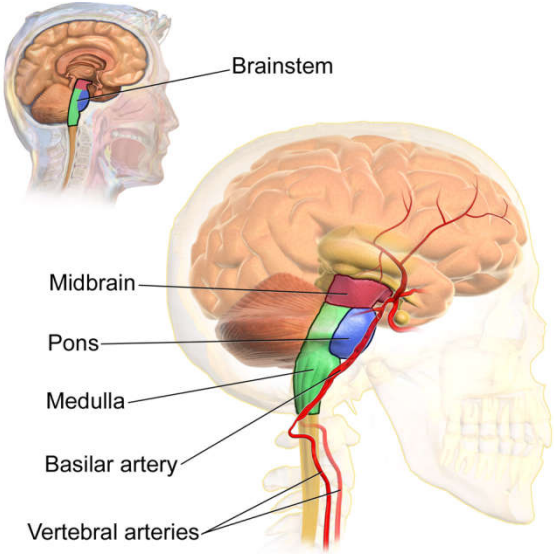



107 genome-wide tissue-specific protein interaction networks

- 584 tissue-specific cellular functions
- Examples (tissue, cellular function):
  - (renal cortex, cortex development)
  - (artery, pulmonary artery morphogenesis)

# Brain Tissues




9 brain tissue PPI networks in two-level hierarchy

# Meaningful Node Embeddings