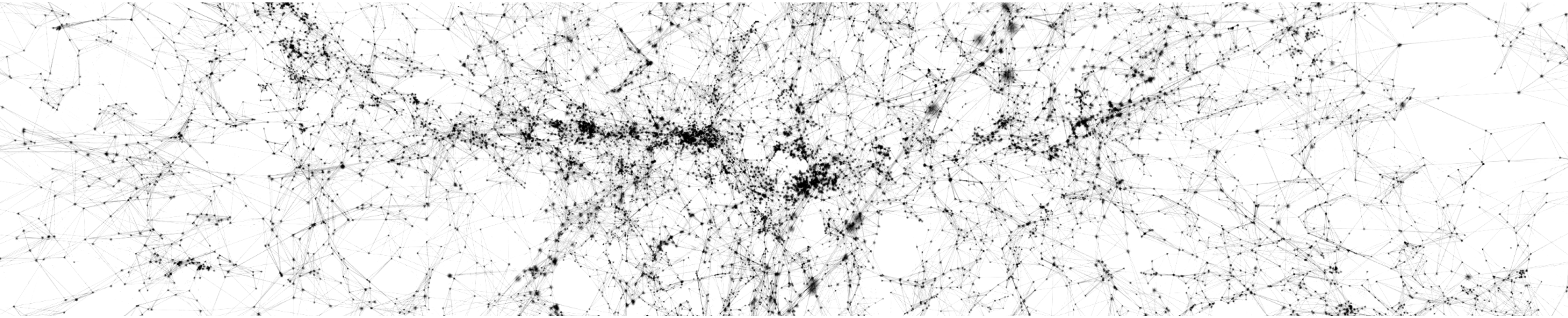
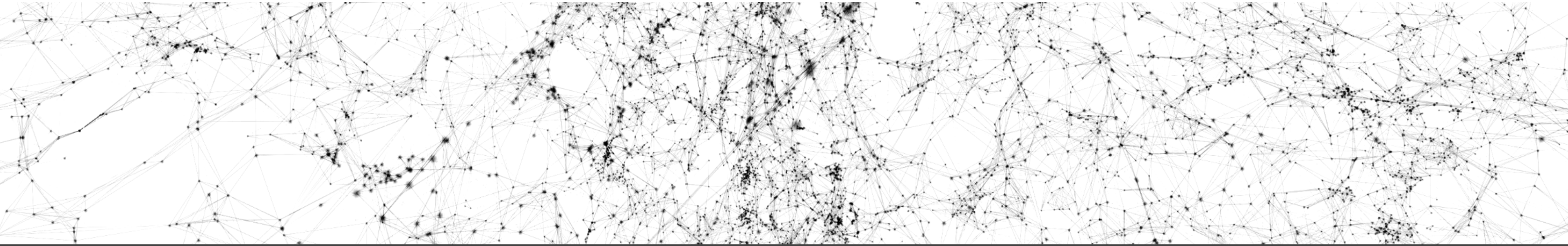


Node Embedding

Analysis of complex interconnected data

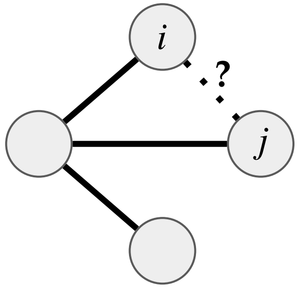


Common prediction tasks

- Link Prediction

  Classic example:

  $$z(i,j) = |\mathcal{N}(i) \cap \mathcal{N}(j)|$$



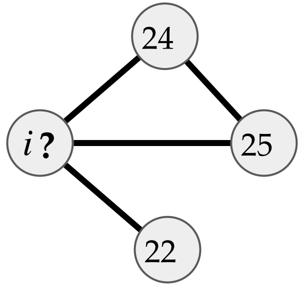
$$z(i,j) = f(h_i, h_j)$$

e.g. how likely it is for them to become friend?

- Node Classification

  What can be a simple predictor?

  $$z(i) = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} z(j)$$



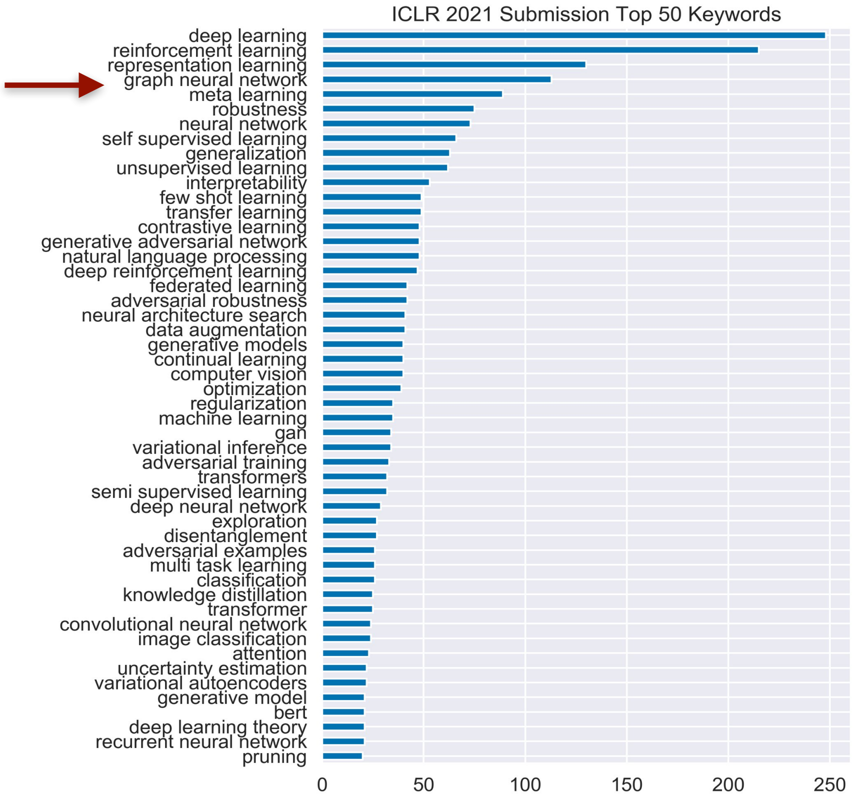
$$z(i) = f(h_i)$$

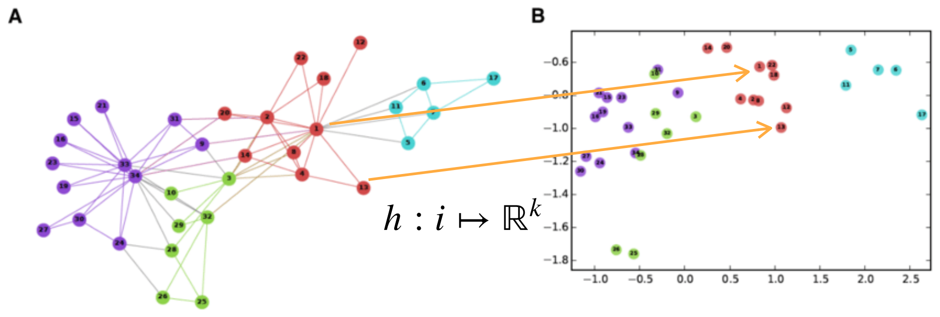
e.g. what is the age of a user based on his friends ages?

Instead of an hand-crafted measure, we can learn this

Graph Representation Learning

One of the hottest research topic in the ML community





Node embedding methods derive a vector representation per each node in the graph so that structurally similar nodes have closer vectors

From A Tutorial on Network Embeddings, 2018

Basic Vector Representation for Nodes

- Row in the Adjacency matrix:

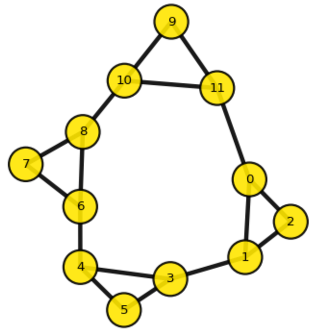
$$h_{10} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1]^\top$$

What is the common neighbour predictor based on h?

$$z(i,j) = h_i^\top h_j$$

$A \in \{0,1\}^{N \times N}$

	0	1	2	3	4	5	6	7	8	9	10	11	
0	0	1	1	0	0	0	0	0	0	0	0	1	3
1	1	0	1	1	0	0	0	0	0	0	0	0	3
2	1	1	0	0	0	0	0	0	0	0	0	0	2
3	0	1	0	0	1	1	0	0	0	0	0	0	3
4	0	0	0	1	0	1	1	0	0	0	0	0	3
5	0	0	0	1	1	0	0	0	0	0	0	0	2
6	0	0	0	0	1	0	0	1	1	0	0	0	3
7	0	0	0	0	0	0	1	0	1	0	0	0	2
8	0	0	0	0	0	0	1	1	0	0	1	0	3
9	0	0	0	0	0	0	0	0	0	0	1	1	2
10	0	0	0	0	0	0	0	0	1	1	0	1	3
11	1	0	0	0	0	0	0	0	0	1	1	0	3
	3	3	2	3	3	2	3	2	3	2	3	3	



- Row in the Laplacian matrix

- k-smallest nontrivial eigenvectors of Graph Laplacian a.k.a. Laplacian eigenmaps (LE)
  - K-means on this gives the Spectral Clustering

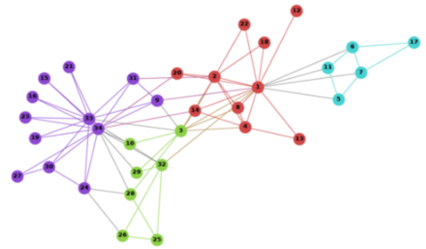
- Learn $h : i \mapsto \mathbb{R}^k$ so that
  - distance in the embedded space $\Rightarrow$ link prediction
  - decision boundaries in the embedded space $\Rightarrow$ node classification

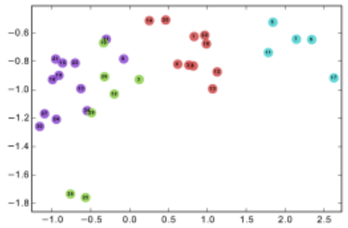
Graph Representation Learning

Map each node to a vector: $h : i \in G \mapsto \mathbb{R}^k$

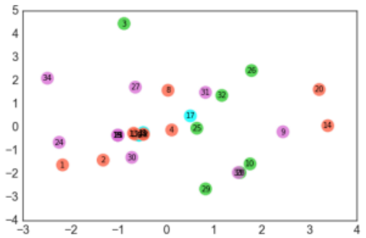
Embed the graph in vector space: $G \rightarrow H^{n \times k}$ {each row gives the embedding of one node}

- distance in the embedded space $\Rightarrow$ link prediction: $h_i^\top h_j$
- decision boundaries in the embedded space $\Rightarrow$ node classification

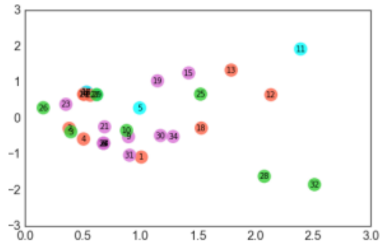




(a) Output: DeepWalk



(e) Output: LE



(f) Output: SVD

This can be trained unsupervised, e.g. based on cross-entropy loss puts connected nodes close-by, to preserves the edge structure:

$$\sum_{(i,j)\in E} \underbrace{\log \sigma(h_i^\top h_j)}_{\text{Links}} + \sum_{(i,j)\notin E} \underbrace{\log(1-\sigma(h_i^\top h_j))}_{\text{non-Links}}$$

See A Tutorial on Network Embeddings, 2018

logistic/squashing/activation function: $\sigma(x) = \frac{1}{1+e^{-x}} : \mathbb{R} \rightarrow (0,1)$
Gives a single probability

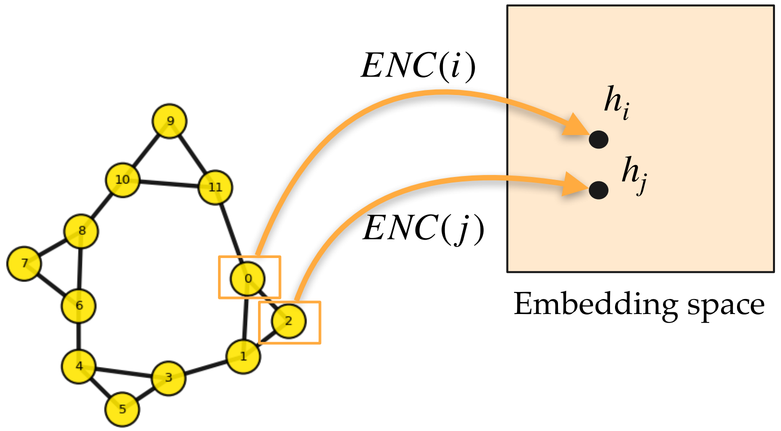
An Encoder-Decoder Perspective

Goal: Similarity in the embedding space (e.g. $h_i^\top h_j$) approximates the similarity in the graph

We need to define:

- **Encoder** gives low dimensional embedding that summarizes the graph position and structure in local neighbourhood
$$ENC(i) = h_i : i \to \mathbb{R}^k$$
- **Decoder** reconstructs this neighbourhood given the embedding of the node, or the pairwise similarities
$$S_G(i,j) \approx DEC(h_i, h_j) : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}^+$$

What can be a simple similarity measure here? A

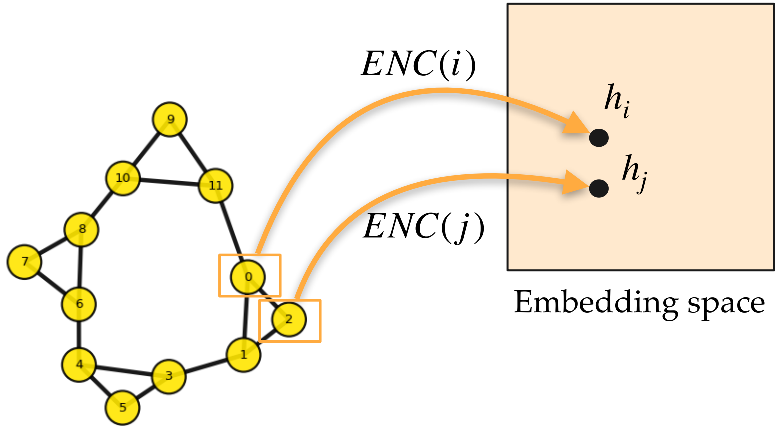


$$\mathscr{L} = \sum_{i,j} l(DEC(h_i, h_j), S_{ij})$$

Minimize using SGD, we directly optimize the embedding of each node

https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_3-Node_Embeddings.pdf

An example embedding



$DEC(h_i, h_j) = \|h_i - h_j\|_2^2$: L2-distance

$l(DEC(h_i, h_j), S_{ij}) = DEC(h_i, h_j) \,.\, S_{ij}$

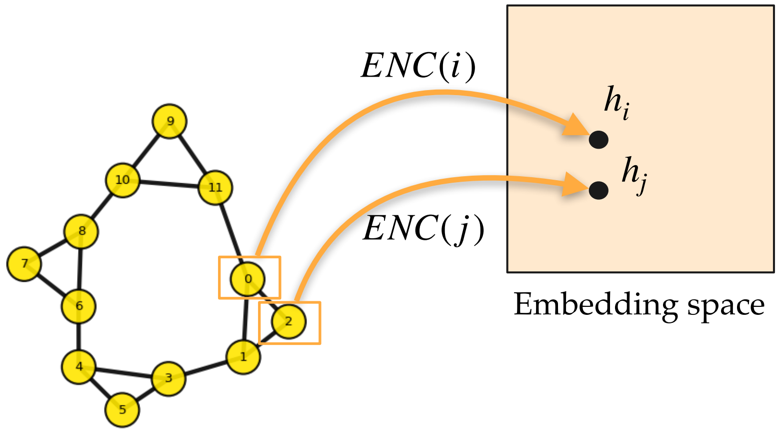
Intuition: similar nodes with far away embeddings have higher loss

$S_G(i, j)$ => if S is according to Laplacian, this gives identical to the solution for spectral clustering, i.e. k smallest eigenvectors of the Laplacian, that is the **Laplacian eigenmaps (LE) technique**

Laplacian Eigenmap

https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_3-Node_Embeddings.pdf

Inner-product methods



$DEC(h_i, h_j) = h_i^\top h_j$: dot product

$l(DEC(h_i, h_j), S_{ij}) = (DEC(h_i, h_j) - S_{ij})^2 \Rightarrow \mathscr{L} = \|HH^\top - S\|_2^2$

Intuition: distance in embedding space be the same as the distance in graph

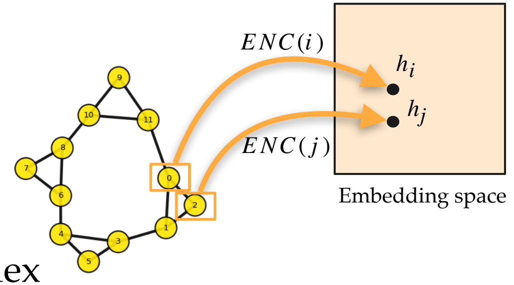
$S_G(i, j)$ => If $S = A$, the method is the Graph Factorization (GF) approach1 [Ahmed et al., 2013]

If S is set based on powers of A, $A, A^2 \ldots A^k$, the method is called GraRep [Cao et al., 2015]

If S is any classic neighbourhood overlap measure, the method is called HOPE [Ou et al., 2016]

These methods use a deterministic measure of similarity, which is limited and can be expensive to compute, it is it is better to use stochastic measure of neighbourhood overlap

Random walk embeddings



$$DEC(h_i, h_j) = \frac{e^{h_i^\top h_j}}{\sum_k e^{h_i^\top h_k}}$$: Softmax function: $\mathbb{R}^k \to \Delta_k$, probability simplex

$$p_k \in \Delta_k \Rightarrow \sum_k p_k = 1$$

$$l(DEC(h_i, h_j), S_{ij}) = -S_{ij} \log(DEC(h_i, h_j))$$: Cross-entropy loss, negative log likelihood that when minimized maximizes the likelihood of the graph

Intuition: nodes have similar embeddings are more likely to co-occur on short random walks over the graph

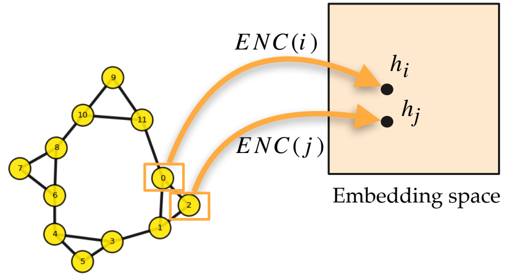
$S_G(i, j) = p_{\mathcal{G},\mathcal{T}}(j \mid i)$: probability of visiting j on a length $\mathcal{T}$ random walk from i
=> stochastic and asymmetric: $S_{ij} \neq S_{ji}$

$$l = -\log\left(\frac{e^{h_i^\top h_j}}{\sum_k e^{h_i^\top h_k}}\right) p_{\mathcal{G},\mathcal{T}}(j \mid i) \Rightarrow \mathcal{L} = -\sum_i \sum_{j \in \mathcal{N}_R(i)} \log\left(\frac{e^{h_i^\top h_j}}{\sum_k e^{h_i^\top h_k}}\right)$$

sum over nodes j seen on random walks starting from i

This is expensive to compute

Random walk embeddings



$$DEC(h_i, h_j) = \frac{e^{h_i^\top h_j}}{\sum_k e^{h_i^\top h_k}}$$: Softmax function

$$l(DEC(h_i, h_j), S_{ij}) = -S_{ij}\log(DEC(h_i, h_j))$$: Cross-entropy loss, negative log likelihood that when minimized maximizes the likelihood of the graph

$$S_G(i,j) = p_{\mathcal{G},\mathcal{T}}(j \mid i)$$: probability of visiting *j* on a length $\mathcal{T}$ random walk from *i*

This is expensive to compute as is:

If approximated with a hierarchical softmax the method is called **DeepWalk** [Perozzi et al., 2014]

If *loss* is approximated with a negative sampling, the method is called **Node2Vec** [Grover and Leskovec, 2016]

We can reformulate the loss as:

$$l \approx -\log(\sigma(h_i^\top h_j)) - \sum_k \log(\sigma(h_i^\top h_k))$$

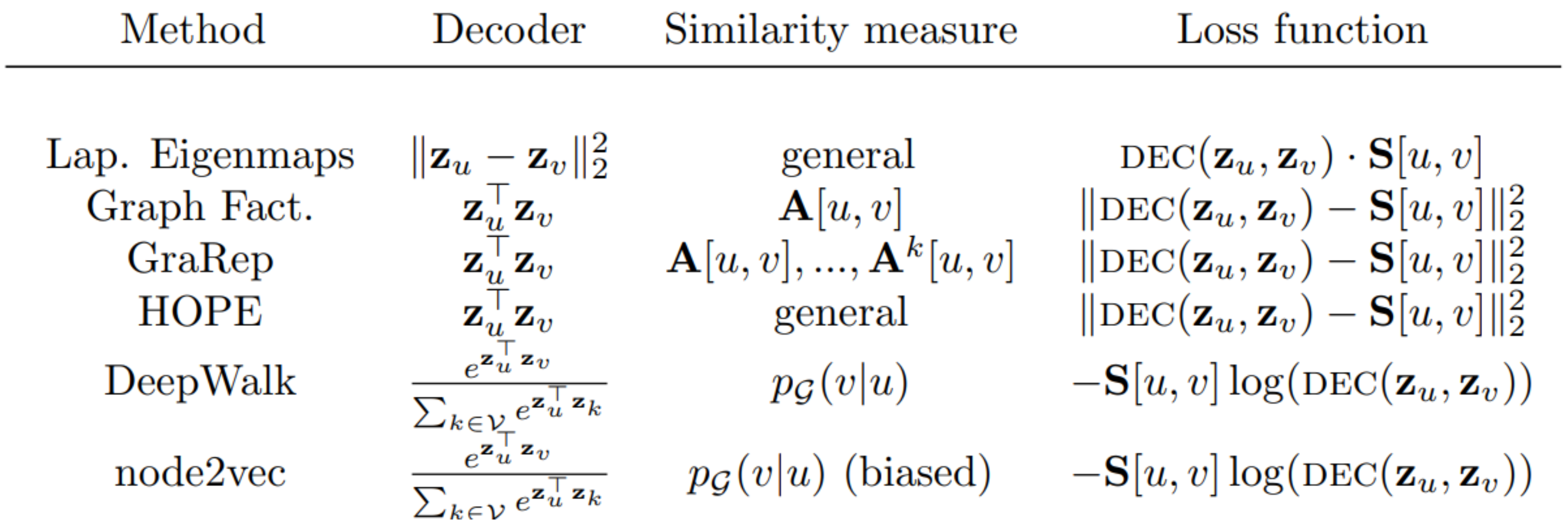
Explanation for the derivation: https://arxiv.org/pdf/1402.3722.pdf

logistic function: $\sigma(x) = \frac{1}{1+e^{-x}} : \mathbb{R} \rightarrow (0,1)$

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

More negative samples => more robust
In practice between 5 to 20

A summary of shallow embedding algorithms

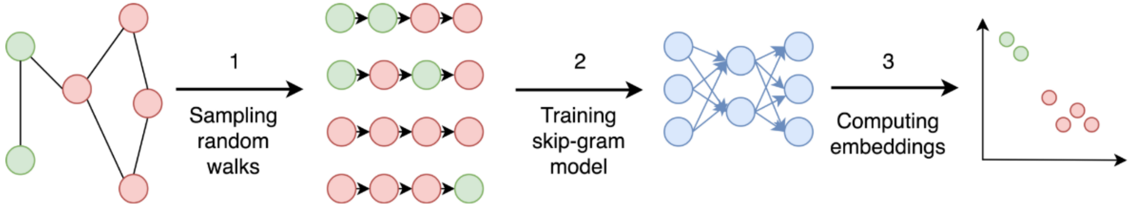


	$DEC(h_i, h_j)$	$S_G(i, j)$	$l(DEC(h_i, h_j), S_{ij})$
Method	Decoder	Similarity measure	Loss function
Lap. Eigenmaps	$\|\mathbf{z}_u - \mathbf{z}_v\|_2^2$	general	$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]$
Graph Fact.	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
GraRep	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v], ..., \mathbf{A}^k[u, v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
HOPE	$\mathbf{z}_u^\top \mathbf{z}_v$	general	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
DeepWalk	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v\|u)$	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$
node2vec	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v\|u)$ (biased)	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$

https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_3-Node_Embeddings.pdf

Deepwalk

```
Algorithm 1 DeepWalk($G$, $w$, $d$, $\gamma$, $t$)
Input: network $G(V, E)$
    window size $w$
    embedding size $d$
    walks per vertex $\gamma$
    walk length $t$
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V|\times d}$
1: Initialization: Sample $\Phi$ from $\mathcal{U}^{|V|\times d}$
2: Build a binary Tree $T$ from $V$
3: for $i = 0$ to $\gamma$ do
4:   $\mathcal{O} = \text{Shuffle}(V)$
5:   for each $v_i \in \mathcal{O}$ do
6:     $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
7:     SkipGram($\Phi$, $\mathcal{W}_{v_i}$, $w$)
8:   end for
9: end for
```



Phases of DeepWalk approach

```
Algorithm 2 SkipGram($\Phi$, $\mathcal{W}_{v_i}$, $w$)
1: for each $v_j \in \mathcal{W}_{v_i}$ do
2:   for each $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ do
3:     $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
4:     $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
5:   end for
6: end for
```

SkipGram is a language model that maximizes the co- occurrence probability among the words that appear within a window, w, in a sentence

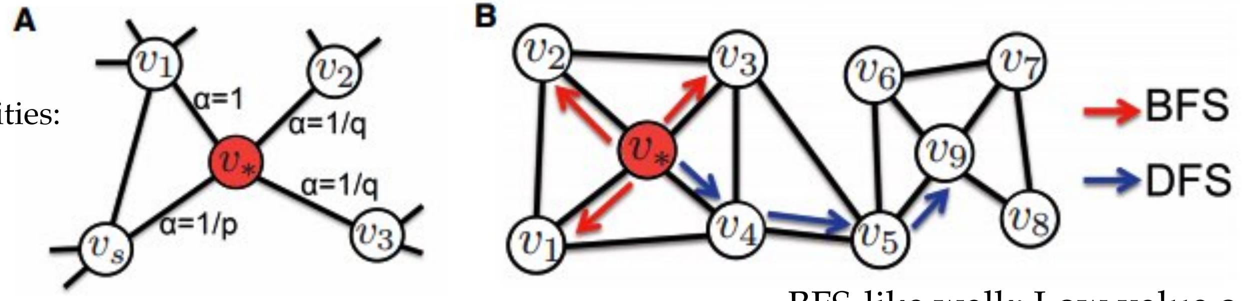
- 32 to 64 random walks from each node of a length of about 40 steps
- Random walks as sentences, maximize probability of predicting neighbour nodes

https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007
https://arxiv.org/pdf/1403.6652.pdf

Node2vec

Similar to Deepwalk but interpolates between random walks that discover larger neighborhood (Q), and those that stay local (P)

unnormalized probabilities:
{1, 1/p, 1/q, 1/q}



BFS-like walk: Low value of *p*

DFS-like walk: Low value of *q*

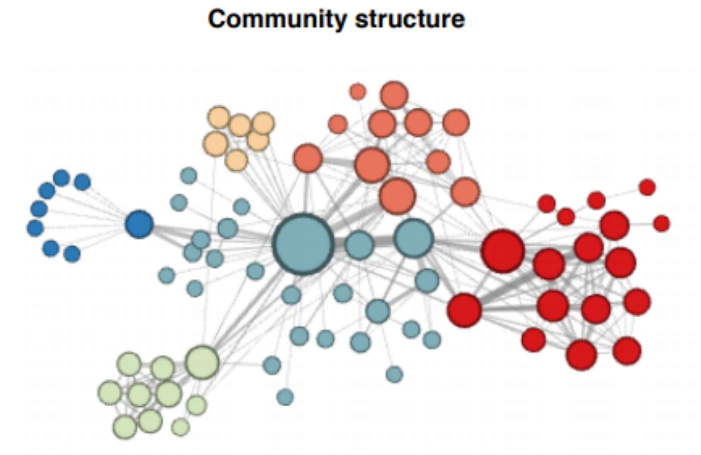
Two parameters:

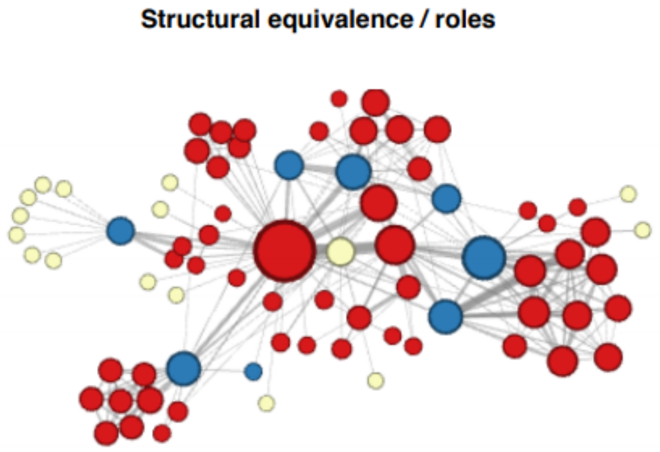
- Return parameter ***p***: Return back to the previous node
- In-out parameter ***q***: Moving outwards (DFS) vs. inwards (BFS): Intuitively, *q* is the "ratio" of BFS vs. DFS

Node2Vec Different ways to embed

Embedding so that nodes

- in the same cluster are placed close together (DFS)
- with similar roles are placed close together (BFS)





https://arxiv.org/pdf/1607.00653.pdf

Limitations of Shallow Embeddings

No parameter sharing $\Rightarrow$ less scalable

- encoder directly optimizes a unique embedding vector for each node
- the number of parameters grows with size of graph

Ignores features or attributes & labels

Inherently transductive $\Rightarrow$ can not process unseen nodes

Read more:
A Tutorial on Network Embeddings, 2018 &
Representation Learning on Graphs, 2017 &
GLR book's chapter on node embedding, 2020

From Shallow Embeddings to Graph Neural Nets

- No parameter sharing ⇒ less scalable
- Ignores features or attributes
- Inherently transductive ⇒ can not process unseen nodes

optimized a unique embedding vector for each node ⇒ more complex encoder models, graph neural networks which work based on feature propagation

$$f(X, A)$$

- Number of parameters doesn't grow with graph size but feature dimension
- Naturally incorporates node features
- Inherently inductive ⇒ infer embedding for unseen nodes

Watch https://www.cs.mcgill.ca/~wlh/grl_book/files/hamilton_grl_talk.mp4

Resources: Libraries and Datasets

OGB

Ogb.stanford.edu

PyG

PyTorch geometric

github.com/rusty1s/pytorch_geometric

Spektral

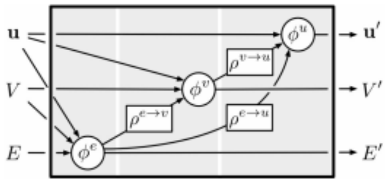
graphneural.network

TUDataset

Graphlearning.io

DGL

dgl.ai



github.com/deepmind/graph_nets

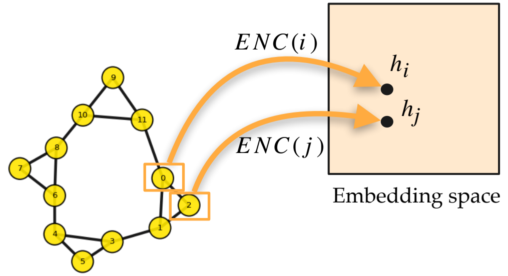
github.com/deepmind/jraph

https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html

github.com/graphdeeplearning/benchmarking-gnns

based on https://petar-v.com/talks/GNN-Wednesday.pdf

Graph Likelihood



We want to maximize the graph likelihood $= Max \prod_{(i,j)\in E} \underbrace{p(i,j)^{s_{ij}}}_{\text{Links be likely}} \prod_{(i,j)\notin E} \underbrace{(1-p(i,j))}_{\text{non-Links be not likely}}$

This is similar to minimizing the negative log likelihood

$$= Min - \sum_{(i,j)\in E} \underbrace{log(p(i,j))s_{ij}}_{\text{Links}} - \sum_{(i,j)\notin E} \underbrace{\log(1-p(i,j))}_{\text{non-Links}}$$

With S = A, and $p(i,j) = \sigma(h_i^\top h_j)$ we get

$$h^* = argmin_h - \sum_{(i,j)\in E} \underbrace{\log \sigma(h_i^\top h_j)}_{\text{Links}} - \sum_{(i,j)\notin E} \underbrace{\log(1-\sigma(h_i^\top h_j))}_{\text{non-Links}}$$

$$h^* = argmin_h - \sum_{(i,j)\in E} \underbrace{\log \sigma(h_i^\top h_j)}_{\text{Links}} + \sum_{(i,j)\notin E} \underbrace{\log(\sigma(h_i^\top h_j))}_{\text{non-Links}}$$

$\sigma(x) = \frac{1}{1+e^{-x}} : \mathbb{R} \to (0,1)$

logistic/squashing/activation function: gives a single probability