<Why Embedding?>

Goal: Efficient task-independent (与下游任务无关) feature learning for ML with graph!

f: (node) $u \longrightarrow \mathbb{R}^d$ 将节点映射为 d 维向量
↳ unsupervised.

{ 低维：向量维度 ≪ 节点数
连续：每个元素都是实数.
稠密：每个元素都不为 0. (与 sparse 相对 ⇒ distributed)

向量相似度 → 节点相似度.

① similarity of embeddings between nodes indicates their similarity in the network.

② Encode network information 嵌入向量包含网络连接信息.

③ downstream predictions: node classification / link prediction / graph classification / clustering ...

<Set Up>

G { V: vertex set 节点集
A: adjacency matrix (assume binary) 无权图.
⇒ No node features or extra information is used.
(仅利用节点连接信息，没有节点的属性信息)

$V = \{1, 2, 3, 4\}$

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

<Encoder & Decoder>
编码器与解码器.

<Embedding Nodes>

encode nodes so that similarity in the embedding space (dot production) approximates similarity in the graph

向量点乘数值 (余弦相似度) ⟶ 节点相似度.

(similarity) $z_v^T z_u \approx$ similarity (u,v) (in original network) ← 人为定义.

maximize: $z_v^T z_u$ for (u,v)

※: 迭代优化每个节点的D维向量，
使得图中相似节点向量数量积大，
不相似节点向量数量积小.

① Encoder maps from nodes to embeddings. $\text{ENC}(v) = z_v$. ← low-dim vector.

② Define a node similarity function — a measure of similarity in the original network.

③ Decoder DEC maps from embeddings to the similarity score.

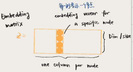
④ Optimize the parameters of the encoder so that: similarity $(u,v) \approx z_v^T z_u$.

"Shallow" Encoding (DeepWalk, node2vec)

└ simplest encoding approach: Encoder is just an embedding-lookup (查表).

$\text{ENC}(v) = z_v = Z \cdot v$ (提取Z矩阵的某一列 —— 一个节点的嵌入向量)

$Z \in \mathbb{R}^{d \times |V|}$ (d行, |V| 节点个数)
$v \in \mathbb{I}^{|V|}$ → one-hot: 只有一个元素为1, 其余为0. indicator vector.



· Each node is assigned a unique embedding vector → directly optimize (直接优化Z矩阵.) the embedding of each node.

✱ This is unsupervised / self-supervised. task-independent.
(无监督 / 自监督：无节点类别标签，无节点属性特征，直接优化嵌入向量，与下游任务无关.)

<Random Walk Approaches>
随机游走.

① Vector z_u: The embedding of node u → aim to find

② Probability $P(v|z_u)$: of visiting node v on random walks starting from node u.
↳ 从 u 节点出发的随机游走序列经过 v 节点的概率.

✱ Softmax: turns vector of K real values (model predictions) into K probabilities that sum to 1.

$$\sigma(z)[i] = \frac{e^{z[i]}}{\sum_{j=1}^{K} e^{z[j]}} \qquad S = \frac{1}{1+e^{-x}}.$$

✱ Sigmoid: S-shaped function that turns real values into the range of (0,1)

non-linear functions used to produce predicted probabilities.

▲ 归一化函数：
将一个含任意实数的K维向量z"压缩"到另一个K维实向量 σ(z) 中，使得每一个元素的范围都在 (0,1) 之间，且所有元素和为1. (Σ=1)

Ch.

Date /

节点"相似"的定义：
共同出现在同一个随机游走序列中.

① 采样得到若干随机游走序列.
计算其条件概率

② 迭代优化每个节点的 D 维向量.
使得序列中共现节点向量数量积大.
不 …… 小.

$z_u^T \cdot z_v \approx$ probability that u & v co-occur on a random walk over the graph

① Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R. $P_R(v|u)$

② Optimize embeddings to encode these random walk statistics. $\theta \propto P_R(v|u)$

Pros: expressivity & efficiency.

dot product $= \cos(\theta)$ represents similarity

<Unsupervised Feature Learning>

Given $G=(V,E)$, learn a mapping $f: u \to \mathbb{R}^d$: $f(u)=z_u$.

- log-likelihood objective: 似然目标函数.

$$\max_f \sum_{u \in V} \log P(N_R(u) | z_u)$$

→ (希望所有事件都被预测正确的概率最大化)
↳ 相乘.

↳ neighborhood of node u by strategy R.

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 $N_R(u)$, the multiset* of nodes visited on random walks from u.

③ Optimize embeddings according to:

Given node u, predict its neighbors $N_R(u)$.

$$\max_f \sum_{u \in V} \log P(N_R(u) | z_u)$$ → Maximum likelihood objective

sum over all nodes (遍历所有节点) ↑

$$L = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(z_u^T \cdot z_v)}{\sum_{n \in V} \exp(z_u^T \cdot z_n)}\right)$$

遍历结点对. $O(|V|^2)$ complexity.

↓ sum over nodes v seen on random walks starting from u
(遍历从 u 节点出发的随机游走序列所有邻域节点)

↳ predicted probability of u and v co-occuring on random walk
(节点 u 和 v 在该随机游走序列中共现)

▲ Optimizing random walk embeddings = Finding embeddings z_u that minimize L.

<Negative Sampling>
负采样.

* 理论上，同一个随机游走序列中的节点，不应该被采样为"负样本".

$$\log\left(\frac{\exp(z_u^T \cdot z_v)}{\sum_{n \in V} \exp(z_u^T \cdot z_n)}\right) \approx \log(\sigma(z_u^T \cdot z_v)) - \sum_{i=1}^{k} \log(\sigma(z_u^T \cdot z_{n_i})),\ n_i \sim P_V.$$

sigmoid function; (k个负样本：随机); random distribution → 非均匀分布. 随机采样.

- sample k negative nodes each with prob. proportional to its degree.

k-value { Higher k gives more robust estimates.
Higher k corresponds to higher bias on negative events → in practice $k = 5 \sim 20$.

<Stochastic Gradient Descent>
随机梯度下降.

随机初始化

① Initialize z_u at some randomized value for all nodes u.

② Iterate until convergence:

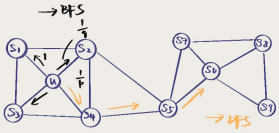
求所有节点 u 总梯度: For all u, compute the derivative $\frac{\partial L}{\partial z_u}$

迭代更新: For all u, make a step in reverse direction of derivative: $z_u \leftarrow z_u - \eta \frac{\partial L}{\partial z_u}$ (learning rate η)

Ch.

Date /

Node2 Vec



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

$N_{BFS}(u) = \{s_1, s_2, s_3\}$ → Local (微)

$N_{DFS}(u) = \{s_4, s_5, s_6\}$ → Global (宏)

Goal: Embed nodes with similar network neighborhoods close in the feature space.

↳ maximum likelihood optimization.

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.

↓ trade-off between local & global (BFS / DFS)

- return parameter p: return back to the previous node (p小 好走)
- In-Out parameter q: "ratio" of BFS vs. DFS (p大 q小)

- 1/p, 1/q, 1 are unnormalized probabilities.

<Algorithm>: (Individually parallelizable: 独立并行)

Linear-time

① Compute random walk probabilities

② Simulate r random walks of length l starting from each node u.

③ Optimize the node2 vec objective using Stochastic Gradient Descent.

<Matrix Factorization>

- Simplest node similarity: Nodes u, v are similar if they're connected by an edge.

↳ means: $z_v^T \cdot z_u = A_{u,v}$ which is the (u,v) entry of the graph adjacency matrix A.

$\Rightarrow Z^T Z = A$, Z: $(N \times d)$, Z^T: $(d \times N)$, A: $(N \times N)$ 特殊矩阵. ⇒ approximately (近似值计算估计).

Optimize Z: minimize the L_2-norm of $(A - Z^T Z)$

Objective: $\min_Z \|A - Z^T Z\|_2$ ↳ Frobenius norm 费罗贝尼乌斯.

✱ Conclusion: Inner Product decoder with node similarity defined by edge connectivity is 相当 equivalent to matrix factorization of A.

- Node2 Vec matrix:

volume of graph: 2 × 连接个数.

$vol(G) = \sum_i \sum_j A_{i,j}$

对角矩阵 节点连接数.

Diagonal matrix D

$D_{u,u} = deg(u)$

$$\log\left(vol(G)\left(\frac{1}{T}\sum_{r=1}^{T}(D^{-1}\cdot A)^r\right)D^{-1}\right) - \log b$$

上下文滑窗宽度 context window size

$T = |N_R(u)|$

power of normalized adjacency matrix

Number of negative samples 负采样 样本数.

<Limitations>

↓

Solution:

Deep Representation Learning & Graph Neural Network.

① Cannot obtain embeddings for nodes not in the training set. 无法立刻泛化到新加入的节点. 某种程度的过拟合.

② Cannot capture structural similarity.

③ Cannot utilize node, edge and graph features.

<Embedding Entire Graph>

嵌入整个图.

Goal: Want to embed a subgraph or an entire graph G.

① $z_G = \sum_{v \in G} z_v$ (直接对所有节点嵌入求和)

② "virtual node" (虚拟节点, represent the subgraph)

Ch.

Date /

< Anonymous walks >
匿名游走.

- states in anonymous walks correspond to the index of the first time we visited the node in a random walk. (每次见到不同节点. 就发一个新编号).

* length 3: $w_1 = 111$, $w_2 = 112$, $w_3 = 121$, $w_4 = 122$, $w_5 = 123$

↳ number of anonymous walks grows exponentially. (4个: 15 / 5个: 52 / … 10个: 115k)

Bag of Anonymous walks ←

- simulate anon. walks w_i of l steps and record their count. 采样出不同匿名随机游走序列的个数.

represent the graph as a probability distribution over these walks

- 匿名随机游走长度固定时.

$m = \lceil \frac{2}{\varepsilon^2} (\log(2^l - 2) - \log(\delta)) \rceil$

ε: error $> \varepsilon$; δ: prob $< \delta$.

欲使误差大于 ε 的概率小于 δ.

需采样 m 次.

<Learn Walk Embedding>
给每种匿名随机游走序列学独热嵌入编码.

→ Learn to predict walks that co-occur in Δ-size window

$$\max \frac{1}{T} \sum_{t=0}^{T} \log P(w_t \mid \{w_{t-\Delta}, \dots w_{t-1}, z_G\})$$

How to use Embedding ?

① Clustering / community detection: Cluster points z_i.

② Node classification: Predict label of node i based on z_i.

③ Link prediction: predict edge (i, j) based on (z_i, z_j)

- 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: $f(z_i, z_j) = g(\|z_i - z_j\|_2)$

④ Graph classification: Graph embedding z_G via aggregating node embeddings or anonymous random walks. Predict label based on graph embedding z_G.