

Learning from Networks

Nodes and Graph Embedding

Fabio Vandin

November 13th, 2024

Course Recap

Until now, we have seen how to compute interesting measures for a network or its nodes, e.g.:

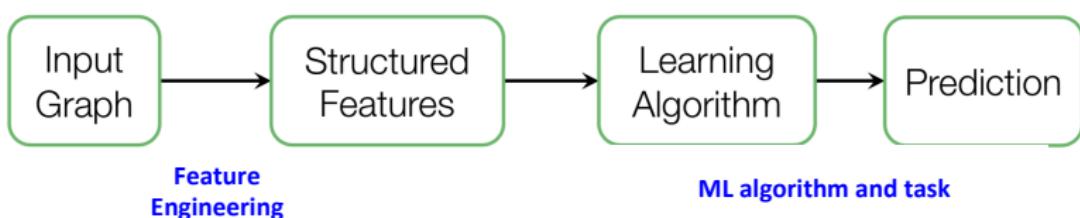
- centrality scores
- clustering coefficient
- graphlet/motifs score

Why?

- they provide useful information on the network (e.g., network motifs)
- they can be used as features in a vector representation of nodes/graphs to be used as input for a Machine Learning (ML) task

We are now going to focus on this second aspect.

The Traditional Framework for ML with Graphs



Given a graph:

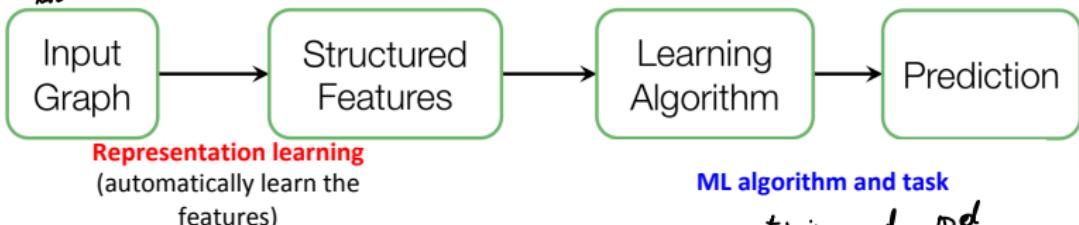
- extract node-level or graph-level features (*feature engineering*)
- learn a model that maps feature vectors to labels

↓
1 per node

Note: feature engineering is required for every task/application!

Graph Representation Learning Framework

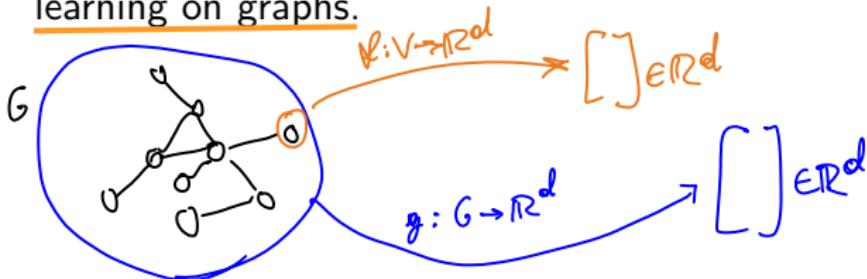
Representation learning: automatically learn the features to be used for the downstream prediction task



Node Embedding Task: map nodes into an embedding space

Graph Embedding Task: map graphs into an embedding space

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



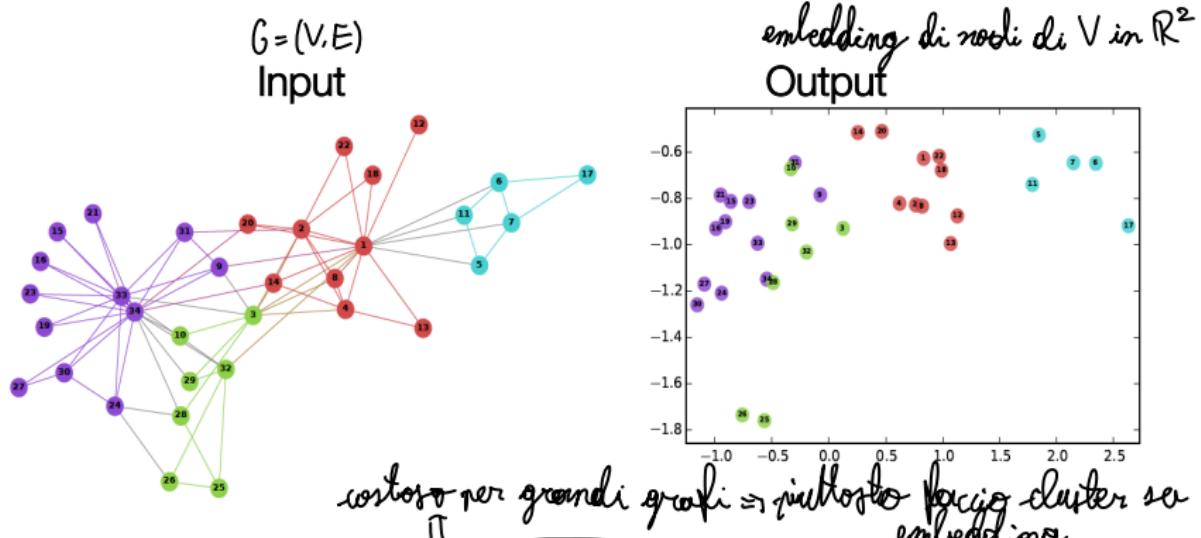
Motivation

Node Embedding Task: map nodes into an embedding space

Why?

- the similarity of embeddings indicates the similarity in the network
- encode the network information
- can be used for many downstream task predictions

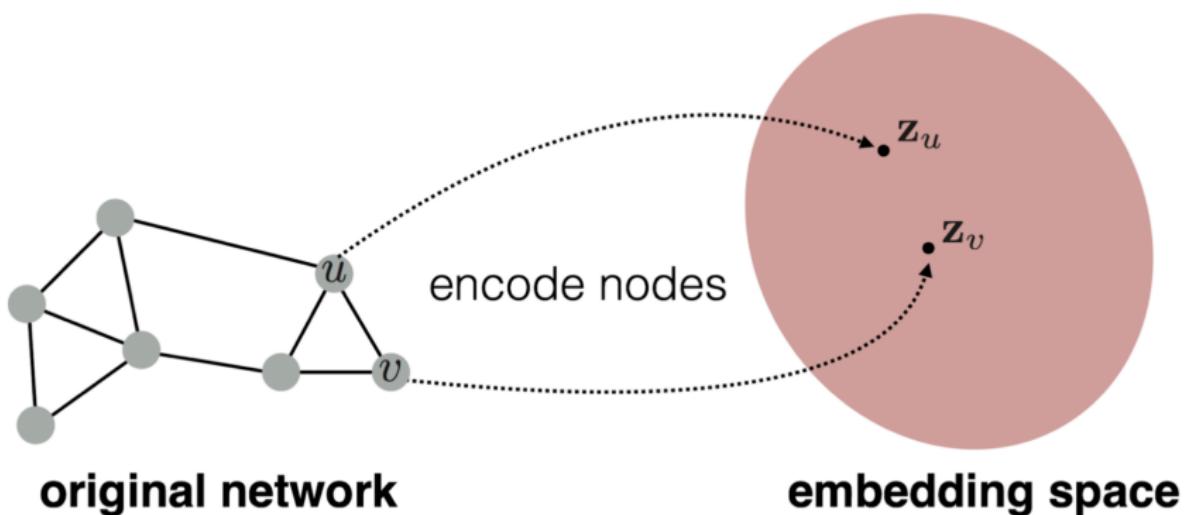
Example: Zachary's Karate Club



Colors: labels from clustering on the network (not available in input to the embedding method)

Node Embedding: General Framework

Given a graph $G = (V, E)$, we want to embed its nodes so that the similarity in the embedding space approximates the similarity in the graph.



Node Embedding: Encoder-Decoder Framework

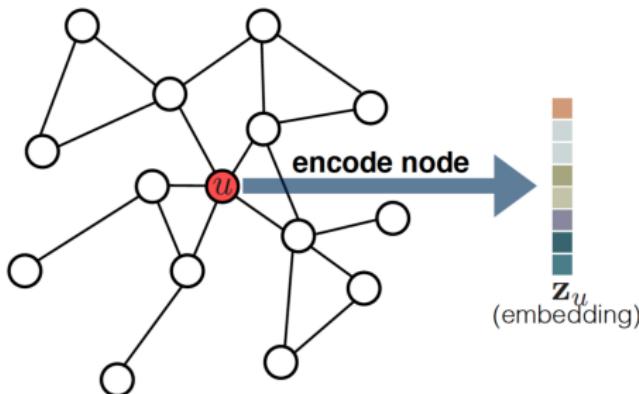
Overall Idea:

- **encoder:** maps nodes to their embeddings
- **similarity function:** defines the similarity of pairs of nodes in the (original) network
- **decoder:** maps embeddings to the (estimated) similarity score
- **optimize the parameters of the encoder so that the similarity of two nodes in the network is approximated by their embeddings**

Encoder-Decoder Components: Encoder

$$ENC : V \rightarrow \mathbb{R}^d$$

Given $v \in V$, ENC generates the embedding $ENC(v) = z_v$ of node v

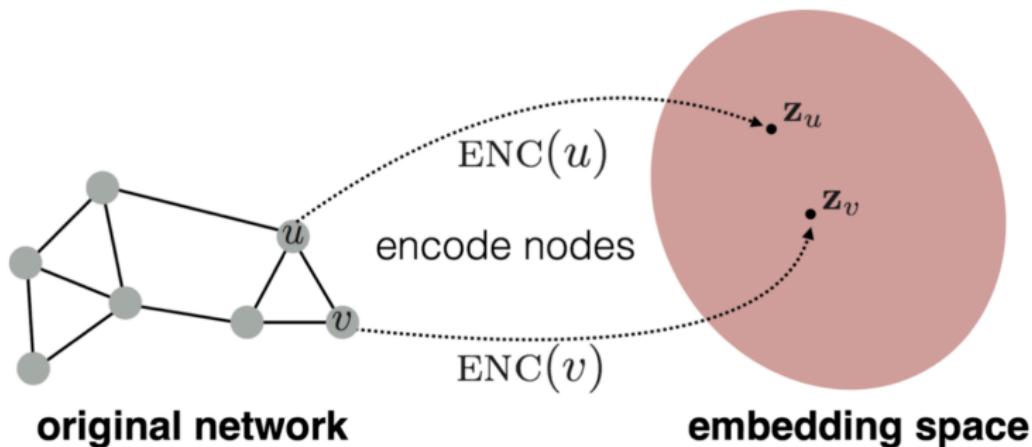


The encoder ENC depends on several parameters that are trained/optimized during the *learning* phase.

Encoder-Decoder Components: Similarity Function

We consider pairwise similarity functions: $S : V \times V \rightarrow \mathbb{R}$

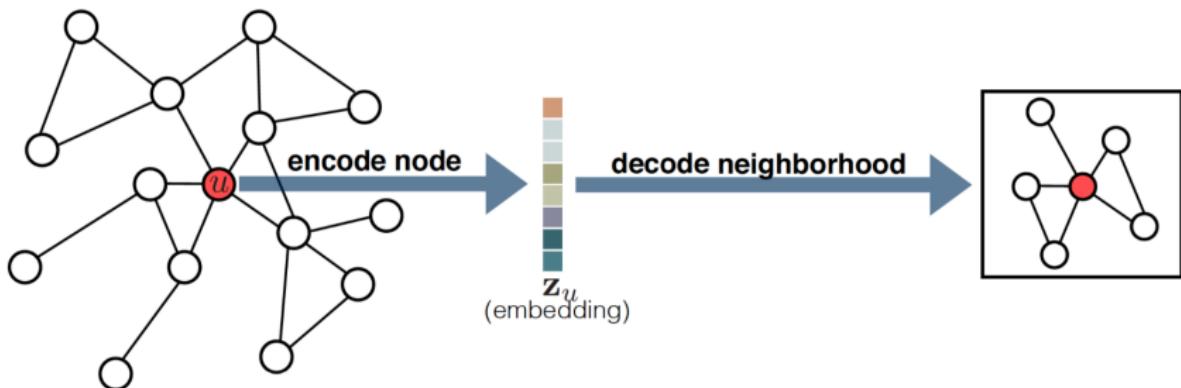
Given u and v in V , $S(u, v)$ measures the similarity between u and v in G



Usually $S(u, v)$ is represented by a matrix S with $S_{u,v} = S(u, v)$

Encoder-Decoder Components: Decoder

In general: a function that, given a set of node embeddings,
computes a (user specified) graph statistic.

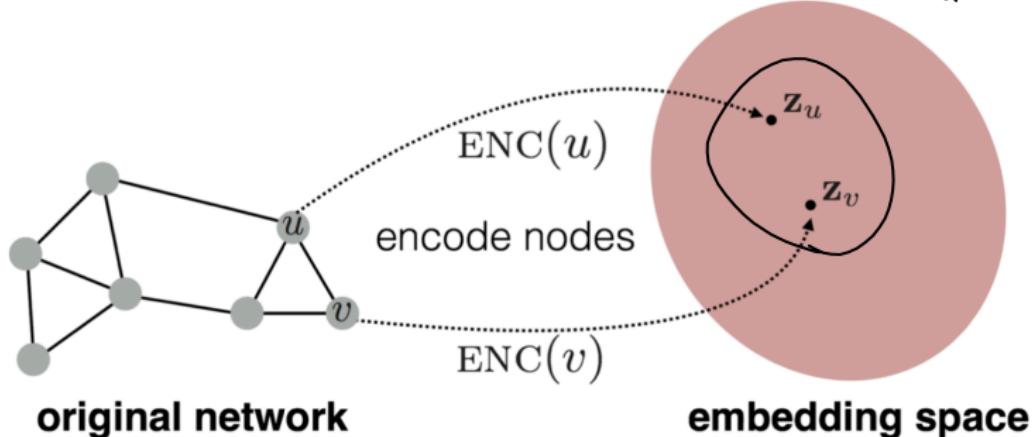


Encoder-Decoder Components: Decoder (continue)

We focus on commonly used pairwise decoders:

$$DEC : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$$

$$DEC(\vec{z}_u, \vec{z}_v) \approx \text{sim}(u, v)$$



Given embeddings of z_u and z_v of u and v in V , it reconstructs the similarity of nodes u and v in G .

Usually DEC has no trainable parameters.

Loss Function

Given nodes $u, v \in V$, the loss function is:

$$\ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v))$$

$\ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v))$ measures the discrepancy between the decoded similarity value $DEC(\mathbf{z}_u, \mathbf{z}_v)$ and the true similarity $S(u, v)$.

Overall Goal: optimize the parameters of ENC so that each pair $u, v \in V$:

$$DEC(\mathbf{z}_u, \mathbf{z}_v) = DEC(ENC(u), ENC(v)) \approx S(u, v)$$

\approx : captured by $\ell(DEC(ENC(u), ENC(v)), S(u, v))$

Encoder-Decoder Framework

In practice: set ENC parameters to minimize the *empirical loss*:

$$\mathcal{L} = \sum_{u,v \in V \times V} \ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v))$$

*risolve media come in
ML
↓
con notazione più
facile*

Sometimes: a set \mathcal{D} of training pairs of nodes is used

$$\mathcal{L} = \sum_{u,v \in \mathcal{D}} \ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v))$$

"Shallow" Encoding

We now consider the simplest type of encoder: ENC is a look-up function

$$ENC(v) = z_v = Z \times v$$

where

- $Z \in \mathbb{R}^{d \times |V|}$: matrix where each column is a node embedding
- $v \in \{0, 1\}^{|V|}$: indicator vector, all 0's except a single 1 indicating node v

“Shallow” Encoding (continue)

embedding matrix \vec{v} embedding \vec{z}_v di v

$\vec{z} = \begin{bmatrix} \vec{v} \\ \vdots \\ \vec{v} \end{bmatrix}$

dim. embedding \Rightarrow per \mathbb{R}^d , allora d

$$\text{ENC}(v) = \vec{z} \cdot \vec{v} = \left[\begin{array}{c|c} \vec{v} & \vec{z} \\ \hline \end{array} \right] \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \vec{v} \\ \vdots \\ \vec{v} \end{bmatrix}$$

dim. M

“Shallow” Encoding: Notes

We directly optimize the embedding of each node

Parameters? All entries of Z

Note: with “shallow” embeddings we are learning a representation (vector in \mathbb{R}^d) for each node $v \in V$, but we are not learning a function to obtain the representation of a node $v \in V$.

It is unsupervised/self-supervised way of learning node embeddings:

- we are not using node labels
- we are not using node features

These embeddings are task independent: they are not trained for a specific task but can be used for *any* task.

“Shallow” Encoding: Methods

Several methods to obtain “shallow” embeddings have been proposed.

Key difference: *definition of node similarity* $S(u, v)$

Question: when are two nodes u and v similar?



When they:

- are directly connected?
- have a similar neighborhood?
- ?

Adjacency-based Similarity Embeddings

Let A be adjacency matrix of G .

Similarity: $S(u, v) = A_{u,v} = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases}$

Encoder: shallow encoder $ENC(v) = \mathbf{Z} \times \mathbf{v} = \mathbf{z}_v$

Decoder: $DEC(\mathbf{z}_u, \mathbf{z}_v) = \mathbf{z}_u^T \mathbf{z}_v \Rightarrow$ praticamente sempre usato per shallow

Loss function: squared-loss

$$\ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v)) = (\mathbf{z}_u^T \mathbf{z}_v - A_{u,v})^2$$

Adjacency-based Similarity Embeddings (continue)

Empirical loss:

$$\mathcal{L} = \sum_{u,v \in V \times V} \ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v)) = \sum_{u,v \in V \times V} (\mathbf{z}_u^T \mathbf{z}_v - A_{u,v})^2$$

Therefore the embeddings \mathbf{z}_v for each $v \in V$ are given by the solution \mathbf{Z}^* to

$$\mathbf{Z}^* = \arg \min_{\mathbf{Z}} \sum_{u,v \in V \times V} (\mathbf{z}_u^T \mathbf{z}_v - A_{u,v})^2$$

Example



embedding in $\mathbb{R}^2 \Rightarrow \vec{z}_u = \begin{bmatrix} z_u^1 \\ z_u^2 \end{bmatrix}, \vec{z}_v, \vec{z}_w$

adjacency matrix $A = \begin{bmatrix} u & v & w \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ \Rightarrow goal: $\vec{z}_u^\top \cdot \vec{z}_v^\top \approx A_{u,v}, \vec{z}_v \cdot \vec{z}_w \approx 1, \vec{z}_v \cdot \vec{z}_u \approx 0$

misratio do squared-loss

$$\vec{z}_u = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \vec{z}_v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \vec{z}_w = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow \text{sol. ottimale}$$

$$G: \begin{array}{c} v \\ \diagdown \quad \diagup \\ u \quad w \end{array} \Rightarrow A = \begin{matrix} u & v & w \\ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \\ v \\ w \end{matrix}$$

embedding in $\mathbb{R}^2 \rightarrow \vec{z}_u = \vec{z}_v = \vec{z}_w = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

HW:

dato $G = (V, E)$ con $|E| = m$, \exists embedding in \mathbb{R}^m tale che $L = 0$?

Computing Adjacency-based Similarity Embeddings

We want to compute

$$\mathbf{Z}^* = \arg \min_{\mathbf{Z}} \sum_{u,v \in V \times V} (\mathbf{z}_u^T \mathbf{z}_v - A_{u,v})^2$$

How do we compute \mathbf{Z}^* ?

Two ways:

$$\arg \min_{\mathbf{Z}} \|\mathbf{Z}^T \mathbf{Z} - \mathbf{A}\|_F$$

FROBENIUS FORM:
 $\|\mathbf{x}\|_F = \sqrt{\sum_{i,j} x_{ij}^2}$

- matrix decompositions (e.g., SVD): do not scale
- general optimization methods: ...

Gradient Descent (GD)

General approach for *minimizing* a differentiable convex function
 $f(\mathbf{z})$

Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a differentiable function

Definition

The *gradient* $\nabla f(\mathbf{z})$ of f at $\mathbf{z} = (z_1, \dots, z_d)$ is

$$\nabla f(\mathbf{z}) = \left(\frac{\partial f(\mathbf{z})}{\partial z_1}, \dots, \frac{\partial f(\mathbf{z})}{\partial z_d} \right)$$

Intuition: the gradient points in the direction of the greatest rate of increase of f around \mathbf{z}

Let $\eta \in \mathbb{R}, \eta > 0$ be a parameter.

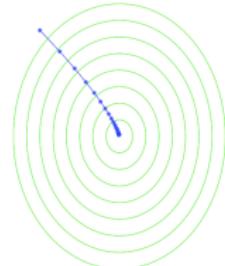
GD algorithm:

$z \leftarrow$ random vector in \mathbb{R}^d ;

while *not converged* **do**

$z \leftarrow z - \eta \nabla f(z)$;

return z ;



η is called *learning rate*; sometimes a *time dependent* η is used
(e.g., “move” more at the beginning than at the end)

Stochastic Gradient Descent (SGD)

Idea: instead of using exactly the gradient, we take a (random) vector with *expected value* equal to the gradient direction.

SGD algorithm:

$\mathbf{z} \leftarrow$ random vector in \mathbb{R}^d ;

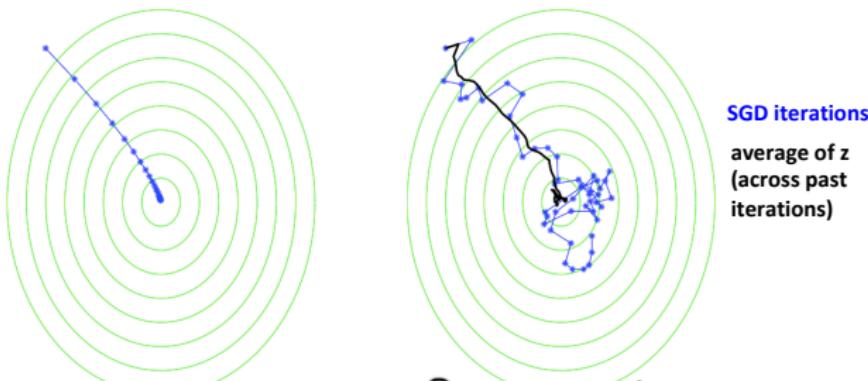
while not converged **do**

choose \mathbf{v} at random from a distribution such that $\mathbb{E}[\mathbf{v}|\mathbf{z}] \in \nabla f(\mathbf{z})$;

/* \mathbf{v} has *expected value* equal to the gradient of $f(\mathbf{z})$ */

$\mathbf{z} \leftarrow \mathbf{z} - \eta \mathbf{v}$;

return \mathbf{z} ;



Usually SGD is more efficient than GD for ML tasks

Adjacency-based Similarity Embeddings: GD

We want to find the vectors $\vec{z}_u, u \in V$, that minimize

$$\mathcal{L} = \sum_{u, v \in V} (\vec{z}_u^\top \vec{z}_v - A_{u,v})^2$$

$$= \sum_{u \in V} \sum_{v \in V} (\vec{z}_u^\top \vec{z}_v - A_{u,v})^2$$

GD algorithm:

- 1) initialize \vec{z}_u to a random vector, $\forall u \in V$
- 2) iterate until convergence:

for each $u \in V$:

i) compute $\frac{\partial \mathcal{L}}{\partial \vec{z}_u}$

ii) $\vec{z}_u \leftarrow \vec{z}_u - \eta \frac{\partial \mathcal{L}}{\partial \vec{z}_u}$

- 3) return $\vec{z}_u, \forall u \in V$

Complexity? At each iteration (step 2):

\mathcal{L} contains $|V|^2$ terms

\Rightarrow each iteration requires $\mathcal{O}(|V|^2)$ operations!

Can we do better?

SGD?

Adjacency-based Similarity Embeddings: SGD

We want to minimize: $\mathcal{L} = \sum_{u \in V} \sum_{v \in V} (\vec{z}_u^\top \vec{z}_v - A_{u,v})^2$

SGD algorithm:

- 1) initialize \vec{z}_u to a random vector in \mathbb{R}^d , $\forall u \in V$
- 2) iterate until convergence:

- i) pick $u \in V$ uniformly at random:

$$\mathcal{L}^{(u)} = \sum_{v \in V} (\vec{z}_u^\top \vec{z}_v - A_{u,v})^2$$

- ii) for all vertices $v \in V$:

- compute $\frac{\partial \mathcal{L}}{\partial \vec{z}_v}$

$$- \vec{z}_v \leftarrow \vec{z}_v - \eta \frac{\partial \mathcal{L}^{(u)}}{\partial \vec{z}_v}$$

- 3) return the values \vec{z}_u , $\forall u \in V$

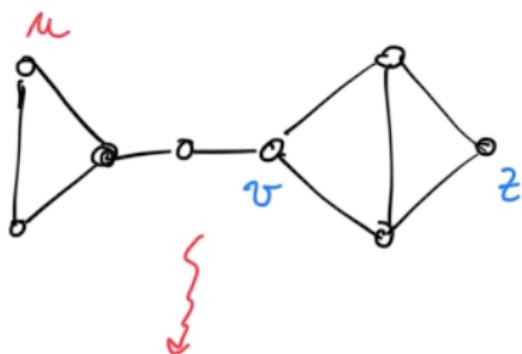
Complexity? In each iteration (step 2):

$\mathcal{L}^{(n)}$ has $\mathcal{O}(|V|)$ terms
 $\Rightarrow \mathcal{O}(|V|)$ operations

Improvement: consider "batches" of nodes (> 1 node)
so in step 2) you pick b nodes, and
compute the contribution to the loss of such nodes
and the gradient w.r.t. such contributions

Adjacency-based Similarity Embeddings: Limitation

While useful, adjacency-based similarity only considers direct connections among nodes



Adjacency-based similarity

$$S(v, z) = A_{v,z} = 0$$

$$S(v, u) = 0$$

$$S(u, z) = 0$$

Extended-Neighborhood Similarity Embeddings: Cons

Idea: define similarity between u and v as a measure of similarity between their neighborhoods $\mathcal{N}(u)$ and $\mathcal{N}(v)$



Jaccard-Similarity

Definition

Given two vertices u and v , the *Jaccard similarity* between their neighborhoods is:

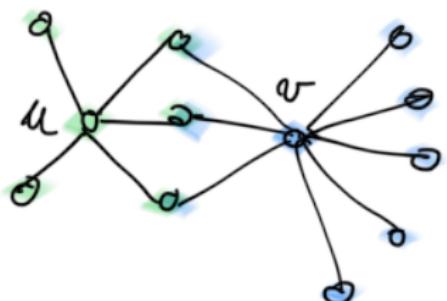
$$J(\mathcal{N}(u), \mathcal{N}(v)) = \frac{|\mathcal{N}(u) \cap \mathcal{N}(v)|}{|\mathcal{N}(u) \cup \mathcal{N}(v)|}$$

Here: $\mathcal{N}(u)$ also includes u

$$\mathcal{N}(u) = \{u\} \cup \{v : \{u, v\} \in E\}$$

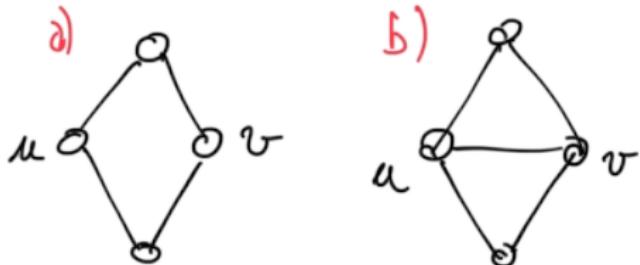
In this way, if $v \in \mathcal{N}(u)$, then $v \in \mathcal{N}(u) \cap \mathcal{N}(v)$

Example



$$\begin{aligned} \mathcal{J}(N(u), N(v)) \\ = \frac{3}{12} \quad (\text{if } z \notin N(z) : \frac{3}{10}) \end{aligned}$$

(if $z \in N(z) : \mathcal{J}(N(u), N(v))$)



$$a) \frac{2}{2} = 1$$

$$b) \frac{2}{4} = \frac{1}{2})$$

$$\begin{aligned} \mathcal{J}(N(u), N(v)) \\ = \frac{2}{4} = \frac{1}{2} \end{aligned}$$

$$\begin{aligned} \mathcal{J}(N(u), N(v)) \\ = \frac{4}{4} = 1 \end{aligned}$$

Embedding with Jaccard Similarity

Similarity: $S(u, v) = J(\mathcal{N}(u), \mathcal{N}(v))$

Encoder: shallow encoder $ENC(v) = \mathbf{z} \times \mathbf{v} = \mathbf{z}_v$

Decoder: $DEC(\mathbf{z}_u, \mathbf{z}_v) = \mathbf{z}_u^T \mathbf{z}_v$

Loss function: squared-loss

$$\ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v)) = (\mathbf{z}_u^T \mathbf{z}_v - J(\mathcal{N}(u), \mathcal{N}(v)))^2$$

Embeddings with Jaccard Similarity (continue)

Empirical loss:

$$\begin{aligned}\mathcal{L} &= \sum_{u,v \in V \times V} \ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v)) \\ &= \sum_{u,v \in V \times V} (\mathbf{z}_u^T \mathbf{z}_v - J(\mathcal{N}(u), \mathcal{N}(v)))^2\end{aligned}$$

Therefore the embeddings \mathbf{z}_v for each $v \in V$ are given by the solution \mathbf{Z}^* to

$$\mathbf{Z}^* = \arg \min_{\mathbf{Z}} \sum_{u,v \in V \times V} (\mathbf{z}_u^T \mathbf{z}_v - J(\mathcal{N}(u), \mathcal{N}(v)))^2$$

Embeddings with Jaccard Similarity: Notes

How do we compute \mathbf{Z}^* ? SGD

This approach can be generalized to consider an *extended neighborhood* of nodes, e.g. all nodes within distance k from u

$$\mathcal{N}_k(u) = \{v \in V : d(u, v) \leq k\}$$

distance between u, v in
the graph G

$$S(u, v) = \mathcal{J}(\mathcal{N}_k(u), \mathcal{N}_k(v))$$

Node Embeddings: Random-Walk Approaches

Idea: the similarity between u and v is given by the probability of visiting v on a random walk of length k starting at node u



Node Embeddings: Random-Walk Approaches

Idea: the similarity between u and v is given by the probability of visiting v on a random walk of length k starting at node u

Definition

Given a graph $G = (V, E)$, we define $\text{Pr}_k[v|u]$ as the probability of visiting node v on a random walk of length k starting at u .

Random Walks

What is a random walk of length k starting at u ?

Algorithm RandomWalk(u, k)

Input: node $u \in V$ of graph $G = (V, E)$; $k \in \mathbb{N}$

Output: random walk of length k starting from u

$v^{(0)} \leftarrow u;$

for $i \leftarrow 1$ to k **do**

$v^{(i)} \leftarrow$ random vertex chosen uniformly at random from $\mathcal{N}(v^{(i-1)})$;

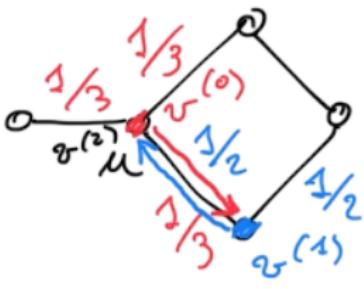
return;

$$v \notin \mathcal{N}(v)$$

Random Walk (u, v)

Example

G

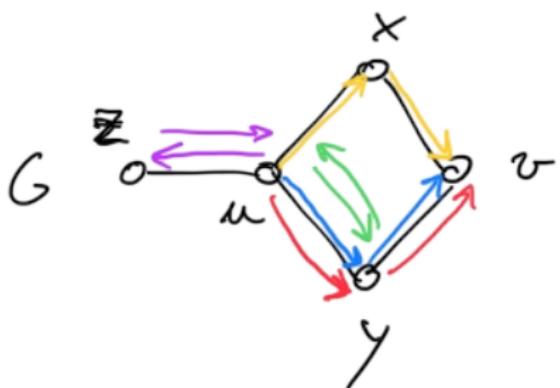


Probability of this specific random walk in G (starting from u)

$$\frac{1}{6} = \frac{1}{3} \cdot \frac{1}{2}$$

We want: $\Pr_k[v|u]$

Example



$$\Pr_1[v|u] = 0$$

$$\begin{aligned}\Pr_1[z|u] &= \frac{1}{3} = \Pr_1[x|u] \\ &= \Pr_1[y|u]\end{aligned}$$

$$\Pr_2[v|u] = \frac{1}{3} \cdot \frac{1}{2} + \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{3}$$

$$\Pr_2[y|u] = \frac{1}{3} \cdot \frac{1}{2} + \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{3}$$

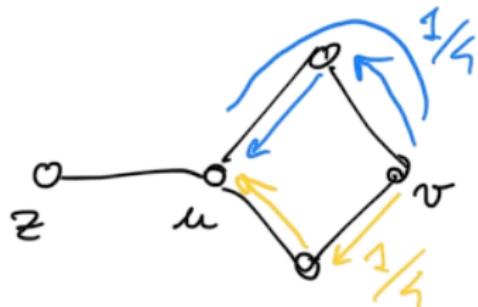
$$\Pr_2[x|u]$$

$$\Pr_2[z|u] = \frac{1}{3} \cdot 1 = \frac{1}{3}$$

$$S(u, v) = \Pr_2[v|u] = \frac{1}{3}$$

Node Embeddings: Random-Walk Approaches (continue)

Observation: in general, $\Pr_k[v|u] \neq \Pr_k[u|v]$



$$\Pr_2[z|u] = \frac{1}{3} \neq \frac{1}{4} = \Pr_2[u|z]$$
$$\Pr_2[v|u] = \frac{1}{3} \neq \frac{1}{2} = \Pr_2[u|v]$$

Random-Walk Approaches: Motivation

Why Random Walks?

Expressivity: flexible stochastic definition of node similarity that incorporates both *local* and *higher-order* neighborhood information

Efficiency: no need to consider all node pairs to learn the embedding, only need to consider pairs that co-occur on random walks.

[We will see this later.]

Random-Walk Approaches: Components

Similarity: $S(u, v) = \Pr_k[v|u]$

Encoder: shallow encoder $ENC(v) = \mathbf{z} \times \mathbf{v} = \mathbf{z}_v$

Decoder: $DEC(\mathbf{z}_u, \mathbf{z}_v) = \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$ (softmax function)

Note: the values $DEC(\mathbf{z}_u, \mathbf{z}_v)$ for all $v \in V$ define a *probability distribution*

In general: $S(u, v) \neq S(v, u)$

Is it true that $DEC(\vec{\mathbf{z}}_u, \vec{\mathbf{z}}_v) \neq DEC(\vec{\mathbf{z}}_v, \vec{\mathbf{z}}_u)$? YES: 21.5
NO: 9.5

missing mass: $\approx 2\%$

Random-Walk Approaches: Loss Function

Loss function: measure of distance between the true values $\Pr_k[v|u]$ and the predicted values $DEC(\mathbf{z}_u, \mathbf{z}_v) = \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$

The loss function is

$$\ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v)) = -\Pr_k[v|u] \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$

Random-Walk Approaches: Empirical Loss

Empirical loss:

$$\mathcal{L} = - \sum_{u,v \in V \times V} \Pr_k[v|u] \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$

Therefore the embeddings \mathbf{z}_v for each $v \in V$ are given by the solution \mathbf{Z}^* to

$$\mathbf{Z}^* = \arg \min_{\mathbf{Z}} - \sum_{u,v \in V \times V} \Pr_k[v|u] \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$

Loss Function: Motivation

Let's go back to the loss function:

$$\ell(DEC(\mathbf{z}_u, \mathbf{z}_v), S(u, v)) = -\Pr_k[v|u] \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$

Where does it come from?

Loss function: measure of distance between the true values $\Pr_k[v|u]$ for all $v \in V$ and the corresponding predicted values

$$DEC(\mathbf{z}_u, \mathbf{z}_v) = \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}} \text{ for all } v \in V$$

⇒ measure of the difference between two sets of probabilities

Loss Function: Motivation (continue)

Cross-entropy $H(p, q)$ between two probability distributions p, q is a measure of the difference between p and q .

Definition

Given two discrete probability distributions p and q with the same support \mathcal{X} , the cross-entropy between p and q is:

$$-\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

Therefore, for a given u , the difference between the true values $\Pr_k[v|u]$ and their predicted values $\frac{e^{z_u^T z_v}}{\sum_{i \in V} e^{z_u^T z_i}}$ is

$$-\sum_{v \in V} \underbrace{\Pr_k[v|u]}_{p(x)} \log \underbrace{\frac{e^{z_u^T z_v}}{\sum_{i \in V} e^{z_u^T z_i}}}_{q(x)}$$

Loss Function: Motivation (continue)

Summing such difference for all nodes u gives:

$$-\sum_{u \in V} \sum_{v \in V} \Pr_k[v|u] \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$

that is exactly the empirical loss

$$-\sum_{u,v \in V \times V} \Pr_k[v|u] \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$

Random-Walk Approaches: Computation

How do we find the best embedding \mathbf{Z}^* ?

$$\mathbf{Z}^* = \arg \min_{\mathbf{Z}} - \sum_{u,v \in V \times V} \Pr_k[v|u] \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$

From the previous slides:

$$\begin{aligned} \mathbf{Z}^* &= \arg \min_{\mathbf{Z}} \mathcal{L} \\ &= - \sum_{u \in V} \sum_{v \in V} \Pr_k[v|u] \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}} \end{aligned}$$

Complexity to evaluate \mathcal{L} ? $\Theta(|V|^2)$

⇒ too expensive for large networks!

Random-Walk Approaches: Optimizations

$$-\sum_{u \in V} \sum_{v \in V} \Pr_k[v|u] \log \frac{e^{z_u^T z_v}}{\sum_{i \in V} e^{z_u^T z_i}}$$

Instead of considering all terms $\Pr_k[v|u]$, consider only vertices v with *high* $\Pr_k[v|u]$.

Given u , how do we find vertices v with high $\Pr_k[v|u]$?

$\Pr_k[v|u]$ as the probability of visiting node v on a random walk of length k starting at u

⇒ run random walks of length k from u : if $\Pr_k[v|u]$ is high, v will appear in the random walks!

Random-Walk Approaches: Optimizations (continue)

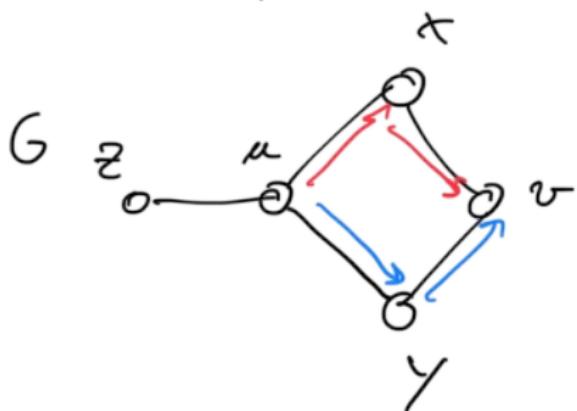
Definition

Let $\mathcal{N}_{RW,k}(u)$ be the set of nodes visited on a random walk of length k starting from node u .

Definition

Consider s random walks of length k starting from u . Let $n_{RW,k,s}(v|u)$ be the number of such walks containing v .

$$K=2, S=2$$



Example

$$\mathcal{N}_{RW,2}(u) = \{x, v\}$$

$$\mathcal{N}_{RW,2}(u) = \{y, v\}$$

$$n_{RW,2,2}(x|u) = 1$$

$$n_{RW,2,2}(y|u) = 1$$

$$n_{RW,2,2}(v|u) = 2$$

$$n_{RW,2,2}(z|u) = 0$$

Random-Walk Approaches: Optimizations (continue)

Note that $n_{RW,k,s}(v|u)$ is a random variable.

Proposition

$$\mathbb{E}\left[\frac{n_{RW,k,s}(v|u)}{s}\right] = \Pr_k[v|u]$$

Proof For $i = 1, \dots, s$ let $X_{v,i} = \begin{cases} 1 & \text{if } v \text{ appears in} \\ & \text{the } i\text{-th RW (of} \\ & \text{length } k) \text{ frame} \\ 0 & \text{otherwise} \end{cases}$

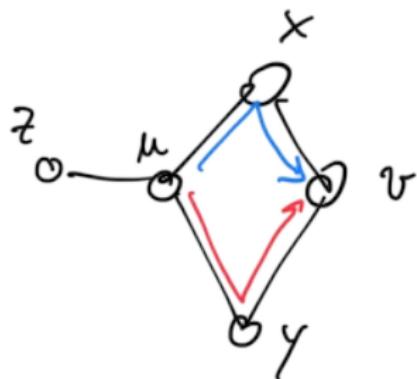
Then $n_{RW,k,s}(v|u) = \sum_{i=1}^s X_{v,i}$. Then

$$\begin{aligned} \mathbb{E}\left[\frac{n_{RW,k,s}(v|u)}{s}\right] &= \frac{1}{s} \mathbb{E}[n_{RW,k,s}(v|u)] = \frac{1}{s} \sum_{i=1}^s \mathbb{E}[X_{v,i}] \\ &= \frac{1}{s} \sum_{i=1}^s \Pr[X_{v,i} = 1] = \frac{1}{s} \sum_{i=1}^s \Pr_k[v|u] = \Pr_k[v|u]. \quad \square \end{aligned}$$

Random-Walk Approaches: Optimizations (continue)

Let $\mathcal{M}_{RW,k,s}(u)$ be the multiset of nodes visited during s random walks of length k starting from u , where vertex v appears in $\mathcal{M}_{RW,k,s}(u)$ exactly $n_{RW,k,s}(v|u)$ times

Example



$$\mathcal{M}_{RW,2,2}(u) = \{x, v, y, v\}$$

Random-Walk Approaches: Optimizations (continue)

We can use $\frac{n_{RW,k,s}(v|u)}{s}$ instead of $\Pr_k[v|u]$ in the empirical loss.

$$\mathcal{L} = - \sum_{u \in V} \sum_{v \in V} \Pr_k[v|u] \log \left(\frac{e^{\vec{z}_u^\top \vec{z}_v}}{\sum_{i \in V} e^{\vec{z}_u^\top \vec{z}_i}} \right) = (\star)$$

$$\approx - \sum_{u \in V} \sum_{v \in V} \frac{n_{RW,k,s}(v|u)}{s} \log (\star)$$

$$\text{constant} = - \left(\frac{1}{s} \right) \sum_{u \in V} \sum_{v \in V} n_{RW,k,s}(v|u) \log (\star)$$

$$\Rightarrow \vec{z}^* = \arg \min_{\vec{z}} \mathcal{L} \approx \arg \min_{\vec{z}} - \sum_{u \in V} \sum_{v \in V} n_{RW,k,s}(v|u) \log (\star)$$

$$\text{Consider: } \sum_{u \in V} n_{RW,k,s}(v|u) \log (\star)$$

Note that:

- if v is not visited by the s random walks of length k starting from node $u \Rightarrow h_{RW, k, s}(v|u) = 0 \Rightarrow$ ignore v
(in the sum above)

- if $h_{RW, k, s}(v|u) = l_v \neq 0$, then the term $\log(\star)$ appears l_v times in the sum above due to v , and l_v is the # of times that v appears in the multiset

$M_{RW, k, s}(u)$

v appears exactly
 $h_{RW, k, s}(v|u)$
Times

Therefore:

$$\sum_{v \in V} h_{RW, k, s}(v|u) \log(\star) = \sum_{v \in M_{RW, k, s}(u)} \log(\star)$$

$$\text{Thus: } Z^* = \arg \min_Z \mathcal{L} \approx \arg \min_Z - \sum_{u \in V} \sum_{v \in M_{RW, k, s}(u)} \log(\star)$$

Random-Walk Approaches: Optimizations (continue)

Overall, the approach is:

- ① run s random walks of length k from each node $u \in V$ to }
obtain $\mathcal{M}_{RW,k,s}(u)$ can be done in parallel
- ② solve the following problem:

$$\mathbf{z}^* = \arg \min_{\mathbf{z}} \mathcal{L}$$

$$\approx \arg \min_{\mathbf{z}} - \sum_{u \in V} \sum_{v \in \mathcal{M}_{RW,k,s}(u)} \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$

How? GD/SGD

Random-Walk Approaches: Optimizations (continue)

Overall, the approach is:

- ① run s random walks of length k from each node $u \in V$ to obtain $\mathcal{M}_{RW,k,s}(u)$
- ② solve the following problem:

$$\begin{aligned}\mathbf{Z}^* &= \arg \min_{\mathbf{Z}} \mathcal{L} \\ &= - \sum_{u \in V} \sum_{v \in \mathcal{M}_{RW,k,s}(u)} \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}\end{aligned}$$

Complexity? $\Theta(|V|^2) \dots$

DeepWalk

What we have described up to know corresponds to the method DeepWalk, described in Perozzi et al. (2014), *DeepWalk: online learning of social representations*, ACM KDD.

The paper presents also an optimization to reduce the complexity to $\Theta(|V| \log |V|)$

DeepWalk: Experimental Evaluation

From Perozzi et al. (2014), *DeepWalk: online learning of social representations*, ACM KDD.

Datasets:

Name	BLOGCATALOG	FLICKR	YOUTUBE
$ V $	10,312	80,513	1,138,499
$ E $	333,983	5,899,882	2,990,443
$ \mathcal{Y} $	39	195	47
Labels	Interests	Groups	Groups

- randomly sample a portion of the labeled nodes, use them as training data
- rest of the nodes are used as test
- repeat the process 10 times, and report the average
- dimension $d = 128$ for DeepWalk
- use a one-vs-rest logistic regression for classification

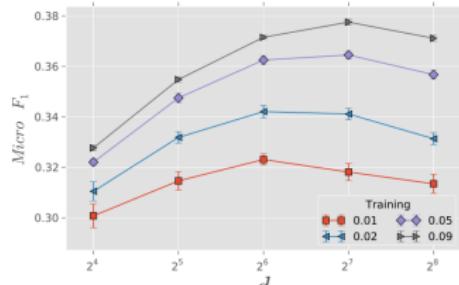
DeepWalk: Experimental Evaluation (continue)

	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1(%)	DEEPWALK	32.4	34.6	35.9	36.7	37.2	37.7	38.1	38.3	38.5	38.7
	SpectralClustering	27.43	30.11	31.63	32.69	33.31	33.95	34.46	34.81	35.14	35.41
	EdgeCluster	25.75	28.53	29.14	30.31	30.85	31.53	31.75	31.76	32.19	32.84
	Modularity	22.75	25.29	27.3	27.6	28.05	29.33	29.43	28.89	29.17	29.2
	wvRN	17.7	14.43	15.72	20.97	19.83	19.42	19.22	21.25	22.51	22.73
	Majority	16.34	16.31	16.34	16.46	16.65	16.44	16.38	16.62	16.67	16.71
Macro-F1(%)	DEEPWALK	14.0	17.3	19.6	21.1	22.1	22.9	23.6	24.1	24.6	25.0
	SpectralClustering	13.84	17.49	19.44	20.75	21.60	22.36	23.01	23.36	23.82	24.05
	EdgeCluster	10.52	14.10	15.91	16.72	18.01	18.54	19.54	20.18	20.78	20.85
	Modularity	10.21	13.37	15.24	15.11	16.14	16.64	17.02	17.1	17.14	17.12
	wvRN	1.53	2.46	2.91	3.47	4.95	5.56	5.82	6.59	8.00	7.26
	Majority	0.45	0.44	0.45	0.46	0.47	0.44	0.45	0.47	0.47	0.47

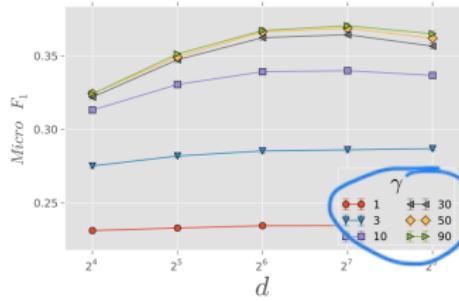
Table 3: Multi-label classification results in FLICKR

DeepWalk: Experimental Evaluation (continue)

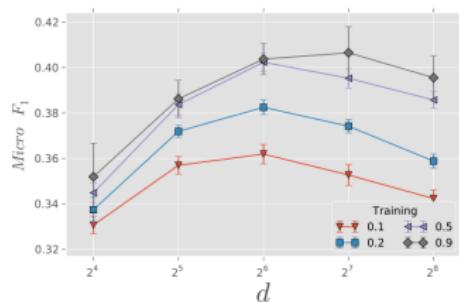
IR



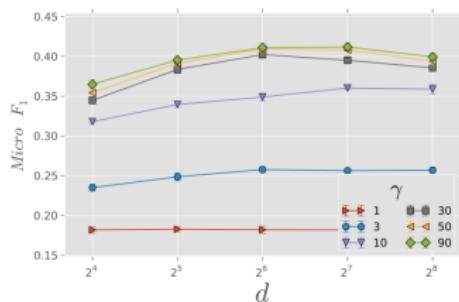
(a1) FLICKR, $\gamma = 30$



(a2) FLICKR, $T_R = 0.05$



(a3) BLOGCATALOG, $\gamma = 30$



(a4) BLOGCATALOG, $T_R = 0.5$

(a) Stability over dimensions, d

random walks

node2vec

A. Grover et al. (2016), *node2vec: Scalable Feature Learning for Networks*, ACM KDD.

Basic idea: as described for random walks approaches.

Presents a different approach to reduce the complexity.

Also based on a different type of random walks.

→ *See the DRAFT slides.*

node2vec: Motivation

The random walk approach is:

- ① run s random walks of length k from each node $u \in V$ to obtain $\mathcal{M}_{RW,k,s}(u)$
- ② solve the following problem:

$$\mathbf{Z}^* = \arg \min_{\mathbf{Z}} \mathcal{L}$$

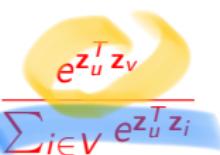
$$= \arg \min_{\mathbf{Z}} - \sum_{u \in V} \sum_{v \in \mathcal{M}_{RW,k,s}(u)} \log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$

Complexity is $\Theta(|V|^2)$, due to computation of $\underbrace{\log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}}_{\text{for all } u \in V}$ for all $u \in V$

Can we approximate it?

Negative Sampling

Idea: the softmax

$$\log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$


is a way to compare the (decoded) probability for node v (given node u) and the *background* probabilities of all nodes given u .

Negative Sampling

Idea: the softmax

$$\log \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{i \in V} e^{\mathbf{z}_u^T \mathbf{z}_i}}$$

is a way to compare the (decoded) probability for node v (given node u) and the *background* probabilities of all nodes given u .

Therefore:

- use a different function to obtain a probability for a node v (given node u)
- sample random nodes to obtain the *background* probabilities

Negative Sampling (continue)

node2vec uses the following to compare the (decoded) probability for node v (given node u) and the *background* probability:

$$\log \left(\sigma(\mathbf{z}_u^T \mathbf{z}_v) \right) - \sum_{i=1}^r \log \left(\sigma(\mathbf{z}_u^T \mathbf{z}_i) \right)$$

where:

- i is a node chosen from $|V|$ with probability proportional to its degree
- $\sigma()$: sigmoid function

The r samples are called **negative events**.

Negative Sampling (continue)

What about the choice of r ?

- large $r \Rightarrow$ robust estimates
- large $r \Rightarrow$ stronger focus on negative events

In practice: $r \in \{5, \dots, 20\}$

Putting everything together, the problem to be solved is

$$\begin{aligned}\mathbf{z}^* &= \arg \min_{\mathbf{z}} \mathcal{L} \\ &= \arg \min_{\mathbf{z}} - \sum_{u \in V} \sum_{v \in \mathcal{M}_{RW,k,s}(u)} \left(\log \left(\sigma(\mathbf{z}_u^T \mathbf{z}_v) \right) - \sum_{i=1}^r \log \left(\sigma(\mathbf{z}_u^T \mathbf{z}_i) \right) \right)\end{aligned}$$

How to find \mathbf{z}^* ? GD / SGD

node2vec: Random Walks

Other difference between DeepWalk and node2vec: type of random walks used:

- DeepWalk: random walks as presented before
- node2vec: uses *biased random walks*

node2vec: Biased Random Walks

Idea: in iteration i and the current node $v^{(i)}$, compute weights to choose the next node $v^{(i+1)}$ in the node; the weights depend on the distance of a candidate next node from the previous vertex $v^{(i-1)}$.

Algorithm RandomWalk(u, k, p)

Input: node $u \in V$ of graph $G = (V, E)$; $k \in \mathbb{N}$; function

$$p : V \times V \rightarrow [0, 1]$$

Output: random walk of length k starting from u

$$v^0 \leftarrow u;$$

for $i \leftarrow 1$ to k **do**

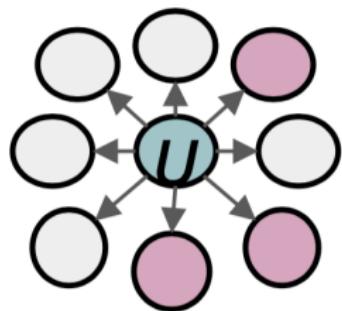
$v^{(i)} \leftarrow$ vertex chosen from $\mathcal{N}(v^{(i-1)})$ according to the probabilities
 $p(v^{(i-1)}, v^{(i)})$;

return;

node2vec: Biased Random Walks (continue)

Idea: the weights are chosen to combine and balance BFS-like visits and DFS-like visits

BFS vs DFS



BFS



DFS

BFS: micro-view of the neighborhood

DFS: macro-view of the neighborhood

node2vec: Biased Random Walks (continue)

node2vec uses random walks of length k where given a node u the weights of edges between u and the nodes in $\mathcal{N}(u)$ depend on two parameters:

- *return parameter p* : regulates the probability to go back to the previous node in the walk
- *in-out parameter q* : regulates the “ratio” of BSF moves vs DFS moves

node2vec: Biased Random Walks (continue)

Assume the random walk has just moved from node u to node v .

Observation: the neighbours of v can only be:

- the previous node u
- at the same distance of v from u
- further than v from u

Assign to each edge (v, z) , with $z \in \mathcal{N}$, a weight $w(v, z)$ as follows:

- $w(v, u) = \frac{1}{p}$
- $w(v, z) = 1$ if $z \in \mathcal{N}(u)$
- $w(v, z) = \frac{1}{q}$ if $z \notin \mathcal{N}(u)$

Then the probability $p(v, z)$ is

$$p(v, z) = \frac{w(v, z)}{\sum_{z' \in \mathcal{N}(v)} w(v, z')}$$

Example

node2vec: Biased Random Walks (continue)

The parameters p and q control the *type* of random walk performed.

Assume the random walk has just moved from node u to node v .

Impact of p :

- p is “high” ($> \max\{q, 1\}$): reduced probability to go back to u , encourages moderate exploration
- p is low ($< \min\{q, 1\}$): lead the walk to *backtrack* a step, keep the walk “local”

Impact of q :

- q is “high” (> 1): random walk is biased towards nodes close to node $u \Rightarrow$ BFS-like walk
- q is low (< 1): random walk is biased towards nodes far from node $u \Rightarrow$ DFS-like walk

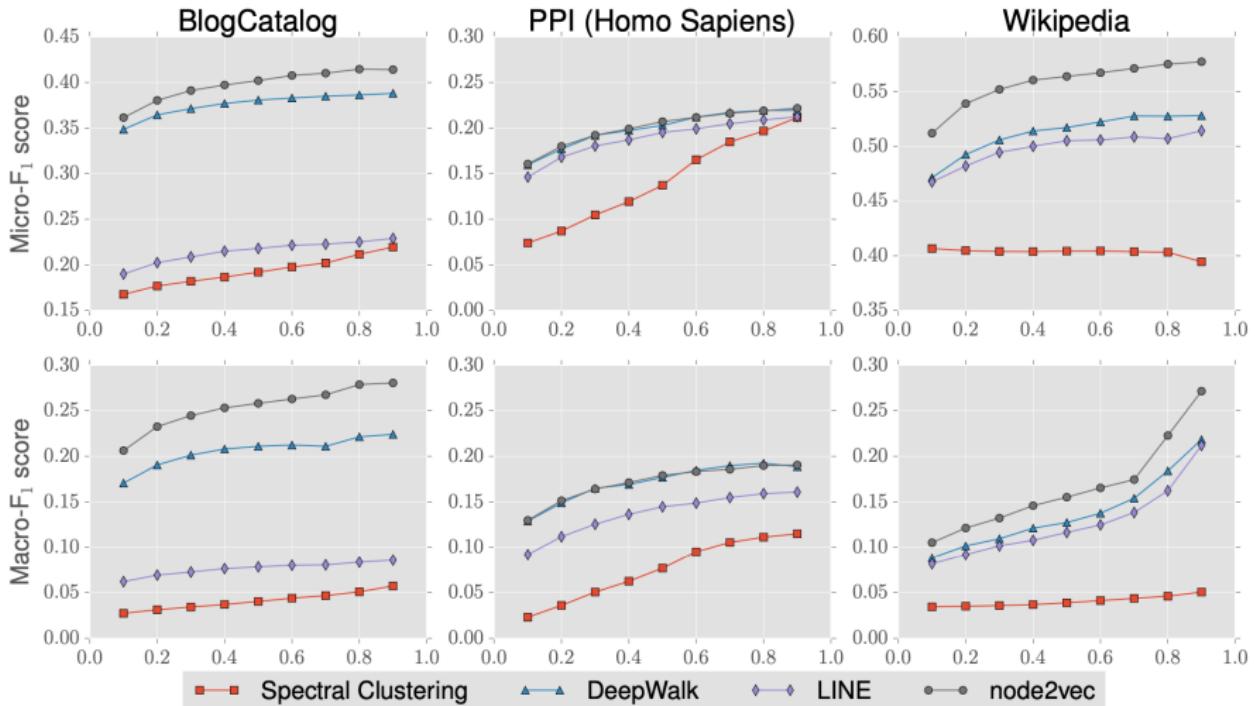
node2vec: Summary

The node2vec algorithm consists of the following steps:

- ① compute random walk probabilities $p(u, v)$ for all $u \in V$ and all $v \in N(u)$ [Details in the DRAFT slides]
- ② perform s biased random walks of length k from each $u \in V$ to compute $\mathcal{M}_{RW,k,s}(u)$
- ③ use SGD to find Z that minimizes

$$-\arg \min_Z \sum_{u \in V} \sum_{v \in \mathcal{M}_{RW,k,s}(u)} \left(\log (\sigma(z_u^T z_v)) - \sum_{i=1}^r \log (\sigma(z_u^T z_i)) \right)$$

node2vec: Experimental Evaluation



Random Walks: Other Approaches

Several other approaches based on random walks have been proposed.

- different kinds of biased random walks
 - using node attributes (Dong et al., 2017)
 - using weights learned from data (Abu-El-Haija et al., 2017)
- other optimization approaches
 - optimize 1-hop and 2-hop random walk probabilities (e.g., LINE from Tang et al., 2015)
- run random walks on modified versions of the initial graph (e.g., HARP from Chen et al., 2016; struct2vec from Ribeiro et al, 2017)

No method is the best in all situations!

In general: must choose the definition of node similarity that best suits the application