Graph embedding learning

Norbert TSOPZE

University of Yaounde I

tsopze.norbert@gmail.com

July 5, 2019

Overview

Overview

Definitions

Graph

Couple $G = (V, E)$, $V \neq \emptyset$ a set of vertices and $E \subseteq V \times V$ a set of edges

G is non-directed if $\forall (x, y) \in E$, $(y, x) \in E$ and directed otherwise. Attributed graph $G' = (V, E, X)$ is a graph with X a matrix of value described each node.

degree of $v \in V$

number of $u \in V$ connected to v

1. Out-degree
2. In-degree

Definitions

Path

Let u, v two nodes of V,a road from u to v is a set of $e_1, e_2, \ldots, e_k \in E$ such that $e_i = (v_{i-1}, v_i), v_0 = u$ et $v_k = v$

Connected component

subset $V_1 \subset V$ such that $\forall u, v \in V_1$ there exists a path from u to v

Representation

Approaches:

1. Incidence list ie $Succ(u) = \{v \ (u, v) \in E\}$
2. Adjacency matrix:

$$M_{i,j} = \begin{cases} 0 & (i,j) \notin E \\ 1 & (i,j) \in E \end{cases}$$

3. Incidence Matrix

Types of graph:

- Attributed graph : $G = (V, E, X)$
- Attributed and labelled graph : $G = (V, E, X, Y)$

Graph importance

- Capture interactions between actors representing by its nodes
- Important role in ML for making predictions or discovering new patterns, predict new role for an actor, recommend new friend...
- Applications:
  1. Recommender systems
  2. Social networks
  3. Bioinformatics

Overview

1. Introduction
2. Problem statement
   - Problem
   - Traditional approaches
3. Node representation learning
4. Subgraph representation learning

Problem

Question

How to incorporate information about graph structure into ML models?

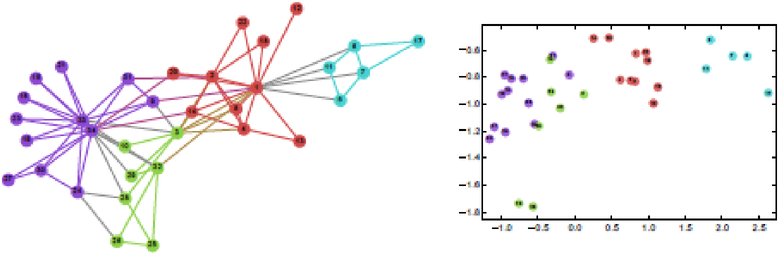
Idea

finding a way to represent, or encode, graph structure so that it can be easily exploited by machine learning models

Applications:

- Link prediction
- Node classification
- Graph clustering

Goal



(a) Input: Karate Graph (b) Output: Representation

Figure: Example of graph embedding

REf: [Perozzi et al (2017)]

Traditional approaches

Main idea

in the most cases user defined heuristics to extract features

- degree statistics
- kernel function
- Random walk
- measures of local neighborhood structures

Limits:

- inflexible : difficult to adapt during the learning process
- time-consuming
- User knowledge depending

Representation learning approach

Main idea

learn a mapping that embeds nodes, or subgraphs, as points in a low-dimensional vector space R^d

use as input adjacency matrix A and node attributes matrix X to map each node, or a subgraph, to a vector $z \in R^d$, where $d \ll |V|$.

- optimize this mapping so that geometric relationships in the embedding space reflect the structure of the original graph
- learned embeddings can be used as feature inputs for downstream machine learning tasks
- integrate as step in the model construction rather than the pre-processing step

Overview

Goal

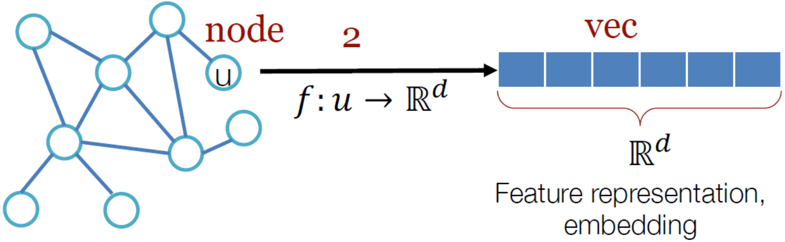


Figure: Main objective

Goal

encode nodes in a new space (feature vector) by approximating similarity in the original network

encoder

$Enc : V \rightarrow R^d$; maps nodes to vector embeddings $z_i \in R^d$

decoder

$Dec : R^d \times R^d \rightarrow R^+$;
$Dec(Enc(v_i), Enc(v_j)) = Dec(z_i, z_j) \approx Sim(v_i, v_j)$
accepts a set of node embeddings and decodes user-specified graph statistics from these embeddings

Loss function

$L = \sum_{(v_i,v_j)\in D} l(Dec(z_i, z_j), sim(v_i, v_j))$

Matrix factorization

Laplacian eigenmaps

$Dec(Z_i, Z_j) = \| Z_i - Z_j \|_2^2$
$l() = Dec(Z_i, Z_j).Sim(v_i, v_j)$

Inner Product

$Dec(Z_i, Z_j) = Z^T Z_j$
$l() = \| Dec(Z_i, Z_j) - Sim(v_i, v_j) \|_2^2$

Similarity functions:

1. HOPE : Jaccard
2. GRAREP: $Sim(v_i, v_j) \cong A_{ij}^2$
3. GF : $Sim(v_i, v_j) \cong A_{ij}$

Deepwalk

Random walk rooted at vertex v_i as W_{v_i}

stochastic process with random variables $W^1_{v_i}$, $W^2_{v_i}$,..., $W^k_{v_i}$ such that $W^{k+1}_{v_i}$ is a vertex chosen at random from the neighbors of vertex v_k after $k+1$ steps.

Language Modeling goal (one)

given a sequence of words $W^1_n = (w_0, w_1, ..., w_n)$, maximize $Pr(w_n / w_0, w_1, ..., w_{n-1})$ over the training corpus

By analogy RW is to estimate the likelihood $Pr(v_i / (v_1, v_2, ..., v_{i-1}))$

Deepwalk

Given a mapping function $\Phi : V \rightarrow R^{|V| \times d}$, the problem is to estimate $Pr(v_i / (\Phi(v_1), \Phi(v_2), ..., \Phi(v_{i-1})))$

Transformed problem

Inspired by Language Modeling, this yields the optimization problem: $minimize(-logPr(\{v_{i-w}, ..., v_{i+w}\} | v_i / \Phi(v_i)))$ under Φ

approximate by Skypgram as:

$$\prod_{j=i-w, j \neq i}^{i+w} Pr(v_j / \Phi(v_i))$$

Node2vec

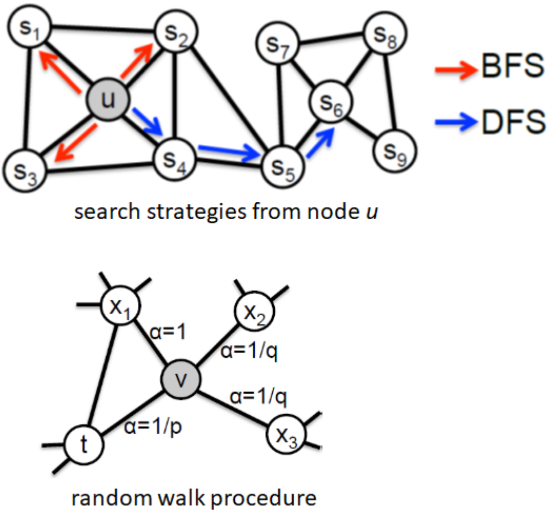


Figure: Node2vec strategy

Node2vec - Walks strategies

- Breadth-First-Sampling (BFS): Neigborhood N_s is restricted to nodes nodes which are immediate neigbors of the source u
- Depth-First-Sampling (DFS): Neigborhood consists of a sequence of sampling by increasing distance from the source u

Generation of C_i (i^{th} node in the walk) with $c_0 = u$

$$P(c_i = x / c_{i-1} = v) = \begin{cases} \frac{\prod_{vx}}{Z} & if \quad (v, x) \in E \\ 0 & else \end{cases}$$

$\prod_{vx}$: unnormalized transition probability between v and u;
Z normalizing constant

Node2vec

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & if \quad d_{tx} = 0 \\ 1 & if \quad d_{tx} = 1 \\ \frac{1}{q} & if \quad d_{tx} = 2 \end{cases}$$

with $d_{tx} \in \{0,1,2\}$
Consider a random walk that just traversed the edge $(t;v)$ and now resides at node v. The transition probabilities are evaluated $\prod_{vx}$ on edges $(v;x)$ leading from v

$$\prod_{vx} = \alpha_{pq}(t,x).w_{vx}$$

$\prod_{vx}$ is submitted to Sky-gram

Node2vec - edge features

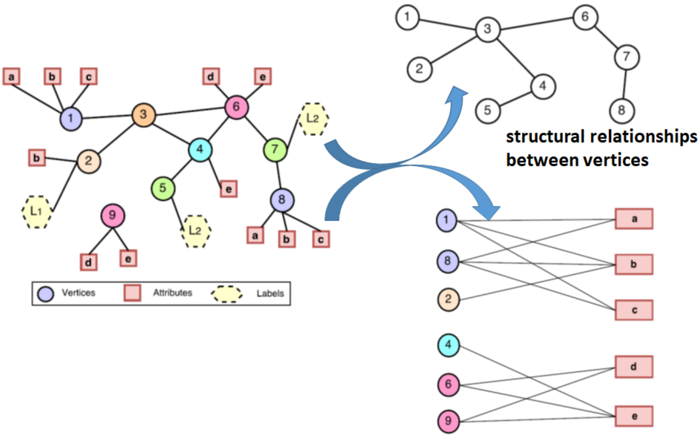
Given two nodes u and v, the edge (u, v) is embedded to $g(u, v) = f(u) \circ f(v)$.

- Average: $\frac{f(u)+f(v)}{2}$
- Hadamard $f(u) * f(v)$
- Weighted-L1 $|f(u) - f(v)|$
- Weighted-L2 $|f(u) - f(v)|^2$

Material: [Nasrullah et al (2019)]
Code: http://snap.stanford.edu/node2vec.

GAT2VEC

1. Network generation.
2. Random walks.
3. Representation learning.



GAT2VEC

1. Graph generation
   - structural graph $G_s = (V_s, E)$, $V_s \subseteq V$
   - bipartite graph $G_a = (V_a, A, E_a)$, $Va = \{v : A(v) \neq \emptyset\}$; $E_a = \{(v, a) : a \in A(v)\}$
2. perform on G_s and G_a
   - γ_s random walks of length λ_s for a corpus R
   - γ_a random walks of length λ_a to build a corpus W.
3. use the SkipGram to jointly learn an embedding based on these two contexts R and G

GAT2VEC

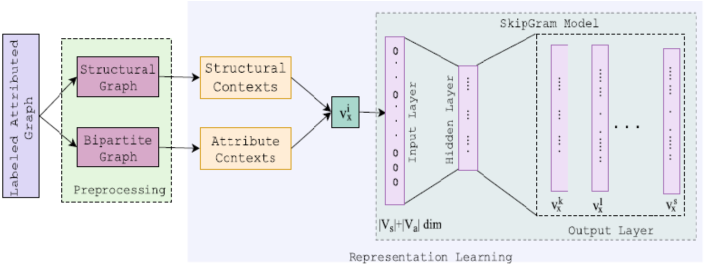


Figure: GAT2VEC Architecture

Materials: https://github.com/snash4/GAT2VEC.

Neighborhood autoencoder

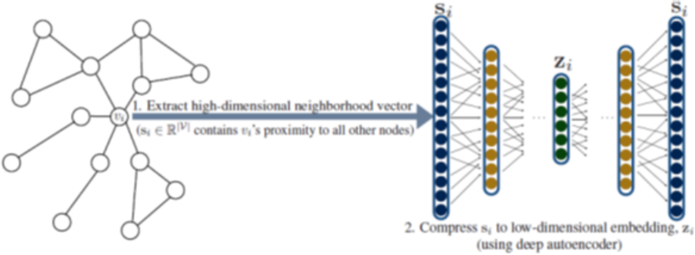


Figure: Neighborhood autoencoder Architecture

Neighborhood autoencoder

Goal

compress the node's neighborhood information into a low-dimensional vector.

1. Incorporate graph structure into the encoder algorithm
2. use autoencoders in order to compress information about a node's local neigborhood

Neighborhood autoencoder

For each node v_i with $s_i \in R^{|V|}$ its neigborhood vector cooresponds to to v_i's row in the similarity matrix S ($S_{ij} = S_g(v_i, v_j)$)

Objective

Embed nodes using the s_i vectors such that the s_i vector can be reconstructed:

$$Dec(Enc(S_i)) = Dec(Z_i) \approx s_i$$

Training autoencoder: minimize the loss function

$$\|Dec(Z_i) - s_i\|_2^2$$

- input dimension to the autoencoder is fixed at $|V|$, can be extremely costly and even intractable for graphs with millions of nodes.
- the structure and size of the autoencoder is fixed,cannot cope with evolving graphs, or generalize across graphs

Neighborhood aggregation

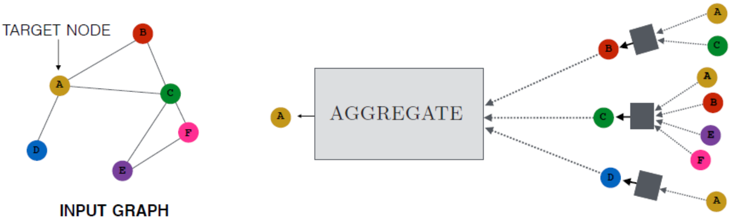


Figure: Neighborhood aggregation Architecture

Neighborhood aggregation

idea

generate embeddings for a node by aggregating information from its local neighborhood

- rely on node features or attributes
- use simple graph statistics as attributes in case where node features are not available

Neighborhood aggregation

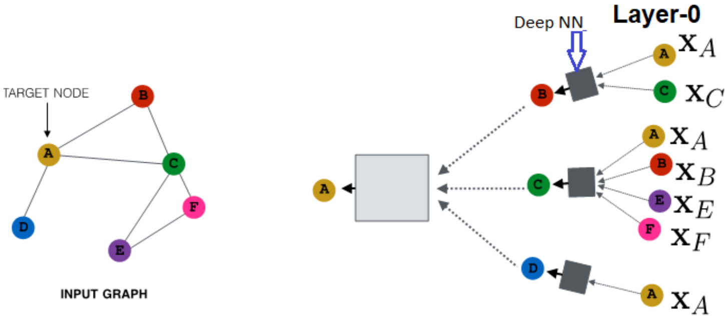


Figure: Neighborhood aggregation training

Ref: http://snap.stanford.edu/proj/embeddings-www/

Neighborhood aggregation

1. Node embedding initialization: using the nodes attributes
2. For each iteration:
    1. nodes aggregate the embeddings of their neighbors, using an aggregation function
    2. assign a new embedding to every node, equal to its aggregated neighborhood vector combined with its previous embedding from the last iteration
    3. feed the combined (average) embedding through a dense neural network layer
3. output the final embedding vectors

Overview

Neighborhood aggregation

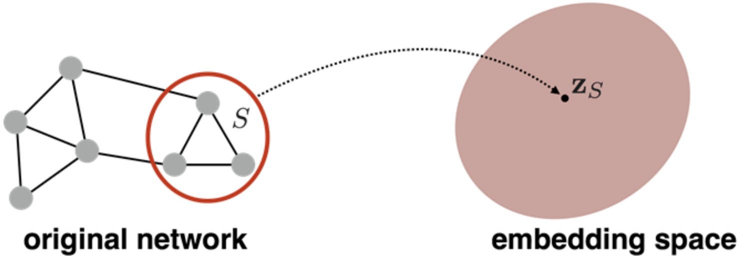


Figure: Subgraph embedding

Sets of node embeddings

Goal

encode a set of nodes and edges into a low-dimensional vector embedding

use the convolutional neighborhood aggregation idea to generate embeddings for nodes and then use additional modules to aggregate sets of node embeddings corresponding to subgraphs

Sum-based approaches

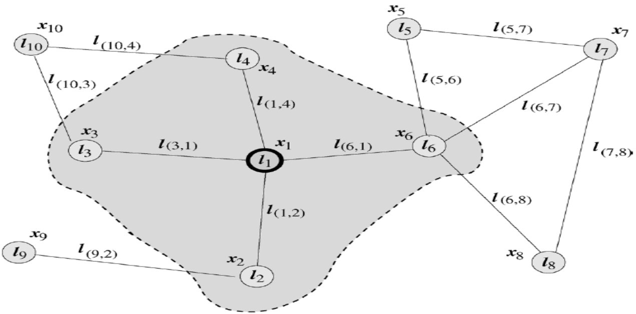
Main idea

represent subgraphs by summing all the individual node embeddings in the subgraph

$$Z_S = \sum_{v_i \in S} z_i$$

where $v_i \in S$ and its embedding z_i is generated using one of the previous algorithms

Graph neural networks



$$x_1 = f_w(l_1, \underbrace{l_{(1,2)}, l_{(3,1)}, l_{(1,4)}, l_{(6,1)}}_{l_{co[1]}}, \underbrace{x_2, x_3, x_4, x_6}_{x_{ne[1]}}, \underbrace{l_2, l_3, l_4, l_6}_{l_{ne[n]}})$$

Figure: Subgraph embedding

Ref: [Scarselli et al (2009)]

GNN

1. $x_n \in R^s$: label attached to the node n
2. local transition function f_w that expresses the dependence of a node on its neighborhood:
$$x_n = f_w(l_1, l_{co[1]}, x_{ne[1]}, l_{ne[n]})$$
3. local output function g_w, produces the output:
$$o_n = g_w(x_n, l_n)$$

l_n : label of n
$l_{co[1]}$: labels of n's edges,
$x_{ne[1]}, l_{ne[n]}$: states and labels of the nodes in the neighborhood of n

GNN

1. $f^{kn}_{w_{kn}}$: funtion f_w defined on the group of nodes kn
2. $g^{kn}_{w_{kn}}$: funtion g_w defined on the group of nodes kn
3. For the overall graph:$x = F_w(x, l)$ and $o = G_w(x, l_N)$
4. Learning task:

$$L = \{(G_i, n_{ij}, t_{ij}) | G_i = (N_i, E_i) \in G; n_{ij} \in N_i, t_{ij} \in R^m; \}$$

5. Cost function:

$$e_w = \sum_{i=1}^{p} \sum_{j=1}^{q_i} (t_{ij} - \varphi(G_i, n_{ij}))^2$$

References

William L. Hamilton, Zhitao Ying, Jure Leskovec,
Representation Learning on Graphs: Methods and Applications; in *IEEE Data Eng. Bull* 40,pp. 52-74, 2017.

Perozzi Bryan, Al-Rfou' Rami, Skiena Steven
DeepWalk: online learning of social representations; in *Proc. of KDD'14*, pp. 701-710, 2014.

Nasrullah Sheikh,Zekarias T. Kefato, Alberto Montresor
gat2vec: representation learning for attributed graphs; in *Computing*, Vol 101 pp. 187–209, 2019.

Grover Aditya, Leskovec Jure
node2vec: Scalable Feature Learning for Networks; in *Proc. of KDD'16*, 2016.

F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, and G. Monfardini.
The graph neural network model; in *IEEE Transactions on Neural Networks*, 20(1):6180, 2009.

Thank you