Deep Learning for Graphs - I

Michalis Vazirgiannis and Giannis Nikolentzos

DaSciM, LIX, École Polytechnique

November, 2023

Outline

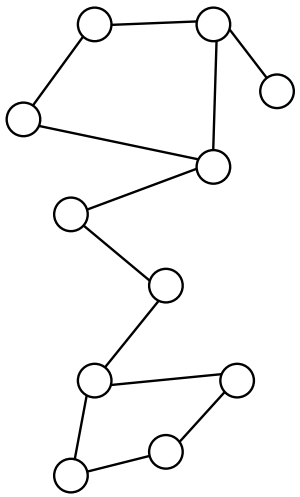
1. Learning Node Representations
   - Introduction
   - Unsupervised Methods
     - Proximity-based Approaches
     - Structural Equivalence-based Approaches
   - Supervised Methods

Outline

1. Learning Node Representations
   - Introduction
   - Unsupervised Methods
     - Proximity-based Approaches
     - Structural Equivalence-based Approaches
   - Supervised Methods

Traditional Node Representation

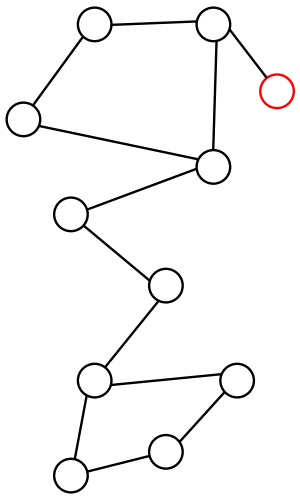
Representation: row of adjacency matrix



$$\rightarrow \quad \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \dots & 0 \end{pmatrix}$$

Traditional Node Representation

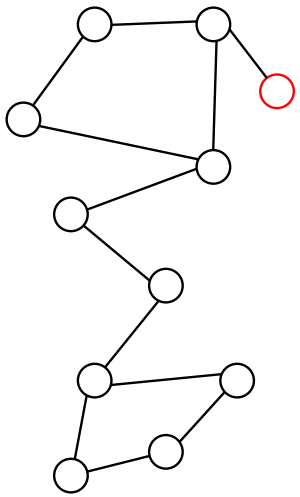
Representation: row of adjacency matrix



$$\rightarrow \quad \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \dots & 0 \end{pmatrix}$$

Traditional Node Representation

Representation: row of adjacency matrix



$$\rightarrow \quad \begin{pmatrix} 0 & 1 & \dots & 0 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \dots & 0 \end{pmatrix}$$

However, such a representation suffers from:

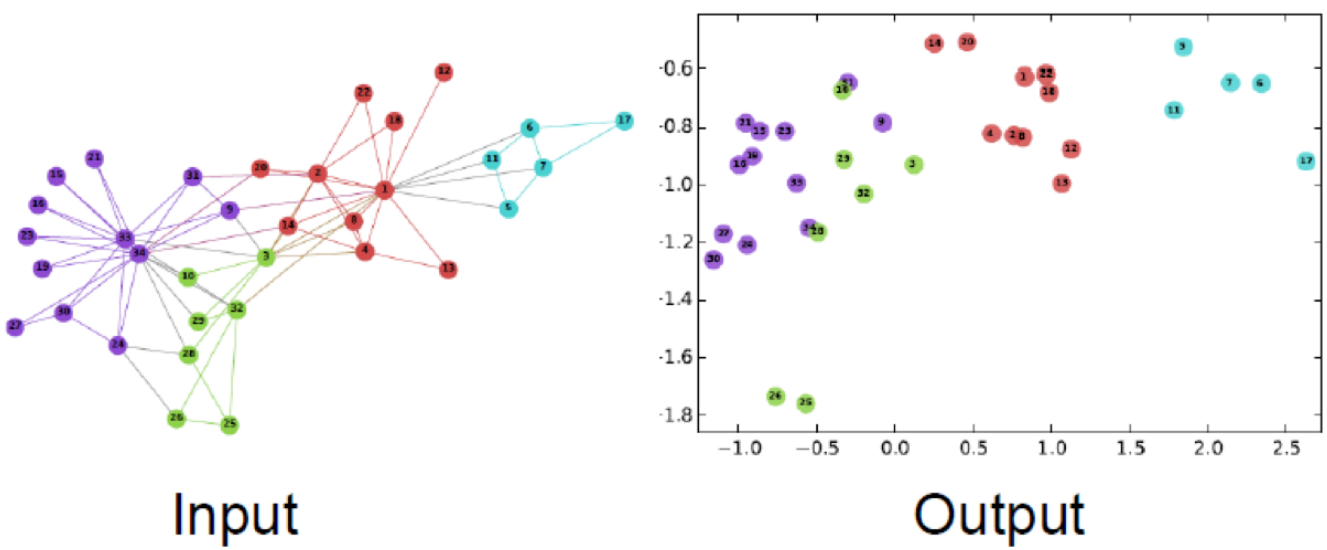
- data sparsity
- high dimensionality

$\vdots$

Node Embedding Methods

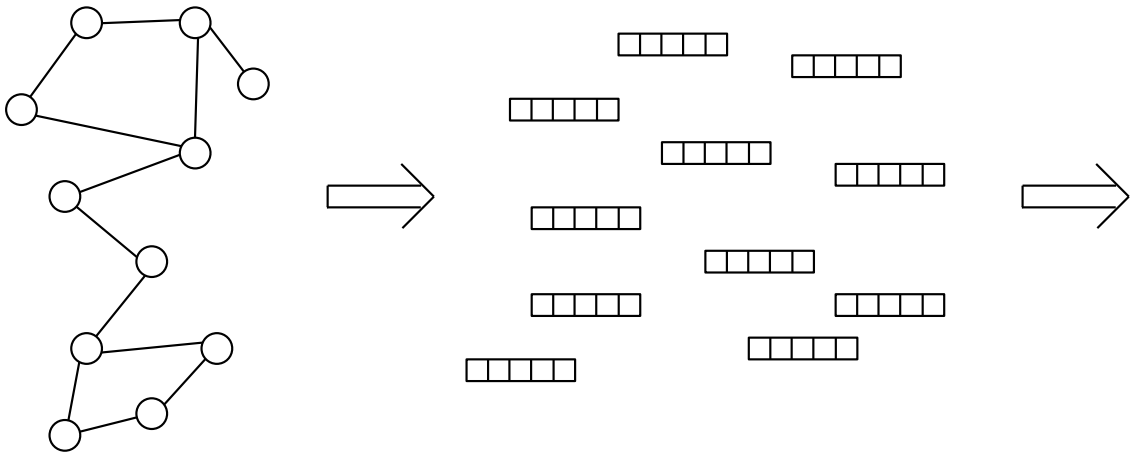
Map vertices of a graph into a low-dimensional space:

- dimensionality $d \ll |V|$
- similar vertices are embedded close to each other in the low-dimensional space



Input Output

Why Learning Node Representations?



Node Classification
Anomaly Detection
Link Prediction
Clustering
Recommendation
⋮

Examples:

- Recommend friends
- Detect malicious users

Early Methods

- Focused mainly on matrix-factorization approaches (e. g., Laplacian eigenmaps)
- Laplacian eigenmaps projects two nodes i and j close to each other when the weight of the edge between the two nodes A_{ij} is high
- Embeddings are obtained by the following objective function:

$$y^* = \arg\min \sum_{i \neq j} (y_i - y_j)^2 A_{ij} = \arg\min y^T L y$$

  where L is the graph Laplacian

- The solution is obtained by taking the eigenvectors corresponding to the d smallest eigenvalues of the normalized Laplacian matrix

[Belkin and Niyogi, NIPS'02]

Recent Methods

Most methods belong to the following groups:

1. Random walk based methods: employ random walks to capture structural relationships between nodes
2. Edge modeling methods: directly learn node embeddings using structural information from the graph
3. Matrix factorization methods: generate a matrix that represents the relationships between vertices and use matrix factorization to obtain embeddings
4. Deep learning methods: apply deep learning techniques to learn highly non-linear node representations

Outline

1. Learning Node Representations
   - Introduction
   - Unsupervised Methods
     - Proximity-based Approaches
     - Structural Equivalence-based Approaches
   - Supervised Methods

Outline

1. Learning Node Representations
   - Introduction
   - Unsupervised Methods
     - Proximity-based Approaches
     - Structural Equivalence-based Approaches
   - Supervised Methods

Node Embedding Methods

Map vertices of a graph into a low-dimensional space:

- dimensionality $d \ll |V|$
- **similar vertices** are embedded close to each other in the low-dimensional space

When two vertices are **similar** to each other?

- first-order proxmity
- second-order proxmity
- third-order proxmity

⋮

Node Embedding Methods

Map vertices of a graph into a low-dimensional space:

- dimensionality $d \ll |V|$
- **similar vertices** are embedded close to each other in the low-dimensional space

When two vertices are **similar** to each other?

$->$ first-order proxmity

$->$ second-order proxmity

$->$ third-order proxmity

$\vdots$

Proximities

Definition (First-order proximity)

The first-order proximity captures the direct neighboring relationships between vertices. If two vertices v and u are linked by an edge, the first-order proximity between them is determined by their edge weight, otherwise is equal to 0.

Definition (Second-order proximity)

The second-order proximity captures the 2-step relations between two vertices v and u. It describes the proximity of the neighborhood structures of v and u, and is thus determined by the number of common neighbors shared by the two vertices.

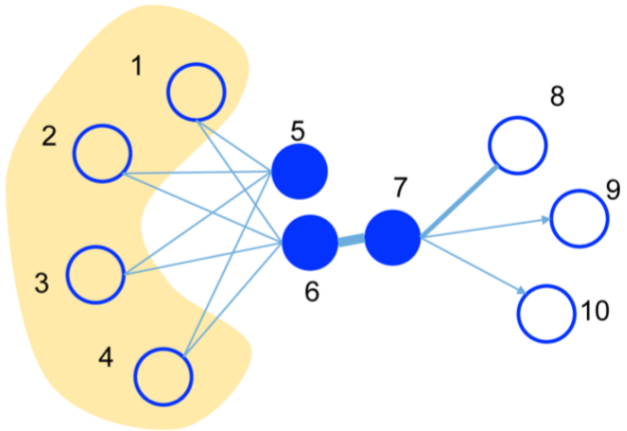
Definition (High-order proximity)

The high-order proximity captures the k-step relations ($k \geq 3$) between two vertices v and u. It is determined by the number of k-step paths from v to u.

Proximities

First-order proximity: observed links in the network

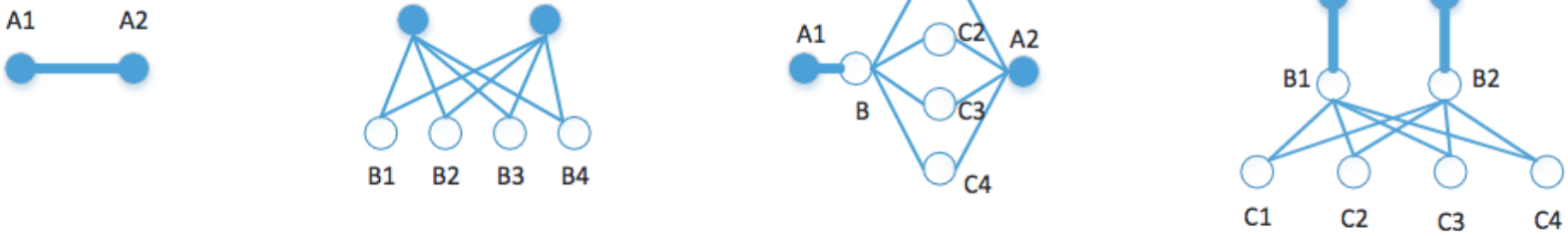
Second-order proximity: shared neighborhood structures



- Vertices 6 and 7 have a high *first-order proximity* since they are connected through a strong tie $\rightarrow$ they should be placed closely in the embedding space
- Vertices 5 and 6 have a high *second-order proximity* since they share similar neighbors $\rightarrow$ they should also be placed closely

Proximities

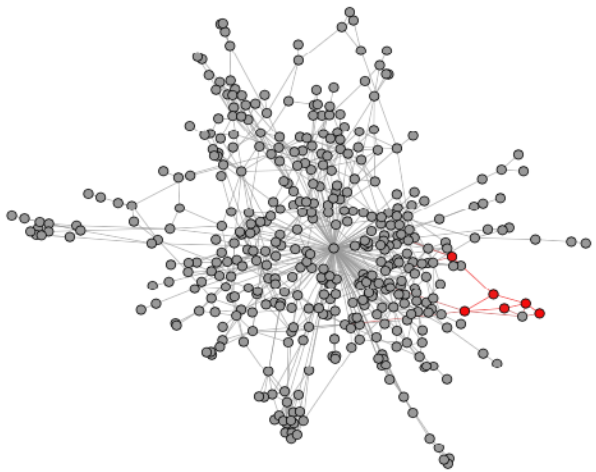
k-order proximities for $k = 1, \ldots, 4$



- Second-order and high-order proximities capture similarity between vertices with similar structural roles
- Higher-order proximities capture more global structure

DeepWalk

Inspired by recent advances in language modeling



$\rightarrow$

$v_5 \rightarrow v_8 \rightarrow v_{32} \rightarrow v_{28} \rightarrow v_6 \rightarrow v_{10} \rightarrow v_9$

$v_3 \rightarrow v_5 \rightarrow v_{28} \rightarrow v_8 \rightarrow v_9 \rightarrow v_{10} \rightarrow v_{25}$

$v_{20} \rightarrow v_{10} \rightarrow v_{12} \rightarrow v_6 \rightarrow v_8 \rightarrow v_4 \rightarrow v_5$

$v_{23} \rightarrow v_5 \rightarrow v_{32} \rightarrow v_{10} \rightarrow v_8 \rightarrow v_3 \rightarrow v_1$

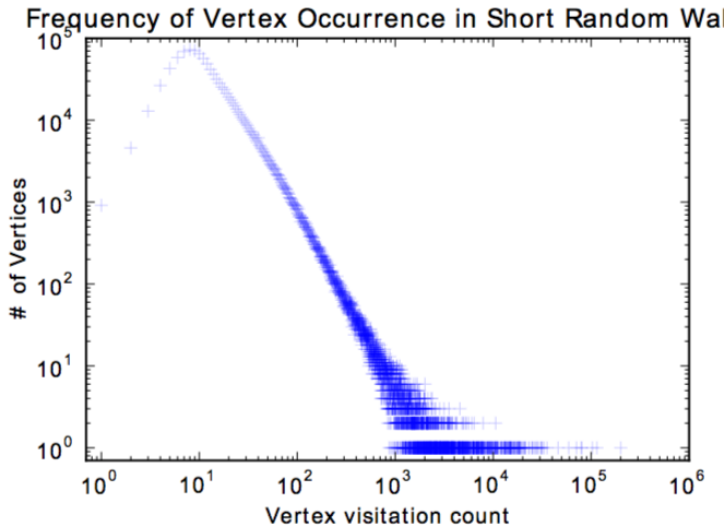
$v_4 \rightarrow v_3 \rightarrow v_1 \rightarrow v_5 \rightarrow v_1 \rightarrow v_{12} \rightarrow v_{10}$

$\vdots$

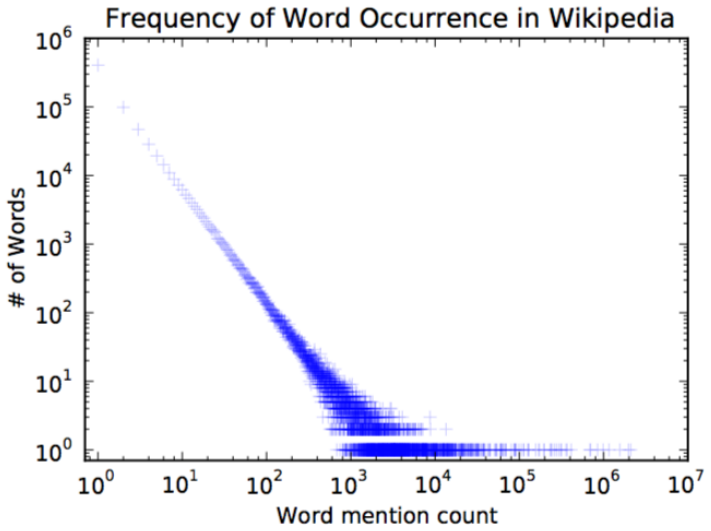
- Simulates a series of short random walks

[Perozzi et al., KDD'14]

Inspired by recent advances in language modeling



(a) YouTube Social Graph



(b) Wikipedia Article Text

- Simulates a series of short random walks
- **Main Idea**: Short random walks = Sentences

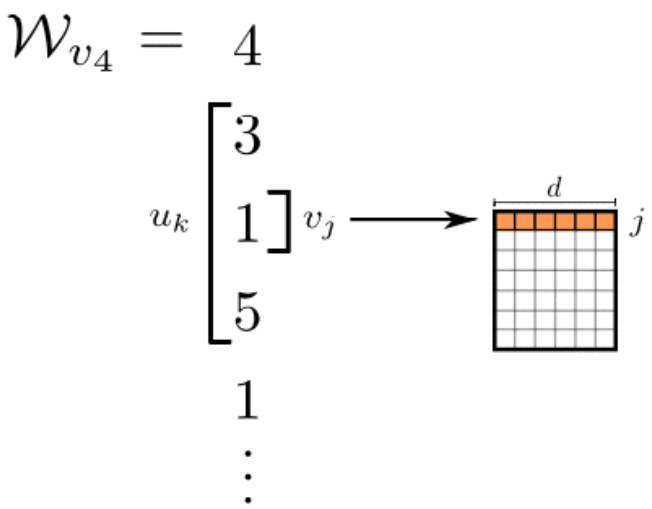
[Perozzi et al., KDD'14]

Skipgram

Skipgram is a recently-proposed language model that:

- uses one word to predict the context
- context is composed of words appearing to both the right and left of the given word
- removes the ordering constraint on the problem (i. e. does not take into account the offset of context words from the given word)

In our setting:



- Slide a window of length $2w + 1$ over the random walk
- Use the representation of central vertex to predict its neighbors

Skipgram

This yields the optimization problem:

$$\underset{f}{\text{minimize}} \quad -\frac{1}{T}\sum_{i=1}^{T} \log P(\{v_{i-w}, \ldots, v_{i+w}\} \setminus v_i | f(v_i))$$

v_i: central vertex
$v_{i-w}, \ldots, v_{i+w}$: neighbors of central vertex
$f(v)$: embedding of vertex v

Skipgram approximates the above conditional probability using the following independence assumption:

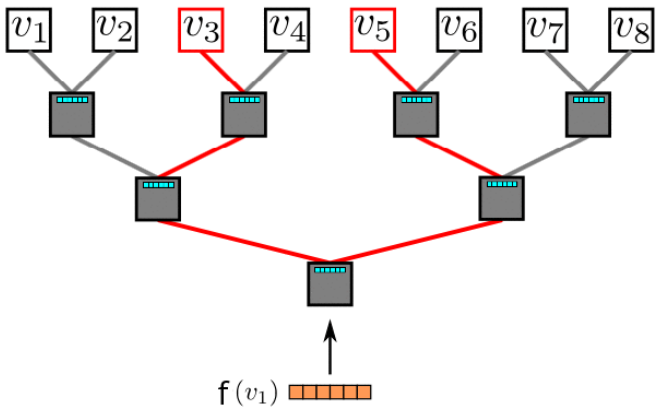
$$\underset{f}{\text{minimize}} \quad -\frac{1}{T}\sum_{i=1}^{T} \sum_{\substack{j=i-w \\ j\neq i}}^{i+w} \log P(v_j | f(v_i))$$

- We can learn such a posterior distribution using several choices of classifiers
- **However**, most of them (e. g., logistic regression) would produce a huge number of labels (i. e. $|V|$ labels)
- Instead, we approximate the distribution using the Hierarchical Softmax

Hierarchical Softmax

Reduces complexity from $\mathcal{O}(|V|)$ to $\mathcal{O}(\log|V|)$ using a binary tree

- Assigns the vertices to the leaves of a binary tree
- New problem: Maximizing the probability of a specific path in the hierarchy



If the path to vertex v_j is identified by a sequence of tree nodes $(b_0, b_1, \ldots, b_{\lceil \log|V| \rceil})$ then

$$P(v_j|f(v_i)) = \prod_{l=1}^{\lceil \log|V| \rceil} P(b_l|f(v_i))$$

where

$$P(b_l|f(v_i)) = 1/(1+e^{-f(v_i)^\top f'(b_l)}) = \sigma(f(v_i)^\top f'(b_l))$$

and $f'(b_l) \in \mathbb{R}^d$ is the representation assigned to tree node b_l's parent

node2vec

Like DeepWalk, node2vec is also a random walk based method

DeepWalk uses a *rigid* search strategy

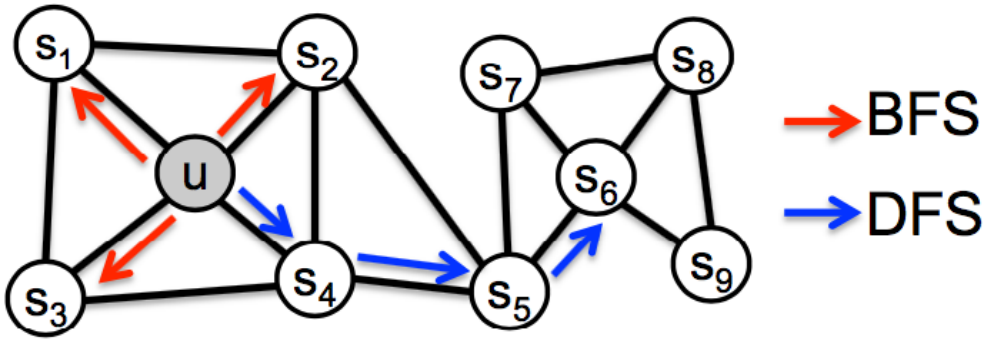
Conversely, node2vec simulates a family of biased random walks which

- explore diverse neighborhoods of a given vertex
- allow it to learn representations that organize vertices based on
  - their network roles
  - the communities they belong to

[Grover and Leskovec, KDD'16]

Two Extreme Sampling Strategies

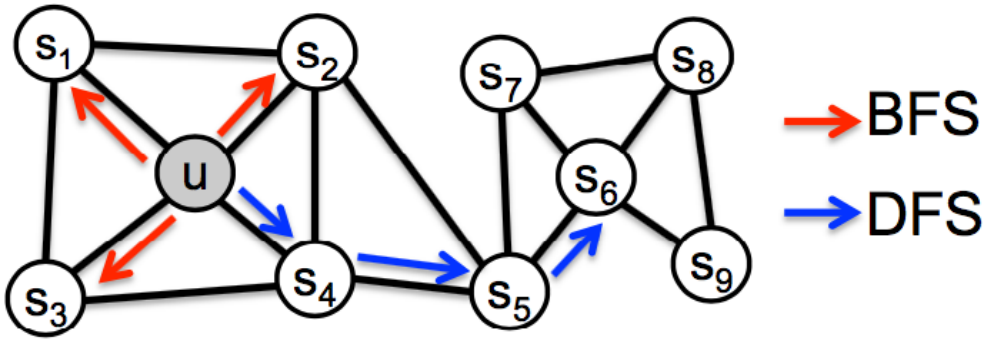
The *breadth-first sampling* (BFS) and *depth-first sampling* (DFS) represent extreme scenarios in terms of the search space



Goal: Given a source node u, sample its neighborhood $\mathcal{N}(u)$ where $|\mathcal{N}(u)| = k$

Two Extreme Sampling Strategies

The *breadth-first sampling* (BFS) and *depth-first sampling* (DFS) represent extreme scenarios in terms of the search space

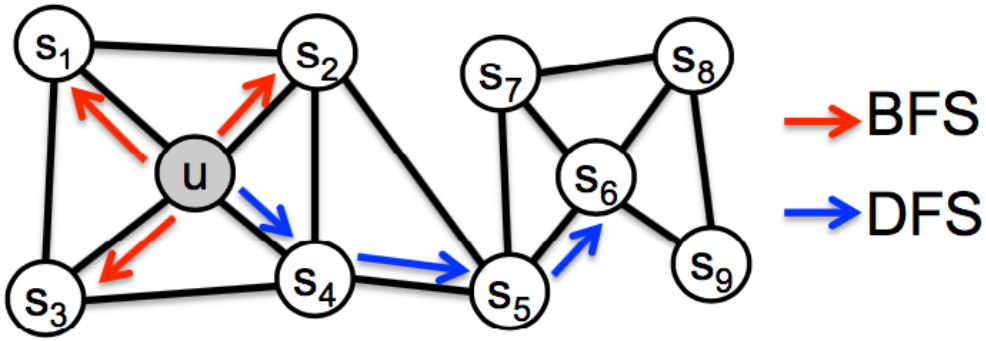


In most applications, we are interested in two kinds of similarities between vertices:

1. homophily: nodes that are highly interconnected and belong to similar communities should be embedded closely together (e. g., s_1 and u)
2. structural equivalence: nodes that have similar structural roles should be embedded closely together (e. g., u and s_6)

Two Extreme Sampling Strategies

The *breadth-first sampling* (BFS) and *depth-first sampling* (DFS) represent extreme scenarios in terms of the search space



BFS and DFS strategies play a key role in producing representations that reflect these two properties:

- The neighborhoods sampled by BFS lead to embeddings that correspond closely to structural equivalence
- The neighborhoods sampled by DFS reflect a macro-view of the neighborhood which is essential in inferring communities based on homophily

Random Walks of node2vec

Given a source node, node2vec simulates a random walk of fixed length l

$$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \ldots \rightarrow v_l$$

The i^{th} node in the walk is generated as follows:

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z}, & \text{if } (v, x) \in E \\ 0, & \text{otherwise} \end{cases}$$

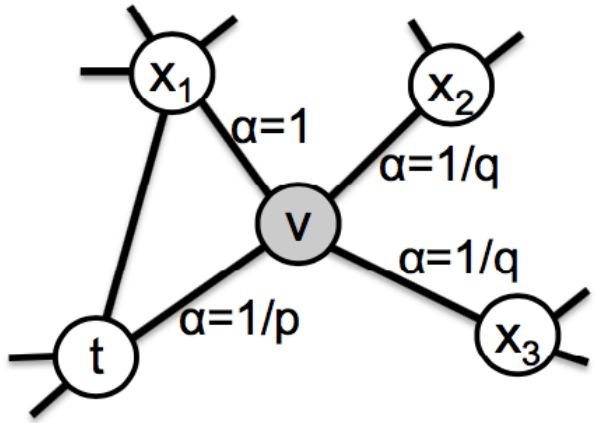
where π_{vx} is the unnormalized transition probability between v and x, and Z is a normalizing factor

To capture both structural equivalence and homophily, node2vec uses a neighborhood sampling strategy which

- is based on a flexible biased random walk procedure
- allows it to smoothly interpolate between BFS and DFS

Random Walks of node2vec

The random walk shown below just traversed edge (t, v) and now resides at node v



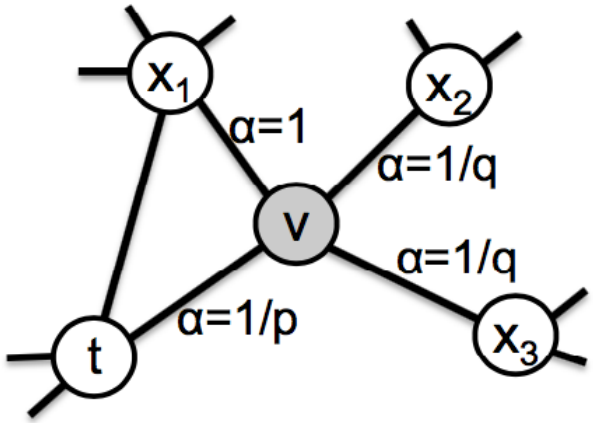
The unnormalized transition probability is $\pi_{vx} = w_{vx}\alpha_{pq}(t, x)$, where:

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

where d_{tx} denotes the shortest path distance between t and x

Random Walks of node2vec

The random walk shown below just traversed edge (t, v) and now resides at node v

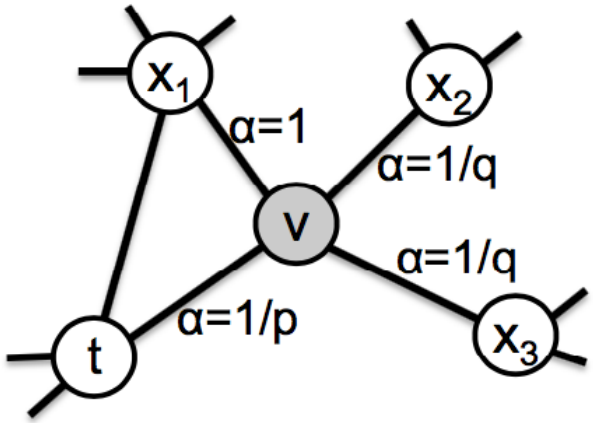


The *return parameter* p controls the likelihood of immediately revisiting a node in the walk

- if p is high, we are less likely to sample an already-visited node in the following two steps
- if p is low, it would keep the walk in the local neighborhood of the starting node

Random Walks of node2vec

The random walk shown below just traversed edge (t, v) and now resides at node v



The *in-out parameter* q allows the search to differentiate between "inward" and "outward" nodes.

- if q is high, the random walk is biased towards nodes close to node t
- if q is low, the walk is more inclined to visit nodes which are further away from the node t

Optimization

After defining the neighborhood $\mathcal{N}(v) \subset V$ of each node v, node2vec uses the Skipgram architecture:

$$\underset{f}{\text{minimize}} \quad -\sum_{v \in V} \log \prod_{u \in \mathcal{N}(v)} P(u|f(v))$$

where conditional likelihood is modelled as a softmax unit parametrized by a dot product of their features:

$$P(u|f(v)) = \frac{e^{f'(u)^\top f(v)}}{\sum_{k=1}^{|V|} e^{f'(v_k)^\top f(v)}}$$

and $f'(u) \in \mathbb{R}^d$ is the representation of node u when considered as context

The objective function thus becomes:

$$\underset{f,f'}{\text{minimize}} \quad -\sum_{v \in V} \left(-\log \sum_{u \in V} e^{f'(u)^\top f(v)} + \sum_{u \in \mathcal{N}(v)} f'(u)^\top f(v) \right)$$

Since learning the above posterior distribution is very expensive, node2vec approximates it using negative sampling

LINE

LINE employs an objective function that explicitly uses structural information from the graph to learn node representations

Specifically, LINE

- preserves both the first-order and second-order proximities
- trains two models separately
- concatenates the two learned embeddings for each vertex

[Tang et al., WWW'15]

LINE with First-order Proximity

To model the first-order proximity, for each undirected edge (v_i, v_j), define the joint probability between v_i and v_j as follows:

$$P_1(v_i, v_j) = \frac{1}{1 + e^{-f(v_i)^\top f(v_j)}}$$

where $f(v_i) \in \mathbb{R}^d$ is the low-dimensional vector representation of vertex v_i

The empirical probability can be defined as:

$$\hat{P}_1(v_i, v_j) = \frac{w_{ij}}{W}$$

w_{ij}: weight of the edge between v_i, v_j
W: sum of weights of all edges

LINE minimizes the KL-divergence of the two probability distributions:

$$\underset{f}{\text{minimize}} \quad - \sum_{(v_i, v_j) \in E} w_{ij} \log P_1(v_i, v_j)$$

LINE with Second-order Proximity

To model the second-order proximity, for each edge (v_i, v_j), LINE defines the probability of context v_j generated by vertex v_i:

$$P_2(v_j|v_i) = \frac{e^{f'(v_j)^\top f(v_i)}}{\sum_{k=1}^{|V|} e^{f'(v_k)^\top f(v_i)}}$$

$f(v_i)$: representation of v_i when treated as a vertex
$f'(v_i)$: representation of v_i when treated as context

The empirical probability can be defined as:

$$\hat{P}_2(v_j|v_i) = \frac{w_{ij}}{d_i}$$

d_i: out-degree of v_i

LINE minimizes the KL-divergence of the two probability distributions:

$$\underset{f,f'}{\text{minimize}} \quad -\sum_{(v_i,v_j)\in E} w_{ij} \log P_2(v_j|v_i)$$

LINE with Second-order Proximity

Optimizing the objective of the second-order proximity is computationally very *expensive*

Instead, use negative sampling: for each edge, sample multiple negative edges according to some noisy distribution

Every $\log P_2(v_j|v_i)$ term in the objective is replaced with:

$$\log \sigma(f'(v_j)^\top f(v_i)) + \sum_{k=1}^{K} \mathbb{E}_{v_k \sim P_n(v)}[\log \sigma(-f'(v_k)^\top f(v_i))]$$

where $\sigma = 1/(1+e^{-x})$ is the sigmoid function and K the number of negative edges

Experimental Evaluation

Experimental comparison conducted in [1]

Compared algorithms:

- DeepWalk
- GraRep
- SDNE
- LINE
- Laplacian Eigenmaps (LE)

[Wang et al., KDD'16]

Datasets

Five datasets:

- three social networks
- one citation network
- one language network

Dataset	#(V)	#(E)
BLOGCATALOG	10312	667966
FLICKR	80513	11799764
YOUTUBE	1138499	5980886
ARXIV GR-QC	5242	28980
20-NEWSGROUP	1720	Full-connected

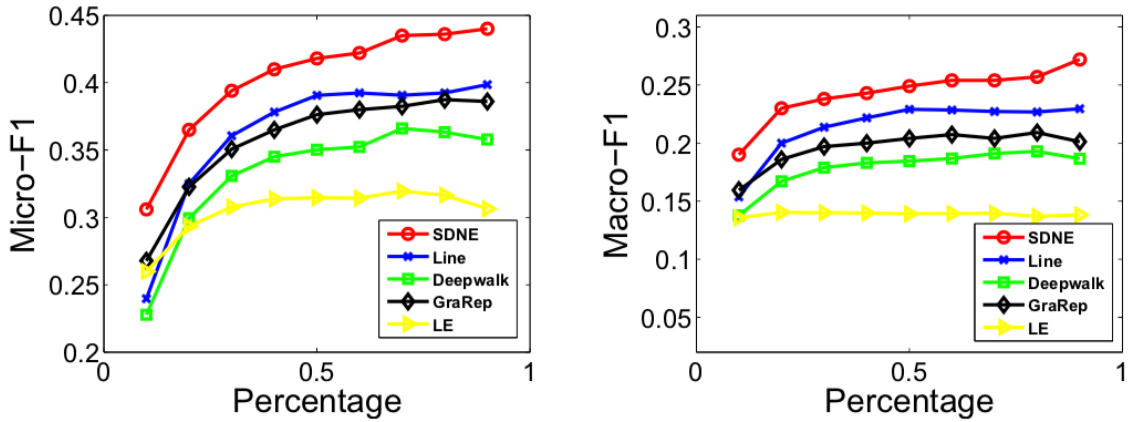
Three real-world applications

- node classification
- link prediction
- visualization

Node Classification

Vertex representations generated from node embedding methods and given as input to a logistic regression classifier to predict a set of labels for each vertex

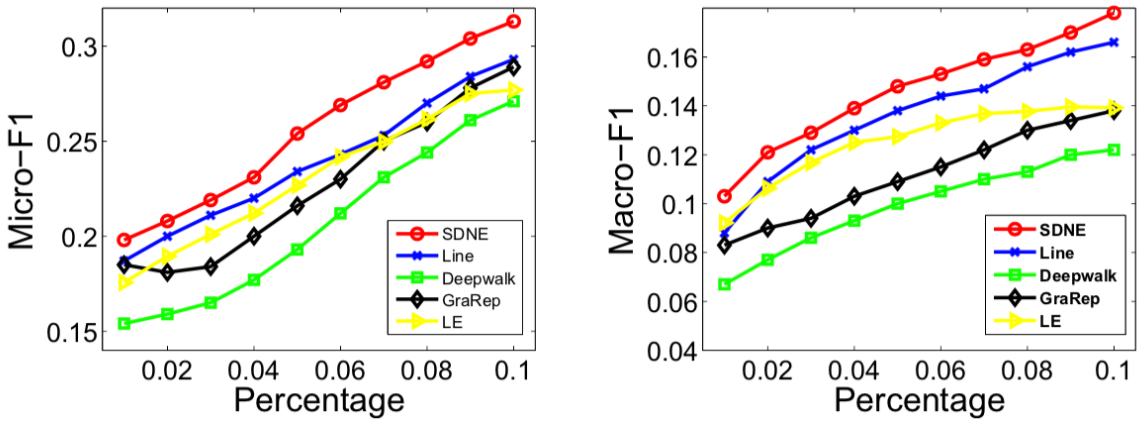
BLOGCATALOG



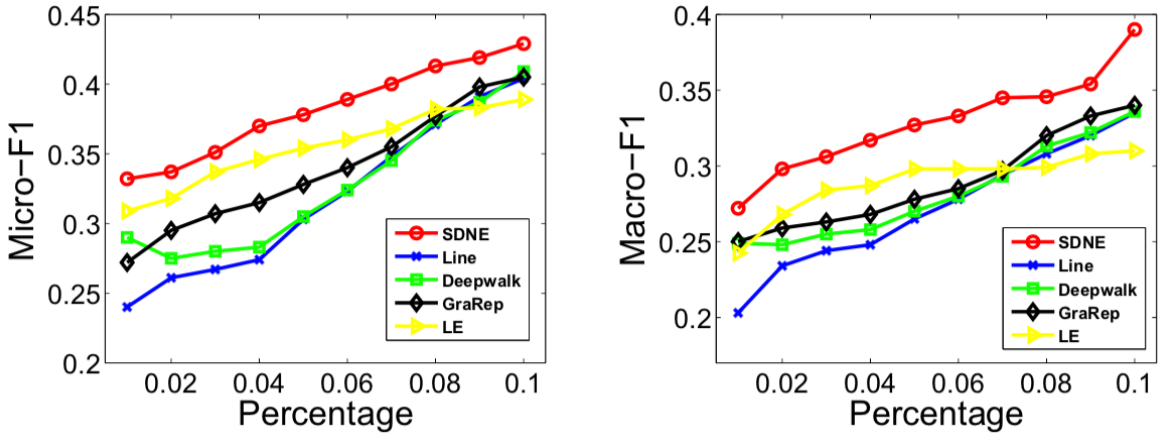
For BLOGCATALOG, the training/test ratio is increased from 10% to 90%

Node Classification

FLICKR



YOUTUBE

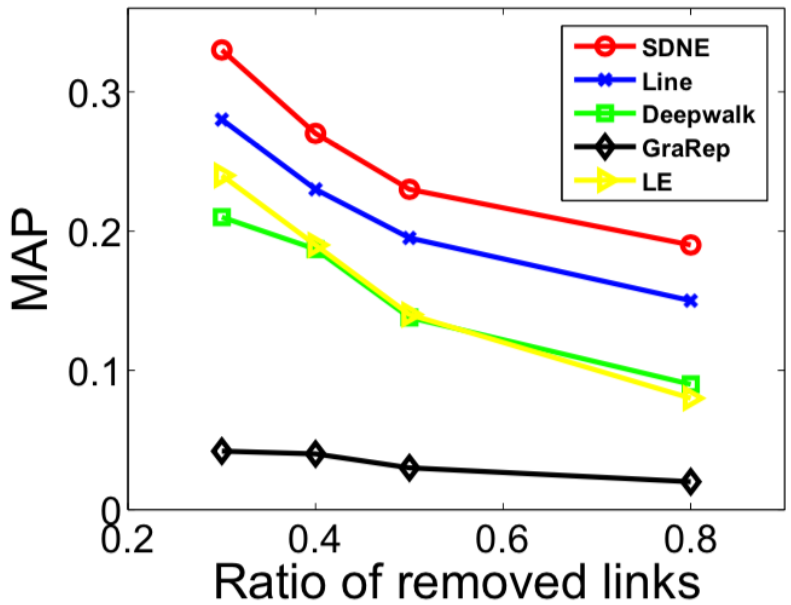


For FLICKR and YOUTUBE, the training/test ratio is increased from 1% to 10%

Link Prediction

Followed procedure:

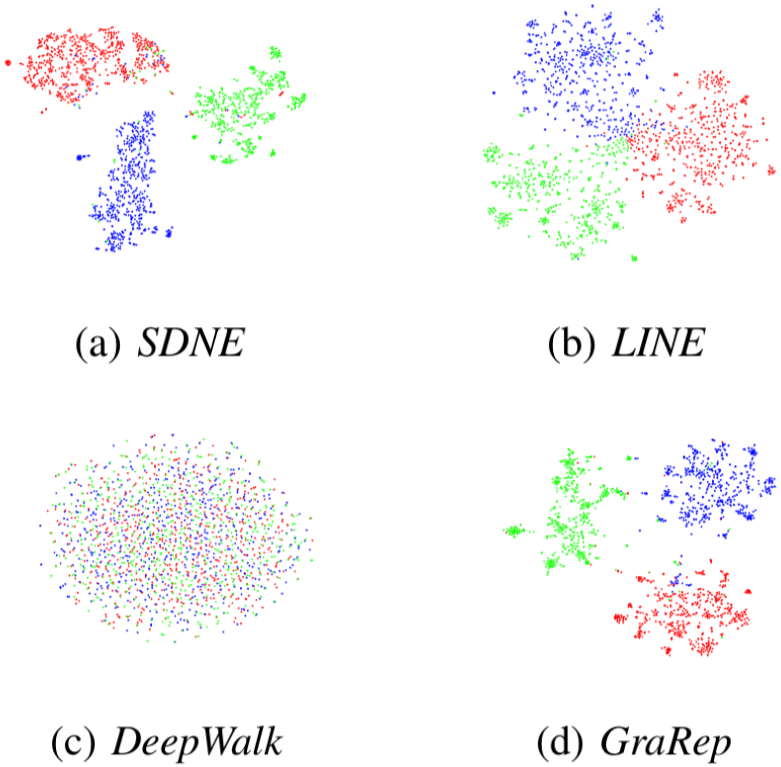
- Remove a portion of ARXIV GR-QC's edges
- Use the emerging network to learn node embeddings
- Predict missing links



Visualization

Visualization of 20-NEWSGROUP

- Each point indicates one document
- Color of a point indicates the category of the document



(a) *SDNE* (b) *LINE*

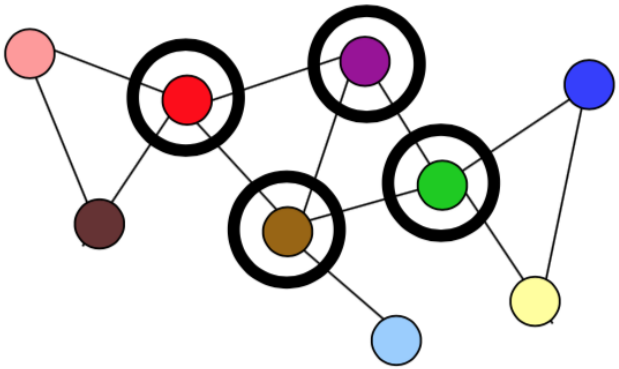
(c) *DeepWalk* (d) *GraRep*

Outline

1. Learning Node Representations
   - Introduction
   - Unsupervised Methods
     - Proximity-based Approaches
     - Structural Equivalence-based Approaches
   - Supervised Methods

Structural Identity

- Nodes in networks have specific roles
  - e. g., individuals, web pages, proteins, etc
- Structural identity
  - identification of nodes based on network structure (no other attribute)
  - often related to role played by node
- Automorphism: strong structural equivalence



Red, Green: structurally identical
Purple, Brown: structurally similar

struc2vec

- Learns node representations based on structural identity
  - structurally similar nodes close in space

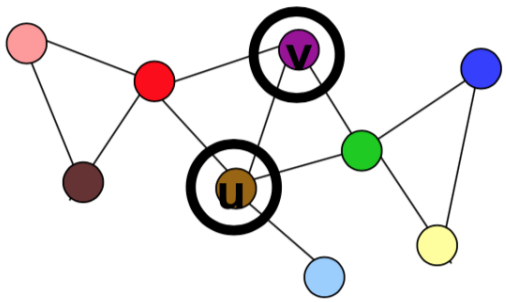
Key ideas:

- Structural similarity does not depend on hop distance
  - neighbor nodes can be different, far away nodes can be similar
- Structural identity as a hierarchical concept
  - depth of similarity varies
- Flexible four step procedure
  - operational aspect of steps are flexible

[Ribeiro et al., KDD'17]

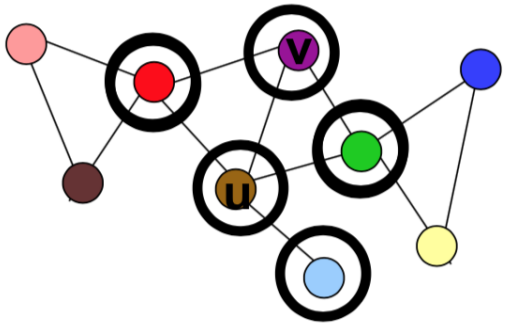
Step 1: Structural Similarity

- Hierarchical measure for structural similarity between two nodes
- $R_k(v)$: set of nodes at distance k from v (ring)
- $s(S)$: ordered degree sequence of set S



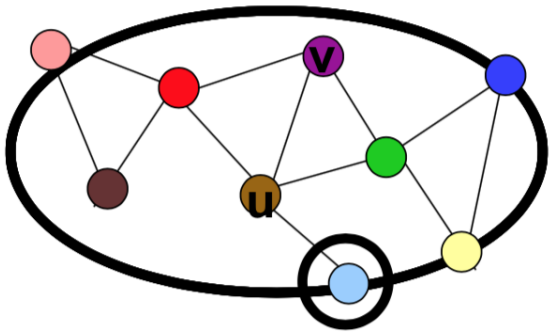
$s(R_0(u)) = 4$

$s(R_0(v)) = 3$



$s(R_1(u)) = 1, 3, 4, 4$

$s(R_1(v)) = 4, 4, 4$



$s(R_2(u)) = 2, 2, 2, 2$

$s(R_2(v)) = 1, 2, 2, 2, 2$

Step 1: Structural Similarity

- $g(D_1, D_2)$: distance between two ordered sequences
  - cost of pairwise alignment: $\max(a,b)/\min(a,b) - 1$
  - optimal alignment by Dynamic Time Warping in our framework

$s(R_0(u)) = 4$ $\quad$ $s(R_1(u)) = 1, 3, 4, 4$ $\quad$ $s(R_2(u)) = 2, 2, 2, 2$

$s(R_0(v)) = 3$ $\quad$ $s(R_1(v)) = 4, 4, 4$ $\quad$ $s(R_2(v)) = 1, 2, 2, 2, 2$

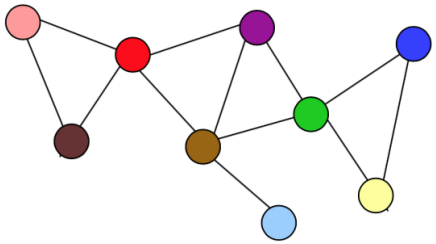
$g(\cdot, \cdot) = 0.33$ $\quad$ $g(\cdot, \cdot) = 3.33$ $\quad$ $g(\cdot, \cdot) = 1$

- $f_k(v, u)$: structural distance between nodes v and u considering first k rings
  - $f_k(v, u) = f_{k-1}(v, u) + g(s(R_k(v)), s(R_k(u)))$

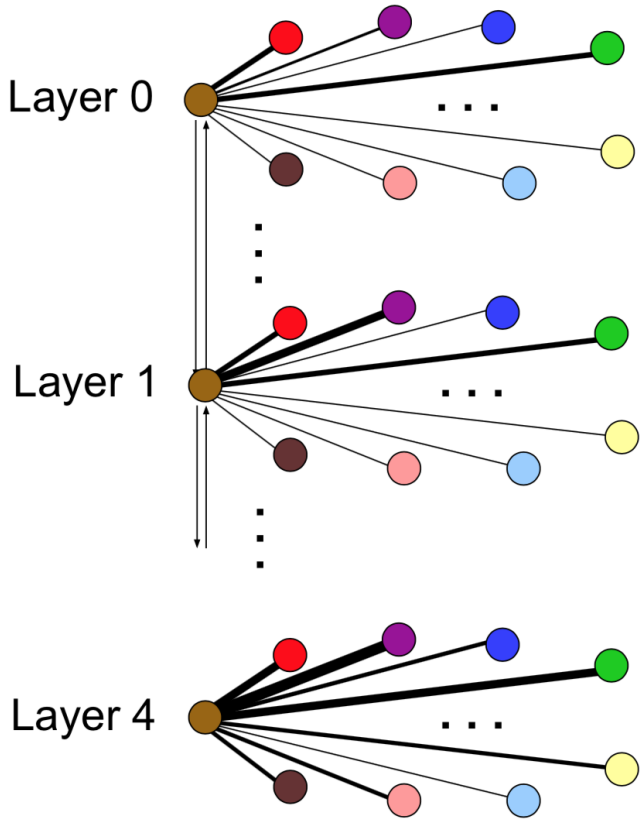
$f_0(v, u) = 0.33$ $\quad$ $f_1(v, u) = 3.66$ $\quad$ $f_2(v, u) = 4.66$

Step 2: Multi-layer graph

- Encodes structural similarity between all node pairs



- Each layer is a weighted complete graph
  - corresponds to similarity hierarchies
- Edge weights in layer k
  - $w_k(v, u) = e^{-f_k(v,u)}$
- Connect corresponding nodes in adjacent layers

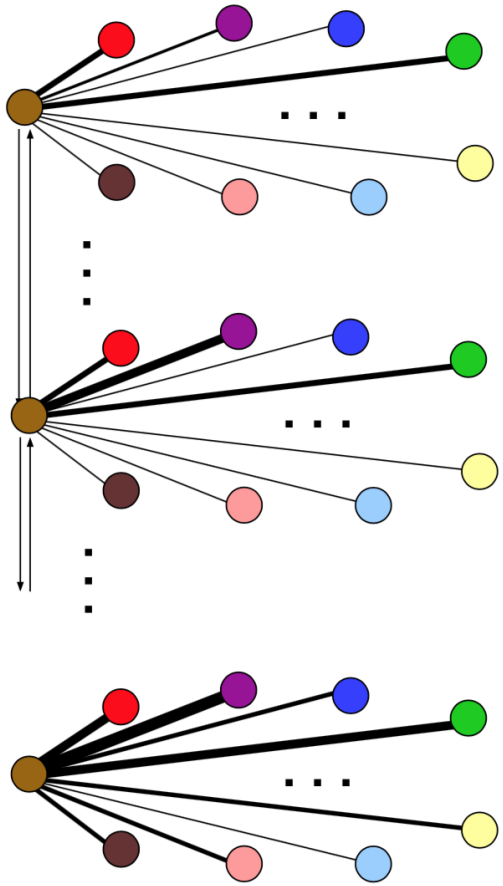


Step 3: Generate Context

- Context generated by biased random walk
  - walking on multi-layer graph
- Walk in current layer with probability p
  - choose neighbor according to edge weight
  - RW prefers more similar nodes
- Change layer with probability $1 - p$
  - jump to the corresponding node
  - choose up/down according to edge weight
  - RW prefers layer with less similar neighbors

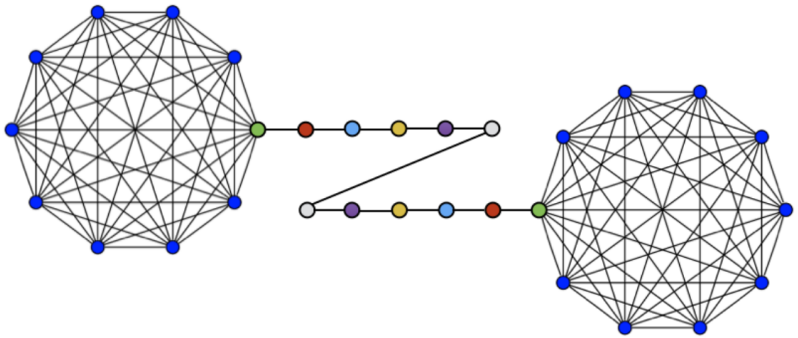
Step 3: Learn Representation

- For each node, generate set of independent and relative short random walks
  - context for node $\rightarrow$ sentences of a language

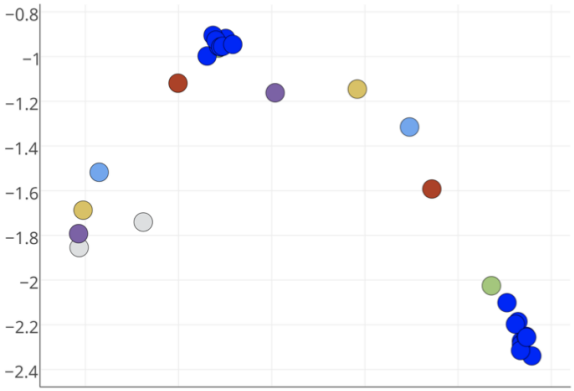


- Train a neural network to learn latent representation for nodes
  - maximize probability of nodes within context
  - Skip-gram (Hierarchical Softmax) adopted

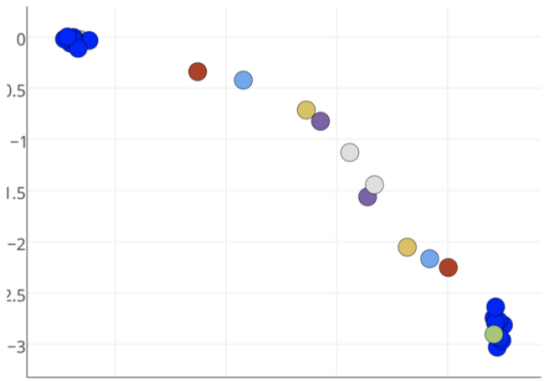
Barbell Network



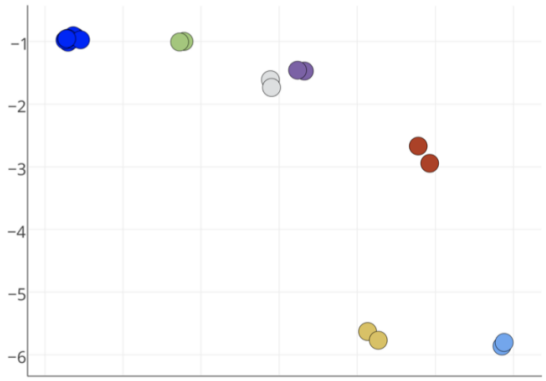
(a) Barbell Graph B(10, 10)



(c) *DeepWalk*



(d) *node2vec*



(e) *struc2vec*

- struc2vec embeds isomorphic nodes very close to each other in space

SEGK: Structural Node Embeddings using Graph Kernels

Another algorithm for learning node representations based on structural identity

- structurally similar nodes close in space

Main idea: The task of learning structural node representations involves comparing the structure of the neighborhoods of nodes

- can use existing algorithms to compare the neighborhoods

SEGK:

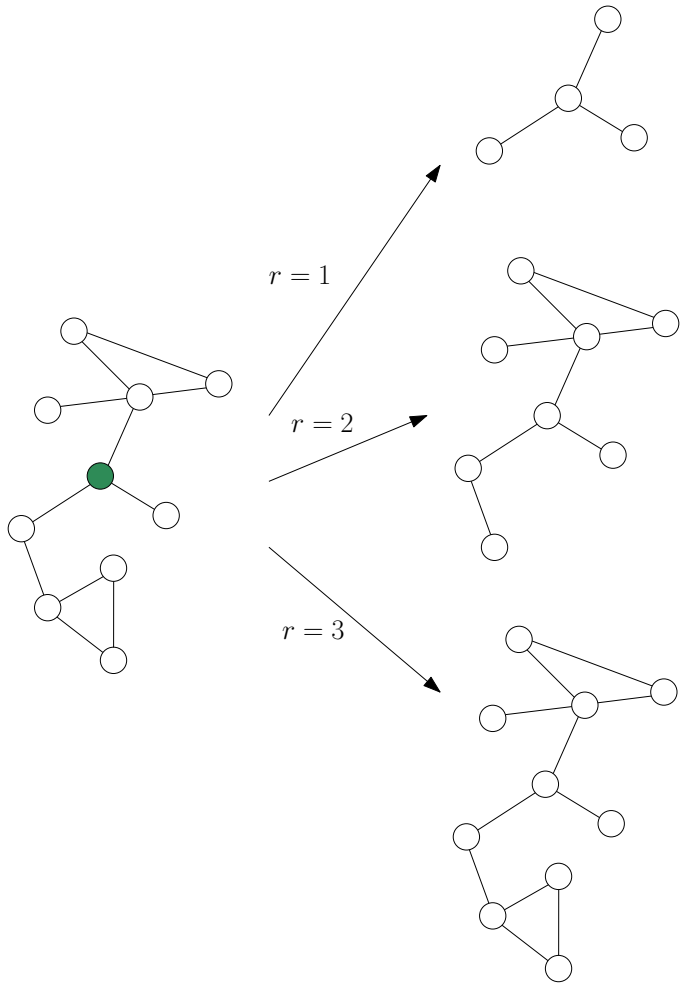
- uses graph kernels to compare nodes' neighborhoods
- builds a kernel matrix that incorporates structural similarity between nodes
- generates structural node representations by decomposing that matrix

[Nikolentzos and Vazirgiannis, TKDE]

Neighborhood Extraction and Labeling

Extracts the $1, 2, \ldots, R$-hop neighborhood of each node

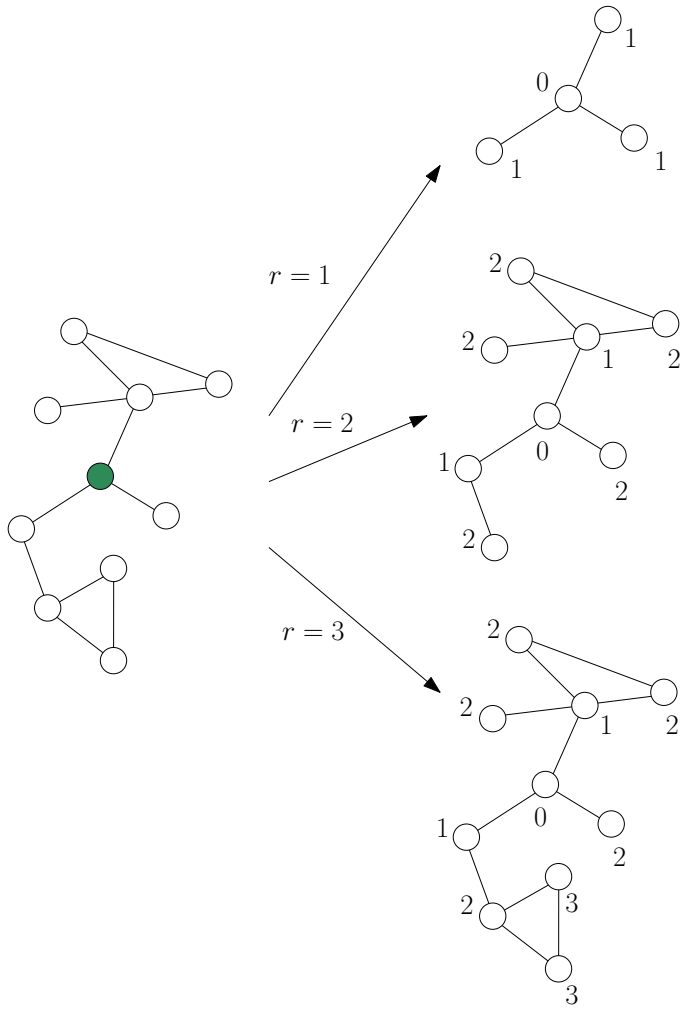
Example: Extraction of the 1-hop, 2-hop, and 3-hop neighborhoods of the **green** node



Neighborhood Extraction and Labeling

Assigns a label to each node of the r-hop neighborhood
$\hookrightarrow$ label equal to the shortest path distance from root

Exaple of assignment of labels to the nodes of the 3 neighborhood subgraphs



Similarity Computation

Uses graph kernels that can handle node labels to compare neighborhood subgraphs to each other

- Let $\{G_i^1, G_i^2, \ldots, G_i^R\}$ and $\{G_j^1, G_j^2, \ldots, G_j^R\}$ be the $1, 2, \ldots, R$-hop neighborhoods of two nodes v_i and v_j
- Then, SEGK compares two nodes by computing the following kernel:

$$k(v_i, v_j) = \sum_{r=1}^{R} \hat{k}_G(G_i^r, G_j^r)\, \hat{k}_G(G_i^{r-1}, G_j^{r-1})$$

where $\hat{k}_G(G_i^0, G_j^0) = 1$ and $\hat{k}_G$ is a normalized kernel between graphs k_G:

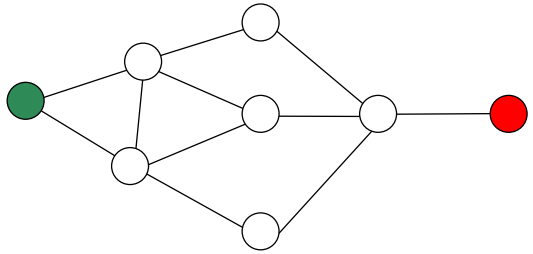
$$\hat{k}_G(G_i, G_j) = \frac{k_G(G_i, G_j)}{\sqrt{k_G(G_i, G_i)\, k_G(G_j, G_j)}}$$

SEGK puts more emphasis on local neighborhoods than on more distant ones:

- For any r and nodes v_i, v_j, it holds that $0 \leq \hat{k}_G(G_i^r, G_j^r) \leq 1$
- Product inside the sum no greater than the minimum of the two kernels

Example

Computing the kernel/similarity between the **green** and red nodes based on their 1-hop and 2-hop neighborhoods



$$k(\bullet, \bullet) = \hat{k}_G(\ \ ,\ \ ) + \hat{k}_G(\ \ ,\ \ ) \cdot \hat{k}_G(\ \ ,\ \ )$$

Embedding Generation

After constructing the kernel matrix $\mathrm{K} \in \mathbb{R}^{n \times n}$ (where n is the number of nodes of the graph), we can generate structural node embeddings by factorizing it:

$$\mathrm{K} = \mathrm{Q}\,\mathrm{Q}^{\top}$$

Then, the i^{th} row of Q corresponds to the embedding of the i^{th} node

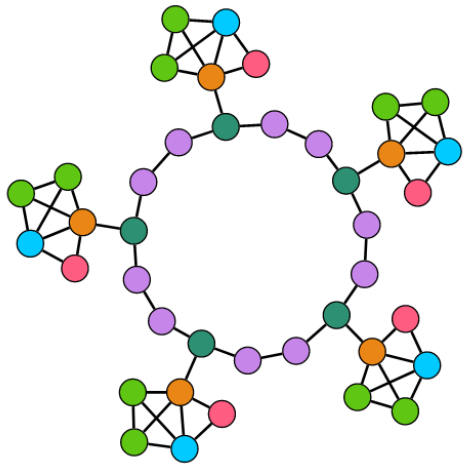
In case n is very large (i. e. hundreds of thousands, millions or billions) computing matrix K:

- can be very inefficient
- can be prohibitive in terms of the required memory

To avoid explicitly constructing the kernel matrix, SEGK resorts to low-rank approximation algorithms
$\hookrightarrow$ e. g., the Nyström method allows us to obtain $\mathrm{Q} \in \mathbb{R}^{n \times d}$ (with $d \ll n$) such that $\mathrm{K} \approx \mathrm{Q}\,\mathrm{Q}^{\top}$

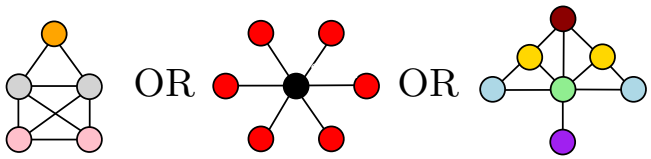
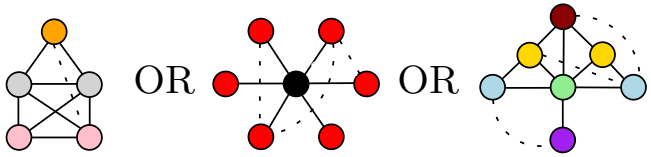
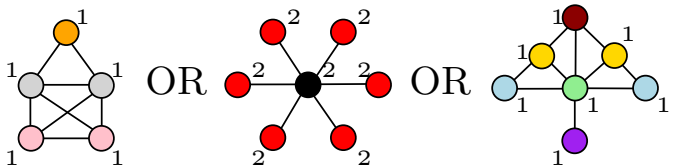
Synthetic Node Classification Dataset

- Generated synthetic graphs with planted structural equivalences
- Structurally equivalent nodes are assigned the same class labels
- generated graphs consist of a cycle of length 40 and some basic shapes ("house", "fan", "star") which are regularly placed along the cycle
- "basic" setup: 10 instances of only one shape are placed along the cycle
- "varied" setup: 10 of each one of the 3 shapes are placed along the cycle

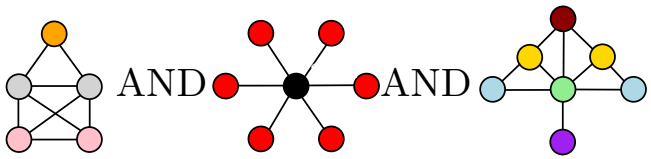
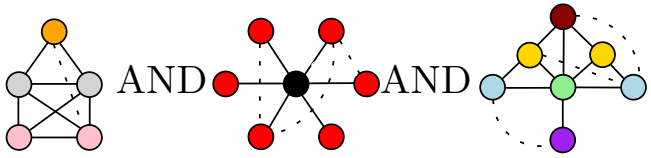
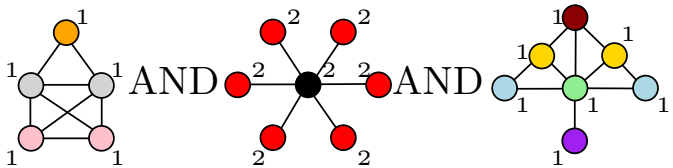


- "basic perturbed" and "varied perturbed": noisy scenarios where edges are added uniformly at random on the generated graphs
  ↪ Number of added edges: 10% of the edges of the graph
- "basic labeled" and "varied labeled": the nodes are assigned node labels
  ↪ Two nodes are assigned the same class labels if they are structurally equivalent and have the same label

Node Classification Results (1/2)

Configuration	Shapes placed along a cycle	Method	Accuracy	F1-score
basic		DeepWalk	0.442	0.295
		RolX	**1.000**	**1.000**
		struc2vec	0.784	0.708
		DRNE	0.987	0.980
		GraphWave	0.995	0.993
		SEGK-SP	**1.000**	**1.000**
		SEGK-WL	**1.000**	**1.000**
		SEGK-GR	**1.000**	**1.000**
basic perturbed		DeepWalk	0.488	0.327
		RolX	0.928	0.886
		struc2vec	0.703	0.632
		DRNE	0.862	0.800
		GraphWave	0.906	0.861
		SEGK-SP	0.941	0.907
		SEGK-WL	0.907	0.850
		SEGK-GR	**0.956**	**0.925**
basic labeled		DeepWalk	0.439	0.263
		RolX	0.987	0.974
		struc2vec	0.617	0.470
		DRNE	0.697	0.547
		GraphWave	0.768	0.608
		SEGK-SP	**0.990**	**0.984**
		SEGK-WL	**0.990**	**0.984**
		SEGK-GR	0.894	0.855

Node Classification Results (2/2)

Configuration	Shapes placed along a cycle	Method	Accuracy	F1-score
varied		DeepWalk	0.329	0.139
		RolX	**0.998**	**0.996**
		struc2vec	0.738	0.592
		DRNE	0.930	0.876
		GraphWave	0.982	0.965
		SEGK-SP	**0.998**	**0.996**
		SEGK-WL	0.994	0.988
		SEGK-GR	0.937	0.923
varied perturbed		DeepWalk	0.313	0.128
		RolX	0.856	0.768
		struc2vec	0.573	0.412
		DRNE	0.734	0.605
		GraphWave	0.793	0.682
		SEGK-SP	**0.892**	**0.818**
		SEGK-WL	0.876	0.790
		SEGK-GR	0.882	0.817
varied labeled		DeepWalk	0.315	0.137
		RolX	0.940	0.879
		struc2vec	0.524	0.349
		DRNE	0.548	0.424
		GraphWave	0.726	0.547
		SEGK-SP	0.940	0.902
		SEGK-WL	**0.960**	**0.931**
		SEGK-GR	0.783	0.776

Enron Dataset

An e-mail network encoding communication between employees in a company. There are 143 nodes and 2,583 edges:

- Nodes represent Enron employees
- Edges correspond to e-mail communication between the employees

We expect structural equivalences in job titles due to corporate organizational hierarchy:

- An employee has one of 7 functions in the company (e. g., CEO, manager)
- These functions provide ground-truth information about roles of the corresponding nodes in the network

Enron Dataset

Method	Homogeneity	Completeness	Silhouette	Accuracy	F1-score
DeepWalk	0.240	0.081	0.214	0.324	0.202
RolX	0.178	**0.141**	0.040	0.264	0.154
struc2vec	0.243	0.122	0.246	0.323	0.190
DRNE	**0.344**	0.112	**0.420**	0.201	0.111
GraphWave	0.203	0.092	0.249	0.257	0.149
SEGK-SP	0.227	0.064	0.011	0.264	0.151
SEGK-WL	0.291	0.064	0.283	**0.360**	**0.222**
SEGK-GR	0.144	0.088	0.127	0.294	0.172

Table: Performance of the baselines and and the proposed SEGK instances for learning structural embeddings on the Enron dataset.

Outline

1. Learning Node Representations
   - Introduction
   - Unsupervised Methods
     - Proximity-based Approaches
     - Structural Equivalence-based Approaches
   - Supervised Methods

GCN

Given the adjacency matrix A of a graph, GCN first computes:

$$\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$

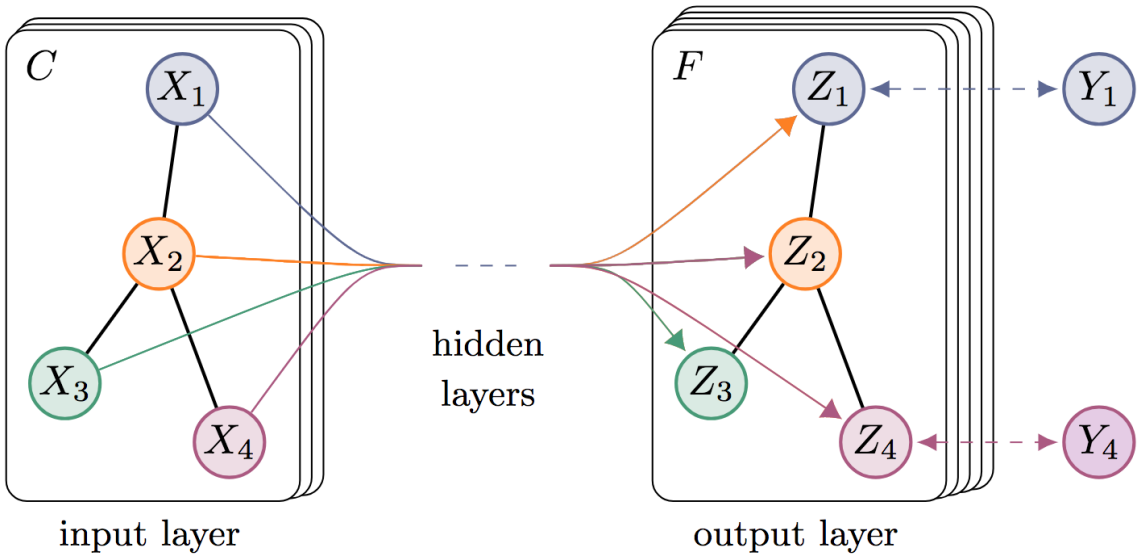
where
$\tilde{A} = A + I$
$\tilde{D}$: a diagonal matrix such that $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

Then, the output of the model is:

$$Z = softmax(\hat{A}\ ReLU(\hat{A}\ X\ W^0)\ W^1)$$

where
X: matrix whose rows contain the attributes of the nodes
W^0, W^1: trainable weight matrices

[Kipf and Welling, ICLR'17]



To learn node embeddings, GCN minimizes the following loss function:

$$\mathcal{L} = -\sum_{i \in I} \sum_{j=1}^{|\mathcal{C}|} \mathsf{Y}_{ij} \log \mathsf{Z}_{ij}$$

I: indices of the nodes of the training set
$\mathcal{C}$: set of class labels

Experimental Evaluation

Experimental comparison conducted in [1]

Compared algorithms:

- DeepWalk
- ICA [2]
- Planetoid
- GCN

Task: node classification

[Kipf and Welling, ICLR'17]
[Lu and Getoor, ICML'03]

Datasets

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Label rate: number of labeled nodes that are used for training divided by the total number of nodes

Citation network datasets:

- nodes are documents and edges are citation links
- each node has an attribute (the bag-of-words representation of its abstract)

NELL is a bipartite graph dataset extracted from a knowledge graph

Results

Classification accuracies of the 4 methods

Method	Citeseer	Cora	Pubmed	NELL
DeepWalk	43.2	67.2	65.3	58.1
ICA	69.1	75.1	73.9	23.1
Planetoid	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)

Observation: DeepWalk $\rightarrow$ unsupervised learning of embeddings

$\hookrightarrow$ fails to compete against the supervised approaches