


# Deep Learning for Graphs - I

Michalis Vazirgiannis and Giannis Nikolentzos

 DaSciM, LIX, École Polytechnique

January, 2022

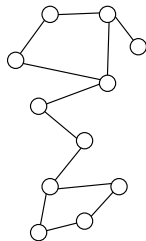
- 1 Learning Node Representations
  - Introduction
  - Unsupervised Methods
    - Proximity-based Approaches
    - Structural Equivalence-based Approaches
  - Supervised Methods

## 1 Learning Node Representations

- Introduction
- Unsupervised Methods
  - Proximity-based Approaches
  - Structural Equivalence-based Approaches
- Supervised Methods

# Traditional Node Representation

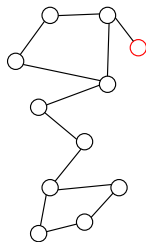
Representation: row of adjacency matrix



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

# Traditional Node Representation

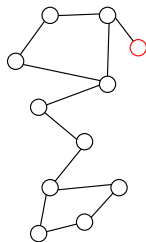
Representation: row of adjacency matrix



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

# Traditional Node Representation

Representation: row of adjacency matrix



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

However, such a representation suffers from:

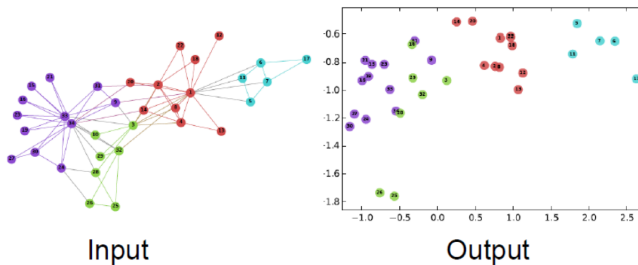
- data sparsity
- high dimensionality

⋮

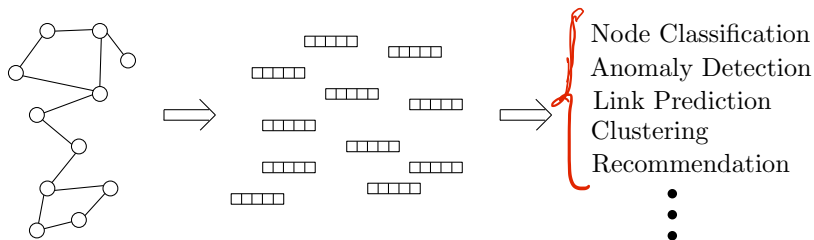
# Node Embedding Methods

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



# Why Learning Node Representations?



Examples:

- Recommend friends
- Detect malicious users



- 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:

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

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

Most methods belong to the following groups:

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

## 1 Learning Node Representations

- Introduction
- Unsupervised Methods
  - Proximity-based Approaches
  - Structural Equivalence-based Approaches
- Supervised Methods

## 1 Learning Node Representations

- Introduction
- Unsupervised Methods
  - Proximity-based Approaches
  - Structural Equivalence-based Approaches
- Supervised 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 proximity
- > second-order proximity
- > third-order proximity
- $\vdots$

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 proximity
- > second-order proximity
- > third-order proximity
- $\vdots$

## 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)

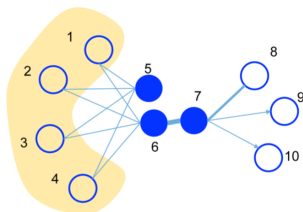
The second-order proximity captures the 2-step relations between two vertices  $v$  and  $u$ . It describes the proximity of the neighborhood structures of  $v$  and  $u$ , and is thus determined by the number of common neighbors shared by the two vertices.

## 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  $u$ .

**First-order proximity:** observed links in the network

**Second-order proximity:** shared neighborhood structures



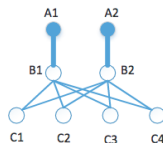
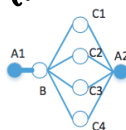
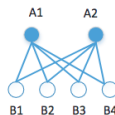
- Vertices 6 and 7 have a high *first-order proximity* since they are connected through a strong tie → they should be placed closely in the embedding space
- Vertices 5 and 6 have a high *second-order proximity* since they share similar neighbors → they should also be placed closely



# Proximities

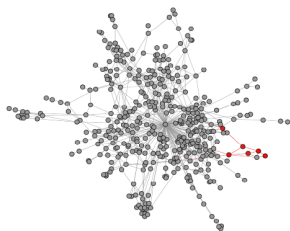
$k$ -order proximities for  $k = 1, \dots, 4$

many roads of length 2  
from  $A_1$  to  $A_2$



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

Inspired by recent advances in language modeling



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

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

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

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

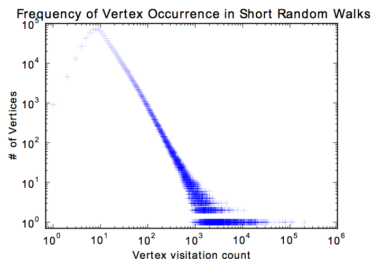
$v_4 \rightarrow v_3 \rightarrow v_1 \rightarrow v_5 \rightarrow v_1 \rightarrow v_{12} \rightarrow v_{10}$

$\vdots$

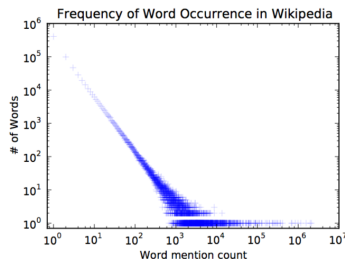
- Simulates a series of short random walks pre-fixed length  
create context of vertex,

[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

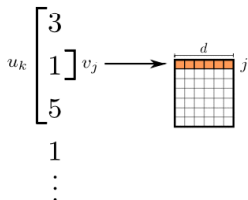
[Perozzi et al., KDD'14]

Skipgram 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:

$$\mathcal{W}_{v_4} = 4$$



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

This yields the optimization problem:

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

$v_i$ : central vertex

$v_{i-w}, \dots, v_{i+w}$ : neighbors of central vertex

$f(v)$ : embedding of vertex  $v$

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

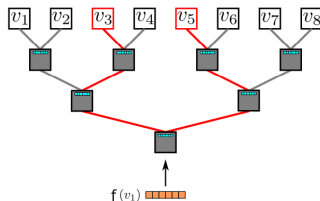
$$\underset{f}{\text{minimize}} \quad -\frac{1}{T} \sum_{i=1}^T \sum_{\substack{j=i-w \\ j \neq i}}^{i+w} \log P(v_j | f(v_i))$$

- 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  $(b_0, b_1, \dots, b_{\lceil \log |V| \rceil})$  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

Like DeepWalk, node2vec is also a random walk based method

DeepWalk uses a *rigid* search strategy

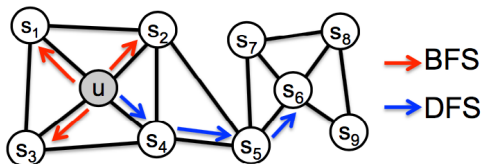
Conversely, node2vec simulates a family of biased random walks which

- 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

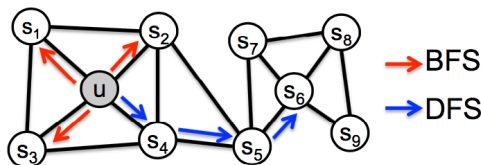


Goal: Given a source node  $u$ , sample its neighborhood  $\mathcal{N}(u)$  where  $|\mathcal{N}(u)| = k$



# Two Extreme Sampling Strategies

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

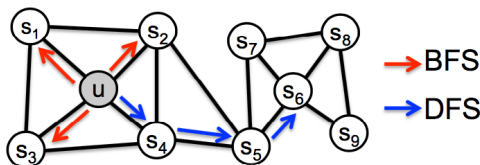


In most applications, we are interested in two kinds of similarities between vertices:

- 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  $l$

$$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \dots \rightarrow v_l$$

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

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

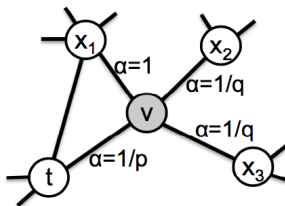
where  $\pi_{vx}$  is the unnormalized transition probability between  $v$  and  $x$ , and  $Z$  is a normalizing factor

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  $v$



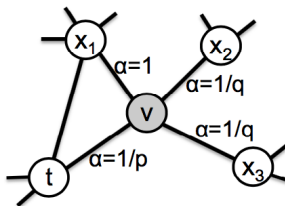
The unnormalized transition probability is  $\pi_{vx} = w_{vx} \alpha_{pq}(t, x)$ , where:

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

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  $v$

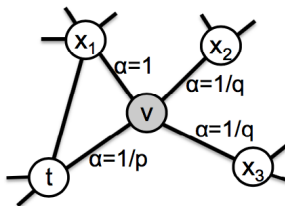


The *return parameter*  $p$  controls the likelihood of immediately re-visiting a node in the walk

- 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:

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

function of inner product  
between  $f(u)$  and  $f(v)$

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**

## GraRep

- constructs transition matrices
- applies matrix factorization to generate node embeddings

### k-step Transition Probabilities

Let  $S$  be the adjacency matrix of a graph, and  $D$  the diagonal degree matrix:

$$D_{ij} = \begin{cases} \sum_p S_{ip} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Then, the 1-step probability transition matrix is defined as:  $D_{ij}^{-1} = \text{diag}(\frac{1}{\text{degree}_1}, \dots)$

$$A = D^{-1}S$$

and then the  $k$ -step probability transition matrix is defined as:

$$A^k = \underbrace{A \cdots A}_k \quad i \xrightarrow{k\text{-step}} j$$

Let  $P_k(v_j|v_i)$  denote the probability for a transition from  $v_i$  to  $v_j$  in exactly  $k$  steps. Then,

$$P_k(v_j|v_i) = A_{ij}^k$$

[Cao et al., CIKM'15]



For a given  $k$ , the loss function of GraRep is: *learned similarity.*

$$\underset{f, f'}{\text{minimize}} \quad - \sum_{v_i \in V} \left( \sum_{v_j \in V} P_k(v_j | v_i) \log \sigma(f'(v_j)^\top f(v_i)) \right) + \lambda \mathbb{E}_{v_c \sim P_k(V)} [\log \sigma(-f'(v_c)^\top f(v_i))] )$$

$\lambda$ : a hyper-parameter indicating the number of negative samples

$P_k(V)$ : distribution over the vertices

$f(v)$ : central node embedding

$f'(v)$ : embedding as context

Given a specific starting vertex  $v_i$  and ending vertex  $v_j$ , the local loss over that pair is defined as:

$$L_k(v_i, v_j) = -P_k(v_j | v_i) \log \sigma(f'(v_j)^\top f(v_i)) - \lambda P_k(v_j) \log \sigma(-f'(v_j)^\top f(v_i))$$

and  $P_k(v_j)$  can be computed as:

$$P_k(v_j) = \frac{1}{|V|} \sum_{v_i \in V} A_{ij}^k$$

This leads to:

$$L_k(v_i, v_j) = -A_{ij}^k \log \sigma(f'(v_j)^\top f(v_i)) - \frac{\lambda}{|V|} \sum_{v_l \in V} A_{lj}^k \log \sigma(-f'(v_j)^\top f(v_l))$$

By defining  $e = f(v_i)^\top f'(v_j)$  and setting  $\frac{\partial L_k}{\partial e} = 0$ , we get:

$$Y_{ij}^k = f(v_i)^\top f'(v_j) = W_i^k C_j^k = \log \left( \frac{A_{ij}^k}{\sum_{v_l \in V} A_{lj}^k} \right) - \log \left( \frac{\lambda}{|V|} \right)$$

Hence, optimizing the proposed loss essentially involves a matrix factorization problem

To reduce noise, GraRep replaces all negative entries in  $Y^k$  with 0:

$$X_{ij}^k = \max(Y_{ij}^k, 0)$$

And then decomposes  $X^k$  using SVD:

$$X^k = U^k \Sigma^k (V^k)^\top$$

$X_{d,ij}^k$  = similarity / adjacency among nodes  $i$  and  $j$  connected with

Let  $X_d^k$  be a low-rank approximation of  $X^k$  (by keeping the top  $d$  path singular values). Then, length  $k$ .

$$X^k \approx X_d^k = U_d^k \Sigma_d^k (V_d^k)^\top = W^k C^k$$

where

$$W^k = U_d^k (\Sigma_d^k)^{\frac{1}{2}}$$

$$C^k = (\Sigma_d^k)^{\frac{1}{2}} (V_d^k)^\top$$

To capture high-order proximities between vertices, GraRep:

- computes the  $k$ -step transition probability matrix  $A^k$  for each  $k = 1, 2, \dots, K$
- computes each  $k$ -step representation
- concatenates all  $k$ -step representations

**Main disadvantage:** by setting  $K$  to large values, GraRep fails to efficiently scale to large networks

Most real-world networks are very complex

Shallow models

- cannot capture the highly non-linear network structure
- generate sub-optimal node representations

SDNE is a *deep* model which

- has multiple layers of non-linear functions
- preserves the first-order and second-order proximities

[Wang et al., KDD'16]

To preserve the second-order proximity, SDNE employs a deep autoencoder

Given an input  $x_i$  ( $i^{th}$  row of adjacency matrix), the hidden representations at layers  $1, \dots, k$  are:

$$\begin{aligned} y_i^{(1)} &= \sigma(W^{(1)}x_i + b^{(1)}) \\ y_i^{(k)} &= \sigma(W^{(k)}y_i^{(k-1)} + b^{(k)}) \end{aligned}$$

where  $\sigma$  is a non-linear activation function (e.g., sigmoid function)

After obtaining  $y_i^{(k)}$  (node  $i$ 's embedding), we compute the reconstructed input  $\hat{x}_i$  by reversing the above calculation process

The objective function is then:

$$\mathcal{L}_{2nd} = \sum_{i=1}^n ||(\hat{x}_i - x_i) \odot b_i||_2^2$$

*element-wise*

where  $\odot$  is the Hadamard product,  $b_{ij} = 1$  if nodes  $i$  and  $j$  are not connected by an edge, and  $b_{ij} > 1$  otherwise

Vertices that have similar neighborhoods are mapped close to each other in the embedding space

To capture the first-order proximity, SDNE borrows the idea of Laplacian Eigenmaps:

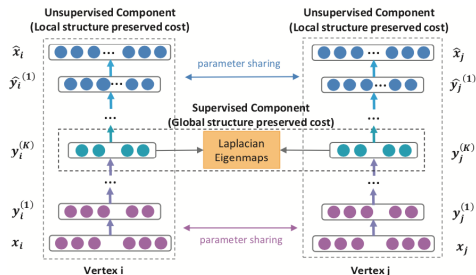
$$\mathcal{L}_{1st} = \sum_{i=1}^n \sum_{j=1}^n x_{ij} \|y_i^{(k)} - y_j^{(k)}\|_2^2$$

Vertices linked by edges with high weights are thus mapped close to each other

SDNE then jointly minimizes the following objective function:

$$\mathcal{L} = \mathcal{L}_{2nd} + \alpha \mathcal{L}_{1st} + \nu \mathcal{L}_{reg}$$

where  $\mathcal{L}_{reg}$  is an  $l_2$ -norm regularizer term to prevent overfitting



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 comparison conducted in [1]

Compared algorithms:

- DeepWalk
- GraRep
- SDNE *most performant.*
- LINE
- Laplacian Eigenmaps (LE)

[Wang et al., KDD'16]

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

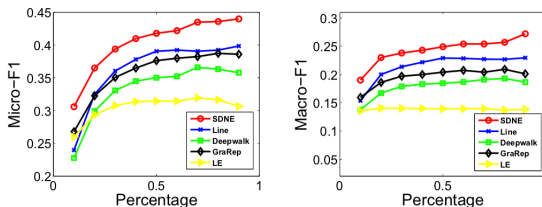
Three real-world applications

- node classification
- link prediction
- visualization

# Node Classification

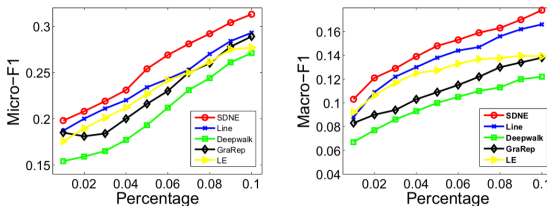
Vertex representations generated from node embedding methods and given as input to a logistic regression classifier to predict a set of labels for each vertex

## BLOGCATALOG

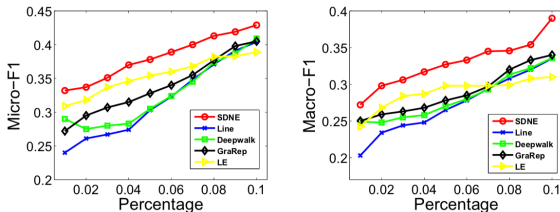


For BLOGCATALOG, the training/test ratio is increased from 10% to 90%

## FLICKR



## YOUTUBE

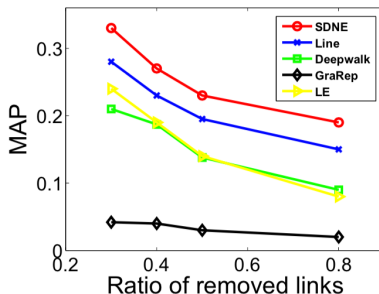


For FLICKR and YOUTUBE, the training/test ratio is increased from 1% to 10%

# Link Prediction

Followed procedure:

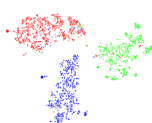
- Remove a portion of ARXIV GR-QC's edges
- Use the emerging network to learn node embeddings
- Predict missing links



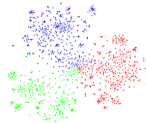


## Visualization of 20-NEWSGROUP

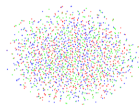
- Each point indicates one document (*one node*)
- Color of a point indicates the category of the document



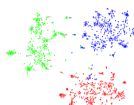
(a) *SDNE*



(b) *LINE*



(c) *DeepWalk*



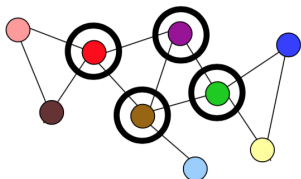
(d) *GraRep*

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

# Structural Identity

## 基于结构的 Node embedding

- 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

# RolX: Structural Role Extraction & Mining in Large Graphs

An unsupervised learning approach for **automatically extracting structural roles** from networks

## Key Ideas:

- Automatic feature extraction, based exclusively on the graph structure
- Assignment of a mixed-membership of roles to each node
- Feature grouping and role extraction in **linear** time on the number of edges.

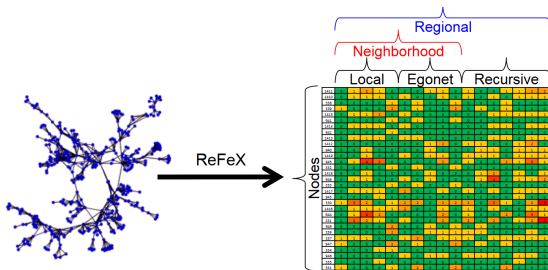
## Applications:

- Transfer learning: The structural roles generalize across disjoint networks
- Structural Similarity of networks by comparison of the role distributions
- Sense-making: Structural roles highlight different contextual roles

[Henderson et al., KDD'12]

# Step 1: Feature Extraction

- ReFeX turns network connectivity into **recursive** structural features
  - Local and egonet features seed the recursive ReFeX process.
  - Local features: measures of the node degree
  - Egonet features: information of the induced subgraph of each node, i.e. neighbors and edges occurring in the subgraph



[Henderson et al., KDD'11]

## Step 2: Embedding Generation

- To identify the roles that each node plays in the network, RoIX performs soft clustering in the structural feature space
- Let  $V$  denote the node-feature matrix
- The algorithm computes a low-rank approximation of  $V$ ,  $V \approx GF$
- To compute the low-rank approximation, the algorithm uses

**nonnegative matrix factorization:**

*NMF*

$$\arg \min_{G, F} \|V - GF\|_F, \text{ such that } G \geq 0, F \geq 0$$

$$X_{m \times n} \approx U_{m \times k} \times V_{k \times n}^T$$

- This type of factorization

- is computationally efficient
- simplifies the interpretation of roles and memberships

*heuristic approach:*

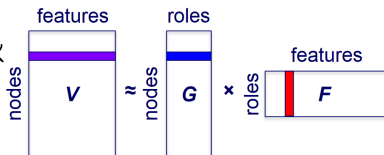
$$k \ll \min(m, n)$$

*choose*

$k$ : such that

- Rows of matrix  $V$  considered as structural embeddings of nodes

*G: as embedding matrix*



$$\frac{\sum_{i=1}^k \sigma_i}{\sum_{i=1}^n \sigma_i} > 0.8$$

$\sigma_i$ : singular values.

What is the best choice of rank  $r$  of the approximation  $GF$ ?

*optimisation approach:*

- Minimum description length for optimal size  $r$  of model
  - $L$ : description length
  - $M$ : number of bits required to describe the model
  - $E$ : description cost of the reconstruction error in  $V - GF$
  - **Goal**: Minimize  $L = M + E$
- Errors in  $V - GF$  are not distributed normally  $\rightarrow$  **KL divergence**

$$E = \sum_{i,j} \left( V_{i,j} \log \frac{V_{i,j}}{(GF)_{i,j}} - V_{i,j} + (GF)_{i,j} \right)$$

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

## Key ideas:

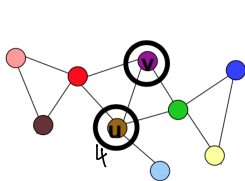
- Structural similarity does not depend on hop distance
  - neighbor nodes can be different, far away nodes can be similar
- Structural identity as a hierarchical concept
  - depth of similarity varies
- Flexible four step procedure
  - operational aspect of steps are flexible

[Ribeiro et al., KDD'17]



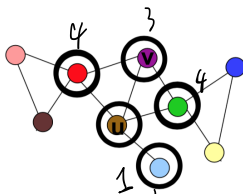
# Step 1: Structural Similarity

- 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  $S$



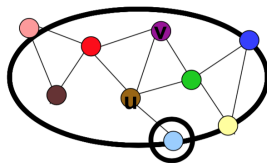
$$s(R_0(u)) = 4$$

$$s(R_0(v)) = 3$$



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

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



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

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

## Step 1: Structural Similarity

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

$$s(R_0(u)) = 4$$

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

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

$$s(R_0(v)) = 3$$

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

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

$$g(\cdot, \cdot) = 0.33$$

$$g(\cdot, \cdot) = 3.33$$

$$g(\cdot, \cdot) = 1$$

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

$$f_0(v, u) = 0.33$$

$$f_1(v, u) = 3.66$$

$$f_2(v, u) = 4.66$$

## Step 2: Multi-layer graph

- Encodes structural similarity between all node pairs



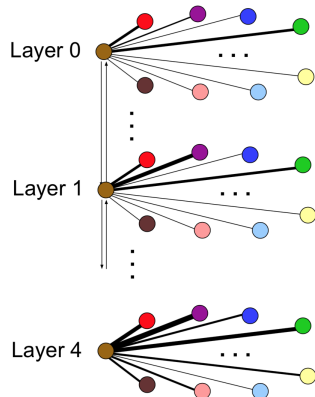
- Each layer is a weighted complete graph

- corresponds to similarity hierarchies

- Edge weights in layer  $k$

- $w_k(v, u) = e^{-f_k(v, u)}$


- Connect corresponding nodes in adjacent layers

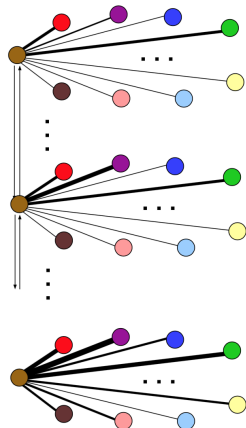


## Step 3: Generate Context

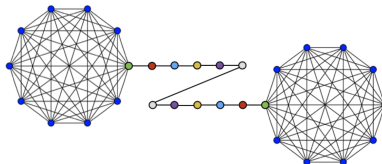
- Context generated by biased random walk
  - walking on multi-layer graph
- Walk in current layer with probability  $p$ 
  - choose neighbor according to edge weight
  - RW prefers more similar nodes
- Change layer with probability  $1 - p$ 
  - jump to the corresponding node
  - choose up/down according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

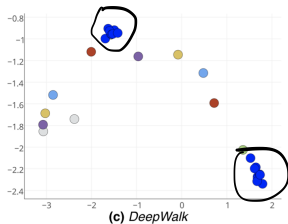


# Barbell Network

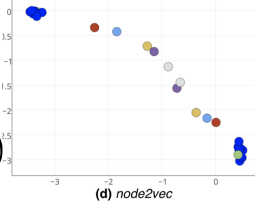


(a) Barbell Graph  $B(10, 10)$

two groups of blue nodes

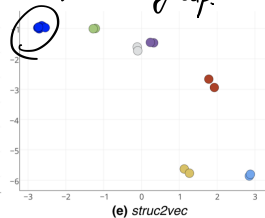


(c) DeepWalk



(d) node2vec

They have similar structure.  
only one group.



(e) struc2vec

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

## Preliminaries:

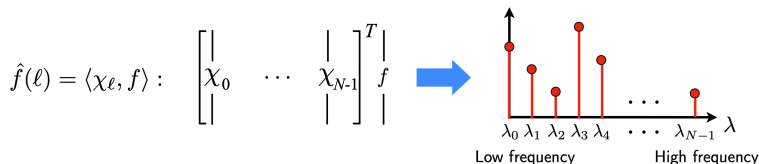
- Let  $L = D - A$  be the *Laplacian* of graph  $G$ , where  $D$  the diagonal degree matrix and  $A$  the adjacency matrix of  $G$ 
  - $L$  is symmetric
  - $L$  is positive-semidefinite and, thus, its eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_N$  are real, non-negative numbers
- Let the eigendecomposition of  $L$ ,  $L = U\Lambda U^T$
- Matrix  $U$  contains the *eigenvectors* of  $L$  and matrix  $\Lambda$  is a diagonal matrix with the *eigenvalues* of  $L$  on the main diagonal

The spectral decomposition of the Laplacian of a graph reveals several structural characteristics of the graph:

- number of components
- sparsest cut
- etc.

# Spectral graph wavelets (1/2)

- The graph *Laplacian*  $L$  satisfies the eigendecomposition:  
 $Lx_\ell = \lambda_\ell x_\ell$
- The eigenvalues  $\lambda_i, i = 1, \dots, n$  correspond to different frequencies in the frequency domain:



- The equivalent of the Fourier transform can be defined in the graph space:

	Real Space	Graph Space
<b>Laplacian Operator</b>	$\frac{d^2}{dx^2}$	$L$
<b>Eigenfunctions</b>	$e^{j\omega x}$	$x_\ell$
<b>Fourier Transform</b>	$\hat{f}(\omega) = \int (e^{j\omega x})^* f(x) dx$	$\hat{f}(\ell) = \sum_{i=1}^n x_\ell^*(i) f(i)$
<b>Inv. Fourier Transform</b>	$f(\omega) = \frac{1}{2\pi} \int e^{j\omega x} \hat{f}(x) dx$	$f(i) = \sum_{\ell=1}^n \hat{f}(\ell) x_\ell(i)$



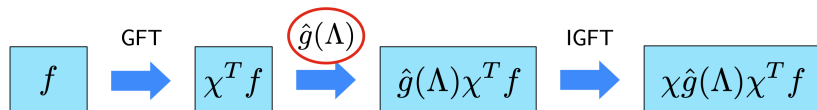
# Spectral graph wavelets (2/2)

**Spectral Graph filtering:** Why do we need filtering?

Filters keep specific frequencies/eigenvalues of the signal!

⇒ Different structural aspects of the graph!!

- Apply filter with transfer function  $\hat{g}(\cdot)$  to a graph signal  
 $f : V \rightarrow \mathbb{R}^n$



$$\hat{g}(\Lambda) = \begin{bmatrix} \hat{g}(\lambda_0) & & 0 \\ & \ddots & \\ 0 & & \hat{g}(\lambda_{N-1}) \end{bmatrix}$$

[Dong et al., IEEE Transactions on Signal Processing]

## Key idea:

- Represent a node's topological signature as a distribution over the coefficients of the heat scaling wavelet centered around the node

## Spectral graph wavelet:

$$\psi_i = U Z U^T \delta_i$$

where  $Z$  a diagonal matrix with  $g_s(\lambda_1), g_s(\lambda_2), \dots, g_s(\lambda_n)$  on the main diagonal,  $g_s(\cdot)$  a scaling wavelet (filter kernel with scaling parameter  $s$ ) and  $\delta_i$  the one-hot vector of node  $v_i$

- heat kernel:  $g_s(\lambda) = e^{-\lambda s}$
- $m^{th}$  wavelet coefficient:  $\psi_{m,i}(s) = \sum_{j=1}^n g_s(\lambda_j) U_{m,j} U_{i,j}$

[Donnat et al., KDD'18]

# GraphWave: The Algorithm

Given a graph  $G = (V, E)$ , a scale  $s$  and evenly-spaced sampling points  $\{t_1, t_2, \dots, t_d\}$ , GraphWave:

- 1 computes the eigenvalue decomposition of the Laplacian of  $G$ :  
 $\Psi = U g_s(\Lambda) U^\top$
- 2 computes:

$$\phi_i(t) = \frac{1}{n} \sum_{j=1}^n \exp^{it\Psi_{ji}}$$

for every node  $v_i \in V$  and for every  $t \in \{t_1, t_2, \dots, t_d\}$

The embedding of a node  $v_i$  is the defined as follows:

$$h_i = [\text{Re}(\phi_i(t_1)), \text{Im}(\phi_i(t_1)), \dots, \text{Re}(\phi_i(t_d)), \text{Im}(\phi_i(t_d))]$$

Hyperparameters:

- scale  $s \rightarrow$  determines the size of the considered neighborhood around each node
- sampling points  $t_1, t_2, \dots, t_d$

# SE GK: 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

SE GK:

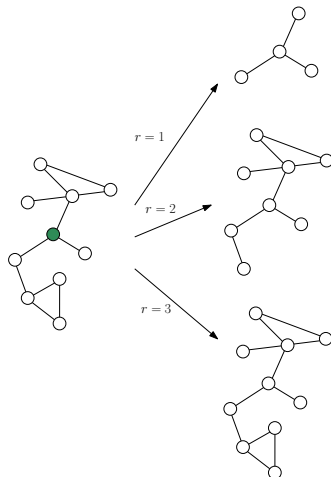
- uses graph kernels to compare nodes' neighborhoods
- builds a kernel matrix that incorporates structural similarity between nodes
- generates structural node representations by decomposing that matrix

[Nikolentzos and Vazirgiannis, TKDE]

# Neighborhood Extraction and Labeling

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

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

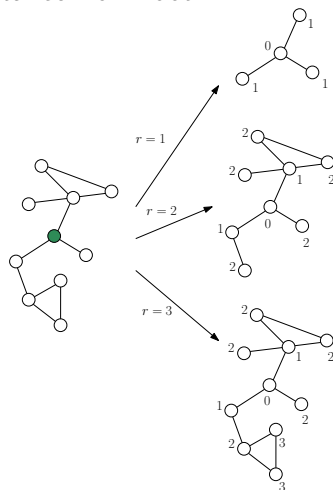


# Neighborhood Extraction and Labeling

Assigns a label to each node of the  $r$ -hop neighborhood

↪ label equal to the shortest path distance from root

Example 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  $\{G_i^1, G_i^2, \dots, G_i^R\}$  and  $\{G_j^1, G_j^2, \dots, G_j^R\}$  be the  $1, 2, \dots, R$ -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  $k_G$ :

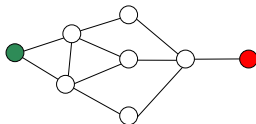
$$\hat{k}_G(G_i, G_j) = \frac{k_G(G_i, G_j)}{\sqrt{k_G(G_i, G_i) k_G(G_j, G_j)}}$$

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



$$k(\text{green}, \text{red}) = \hat{k}_G(\text{green}, \text{red})$$

$$+ \hat{k}_G(\text{green}, \text{red}) \cdot \hat{k}_G(\text{green}, \text{red})$$



# Embedding Generation

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

$$K = Q Q^T$$

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

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

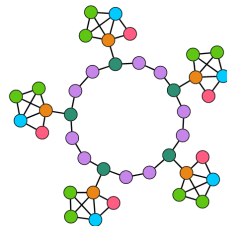
- can be very inefficient
- can be prohibitive in terms of the required memory

To avoid explicitly constructing the kernel matrix, SEGK resorts to low-rank approximation algorithms

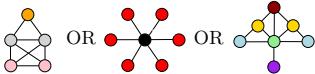
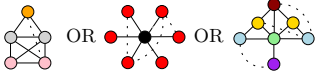
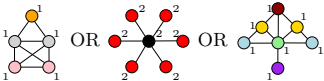
↪ e. g., the Nyström method allows us to obtain  $Q \in \mathbb{R}^{n \times d}$  (with  $d \ll n$ ) such that  $K \approx Q Q^T$

# Synthetic Node Classification Dataset

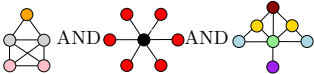
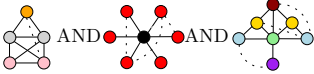
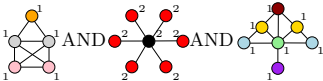
- Generated synthetic graphs with planted structural equivalences
- Structurally equivalent nodes are assigned the same class labels
- generated graphs consist of a cycle of length 40 and some basic shapes (“house”, “fan”, “star”) which are regularly placed along the cycle
- “basic” setup: 10 instances 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	<b>1.000</b>	<b>1.000</b>
		STRUC2VEC	0.784	0.708
		DRNE	0.987	0.980
		GRAPHWAVE	0.995	0.993
		SEKG-SP	<b>1.000</b>	<b>1.000</b>
		SEKG-WL	<b>1.000</b>	<b>1.000</b>
		SEKG-GR	<b>1.000</b>	<b>1.000</b>
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
		SEKG-SP	0.941	0.907
		SEKG-WL	0.907	0.850
		SEKG-GR	<b>0.956</b>	<b>0.925</b>
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
		SEKG-SP	<b>0.990</b>	<b>0.984</b>
		SEKG-WL	<b>0.990</b>	<b>0.984</b>
		SEKG-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	<b>0.998</b>	<b>0.996</b>
		STRUC2VEC	0.738	0.592
		DRNE	0.930	0.876
		GRAPHWAVE	0.982	0.965
		SEGK-SP	<b>0.998</b>	<b>0.996</b>
		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	<b>0.892</b>	<b>0.818</b>
		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	<b>0.960</b>	<b>0.931</b>
		SEGK-GR	0.783	0.776

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

METHOD	HOMOGENEITY	COMPLETENESS	SILHOUETTE	ACCURACY	F1-SCORE
DEEPWALK	0.240	0.081	0.214	0.324	0.202
RoLX	0.178	<b>0.141</b>	0.040	0.264	0.154
STRUC2VEC	0.243	0.122	0.246	0.323	0.190
DRNE	<b>0.344</b>	0.112	<b>0.420</b>	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	<b>0.360</b>	<b>0.222</b>
SEGK-GR	0.144	0.088	0.127	0.294	0.172

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

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

## Planetoid

- assumes node attributed graphs (e. g., a feature vector is associated with each vertex)
- takes into account both the class labels and the graph structure to learn node embeddings
- minimizes the following loss function:  $\mathcal{L} = \mathcal{L}_s + \lambda \mathcal{L}_u$   
 $\mathcal{L}_s$ : a supervised loss of predicting the labels  
 $\mathcal{L}_u$ : an unsupervised loss of predicting the graph context

[Yang et al., ICML'16]



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  $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

$$\hat{A} \times W^0$$

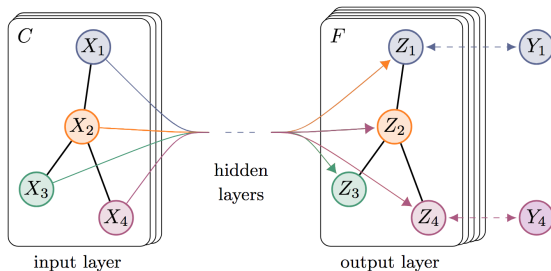
Then, the output of the model is:

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

where

$X$ : matrix whose rows contain the attributes of the nodes

$W^0, W^1$ : trainable weight matrices



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

$$\mathcal{L} = - \sum_{i \in I} \sum_{j=1}^{|C|} Y_{ij} \log Z_{ij} \quad \text{minimize the cross entropy loss}$$

$I$ : indices of the nodes of the training set

$C$ : set of class labels

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]

Dataset	Type	Nodes	Edges	Classes	Features	<u>Label rate</u>
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

## 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)
<b>GCN</b>	<b>70.3 (7s)</b>	<b>81.5 (4s)</b>	<b>79.0 (38s)</b>	<b>66.0 (48s)</b>

Observation: DeepWalk → unsupervised learning of embeddings

↪ fails to compete against the supervised approaches