GRAPH REPRESENTATIONAL LEARNING

Analysis of Large Scale Social Networks

Bart Thijs

This last lecture is focussed on learning representations of graphs at local and global level:

These representations are being used in many ML applications:

- Link prediction, Classification, unsupervised learning, clustering…

This requires informative, independent and discriminative features that can be transformed into some vector representation.

WHAT?

Different approaches to achieve this vector representation:

- Matrix Factorization
- Direct Methods
- Attributes and neighbourhoods
- Convolutional Neural Networks over Graphs (GCN)

Goal: Optimize mapping so that the geometric relations in the learned space mirror the structure of the original graph.

See (Hamilton, Ying & Leskovec, 2016)

WHAT?

DIRECT ENCODING METHODS

Eg :

DeepWalk (Perozzi et al, 2014).
Node2Vec (Grover & Leskovec, 2016)

Drawbacks:

- No parameter sharing
- Node attributes are neglected
- Transductive

DEEP WALK

Learning social representation with these four characteristics:

1. **Adaptability**: capture new links or relations
2. **Community Aware**: Distance between representations should correspond to social similarity
3. **Low Dimensionality**: for better generalizations, speed of convergence and inference
4. **Continuous**: more nuanced view, smooth decision boundaries and more robust classifications

DEEP WALK

Learning social representation with these four characteristics:

1. **Adaptability**: capture new links or relations
2. **Community Aware**: Distance between representations should correspond to social similarity
3. **Low Dimensionality**: for better generalizations, speed of convergence and inference
4. **Continuous**: more nuanced view, smooth decision boundaries and more robust classifications

DEEP WALK

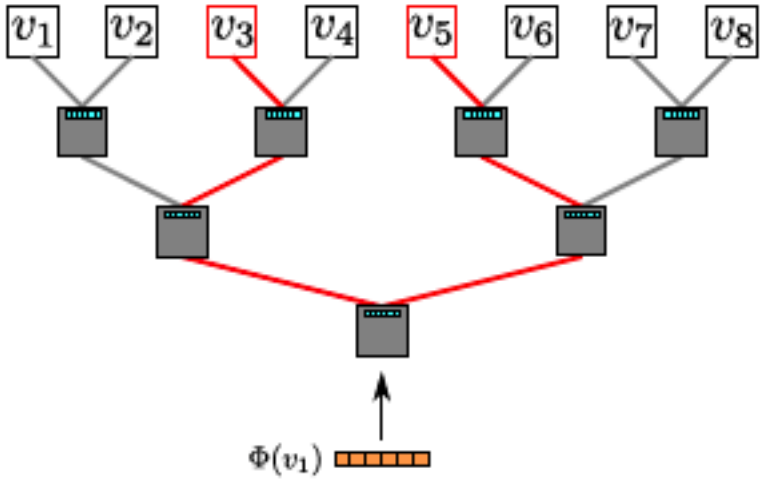
Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_t} = RandomWalk(G, v_i, t)$
    SkipGram(Φ, $\mathcal{W}_{v_t}$, w)
  end for
end for

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_t}$, w)

for each $v_j \in \mathcal{W}_{v_t}$ **do**
  for each $u_k \in \mathcal{W}_{v_t}[j - w : j + w]$ **do**
    $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
    $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
  end for
end for



(c) Hierarchical Softmax.

Their contribution:

1. Learning social representation instead of computing statistics
2. Not an extension of the classification procedure itself
3. Scalable solution
4. Unsupervised learning

DEEP WALK

NODE2VEC

The Skip-gram model represents a sequence in a random walk as a sequence of words. An important difference is that the random walk implies a sampling strategy.

Node2Vec introduces a flexible and tuneable sampling strategy. The method uses the same Skip-gram model, hierarchical softmax and SGD for the optimization and parameter estimation.

In addition, the method shows that the embeddings of individual nodes can be extended to pairs of nodes and thus edges. This opens the door to predictive tasks on edges (link prediction)

Real network present mixture of community structure and structural roles:

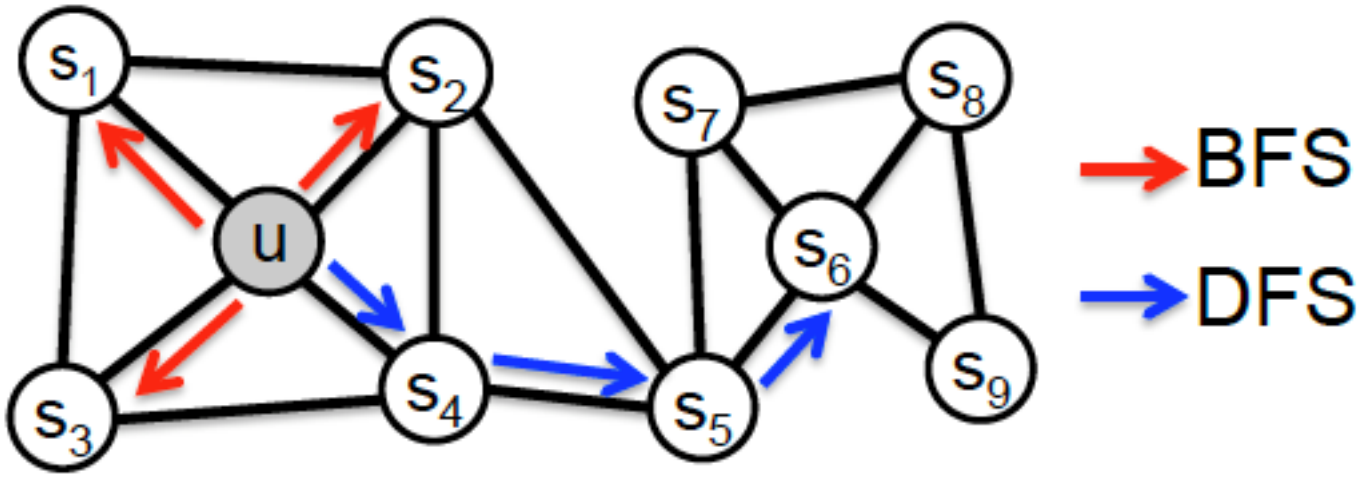


Figure 1: BFS and DFS search strategies from node u ($k = 3$).

Flexible algorithm that can learn representations that embed nodes from the same cluster close together or provide similar embeddings for nodes with similar roles.

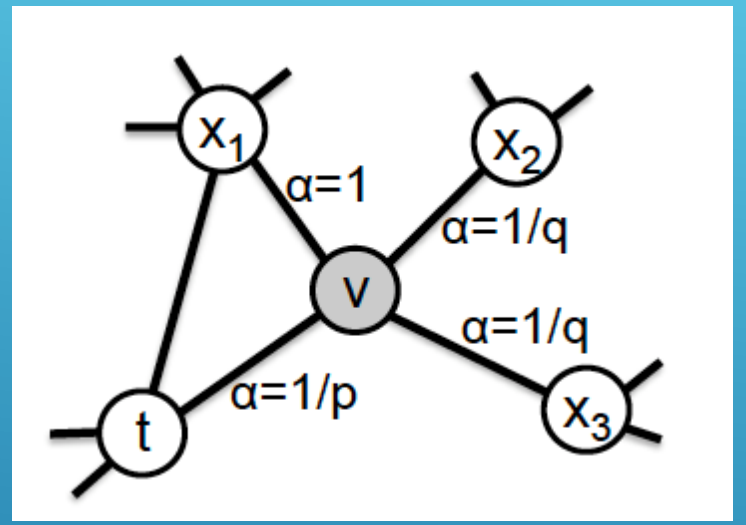
NODE2VEC

This flexibility is achieved through the adaptation of the random walk procedure in order to obtain different notions of a node's neigbourhood:

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

With:

p: Return parameter
q: In-out parameter



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

See (Grover & Leskovec, 2016)

NODE2VEC

NODE2VEC

The three phases are sequentially executed

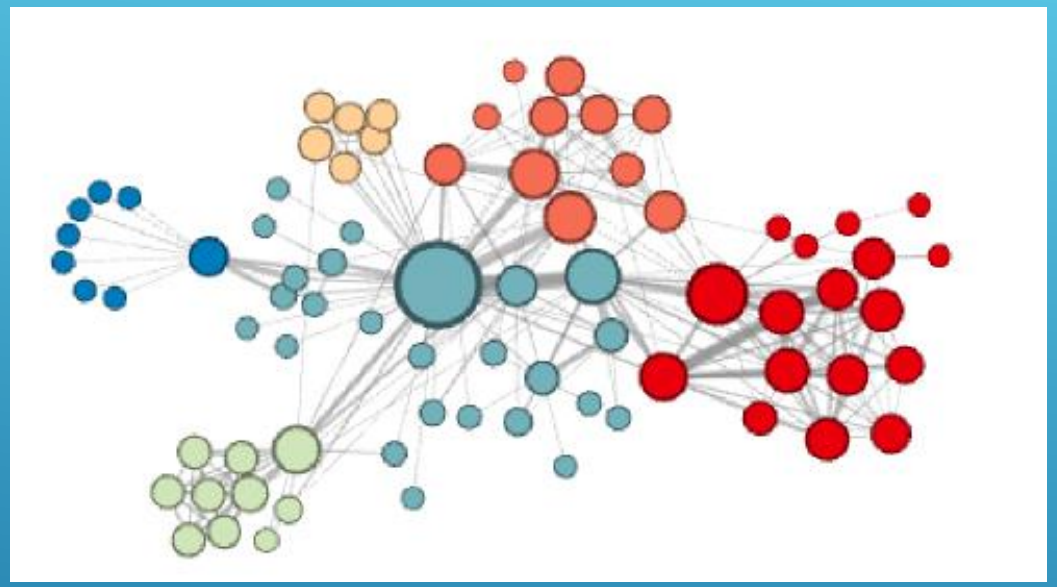
1. 2nd order transition probabilities are calculated
2. Sampling, random walk simulations. No sliding window like Deep Walk, each walk starts with source node *u*;
3. SGD .

Algorithm 1 The *node2vec* algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d, Walks per node r, Walk length l, Context size k, Return p, In-out q)
- π = PreprocessModifiedWeights(G, p, q)
- $G' = (V, E, \pi)$
- Initialize $walks$ to Empty
- **for** $iter = 1$ **to** r **do**
  - **for all** nodes $u \in V$ **do**
    - $walk$ = node2vecWalk(G', u, l)
    - Append $walk$ to $walks$
- f = StochasticGradientDescent(k, d, $walks$)
- **return** f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u, Length l)
- Inititalize $walk$ to $[u]$
- **for** $walk\_iter = 1$ **to** l **do**
  - $curr = walk[-1]$
  - V_{curr} = GetNeighbors($curr$, G')
  - s = AliasSample(V_{curr}, π)
  - Append s to $walk$
- **return** $walk$

Results: Les Misérables Network



Community Structure
p=1, q=0.5 (DFS)

Structural Roles
p=1, q=2 (BFS)

See (Grover & Leskovec, 2016)

NODE2VEC

Learning Edge features:

Defining a binary operator ° over the corresponding feature vectors:

Operator	Symbol	Definition
Average	$\boxplus$	$[f(u) \boxplus f(v)]_i = \frac{f_i(u)+f_i(v)}{2}$
Hadamard	$\boxdot$	$[f(u) \boxdot f(v)]_i = f_i(u) * f_i(v)$
Weighted-L1	$\|\cdot\|_{\overline{1}}$	$\|f(u) \cdot f(v)\|_{\overline{1}i} = \|f_i(u) - f_i(v)\|$
Weighted-L2	$\|\cdot\|_{\overline{2}}$	$\|f(u) \cdot f(v)\|_{\overline{2}i} = \|f_i(u) - f_i(v)\|^2$

Operators are defined over any pair of nodes (also unconnected)

See (Grover & Leskovec, 2016)

NODE2VEC

However, these two approaches are transductive, they can only provide embeddings for a fixed graph.

They are not generalizable to unseen nodes added to the network

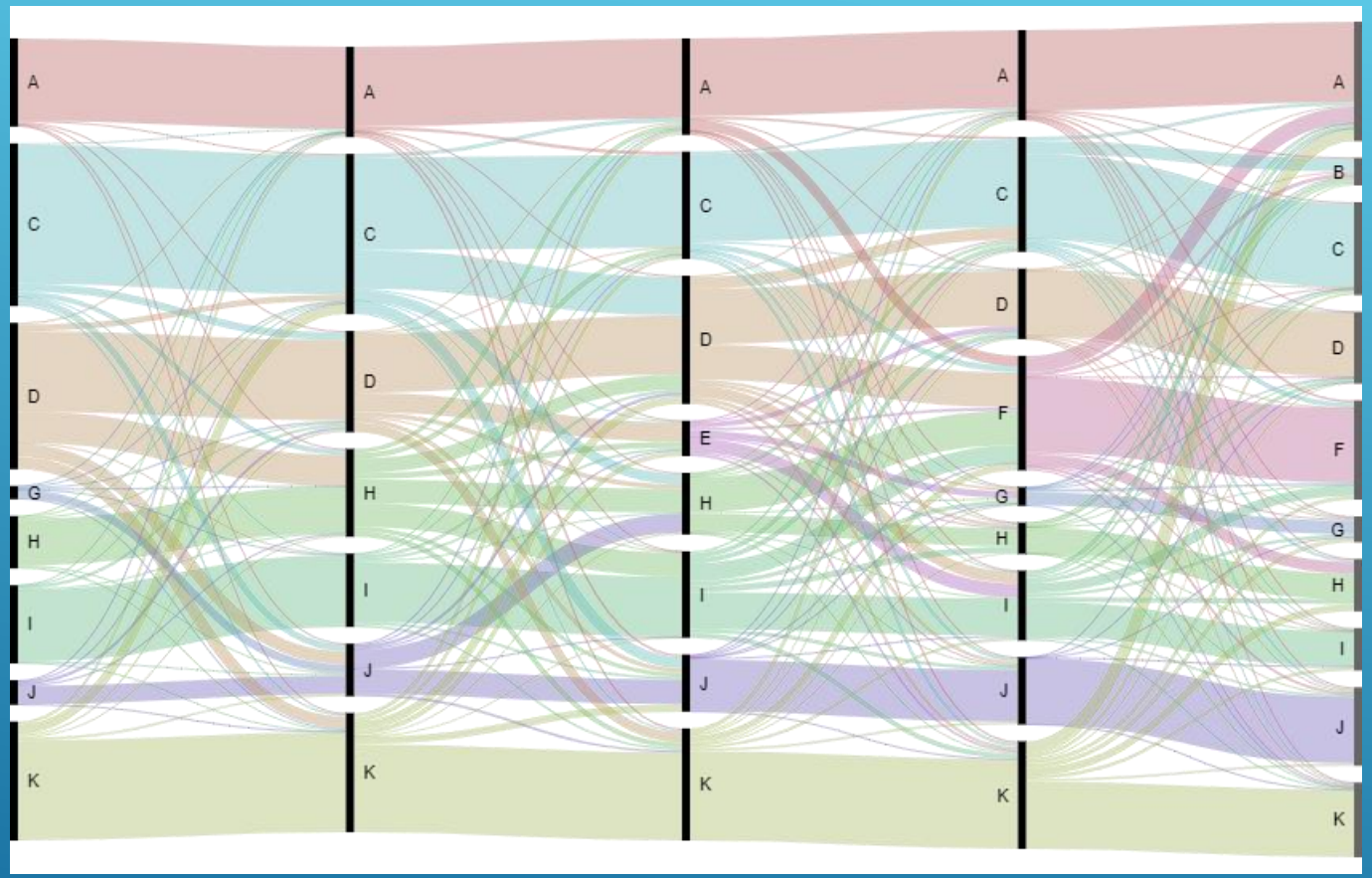
New approaches introduce techniques that are generalizable even to different networks.

Eg :

- Attri2Vec
- GraphSage

OTHER MODELS?

Difference in network structure based purely on links (left) or purely on attributes (text; right) and hybrid combinations

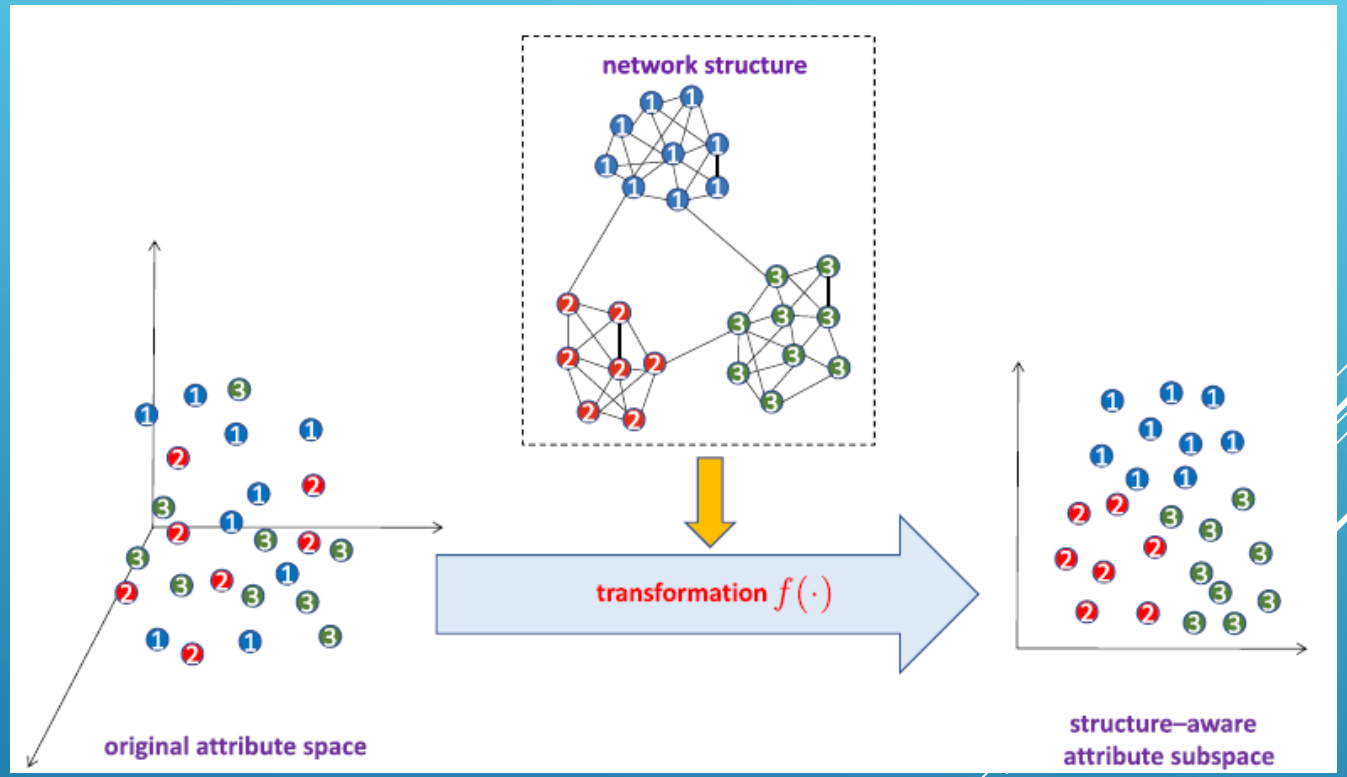


A Patents & Technology
B Impact Measures
C Collaboration
D Journal Impact
F Research Assessment
G Web-related Studies
H Research Production
I Citation Analysis
J Field & Regional Studies
K Science Mapping

LINKS VS ATTRIBUTES

Discovering the latent subspace of node attributes that well respects the network structure.:

The transformed attribute vector is used to predict the context nodes in a random walk. Nodes sharing neighbourhoods are located closely to each other in the subspace



(see Zhang, Yin, Zhu & Zhang, 2019)

ATTRI2VEC

4 Different mapping functions (transformations) are tested

W^{in} is the trainable input-hidden (m x d)-weight matrix.

(1) *linear mapping*

$$f(\boldsymbol{x}_i) = W^{in^{\mathrm{T}}}\boldsymbol{x}_i = \left[\boldsymbol{w}_1^{in^{\mathrm{T}}}\boldsymbol{x}_i, \ldots, \boldsymbol{w}_d^{in^{\mathrm{T}}}\boldsymbol{x}_i\right]^{\mathrm{T}};$$

(2) *rectified linear unit (ReLU) mapping*

$$f(\boldsymbol{x}_i) = \left[\max(0, \boldsymbol{w}_1^{in^{\mathrm{T}}}\boldsymbol{x}_i), \ldots, \max(0, \boldsymbol{w}_d^{in^{\mathrm{T}}}\boldsymbol{x}_i)\right]^{\mathrm{T}};$$

(3) *approximated kernel mapping* used in UPP-SNE (Zhang et al. 2017):

$$f(\boldsymbol{x}_i) = \frac{1}{\sqrt{m}}\left[\cos(\boldsymbol{w}_1^{in^{\mathrm{T}}}\boldsymbol{x}_i), \ldots, \cos(\boldsymbol{w}_{d/2}^{in^{\mathrm{T}}}\boldsymbol{x}_i), \sin(\boldsymbol{w}_1^{in^{\mathrm{T}}}\boldsymbol{x}_i), \ldots, \sin(\boldsymbol{w}_{d/2}^{in^{\mathrm{T}}}\boldsymbol{x}_i)\right]^{\mathrm{T}};$$

(4) *sigmoid mapping*

$$f(\boldsymbol{x}_i) = \left[1/(1+\exp(-\boldsymbol{w}_1^{in^{\mathrm{T}}}\boldsymbol{x}_i)), \ldots, 1/(1+\exp(-\boldsymbol{w}_d^{in^{\mathrm{T}}}\boldsymbol{x}_i))\right]^{\mathrm{T}}.$$

(see Zhang, Yin, Zhu & Zhang, 2019)

ATTRI2VEC

ATTRI2VEC

Extensions:

1. A deep neural network is compared to the shallow one-layer non-linear mapping. (Attri2Vec Deep)
   Observations: One-layer is good enough compared to the the multi-layer model
   - Faster
   - Less prone to local minima
2. Inductive: transformation function can be applied on the attribute vector of out-of-sample nodes
3. Link embeddings can be obtained through element-wise operators.
   Training: SVM on sampled node pairs and negative samples.

(see Zhang, Yin, Zhu & Zhang, 2019)

GRAPHSAGE

An inductive approach has many advantages over the transductive counterparts:

Eg.

- Embed new posts in a social network
- Apply protein-protein interactions on a new organism
- Transfer to other graphs with similar structure and features.

But problem is difficult:

Computational expensive (additional rounds of SGD)

See GraphSage (Hamilton, Ying & Leskovec, 2017)

GRAPHSAGE

GraphSage (Sample and Aggregate) extends the transductive Graph Convolutional Networks and adds node attributes to the learning procedure.

Simultaneous learning of topological structure and distribution of node features.

The system does not learn embeddings for individual nodes but training aggregator functions on the node's local neighbourhood.

GRAPHSAGE

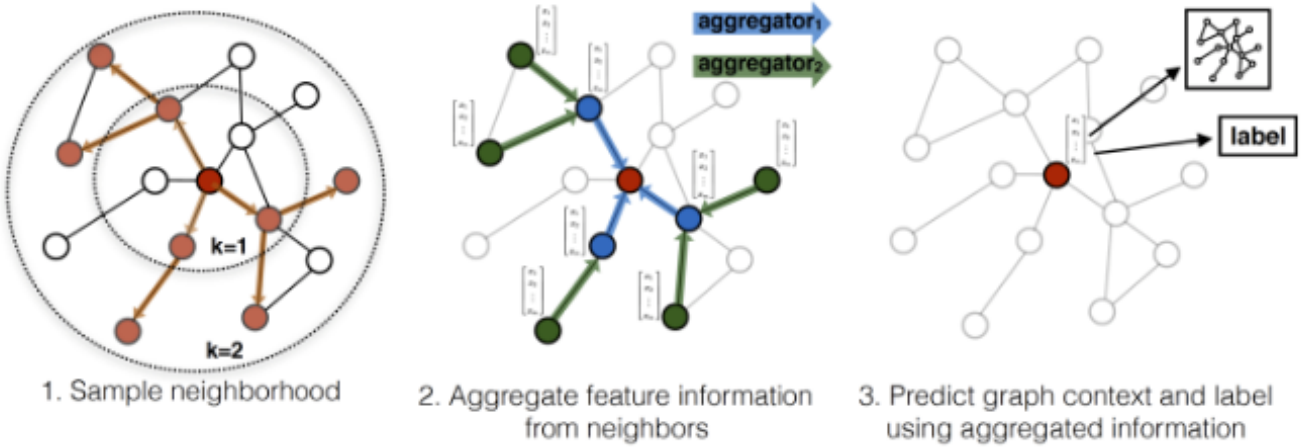


Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

GRAPHSAGE

Sampling the neighbourhood:

Draw a uniform sample from the full neighbourhood of the nodes in the graph at each iteration k.

This reduces space and time complexity by user-specified constants instead of being based on the network size.

Learning the weight parameters and the aggregator functions:

Using SGD on a graph-based loss function that encourages nearby nodes to have similar representations while forcing the representations of disparate nodes to be distinct

See GraphSage (Hamilton, Ying & Leskovec, 2017)

GRAPHSAGE

Aggregator Functions:

- Mean Aggregator: An inductive equivalent to the convolutional propagation rule in GCN. Taking a weighted elementwise mean over the vectors in the neighbourhood and apply a sigmoid nonlinearity.
- LSTM Aggregator: Large expressive capability; Not symmetric (random permutation of neighbours is used)
- Pooling Aggregator : elementwise max-pooling operation with a simple single layered architecture.
  This aggregator is symmetric and trainable

See GraphSage (Hamilton, Ying & Leskovec, 2017)

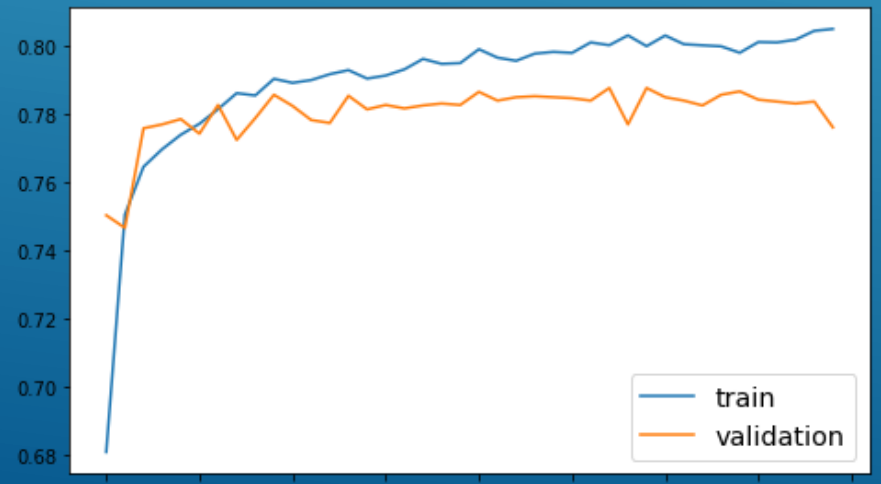
Example:

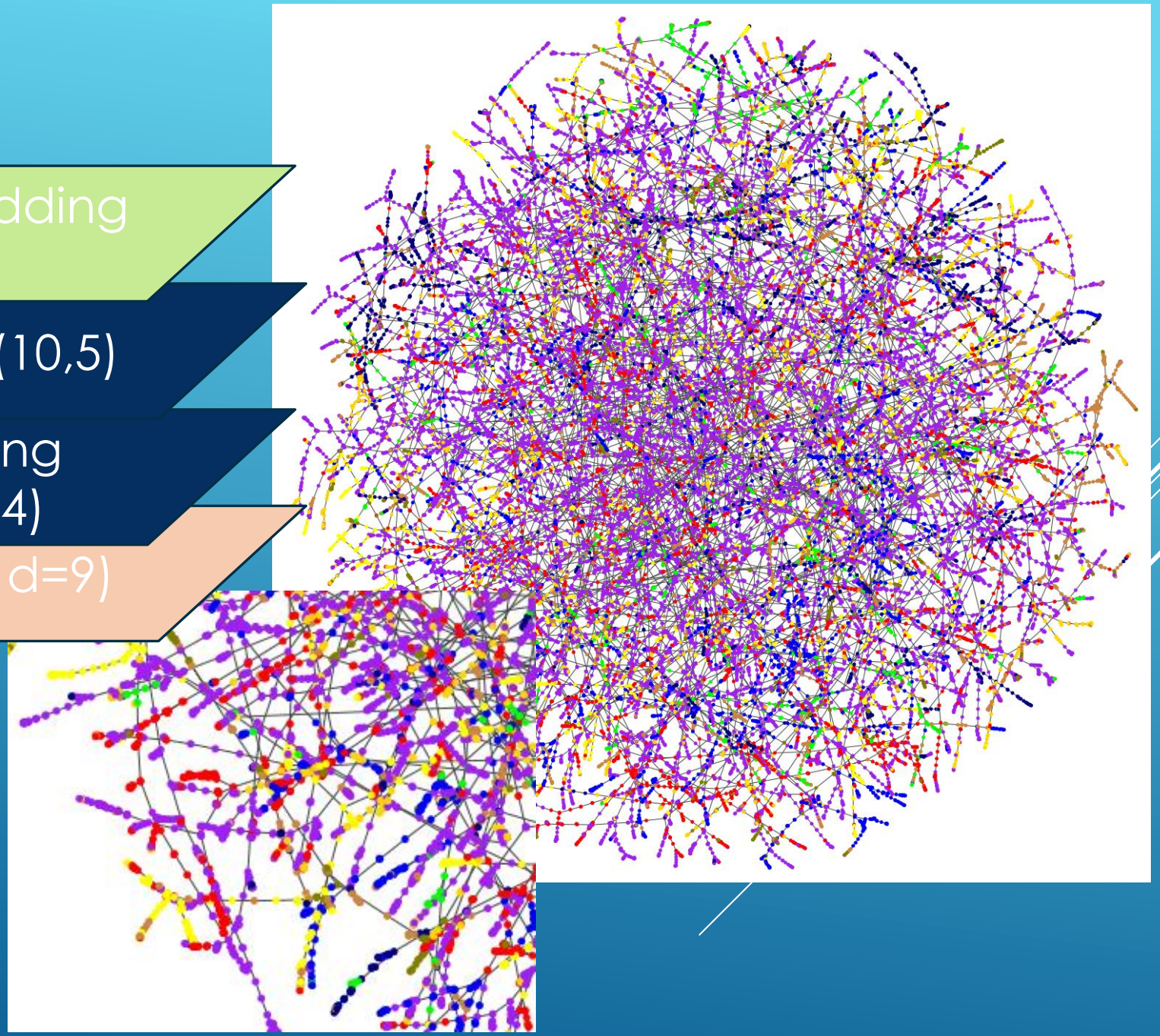
Input = Doc2Vec embedding + Label

Node Flow Generator (10,5)

GraphSage Embedding (drop out=0.5, size=64)

Dense Layer (SoftMax, d=9)





Semi-Supervised classicition with GCN:

Instead of using the Laplacian matrix as a regularization term in the loss function

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2 = f(X)^\top \Delta f(X).$$

GCN proposes to directly encode the graph structure in the neural network model *f(X, A)*

(See Kipf & Welling, 2017)

GCN

Semi-Supervised classicition with GCN:

Layer-wise propagation rule

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right).$$

Using a non-linear activation function (eg ReLU, sigmoid)

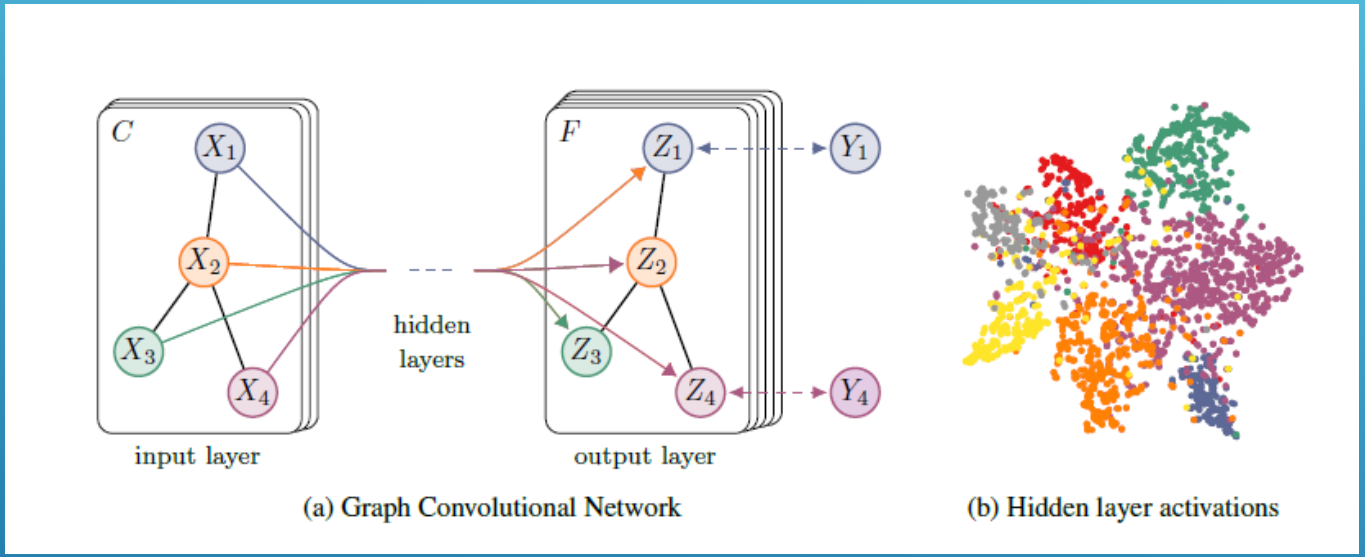
The method takes advantage of eigenvalues and eigenvector properties related to the normalized graph Laplacian to calculate the spectral convolutions.

(See Kipf & Welling, 2017)

GCN

Semi-Supervised classicition with GCN:

The method allows stacking multiple (hidden) layers with different activation functions and dimensions

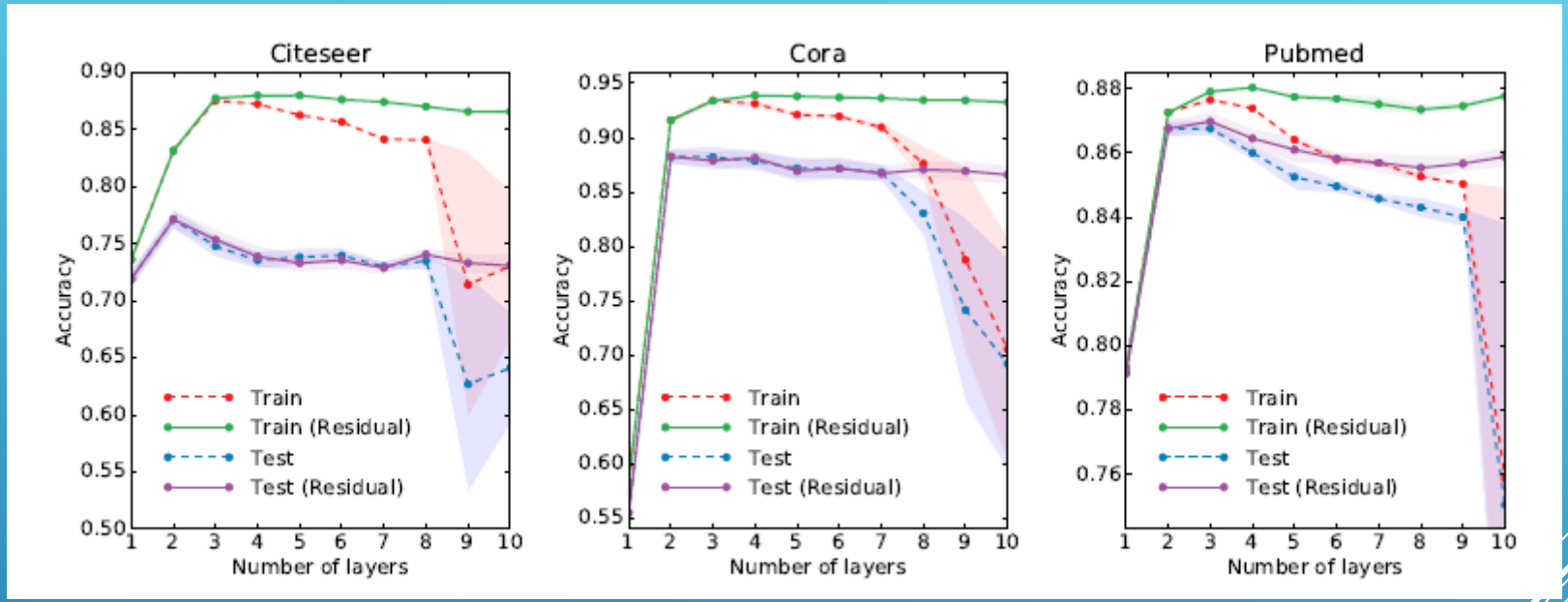


(See Kipf & Welling, 2017)

GCN

Semi-Supervised classicition with GCN:

Model Depth:



Best results are obtained with 2 or 3 layers. Above 7, overfitting becomes a serious issue with increased parameters.

(See Kipf & Welling, 2017)

GCN

GRAPHWAVE

GraphWave

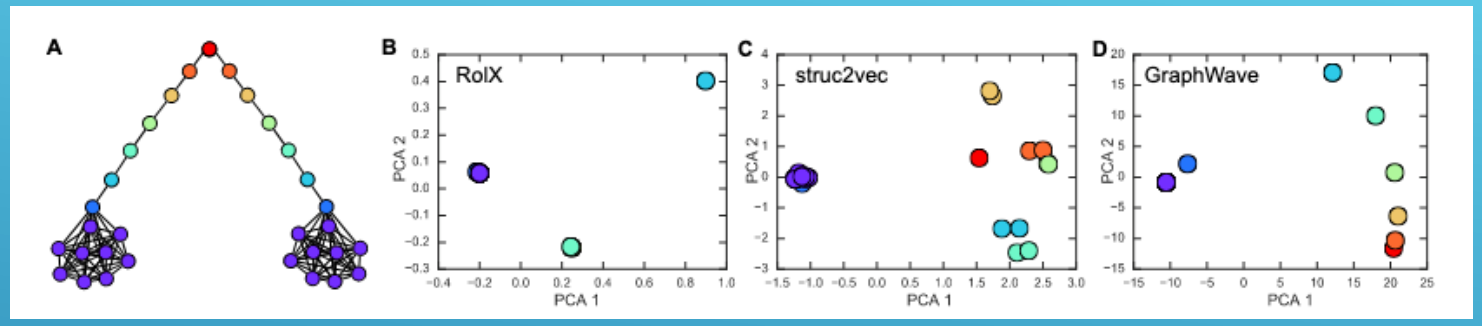
The objective of this embedding is to capture the information that characterize nodes with topologically similar neighbourhoods.

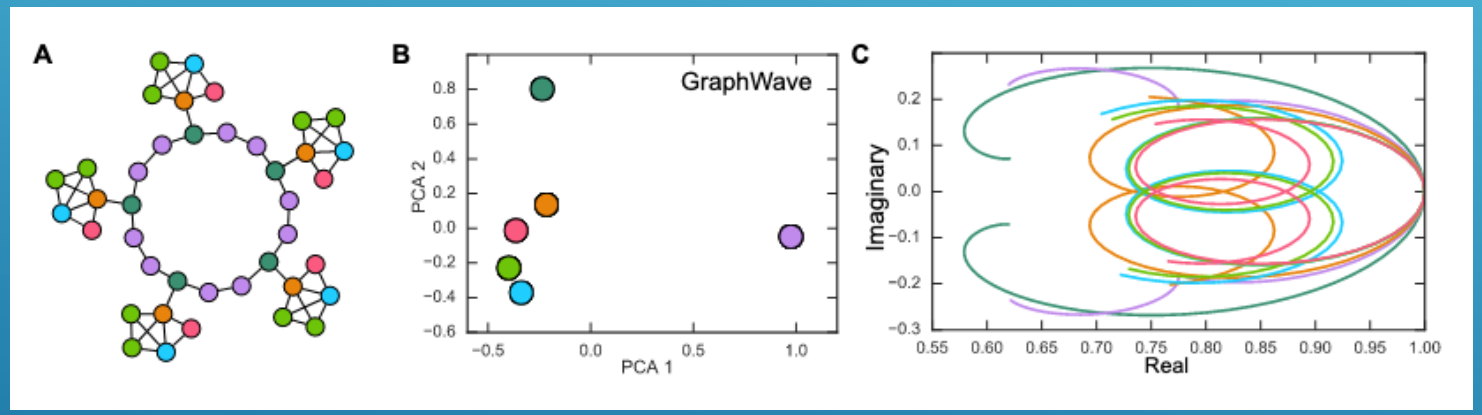
Eg: node on a chain, center of a star, bridge between two clusters,…

A method for detecting these patterns must learn the vector-based embeddings but also an adequate distance metric to distinguish between the different roles.

See Donnat, Zitnik, Hallac & Leskovec, 2018

GraphWave: results





See Donnat, Zitnik, Hallac & Leskovec, 2018

GRAPHWAVE