Spatial-Based Graph Embedding and Its Potential Applications on Network Biology

05/05/2020

Ping-Han Hsieh

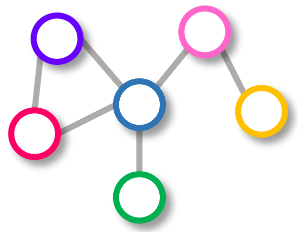
Outline

- Introduction
- Problem Definition
- Methods
  - DeepWalk
  - node2vec
  - GraphSAGE
  - Graph Attention Network (GAT)
- Discussion
- Potential Applications

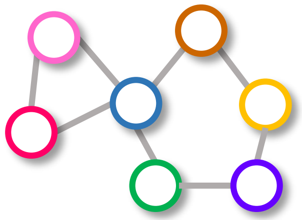
Introduction

- Graph is an important data structure used to store relational information (edge) between entities (vertex)
- Graph is a non-Euclidean data structure.
- How to exploit the structural information in graph.

sample 1: treated



sample 2: control



$$A_1 = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad A_2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Problem Definition

- Given a (undirected, unweight) graph $G = (V, E)$ where $E \subseteq (V, V)$.
- Suppose X is the feature representation of the vertices:

$$\boldsymbol{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,|V|} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,|V|} \\ \vdots & \vdots & \ddots & \vdots \\ X_{d',1} & X_{d',2} & \cdots & X_{d',|V|} \end{bmatrix}$$

features

vertices

$$\mathbf{x}_k = \begin{bmatrix} X_{1,k} \\ X_{2,k} \\ \vdots \\ X_{d',k} \end{bmatrix} \qquad \mathbf{x}_k^r = \begin{bmatrix} X_{k,1} \\ X_{k,2} \\ \vdots \\ X_{k',|V|} \end{bmatrix}$$

- To find an optimal function $f: X \in \mathbb{R}^{d' \times |V|} \longrightarrow Z \in \mathbb{R}^{d \times |V|}$, where Z is the latent representation of vertices which consider the structural information in the graph.

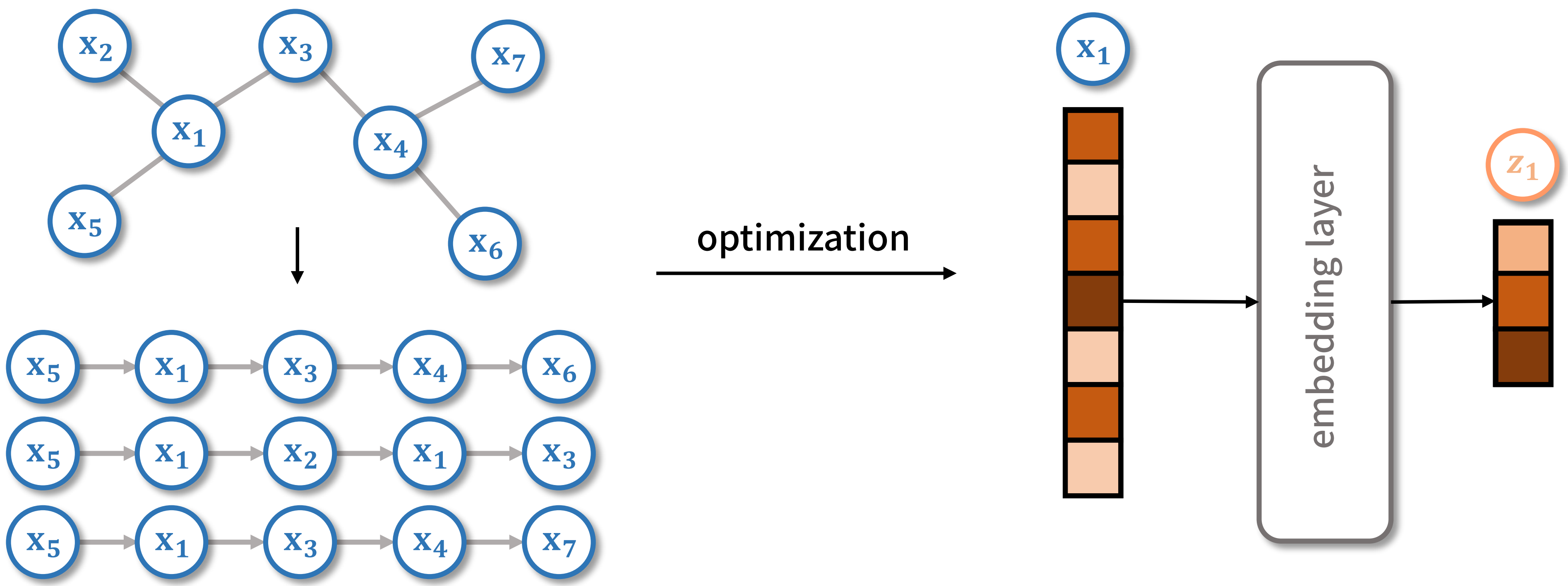
Different Methods

- Vertex Centrality
  - Degree, closeness, betweenness, eigen-centrality.
  - No tunable parameters in this setting.
- Spectral-based graph convolutional neural network
  - Combine the eigen-decomposition of Laplacian matrix with convolutional Theorem in Fourier transform.
  - Not scalable to large graph
  - Requires global re-computation when the structure of graph change
- Spatial-based graph embedding

General Framework

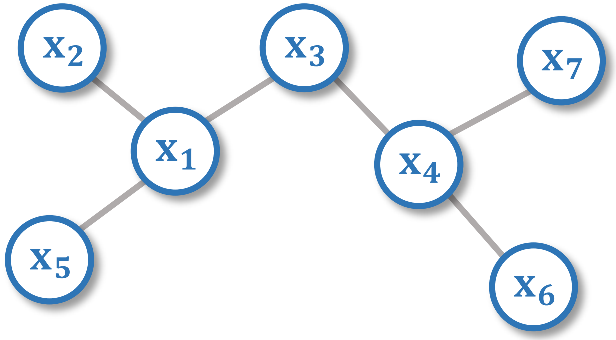
A: Neighborhood sampling (DeepWalk, node2vec)

B: Embedding (GraphSAGE, GAT)

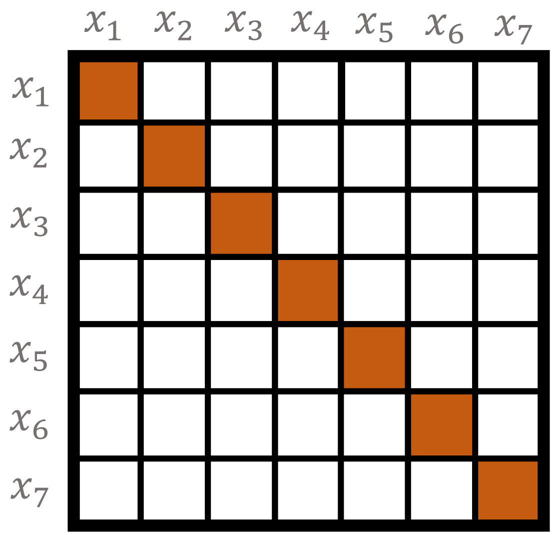


DeepWalk (1)

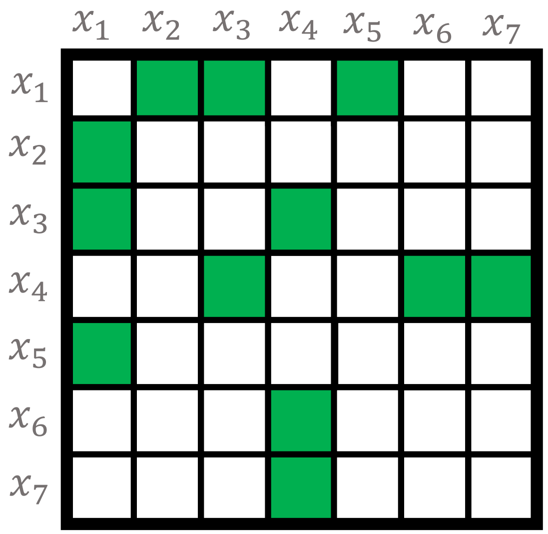
Input Data



input graph



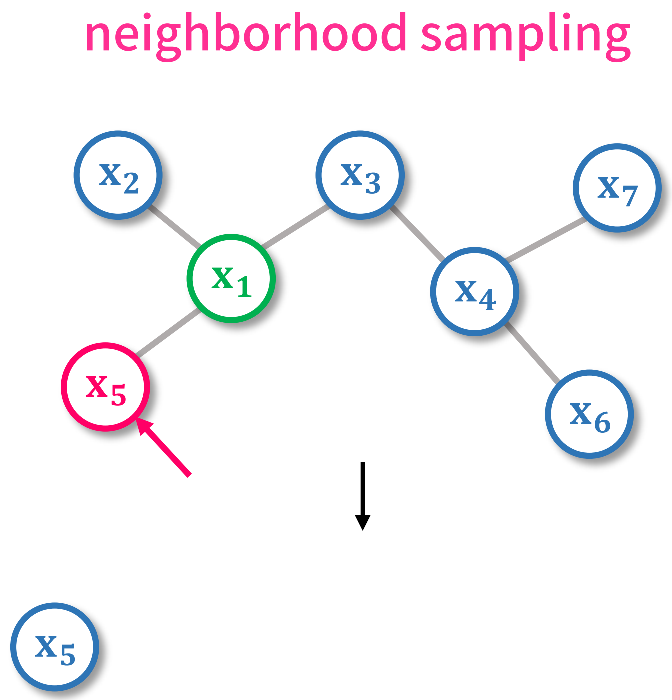
vertices feature



adjacency matrix

DeepWalk (2)

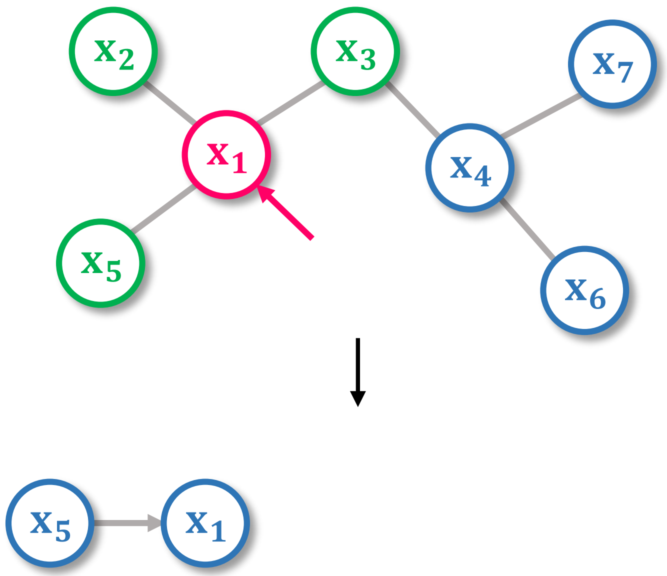
Random Walk



DeepWalk (3)

Random Walk

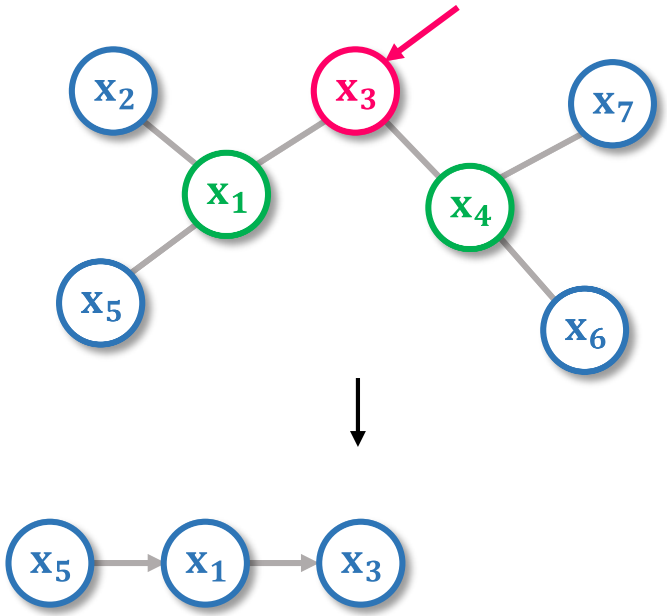
neighborhood sampling



DeepWalk (4)

Random Walk

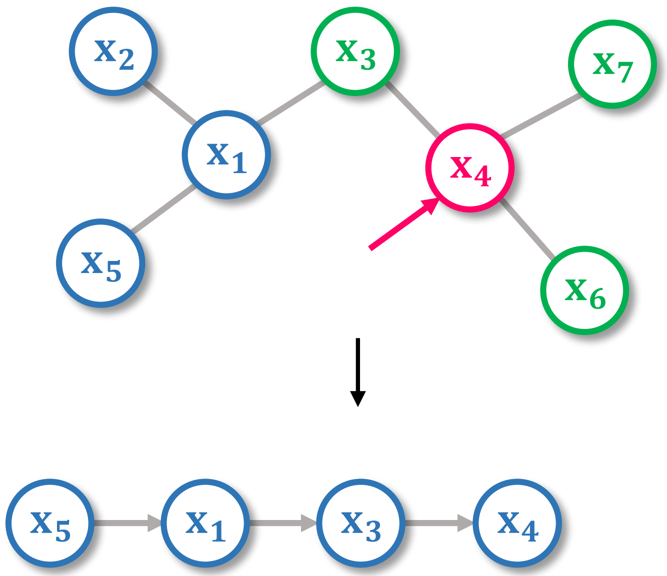
neighborhood sampling



DeepWalk (5)

Random Walk

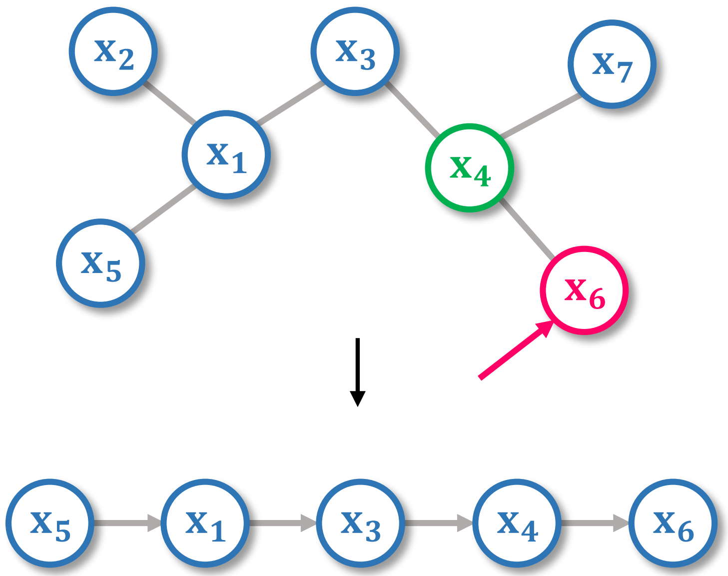
neighborhood sampling



DeepWalk (6)

Random Walk

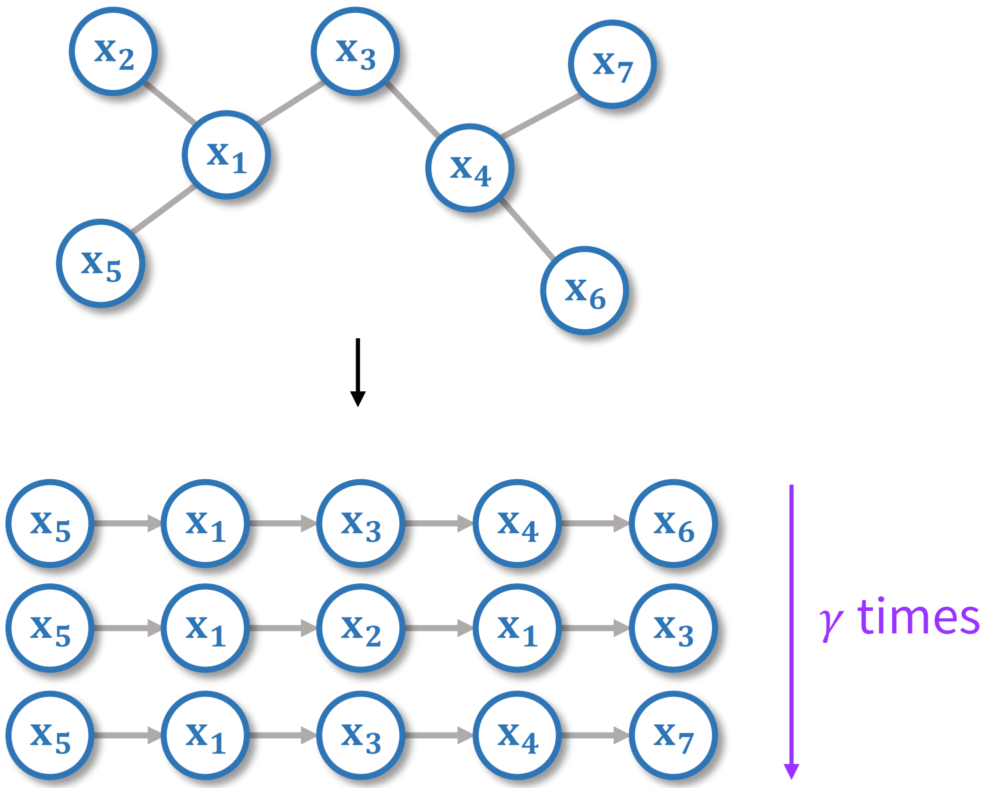
neighborhood sampling



DeepWalk (7)

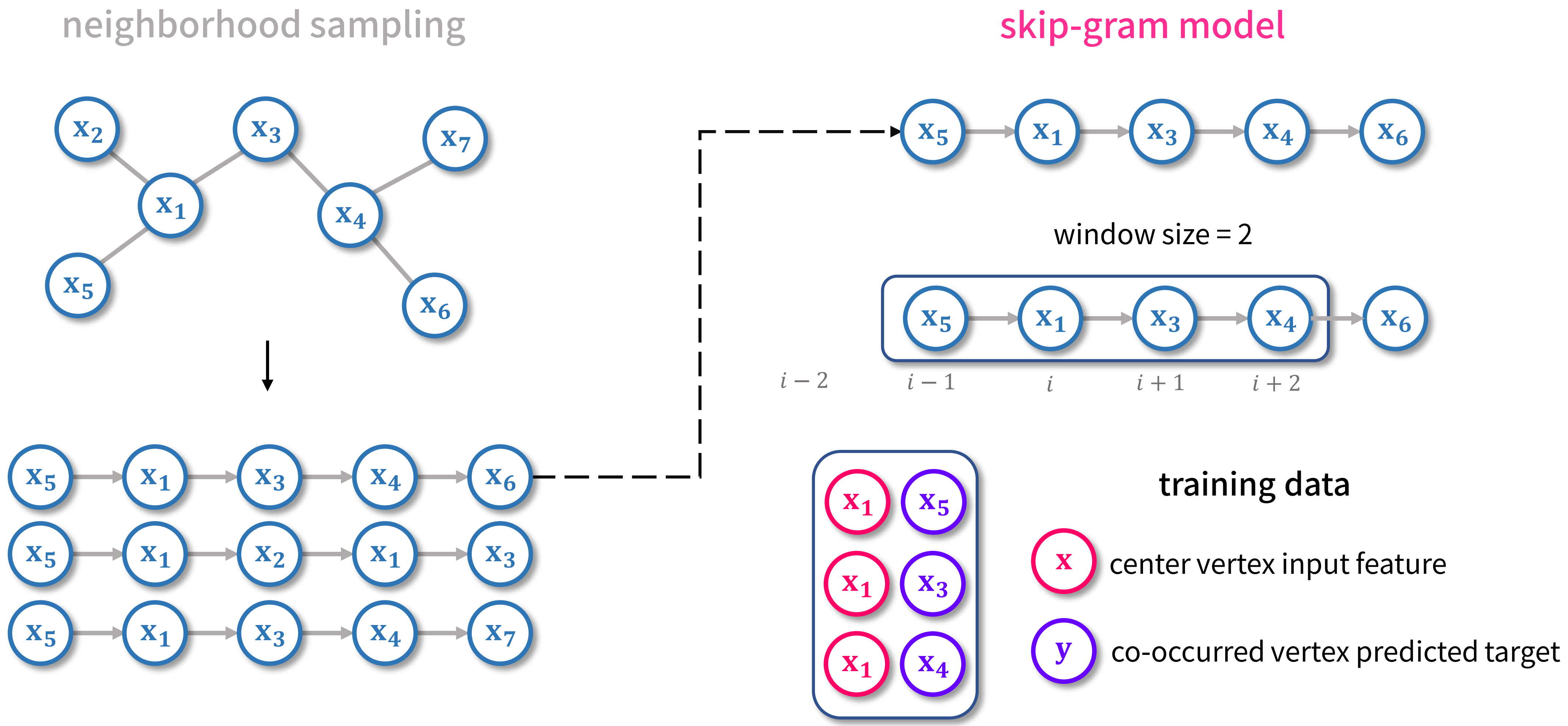
Random Walk

neighborhood sampling



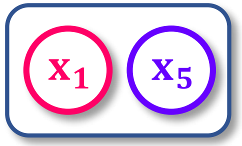
DeepWalk (8)

Random Walk



DeepWalk (9)

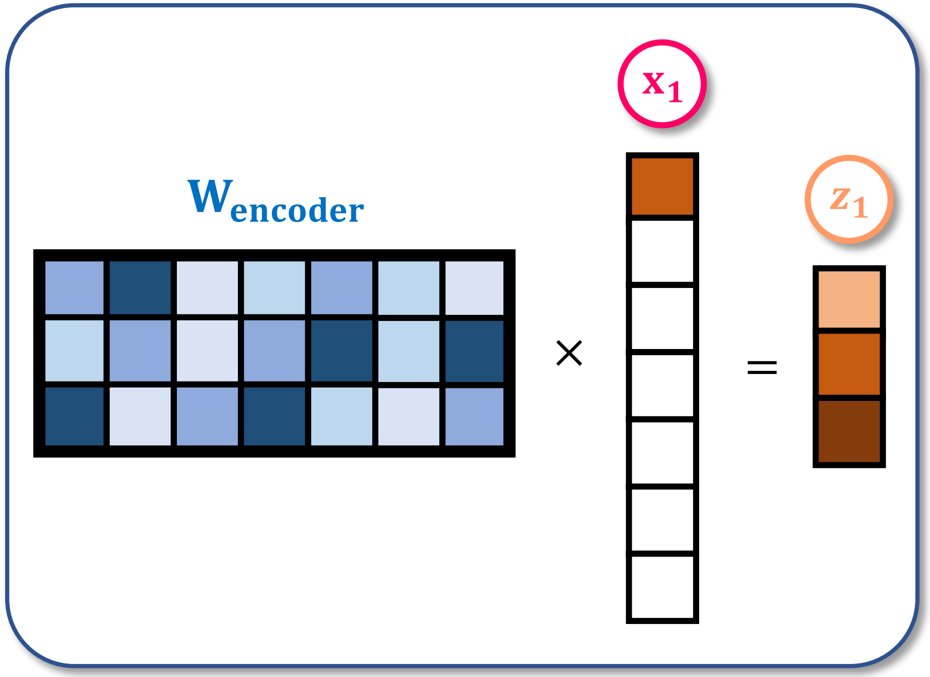
Optimization



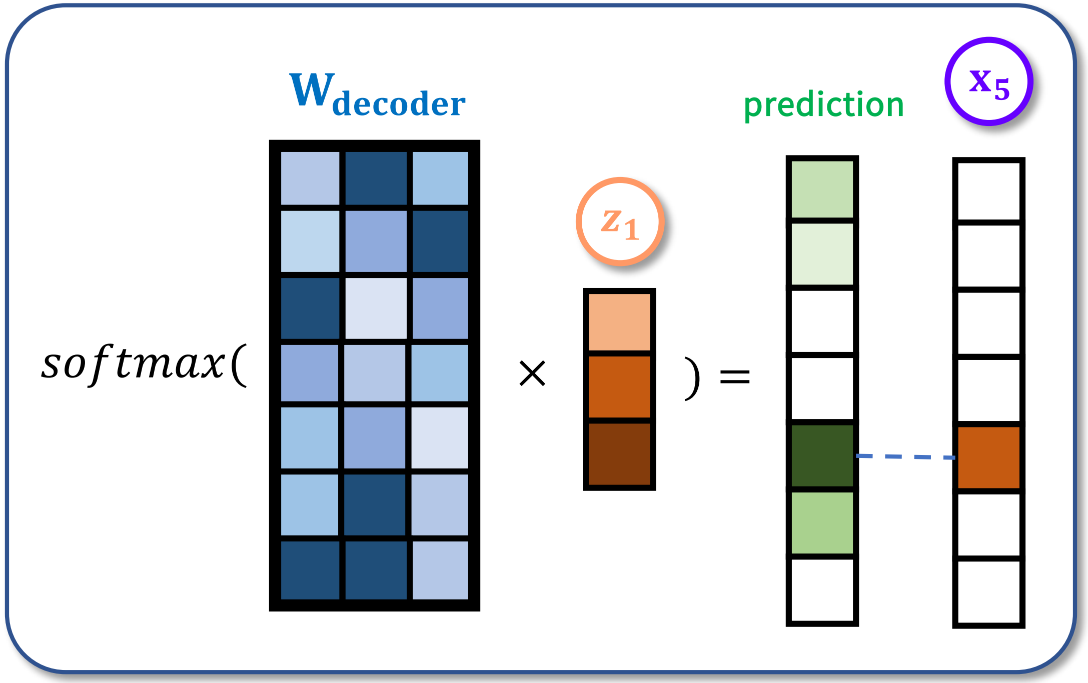
training data

$$H(f(\boldsymbol{x}), \boldsymbol{y}) = -\sum_i \sum_c y_c \ln f_c(x_i)$$

embedding layer

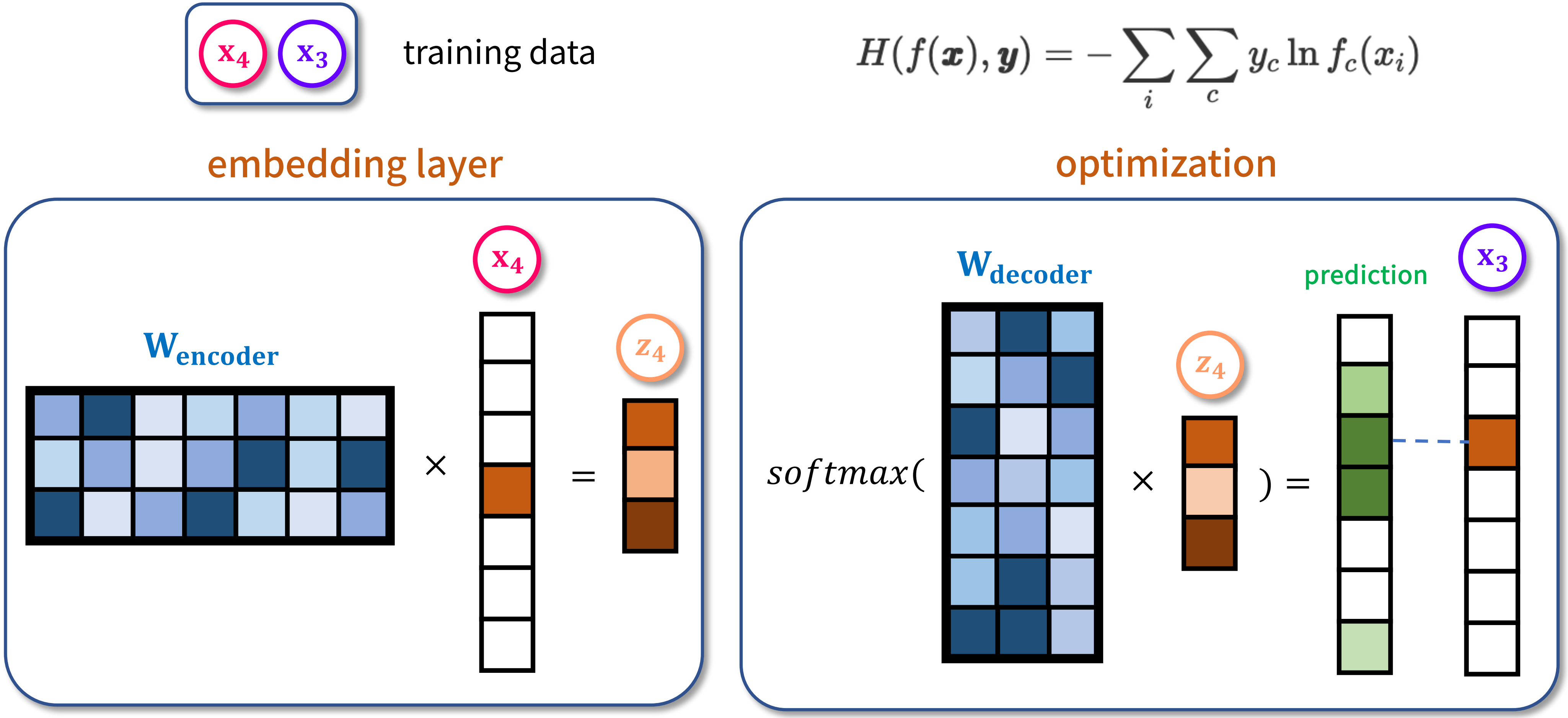


optimization



DeepWalk (10)

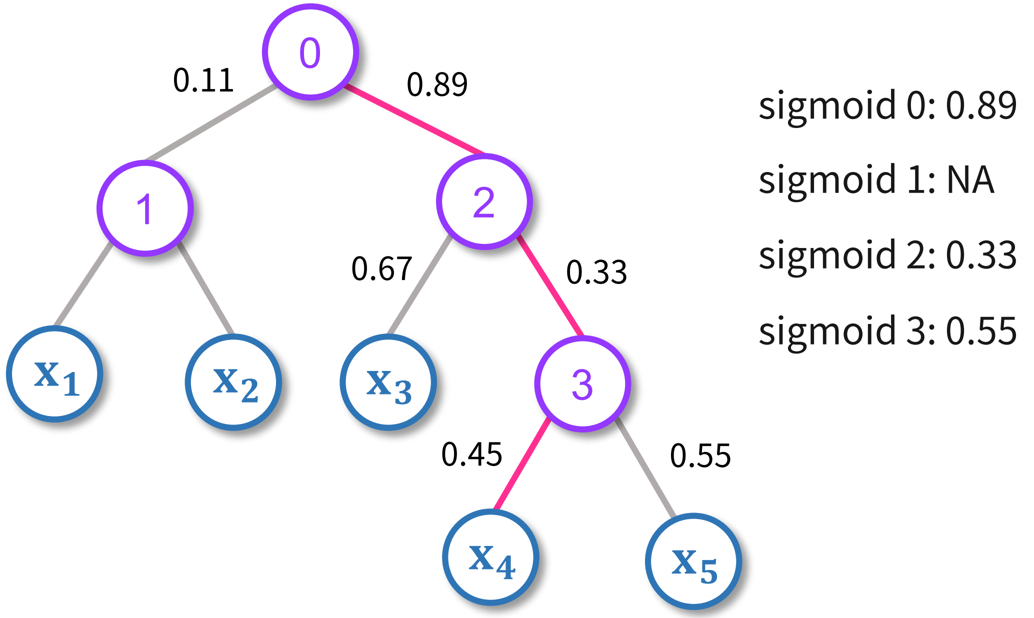
Optimization



DeepWalk (11)

Hierarchical Softmax

- In the optimization process, we try to make a $|V|$ class classifier.
- It will be inefficient to compute when the graph is large.
- By using hierarchical softmax, we can reduce the time complexity to $O(\log(|V|))$



$$\text{Predicted probability} \approx (0.89 \cdot 0.33 \cdot 0.45) = 0.132$$

node2vec

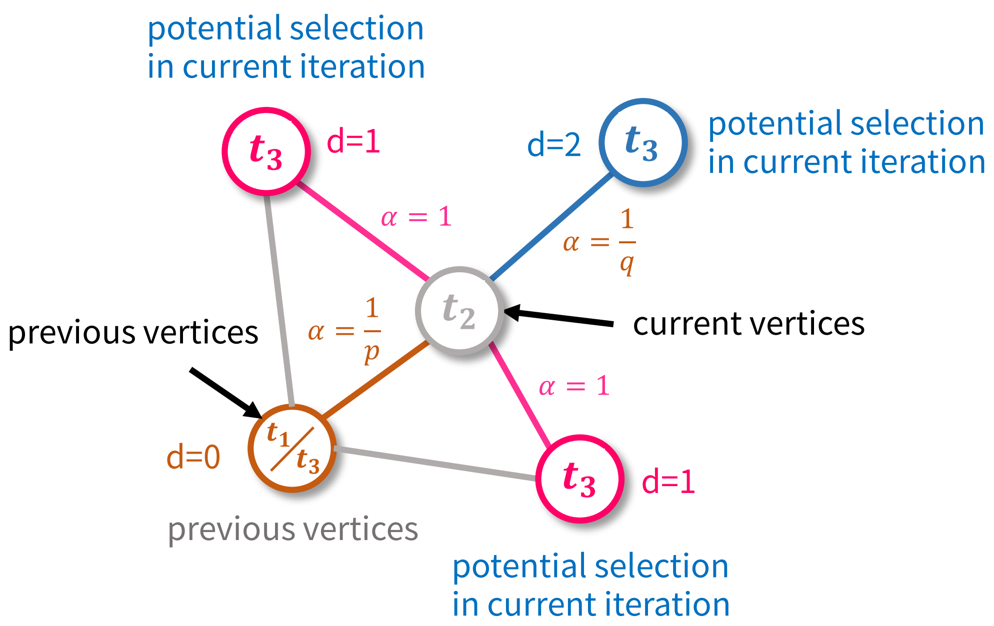
- Depth-first search can better capture the interconnection between vertices.
- Breadth-first search can better capture the structural equivalence.

$$\alpha(v_j|v_i) = \begin{cases} \frac{1}{p} & \text{if } d_{i,j} = 0 \\ 1 & \text{if } d_{i,j} = 1 \\ \frac{1}{q} & \text{if } d_{i,j} = 2 \end{cases}$$

- Use softmax with negative sampling instead of hierarchical softmax in the output layer.

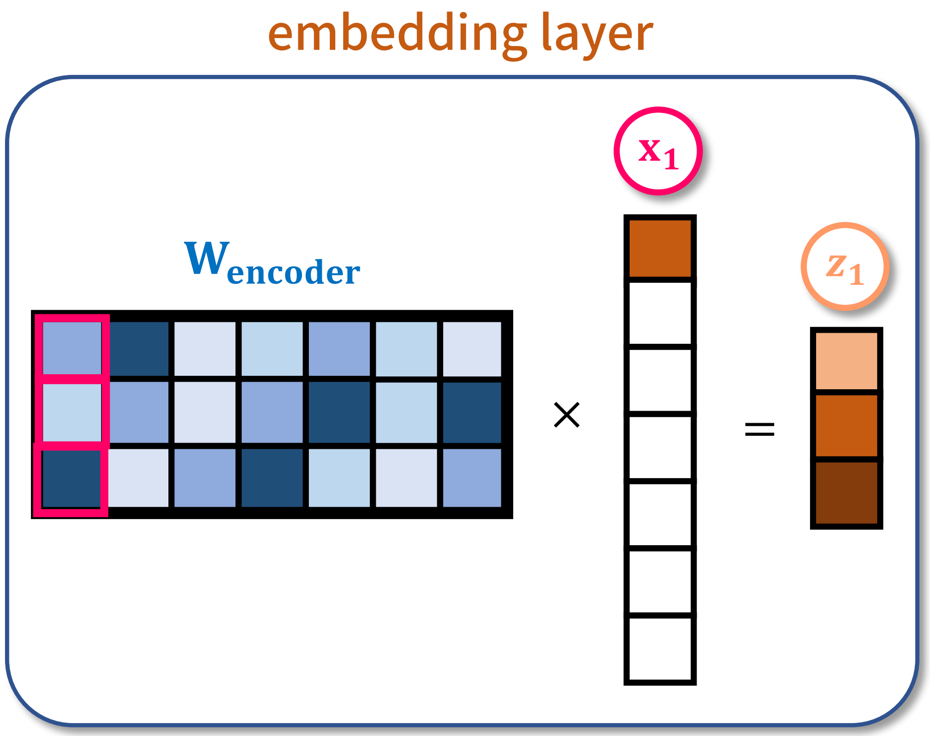
$$\text{softmax}(x_k) = \frac{\exp(x_k)}{\sum_{i \in NS} \exp(x_i)}$$

biased random walk



Limitations of DeepWalk and node2vec

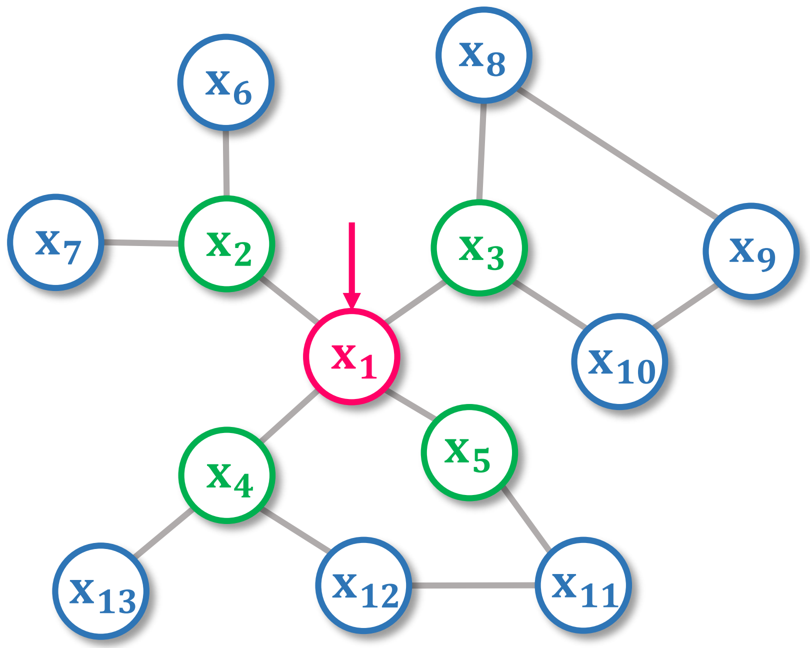
1. no parameter sharing



2. can not perform inductive learning

GraphSAGE (1)

Sampling

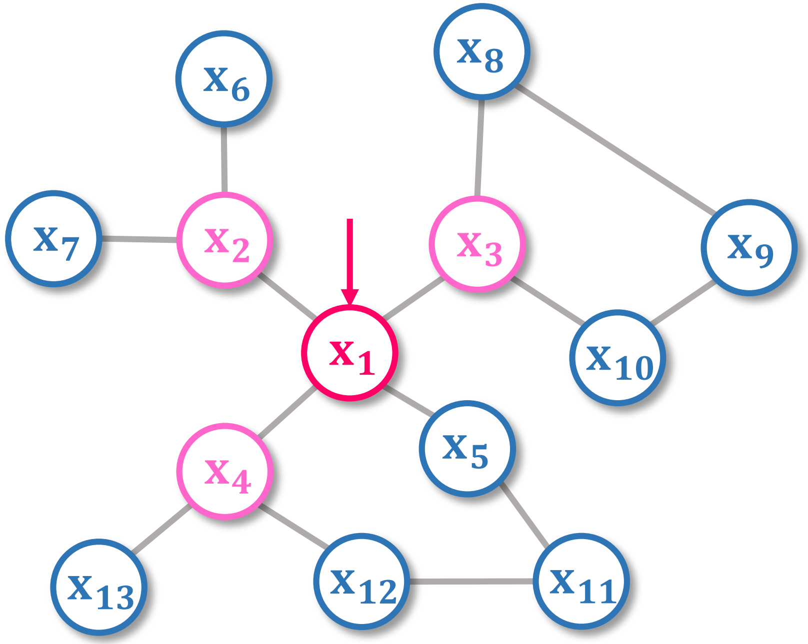


* hyperparameters

1. Sample *3* out of 1st order neighboring vertices.

GraphSAGE (2)

Sampling



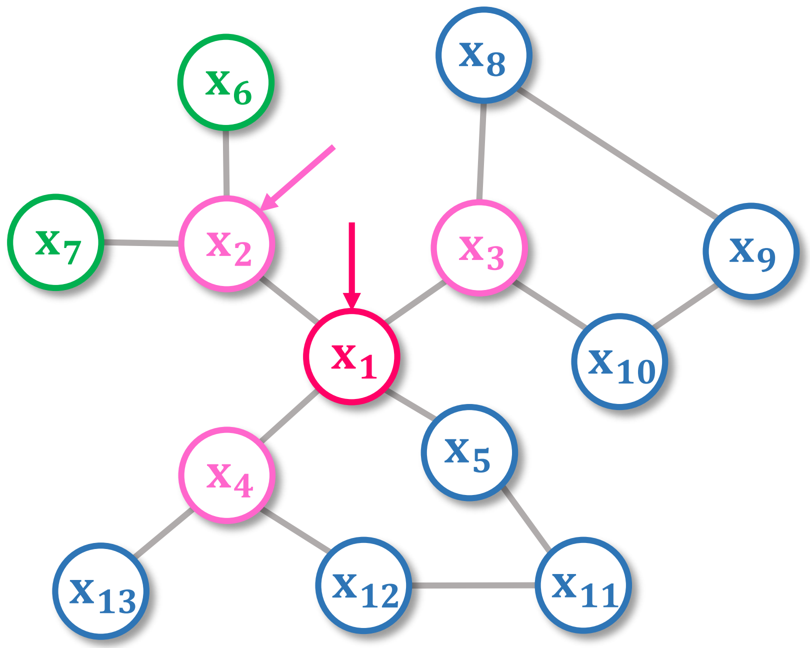
* hyperparameters

1. Sample *3* out of 1st order neighboring vertices.

$N_1^{(S)}(1): \{\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$

GraphSAGE (3)

Sampling



* hyperparameters

1. Sample 3 out of 1st order neighboring vertices.

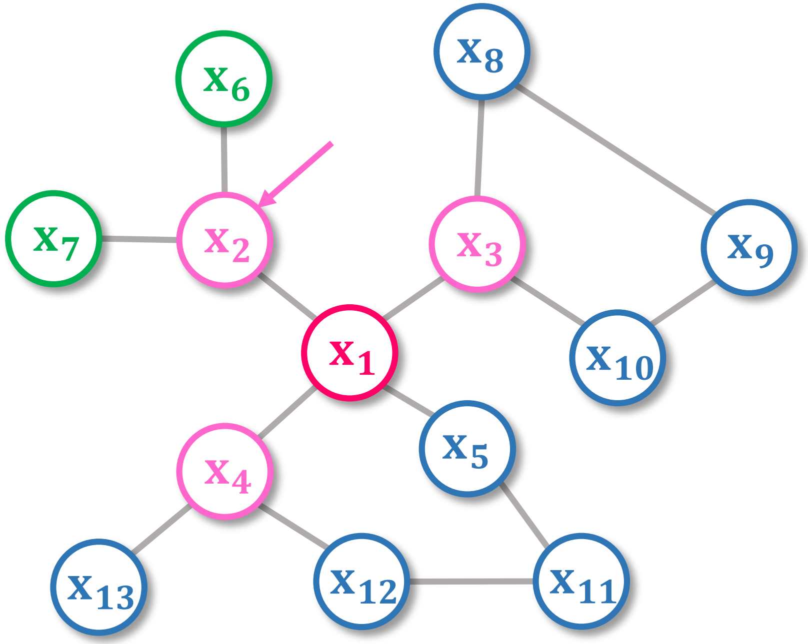
   $N_1^{(S)}(1): \{\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$

2. Sample $S_2 = 2$ out of the 1st order neighboring vertices for all vertices in $N_1^{(S)}(1)$.

   $N_1^{(S)}(2, \mathbf{x}_2): \{\mathbf{x}_6, \mathbf{x}_7\}$

GraphSAGE (4)

Sampling



* hyperparameters

1. Sample *3* out of 1st order neighboring vertices.

   $N_1^{(S)}(1)\colon \{\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$

2. Sample $S_2 = 2$ out of the 1st order neighboring vertices for all vertices in $N_1^{(S)}(1)$.

   $N_1^{(S)}(2, \mathbf{x}_2)\colon \{\mathbf{x}_6, \mathbf{x}_7\}$

   $N_1^{(S)}(2, \mathbf{x}_3)\colon \{\mathbf{x}_8, \mathbf{x}_{10}\}$

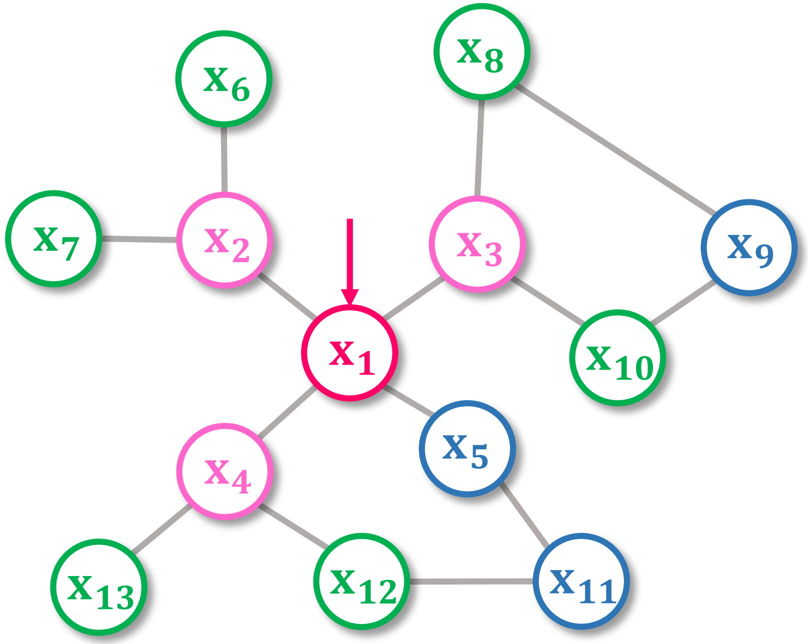
   $N_1^{(S)}(2, \mathbf{x}_4)\colon \{\mathbf{x}_{12}, \mathbf{x}_{13}\}$

   Continue until the $L = 2$-order of neighboring vetices for $\mathbf{x}_1$ has been selected.

*This process occur in the embedding layer, the skip-gram model is not changed

GraphSAGE (5)

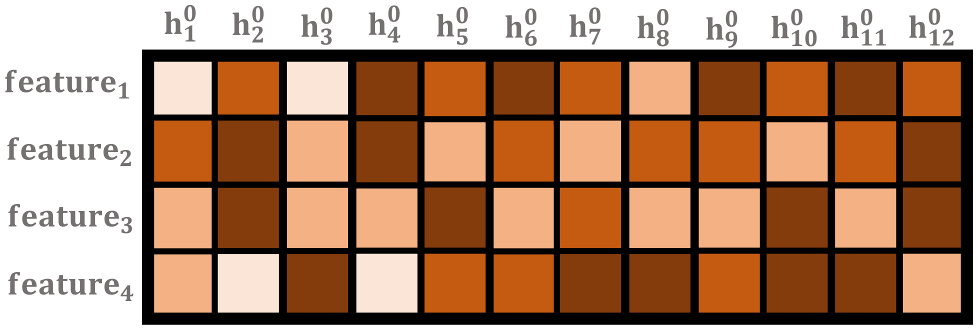
Aggregation



$N_1^{(S)}(1)$: $\{\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$

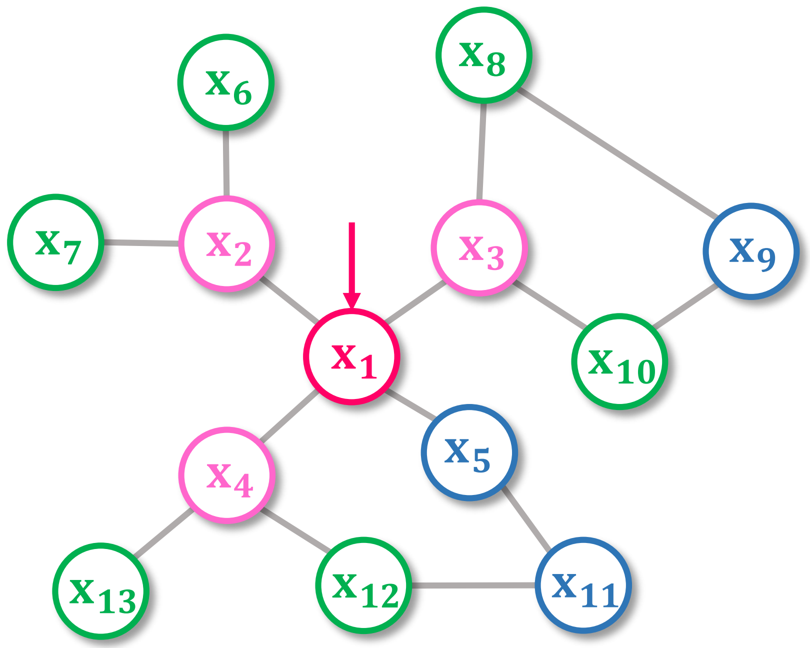
$N_1^{(S)}(2)$: $\{\mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_{10}, \mathbf{x}_{12}, \mathbf{x}_{13}\}$

1. Initialize the temporary embedding $\mathbf{h}_\mathbf{i}^\mathbf{0} = \mathbf{x}_\mathbf{i}$ for all vertices



GraphSAGE (6)

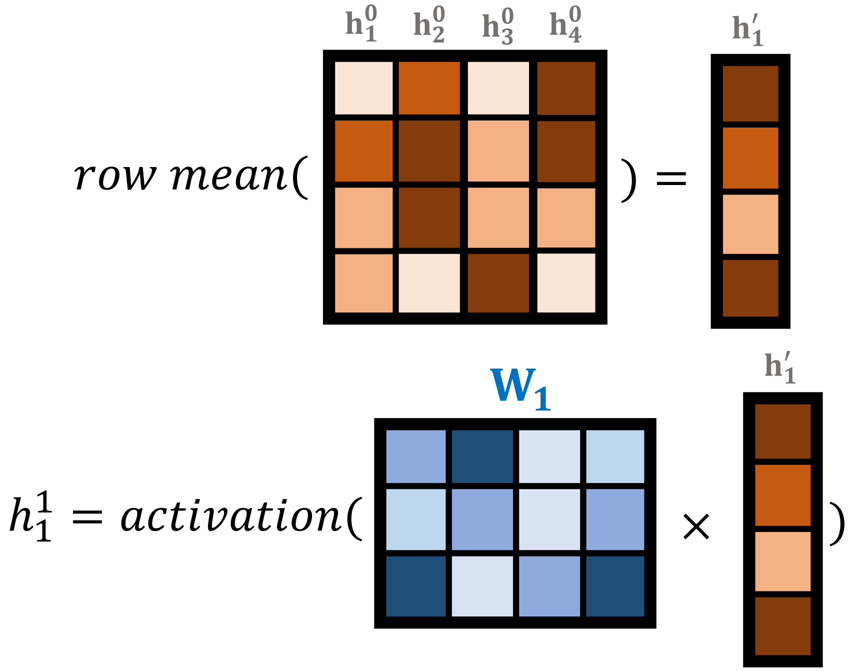
Aggregation



$N_1^{(S)}(1)$: $\{\mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}\}$

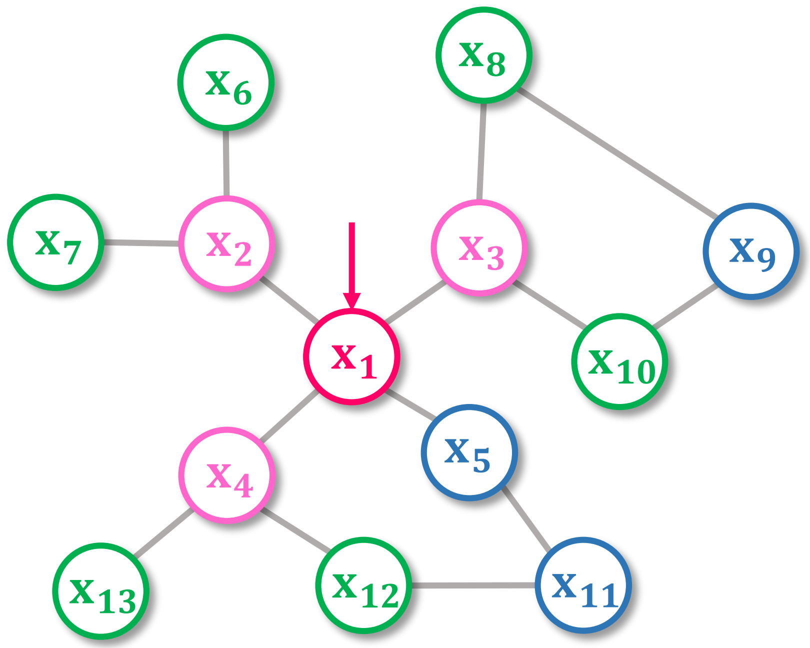
$N_1^{(S)}(2)$: $\{\mathbf{x_6}, \mathbf{x_7}, \mathbf{x_8}, \mathbf{x_{10}}, \mathbf{x_{12}}, \mathbf{x_{13}}\}$

1. Initialize the temporary embedding $\mathbf{h_i^0} = \mathbf{x_i}$ for all vertices
2. Aggregate 1st order neighboring vertices in $N_i^{(s)}(1)$ with h_i^0



GraphSAGE (7)

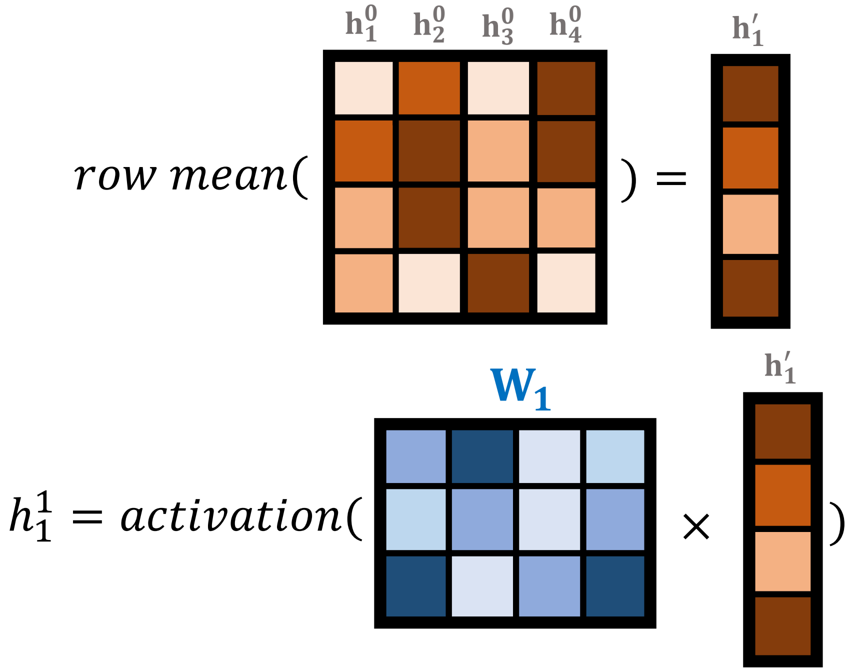
Aggregation



$N_1^{(S)}(1)$: $\{\mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}\}$

$N_1^{(S)}(2)$: $\{\mathbf{x_6}, \mathbf{x_7}, \mathbf{x_8}, \mathbf{x_{10}}, \mathbf{x_{12}}, \mathbf{x_{13}}\}$

1. Initialize the temporary embedding $\mathbf{h_i^0} = \mathbf{x_i}$ for all vertices
2. Aggregate 1st order neighboring vertices in $N_i^{(s)}(1)$ with h_i^0

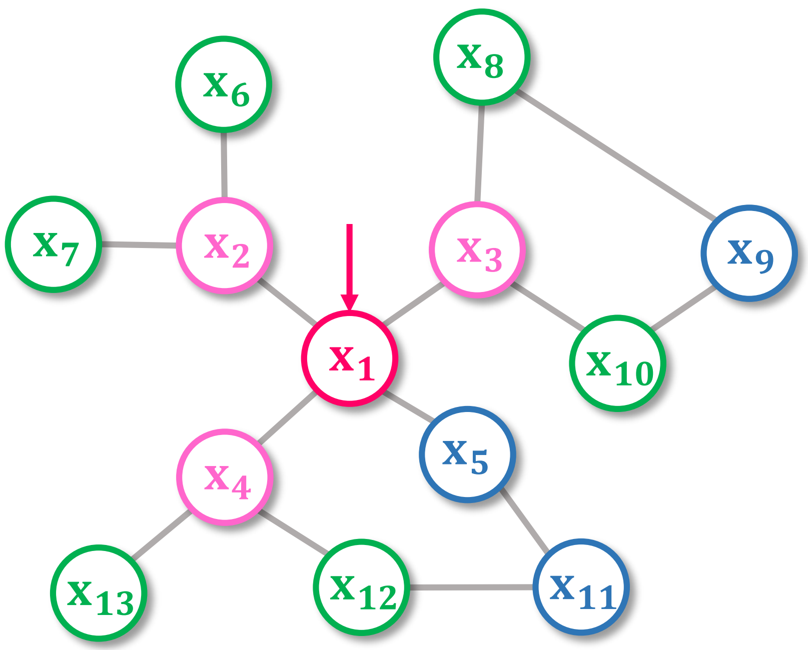


3. Continue the process, the resulting vector $z_i = h_i^L$

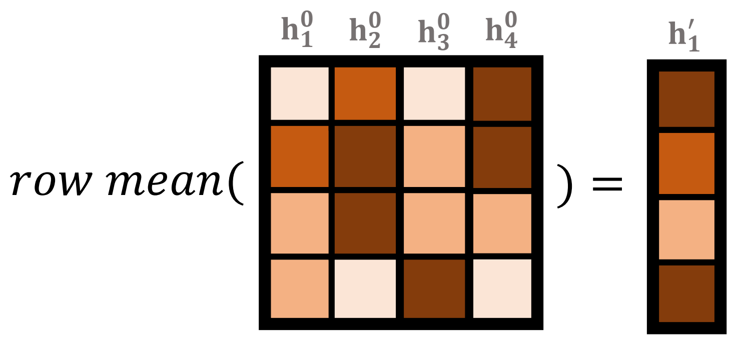
GraphSAGE (8)

Limitations

1. Not all the neighboring vertices are consider in each batch



2. The neighboring vertices are treated equally in the aggregation

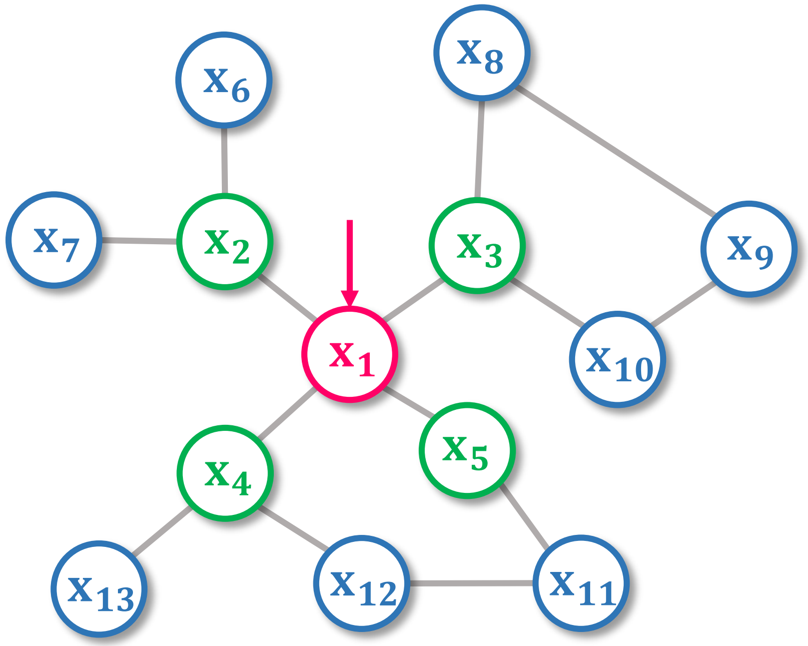


the weight for each column here is the same

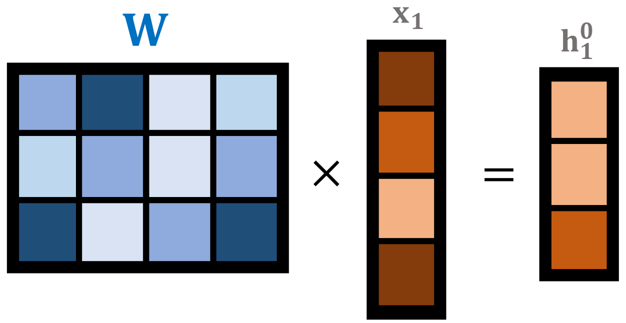
$$h'_{11} = \frac{1}{4}(h^0_{11} + h^0_{12} + h^0_{13} + h^0_{14})$$

Graph Attention Network (1)

Initialization

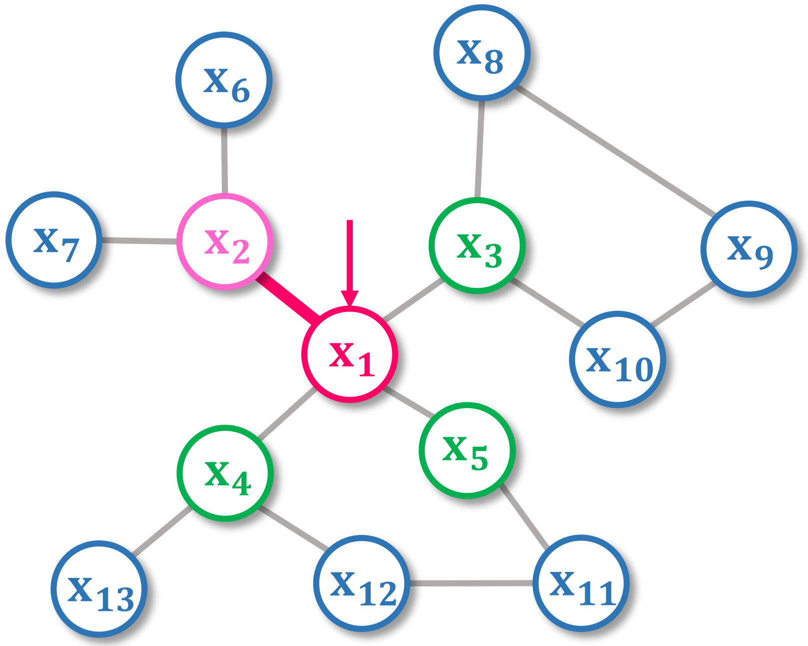


1. Initialize the temporary embedding $\mathbf{h}_\mathbf{i}^\mathbf{0} = \mathbf{W}\mathbf{x}_\mathbf{i}$ for all vertices

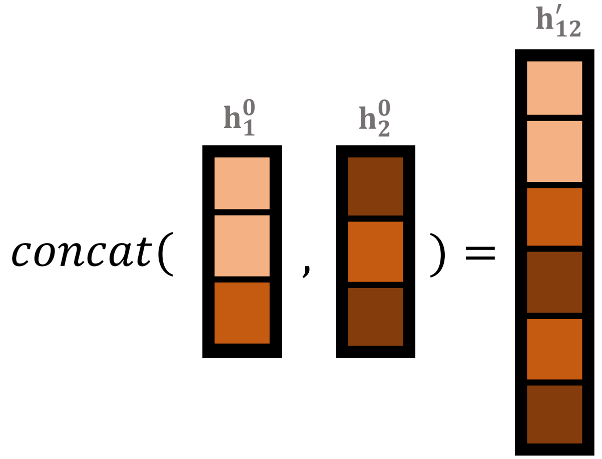


Graph Attention Network (2)

Attention Coefficient

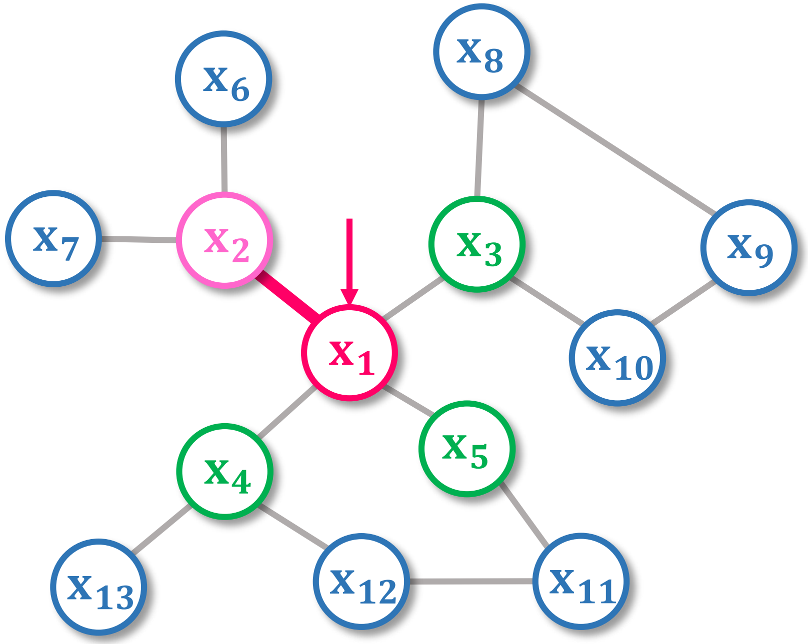


1. Initialize the temporary embedding $\mathbf{h_i^0} = \mathbf{Wx_i}$ for all vertices
2. Compute attention coefficient with neighboring vertices
    1. Concatenate h_i with h_j

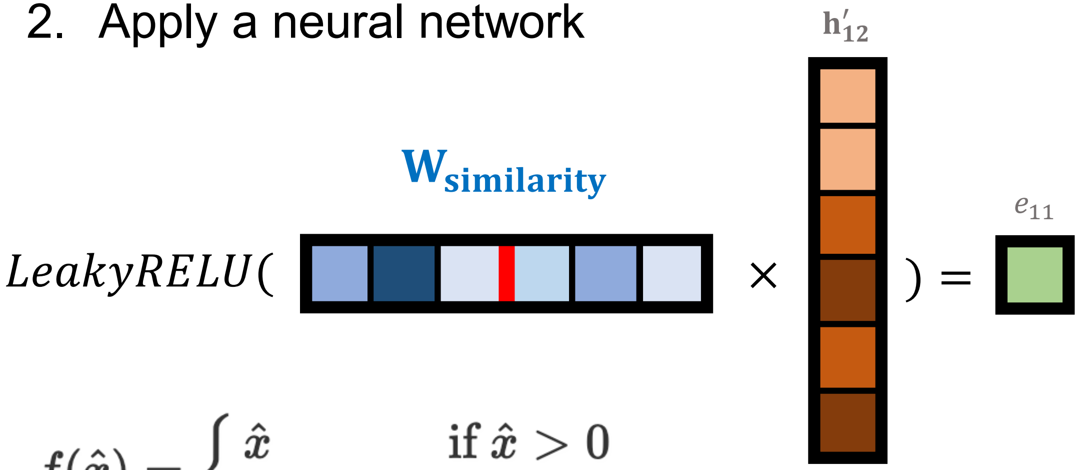


Graph Attention Network (3)

Attention Coefficient



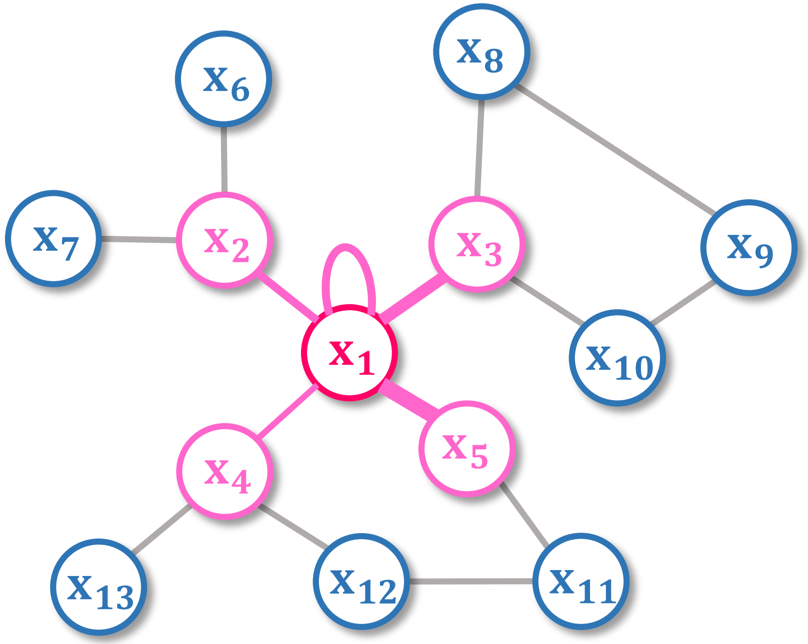
1. Initialize the temporary embedding $\mathbf{h_i^0} = \mathbf{Wx_i}$ for all vertices
2. Compute attention coefficient with neighboring vertices
   1. Concatenate h_i with h_j
   2. Apply a neural network



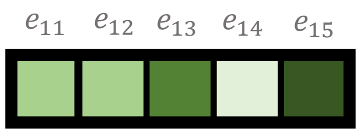
$$f(\hat{x}) = \begin{cases} \hat{x} & \text{if } \hat{x} > 0 \\ a\hat{x} & \text{otherwise} \end{cases}$$

Graph Attention Network (4)

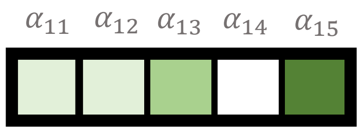
Attention Coefficient



1. Initialize the temporary embedding $\mathbf{h}_\mathbf{i}^\mathbf{0} = \mathbf{W}\mathbf{x}_\mathbf{i}$ for all vertices
2. Compute attention coefficient with neighboring vertices
   1. Concatenate h_i with h_j
   2. Apply a neural network
   3. Apply softmax

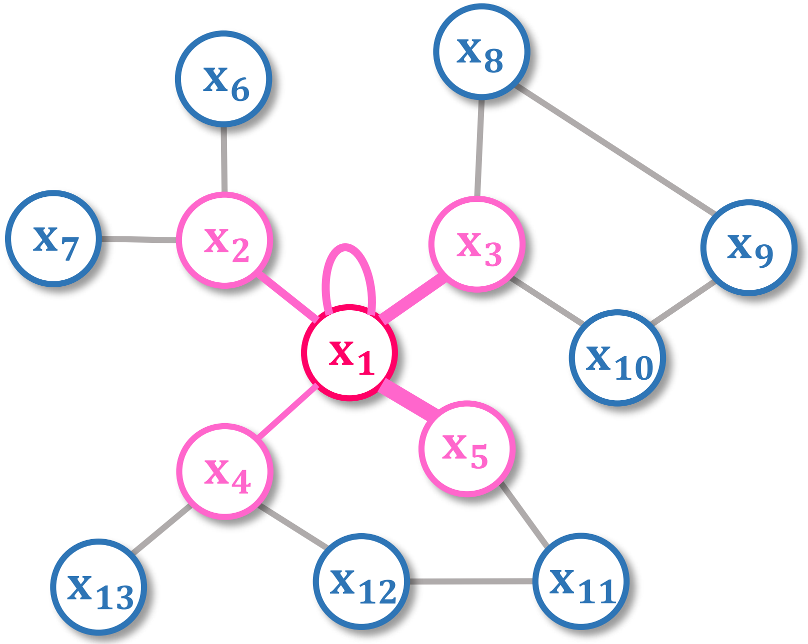


$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{u \in N} \exp(e_{iu})}$$

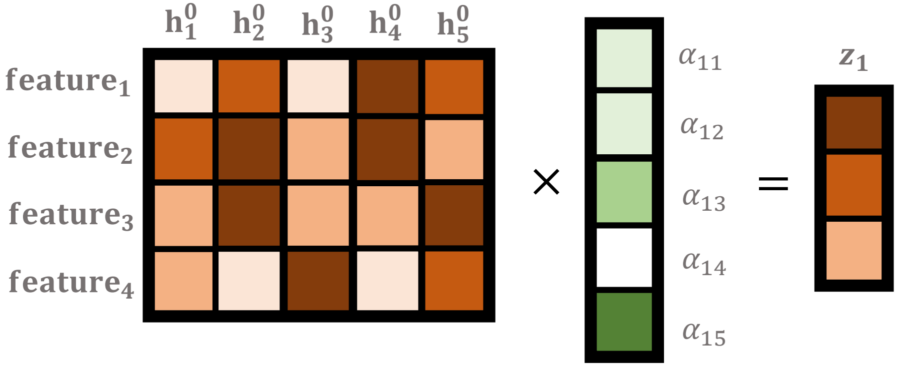


Step 2 and 3 uses a neural network to learn the best similarity function

Graph Attention Network (5)



1. Initialize the temporary embedding $\mathbf{h}_\mathbf{i}^\mathbf{0} = \mathbf{W}\mathbf{x}_\mathbf{i}$ for all vertices
2. Compute attention coefficient with neighboring vertices
3. Use the attention coefficient to aggregate neighboring vertices



Difference between GraphSAGE and GAT

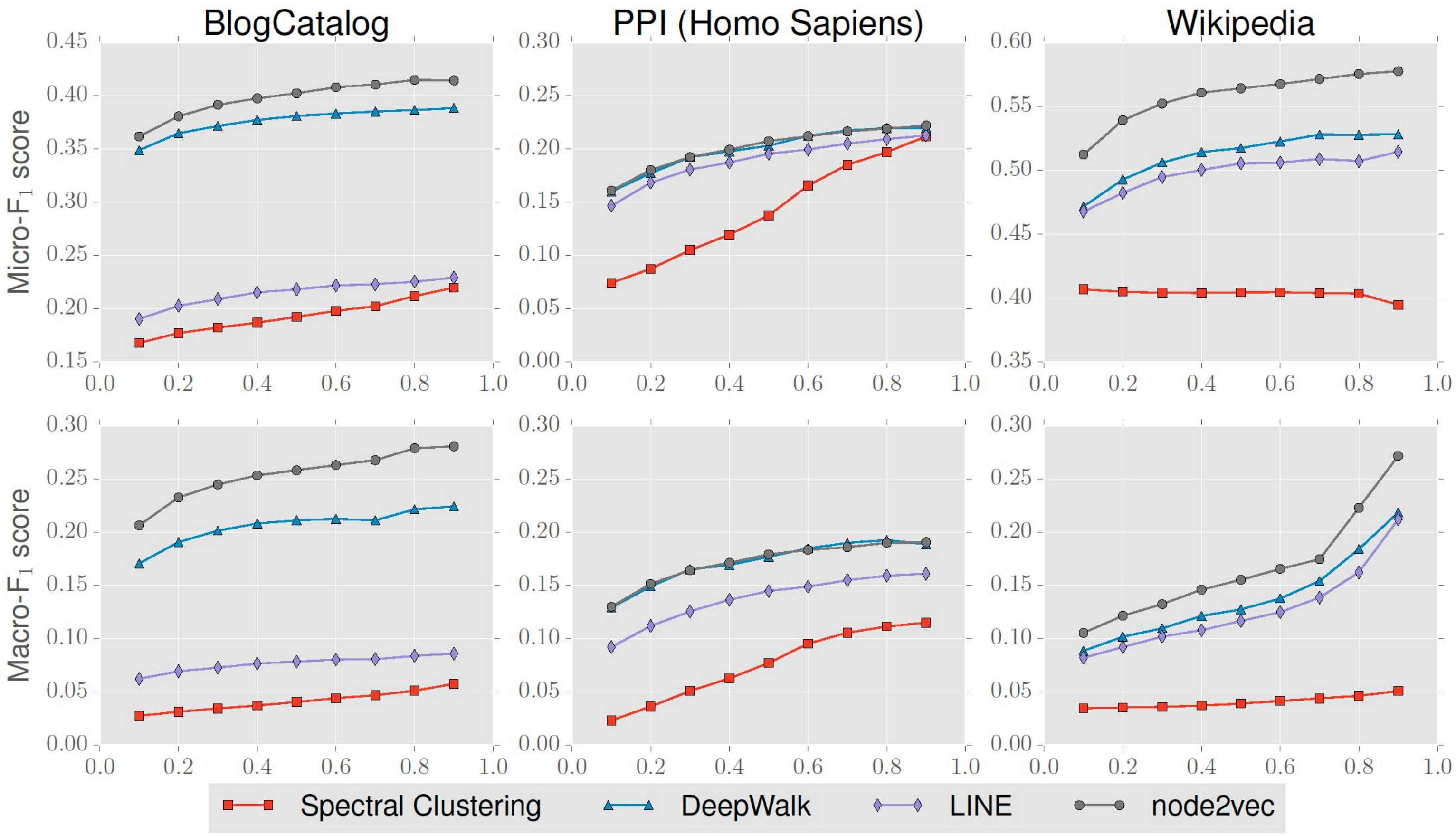
- GraphSAGE
  - $Aggregation \rightarrow Transformation$

$$\mathbf{h}^k_{\mathcal{N}(v)} \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}^{k-1}_u, \forall u \in \mathcal{N}(v)\});$$
$$\mathbf{h}^k_v \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}^{k-1}_v, \mathbf{h}^k_{\mathcal{N}(v)})\right)$$

- Graph Attention Network
  - $Transformation \rightarrow Aggregation\ (with\ attention)$

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \vec{h}_j\right)$$

Result



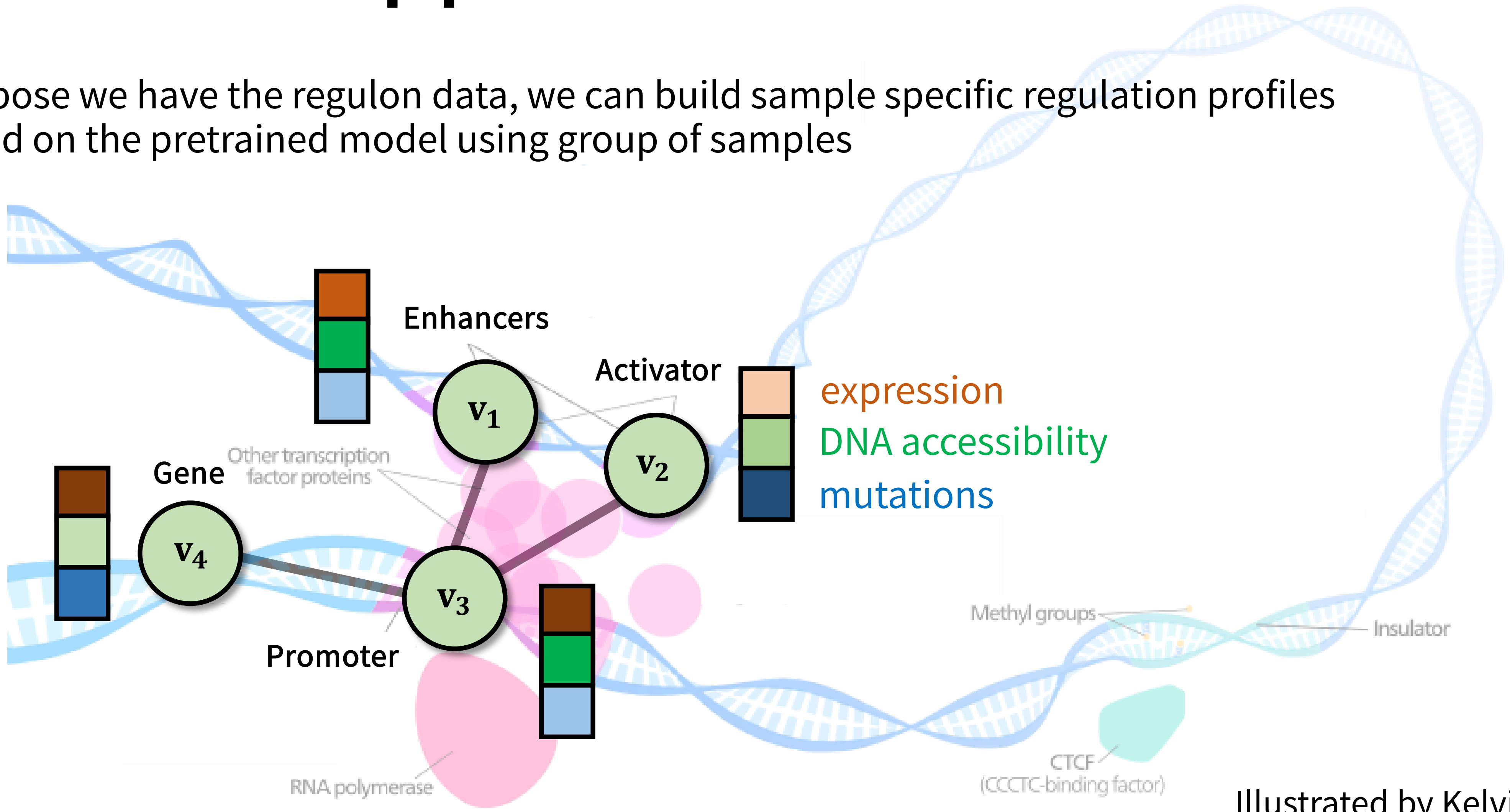
Methods	Micro F1
Random	0.396
MLP	0.422
GraphSAGE-GCN	0.500
GraphSAGE-Mean	0.598
GraphSAGE-LSTM	0.612
GraphSAGE-Pool	0.600
GraphSAGE*	0.768
Const-GAT	0.934
GAT	0.973

Advantage of Using Spatial Based Methods

- Suitable for inductive learning.
- The embedding are computed for each vertex. In other words. the algorithm does not depend on the global graph structure.
- The embedding layer can be fine-tuned for the down-stream analysis.
- Attention mechanism provides interpretability to the model.

Potential Applications

Suppose we have the regulon data, we can build sample specific regulation profiles based on the pretrained model using group of samples



Illustrated by Kelvin Ma

Recommended Publications

- Review papers
  - A Comprehensive Survey of Graph Embedding- Problems, Techniques and Applications. Zonghan *et al*. 2017. IEEE of Transactions on Knowledge and Data Enginnering.
  - Representation Learning on Graphs: Methods and Applications. Hamilton *et al*. 2018. IEEE Data Engineering Bulletin.

- Applications
  - To Embed or Not: Network Embedding as a Paradigm in Computational Biology. Nelson *et al*. 2019. Frontiers in Genetics.

- Gene regulatory network analysis framework
  - CEN-tools: An integrative platform to identify the contexts of essential genes. Sharma *et al.* 2020.

- Methods discussed in this presentation
  - DeepWalk: Online Learning of Social Representations. Perozzi *et al.* 2014. KDD
  - node2vec: Scalable Feature Learning for Networks. Grover *et al,* 2016. KDD.
  - Inductive Representation Learning on Large Graphs. Hamilton *et al*. 2017. NIPS.
  - Graph Attention Networks. Velickovic *et al,* 2018. ICLR.

- More recent works
  - Watch Your Step: Learning Node Embeddings via Graph Attention. Abu-El-Haija *et al*. 2018. NIPS.
  - AdaGCN: Adaboosting Graph Convolutional Networks into Deep Models. Ke Sun *et al*. 2019.
  - Bridging the Gap Between Spectral and Spatial Domains in Graph Neural Networks. Balcilar *et al.* 2020.

Thanks