Machine Learning on Graph

Network Embedding

Presented By

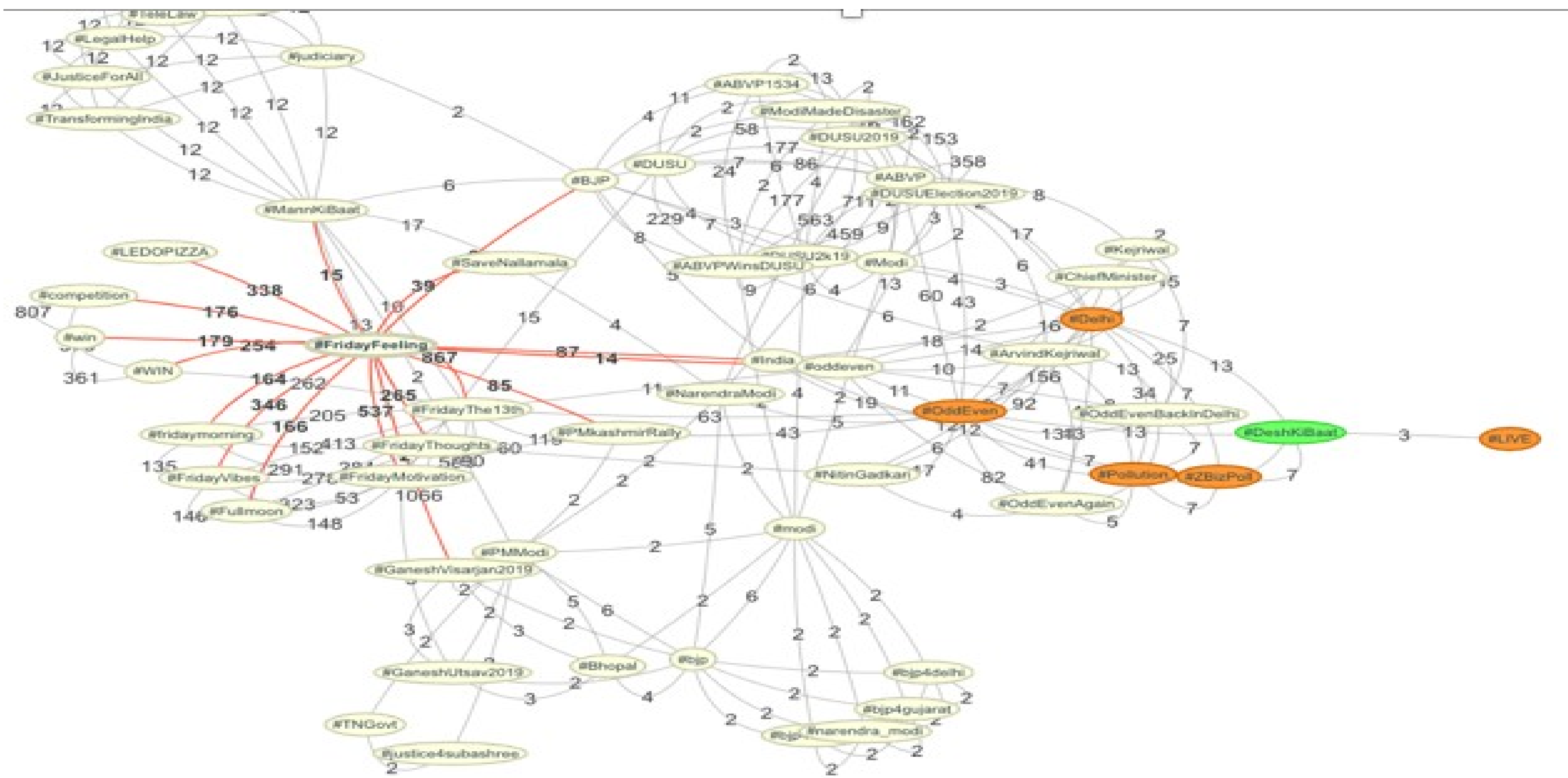
Sanasam Ranbir Singh

Open Source Intelligence Lab
Department of Computer Science and Engineering,
Indian Institute of Technology, Guwahati
Assam, INDIA
http://www.iitg.ernet.in/cseweb/osint/default/index

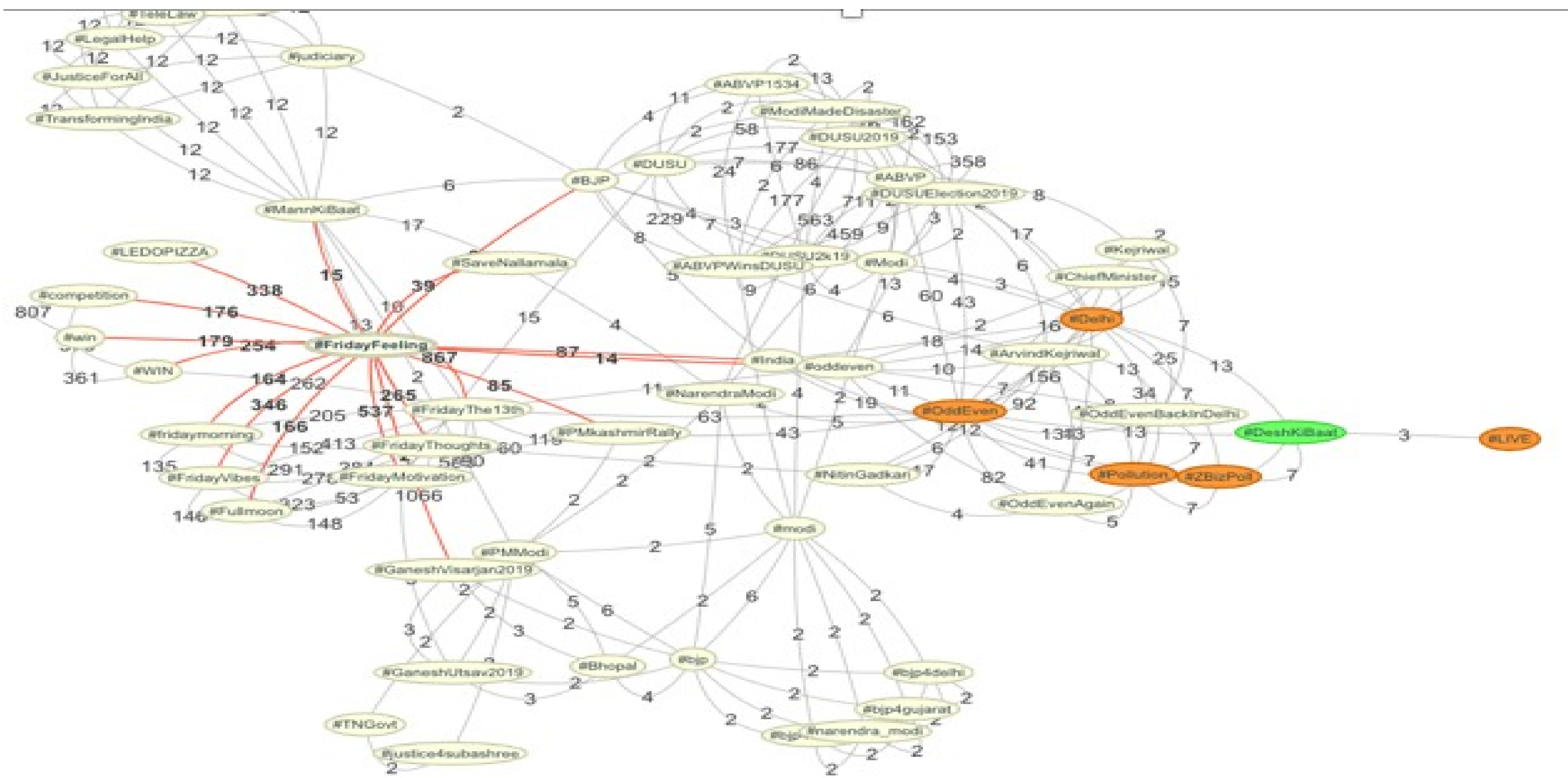
Facebook Friendship Network

Twitter Followers’ Network

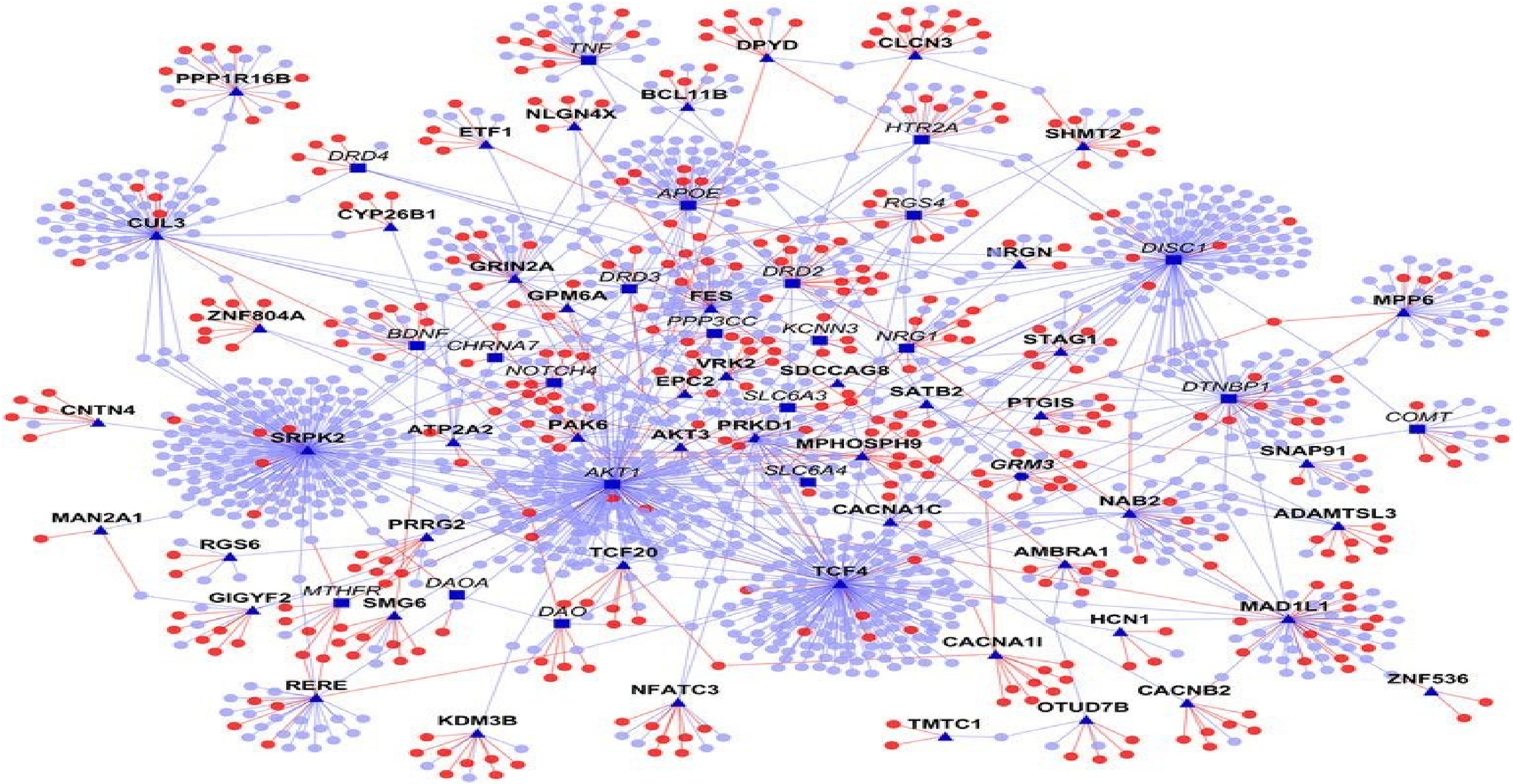
Hashtag Co-occurence Network



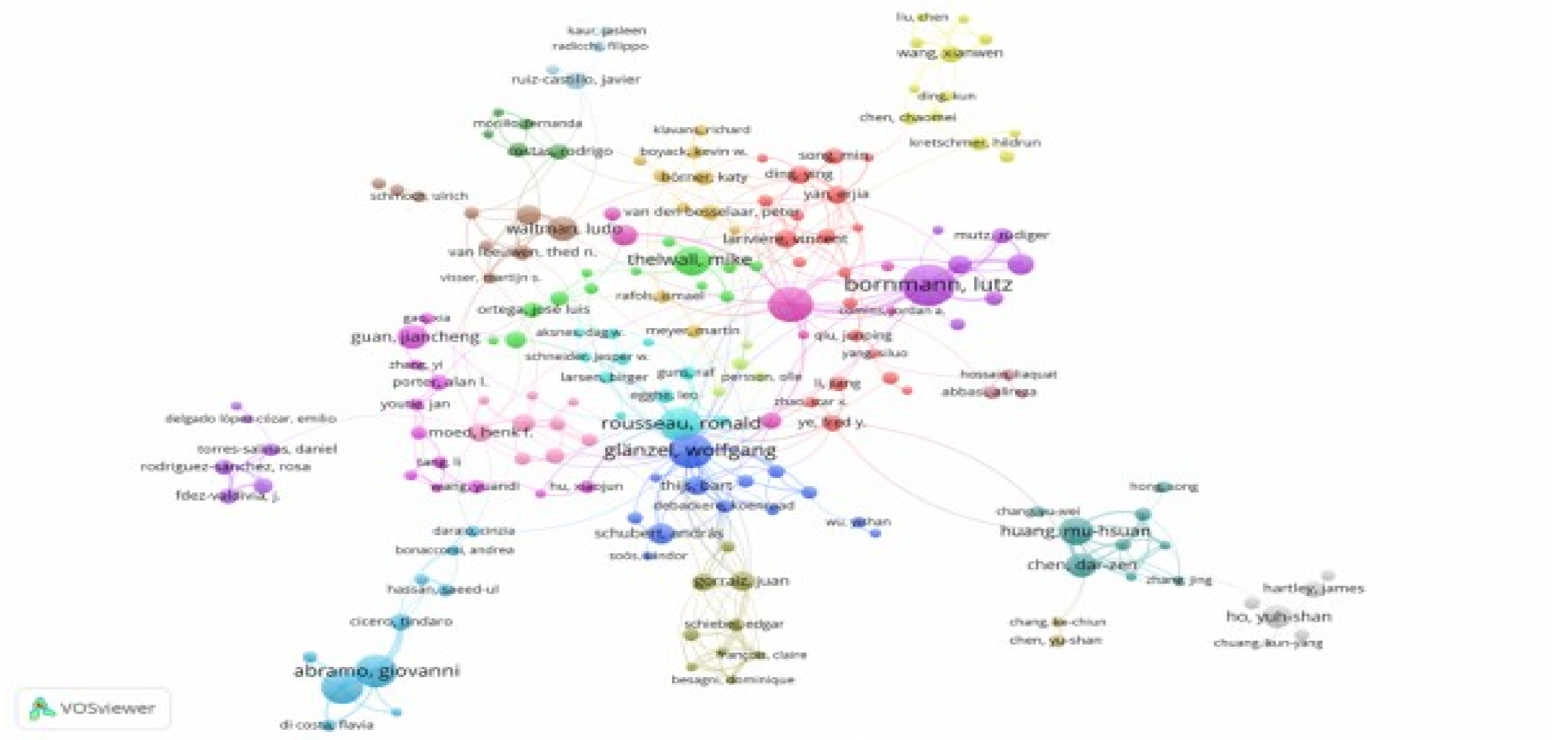
Hashtag Co-occurence Network



Protein-Protein Interaction Network

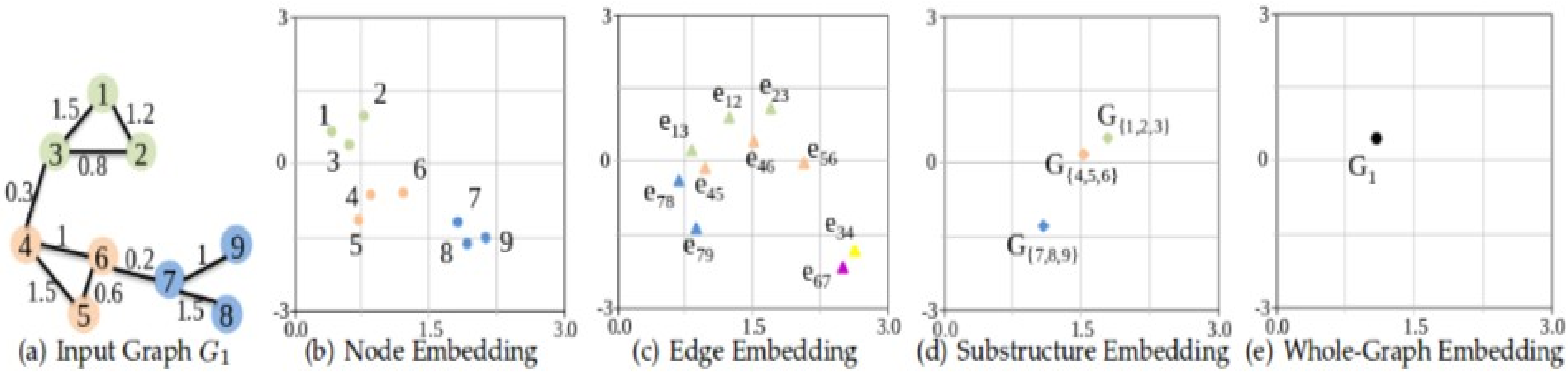


Co-Authorship Network



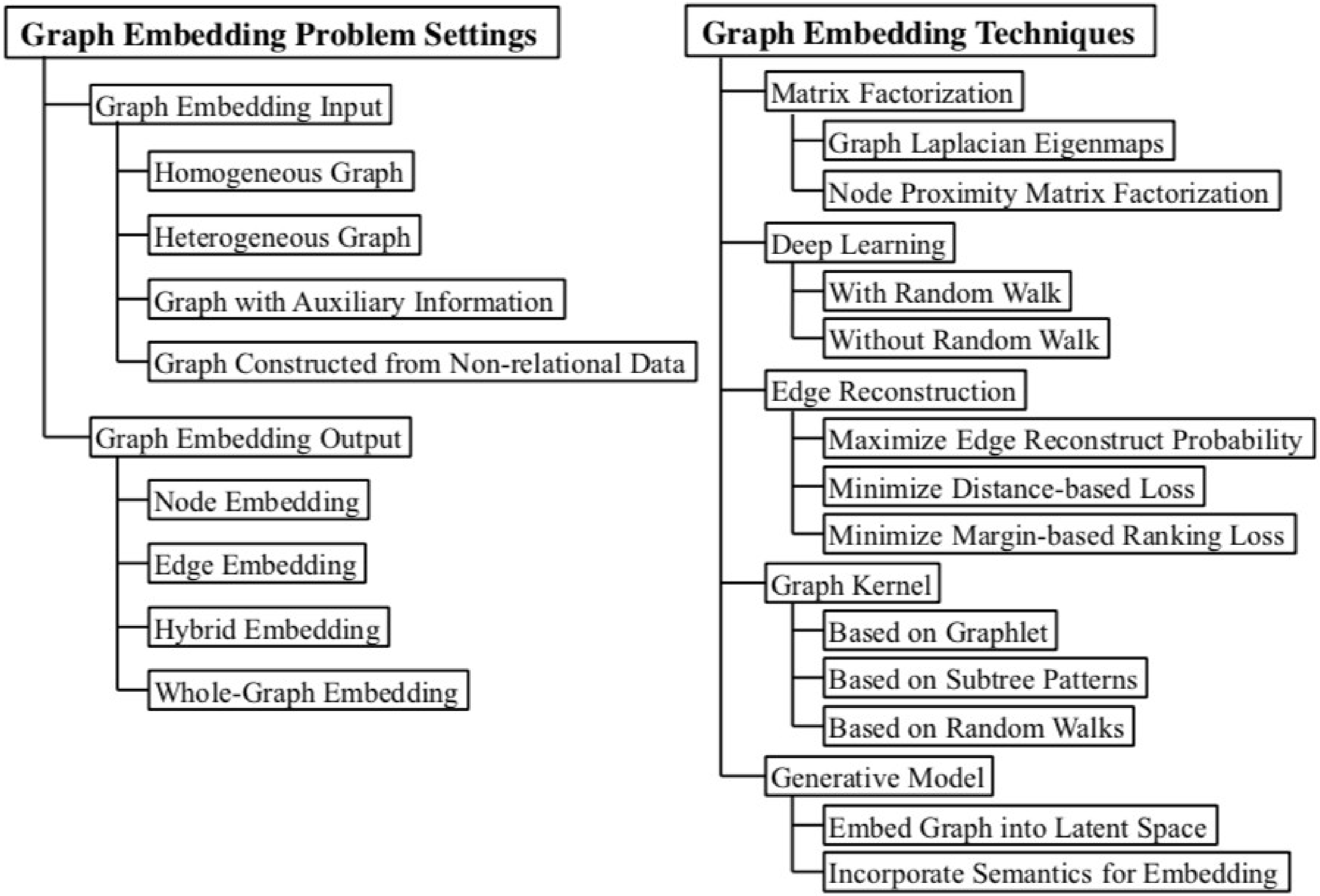
What is Network Embedding?

Suppose *G*(*V*,*E*) represents a network then Network Embedding refers to generating low dimensional network features corresponding to Nodes, Edges, Substructures, and the Whole-Graph.



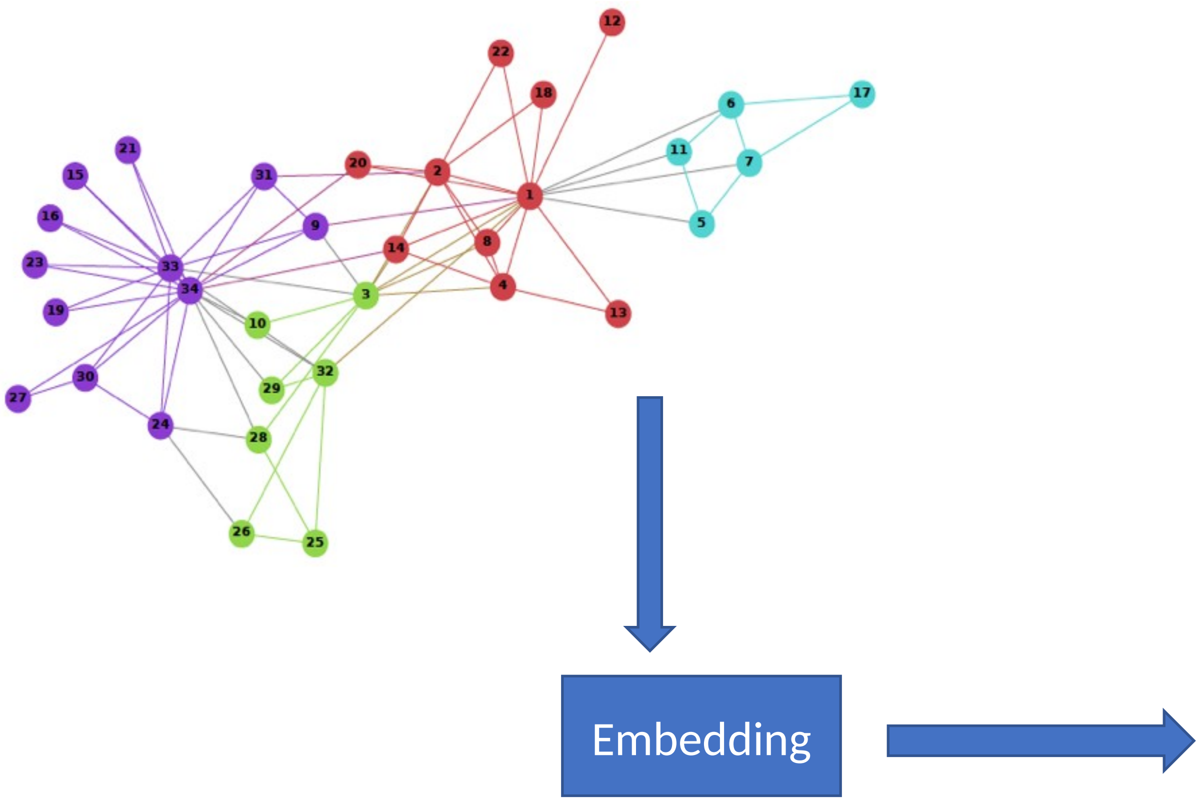
HongYun Cai, Vincent W. Zheng, Kevin Chen-Chuan Chang:**A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications.**IEEE Trans. Knowl. Data Eng. 30(9): 1616-1637 (2018)

Network Embedding - Texonomy



HongYun Cai, Vincent W. Zheng, Kevin Chen-Chuan Chang:**A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications.**IEEE Trans. Knowl. Data Eng. 30(9): 1616-1637 (2018)

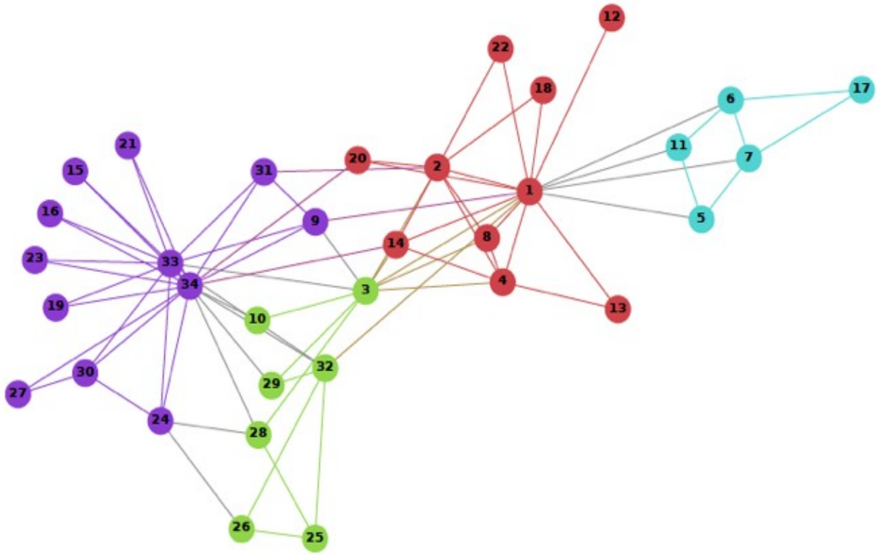
Node Embedding

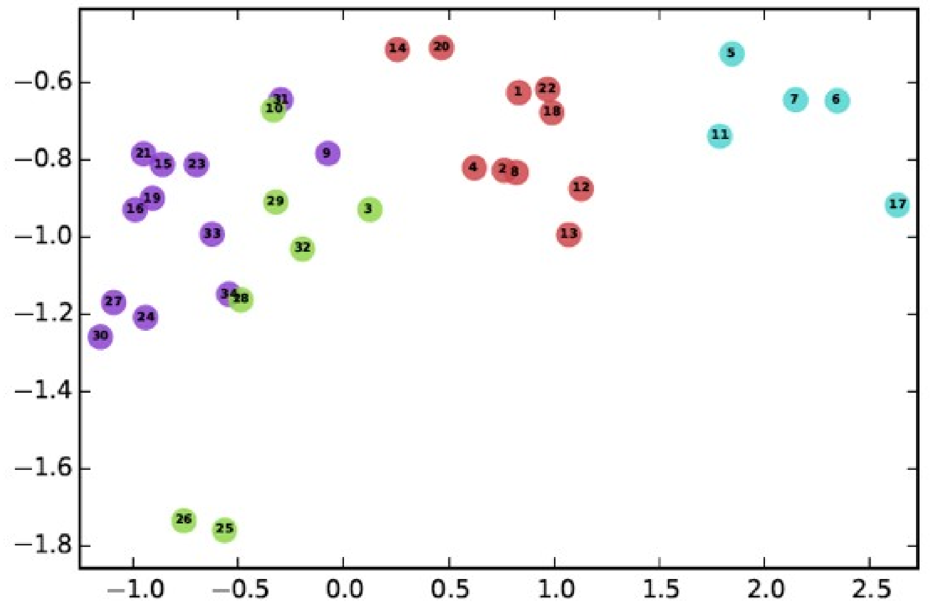



```
0   :  [1.98479516 2.104902  ]
1   :  [1.37277632 1.17311989]
2   :  [0.6859528  1.08576755]
3   :  [1.27303975 0.41918123]
4   :  [ 0.65722556 -0.36297315]
5   :  [ 0.71494817 -0.39592245]
6   :  [ 0.71494817 -0.39592245]
7   :  [ 1.04116841 -0.05324659]
8   :  [-0.27647113 -0.24436752]
9   :  [ 0.65722556 -0.36297315]
10   :  [ 0.37478198 -0.46299671]
11   :  [ 0.63250419 -0.30641371]
12   :  [ 0.66706122 -0.24109217]
13   :  [ 0.64664541 -0.38866438]
14   :  [ 0.27253823 -0.57650996]
15   :  [ 0.64664541 -0.38866438]
16   :  [-0.5209427  -0.58291863]
17   :  [-0.48328386 -0.17530251]
18   :  [-0.24844813 -0.38526576]
19   :  [-0.5175929  -0.23180349]
20   :  [-0.34294104 -0.09404636]
21   :  [-1.56457792  1.8504155 ]
22   :  [ 0.28271458 -0.28822466]
23   :  [-1.76282103  3.02650113]
24   :  [-0.70912996 -0.57803913]
25   :  [-0.70912996 -0.57803913]
26   :  [-0.70912996 -0.57803913]
27   :  [-0.70912996 -0.57803913]
28   :  [-0.70912996 -0.57803913]
29   :  [-1.08578976 -0.27525378]
30   :  [-0.37477941  0.06842669]
31   :  [-1.03412988 -0.27920574]
32   :  [-0.28044074  0.04244915]
33   :  [-0.58710262 -0.38879992]
```

Find embedding of nodes to d-dimensions so that "similar" nodes in the graph have embeddings that are close together.

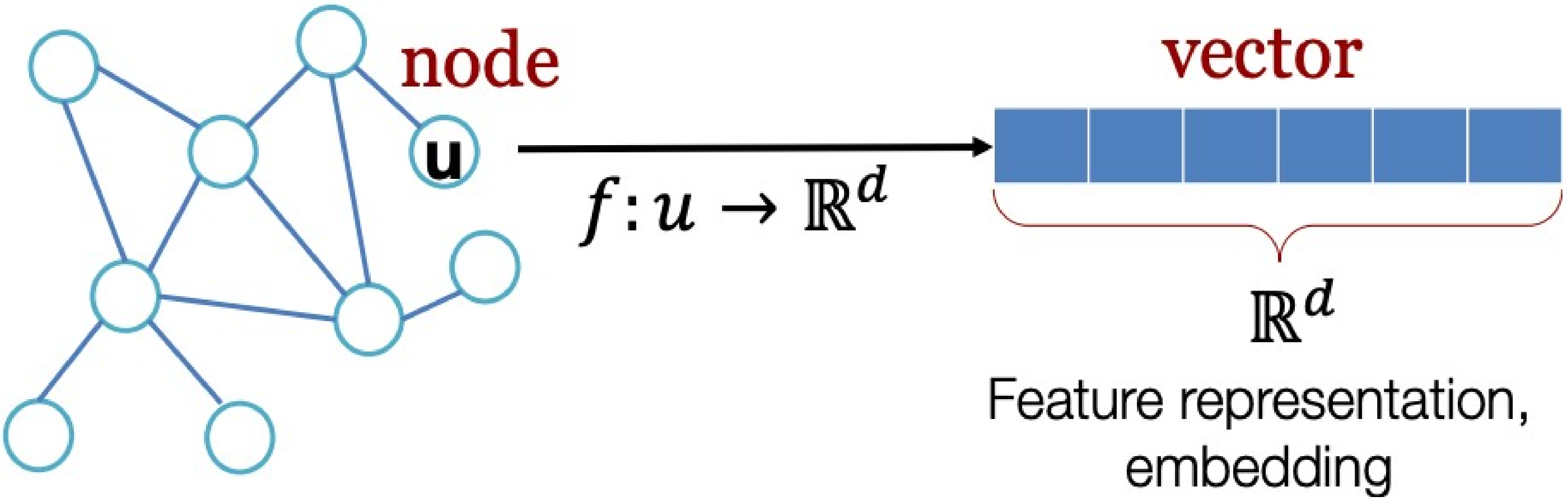
Node Embedding





Find embedding of nodes to d-dimensions so that “similar” nodes in the graph have embeddings that are close together.

Node Embedding



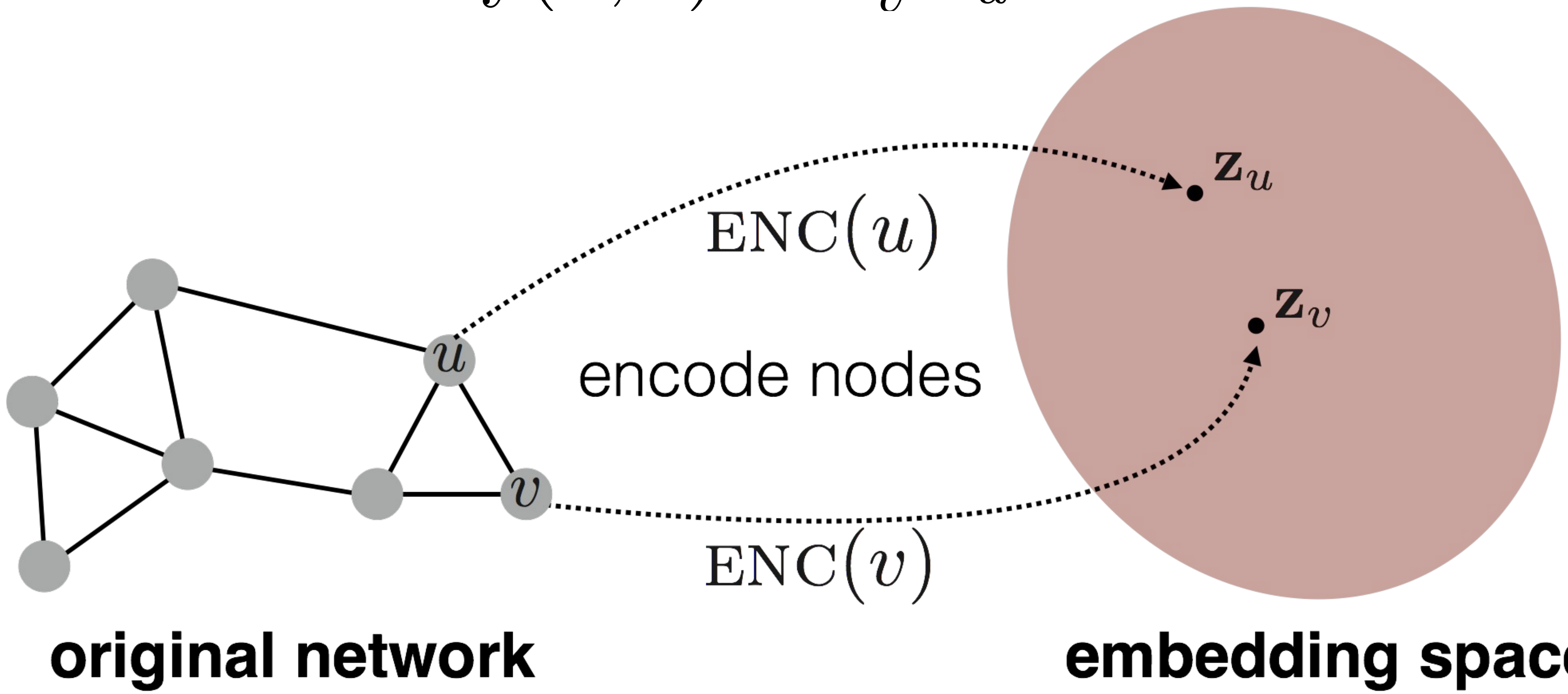
Find embedding of nodes to d-dimensions so that "similar" nodes in the graph have embeddings that are close together.

Node Embedding

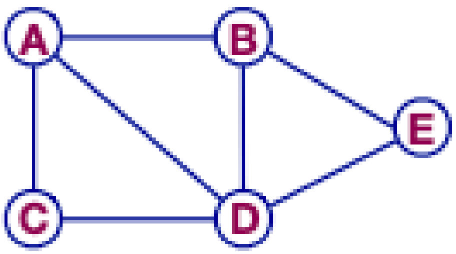
Goal is to encode nodes so that similarity in the embedding space (e.g., dot product)

approximates similarity in the original network.

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$



Vectorization of Nodes



$$
\begin{array}{c}
\\ A \\ B \\ C \\ D \\ E
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} A & B & C & D & E & \text{Feat} & \end{array} \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 & 1 & 0 \\
\boxed{1 \; 0 \; 0 \; 1 \; 1 \; 0 \; 0} \\
1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0
\end{pmatrix}
\end{array}
$$

Many possible ways to create node features:

- Node degree, PageRank score, motifs, …
- Degree of neighbors, PageRank of neighbors, …

Learning Node Embedding

1. **Define an encoder :**

$$\mathrm{ENC}(v) = \mathbf{z}_v$$

d-dimensional embedding

2. **Define a node similarity function** in the original network:

similarity(u,v)

Similarity of u and v in the original network

3. **Optimize the parameters of the encoder so that:**

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Encoding

Once we find encoding matrix Z, encoding of a node v can be defined as

$$\mathrm{ENC}(v) = \mathbf{Z}\mathbf{v}$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is node embedding

$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node *v*

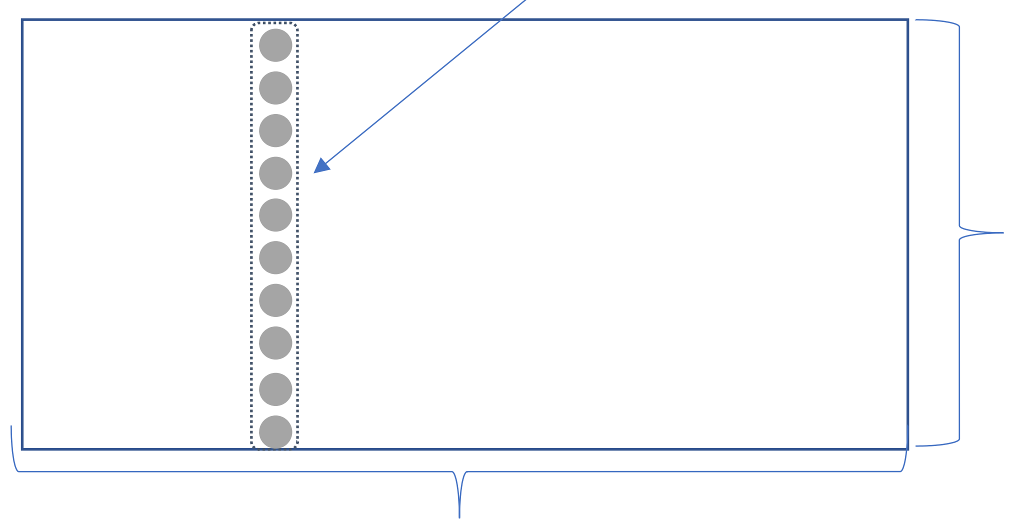
Encoding

Once we find encoding matrix Z, encoding of a node v (one hot vector) can be defined as

$$\mathrm{ENC}(v) = \mathbf{Z}\mathbf{v}$$

embedding matrix

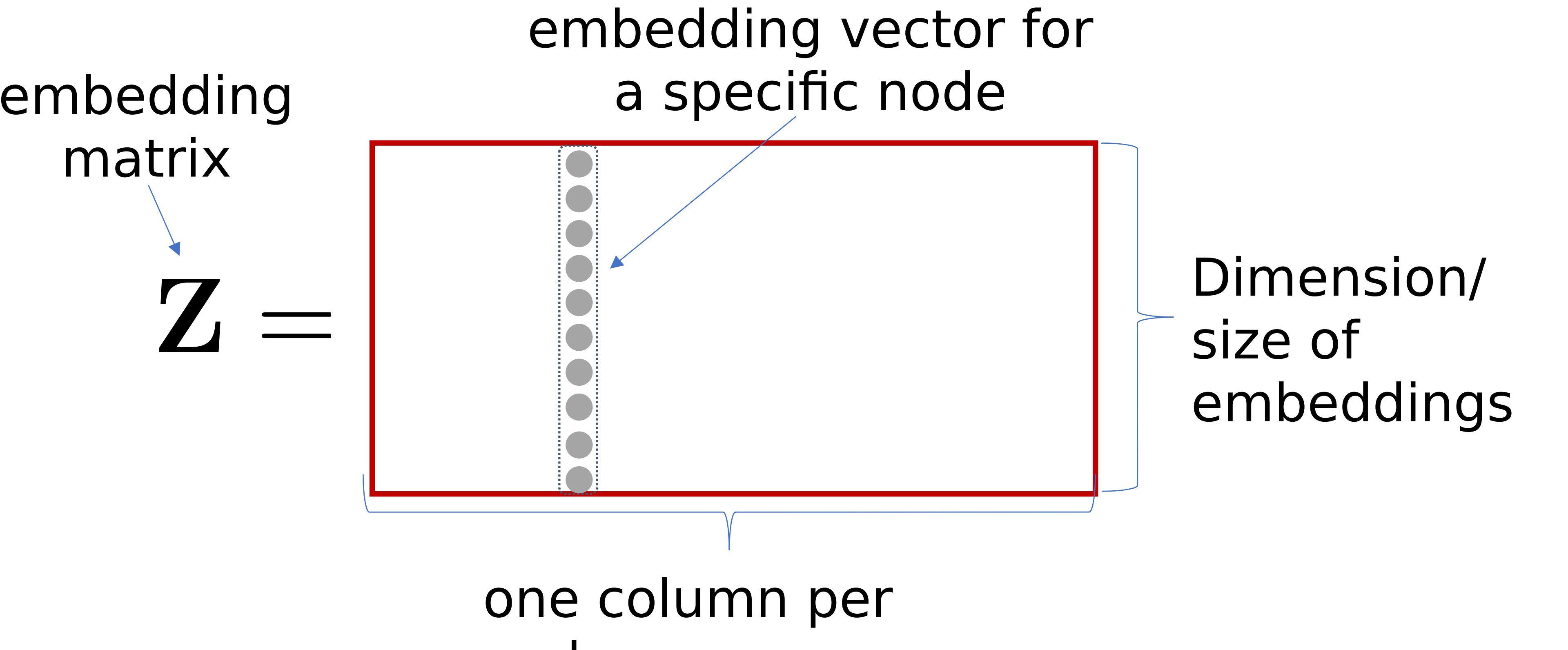
$\mathbf{Z} =$



Learning Node Embedding

Learning node embedding is to learn the matrix Z

$$\mathrm{ENC}(v) = \mathbf{Z}\mathbf{v}$$



Three ways of Learning Z

1. Adjacency-based similarity
2. Multi-hop similarity
3. Deep Learning
   - Unsupervised Approach(Random walk)
     - DeepWalk
   - Supervised Approach
     - GNN
     - GraphSAGE
     - Graph-BERT
     - GCN

Adjacency-based similarity

Let us assume that A is the adjacency matrix and two nodes are connected if they are same

Then, the loss function is

$$\boxed{\mathcal{L}} = \boxed{\sum_{(u,v) \in V \times V}} \| \boxed{\mathbf{z}_u^\top \mathbf{z}_v} - \boxed{\mathbf{A}}_{u,v} \|^2$$

loss (what we want to minimize)

sum over all node pairs

embedding similarity

(weighted) adjacency matrix for the graph

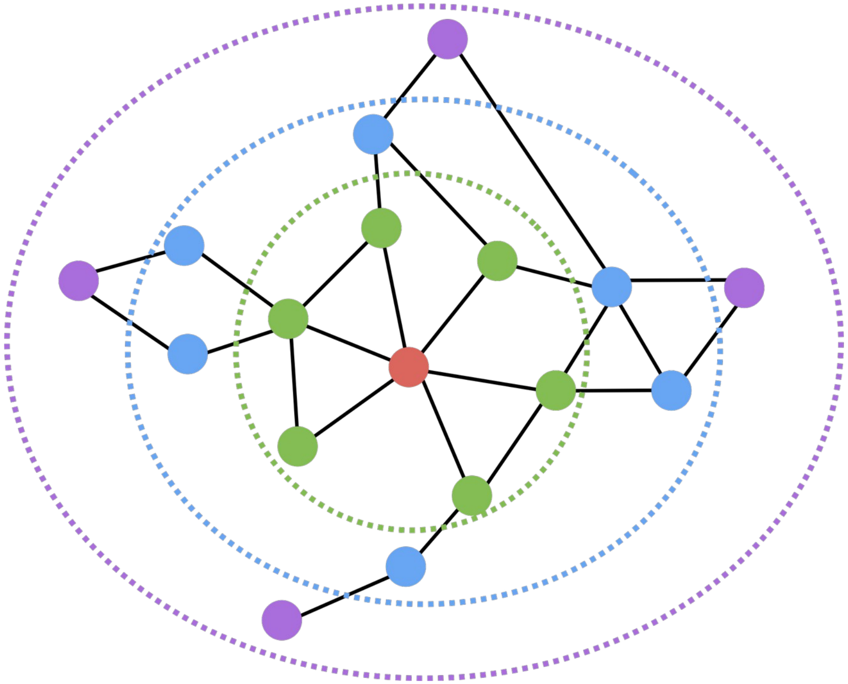
Two approaches

$$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}\|^2$$

- Find embedding matrix that minimizes the loss
    - Option 1: Stochastic gradient descent (SGD).
    - Option 2: Matrix decomposition(SVD).

Multi-Hop Similarity

- **Idea:** Consider k-hop node neighbors.
  - E.g., two or three-hop neighbors.



- **Red:** Target node
- **Green**: 1-hop neighbors
  - $\mathbf{A}$ (i.e., adjacency matrix)
- **Blue**: 2-hop neighbors
  - $\mathbf{A}^2$
- **Purple:** 3-hop neighbors
  - $\mathbf{A}^3$

Katz link prediction method

Multi-Hops Similarity Embedding

$$\mathcal{L} = \sum_{(u,v)\in V\times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{S}_{u,v}\|^2$$

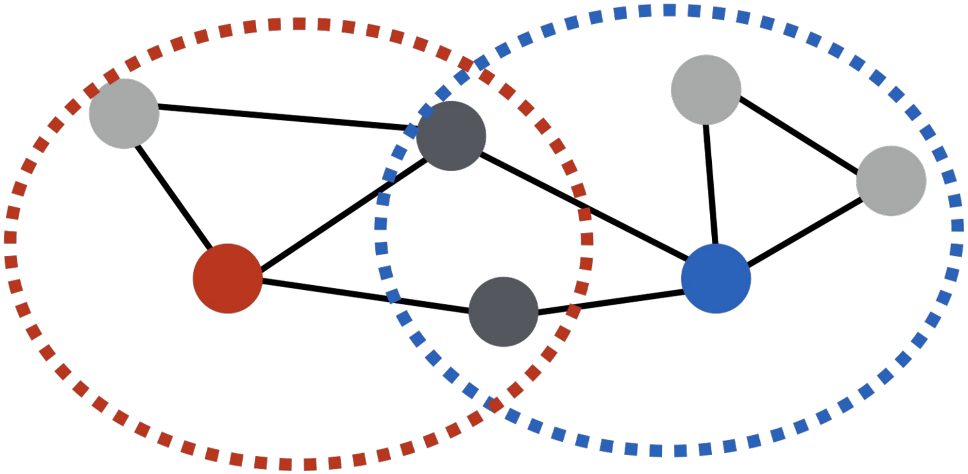
embedding similarity

multi-hop network similarity (i.e., any neighborhood overlap measure)

$\mathbf{S}_{u,v}$ is the neighborhood overlap between u and v.

HOPE (Yan et al., 2016)

- **Another option: Measure overlap between node neighborhoods.**



- Example overlap functions:
  - Common Neighbours
  - Jaccard Co-efficient
  - Adamic-Adar score

GraRep from Cao et al, 2015

- **Basic idea:** $$\mathcal{L} = \sum_{(u,v) \in V \times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}^k\|^2$$

- Train embeddings to predict k-hop neighbors.
- Probabilistic Adjacency Matrix:
  - Use log-transformed, probabilistic adjacency matrix:

$$\tilde{\mathbf{A}}_{i,j}^k = \max\left(\log\left(\frac{(\mathbf{A}_{i,j}/d_i)}{\sum_{l \in V}(\mathbf{A}_{l,j}/d_l)^k}\right)^k - \alpha, 0\right)$$

node degree

constant shift

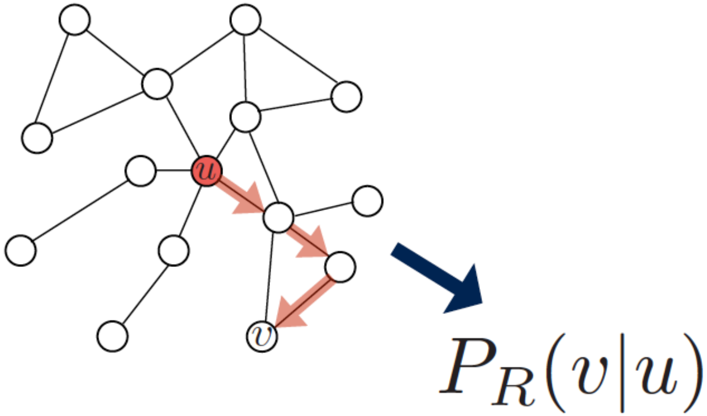
  - Train multiple different hop lengths and concatenate output.

Unsupervised: Random Walk Approach

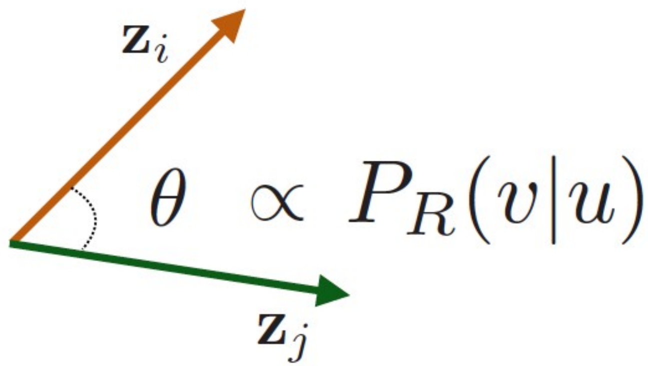
$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$ probability that u and v co-occur on a random walk over the network

Random Walk Approach

1. Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R.



2. Optimize embeddings to encode these random walk statistics.



Random Walk Optimization

1. Run short random walks starting from each node on the graph using some strategy R.
2. For each node u collect $\mathrm{N}_R(u)$ node sequence visited on random walks starting from u.
3. Optimize embeddings to according to:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

Random Walk Optimization

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

- **Intuition:** Optimize embeddings to maximize likelihood of random walk co-occurrences.
- **Parameterize** $\mathrm{P}(v \mid \mathbf{z}_u)$ **using softmax:**

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}$$

Random Walk Optimization

Putting things together:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

Optimizing random walk embeddings =

Finding embeddings z_u that

Random Walk Optimization

Solution: Negative sampling

$$\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n\in V}\exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k} \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

sigmoid function

random distribution over all nodes

i.e., instead of normalizing w.r.t. all nodes, just normalize against k random "**negative samples**"

Random Walk

- **Just run fixed-length, unbiased random walks starting from each node** (i.e., DeepWalk, Perozzi et al., 2013).

- Use flexible, biased random walks that can trade off between **local** and **global** views of the network (i.e., **Node2Vec,** Grover and Leskovec, 2016)

How should we randomly walk?

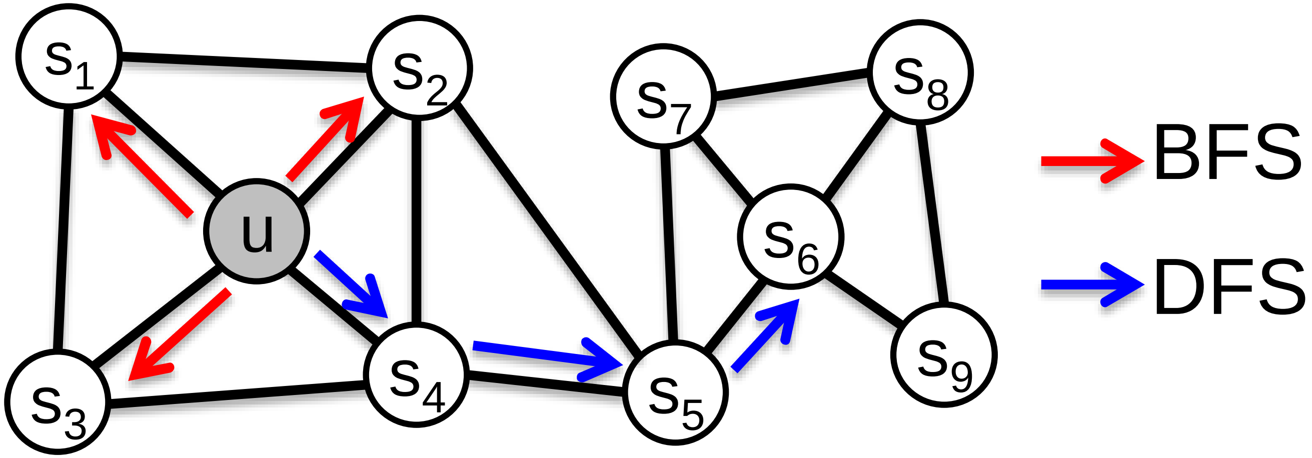
- So far we have described how to optimize embeddings given random walk statistics.
- **What strategies should we use to run these random walks?**
  - Simplest idea: **Just run fixed-length, unbiased random walks starting from each node** (i.e., DeepWalk from Perozzi et al., 2013).
  - But can we do better?

Node2vec

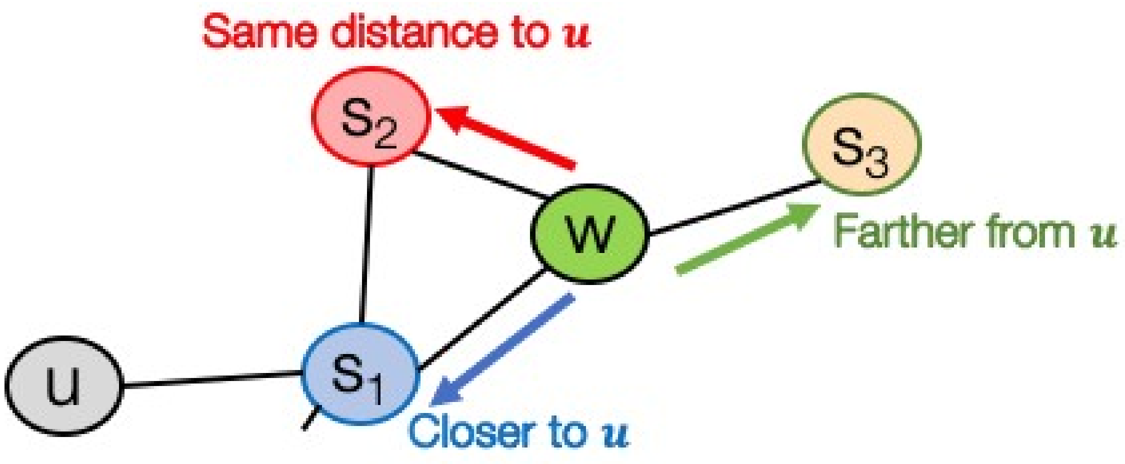
Interpolating BFS and DFS

$$N_{BFS}(u) = \{s_1, s_2, s_3\}$$

$$N_{DFS}(u) = \{s_4, s_5, s_6\}$$



Node2vec: two parameters



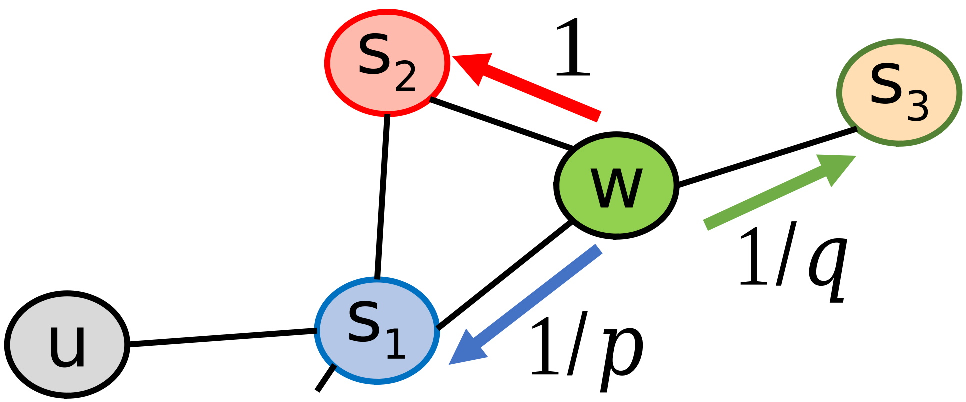
Interpolating BFS and DFS

Biased random walk that given a node generates neighborhood

- Two parameters:
  - Return parameter :
    - Return back to the previous node
  - In-out parameter :
    - Moving outwards (DFS) vs. inwards (BFS)

Node2vec: two parameters

- Walker is at . Where to go next?

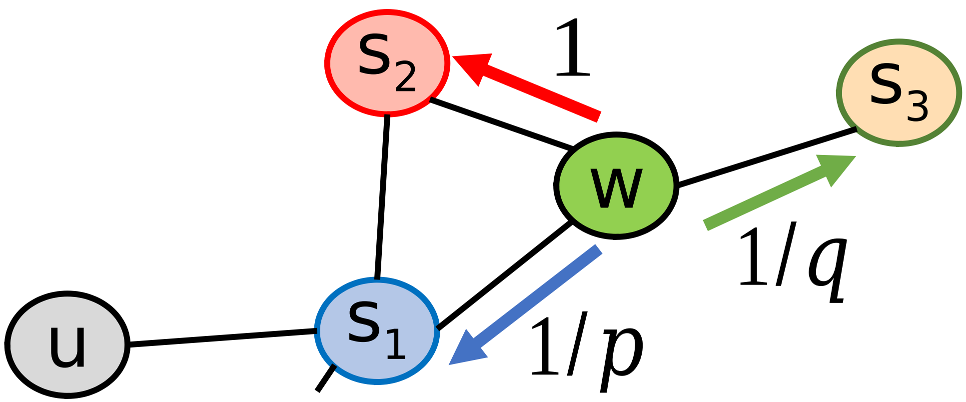


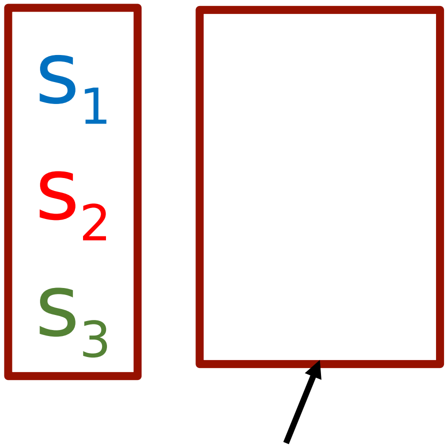
are unnormalized probabilities

- model transition probabilities
  - ... return parameter
  - ... ”walk away” parameter

Node2vec: two parameters

- Walker is at . Where to go next?

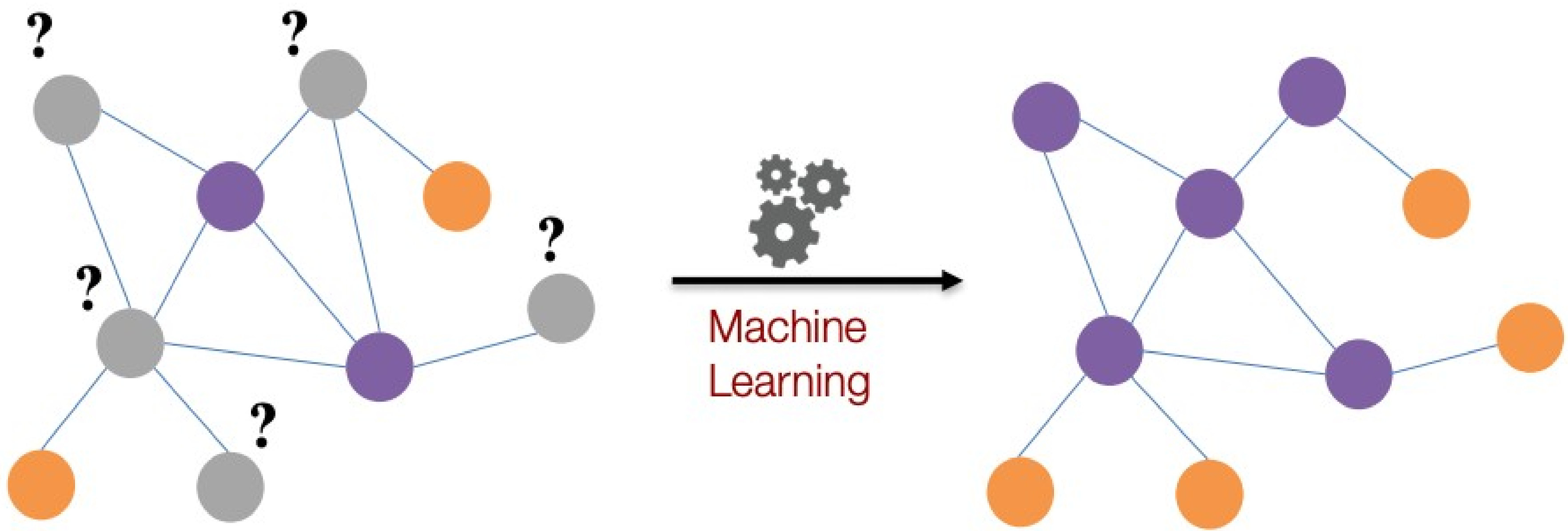




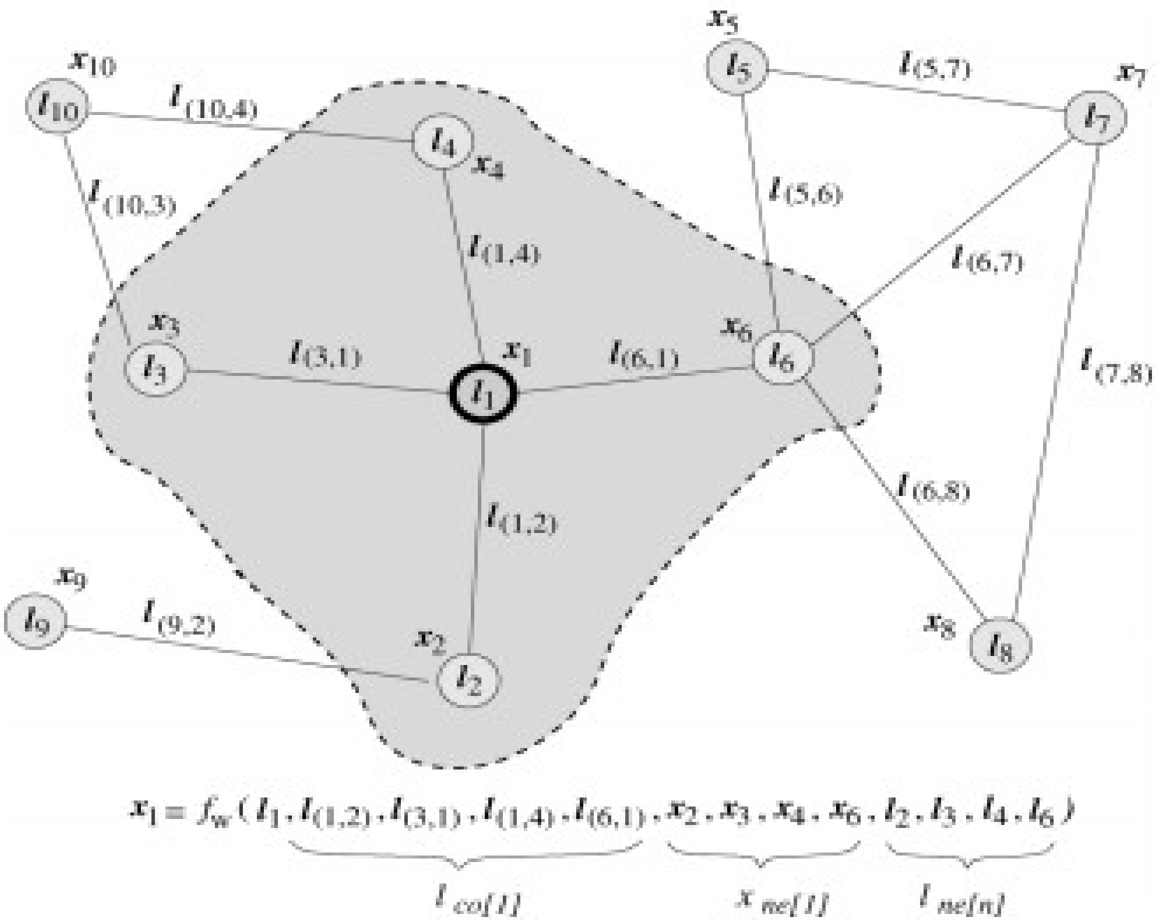
- **BFS-like** walk: Low value of
- **DFS-like** walk: Low value of

are the nodes visited by the walker

Supervised Approach



Graph Neural Network



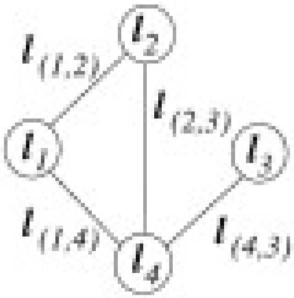
$$\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\text{co}[n]}, \boldsymbol{x}_{\text{ne}[n]}, \boldsymbol{l}_{\text{ne}[n]})$$
$$\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$$

$$\boldsymbol{x} = F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$$
$$\boldsymbol{o} = G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}})$$

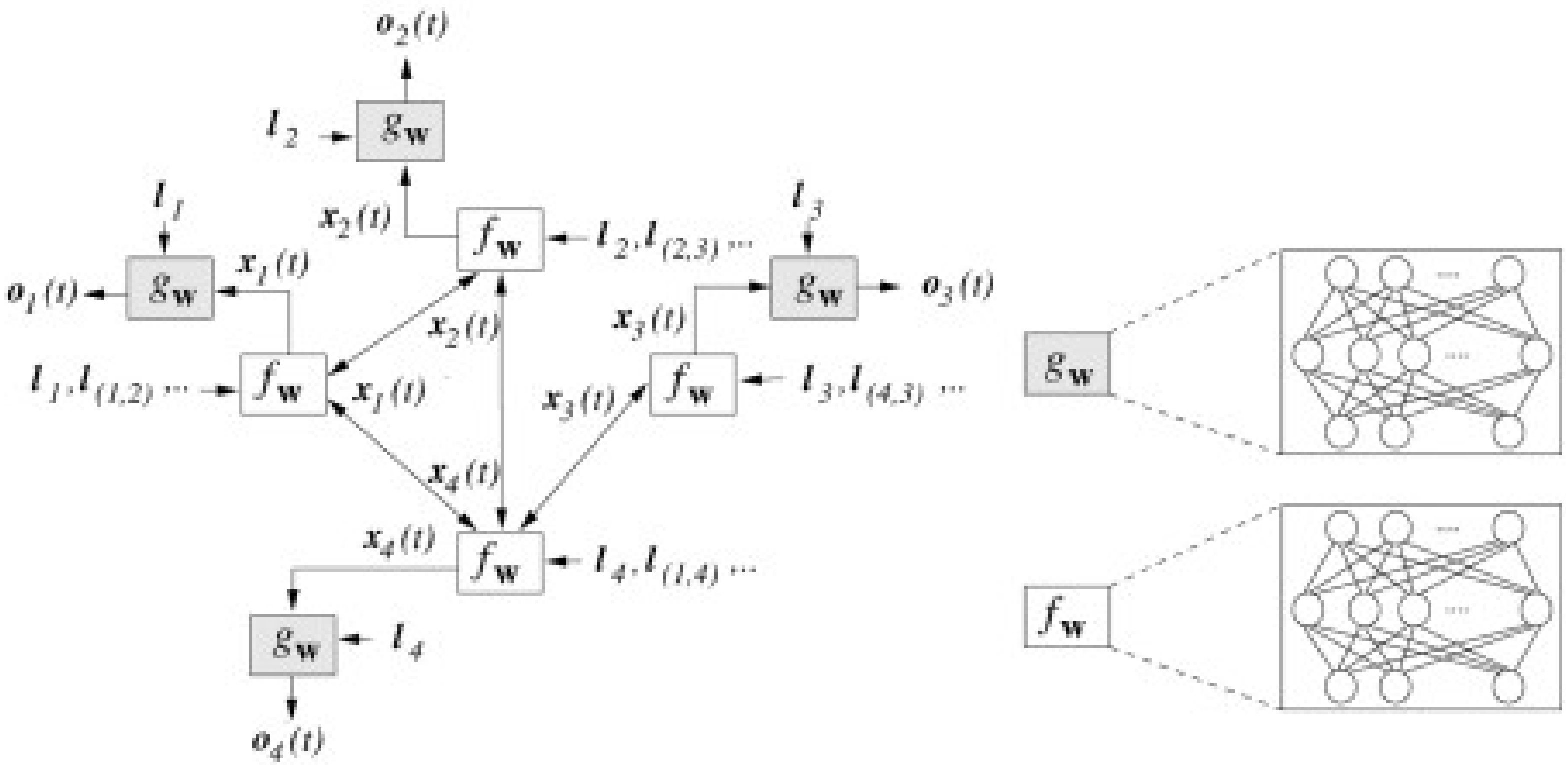
$$loss = \sum_{i=1}^{p} (\mathbf{t}_i - \mathbf{o}_i)$$

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” IEEE TNN 2009, vol. 20, no. 1, pp. 61–80, 2009.

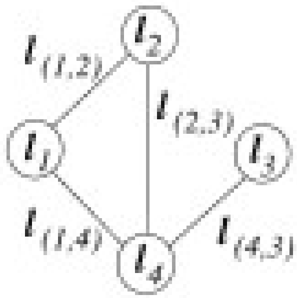
Graph Neural Network

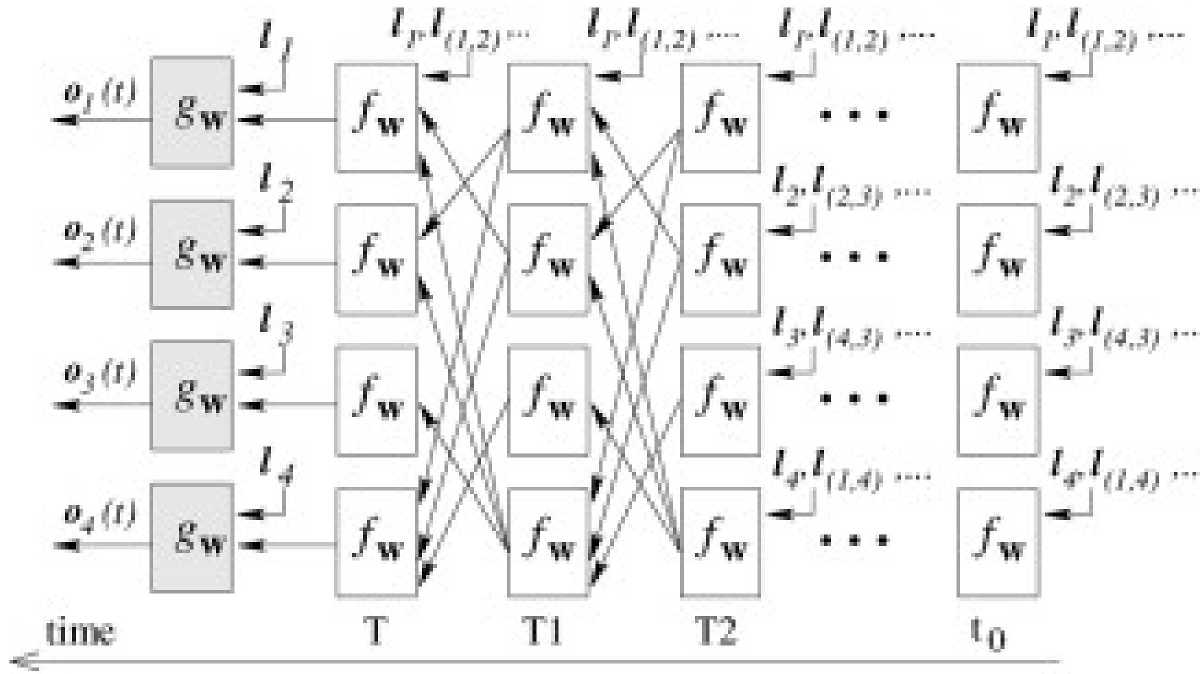


$$\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})$$
$$\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$$



Graph Neural Network

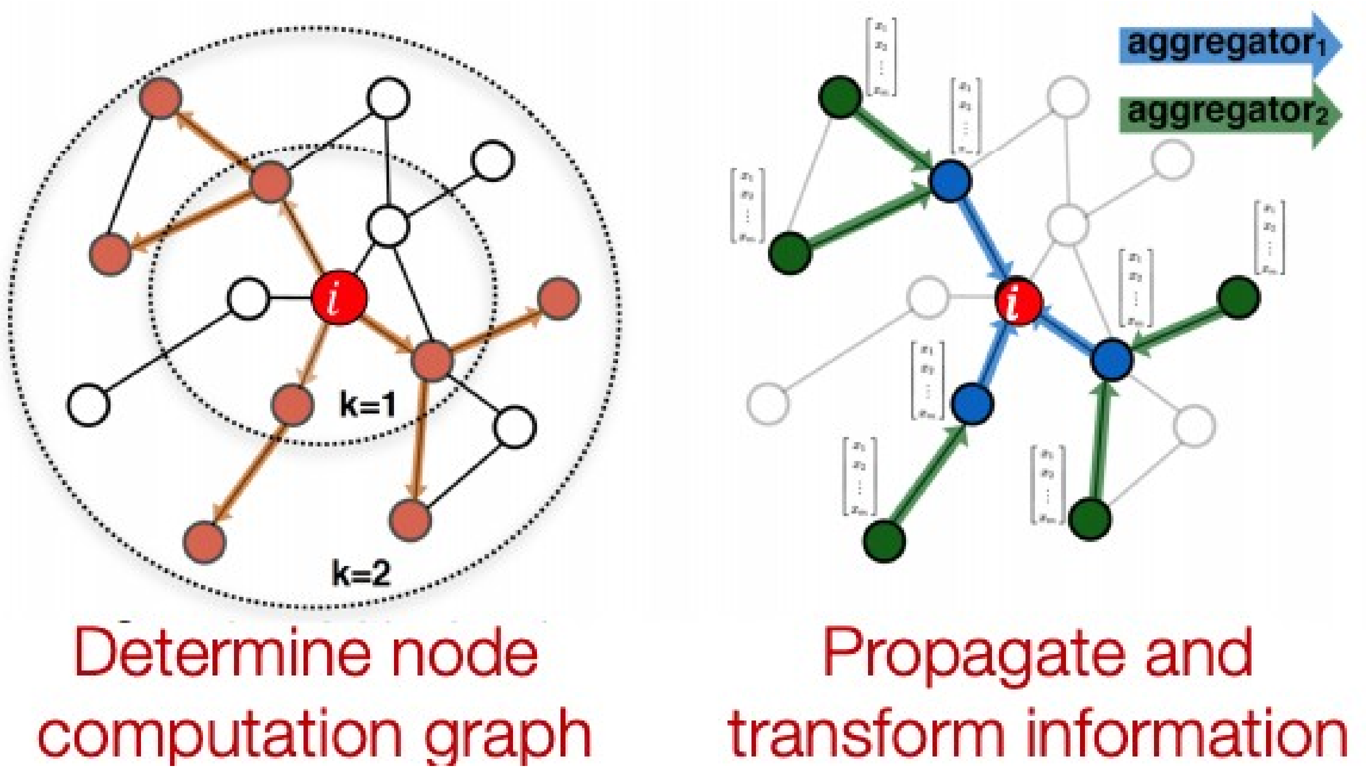




$$loss = \sum_{i=1}^{p} (\mathbf{t}_i - \mathbf{o}_i)$$

GraphSAGA

Idea: Node's neighborhood defines a computation graph



W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," NIPS 2017, pp. 1024–1034, 2017

GraphSAGA

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 $\mathcal{N} : v \to 2^{\mathcal{V}}$

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

```
h_v^0 ← x_v, ∀v ∈ V ;
for k = 1...K do
    for v ∈ V do
        h_N(v)^k ← AGGREGATE_k({h_u^(k-1), ∀u ∈ N(v)});
        h_v^k ← σ(W^k · CONCAT(h_v^(k-1), h_N(v)^k))
    end
    h_v^k ← h_v^k / ||h_v^k||_2, ∀v ∈ V
end
z_v ← h_v^K, ∀v ∈ V
```

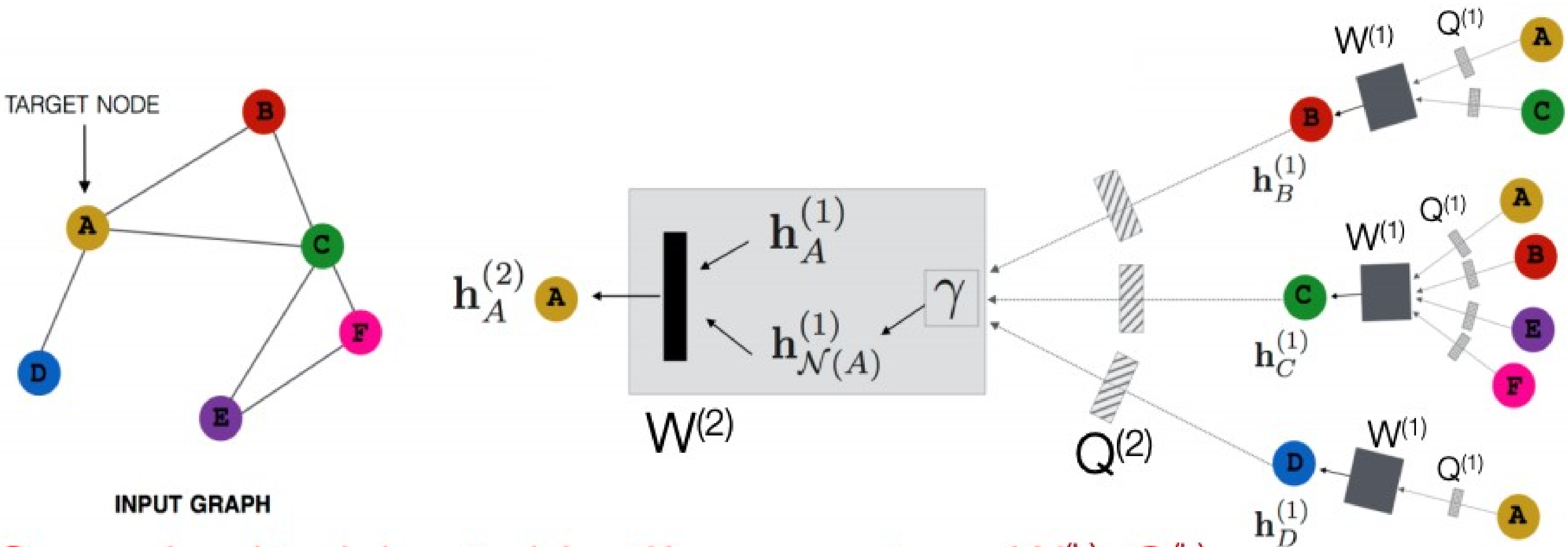
$$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$$
$$\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$$
$$\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$$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

Update for node *i*:

$$h_i^{(k+1)} = ReLU\left(W^{(k)} h_i^{(k)}, \sum_{n \in \mathcal{N}(i)} \left(ReLU(Q^{(k)} h_n^{(k)})\right)\right)$$

- $h_i^{(0)} = X_i$ (directly leverage node attributes)
- $\Sigma(\cdot)$: Aggregator function (avg., LSTM, max-pooling)

GraphSAGA



Supervised training to identify parameters: $W^{(k)}$, $Q^{(k)}$