Node Embedding and Shallow Models in Graphs

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

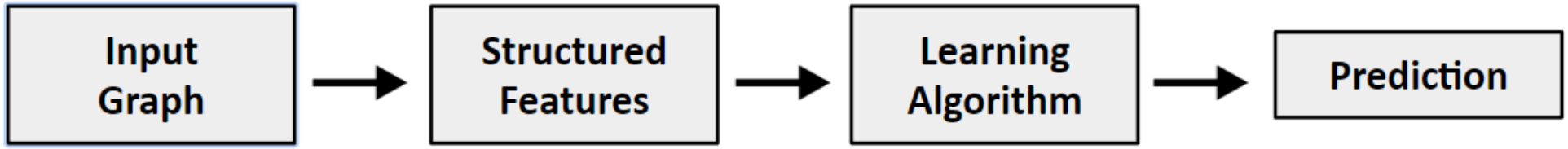
네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA

Contents

- Graph Representation Learning Introduction
  - Traditional Machine Learning for Graphs.
  - Feature Engineering Extraction: Node-level and Graph-level.
  - Graph Representation Learning.
  - Node Embedding and Shallow Encodings.
- Shallow models:
  - Random walk-based: Deep Walk, Node2Vec.
  - Proximity-based: LINE.
  - Role-based: Struc2vec.
  - Homogeneous Graph Embedding: Subgraph2Vec.
  - Heterogeneous graph: Metapath2Vec.

- Given a graph, we can extract features (node-level, graph-level) from the graph, then directly put them to a shallow model to map the features to the ground truth.



Feature Engineering

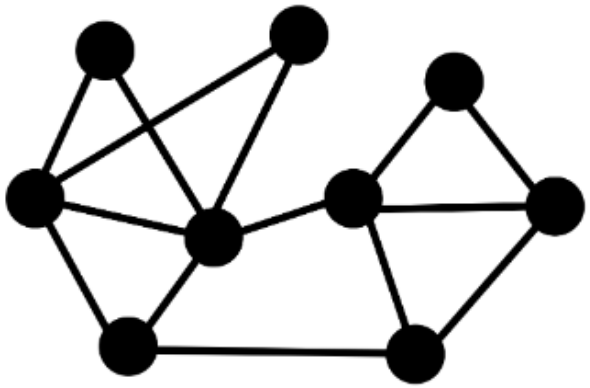
- Node feature
- Edge feature
- Graph feature

- SVM
- Random Forest
- XGBoost
- DNN

- Node-level
- Edge-level
- Graph-level

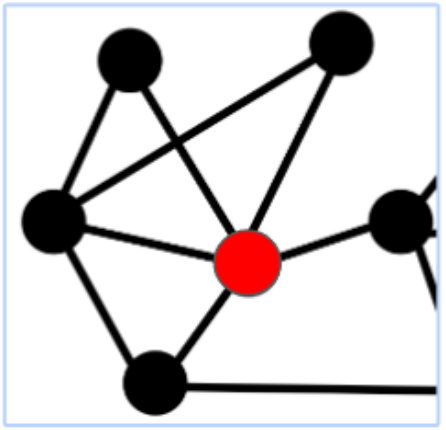
Feature Design

- Using effective features over graphs is the key to achieving good model performance.
- Besides the original node/edge/graph features, can we embed topology structure into node/edge/graph features?



Graph

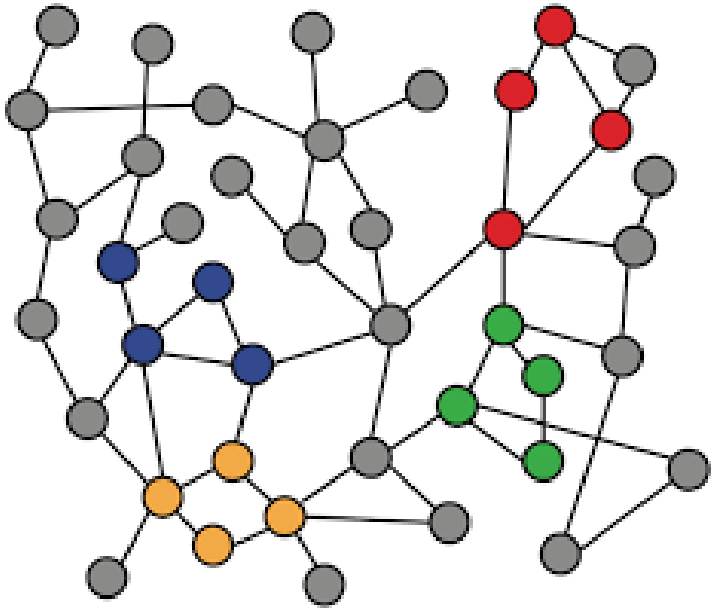
Node prediction



With neighboring topology

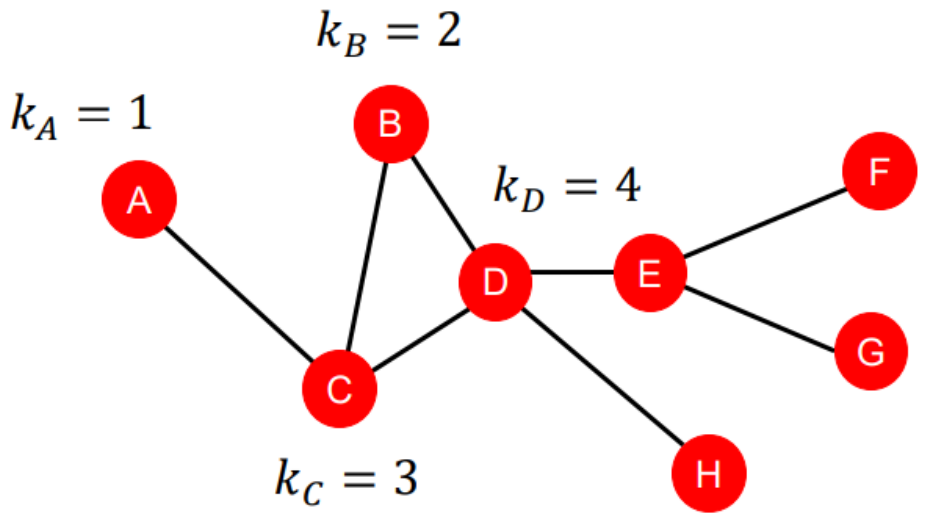
Node-level Features: Overview

- Goal: Characterize the structure and position of a node in the network
  - Importance-based features:
    - Node degree.
    - Node centrality.
  - Structure-based features:
    - Node degree.
    - Clustering coefficient.
    - Graphlets.



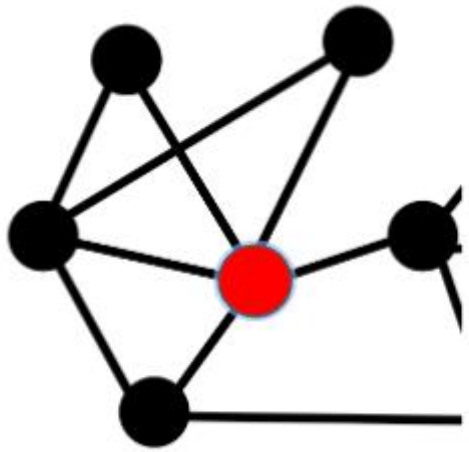
Node Features: Node Degree

- The degree k_v of node v is the number of edges (neighboring nodes) the node has.
  - In-degree: Number of incoming edges to the node.
  - Out-degree: Number of outgoing edges from the node.
  - Total degree: Sum of in-degree and out-degree
- Treats all neighboring nodes equally.



Node Features: Node Centrality

- Node degree counts the neighbouring nodes without capturing their importance.
- Node centrality cv takes the node importance in a graph into account.
- Different ways to evaluate importance:
  - Eigenvector centrality
  - Betweenness centrality
  - Closeness centrality
  - and many others….



➢ Eigenvector Centrality

- ➢ A node v is important if surrounded by important neighboring nodes u ∈ N(v).
- ➢ We model the centrality of node v as the sum of the centrality of neighboring nodes recursively:

$$c_v = \frac{1}{\lambda} \sum_{u \in N(v)} c_u \quad \Longleftrightarrow \quad \lambda \boldsymbol{c} = \boldsymbol{A}\boldsymbol{c}$$

- $\boldsymbol{A}$: Adjacency matrix
  $\boldsymbol{A}_{uv} = 1$ if $u \in N(v)$
- $\boldsymbol{c}$: Centrality vector
- λ: Eigenvalue

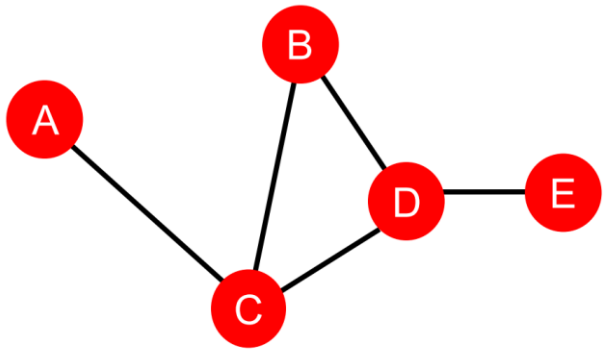
- ➢ We can see that centrality ***c*** is the eigenvector for ***A***

➢ Betweenness Centrality

- ➢ A node is important if it **lies on many shortest paths** between other nodes.

$$c_v = \sum_{s \neq v \neq t} \frac{\#(\text{shortest paths betwen } s \text{ and } t \text{ that contain } v)}{\#(\text{shortest paths between } s \text{ and } t)}$$

- ➢ For example:



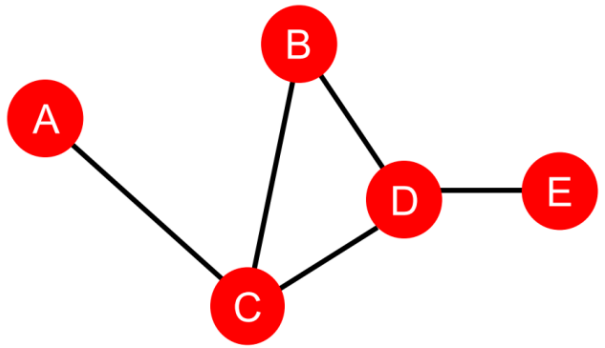
- $C_A = C_B = C_E = 0$
- $C_C = 3$ (A-C-B, A-C-D, A-C-D-E)
- $C_D = 3$ (A-C-D-E, B-D-E, C-D-E)

➢ Closeness Centrality

- ➢ A node is important if it has **small shortest path lengths** to all other nodes.

$$c_v = \frac{1}{\sum_{u \neq v} \text{shortest path length between } u \text{ and } v}$$

- ➢ For example:



- $C_A = 1/(2 + 1 + 2 + 3) = 1/8$

(A-C-B, A-C, A-C-D, A-C-D-E)

- $C_D = 1/(2 + 1 + 1 + 1) = 1/5$

(D-C-A, D-B, D-C, D-E)

Node Features: Node Centrality (4)

- **Katz centrality:**
  - computes the centrality for a node based on the centrality of its neighbours. It is a generalization of the eigenvector centrality.
  - The Katz centrality for node v_i is:

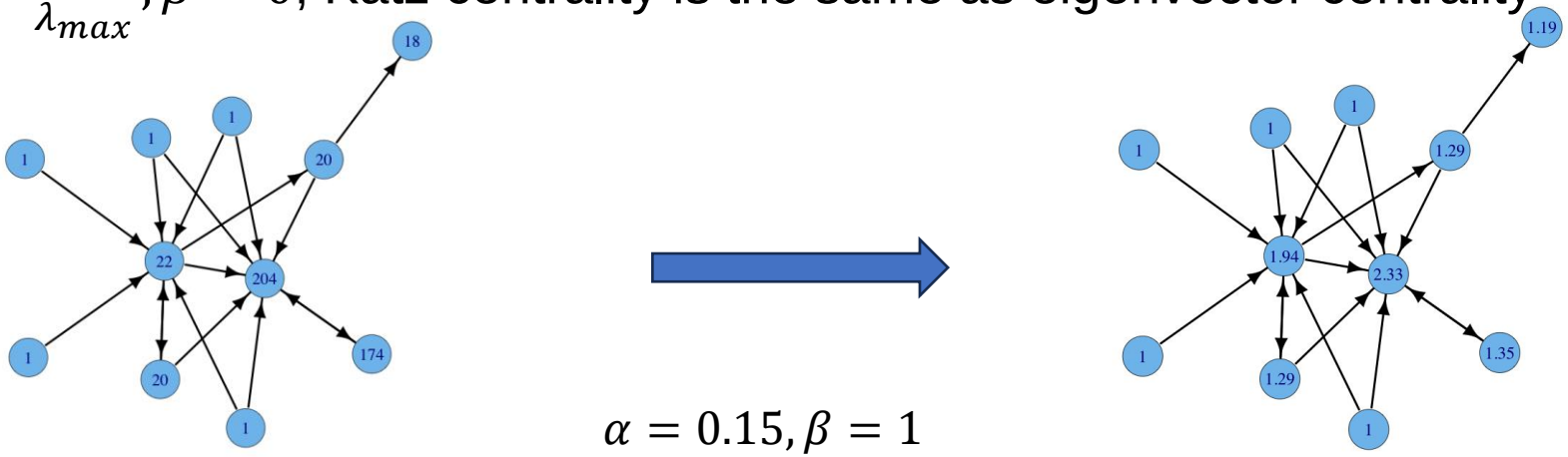
$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

where:

α is a constant called damping factor, and β is a bias constant,

A is the adjacency matrix.

- When $\alpha = \frac{1}{\lambda_{max}}, \beta = 0$, Katz centrality is the same as eigenvector centrality



https://www.sci.unich.it/~francesc/teaching/network/katz.html

Node Features: Clustering Coefficient

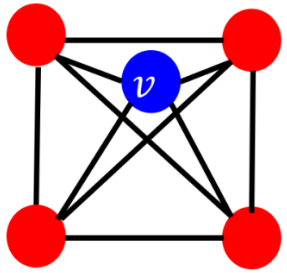
- Measures **how connected v's neighboring nodes** are:

$$e_v = \frac{\#(\text{edges among neighboring nodes})}{\binom{k_v}{2}} \in [0,1]$$

#(node pairs among k_v neighboring nodes)
In our examples below the denominator is 6 (4 choose 2).

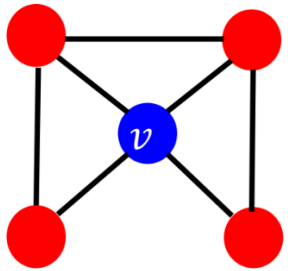
- For example:

6/6

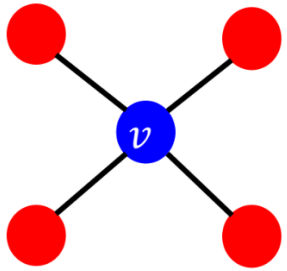


$e_v = 1$

3/6



$e_v = 0.5$

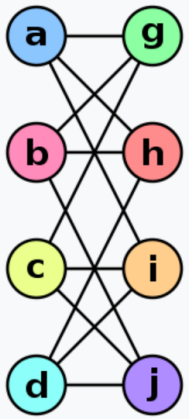
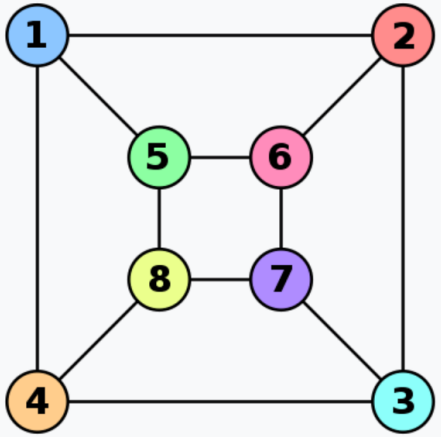


$e_v = 0$

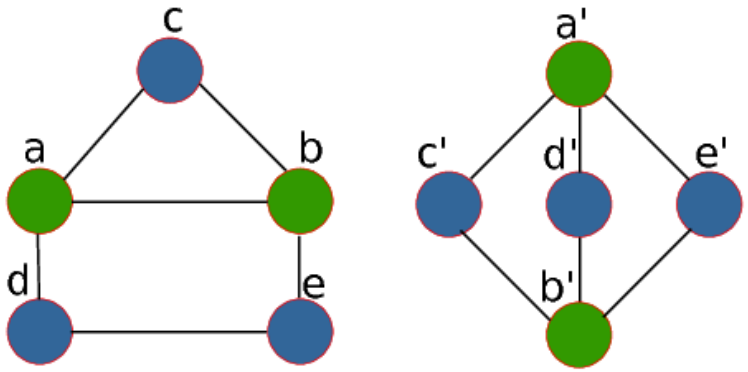
0/6

Graph Isomorphism

- **Isomorphic graphs** having the same number of vertices, edges, and the same edge connectivity.

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

Isomorphic

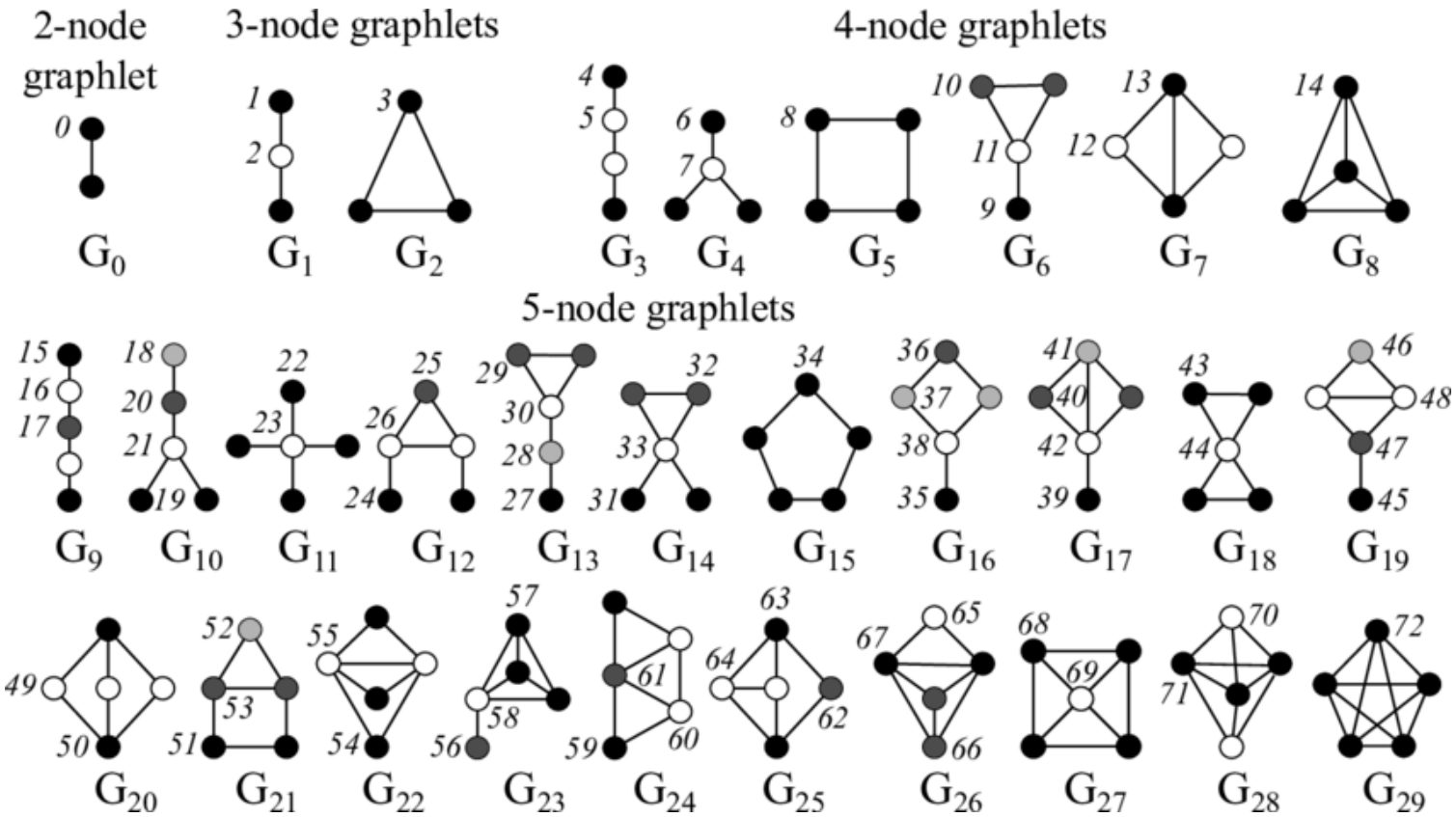


Non-Isomorphic

https://en.wikipedia.org/wiki/Graph_isomorphism

Node Features: Graphlet

- Graphlets are **induced, non-isomorphic** subgraphs that **describe the structure of node u's network neighborhood.**

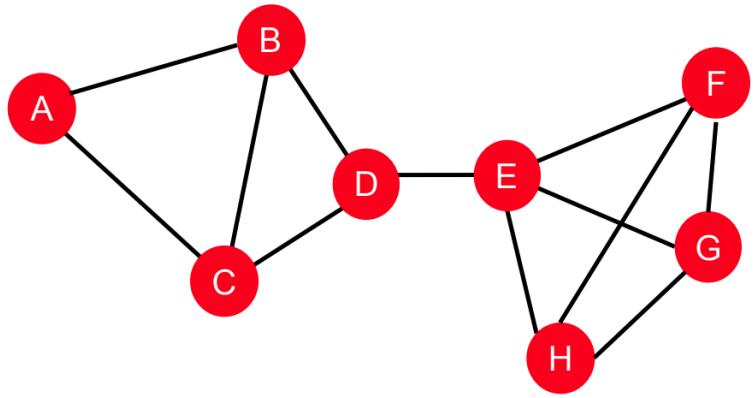


Summary: Ways to Obtain Node Features

- Importance-based features: capture the importance of a node in a graph, useful for predicting influential nodes in a graph
  - Node degree.
  - Node centrality (Eigenvector, Betweenness, Closeness Centrality).

- Structure-based features: capture topological properties of local neighborhood around a node, useful for predicting a particular role a node plays in a graph
  - Node degree.
  - Clustering coefficient.
  - Graphlets.

- Goal: We want features that characterize the structure of an entire graph.
  - Similar graphs has similar features
  - Graph Kernel Methods (kernel methods are widely-used in traditional ML for graph-level prediction.):
  - Graphlet Kernel
  - Weisfeiler-Lehman Kernel



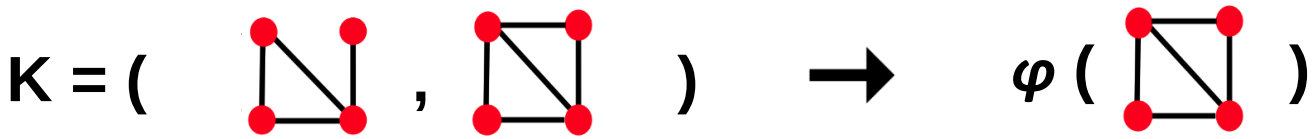
f(graph) = ?

Background: Kernel Methods

- A quick introduction to Kernels:
  - Kernel K(G,G′) ∈ R measures similarity between two graphs (data).
  - There exists a feature representation $\phi(\cdot)$ such that:

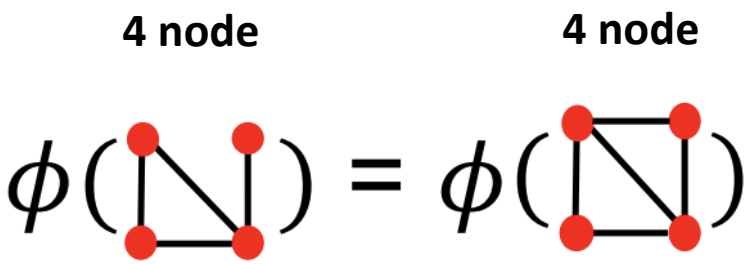
$$K(G, G') = \phi(\mathrm{G})^{\mathrm{T}} \phi(G')$$

  - Kernel enables vectors to be operated in higher dimension
  - Once the kernel is defined, off-the-shelf ML model, such as kernel SVM, can be used to make predictions.



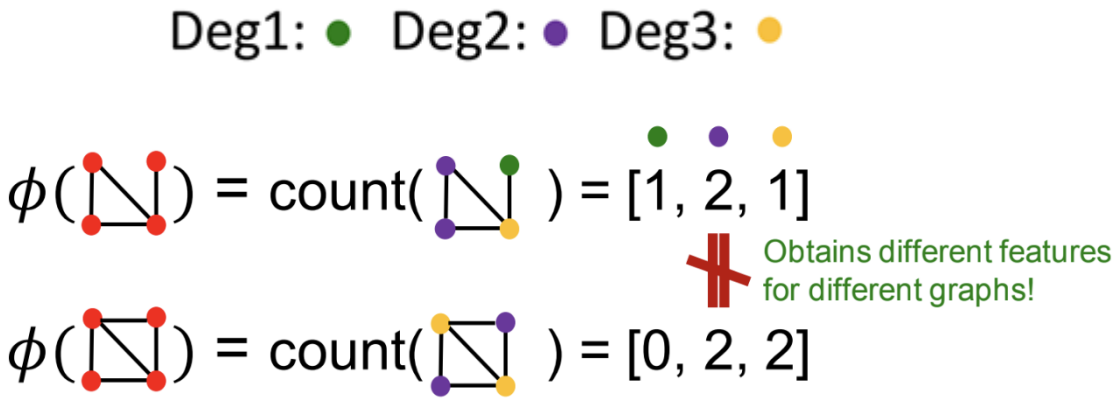
Graph Kernel: Key Idea

- Goal: Design graph kernel $\phi(G)$
- Key Idea: Bag-of-Words (BoW) for a graph
  - In NLP, BoW counts the word's frequency in a document as feature
  - Simplest way on graph: **Regard nodes as words.**
  - For example, we found the features of 2 graphs are same.



Graph Kernel: Key Idea

- Problem: Bag of node counts doesn't work well…
- What if we use Bag of Node Degrees?

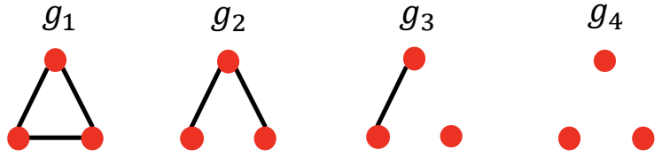


- Both Graph kernel and Weisfeiler-Lehman kernel use Bag-of-* representation of graph, but * is more sophisticated than node degrees!

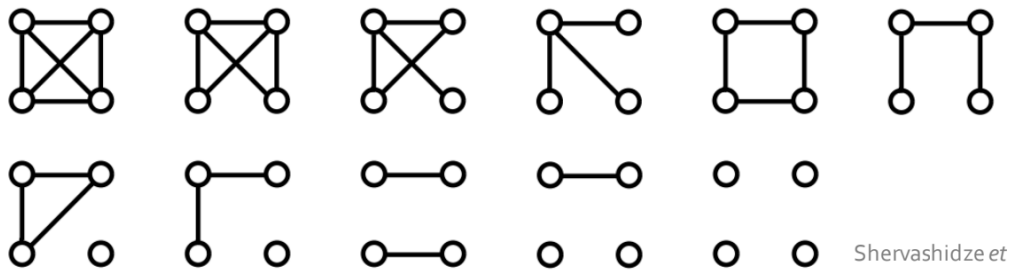
Graphlet Features

- Key Idea: Count the number of different graphlets in a graph
- Graphlet Differences:
  - Nodes do not need to be connected.
  - Graphlets are not rooted.

- For $k = 3$, there are 4 graphlets.



- For $k = 4$, there are 11 graphlets.

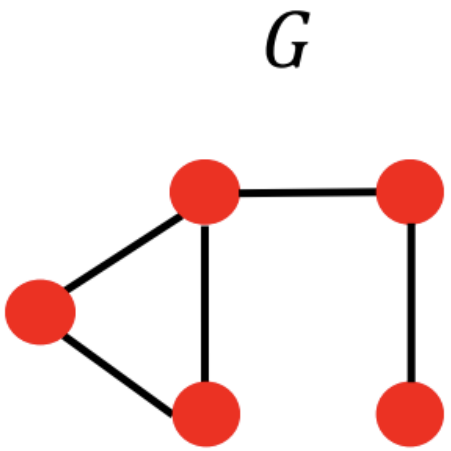


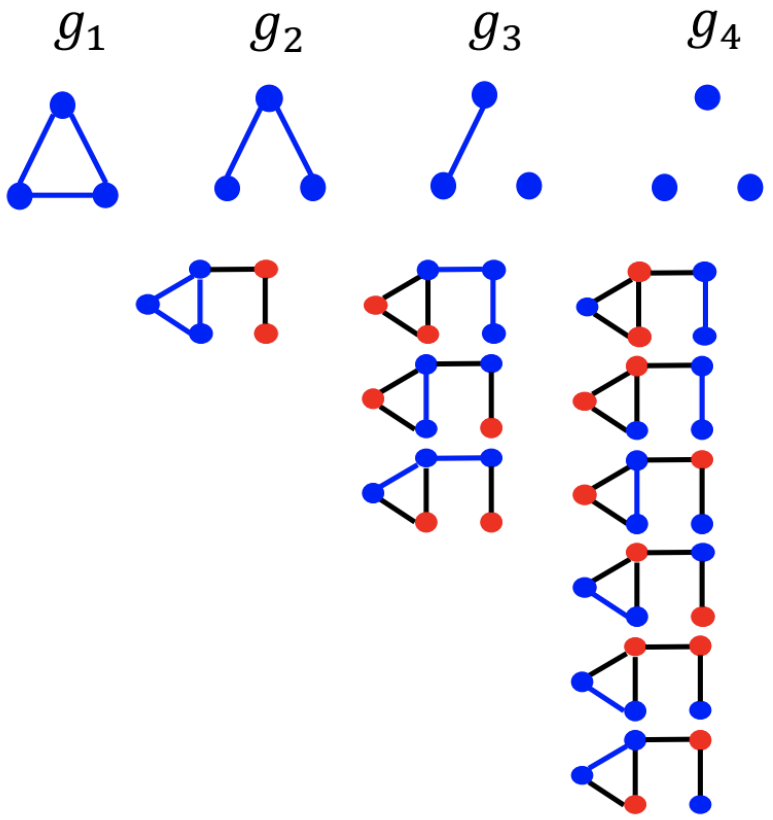
Shervashidze *et al.*, AISTATS 2011

Graphlet Features

- For example:

- For k = 3:

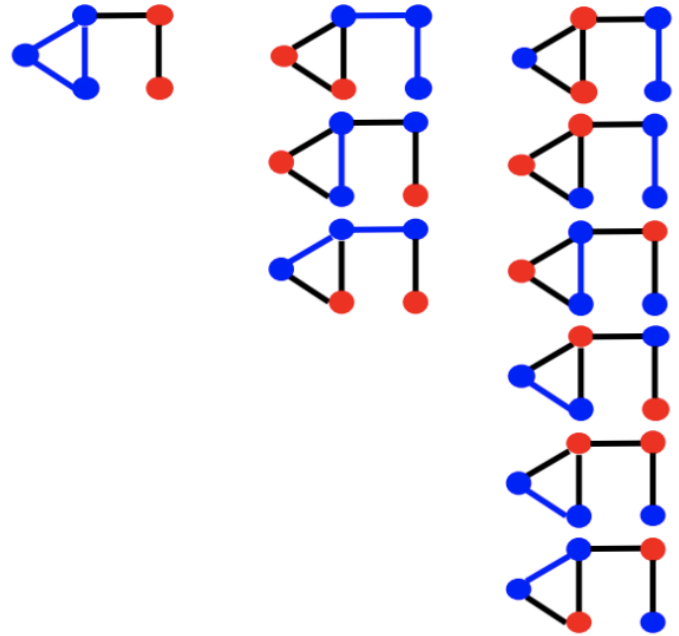




$$f_G = (1, \quad 3, \quad 6, \quad 0)^{\mathrm{T}}$$

Limitations of Graphlet Kernel

- Counting graphlets is expensive!
- Counting size k graphlets for graph with n nodes by enumeration takes n^k time.
- If our graph changes with time, we have to recalculate the features again.



→ **Can we design a more efficient graph kernel?**

Weisfeiler-Lehman Kernel

- Goal: Design an efficient graph feature descriptor $\phi(G)$
- Key Idea:
  - Iteratively use neighborhood structure to describe node's neighboring topology.
  - Generalized version of Bag of node degrees (node degree only contains one-hop neighborhood information).

→ **Color Refinement Algorithm**

- For a graph G with nodes V:
- Assign initial color $c^{(0)}(v)$ to each node v
- Iteratively refine node colors by:

$$c^{(k+1)}(v) = \text{HASH}\left(\left\{c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\right\}\right)$$

where HASH maps different inputs to different colors.

- After K steps of iteration, c(K)(v) represents the structure of the K-hop neighborhood.

Weisfeiler-Lehman Kernel

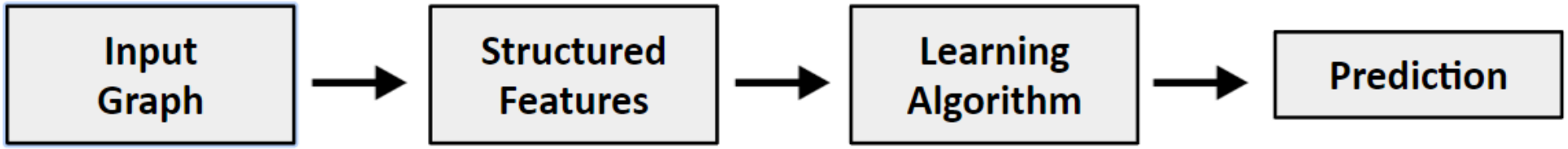
- WL kernel benefits:
  - Computationally efficient (color refinement need #(edges) steps)
  - #(colors) depends on total number of nodes.
  - Can be used to check graph isomorphism

Summary: Graph-level Features

- Graphlet Kernel:
  - Bag of graphlets.
  - Computationally expensive.
- Weisfeiler-Lehman Kernel:
  - Bag of colors.
  - Capture graph structure within K-hop.
  - Computationally efficient.
  - Closely related to graph neural networks!

Graph Representation Learning

- Graph Representation Learning aims to generate graph representation vectors that describe graph structures.
- We don't need to do feature engineering every single time.



Feature Engineering

Representation Learning

learn the features by itself

- SVM
- Random Forest
- XGBoost
- DNN

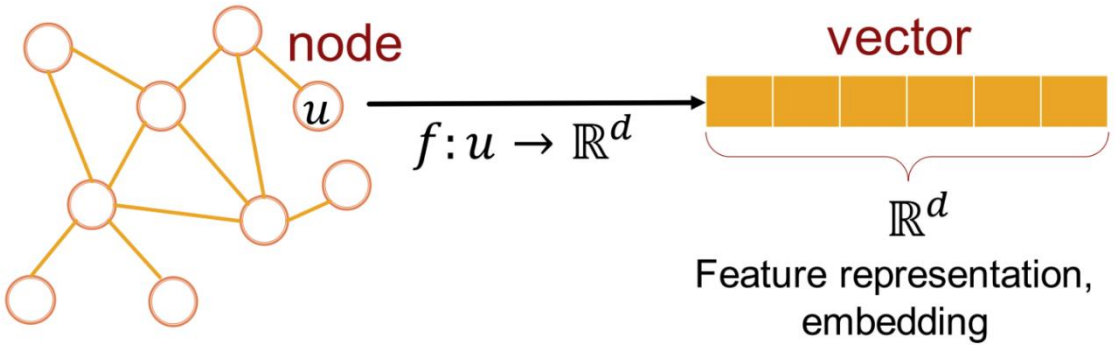
- Node-level
- Edge-level
- Graph-level

Why Embeddings?

- Graph Representation Learning's **goal**: Learn efficient task-independent feature for machine learning with graphs

  → Map nodes into an embedding space.

- Requirements: Similarity of embeddings between nodes indicates their similarity in the network.
- For simplicity, no node features or extra information is used.



Tasks

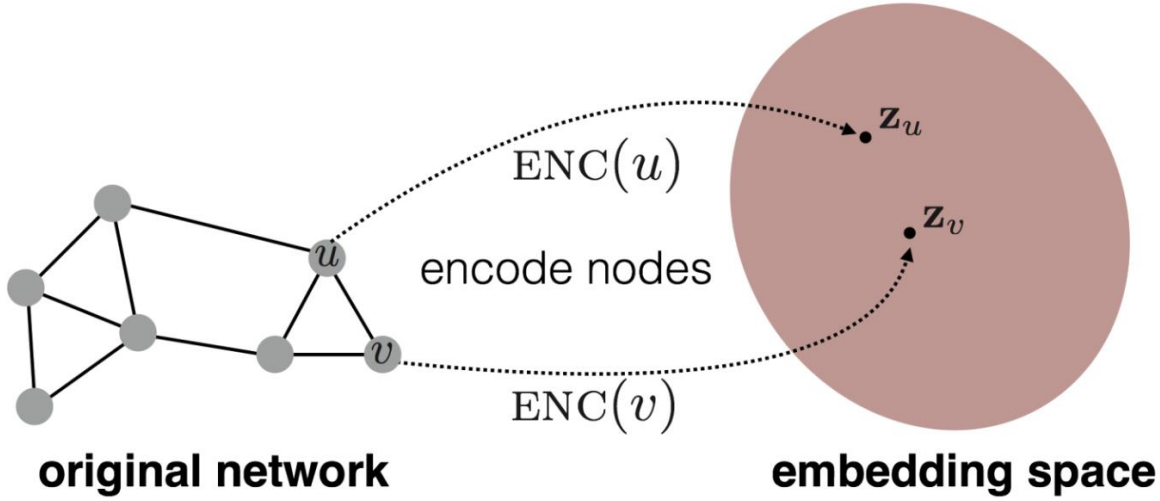
- Node classification/regression
- Edge prediction
- Graph classification/regression.
- Graph clustering
- …

Node Embeddings

- Goal: Mapping nodes from the graph to the embedding space *w.r.t* the similarity between two nodes in the embedding space (via dot product) ≈ similarity between them in the original graph

$$\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u}_{\text{Similarity of the embedding}}$$

We need to define:
ENC(u)
Similarity(u, v)



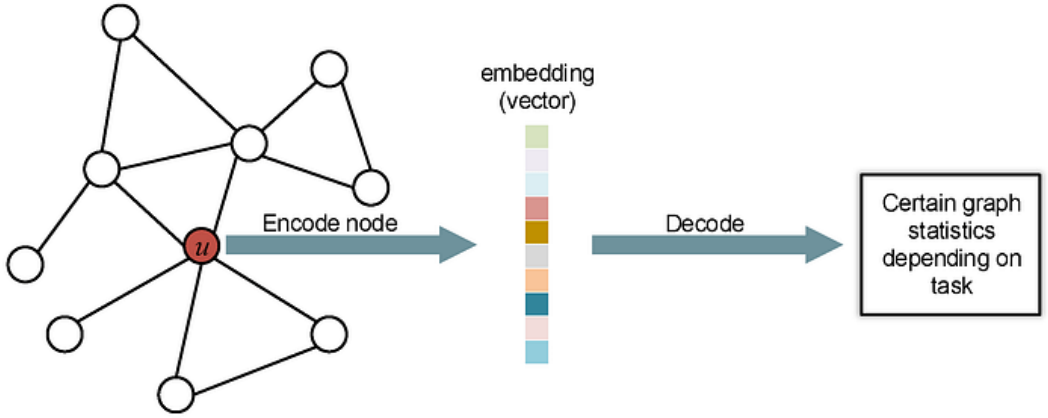
https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/node-representation-learning

Learning Node Embeddings

- Encoder: Mapping nodes → embeddings (d-dimensional vector, ENC(v) = Z_v)
- Decoder: Mapping embeddings → similarity score (dot product)
- Define a node similarity function:
  - Whether 2 nodes are close in the graph, and how close are they?

Dot product between node embedding

- Optimize → $\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u}_{\text{Similarity of the embedding}}$
  - Maximize $\boldsymbol{z}_v^T \boldsymbol{z}_u$ for node pairs (u, v) that are similar.

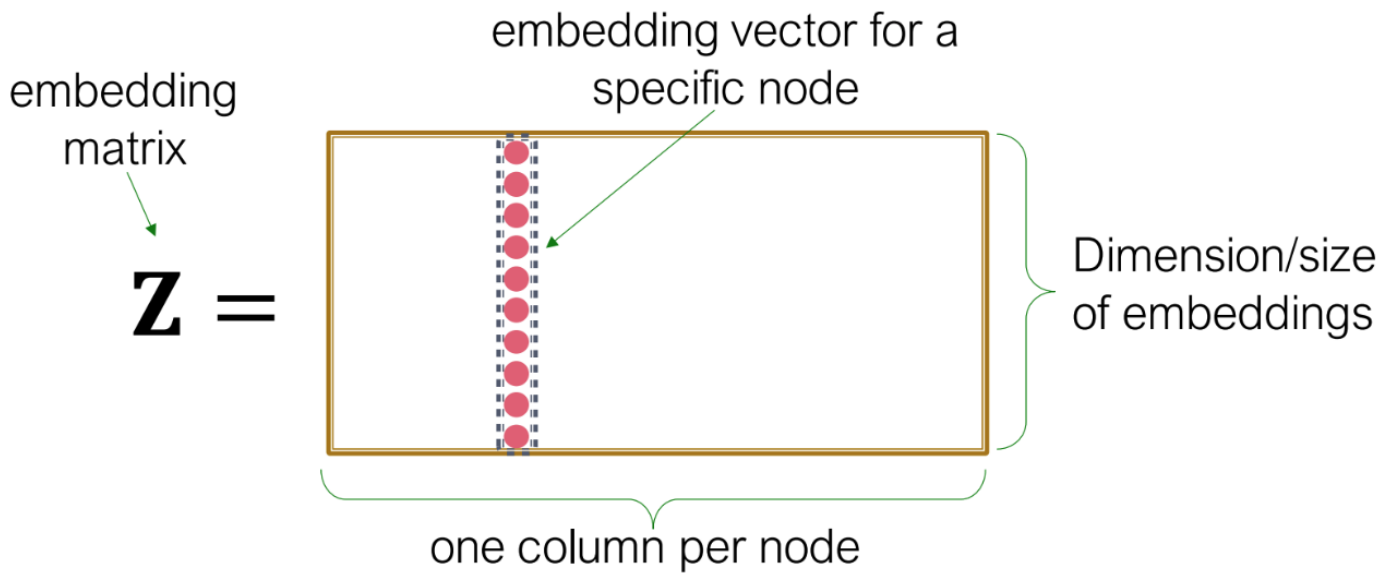


https://medium.com/the-modern-scientist/graph-neural-networks-series-part-3-node-embedding-36613cc967d5

Shallow Encoding Strategies

- How to learn node embeddings?
- Simplest encoding approach: Build an embedding lookup table

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



$$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$$

matrix, each column is a node embedding (this is what we want to learn)

$$v \in \mathbb{I}^{|\mathcal{V}|}$$

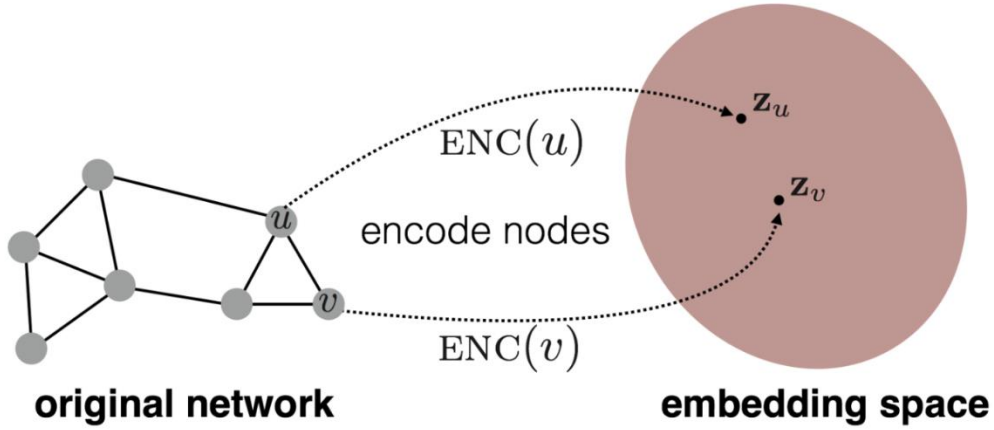
indicator vector, all zeros except a one in column for node v

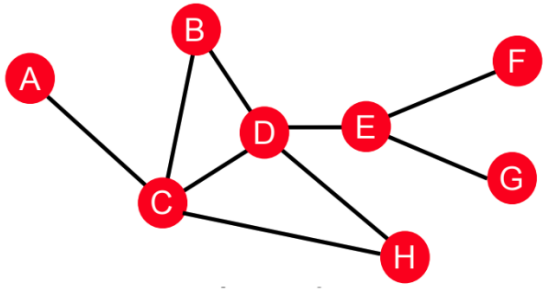
Shallow Encoding Strategies

- Each node is assigned a unique embedding vector.

→ We **directly optimize the embedding of each node.**

- Embedding is optimized to maximize $z_v^T z_u$ for each similar node pairs (u, v).
- Key choice: How to define node similarity? Should two nodes have similar embedding if:
  - They are linked?
  - They share neighbors?
  - They have similar structure roles?





Note on Node Embeddings

- This is unsupervised/self-supervised way of learning node embeddings.
  - Not using node labels
  - Not using node features
  - Directly learn a set of coordinates (the embedding) of a node, so that some feature of the network is preserved.
- These embeddings are task independent
  - They can be used in different downstream predictions.

Adjacency-based Approaches

- One of the simplest and most intuitive approaches to define similarity:
  - adjacency between two nodes v and u.
- Similarity: Two nodes are adjacent to one another within the structure of the graph.
- Encoding: Find the embedding matrix **Z** that minimizes the loss function **L**

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

loss (what we want to minimize)

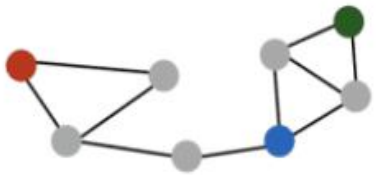
sum over all node pairs

embedding similarity

(weighted) adjacency matrix for the graph

- Limitation: fails to capture the similarity between distant nodes.

How to capture the global graph structures?

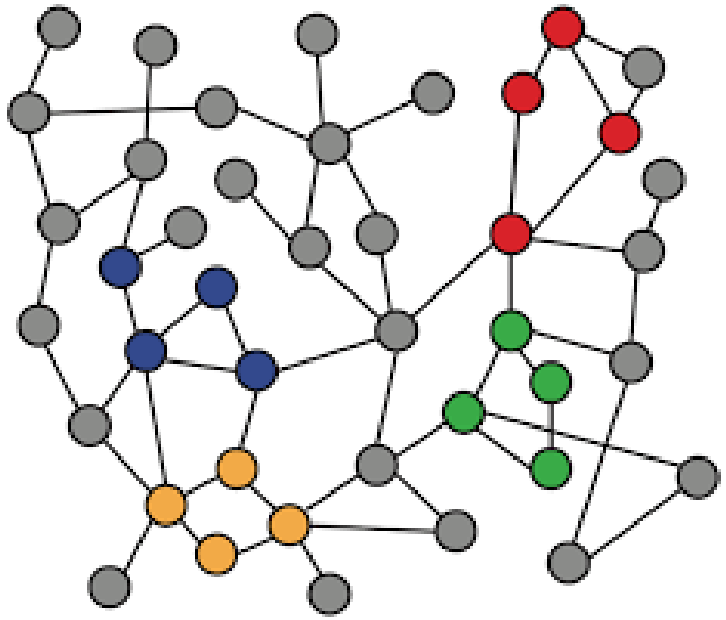


e.g., the blue node is obviously more similar to green compared to red node, despite none having direct connections

https://www.graphable.ai/blog/knowledge-graph-embeddings/

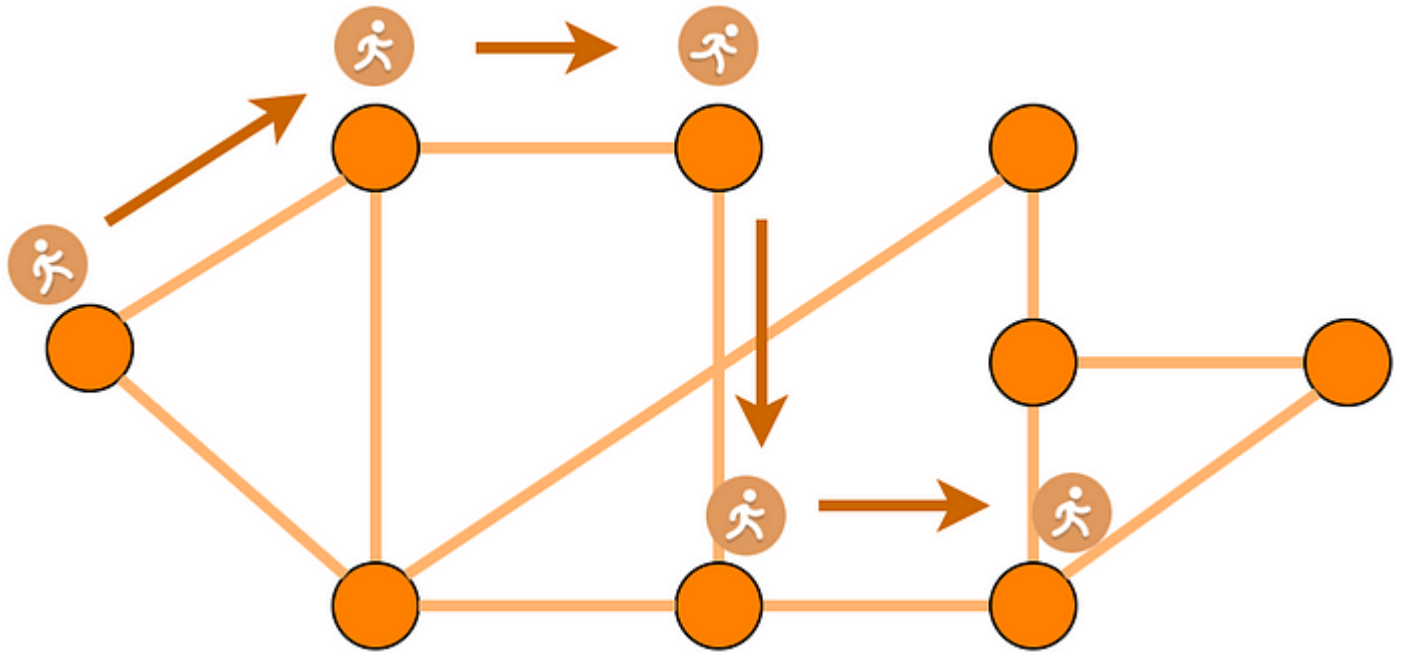
Random Walk-based Methods for Node Embeddings

- Random Walk.
- Learning Objective & Method
- Representatives: DeepWalk, Node2vec, Div2Vec,Node2vec+, WalkLet.



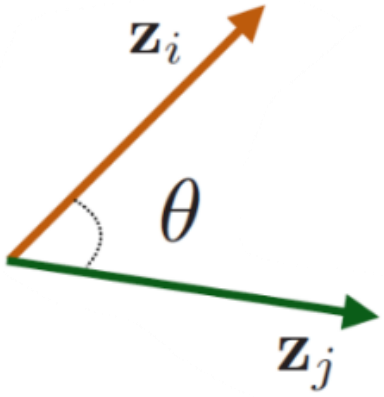
Random Walk

- Given a graph and a starting point, we select a neighbour of it at random, and move to this neighbour.
- Then, we select a neighbor of this point at random, and move to it,…
- The random sequence of nodes visited this way is **a random walk on the graph**

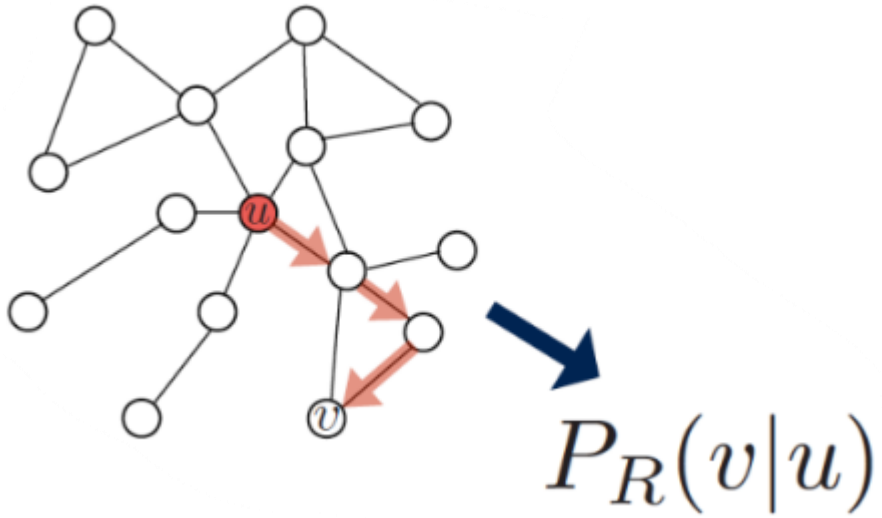


https://medium.com/analytics-vidhya/an-intuitive-explanation-of-deepwalk-84177f7f2b72

- Estimate probability of visiting node v on a random walk starting from node u with strategy R: $P(v|\boldsymbol{z}_u)$.
- Optimize node embeddings to learn the statistics from random walks.



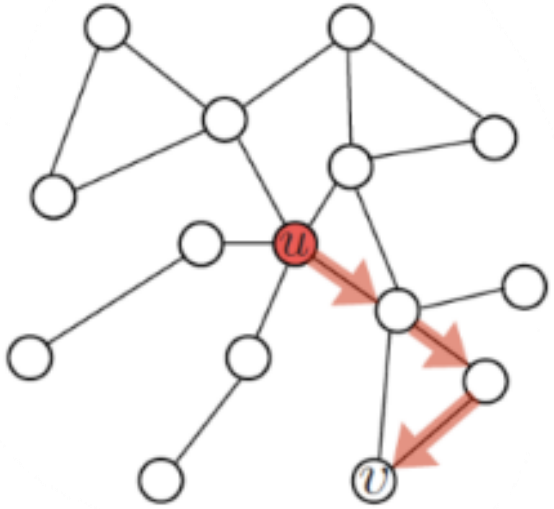
$$\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v \approx$$



Why using Random Walks?

- We can simply define nodes are similar if there are connected, why random walks?

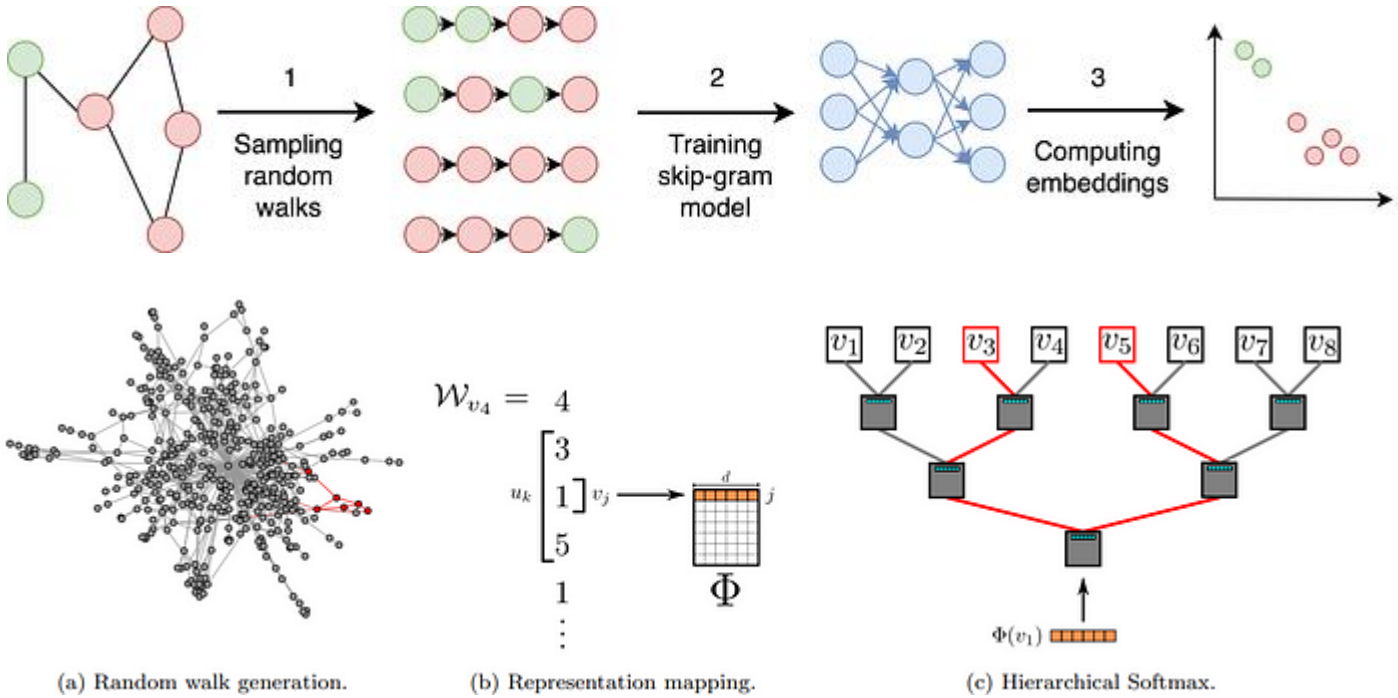
- **Expressivity**:
  - Random walk incorporates both local and higher-order multi-hop neighborhood information (Global structures)



- **Efficiency**:
  - Don’t need to consider all node pairs at training state; only need to consider node pairs on the random walks

DeepWalk: Randomly Walk

- What sampling strategies should we use to explore the graph structure?
- Simplest idea:
  - Run fixed-length, unbiased random walks starting from each node (i.e., DeepWalk from Perozzi et al., 2013).
  - Find embeddings $\boldsymbol{z}_u$ that minimizes $\mathcal{L}$.



(a) Random walk generation. (b) Representation mapping. (c) Hierarchical Softmax.

Bryan Perozzi et al: DeepWalk: Online Learning of Social Representations.

DeepWalk: Feature Learning as Optimization

- Given $G = (V, E)$, goal is to learn mapping $f: u \rightarrow \mathbb{R}^d: f(u) = \boldsymbol{z}_u$.
- Log-likelihood objective:

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) | \mathbf{z}_u)$$

$N_R(u)$ is the neighborhood of node u by random walk R

→ learn feature representations that are predictive of the nodes in its random walk neighborhood $N_R(u)$.

- Equivalently, loss function:

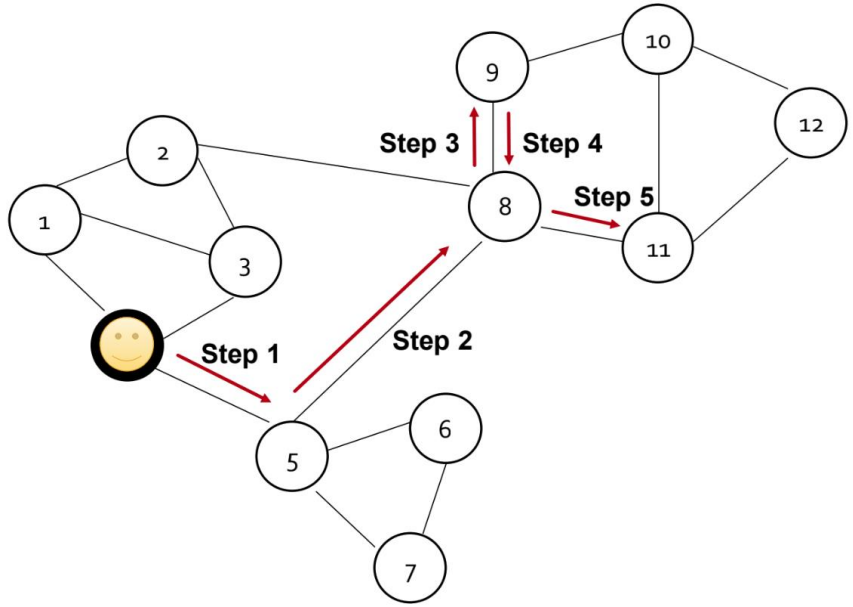
$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

- Parameterize using Softmax:

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

DeepWalk: Randomly Walk

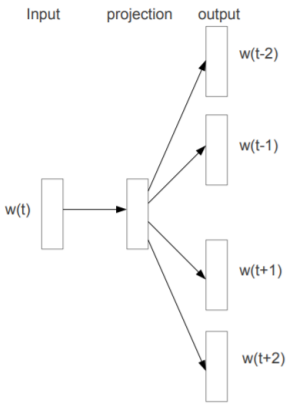
- Stage 1: Local structure discovery
  - Goal: discover neighborhoods in the network, where assumption is that adjacent nodes are similar and should have similar embeddings.
  - Consider a fix-length of each walk l.
  - Generate a fixed number k of random walks starting at each node.
  - When it is finished, we obtain k node sequences of length l or collect the visited node set $N_R(u)$ for each node u.



Bryan Perozzi et al: DeepWalk: Online Learning of Social Representations.

DeepWalk: Skip-Gram Procedure

- Stage 2: Skip Gram
  - Idea: Borrowing idea from NLP
    - Goal: Given a corpus and a window size, SkipGram aims to maximize the similarity of word embeddings of the words that occur in the same window. (window = context in NLP).
    - Assumption: Words that occur in the same context, tend to have close meanings. Therefore, their embeddings should be close to each other as well.
  - In Graph: Each walk produced in the previous step as a context or word window in a text.
    - Equivalent to node sequences in networks correspond to word sequences in text.
  - Algorithm:
    - Generate random vectors of dimension d for each node.
    - Iterate over the set of random walks and update the node embeddings by gradient descent.



Tomas Mikolov et al: Distributed Representations of Words and Phrasesand their Compositionality

DeepWalk: Optimization using Stochastic Gradient Descent

- **Goal**: Optimize $\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$
- (Stochastic) Gradient Descent: a simple way to minimize L
- **SGD Algorithm:** evaluate it for each individual training example
  - Initialize z_u at some randomized value for all nodes u.
  - Iterative until L converges:
    - Sample a node u, for all v calculate the derivative $\frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$.
    - For all v, update:

$$z_v \leftarrow z_v - \eta \frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$$

Learning rate

Limitation of DeepWalk

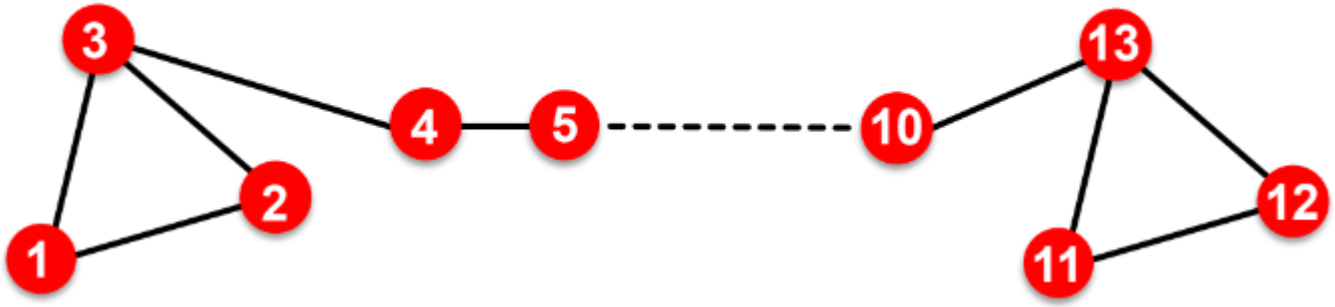
- Note:
  - Frequency of co-occurrence in random walks is an indicator of node similarity.
  - Effect of k (# walks) and l (length) is important:
    - more k is increased, the more the network is explored.
    - l is increased, paths become longer, and more distant nodes are accepted as similar nodes.
- **Issue:** Expensive from nested sum over nodes($O(|V|^2)$)

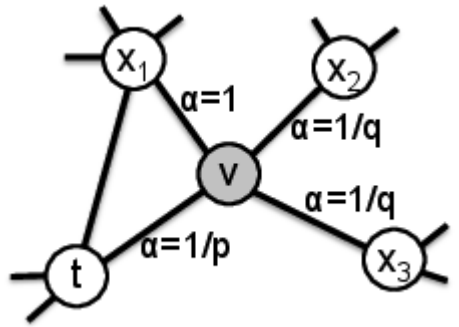
$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

- 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

Node2Vec: Biased Walks

- **Goal**: Embed nodes with similar network neighborhoods close in the embedding space and control the next steps in the random walk
- **Key**: Develop different strategies to capture the local and global graph structures, which depends on the specific graphs.
- **Learning method**: Maximum likelihood optimization problem

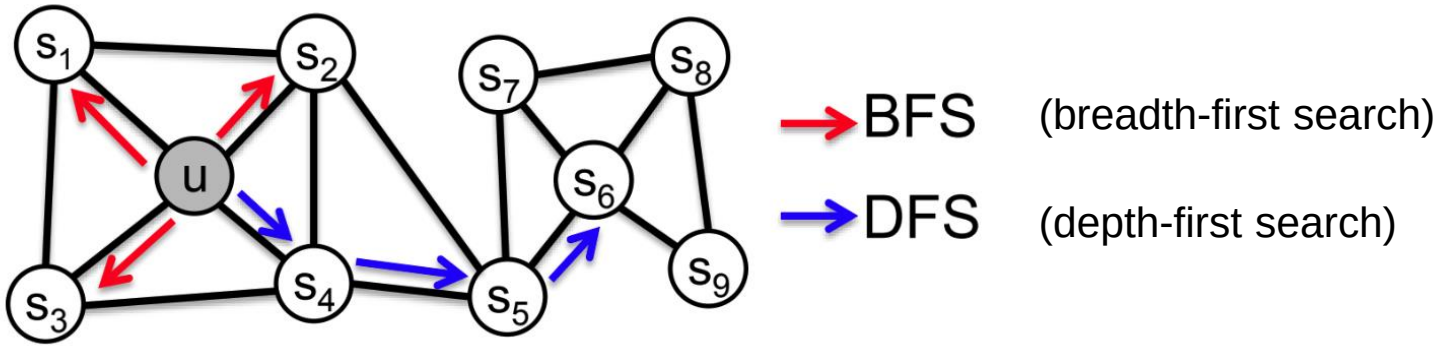




node2vec: Scalable Feature Learning for Networks

Node2Vec: Biased Walks

- **Idea**: Use flexible, biased random walks that can trade-off between local and global views of the network (Grover and Leskovec, 2016).
- Two classic strategies to define a neighborhood $N_R(\boldsymbol{u})$ of a given node u



- Walk of length ($N_R(\boldsymbol{u})$ of size 3):

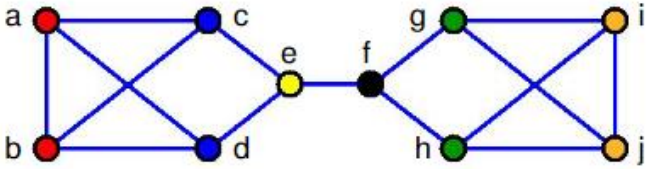
$$N_{BFS}(u) = \{s_1, s_2, s_3\}$$ **Local view**

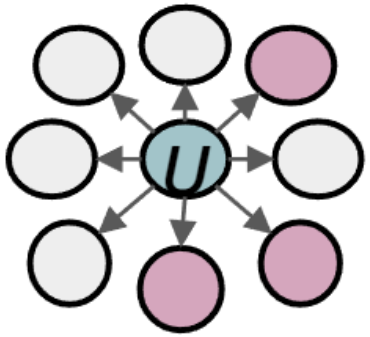
$$N_{DFS}(u) = \{s_4, s_5, s_6\}$$ **Global view**

node2vec: Scalable Feature Learning for Networks

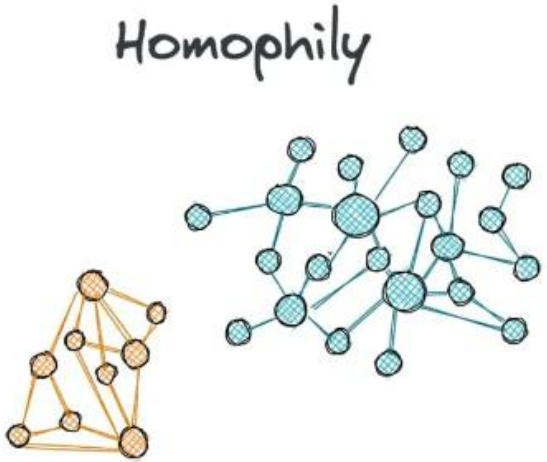
Node2Vec: Equivalence with Interpolating BFS and DFS

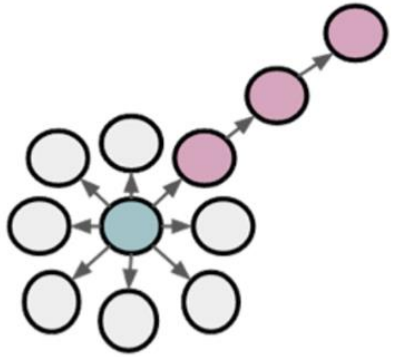
- **Structural Equivalence:** if and only if they have the same structure.
- **Homophily:** tendency of individuals to associate and bond with similar others.
  - Neighborhoods sampled by **BFS** corresponding to **structural** equivalences.
  - **DFS** explore macro-view of the network, which infers communities: **homophily** equivalence.





BFS





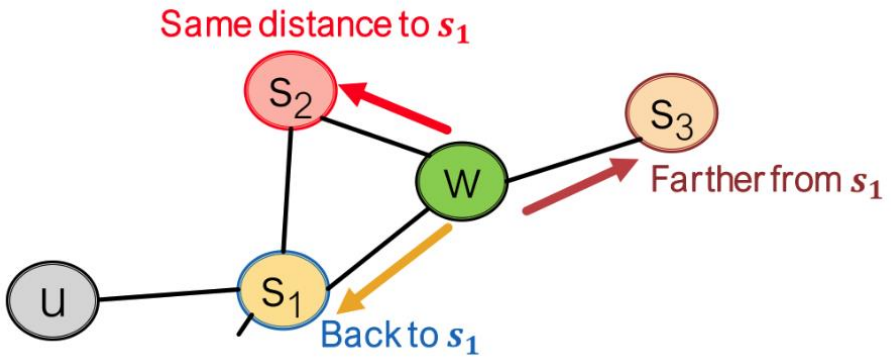
DFS

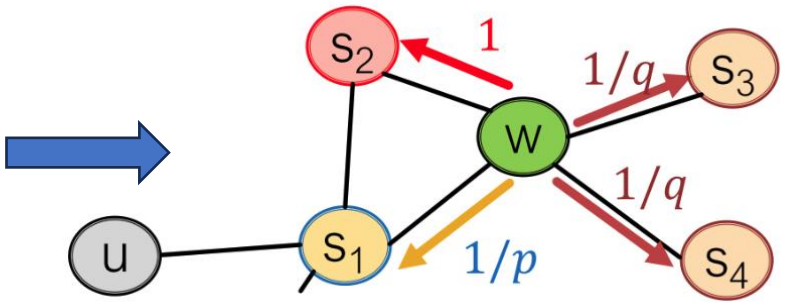
- Biased fixed-length random walk R that given a node u generates neighborhood $N_R(u)$.
  - Two parameters of random walks:
    - Return parameter p: Return to the previous node.
    - In-out parameter q: Moving outwards (DFS) vs. inwards (BFS) from the previous node.

SERGEY IVANOV: GML Newsletter: Homophily, Heterophily, and Oversmoothing for GNNs

Node2Vec: Biased Random Walk

- Two strategies to explore network neighborhood: BFS and DFS
- Walker just traversed edge (s_1, w) and is at w, now he/she can go using a bias probability:





1/p, 1, 1/q are unnormalized probs.
p: return parameter.
q: "walk away" parameter.

- BFS-like walk: Low value of p
- DFS-like walk: Low value of q

	Target t	Prob.	Dist. (s_1, t)
w →	s_1	$1/p$	0
	s_2	1	1
	s_3	$1/q$	2
	s_4	$1/q$	2

Node2Vec: Algorithm

- Compute random walk probabilities:
  - For each edge (s_1, w) we compute walk probabilities (based on p, q) of $(w, .)$
- Simulate r random walks of length l starting from each node u.
- Optimize the node2vec objective using stochastic gradient descent with negative sampling.

→ The training process is same as DeepWalk, except the walking strategy.

Node2Vec: Reminding of DeepWalk Optimization

- Problem: Expensive in summing over nodes ($O(|V|^2)$)

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

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

- The normalization term from the softmax is the culprit
- Solution: **Negative Sampling**:
  - normalize against k random "negative samples" n_i.

Node2Vec: Optimization with Negative Sampling

- **Solution:** Negative Sampling (Softmax → Sigmoid)

$$\log\left(\frac{\exp\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v\right)}{\sum_{n\in V}\exp\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_n\right)}\right) \approx \log\left(\sigma\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v\right)\right) - \sum_{i=1}^{k}\log\left(\sigma\left(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_{n_i}\right)\right), n_i \sim P_V$$

Sigmoid function

random distribution over nodes

- Sample k negative nodes n_i each with probability proportional to its degree.
- Higher k gives more robust estimates.
- In practice, k from 5 to 20.

Node2Vec: Optimization using Stochastic Gradient Descent

- **Goal**: Optimize $\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$
- (Stochastic) Gradient Descent: a simple way to minimize L
- **SGD Algorithm:**
  - Initialize z_u at some randomized value for all nodes u.
  - Iterative until L converges:
    - Sample a node u, for all v calculate the derivative $\frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$.
    - For all v, update:

$$z_v \leftarrow z_v - \eta \frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$$

Learning rate

Random Walks-based Strategies: Summary

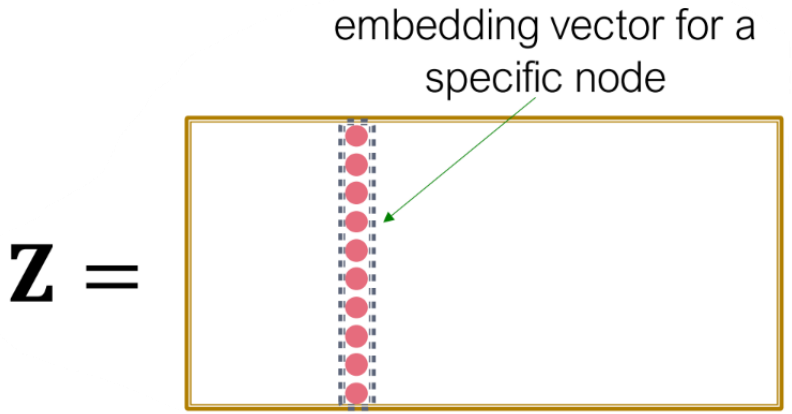
- **What are good embeddings?**

→ We can get good node embeddings that distances between them in embedding space reflect their similarities in the original graph network.

- **Two different methods:**
  - Naive: similar if 2 nodes are connected.
  - Random walk approaches.
- **Which method should we use?**
  - No one method wins in all cases.
  - Choose proper methods depending on specific tasks.

Random Walks for Shallow Encoding:

- **Goal**: Generate a lookup table for node embeddings
- Step-by-step:
  1. Run random walks for each node
  2. Collect the set of visited nodes for each node on the walks
  3. Optimize the embeddings Z_u using stochastic gradient descent



Proximity Preservation: LINE

- **Motivation**:
  - Preserving network proximities: LINE seeks to preserve both the **first-order proximity** (direct connections between nodes) and **second-order proximity** (shared neighborhood structures) in the learned embeddings.
  - Scalable objective functions:
    - consider first-order and second-order proximities separately.
    - different network types: directed, undirected, weighted or unweighted.
  - An edge-sampling algorithm is proposed to help stochastic gradient descent on weighted edges.

Jian Tang et al.: Line: Large-scale information network embedding

LINE: First-order Proximity and Preserving the First-order Proximity

- First-order proximity: local pairwise proximity between two connected nodes.
- For each node pair (v_i, v_j)
  - if $(v_i, v_j) \in E$, the first-order proximity between v_i and v_j is w_{ij}.
  - otherwise, the first-order proximity between v_i and v_j is 0.
- Given an undirected edge (v_i, v_j), the joint probability of v_i and v_j:

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)}$$

$\vec{u}_i$:Embedding of node v_i

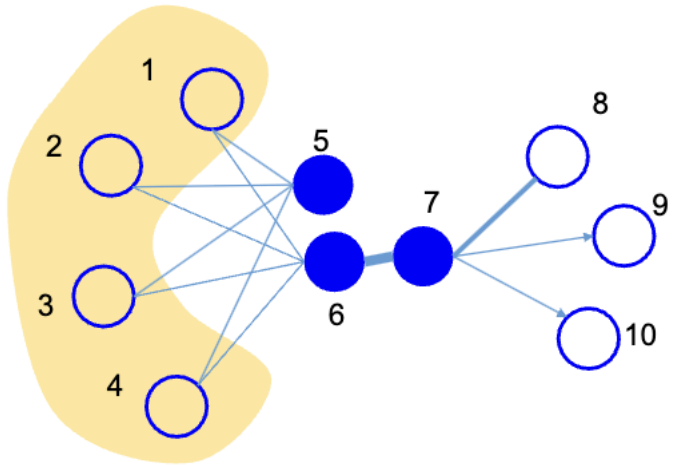
$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{(i',j')} w_{i'j'}}$$

- Objective:

$$O_1 = d(\hat{p}_1(\cdot,\cdot), p_1(\cdot,\cdot))$$

KL-divergence

$$\propto -\sum_{(i,j)\in E} w_{ij} \log p_1(v_i, v_j)$$



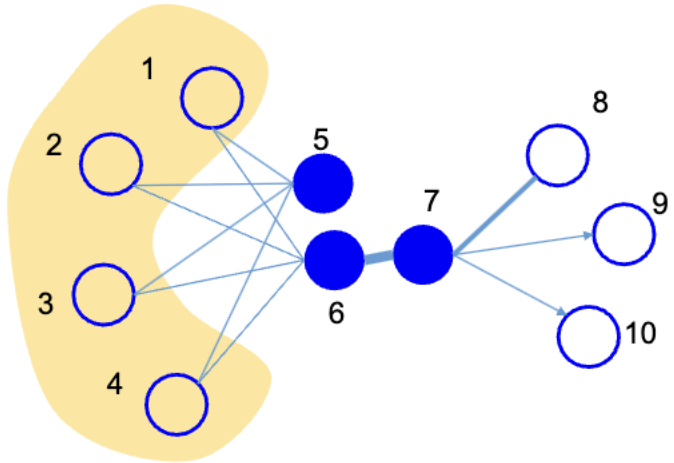
node 6 and 7 have a high first-order proximity

Jian Tang et al.: Line: Large-scale information network embedding

LINE: Second-order Proximity

- Second-order proximity: captures the 2-step relations between each pair of nodes.
- For each node pair (v_i, v_j)
  - determining by the number of common neighbors shared by the two nodes.

$$\hat{p}_u = (w_{u1}, w_{u2}, \dots, w_{u|V|})$$
$$\hat{p}_v = (w_{v1}, w_{v2}, \dots, w_{v|V|})$$



Second-order: node 5 and 6

$$\hat{p}_5 = (1,1,1,1,0,0,0,0,0,0)$$
$$\hat{p}_6 = (1,1,1,1,0,0,5,0,0,0)$$

Jian Tang et al.: Line: Large-scale information network embedding

LINE: Preserving the Second-order Proximity

- Given an undirected edge (v_i, v_j), the joint probability of v_i and v_j:

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)},$$

$\vec{u}_i$:Embedding of node v_i when i is a source node.

$\overrightarrow{u_i'}$:Embedding of node v_i when i is a target node.

$$\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{\sum_{k \in V} w_{ik}}$$

- Objective:

λ_i: Prestige of node in the network $\lambda_i = \sum_j w_{ij}$

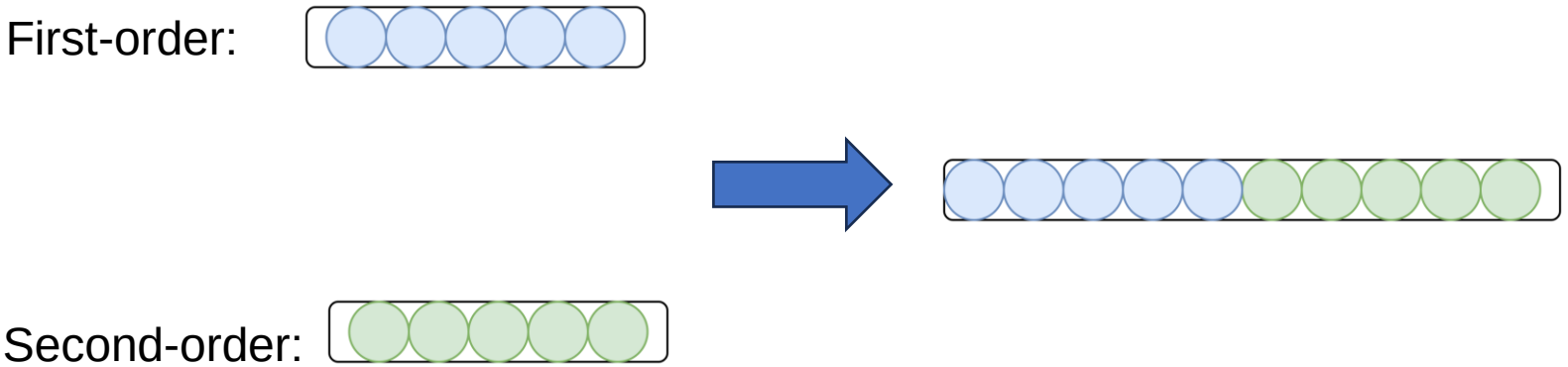
$$O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i)),$$

$$\propto -\sum_{(i,j) \in E} w_{ij} \log p_2(v_j|v_i).$$

Jian Tang et al.: Line: Large-scale information network embedding

LINE: Preserving both the Proximity

- Concatenate the embeddings individually learned by the two proximity:



Jian Tang et al.: Line: Large-scale information network embedding

LINE: Optimization

- Stochastic Gradient Descent + Negative Sampling.
  - Randomly sample an edge and multiple negative edges.
- The gradient w.r.t the embedding with edge (i,j)

$$\frac{\partial O_2}{\partial \vec{u}_i} = w_{ij} \cdot \frac{\partial \log p_2(v_j|v_i)}{\partial \vec{u}_i}$$

Multiplied by the weight of the edge w_{ij}

- Problematic when the weights of the edges diverge
  - The scale of the gradients with different edges diverges.
- **Solution:** Edge sampling
  - Sample the edges according to their weights and treat the edges as binary.
- Complexity: $\Theta(dK|E|)$
  - Linear to the dimension d, number of negative samples K, and number of edges E.

LINE: Algorithm

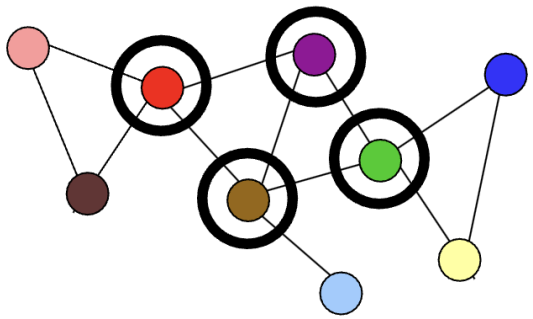
- **Initialization**: Initialize the embedding vectors for all nodes randomly.
- **Edge Sampling**: Sample edges from the network based on their weights.
- **Gradient Descent**: For each sampled edge, update the embedding vectors using SGD.
- **Negative Sampling**: For each positive edge, sample negative edges and update the embedding vectors to maximize the difference between positive and negative samples.
- **Iteration**: Repeat the edge sampling and gradient descent steps until convergence.

→ Same as DeepWalk, Node2Vec, and Node2Vec+ but consider the preservation of proximity and directed edge sampling.

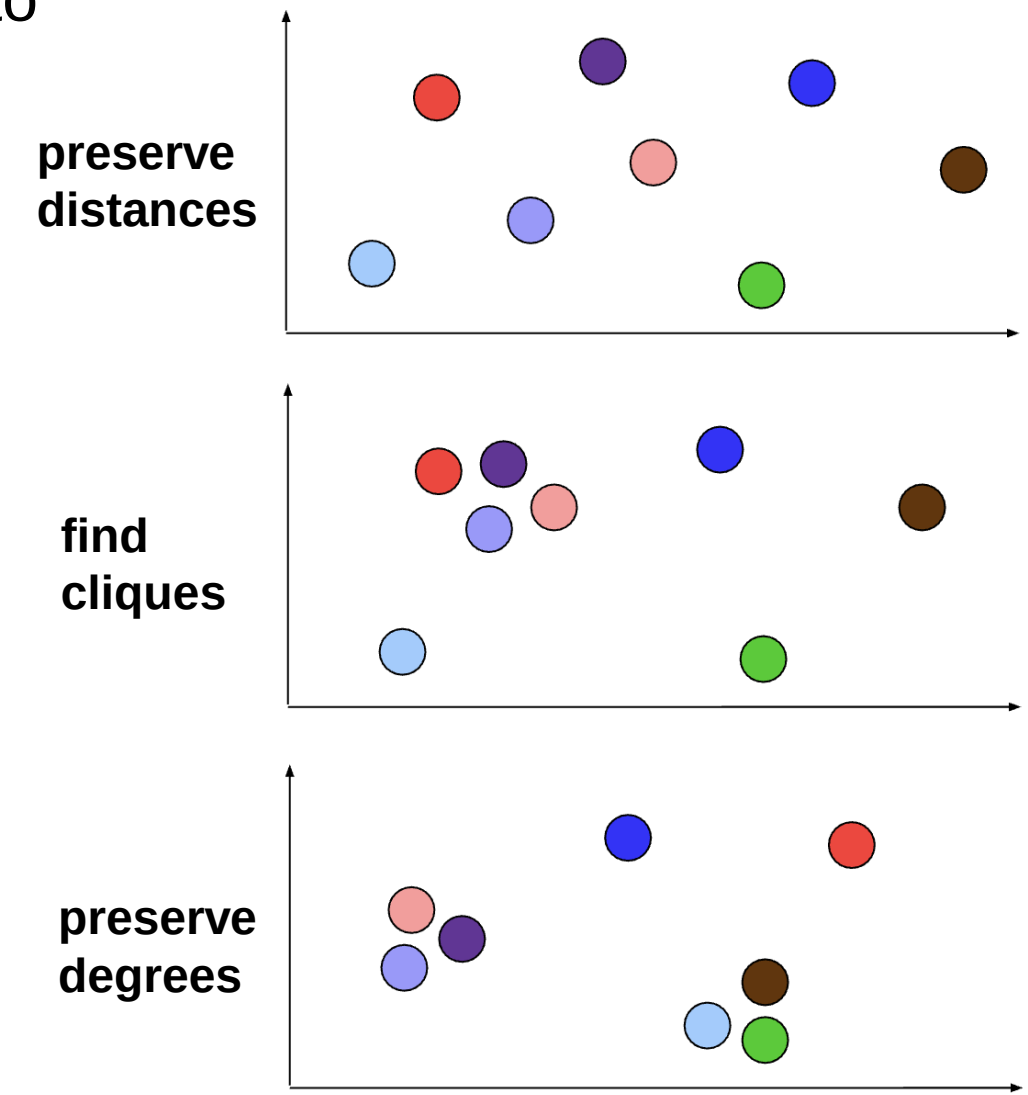
Jian Tang et al.: Line: Large-scale information network embedding

Role-based: Node Representations and Structural Identity

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



Red, Green: automorphism.
Purple, Brown: structurally similar.



Leonardo F. R. Ribeiro et al: struc2vec: Learning Node Representations from Structural Identity

struc2vec: Learning Node Representations from Structural Identity

- **Ideas:** based on structural identity.
- 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.

struc2vec: Step 1 - Structural Similarity

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

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

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

$g(.\ ,\ .) = 0.33$ $g(.\ ,\ .) = 3.33$ $g(.\ ,\ .) = 1$

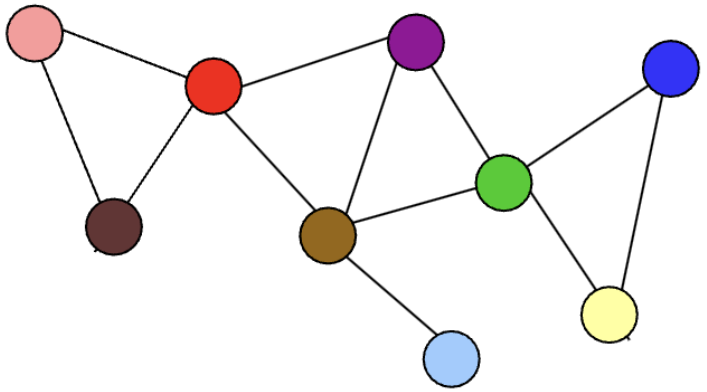
- $f_k(u,v)$: structural distance between nodes u and v considering first k rings
  - $f_k(u,v) = f_{k-1}(u,v) + g(s(R_k(u)), s(R_k(v)))$.

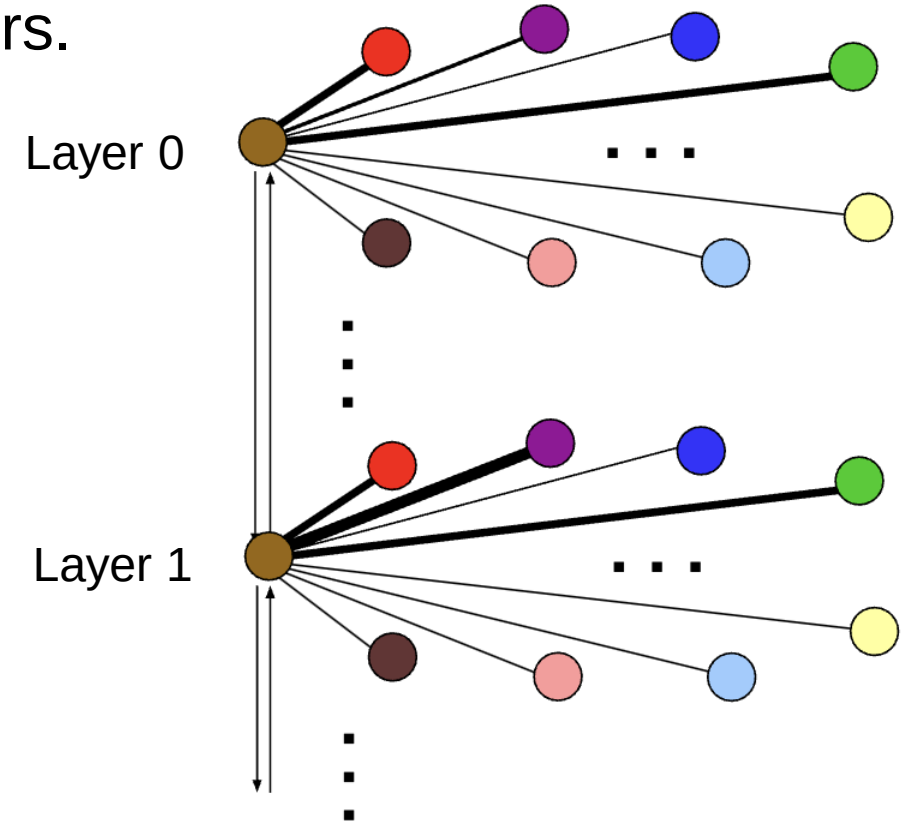
$f_0(u,v) = 0.33$ $f_1(u,v) = 3.66$ $f_2(u,v) = 4.66$

Leonardo F. R. Ribeiro et al: struc2vec: Learning Node Representations from Structural Identity

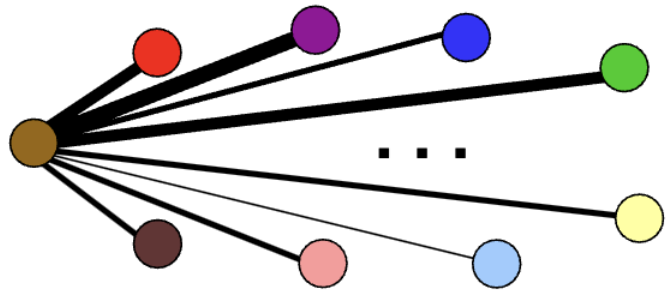
struc2vec: Step 2 - Multi-layer graph

- Encodes structural similarity between all node pairs.





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



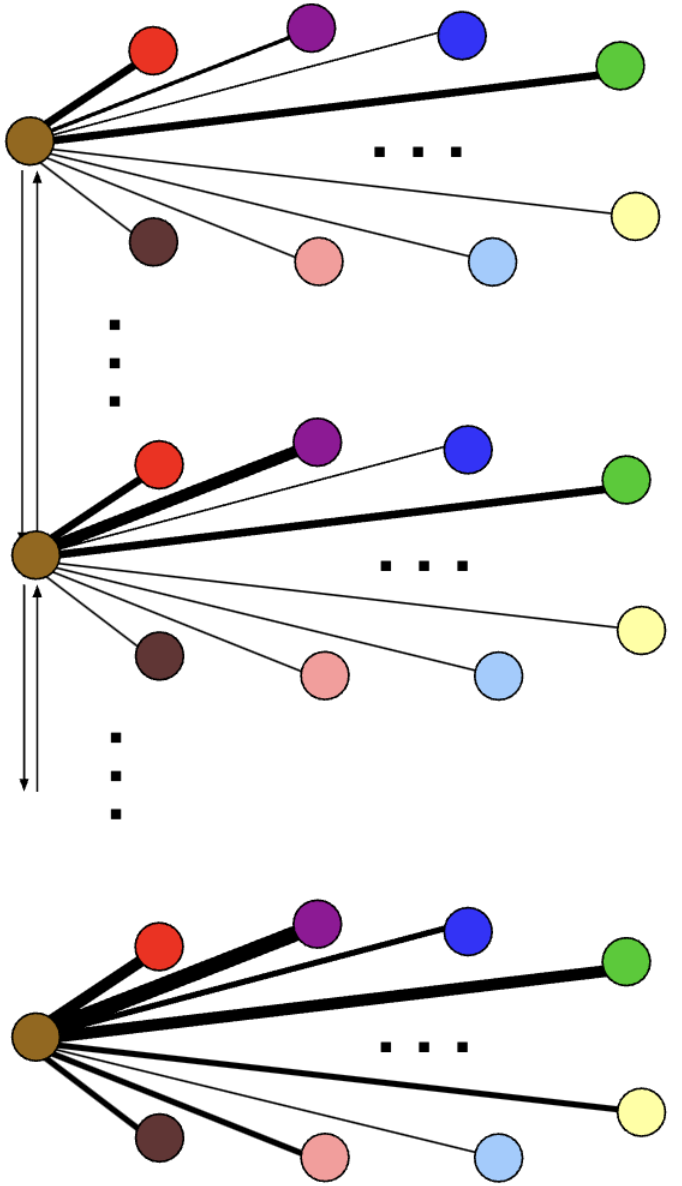
struc2vec: Step 3 - Generate Context

- Context generated by biased random walk (same as Node2vec)
  - 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
  - choose up/down according to edge weight.
  - RW prefer layer with less similar neighbors.

struc2vec: Step 4 – Learn Representation

- For each node, generate set of independent and relative short random walks
  - context for node; 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.



Leonardo F. R. Ribeiro et al: struc2vec: Learning Node Representations from Structural Identity

struc2vec: Optimization

- Reduce time to generate/store multi-layer graph and context for nodes:
  - Option 1: Reduce length of degree sequences
    - use pairs (degree, number of occurrences).
  - Option 2: Reduce number of edges in multi-layer graph
    - only log n neighbors per node.
  - Option 3: Reduce number of layers in multi-layer graph
    - fixed (small) number of layers.
  - Scales quasi-linearly
    - over 1 million nodes.

Leonardo F. R. Ribeiro et al: struc2vec: Learning Node Representations from Structural Identity

Subgraph2Vec: Motivation

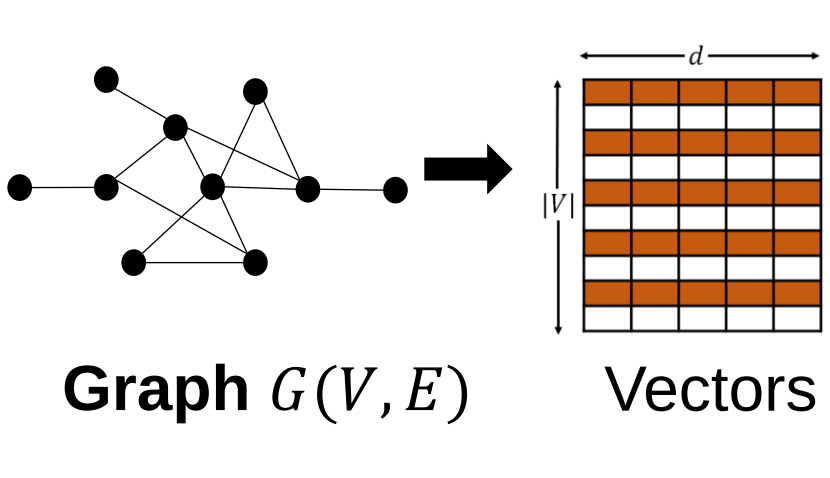
- Popular previous works include

DeepWalk[Perozzi+, KDD2014]

Node2vec[Grover+, KDD 2016]

SDNE[Wang+, KDD 2016]

LINE[Tang+,WWW 2015]



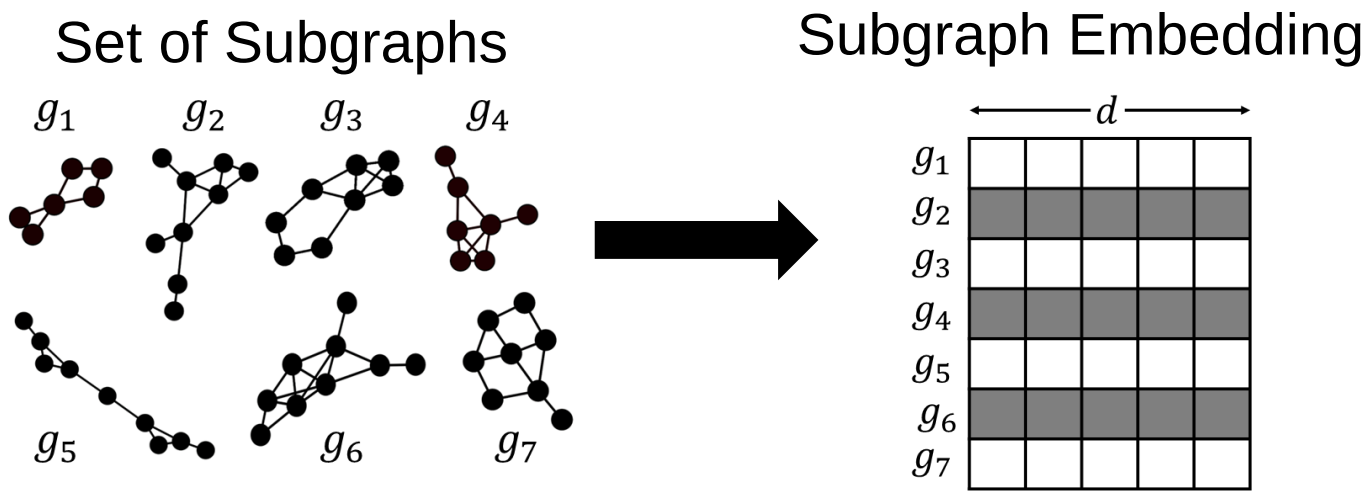
Limited just to the node embeddings

Subgraph2Vec model: Idea

- Learning representation of substructures
  - Extend the WL relabeling strategy to define a proper context for a given subgraph.
  - A modification to the skipgram model enabling it to capture varying length radial contexts

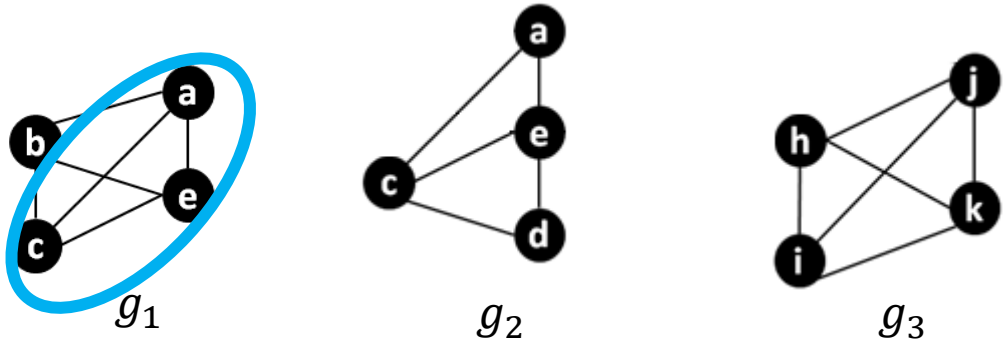
Problem Formulation: Setting

- **Given**:
  - A set $S = \{g_1, g_2, \ldots, g_n\}$ of subgraphs
  - Typically for the same graph
  - An integer d
- **Learning**:
  - d-dimensional embedding for each subgraph
  - Such that pre-defined subgraph property is preserved



Problem formulation: Challenges

- What subgraph property to preserve?
  - Neighbourhood Property:
    - Captures neighbourhood information within the subgraph



- Subgraph g_1 and g_2 share neighbourhood
- Subgraph g_3 does not

Algorithm 1: Generate rooted subgraphs

- Generate rooted subgraphs around every node in a given graph

- Considers all the rooted subgraphs (up to a certain degree) of neighbours of r as the context of target subgraphs.

Algorithm 2: GETWLSUBGRAPH (v, G, d)

input : v: Node which is the root of the subgraph
$G = (V, E, \lambda)$: Graph from which subgraph has to be extracted
d: Degree of neighbours to be considered for extracting subgraph

output: $sg_v^{(d)}$: rooted subgraph of degree d around node v

```
1 begin
2     sg_v^(d) = {}
      if d = 0 then
3         sg_v^(d) := λ(v)
4     else
5         N_v := {v' | (v, v') ∈ E}
6         M_v^(d) := {GETWLSUBGRAPH(v', G, d − 1) | v' ∈ N_v}
7         sg_v^(d) := sg_v^(d) ∪ GETWLSUBGRAPH
          (v, G, d − 1) ⊕ sort(M_v^(d))
8     return sg_v^(d)
```

Algorithm 2: Learn embeddings of those subgraphs

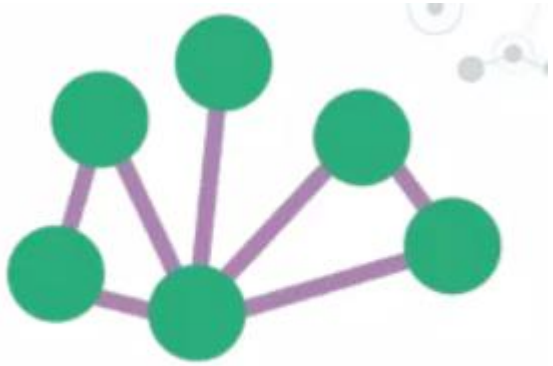
- The skipgram model maximizes co-occurrence probability among the sub-graphs that appear within a given context window.

Algorithm 3: RADIALSKIPGRAM $(\Phi, sg_v^{(d)}, G, D)$

```
1 begin
2     context_v^(d) = {}
3     for v' ∈ NEIGHBOURS(G, v) do
4         for ∂ ∈ {d − 1, d, d + 1} do
5             if (∂ ≥ 0 and ∂ ≤ D) then
6                 context_v^(d) = context_v^(d) ∪
                  GETWLSUBGRAPH(v', G, ∂)
7     for each sg_cont ∈ context_v^(d) do
8         J(Φ) = −log Pr (sg_cont | Φ(sg_v^(d)))
9         Φ = Φ − α ∂J/∂Φ
```

2. Homogeneous vs Heterogeneous Graphs

- Single node type and single edge type
  - E.g.,
    - Users **follow** other Users.
- Heterogeneous Graphs
  - Multiple node and/or edge types.
  - E.g.,
    - Users follow other Users.
    - Users favourite tweets.
    - Users reply to tweets.



homogeneous

heterogeneous

https://medium.com/stellargraph/knowing-your-neighbours-machine-learning-on-graphs-9b7c3d0d5896

Heterogeneous Graphs

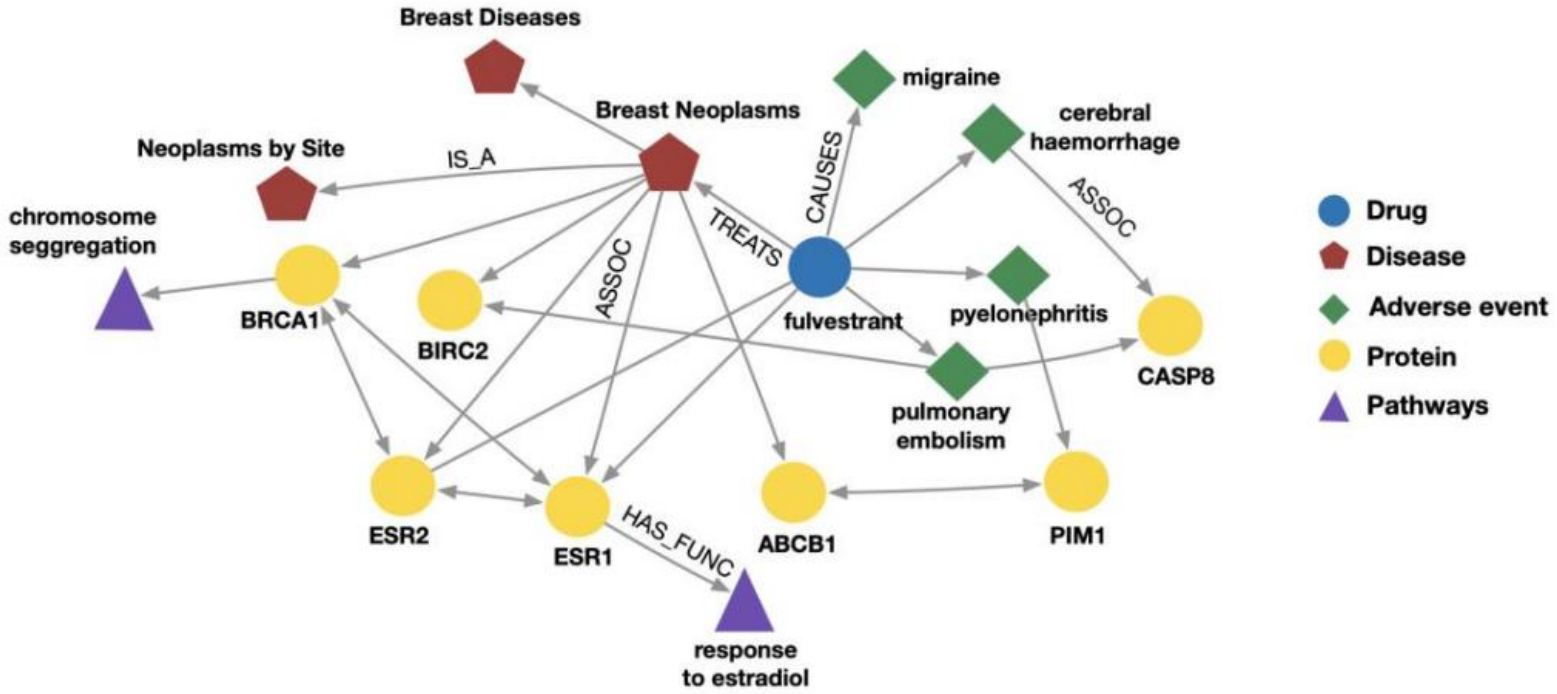
- A heterogeneous graph is defined as:

$$G = (V, E, R, T)$$

- Nodes with node types $v_i \in V$.
- Edges with relation types $(v_i, r, v_j) \in E$.
- Node type $T(v_i)$.
- Relation type $r \in R$.

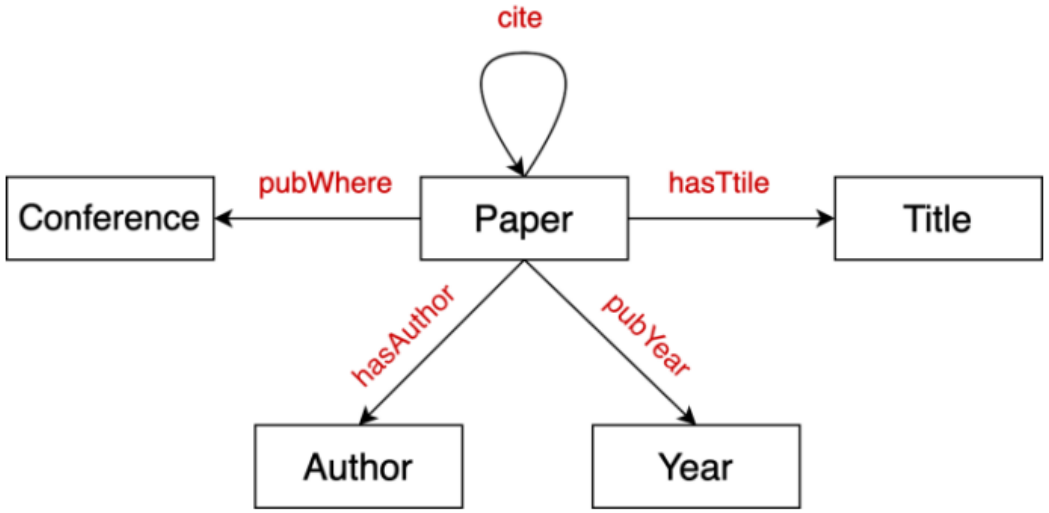
Heterogeneous Graphs: Examples

- Biomedical Knowledge Graphs
  - Example node: Migraine.
  - Example edge: (fulvestrant, Treats, Breast Neoplasms).
  - Example node type: Protein.
  - Example edge type (relation): Causes.



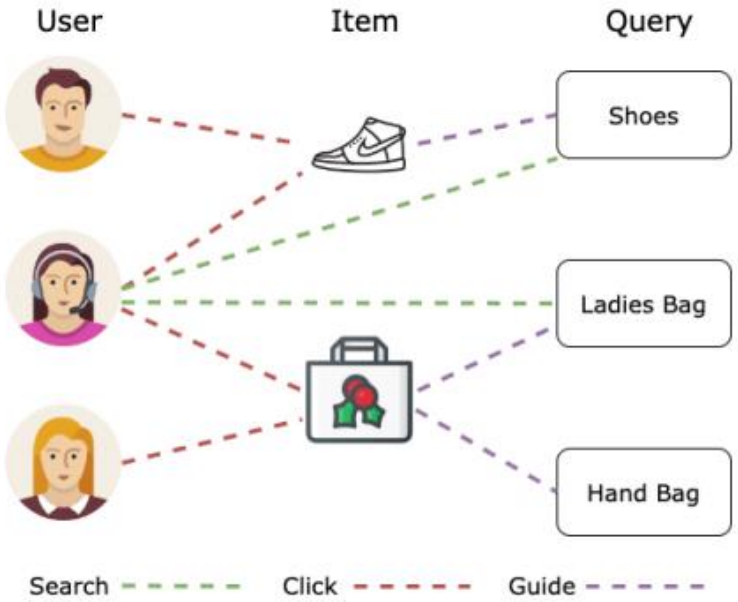
Heterogeneous Graphs: Examples

- Academic Graphs:
  - Example node: ICML.
  - Example edge: (GraphSAGE, NeurIPS).
  - Example node type: Author.
  - Example edge type (relation): pubYear.



Many Graphs are Heterogeneous Graphs

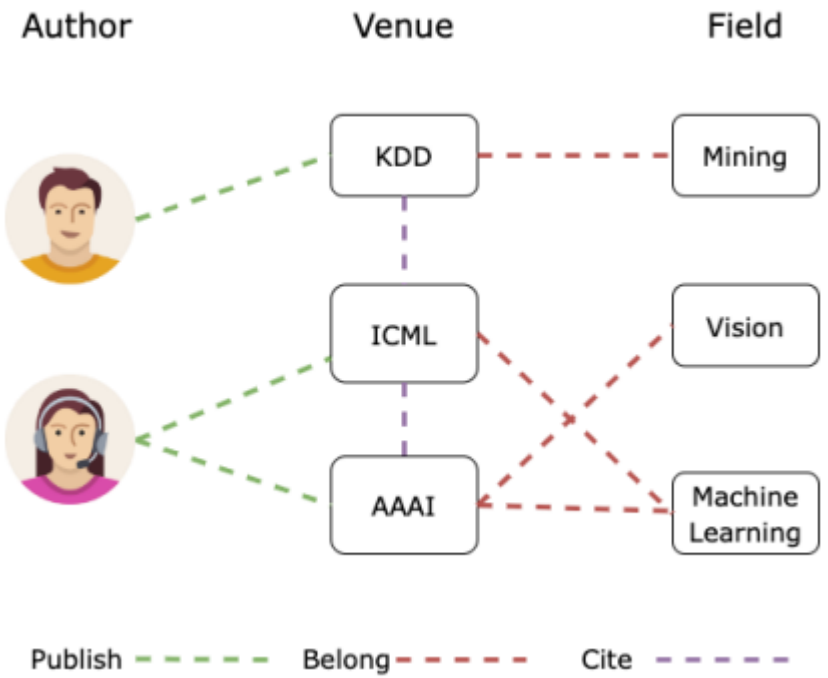
- Example: E-Commerce Graph
- Node types: User, Item, Query, Location, ...
- Edge types: Purchase, Visit, Guide, Search, …
- Different node type's features spaces can be different!



https://ar5iv.labs.arxiv.org/html/1908.02569

Many Graphs are Heterogeneous Graphs

- Example: Academic Graph
- Node types: Author, Paper, Venue, Field, ...
- Edge types: Publish, Cite, …
- Benchmark dataset: Microsoft Academic Graph



Why can’t we use homogeneous learning methods?

- **Complex Structure**
  - The structure in Heterogeneous Graphs is highly semantic-dependent, such as a meta-path structure.
- **Heterogeneous Attributes**
  - different types of nodes and edges have different attributes which are located in different feature spaces.
  - To effectively fuse the attributes of neighbors Heterogeneous methods have to overcome this heterogeneity.

Meta path [Han VLDB'11]

- A sequence of node class sets connected by edge types:

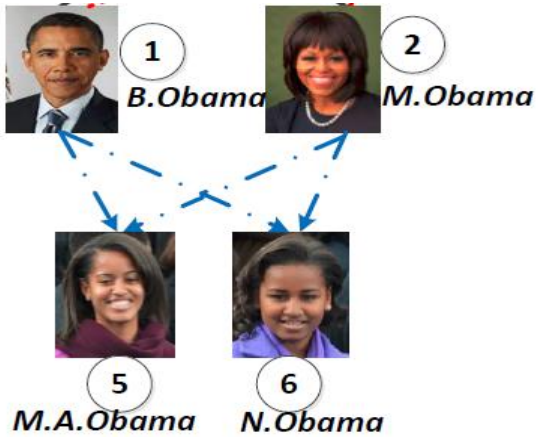
$$\Pi^{1\ldots n} = C_1 \xrightarrow{e_1} \ldots C_i \xrightarrow{e_i} \ldots C_n$$

- Benefits of Meta Paths.
- Multi-hop relationships instead of direct links.
- Combine multiple relationships.

$$m1 : \text{USPresident} \xrightarrow{\text{hasChild}} \text{Person} \xrightarrow{\text{hasChild}^{-1}} \text{USFirstLady},$$

$$m2 : \text{USPresident} \xrightarrow{\text{memberOf}} \text{USPoliticalParty} \xrightarrow{\text{memberOf}^{-1}} \text{USFirstLady},$$

$$m3 : \text{USPresident} \xrightarrow{\text{citizenOf}} \text{Country} \xrightarrow{\text{citizenOf}^{-1}} \text{USFirstLady}.$$



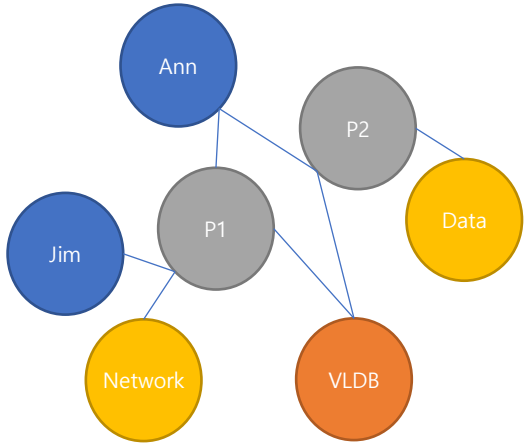
Yuxiao Dong , et. Al., metapath2vec: Scalable Representation Learning for Heterogeneous Networks (KDD 2017)

Similarity score

- Similarity score for a node pair following a single meta-path
  - **Path Count** (PC) [Han ASONAM'11]
    - Number of the paths following a given meta-path.
  - **Path Constrained Random Walk** (PCRW) [Cohen KDD'11]
    - Transition probability of a random walk following a given meta-path.
- Similarity score for a node pair following a combination of multiple meta-paths
  - Aggregate Function F to combine the similarity scores for each single meta path.

Meta Paths Generation

- **Meta Path**
  - Two objects can be connected via different connectivity paths
  - E.g., two authors can be connected by
    - “author-paper-author” (APA)
    - “author-paper-author-paper-author” (APAPA)
    - “author-paper-venue-paper-author” (APCPA)
- Each connectivity path represents a different semantic meaning and implies different similarity semantics.



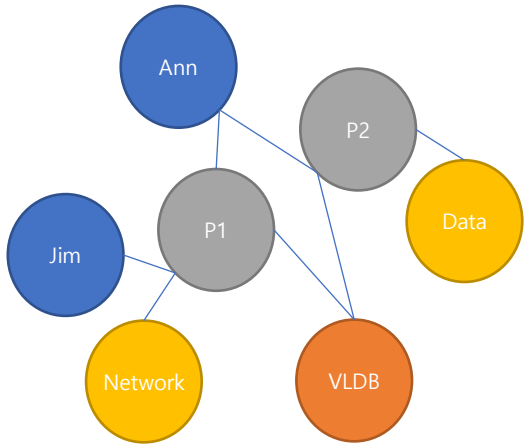
Yuxiao Dong , et. Al., metapath2vec: Scalable Representation Learning for Heterogeneous Networks (KDD 2017)

Meta Paths Generation

- A meta path is a meta level description of the topological connectivity between objects
  - Given a Network Schema, A meta path can be defined as

$$A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_l} A_{l+1}$$

- Can be considered as a new relation defined on type A_1 and A_{l+1}



Conventional Similarity measures

- Path Count:
  - The number of path instances p between x and y following P:

$$s(x, y) = |\{p : p \in P\}|$$

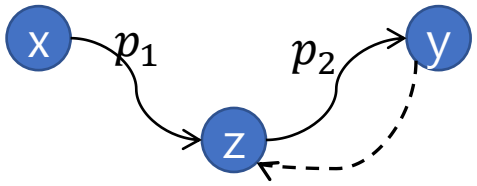
- Random Walk:
  - The probability *Prob(p)* of the random walk that starts from x and ends with y following meta path *P*, which is the sum of the probabilities of all the path instances *p*

$$s(x, y) = \sum_{p \in P} Prob(p)$$

Conventional Similarity measures (2)

- Pairwise Random Walk
  - For a meta path P that can be decomposed into two shorter meta paths with the same length P = $(P_1 P_2)$, pairwise random walk probability is the probabilities starting from x and y and reaching the same middle object z

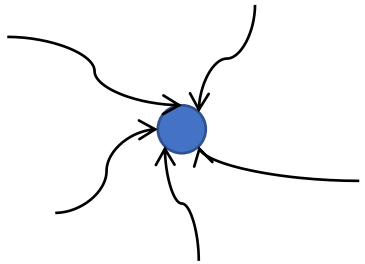
$$s(x,y) = \sum_{(p_1 p_2)\in(P_1 P_2)} \Pr ob(p_1) \Pr ob(p_2^{-1})$$



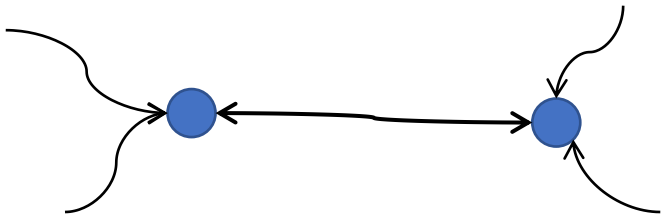
Yuxiao Dong , et. Al., metapath2vec: Scalable Representation Learning for Heterogeneous Networks (KDD 2017)

PathSim: A Novel Meta Path-Based Similarity Measure

- Similarity in terms of ‘Peers’
  - Two similar peer object should not only be strongly connected, but also share comparable visibility.



- Path count and Random walk (RW)
  - Favor highly visible objects (objects with large degrees)
- Pairwise random walk (PRW)
  - Favor pure objects (objects with highly skewed scatterness in their in-links or out-links)
- PathSim
  - Favor “peers” (objects with similar visibility and strong connectivity under the given meta path)

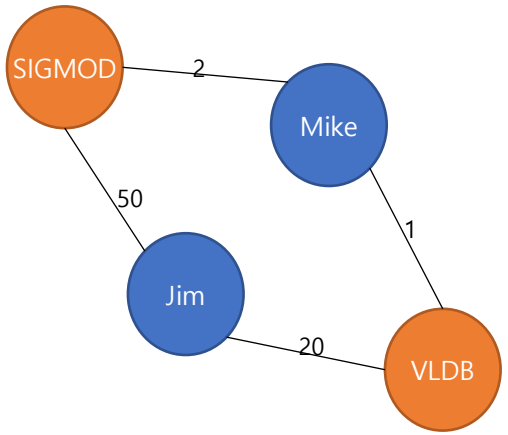


Yuxiao Dong , et. Al., metapath2vec: Scalable Representation Learning for Heterogeneous Networks (KDD 2017)

PathSim: A Novel Meta Path-Based Similarity Measure (2)

- Restricted on Round-Trip Meta Path
  - A round-trip meta path is a path of the form of P = $\left(P_l P_l^{-1}\right)$
  - Guarantees a symmetric relation:

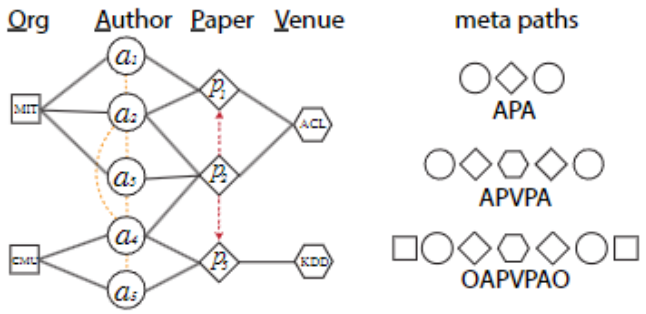
$$s(x,y) = \frac{2 \times |\{p_{x \rightsquigarrow y} : p_{x \rightsquigarrow y} \in \mathcal{P}\}|}{|\{p_{x \rightsquigarrow x} : p_{x \rightsquigarrow x} \in \mathcal{P}\}| + |\{p_{y \rightsquigarrow y} : p_{y \rightsquigarrow y} \in \mathcal{P}\}|}$$



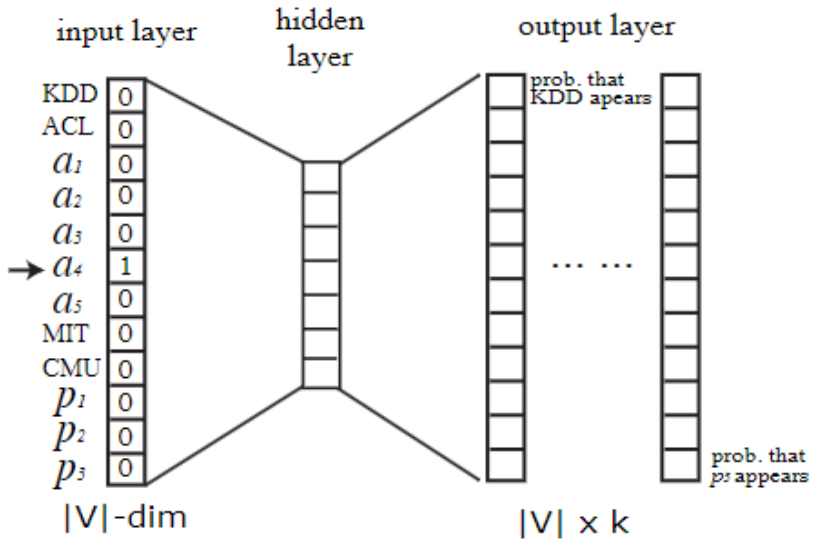
$$s(\text{Mike, Jim}) = \frac{2*(2*50+1*20)}{(2*2+1*1)+(50*50+20*20)} = 0.0826$$

Yuxiao Dong , et. Al., metapath2vec: Scalable Representation Learning for Heterogeneous Networks (KDD 2017)

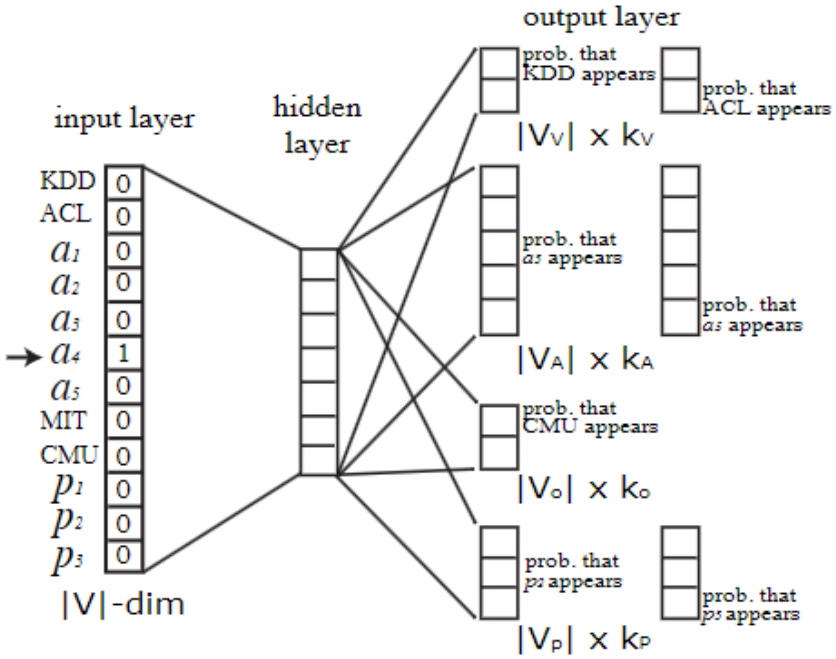
- **Solution**: meta-path based random walk (inspired by DeepWalk and Node2Vec)
  - metapath2vec and metapath2vec++.
- **Goal**: to generate paths that can capture both the semantic and structural correlations between different types of nodes, facilitating the transformation of heterogeneous network structures into skip-gram.



Heterogenous network

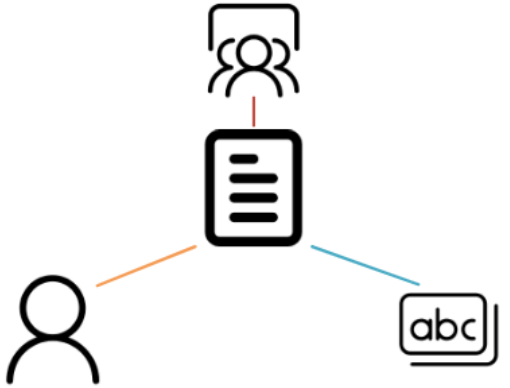


Skip-gram in metapath2vec



Skip-gram in metapath2vec++

- Network schema $S = (V, R)$ of graph G
  - directed graph defined over node types V and with edges as relations from R



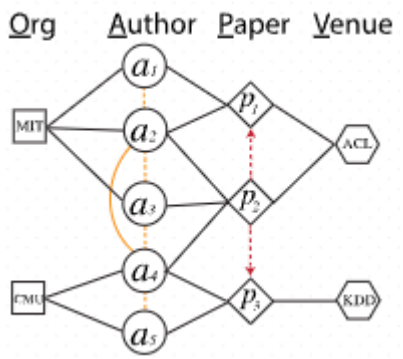
- meta-path: based on network schema S.
  - Denoted as $V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \cdots V_t \xrightarrow{R_t} V_{t+1} \cdots \xrightarrow{R_{l-1}} V_l$
  - Node types $V_1, V_2, \ldots, V_l \in V$ and edge type $R_1, R_2, \ldots, R_{l-1} \in R$

- Each meta-path captures the proximity between the nodes on its two ends from a particular semantic perspective.

https://blog.reachsumit.com/posts/2023/05/shallow-heterogeneous-graphs-rep/

- **Metapath2Vec**:
- Given a meta-path scheme:

$$V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \cdots V_t \xrightarrow{R_t} V_{t+1} \cdots \xrightarrow{R_{l-1}} V_l$$



- The transition probability at step i is defined as:

$$p(v^{i+1}|v_t^i, \mathcal{P}) = \begin{cases} \frac{1}{|N_{t+1}(v_t^i)|} & (v^{i+1}, v_t^i) \in E, \phi(v^{i+1}) = t+1 \\ 0 & (v^{i+1}, v_t^i) \in E, \phi(v^{i+1}) \neq t+1 \\ 0 & (v^{i+1}, v_t^i) \notin E \end{cases}$$

- Recursive guidance for random walkers, i.e.,

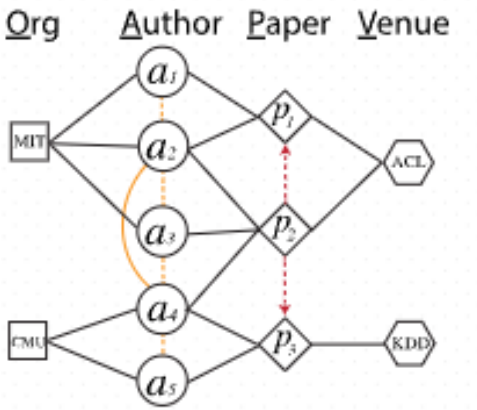
$$p(v^{i+1}|v_t^i) = p(v^{i+1}|v_1^i), \text{ if } t = l$$

- **Metapath2Vec**:
- Given a meta-path scheme (Example):

OAPVPAO

- In a traditional random walk procedure, the next step of a walker on node a4 transitioned from node CMU can be all types of nodes surrounding it - a2,a3, a5, p2, p3, and CMU.

- Under the meta-path scheme ‘OAPVPAO’, for example, the walker is biased towards paper nodes (P) given its previous step on an organization node CMU (O), following the semantics of this meta-path.



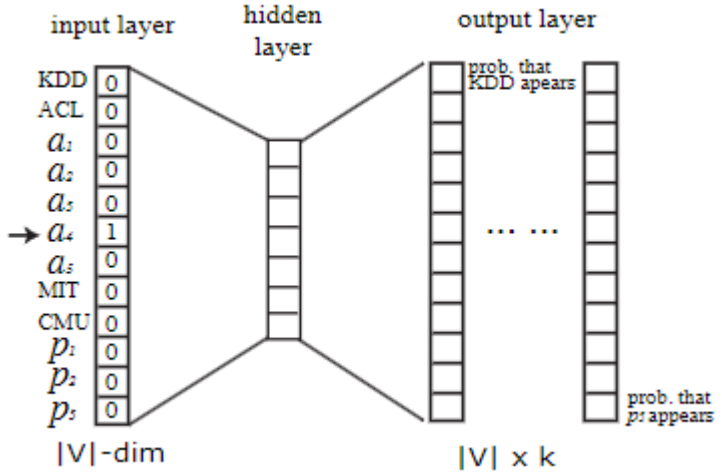
Yuxiao Dong et al:"metapath2vec: Scalable representation learning for heterogeneous networks." *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017.

- Softmax in Metapath2Vec

$$p(c_t|v;\theta) = \frac{e^{X_{c_t} \cdot X_v}}{\sum_{u \in V} e^{X_u \cdot X_v}},$$

Not consider node type

- The potential issue of skip-gram for heterogeneous network embedding:
  - To predict the context node c_t (type t) given a node v, metapath2vec encourages all types of nodes to appear in this context position.



Yuxiao Dong et al:"metapath2vec: Scalable representation learning for heterogeneous networks." *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017.

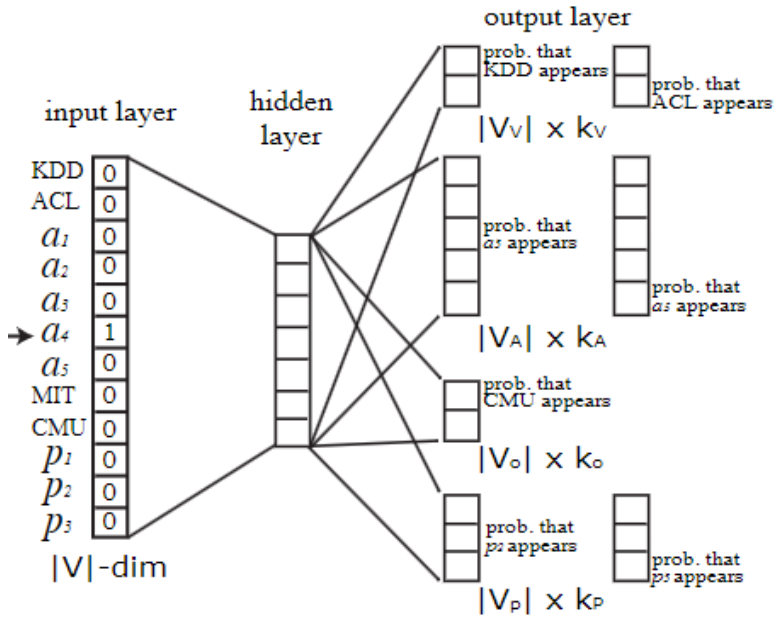
- **Metapath2Vec++**: Heterogeneous Skip-Gram
- Objective function (heterogeneous negative sampling):

$$O(\mathrm{X}) = \log \sigma(X_{c_t} \cdot X_v) + \sum_{m=1}^{M} \mathbb{E}_{u_t^m \sim P_t(u_t)} [\log \sigma(-X_{u_t^m} \cdot X_v)]$$

- Softmax in Metapath2Vec++

$$p(c_t | v; \theta) = \frac{e^{X_{c_t} \cdot X_v}}{\sum_{u_t \in V_t} e^{X_{u_t} \cdot X_v}}$$

Consider node type t



- Stochastic gradient descent

$$\frac{\partial O(\mathrm{X})}{\partial X_{u_t^m}} = (\sigma(X_{u_t^m} \cdot X_v - \mathbb{I}_{c_t}[u_t^m]))X_v$$

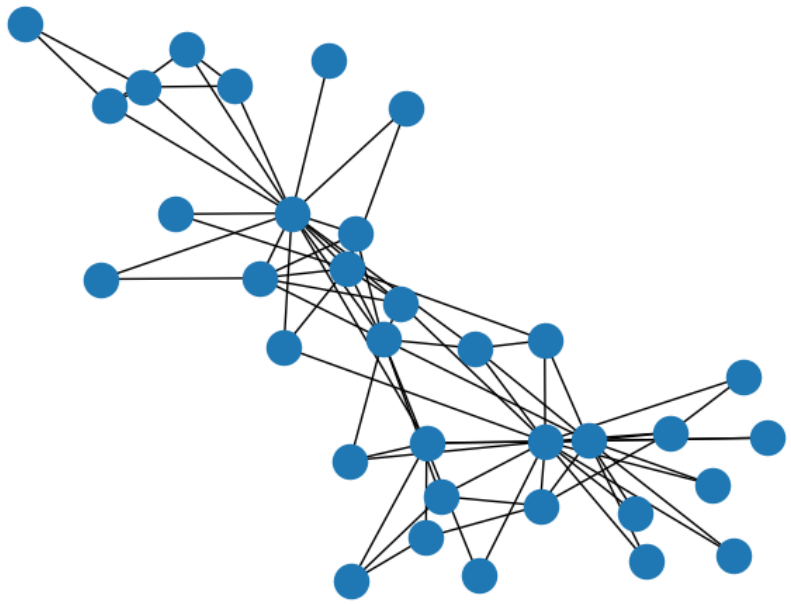
$$\frac{\partial O(\mathrm{X})}{\partial X_v} = \sum_{m=0}^{M} (\sigma(X_{u_t^m} \cdot X_v - \mathbb{I}_{c_t}[u_t^m]))X_{u_t^m}$$

Yuxiao Dong et al:"metapath2vec: Scalable representation learning for heterogeneous networks." *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017.

Metapath2Vec: Sample code with Karate Zachary

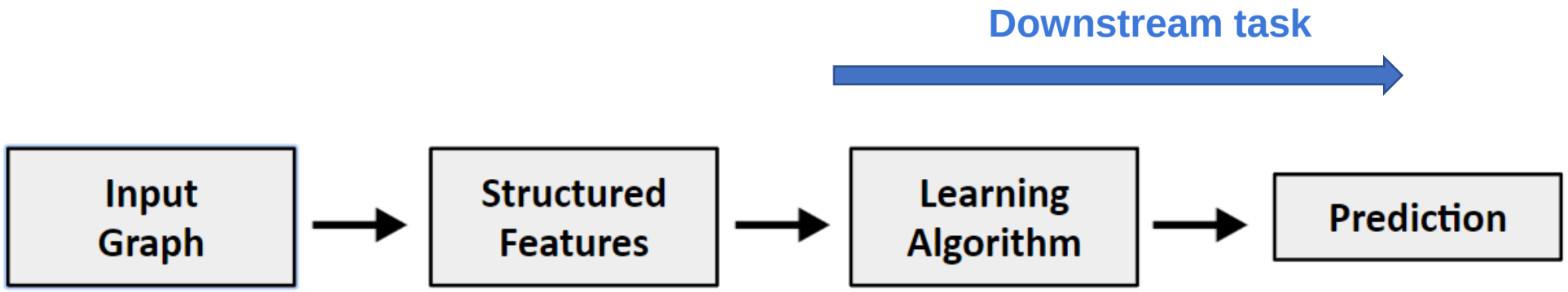

```python
import networkx as nx
import random

# Import the Karate Club graph using NetworkX and create an adjacency matrix
karate_graph = nx.karate_club_graph()
adjacency_list = nx.adjacency_matrix(karate_graph, dtype=int)
adjacency_matrix_array = adjacency_list.toarray()

def random_walk(adj_list, node, walk_length):
  walk = [node]        # Walk starts from this node

  for i in range(walk_length-1):
    node = adj_list[node][random.randint(0,len(adj_list[node])-1)]
    walk.append(node)

  return walk

# Perform random walks on the graph
num_walks = 6

for node in karate_graph.nodes():
  print("Node " + str(node) + " :" + str(random_walk(adjacency_matrix_array, node, num_walks)))
```

```
Node 0 :[0, 2, 5, 0, 0, 0]
Node 1 :[1, 5, 0, 0, 0, 0]
Node 2 :[2, 0, 0, 3, 0, 0]
Node 3 :[3, 3, 3, 0, 2, 4]
Node 4 :[4, 0, 0, 0, 2, 5]
Node 5 :[5, 0, 2, 0, 0, 0]
Node 6 :[6, 0, 0, 0, 3, 3]
Node 7 :[7, 0, 2, 5, 0, 0]
Node 8 :[8, 0, 5, 0, 0, 2]
Node 9 :[9, 0, 0, 3, 0, 0]
Node 10 :[10, 0, 0, 2, 0, 3]
Node 11 :[11, 3, 0, 2, 0, 3]
Node 12 :[12, 0, 2, 0, 2, 2]
Node 13 :[13, 0, 0, 0, 2, 0]
Node 14 :[14, 3, 0, 2, 0, 3]
Node 15 :[15, 0, 0, 3, 3, 0]
Node 16 :[16, 0, 3, 3, 0, 2]
```



What after Node Embeddings?

- Once we have node embeddings (independent to task), we can continue to the downstream prediction.



Shallow Encoding

- SVM
- Random Forest
- XGBoost
- DNN

- Node-level
- Edge-level
- Graph-level

How to Use these Node Embeddings?

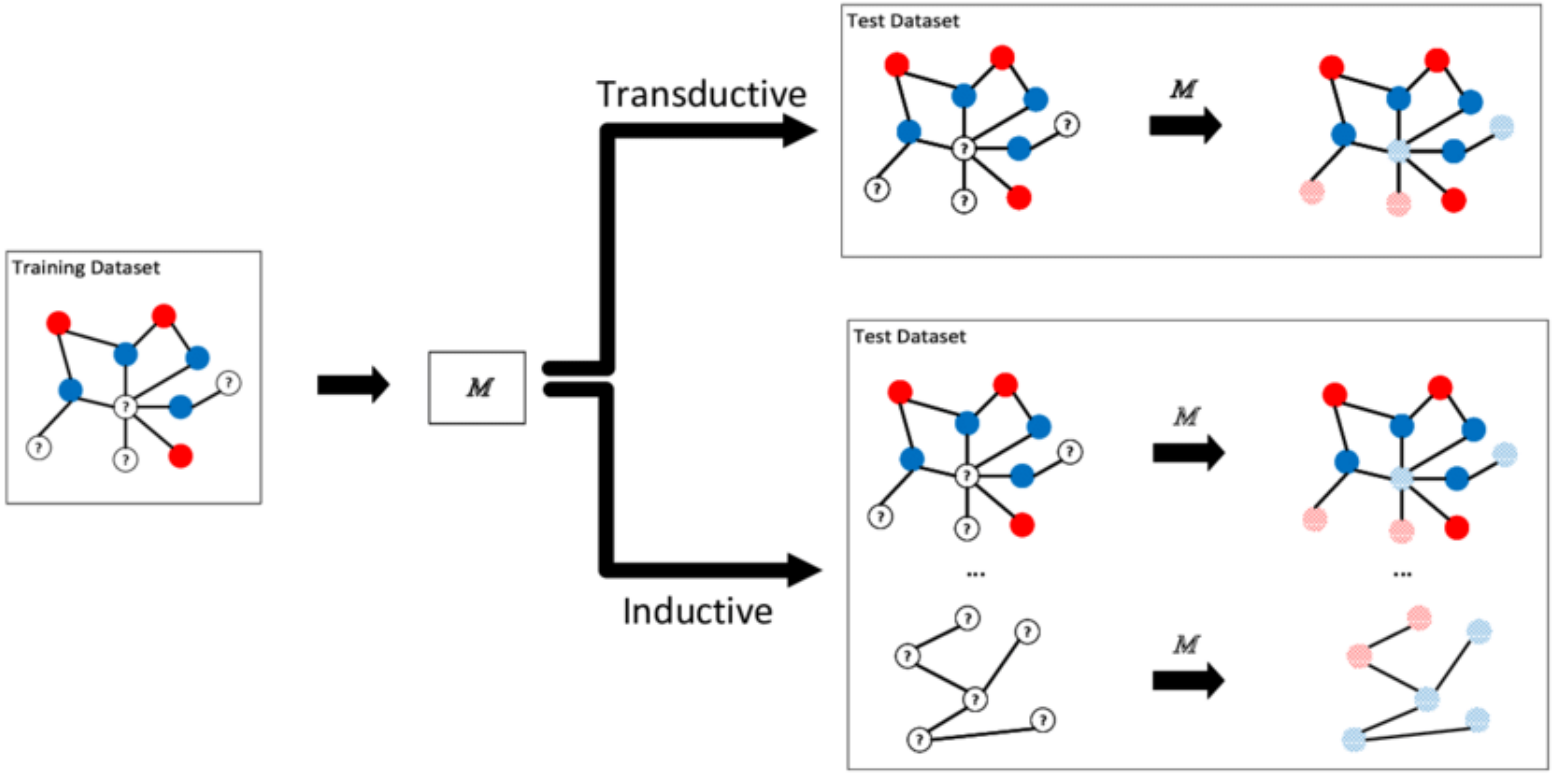
- **Given a node v_i in a graph, we have its embedding $\boldsymbol{Z_i}$, we can do:**
  1. Clustering/community detection: Cluster $\boldsymbol{Z_i}$.
  2. Node classification: Predict label of node v_i based on $\boldsymbol{Z_i}$.
  3. Link prediction: Predict edge (v_i, v_j) based on $(\boldsymbol{Z_i}, \boldsymbol{Z_j})$.
     - Concatenate: f($\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$) = g([$\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$]).
     - Hadamard: f($\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$) = g($\boldsymbol{Z_i} * \boldsymbol{Z_j}$) (per position product).
     - Sum/Average: f($\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$) = g($\boldsymbol{Z_i} + \boldsymbol{Z_j}$).
     - Distance: f($\boldsymbol{Z_i}$, $\boldsymbol{Z_j}$) = g($\|\boldsymbol{Z_i} - \boldsymbol{Z_j}\|_2$).
  4. Graph classification: aggregate node embeddings to form graph embedding $\boldsymbol{Z_G}$. Predict graph label based on graph embedding $\boldsymbol{Z_G}$.

Limitations of Shallow Encoders

- Limitations of shallow embedding methods:
  - O(|V|) parameters are needed:
    - Every node has its own unique embedding.
    - No shared parameters between nodes.
  - Inherently “transductive”:
    - Cannot generate embedding for nodes that are not seen during training.
  - Do not incorporate node features:
    - Nodes in many graphs have node features that we can and should leverage.

Transductive vs. Inductive Learning

- In transductive learning, for new coming graphs, we have to train from scratch to get the embeddings.
  - However, in inductive learning, the way we create embeddings can be generalized to unseen graphs.
- Inductive learning on graphs → Deep Encoder → Graph Neural Networks (GNNs)



네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA