DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Part II

Part 2: Basic GNNs & Applications

Lecture 03

Node Embeddings with GNNs

Discover node embeddings, compact numerical representations of graph nodes capturing essential structural and semantic information. Explore two fundamental techniques: Matrix Factorization, and Skip-gram nodes embeddings. Both approaches offer unique insights into effective node representation.

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Graph Embedding – Why?

Classical Machine Learning and Feature Engineering:

Feature engineering usually transforms data into a meaningful compact representation.

Works well for various problems, but not always optimal for graphs due to their complex structure.

Challenges with Graphs:

Graphs have a structured nature, making it challenging to find suitable representations that capture all useful information.

Kernel Functions and Specific Features:

Applying kernel functions or engineering features to represent desired properties.

Time-consuming, might capture only a subset of essential information.

Advancements: Graph Representation Learning:

Recent decade witnessed significant advancements in creating meaningful and compact graph representations.

Focus on algorithms that learn representations reflecting original graph structure.

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

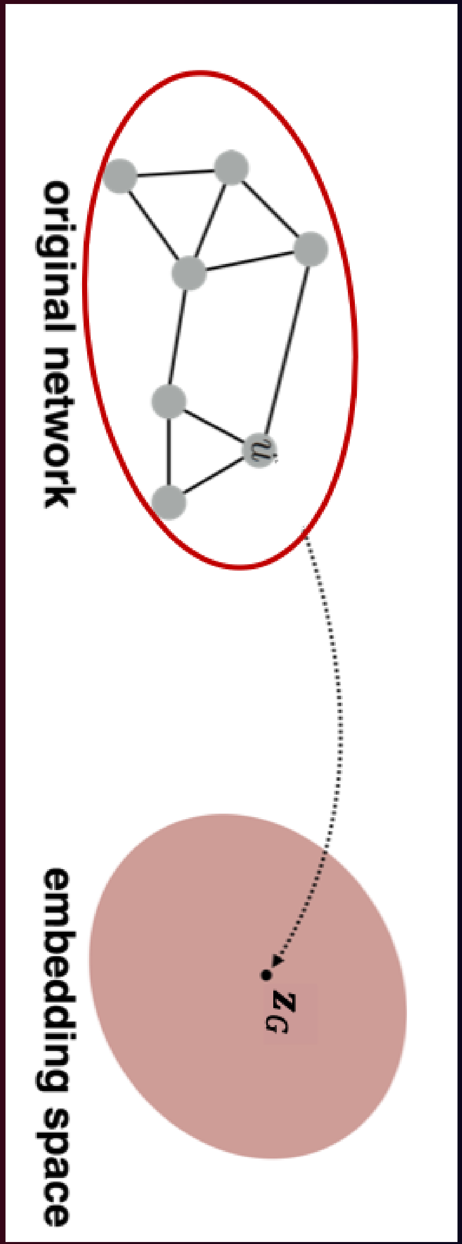
Graph Embedding – What?

Graph Embedding:

- **Definition:** Transforming the complex structure of graphs into numerical vectors.
- **Purpose:** Enable machine learning algorithms to process graph data effectively.

Applications of Graph Embedding:

- **Contextualized Representations:** Capturing rich, context-specific information of nodes, edges, or subgraphs.
- **Information Retrieval:** Enhancing search relevance and recommendation systems.
- **Network Analysis:** Facilitating tasks like community detection and anomaly detection.
- **Link Prediction:** Predicting future connections between nodes based on learned embeddings.



DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Graph Embedding – Formal Definition

Graph Embedding Mapping:

- **Objective:** Learning a mapping function $f: G(\mathrm{N}, \mathrm{E}) \rightarrow \mathbb{R}^d$ for a given graph.
- **Application:** Apply the learned mapping to the graph to obtain a feature set for machine learning.

Node Embedding:

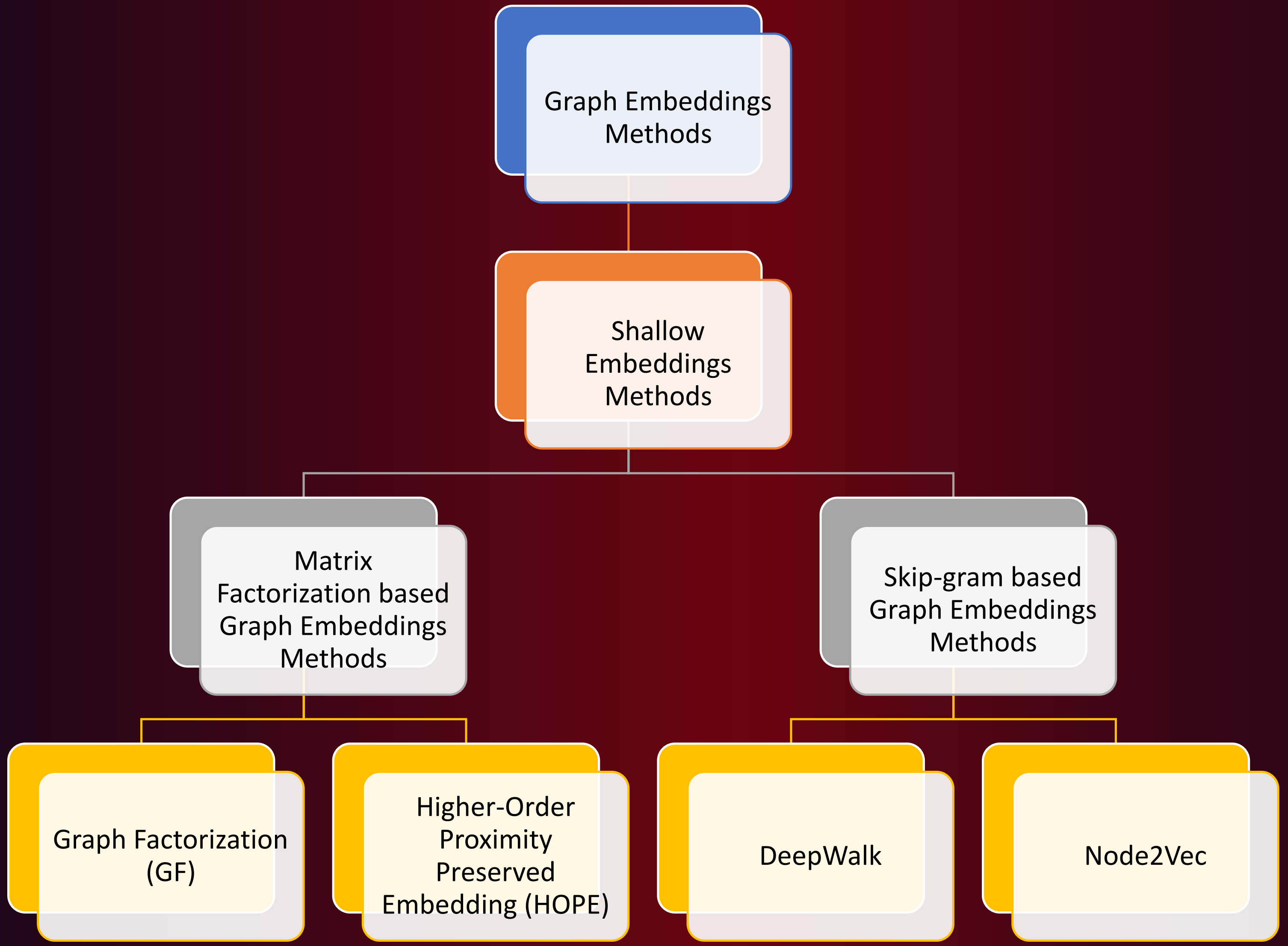
- **Node Embedding:** Learning a function $f: N \rightarrow \mathbb{R}$ for node vector representation (node embedding).

Geometric Relationships in Embedding Space:

- **Embedding** functions aim to build a vector space where geometric relationships reflect the structure of the original graph, nodes, or edges.
- **Similar Structures:** Similar structures in the original space translate to small Euclidean distances in the new space.
- **Dissimilar Structures:** Dissimilar structures result in large Euclidean distances in the new space.

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Graph Embedding - Methods



DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Matrix factorization (MF) Based Methods

MF:

A versatile decomposition technique widely employed across various domains.

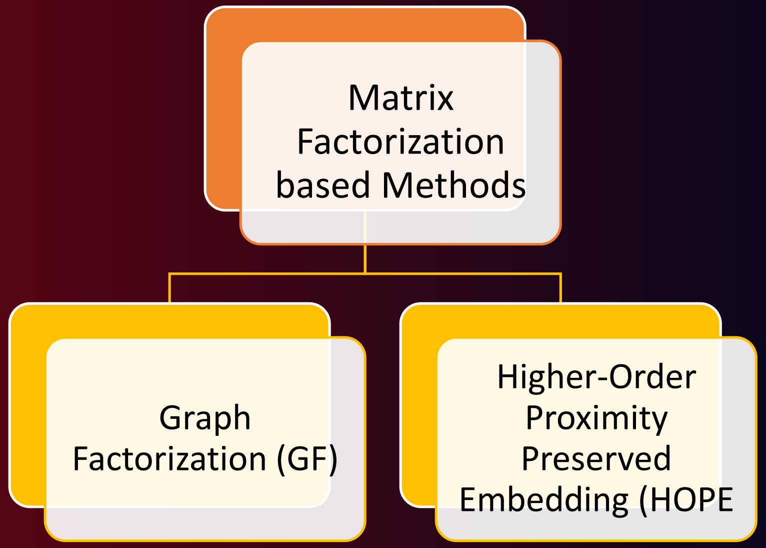
Application to Graphs:
- Adapted to graphs to reduce the high-dimensional adjacency matrix into lower-dimensional embeddings, while preserving important graph properties.

Principles:

Matrix Representation:
- In Matrix Factorization, the graph adjacency matrix is utilized as the basis for representation. The adjacency matrix is decomposed to obtain node embeddings.

Objective Function:
- Minimize the difference between the actual adjacency matrix and the product of the factorized matrices, capturing the observed interactions in the graph.

Embedding Space:
- Each node is represented as a vector (embedding) in an embedding space. MF explicitly factorizes the adjacency matrix into embedding matrices



DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

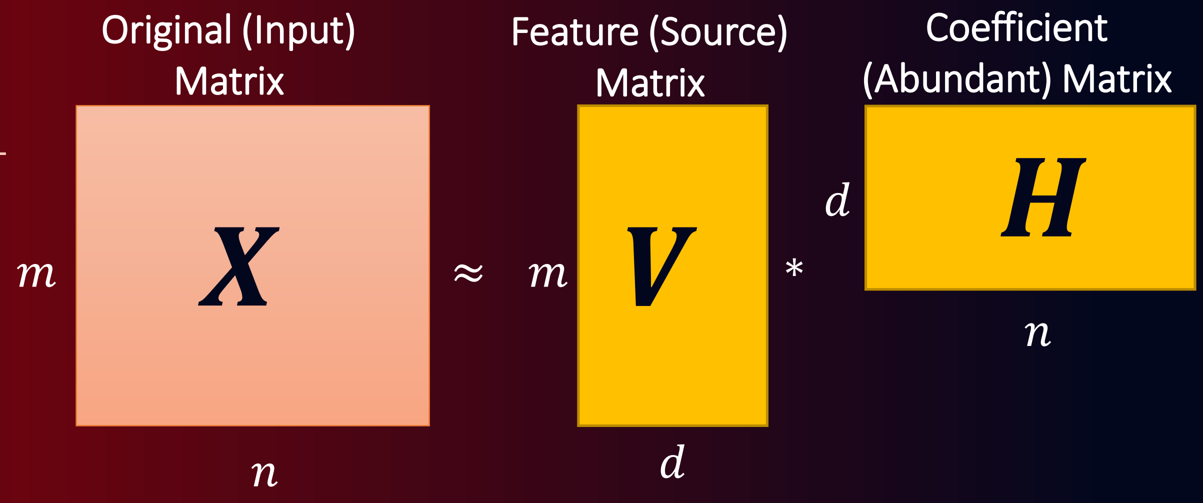
Matrix factorization - Overview

Matrix Factorization Overview:

Input Data: Let $X \in \mathbb{R}^{m*n}$ be the input data matrix.

Decomposition: MF decomposes X into <u>source</u> matrix $V \in \mathbb{R}^{m*d}$ and <u>abundance</u> matrix $H \in \mathbb{R}^{d*n}$, where d is the embedding dimensions.

Loss Function: MF learns V and H matrices by minimizing a loss function, typically the <u>Frobenius</u> <u>norm</u>: $\| \boldsymbol{X} - \boldsymbol{V} * \boldsymbol{H} \|_F^2$.



DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Graph Factorization - Overview

Formal Representation:

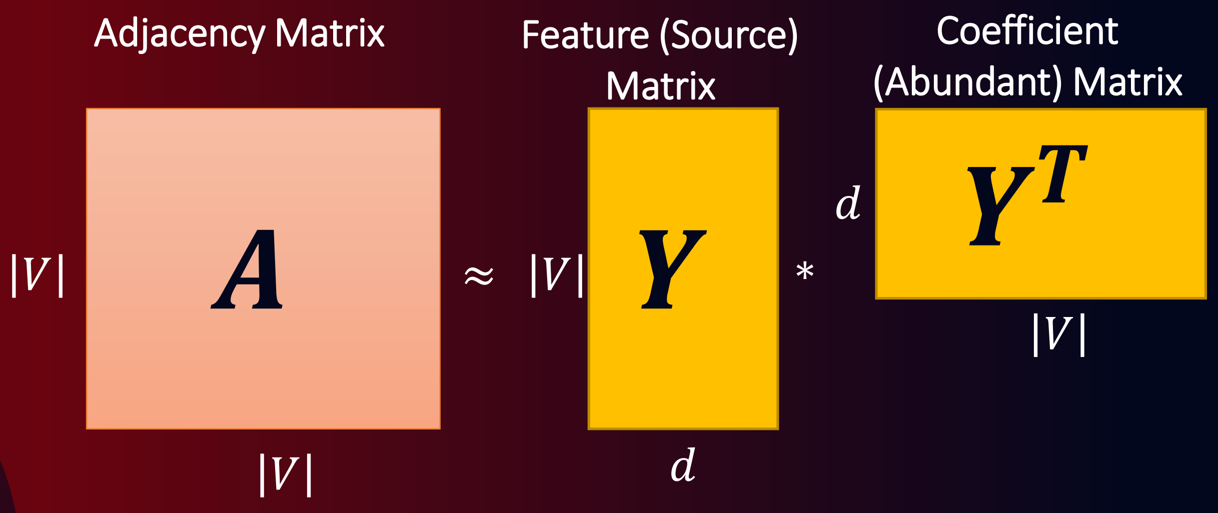
- GF algorithm achieved early success in node embedding due to computational efficiency.
- For a graph represented by $G = (V, E)$, the adjacency matrix A is utilized.

- Loss function (L) used in this matrix factorization problem to improve GF performances and scalability:

$$L = \frac{1}{2} \sum_{(i,j) \in E} (A_{i,j} - Y_{i,:} * Y_{j,:}{}^{T})^2 + \frac{\lambda}{2} \sum_{i} \|Y_{i,:}\|^2$$

Consideration of Symmetry:

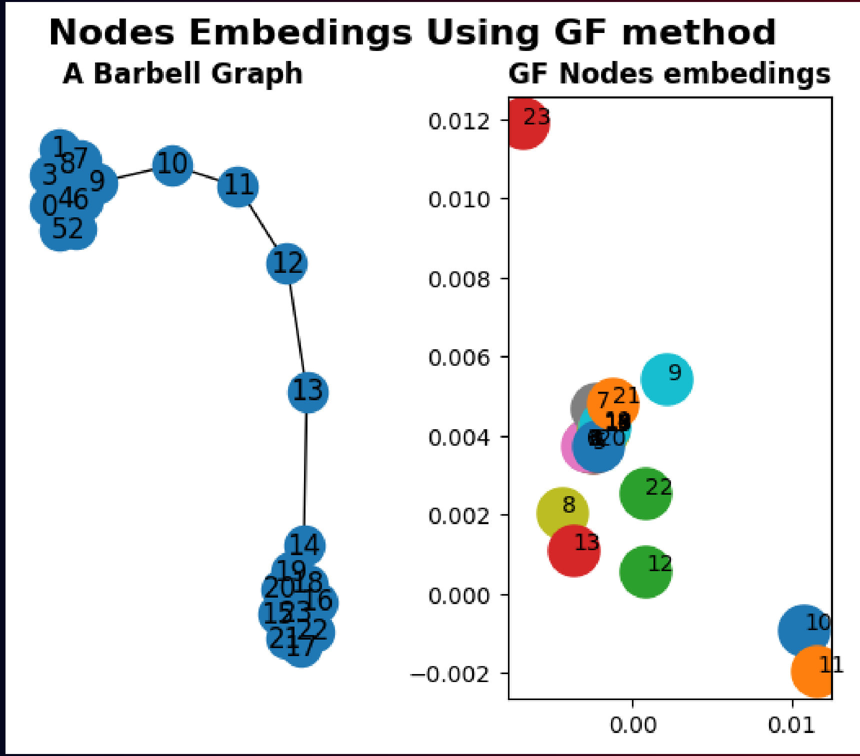
- GF performs strong symmetric factorization
- Suitable for undirected graphs with symmetric adjacency matrices.



$(i,j) \in E$	One of the edges in G
$Y \in \mathbb{R}^{\vert V \vert \times d}$	The matrix containing the d-dimensional embedding.
d	The number of dimensions of the generated embedding space.
λ	is the regularization term used to ensure the problem remains well-posed even in the absence of sufficient data.

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Graph Factorization – Pre-built Implementation Example




```
# Importing necessary libraries
import networkx as nx
from gem.embedding.gf import GraphFactorization

# Creating a barbell graph with 10 fully connected nodes on
each side and 4 connecting nodes
G = nx.barbell_graph(m1=10, m2=4)

# GraphFactorization algorithm parameters Initialization
gf = GraphFactorization(d=2, #Dim. of Embeddings
                        data_set=None, #No dataset to guide
                                       the embedding process
                        max_iter=10000,# Max training epochs
                        eta=1*10**-4, # Learning Rate
                        regu=1.0 # Regularization Strength)

# Training the algorithm to learn the node embeddings from G
gf.learn_embedding(G)

# Retrieving the computed embeddings for each node
embeddings = gf.get_embedding()
```

Using the nxt-gem library:
https://github.com/palash1992/GEM

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Higher-Order proximity Preserved Embedding (HOPE)

Introduction to HOPE Algorithm:

- **HOPE** is a method emphasizing **higher-order proximity**, essential for understanding complex network relationships.

Proximity Definitions:

- **First-Order Proximity:** Direct edge-based connections between nodes.
- **High-Order Proximity: k-step** transition between nodes.

Formal Representation:

- For a graph represented by $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, a similarity matrix $\boldsymbol{S}$ generated from graph $\boldsymbol{G}$ is used.

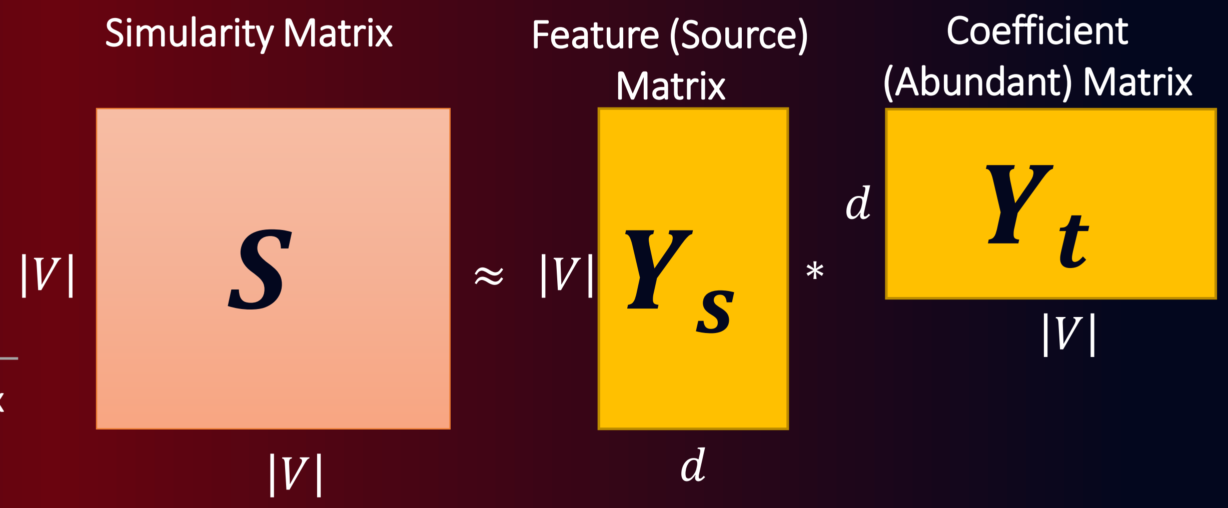
$$\boldsymbol{S} = \boldsymbol{I} * (\boldsymbol{A} * \boldsymbol{D} * \boldsymbol{A})$$

- Loss function $(\boldsymbol{L})$:

$$\boldsymbol{L} = \| \boldsymbol{S} - \boldsymbol{Y_s} * \boldsymbol{Y_t} \|_F^2$$

Consideration of Symmetry:

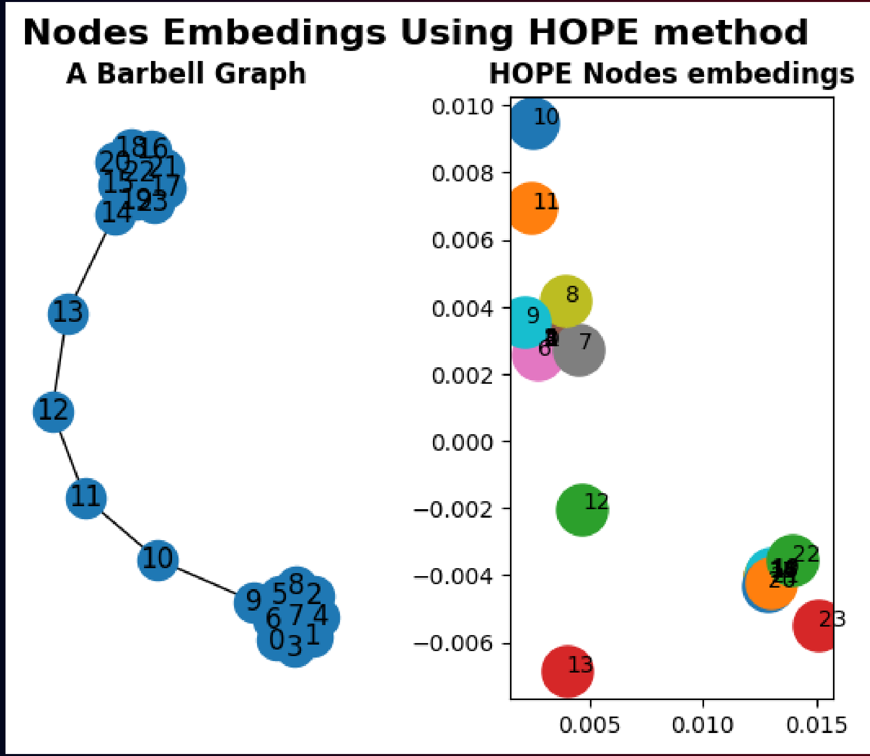
- HOPE does not force symmetric properties on embeddings.
- This characteristic is essential for effectively modeling asymmetric relationships present in directed networks.



$\boldsymbol{I}$	The identity matrix.
$\boldsymbol{A}$	Adjacency Matrix.
$\boldsymbol{D}$	Diagonal matrix computed as $D_{i,j} = 1/(A_{i,j} + A_{j,i})$

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

HOPE - Pre-built Implementation Example




```python
# Importing necessary libraries
import networkx as nx
from gem.embedding.hope import HOPE

# Creating a barbell graph (two complete graphs with m1
nodes) connected by a path of m2 nods
G = nx.barbell_graph(m1=10, m2=4)

# Initializing HOPE with desired parameters
ghope = HOPE(d=4, #Dimension of the embedding space
             beta=0.01)

# Learning the embedding for the given graph
ghope.learn_embedding(G)

# Retrieving the embeddings generated by HOPE
embeddings = ghope.get_embedding()
```

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Skip-gram Based Graph Embeddings Methods

Skip-gram:

Widely used in NLP for word embeddings. Focuses on predicting context words given a target word.

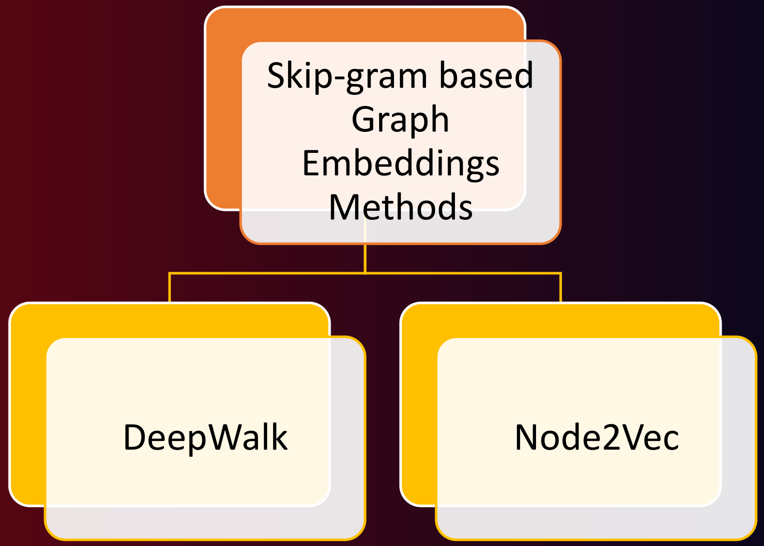
Application to Graphs:
- Adapted to graphs to capture <u>structural context of nodes</u>.
- **Example In Social Networks:** For a person (target node), their structural context could be their immediate friends or connections in the social network.

Principles:

Local Context:
- The central idea is to learn node embeddings based on their local context.

Objective Function:
- maximize the probability of observing the context nodes given the target node.

Embedding Space:
- Each node is represented as a vector (embedding) in an embedding space. The dimensions of this space are learned during the training process.



DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Skip-gram - Overview

The Skip-Gram Model:

Definition: The Skip-Gram model is a simple neural network with one hidden layer designed to predict the probability of a specific word co-occurring with other words in its context.

Training Objective: Trained to predict the presence of words given the input word.

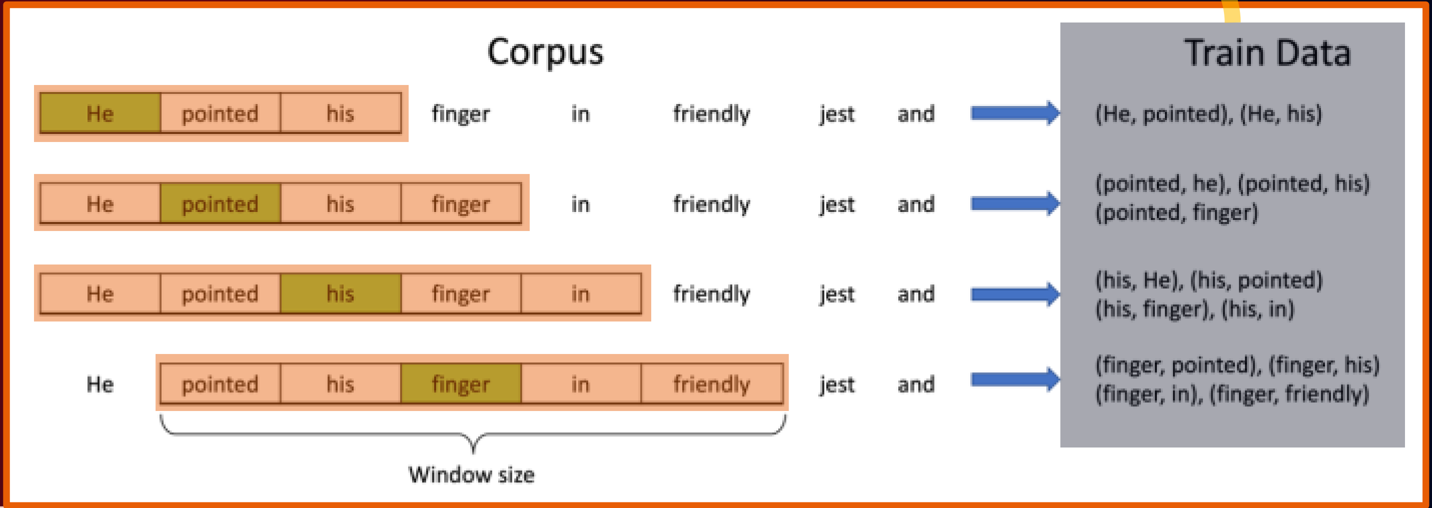
Corpus Reference: **Training data** is constructed using a text corpus.

The skip-gram model is then trained to predict the probability of a word being a context word for the given target

Generating Training Data

Building the Training Data:

- Choose a target word.
- Create a fixed-size rolling window (size w) around the target word.
- Words within the rolling window are context words.
- Form multiple (target word, context word) pairs.



DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

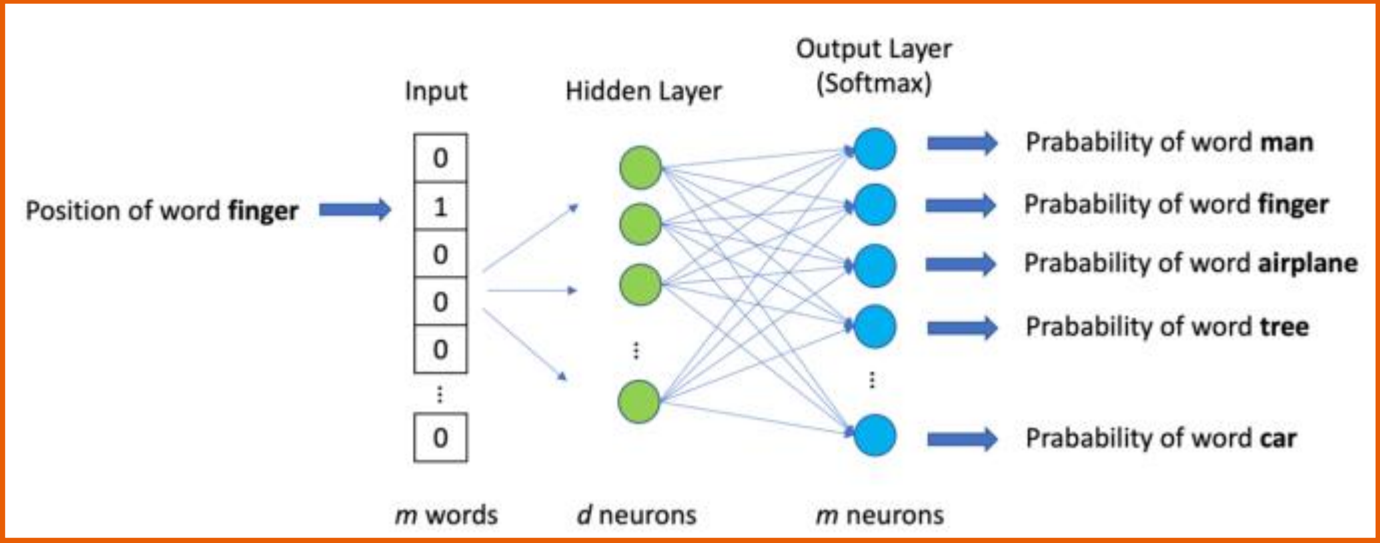
Skip-gram - Overview

Neural Network Architecture

Input Layer:
- **Binary vector** of size m representing words in the dictionary.
- **One-hot encoding:** 1 for the target word, 0 for others (context words).

Hidden Layer:
- d neurons, learning the d-dimensional embedding representation of each word.

Output Layer:
- Dense layer of m neurons (same as input vector size).
- **Softmax activation function:** Probabilities of words being "related" to the input word.



Note: The skip-gram model is widely used in Word to Vector (Word2Vec) model.

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Word2Vec - Example

```
!pip install gensim
```

```
text = """Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc eu sem scelerisque, dictum eros aliquam, accumsan quam. Pellentesque tempus, lorem ut semper fermentum, ante turpis accumsan ex, sit amet ultricies tortor erat quis nulla ........................ ..................... """.split()
```

```
# Import the Word2Vec model from gensim
from gensim.models.word2vec import Word2Vec
# Define a Word2Vec model with specified parameters
model = Word2Vec([text],  # Input text data
                 sg=1,   # Skip-gram model choice
                 vector_size=10,  # Dim. of word vectors
                 min_count=0,  # Min occur. of a word to be included in the model's vocabulary
                 window=2,  # window size: (-2)&(+2) words
                 seed=0  # Seed for reproducibility
                )
# Print the shape of the word embedding matrix
print(f'Shape of W_embed: {model.wv.vectors.shape}')
# Train the Word2Vec model
model.train([text], total_examples=model.corpus_count, epochs=10)
# Print the word vector for the word at index 0
print(f'Word embedding = {model.wv[0]}')
```

```
Word embedding = [ 0.07156403  0.03257632  0.00209916 -0.04374931 -0.03398107 -0.08656936
 -0.09047253 -0.0955243  -0.06482638  0.0660186 ]
```

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

DeepWalk - Overview

Brief Overview:

DeepWalk, introduced in **2014** by **Perozzi et al[1]**, gained widespread popularity in the graph research community.

DeepWalk effectively represents nodes in a graph as feature vectors.

Quick to implement and serves as a reliable baseline for various graph-related tasks.

Key Points:

Goal: Generate high-quality node representations in an unsupervised manner.

Architecture heavily inspired by **Word2Vec** in **NLP**, treating nodes as the dataset instead of words.

Utilizes **random walks** to create sequences of nodes resembling sentences.

[1] Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "***Deepwalk: Online learning of social representations***." Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. 2014.

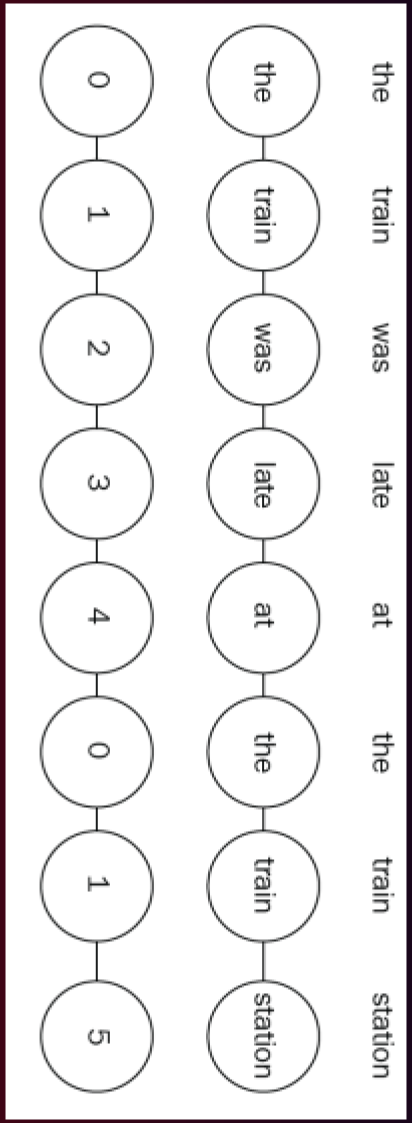


Figure – Sentences can be represented as graphs

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

DeepWalk – Random Walks Overview

Random Walks Brief Overview:

Definition: sequences of nodes generated based on some random process from its neighboring nodes.

Nodes can appear multiple times in a sequence, reflecting their proximity.

Significance: Nodes frequently appearing together in a sequence are likely to be close or similar.

Random Walks Core Concept:

High similarity scores for **close nodes; low scores** for **distant nodes** in the generated sequences.

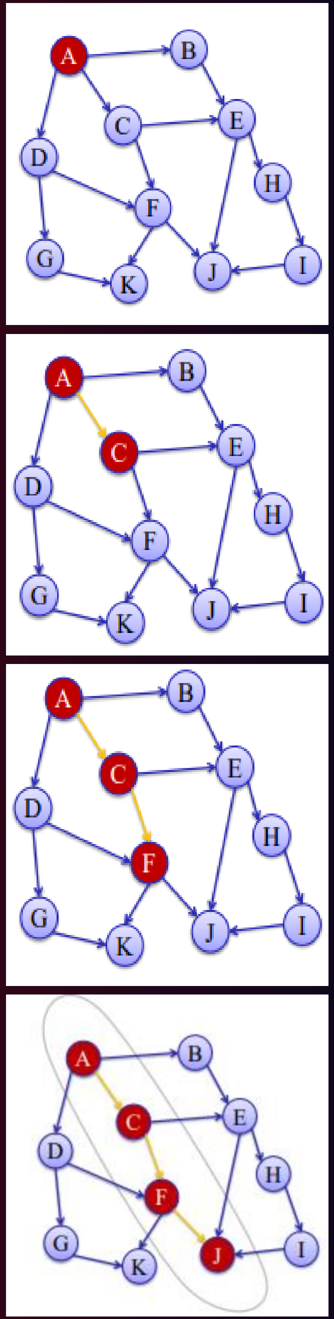
Generation of Random Walks:

Starting Point: Begin from an initial node.

Traversal: Traverse the graph.

Random Neighbor Selection: Randomly choose neighboring nodes.

Iterative Process: Repeat for a set number of steps or until a stopping criterion is reached.



DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

DeepWalk – Random Walks Implementation

```python
# Import the req. libs
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import random
random.seed(0)
```

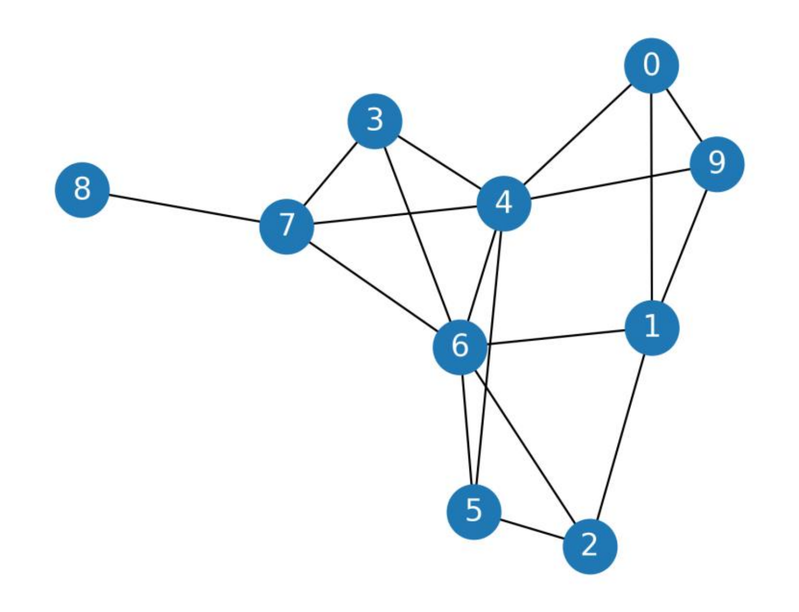
```python
plt.figure(dpi=300)
plt.axis('off')
pos= nx.spring_layout(self.graph)
nx.draw_networkx(
                 self.graph,
                 pos=pos,
                 node_size=600,
                 cmap='coolwarm',
                 font_size=14,
                 font_color='white'
                )
plt.show()
```

```python
class RandomWalk:
    # RandomWalk class initialization with a given graph
    def __init__(self, graph):
        self.graph = graph

    # Perform a random walk on the graph
    def random_walk(self, start, length):
        walk = [str(start)]  # starting node
        for i in range(length):
            neighbors = [node for node
                         in self.graph.neighbors(start)]
            next_node = np.random.choice(neighbors, 1)[0]
            walk.append(str(next_node))
            start = next_node
         return walk

     #Plot the graph using NetworkX and Matplotlib
     def plot_graph(self):
         # code for plotting
```

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

DeepWalk – Random Walks Implementation

- Generate a random graph using the **Erdos-Renyi** model.
- It has **10** nodes and a probability of **0.3** for creating edges between nodes.




```
# Example usage
G = nx.erdos_renyi_graph(10, 0.3, seed=1, directed=False)
random_walker = RandomWalk(G)
walk_result = random_walker.random_walk(start=0, length=10)
random_walker.plot_graph()
```

```
# Printing the result
print(walk_result)
```

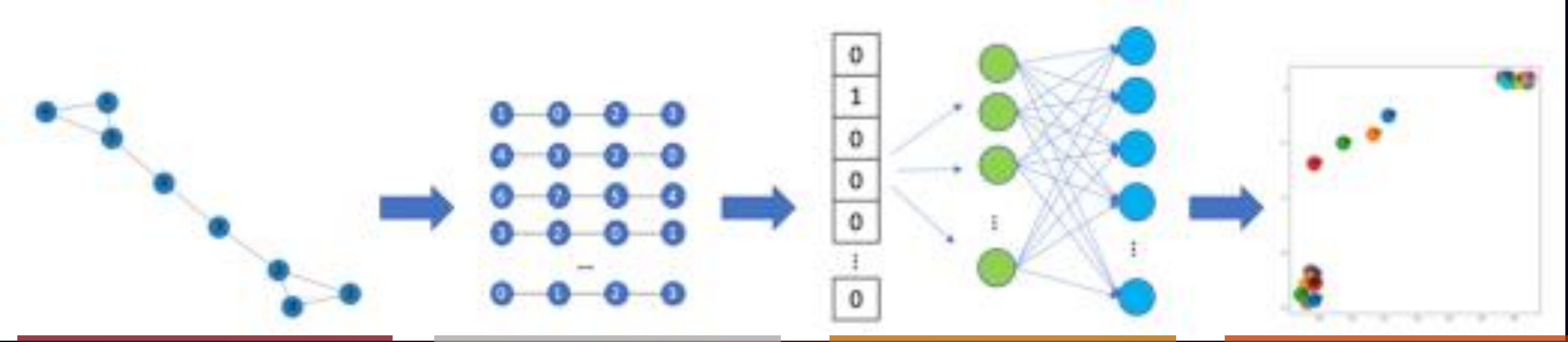
```
['0', '1', '2', '6', '4', '3', '4', '7', '6', '4', '9']
```

Insights

Node Proximity:	Nodes 4 & 6 frequently co-occur, indicating similarity (a level of proximity or closeness between them).
Homophilic Graph:	The graph is described as "**homophilic**," suggesting that nodes with similar characteristics or properties tend to be connected or appear together.
Similarity Capture:	DeepWalk aims to capture and represent node similarities. DeepWalk captures relationships like those between nodes 4 & 6

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

DeepWalk – Big Picture



Input Graph

Random Walk Generation:

- Generate **random walks** for each node in the input graph.
- Each walk has a maximum length (t).

Skip-Gram Training:

- Train a **skip-gram** model using the generated **random walks**.
- Treat the **graph** as a **text corpus** and each **node** as a **word** in this context.
- A random walk represents a sentence, and skip-gram is trained on these "sentences."

Embedding Generation:

- Use the information stored in the **skip-gram** model's hidden layers.
- Extract the **embedding** for each node based on the skip-gram training.

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

DeepWalk – Implementation

```
# Import the req. libs
import networkx as nx
import numpy as np
from gensim.models.word2vec import Word2Vec
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
```

```
    # Build vocabulary
    model.build_vocab(walks)

    # Train model
    model.train(walks,
      total_examples=model.corpus_count,
      epochs=epochs, report_delay=1)

    return model
```

```
class DeepWalk(RandomWalk):
    # DeepWalk class initialization with a given graph
    def __init__(self, graph):
        super().__init__(graph)

    # Generates random walks on the graph for DeepWalk
    def generate_random_walks(self, num_walks,
                                        walk_length):
        walks = []
        for node in self.graph.nodes:
            for _ in range(num_walks):
                walks.append(self.random_walk(node,
                                              walk_length))
        return walks

    # Trains a Word2Vec model on the provided walks.
    def train_word2vec(self, walks, vector_size=100,
                            window=10, epochs=30):
        from gensim.models.word2vec import Word2Vec

        model = Word2Vec(
            walks,
            hs=1,    # Hierarchical softmax
            sg=1,    # Skip-gram
            vector_size=vector_size,
            window=window,
            seed=1
        )
```

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

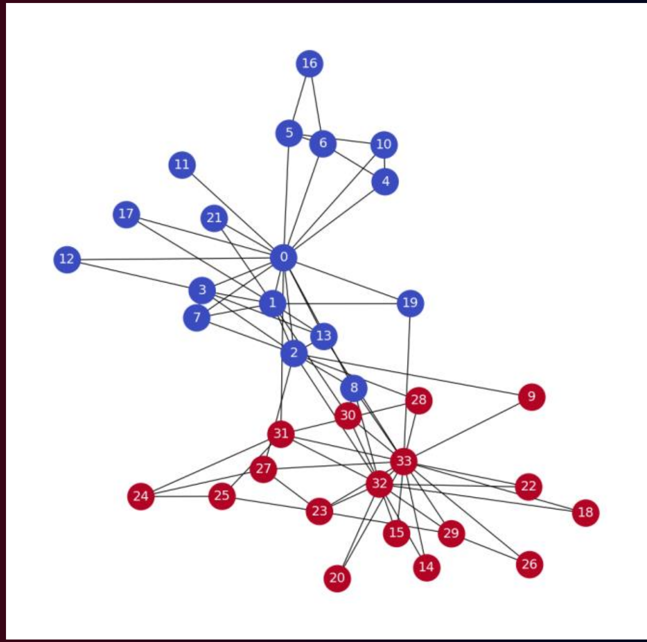
GRAPH EMBEDDING
MF BASED METHODS
SKIP-GRAM BASED METHODS
MF VS SKIP-GRAM BASED METHODS

DeepWalk –Implementation (Continued Code)

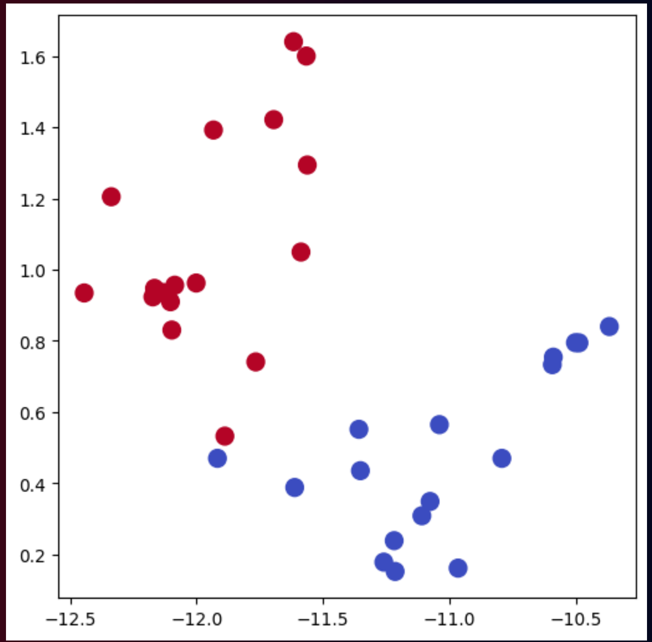
```
# Usage example
G = nx.karate_club_graph()
# Process labels (Mr. Hi = 0, Officer = 1)
labels = [1 if G.nodes[node]['club'] == 'Officer' else 0 for node
in G.nodes]

# Create DeepWalk instance
deep_walker = DeepWalk(G)

# Generate random walks
walks = deep_walker.generate_random_walks(num_walks=100,
                                          walk_length=10)

# Train Word2Vec model
word2vec_model = deep_walker.train_word2vec(walks,
vector_size=100, window=10, epochs=30)

# Visualize embeddings using t-SNE
deep_walker.visualize_tsne(word2vec_model, labels)
```

t-SNE (t-distributed Stochastic Neighbor Embedding) is a technique for visualizing high-dimensional data in a lower-dimensional space, aiming to maintain local relationships and reveal clusters of similar data points for better understanding and analysis.

karate_club_graph (Original Input Graph)



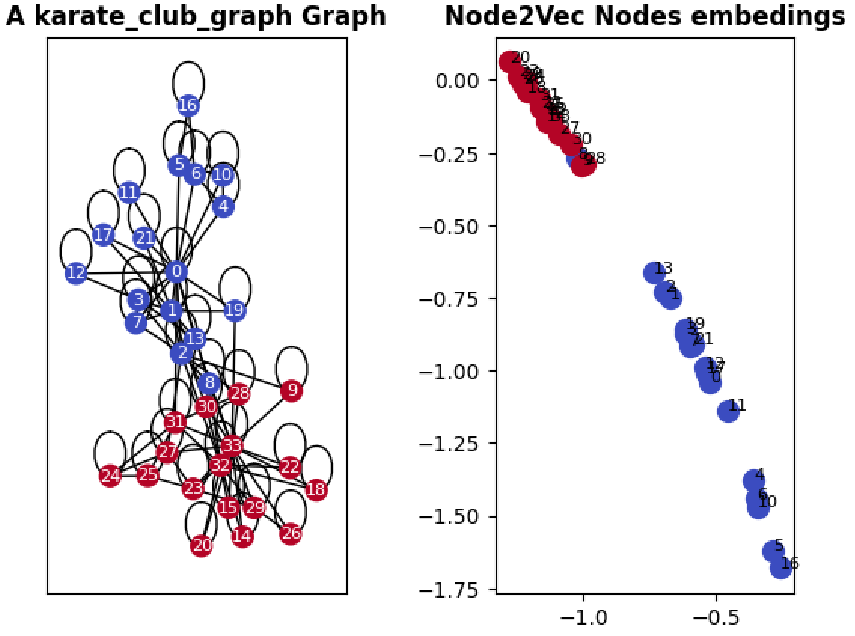
DeepWalk Embedings visualized with t-SNE



DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

DeepWalk - Pre-built Implementation Example

Nodes Embedings Using Node2Vec method




```python
# Importing necessary libraries
import networkx as nx
from karateclub.node_embedding.neighbourhood.deepwalk import
DeepWalk # DeepWalk implementation from KarateClub

# Load the karate club graph
G = nx.karate_club_graph()

# Create a DeepWalk instance with embedding dimensions 2
dw = DeepWalk(dimensions=2)

# Fit DeepWalk on the graph to generate node embeddings
dw.fit(G)

# Obtain the generated embeddings from DeepWalk
embeddings = dw.get_embedding()
```

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Node2Vec - Overview

Brief Overview:

Node2Vec, introduced in **2016** by **Grover and Leskovec [2]** from Stanford University.

Node2Vec retains main components of **DeepWalk**: **Random walks** and **Word2Vec**.

Node2Vec lead to better performance compared to **DeepWalk.**

Key Points:

Goal: Generate high-quality node representations in an unsupervised manner.

Unlike **DeepWalk's uniform distribution**, **Node2Vec** employs carefully **biased random walks**.

The **bias** in random walks is a distinct feature of **Node2Vec**

[2] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. 2016.

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Node2Vec - Introducing Biases in Random Walks

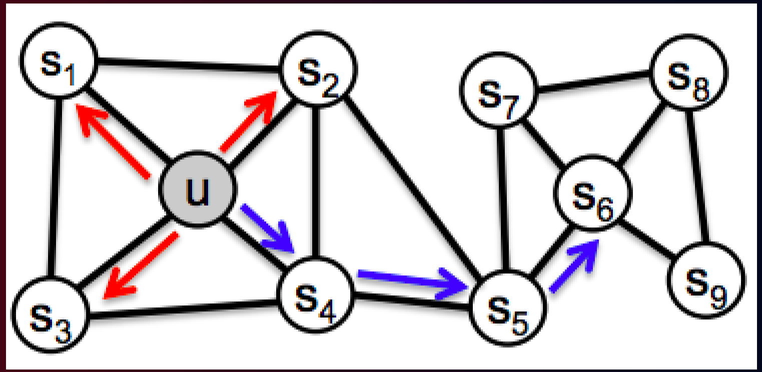
Definition: **Biases** in **Node2Vec** refer to intentional adjustments made to the **random walk** process within a graph to guide or influence the exploration of nodes.

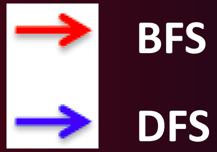
Objective: The goal is to influence the exploration patterns by adjusting parameters in random walks, promoting strategies akin to **Depth-First Search (DFS)** or **Breadth-First Search (BFS)**.

Influence on Random Walks: These biases significantly affect **the node sequences generated** during **random walks**, impacting subsequent node embeddings.

Parameter-Driven Bias: Biases are achieved by **fine-tuning parameters** to encourage specific exploration directions, shaping the random walk process.

Exploration Strategies: Biases play a crucial role in capturing targeted graph characteristics or relationships effectively.



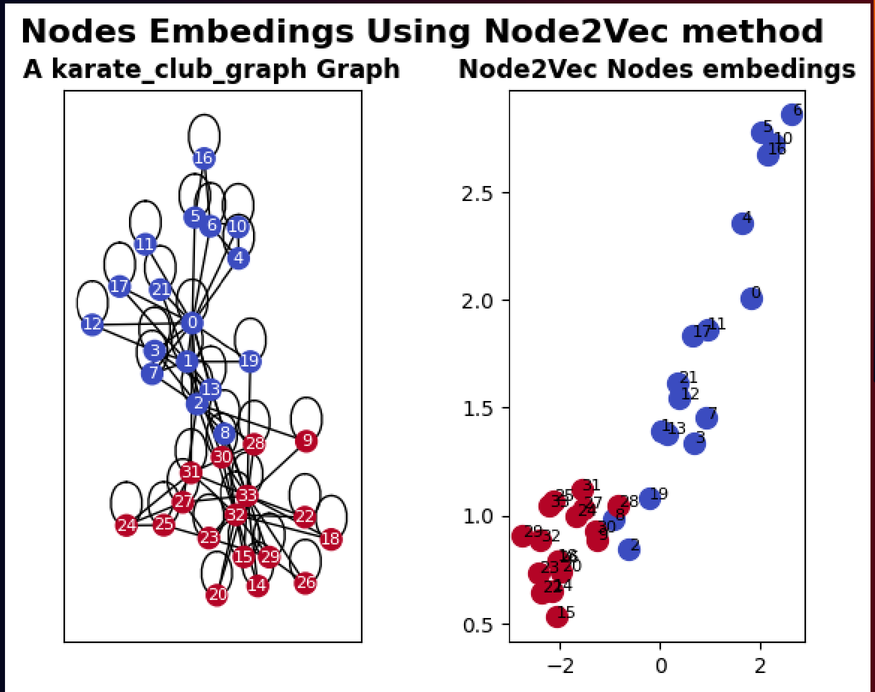


BFS → fine-tuned by parameter p

DFS → fine-tuned by parameter q

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Node2Vec - Pre-built Implementation Example




```python
# Importing necessary libraries
import networkx as nx
from node2vec import Node2Vec
import matplotlib.pyplot as plt

node2vec = Node2Vec(G, dimensions=2, p=1, q=1)
model = node2vec.fit(window=10)
```

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

Matrix Factorization vs. Skip-gram Embeddings

Matrix Factorization	Skip-gram
Factorizes adjacency matrix	Utilizes neighborhood and random walks
Linear algebraic operations	Neural network-based
Preserves structural information	Captures local and global proximity patterns
Rigid in capturing non-linearity	More flexible in capturing non-linearity
Requires setting dimensionality (d)	Requires setting dimensionality (d)
Examples: HOPE, Graph Factorization	Examples: DeepWalk, Node2Vec
Emphasizes structural preservation	Focuses on semantic and proximity relationships
Suitable for certain applications	Versatile across domains

DR. B. KHALDI
Assoc. Prof.
email: b.khaldi@esi-sba.dz

THANK YOU