Representation Learning on Graphs using Node2vec

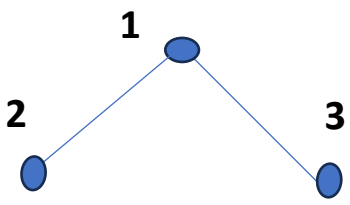
PANKAJ RAJORIA
P.RAJORIA@TU-BRAUNSCHWEIG.DE
MASTER OF DATA SCIENCE, TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG

Introduction

Graphs are a versatile and powerful data structure used to represent relationships between entities. They consist of nodes, or vertices, which represent individual units, and edges, which represent connections between nodes. A graph can be denoted as a pair **(V, E),** where **V** represents the set of vertices or nodes in the graph, and **E** represents the set of edges, indicating the connections between the vertices. Typically, each edge in E is defined as an unordered pair of vertices (u, v), signifying a relationship between nodes u and v.

Graphs serve as a fundamental model for capturing and analysing various real-world scenarios that involve entities and their interactions. In the field of machine learning, graphs are particularly useful for representing structured data. Techniques like representation learning are employed to extract and encode the structural information into low-dimensional vectors, which then serve as features for various machine learning tasks. This enables the discovery of patterns, predictions, classifications, and recommendations based on the relationships between entities.

One of the most commonly used representations of graphs is the adjacency matrix. An adjacency matrix $\mathbf{A_{ij}}$ is a square matrix that provides a compact way to represent the connections between vertices in a graph. The rows and columns of the matrix correspond to the vertices in the graph, and the entries of the matrix indicate whether there is an edge between two vertices.



1	0	**1**	**1**
2	**1**	0	0
3	**1**	0	0

Graph learning refers to the branch of machine learning that focuses on understanding and analysing structured data represented as graphs. In this context, the entities in the data are represented as nodes, and the relationships between them are represented as edges connecting the nodes. The objective of graph learning algorithms is to extract valuable insights and patterns from these interconnected data points.

Graph learning techniques find applications across various domains. For example, in social network analysis, graph learning can be utilized to classify users based on their behaviour or predict user preferences and interests. By leveraging the graph structure of social connections, graph learning algorithms can uncover hidden patterns and make accurate predictions.

In recommendation systems, graph learning plays a crucial role in link prediction. It enables the suggestion of new connections between users and items based on their past interactions, facilitating personalized recommendations. By learning from the graph structure, graph learning algorithms can effectively capture the collaborative filtering effect and enhance recommendation accuracy.

In the field of bioinformatics, graph learning is employed to analyse biological networks and predict protein functions. By representing proteins as nodes and interactions between proteins as edges, graph learning algorithms can classify proteins based on their functional properties and infer functions for unannotated proteins.

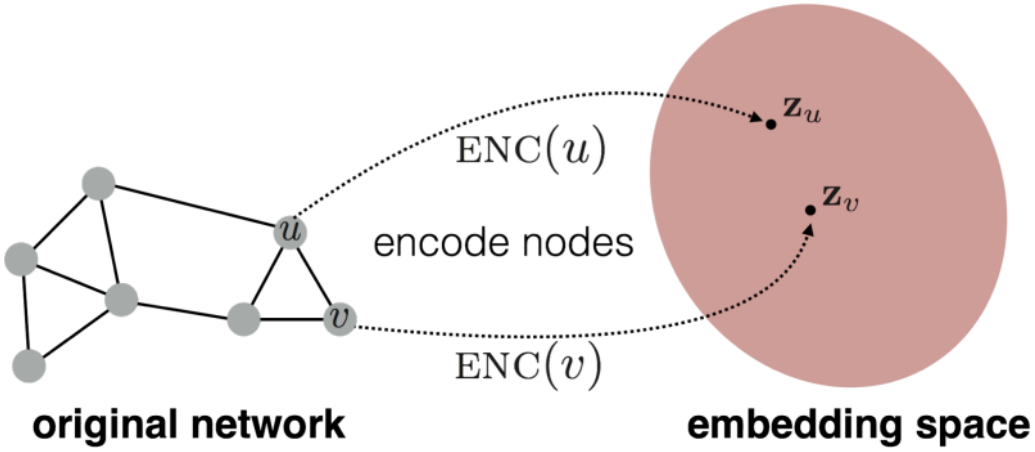
Graph learning also finds applications in knowledge graphs, where entities and their relationships are represented as a large-scale graph. By learning meaningful representations from the knowledge graph, graph learning algorithms can enable semantic search, question answering, and knowledge graph completion. These applications contribute to enhancing the understanding and utilization of vast amounts of interconnected knowledge.

Representation Learning

Representation learning on graphs, also known as graph representation learning, refers to the process of extracting meaningful and low-dimensional representations (embeddings) from graph-structured data. The goal is to encode the structural information of the graph into numerical vectors that can be used as features for various machine learning tasks.

The process of graph representation learning typically involves the following steps:

1. **Node Embedding:** Node embedding aims to map each node in the graph to a low-dimensional vector. The objective is to ensure that nodes with similar structural roles or properties in the graph are represented by similar vectors. There are various algorithms and approaches for node embedding, such as DeepWalk, node2vec, GraphSAGE, and Graph Convolutional Networks (GCNs). These methods typically use random walks, neighbourhood aggregation, or graph convolution operations to capture the structural information and generate node embeddings.



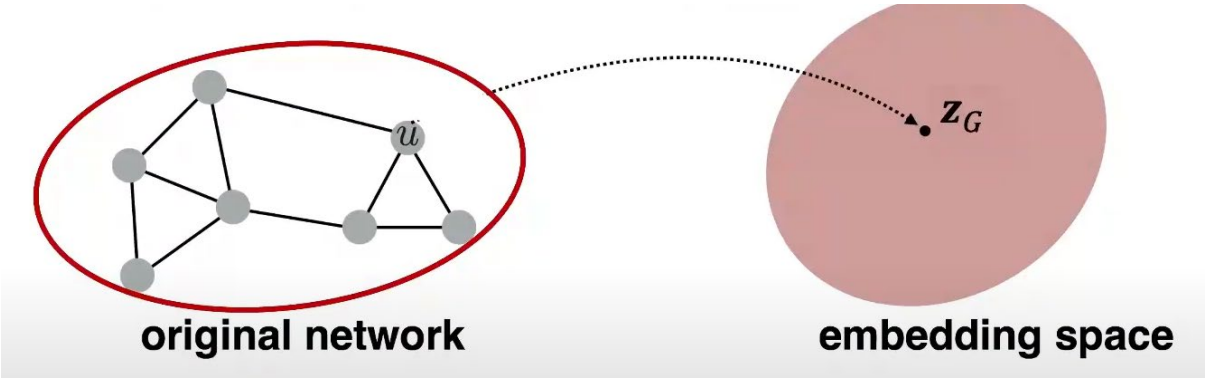
Node embedding methods typically involve the following four methodological components:

1.1. Pairwise similarity function
1.2. Encoder
1.3. Decoder
1.4. Loss function

These four methodological components work together to learn node embeddings from graph data. The pairwise similarity function captures the notion of similarity between nodes, the encoder generates low-dimensional representations, the decoder reconstructs pairwise similarities, and the loss function guides the training process by evaluating the quality of the reconstructions.

2. **Graph Embedding:** In addition to node embedding, some applications require capturing the overall graph-level information. Graph-level embedding aims to summarize the entire graph or subgraphs into a fixed-length vector representation. This can be useful for tasks such as graph classification or clustering. Techniques like Graph Isomorphism Networks

(GIN) and Graph Attention Networks (GAT) can be used to aggregate node embeddings and generate graph-level embeddings.



Random Walks

Random walks are used to explore and capture information from graph or network structures. Random walks involve taking sequences of steps on a graph, with each step determined by randomly selecting a neighbouring node. The goal is to generate sequences that capture the local neighbourhood structure of the graph. Starting with a graph representation, a random walk begins at a randomly chosen node. At each step, the walker moves to a neighbouring node based on a predefined probability distribution. By performing multiple steps, a sequence of visited nodes is generated, reflecting the graph's connectivity patterns. These sequences are then used to learn representations of the nodes or the entire graph, employing techniques such as Skip-gram, DeepWalk, or node2vec. The learned representations enable the capture of important structural information and similarities between nodes in a low-dimensional vector space, facilitating downstream tasks like node classification, link prediction, or community detection. Random walks in representation learning provide a means to explore and encode the local structure of graphs, enabling meaningful analysis and machine learning on graph data.

- Given $G = (V, E)$,
- Our goal is to learn a mapping $f: u \rightarrow \mathbb{R}^d$:
  $f(u) = \mathbf{z}_u$

- 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 strategy R

- Given node u, we want to learn feature representations that are predictive of the nodes in its random walk neighborhood $N_R(u)$.

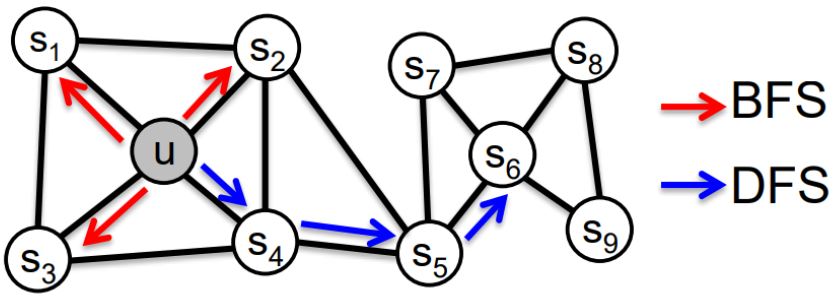
---

1. Run **short fixed-length** random walks starting from each node on the graph
2. For each node u collect $N_R(u)$, the multiset of nodes visited on random walks starting from u.
3. Optimize embeddings using Stochastic Gradient Descent:

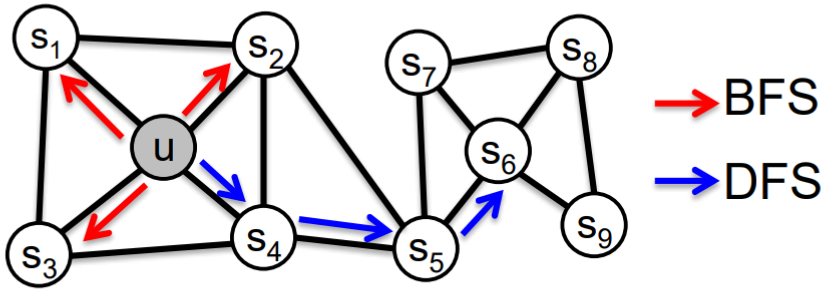
$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

Node2Vec

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 $N_R(u)$ of a given node u:



Walk of length 3 ($N_R(u)$ of size 3):

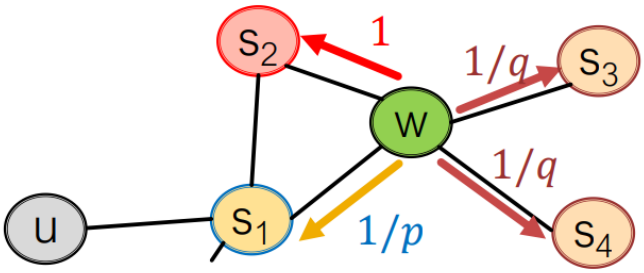
$N_{BFS}(u) = \{s_1, s_2, s_3\}$ Local microscopic view

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

Biased fixed-length random walk R that given a node u generates neighborhood $N_R(u)$

- Two parameters:
  - **Return parameter p:**
    - Return back to the previous node
  - **In-out parameter q:**
    - Moving outwards (DFS) vs. inwards (BFS)
    - Intuitively, q is the "ratio" of BFS vs. DFS

- **Walker came over edge (s_1, w) and is at w. Where to go next?**



$1/p, 1/q, 1$ are unnormalized probabilities

- p, q model transition probabilities
  - p … return parameter
  - q … "walk away" parameter