# Algorithms and Data Structures with Applications in Machine Learning

Graph Representation Learning

December 31, 2024

## Outline

## Outline

## Introduction to Graphs

Definition

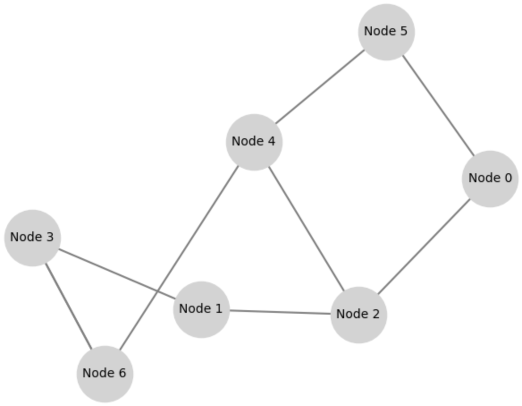
A graph is defined as:

$$G = (V, E, u)$$

- **Nodes (Vertices):** The set $V$ represents the nodes in the graph.
- **Edges:** The set $E \subseteq V \times V$ represents the connections (relationships) between the nodes.
- **Features:** Each node can have a feature vector $u(v)$ representing its attributes.
- **Labels:** Nodes (or edges) can also have labels, which are used for tasks like classification.

## Example Graph

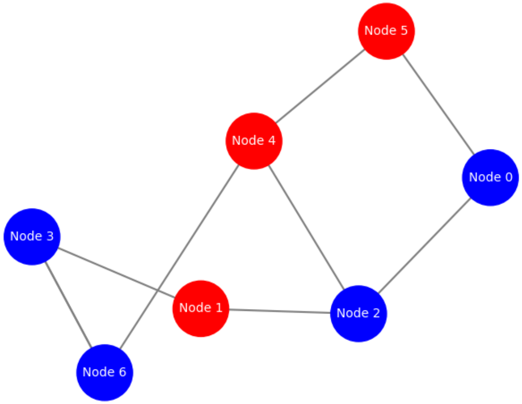
**Example:** The graph below has 7 connected nodes ($V = \{0, 1, 2, 3, 4, 5, 6\}$) and their edges ($E$).



## Example Graph: Node Labels

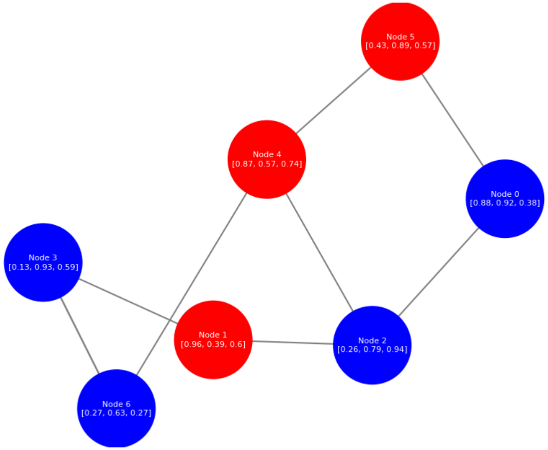
**Example:** Nodes in a graph can be associated with labels.

**Blue nodes**: Label 0  **Red nodes**: Label 1



## Example Graph: Node Features

**Example:** Each node in the graph can have associated features. In this case: Each node has a feature vector of dimension 3.



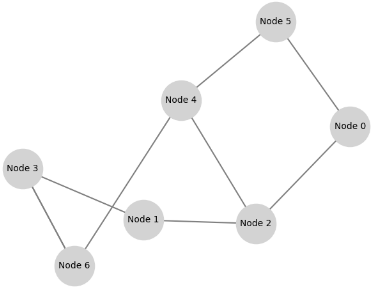
## Adjacency Matrix

**Definition**

The adjacency matrix $A$ of a graph $G = (V, E)$ is a matrix of size $|V| \times |V|$, where:

- $A[i][j] = 1$ if there is an edge between node $i$ and node $j$.
- $A[i][j] = 0$ if there is no edge between node $i$ and node $j$.

**Example:** A graph and its corresponding adjacency matrix:



**Adjacency Matrix:**

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

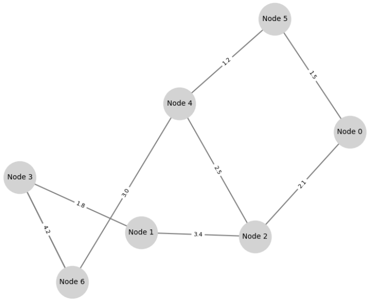
## Weighted Adjacency Matrix

**Definition**

The adjacency matrix $A$ can be extended to a weighted matrix $W$, where:

- $W[i][j]$ represents the weight of the edge between node $i$ and node $j$.

**Example:** A graph and its a weighted adjacency matrix:



**Weighted Matrix:**

$$W = \begin{bmatrix} 0 & 0 & 2.1 & 0 & 0 & 1.5 & 0 \\ 0 & 0 & 3.4 & 1.8 & 0 & 0 & 0 \\ 2.1 & 3.4 & 0 & 0 & 2.5 & 0 & 0 \\ 0 & 1.8 & 0 & 0 & 0 & 0 & 4.2 \\ 0 & 0 & 2.5 & 0 & 0 & 1.2 & 3.0 \\ 1.5 & 0 & 0 & 0 & 1.2 & 0 & 0 \\ 0 & 0 & 0 & 4.2 & 3.0 & 0 & 0 \end{bmatrix}$$

## Applications of Machine Learning on Graphs

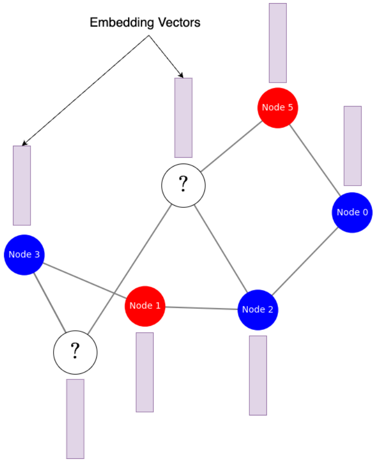
**Applications:** Machine Learning on graphs enables a variety of tasks, including:

- **Node Prediction:** Predict properties or labels of nodes in a graph (e.g., user classification in social networks).
- **Link Prediction:** Predict the existence or strength of a connection between two nodes (e.g., recommendation systems).
- **Graph Classification:** Assign labels to entire graphs (e.g., chemical compound classification).
- **Clustering:** Group nodes into communities or clusters based on their properties or structure.

## Objective: Node Classification

**Objective:** The objective of this course is two-fold:

1. **Learning a *D*-dimensional representation:** Create embedding vectors for nodes that capture the structure of the graph.
2. **Node Classification:** Use the learned embeddings to predict the labels of the nodes.



## Outline

## Graph Structure-Based Embeddings: Introduction

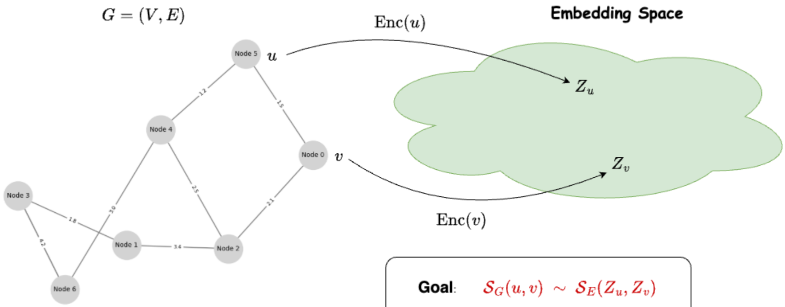
**Objective:** We aim to learn a mapping:

$$f : V \to \mathbb{R}^D$$

where each node $u \in V$ is mapped to a $D$-dimensional vector $\mathbf{Z}_u \in \mathbb{R}^D$.

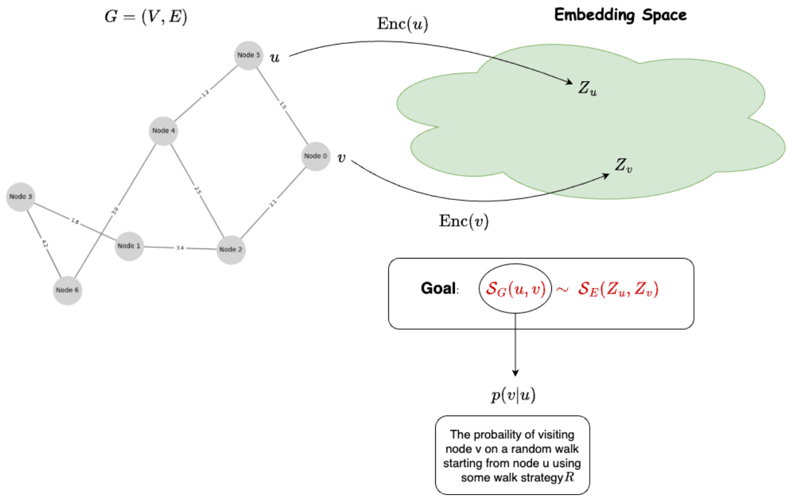
- In this section, we focus on leveraging the graph's **structure** to generate embedding vectors for nodes.
- The embeddings can be used for downstream tasks, such as node classification or link prediction.
- **No use of feature vectors:** We only use the graph topology (connections between nodes) to derive the embeddings.

# Graph Structure-Based Embeddings: Objective



**Goal**: $\mathcal{S}_G(u, v) \sim \mathcal{S}_E(Z_u, Z_v)$

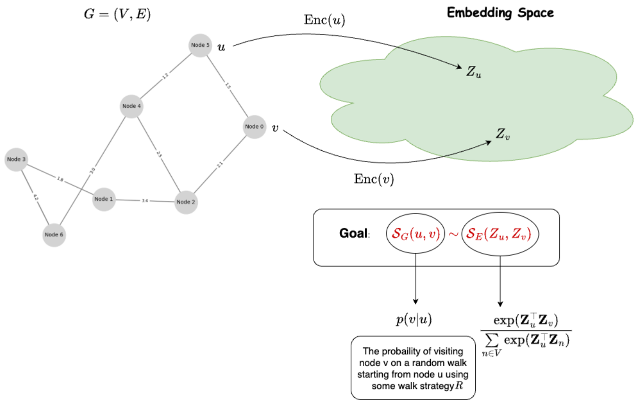
# Graph Structure-Based Embeddings: Objective



**Goal**: $\mathcal{S}_G(u, v) \sim \mathcal{S}_E(Z_u, Z_v)$

The probaility of visiting node v on a random walk starting from node u using some walk strategy $R$

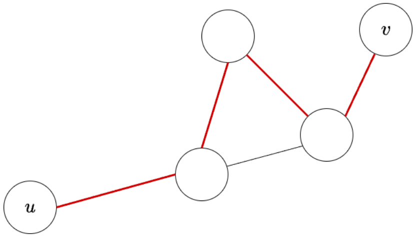
## Graph Structure-Based Embeddings: Objective


$G = (V, E)$
Node 0
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
$u$
$v$
$\mathrm{Enc}(u)$
$\mathrm{Enc}(v)$
Embedding Space
$Z_u$
$Z_v$
Goal: $\mathcal{S}_G(u, v) \sim \mathcal{S}_E(Z_u, Z_v)$
$p(v|u)$
$\dfrac{\exp(\mathbf{Z}_u^\top \mathbf{Z}_v)}{\sum_{n \in V} \exp(\mathbf{Z}_u^\top \mathbf{Z}_n)}$
The probaility of visiting node v on a random walk starting from node u using some walk strategy $R$

## Deep Walk algorithm

**Random Walks:**

- A random walk is a sequence of steps through the graph, starting from a given node $u$, where each step randomly selects a neighboring node.
- The nodes visited during these walks represent the local neighborhood structure around $u$, denoted $\mathcal{N}_R(u)$
- Here is an example of a random walk from node $u$ to node $v$.



## Determining Neighbors Using Random Walks

**Algorithm** Fixed-Length Random Walks

```
Require: Graph $G = (V, E)$, starting node $u$, walk length $L$, number of walks $N$
Ensure: $\mathcal{N}_R(u)$ Multiset of nodes visited during random walks starting from $u$
 1: Initialize an empty multiset of neighbors: neighbors $\leftarrow$ []
 2: for $n = 1$ to $N$ do                                  ▷ Perform $N$ random walks
 3:     Initialize current_node $\leftarrow u$
 4:     for $l = 1$ to $L$ do                                ▷ Walk for $L$ steps
 5:         Sample a random neighbor $v \in$ Neighbors(current_node)
 6:         neighbors.append($v$)
 7:         current_node $\leftarrow v$
 8:     end for
 9: end for
10: return neighbors
```

## Introducing Node2Vec: Biased Random Walks

- The Node2Vec algorithm modifies traditional random walks by introducing **biases** that control how the walk explores the graph.
- This bias allows us to interpolate between two extremes:
  1. **Local Behavior:** Tendency to return to previously visited nodes, capturing local neighborhood structures. This is controlled by the **return hyperparameter** $p$.
  2. **Global Behavior:** Tendency to explore new, distant nodes, capturing the global structure of the graph. This is controlled by the **in-out hyperparameter** $q$.
- By adjusting $p$ and $q$, Node2Vec generates embeddings that can reflect different graph traversal strategies.
- This flexibility makes Node2Vec suitable for capturing diverse graph structures. (See Programming Session 6).

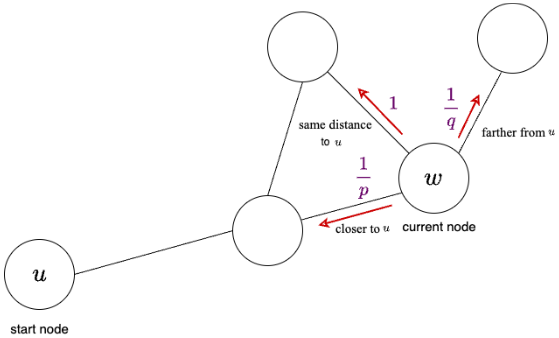
## Introducing Node2Vec: Biased Random Walks

- When the walk moves from node $u$ to $w$, the neighbors of $w$ are categorized based on their distance to $u$.
- We define the following **unnormalized probabilities**:
    1. Nodes closer to $u$ than $w$ receive an unnormalized probability of $\frac{1}{p}$.
    2. Nodes farther from $u$ than $w$ receive an unnormalized probability of $\frac{1}{q}$.
    3. Nodes at the same distance as $w$ from $u$ receive an unnormalized probability of 1.
- These unnormalized probabilities are normalized to form a valid probability distribution, which guides the biased random walk.

## Introducing Node2Vec: Biased Random Walks

Here is an example of assigning the unnormalized probabilities:

- Starting at node $u$, the walk reaches node $w$.
- The probabilities assigned to $w$'s neighbors depend on their distance to $u$, as described in the previous slide.



## Determining Neighbors Using Biased Random Walks

**Algorithm** Biased Random Walks

```
Require: Graph $G = (V, E)$, starting node $u$, walk length $L$, number of walks $N$, return parameter $p$, in-out parameter $q$
Ensure: $\mathcal{N}_R(u)$: Multiset of nodes visited during biased random walks starting from $u$
 1: Initialize an empty multiset of neighbors: neighbors $\leftarrow$ []
 2: for $n = 1$ to $N$ do                          ▷ Perform $N$ biased random walks
 3:     Initialize current_node $\leftarrow u$ and prev_node $\leftarrow$ None
 4:     for $l = 1$ to $L$ do                        ▷ Walk for $L$ steps
 5:         Compute probabilities using prev_node and current_node
 6:         Sample the next node $v$ based on the these probabilities
 7:         neighbors.append($v$)
 8:         Update prev_node and current_node
 9:     end for
10: end for
11: return neighbors
```

## Training the Embedding Vectors

**Defining the Loss Function:**

- Now that we know how to define $\mathcal{N}_R(u)$, we can derive the loss function to train the embeddings.
- The objective is to minimize the following loss function:

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

Where:

- $\mathbf{Z}_i \in \mathbb{R}^D$ is the embedding vectors for nodes $i \in V$.
- $\theta = \{\mathbf{Z}_i \mid i \in V\}$ represents all the embedding parameters to be learned.

## Outline

## Outline

Thank you for your attention