# Contents

# 1. Graph neural networks

## 1.1. Graph definition

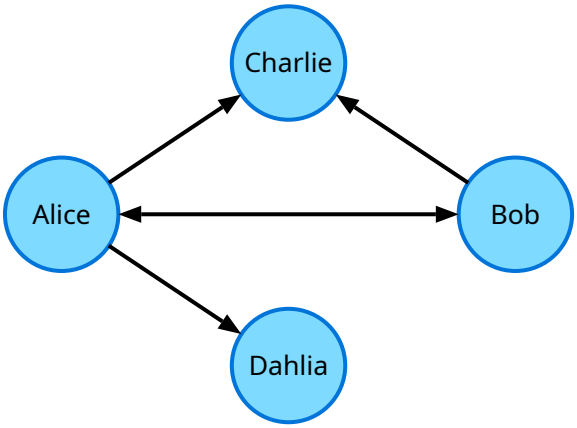
A **graph** $G$ is defined as an ordered couple $(V, E)$, where $V$ is the set of **vertices** or **nodes** and $E$ is the set of **edges**. Each edge represents the existence of a relationship betweeen two vertices. An edge $e \in E$ is therefore defined as an ordered couple $(u, v) \in V \times V$.

A graph is graphically represented using circles as nodes and using arrows as edges, whose tip is oriented in the direction of the edge. If the relationship holds both ways for a certain pair of nodes, the arrow is double-tipped.

A graph is matematically encoded in an **adjacency matrix**, a matrix that contains information concerning the existence of its edges. Formally, for a graph $G = (V, E)$ it is possible to construct an adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$ such that each entry $(i, j)$ has the value 1 if $(i, j) \in E$ and 0 otherwise. Of course, the adjacency matrix of a graph can be constructed only if its edges can be enumerated.

**Exercise 1.1.1**: Alice knows Bob, Charlie and Dahlia, whereas Bob knows Alice and Charlie. Represent the relationship with a graph.

*Solution*:




| | Alice | Bob | Charlie | Dahlia |
|---|---|---|---|---|
| Alice | 0 | 1 | 1 | 1 |
| Bob | 1 | 0 | 1 | 0 |
| Charlie | 0 | 0 | 0 | 0 |
| Dahlia | 0 | 0 | 0 | 0 |

□

The **neighborhood** of a node is the set of all nodes that connected to that node. Given a graph $G = (V, E)$, for any node $v \in V$ the neighborhood $N(v) \subseteq V$ is defined as the set $\{u \mid (v, u) \in E\}$ or, equivalently with respect to its adjacency matrix $A$, $\{u \mid A[v, u] \neq 0\}$.

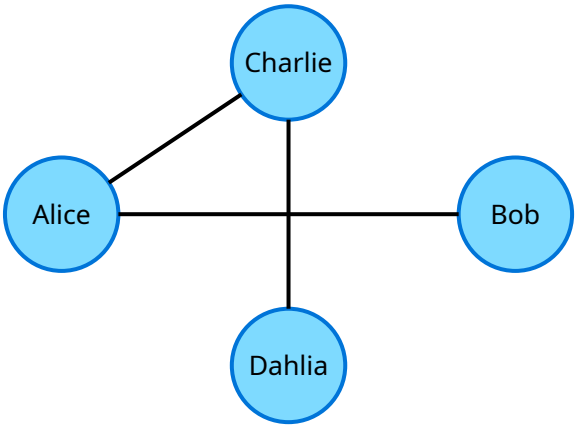
A graph is said to be **connected** if every node appears in at least one edge, except for **loops** (an edge connecting a node to itself). Formally, a graph $G = (V, E)$ is connected if, for any $v \in V$, exists $u \in V - \{v\}$ such that $(u, v) \in E$ or $(v, u) \in E$.

A graph is said to be **undirected** if the relationship between the nodes is symmetric, and holds both ways for every node. Formally, a graph $G = (V, E)$ is undirected if, for any $(u, v) \in E$, it is also true that $(v, u) \in E$. For clarity, the edges of an undirected graph are often drawn tipless. The adjacency matrix of an undirected graph will clearly be symmetric. If a graph is not undirected, it is said to be **directed**.

If a graph is connected, undirected and has no loops, it is called **simple**. It is easy to see that the adjacency matrix of a simple graph has 0 as each element of the diagonal.

**Exercise 1.1.2**: Alice is a friend of Bob and Charlie, whereas Dahlia is a friend of Charlie. Represent the relationship with a graph; is the graph simple?

*Solution*: Yes, the graph would be simple. This is because the "is a friend of" relationship is (assumed to be) symmetric, non reflexive and every person appears at least once as friend of someone else.



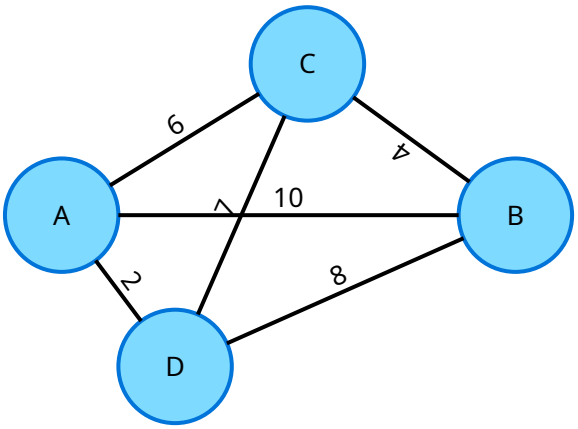

$$\begin{bmatrix} & \text{Alice} & \text{Bob} & \text{Charlie} & \text{Dahlia} \\ \text{Alice} & 0 & 1 & 1 & 0 \\ \text{Bob} & 1 & 0 & 0 & 0 \\ \text{Charlie} & 1 & 0 & 0 & 1 \\ \text{Dahlia} & 0 & 0 & 1 & 0 \end{bmatrix}$$

□

A graph $G = (V, E, W)$, that has a function $W : E \to \mathbb{R} - \{0\}$ that associates a real non-zero number to any edge of the graph is called **weighted graph**. The adjacency matrix of a weighted graph has $W(i, j)$ instead of 1 as the value of the $(i, j)$-th cell. The graphical representation of a weighted graph has the weight of each edge nenoted on the size of the corresponding arrow and, in general, the lenght of the arrow is scaled with respect to the weight.

**Exercise 1.1.3**: The town of $A$ dists 10 from $B$, 6 from $C$ and 2 from $D$, $B$ dists 4 from $C$ and 8 from $D$ and $D$ dists 7 from $C$. Represent the relationship with a graph.

*Solution*:




$$\begin{bmatrix} & \text{A} & \text{B} & \text{C} & \text{D} \\ \text{A} & 0 & 10 & 6 & 2 \\ \text{B} & 10 & 0 & 4 & 8 \\ \text{C} & 6 & 4 & 0 & 7 \\ \text{D} & 2 & 8 & 7 & 0 \end{bmatrix}$$

□

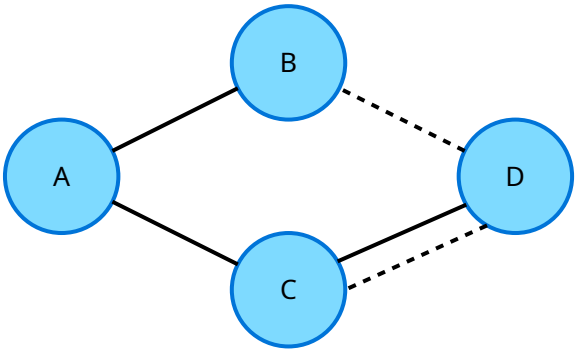
A graph can represent more than one relationship with the same entities at once; a graph with this characteristic is called a **multirelational graph**. Formally, a graph $G = (V, E, T)$ is a multirelational graph if each edge $e$ is defined as a tuple $e = (u, v, \tau)$, where $u, v \in V$ and $\tau \in T$ is the type of the relationship. A multirelational graph is encoded as $|T|$ adjacency matrices, each representing one type of relation. Said matrices can be combined into a single tensor, an **adjacency tensor** $A \in \mathbb{R}^{|V| \times |V| \times |T|}$.

**Exercise 1.1.4**: Drugs can interact with each other, inducing certain side effects when taken together. Suppose that:

- Drug $A$ induces headache when taken with drug $B$ or with drug $C$, and vice versa;
- Drug $B$ induces tachychardia when taken with drug $D$, and vice versa;
- Drug $C$ induces both tachychardia and headache when taken with drug $D$ and vice versa.

Represent this relationship in a multirelational graph.

*Solution*:




$$\begin{bmatrix} & \text{A} & \text{B} & \text{C} & \text{D} \\ \text{A} & 0 & 1 & 1 & 0 \\ \text{B} & 1 & 0 & 0 & 0 \\ \text{C} & 1 & 0 & 0 & 1 \\ \text{D} & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} & \text{A} & \text{B} & \text{C} & \text{D} \\ \text{A} & 0 & 0 & 0 & 0 \\ \text{B} & 0 & 0 & 0 & 1 \\ \text{C} & 0 & 0 & 0 & 1 \\ \text{D} & 0 & 1 & 1 & 0 \end{bmatrix}$$

□

An **eterogeneous graph** is a multirelational graph where nodes have a type, partitioning the set of nodes. Formally, a (multirelational) graph $G = (V, E, T)$ is an eterogeneous graph where the set $V$ is constructed as the union of $n$ sets $V_i$, with $i \in \{1, ..., n\}$, such that:

$$V = \bigcup_i V_i \qquad\qquad V_i \cap V_j = \emptyset, \forall i, j \text{ when } i \neq j$$

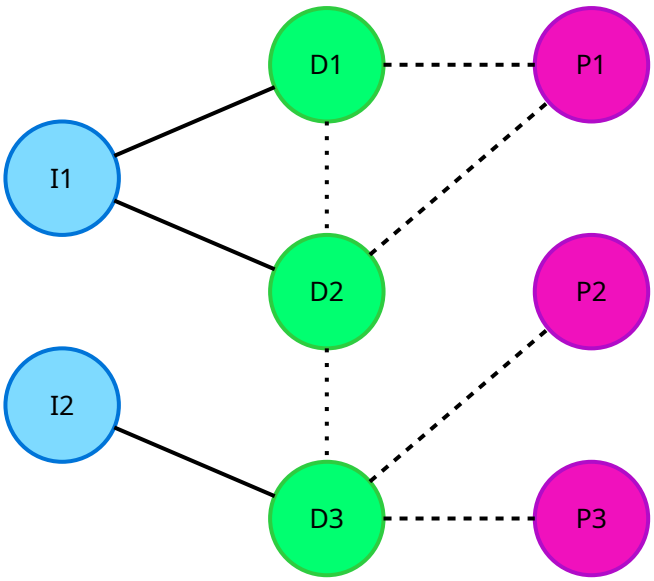
An eterogeneous graph whose edges can only connect nodes of different types is called a **multipartite graph**. Formally, an eterogeneous graph $G = (V, E, T)$ is a multipartite graph if, for any edge $e = (u, v, \tau) \in E$, it follows that $u \in V_i, v \in V_j, i \neq j$.

**Exercise 1.1.5**: Certain illnesses are treated using certain drugs. Drugs can be incompatible with each other and influence the behaviour of certain proteins. Suppose that:

- Illness $I_1$ is treated using either drug $D_1$ or drug $D_2$;
- Illness $I_2$ is treated using drug $D_3$;
- Drug $D_1$ is incompatible with drug $D_2$ (and vice versa);
- Drug $D_2$ is incompatible with drug $D_3$ (and vice versa);
- Protein $P_1$ is influenced by drug $D_1$ and drug $D_2$;
- Protein $P_2$ is influenced by drug $D_2$;
- Protein $P_3$ is influenced by drug $D_3$.

Represent this relationship in a multirelational graph. Is it a multipartite graph?

*Solution*: No, because the relationship "being incompatible with" relates entities of the same type (drugs).




$$\begin{bmatrix} & \text{I1} & \text{I2} & \text{D1} & \text{D2} & \text{D3} \\ \text{I1} & 0 & 0 & 1 & 1 & 0 \\ \text{I2} & 0 & 0 & 0 & 0 & 1 \\ \text{D1} & 1 & 0 & 0 & 0 & 0 \\ \text{D2} & 1 & 0 & 0 & 0 & 0 \\ \text{D3} & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} & \text{D1} & \text{D2} & \text{D3} \\ \text{D1} & 0 & 1 & 0 \\ \text{D2} & 1 & 0 & 1 \\ \text{D3} & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} & \text{D1} & \text{D2} & \text{D3} & \text{P1} & \text{P2} & \text{P3} \\ \text{D1} & 0 & 0 & 0 & 1 & 0 & 0 \\ \text{D2} & 0 & 0 & 0 & 1 & 0 & 0 \\ \text{D3} & 0 & 0 & 0 & 0 & 1 & 1 \\ \text{P1} & 1 & 1 & 0 & 0 & 0 & 0 \\ \text{P2} & 0 & 0 & 1 & 0 & 0 & 0 \\ \text{P3} & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

□

## 1.2. Graph embedding

When manipulating and comparing graphs, there is often interest in knowing whether two graphs are **similar**. The notion of graph similarity is muddy, since there is no set definition of what it means for two graphs to be similar.

In general, it is impractical to directly compute the similarity (however defined) between two graphs, because real graphs tend to have a very high amount of nodes. A better approach consists in retrieving a reasonable approximation of similarity by extracting features from the graph nodes.

The goal of these methods is to *encode* nodes as low-dimensional vectors that summarize their graph position and the structure of their local graph neighborhood. The idea is to **embed** nodes into a **latent space** where geometric relations between its elements correspond to relationships (that is, edges) in the original graph.

This model requires a pair of operation, an **encoder** and a **decoder**. An encoder is a function that maps each node in the graph into a low-dimensional vector, whereas a decoder is a function that takes the low-dimensional node embeddings and uses them to reconstruct information about each node's neighborhood in the original graph.

For this mapping to work it is necessary to assign a unique identifier to each node, so that the mapping is one-to-one. This can be done, given an arbitrary order of the nodes, by assigning an **indicator vector** to each node. The indicator vector $\boldsymbol{v}_i$ for a node $v \in V$ is a binary vector that has 0 in each component except a single component which is 1. In this way, each node has unique and unambiguous ID to be referred to.

Formally, the encoder is a function $\text{Enc} : V \to \mathbb{R}^d$ that maps nodes $v \in V$ to their respective vector embeddings $\boldsymbol{z}_v \in \mathbb{R}^d$, where $d$ is the dimension of the latent space[1]. The easiest way to define this function is to adopt the **shallow embedding** approach, where the encoding function simply performs a "lookup" on a matrix that contains all embedding vectors. More formally:

$$\text{Enc}(v) = \boldsymbol{Z}[v] = \boldsymbol{Z}\boldsymbol{v}_i = \boldsymbol{Z}_{\boldsymbol{v}_i} \qquad \boldsymbol{Z} = \left( (\boldsymbol{z}_{v_1}) \ (\boldsymbol{z}_{v_2}) \ (\dots) \ (\boldsymbol{z}_{v_{|v|}}) \right)$$

Where $\boldsymbol{Z} \in \mathbb{R}^{|V| \times d}$ is a matrix containing the embedding vectors for all nodes, $\boldsymbol{Z}[v]$ denotes the row of $\boldsymbol{Z}$ corresponding to node $v$ and $\boldsymbol{v}_i$ is the indicator vector for $v$. Note how shallow embedding is simply a matrix multiplication.

[1] Realistic applications of encoders have latent spaces whose dimension ranges from roughly 16 to 1000.

The decoder's purpose is to reconstruct certain graph statistics from the node embeddings that are generated by the encoder. For example, given a node embedding $\boldsymbol{z}_v$ of a node $v$, the decoder might attempt to predict the neighborhood of $v$ or its row $A[v]$ in the graph adjacency matrix.

The most common way to define a decoder is to define a **pairwise decoder**, a function $\text{Dec} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ that has two encodings as input and a (predicted) non-zero value as output:

$$\text{Dec}(\text{Enc}(u), \text{Enc}(v)) = \text{Dec}(\boldsymbol{Z}[u], \boldsymbol{Z}[v])$$

Many modern graph embedding approaches such as `GraRep` and `HOPE` define decoding as the dot product between the encodings:

$$\text{Dec}(\boldsymbol{Z}[u], \boldsymbol{Z}[v]) = \boldsymbol{Z}[u]^T \boldsymbol{Z}[v]$$

Of course, it is not possible to simply define the decoder as the inverse of the encoder, because encoding entails a partial loss of information. Therefore, the idea is to define the decoder so that the difference between the "real" value of a similarity measure and its estimated value is as close as possible:

$$\text{Dec}(\boldsymbol{Z}[u], \boldsymbol{Z}[v]) \approx \boldsymbol{S}[u, v]$$

Where $\boldsymbol{S}[u, v]$ is a generic graph-based similarity measure between the nodes $u$ and $v$ and $\boldsymbol{S} \in \mathbb{R}^{|V| \times |V|}$ is the similarity matrix summarizing all pairwise node similarities.

To achieve this "closeness" to the real similarity measure, the standard practice is to minimize an empirical reconstruction loss $L$ over a set of training node pairs $D$:

$$L = \sum_{(u,v) \in D} l(\text{Dec}(\boldsymbol{Z}[u], \boldsymbol{Z}[v]), \boldsymbol{S}[u, v])$$

Where $l$ is any loss function measuring the discrepancy between the decoded (that is, estimated) similarity values $\text{Dec}(\boldsymbol{Z}[u], \boldsymbol{Z}[v])$ and the true values $\boldsymbol{S}[u, v]$.

The overall objective is to train the encoder and the decoder so that pairwise node relationships can be effectively reconstructed on the training set $D$. The choice of $l$ depends on the definition of the decoder and the similarity function, and different graph embedding algorithms use different loss functions.

Algorithms such as `GraRep` and `HOPE` use *mean square error* as loss function, which would give:

$$L = \sum_{(u,v) \in D} ||\text{Dec}(\boldsymbol{Z}[u], \boldsymbol{Z}[v]) - \boldsymbol{S}[u, v]||_2^2$$

Other algorithms, such as `DeepWalk`, use cross entropy:

$$L = \sum_{(u,v) \in D} -\boldsymbol{S}[u, v] \log(\text{Dec}(\boldsymbol{Z}[u], \boldsymbol{Z}[v]))$$

It is therefore necessary to give a definition of $\boldsymbol{S}[u, v]$. As stated before, there is no set definition of similarity, but many were proposed. The easiest one is given by counting the number of neighbors that are shared between the two nodes:

$$\boldsymbol{S}[u, v] \coloneqq |N(u) \cap N(v)|$$

A more refined approach is given by considering general neighborhood overlap measures. One such example is the **Katz index**, obtained by counting the number of paths of any length between a pair of nodes and summing them together:

$$\boldsymbol{S}[u, v] \coloneqq S_{\text{Katz}}[u, v] = \sum_{i=1}^{\infty} \beta^i (\boldsymbol{A}[u, v])^i$$

Where $\beta$ is a user-defined parameter that tunes how much weight is given to short versus long paths. A small value of $\beta$ would give short paths more weights and long paths less weight, for example. The value of $\beta$ must be strictly positive and must be less than the greatest eigenvalue of $\boldsymbol{A}$.

General neighborhood overlap measures are employed as the notion of similarity by graph embedding algorithms such as `HOPE`, whereas algorithms such as `GraRep` define $\boldsymbol{S}$ based on powers of the adjacency matrix.

Note that, even though easy to perform, shallow encoding suffers from some problems. Notably:

- Does not share parameters between nodes on the encoding;
- Does not preserve node features, only indexing;
- Is **transductive**: applies only to the nodes present during the training phase.