

Master Thesis



Federico Calabrò 247976

Department of Mathematics and Computer Science  
2024

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Graph Data Structure</b>                              | <b>3</b>  |
| 1.1      | Formal Definition . . . . .                              | 3         |
| 1.2      | Types of Graphs . . . . .                                | 3         |
| 1.3      | Graph Representation . . . . .                           | 5         |
| <b>2</b> | <b>Neural Networks</b>                                   | <b>5</b>  |
| 2.0.1    | Basic Structure of a Neural Network . . . . .            | 6         |
| 2.1      | Training Neural Networks . . . . .                       | 6         |
| 2.1.1    | Backpropagation . . . . .                                | 6         |
| 2.1.2    | Gradient Descent . . . . .                               | 7         |
| 2.2      | Advanced Topics in Neural Networks . . . . .             | 7         |
| 2.2.1    | Deep Neural Networks . . . . .                           | 7         |
| 2.2.2    | Convolutional Neural Networks . . . . .                  | 8         |
| 2.2.3    | Recurrent Neural Networks . . . . .                      | 8         |
| 2.3      | Graph Neural Networks and Attention Mechanisms . . . . . | 9         |
| 2.3.1    | Graph Neural Networks . . . . .                          | 9         |
| 2.3.2    | Attention Mechanisms . . . . .                           | 9         |
| <b>3</b> | <b>Graph Similarity Problem</b>                          | <b>9</b>  |
| 3.1      | Graph Edit Distance (GED) . . . . .                      | 10        |
| 3.2      | Atomic Operations . . . . .                              | 10        |
| 3.3      | SimGNN . . . . .   | 11        |
| <b>4</b> | <b>Bottleneck Loading GPU Problem</b>                    | <b>11</b> |
| <b>5</b> | <b>Dataset Generation</b>                                | <b>11</b> |
| <b>6</b> | <b>State of the Art Review</b>                           | <b>11</b> |

# 1 Graph Data Structure

A *graph*  $G$  [Figure 1] is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are arcs that connect any two nodes in the graph. Graphs are used to represent relationships between different entities and have applications in many fields including Computer Science, Physics, Biology, Chemistry, Optimization Theory and many more.

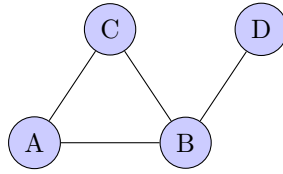


Figure 1: A simple graph

## 1.1 Formal Definition

A graph is formally defined as a tuple  $G = (V, E)$ , where:

- $V$  is a finite set of vertices (or nodes).
- $E$  is a set of edges, where each edge is an unordered pair of distinct vertices from  $V$ . Thus,  $E \subseteq \{\{u, v\} \mid u, v \in V \text{ and } u \neq v\}$ .

## 1.2 Types of Graphs

Graphs can be classified into various types based on their properties, including:

- **Directed Graph** [Figure 2]: A graph in which the edges have a direction, i.e., each edge is an ordered pair of vertices.

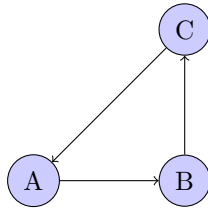


Figure 2: Directed graph example.

- **Undirected Graph** [Figure 3]: A graph in which the edges do not have a direction, i.e., each edge is an unordered pair of vertices.

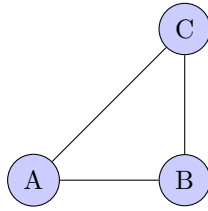


Figure 3: Undirected graph example.

- **Weighted Graph** [Figure 4]: A graph in which a weight (or cost) is associated with each edge.

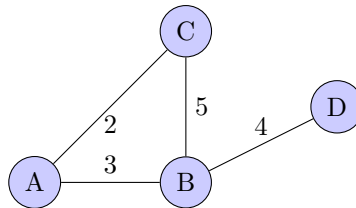


Figure 4: Weighted graph example.

- **Simple Graph** [Figure 5]: A graph with no loops (edges connecting a vertex to itself) and no multiple edges (more than one edge connecting the same pair of vertices).

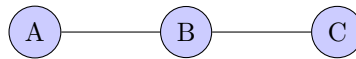


Figure 5: Simple graph example.

- **Complete Graph** [Figure 6]: A graph in which there is exactly one edge between each pair of distinct vertices.

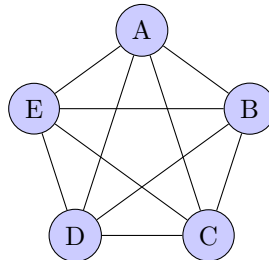


Figure 6: Complete graph example.

- **Bipartite Graph** [Figure 7]: A graph whose vertices can be divided into two disjoint sets  $U$  and  $W$  such that every edge connects a vertex in  $U$  to a vertex in  $W$ .

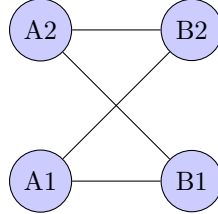


Figure 7: Bipartite graph example.

### 1.3 Graph Representation

Graphs can be represented in various ways, including:

- **Adjacency Matrix** [Figure 8]: A square matrix  $A$  of size  $|V| \times |V|$  where  $A_{ij} = 1$  if there is an edge between vertices  $v_i$  and  $v_j$ , and  $A_{ij} = 0$  otherwise.

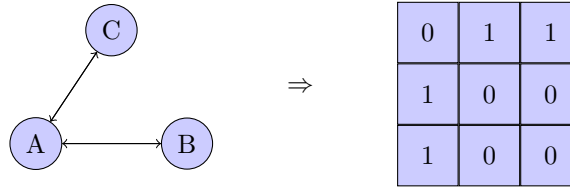
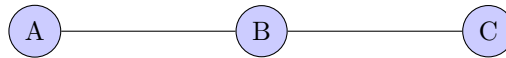


Figure 8: Adjacency Matrix representation.

- **Adjacency List** [Figure 9]: An array of lists. The array contains a list for each vertex, and each list contains the vertices that are adjacent to the corresponding vertex.



$$[ \text{Adj}(A) = [B], \quad \text{Adj}(B) = [A, C], \quad \text{Adj}(C) = [B] ]$$

Figure 9: Adjacency List representation for an undirected graph.

## 2 Neural Networks

A *neural network* is a computational model that is inspired by the way biological neural networks in the human brain process information. It consists of intercon-

nected units called neurons that work together to solve specific problems. The basic building block of a neural network is the perceptron, which computes a weighted sum of its inputs and passes the result through an activation function.

### 2.0.1 Basic Structure of a Neural Network

A simple neural network [Figure 10] consists of three types of layers:

- **Input Layer:** The layer that receives the input data.
- **Hidden Layers:** One or more intermediate layers that process the inputs received from the input layer.
- **Output Layer:** The layer that produces the final output.

The following figure illustrates a basic neural network with one hidden layer:

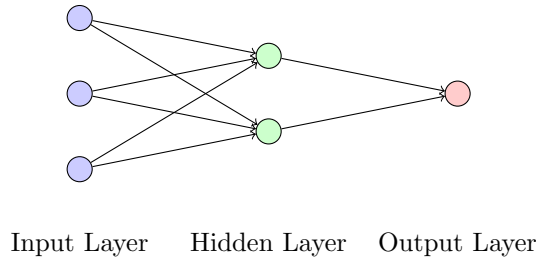


Figure 10: A simple neural network with one hidden layer.

## 2.1 Training Neural Networks

Training a neural network involves adjusting the weights of the connections to minimize the error between the predicted output and the actual output. This is typically done using a method called *backpropagation* along with an optimization algorithm like *gradient descent*.

### 2.1.1 Backpropagation

Backpropagation [Figure 11] is an algorithm used to compute the gradient of the loss function with respect to each weight by the chain rule, iterating backward from the last layer to the first layer.

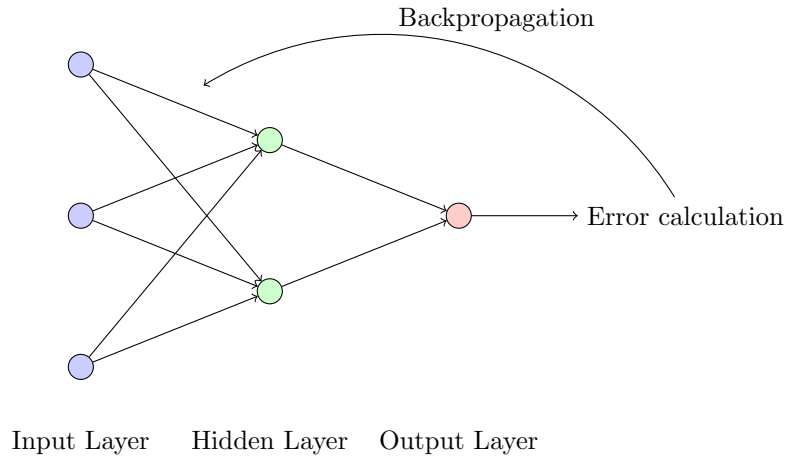


Figure 11: A simple neural network with one hidden layer.

### 2.1.2 Gradient Descent

Gradient descent [Figure 12] is an optimization algorithm used to minimize the loss function by iteratively moving towards the steepest descent, based on the computed gradients.

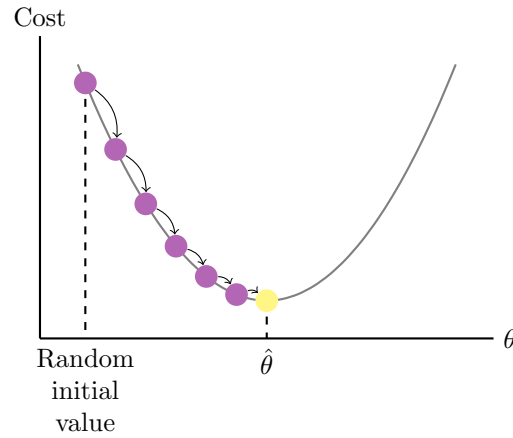


Figure 12: Gradient descent method applied on a simple function.

## 2.2 Advanced Topics in Neural Networks

### 2.2.1 Deep Neural Networks

A *deep neural network* (DNN) is an artificial neural network with multiple hidden layers between the input and output layers. DNNs can model complex

patterns and relationships in data, making them powerful tools for tasks such as image and speech recognition.

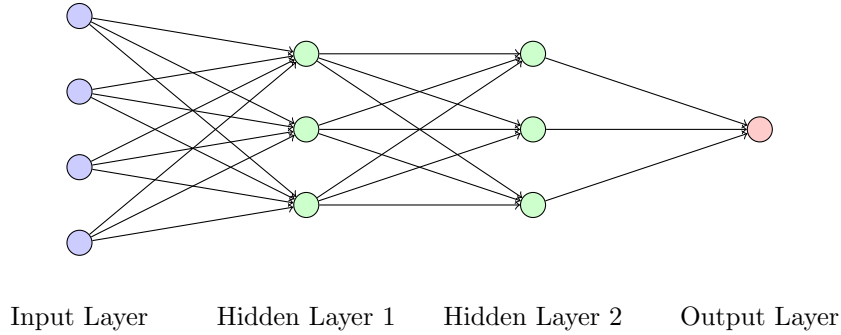


Figure 13: A Deep neural network with two hidden layers.

### 2.2.2 Convolutional Neural Networks

A *convolutional neural network* (CNN) is a specialized type of neural network designed for processing structured grid data, like images. CNNs use convolutional layers that apply a series of filters to the input data to extract features.

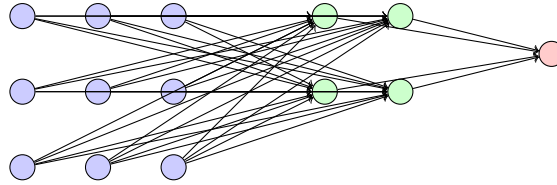


Figure 14: A simple convolutional neural network architecture.

### 2.2.3 Recurrent Neural Networks

A *recurrent neural network* (RNN) is a type of neural network that is well-suited for sequential data, such as time series or natural language. RNNs have connections that form directed cycles, allowing information to persist.

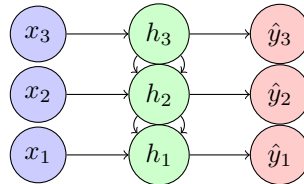


Figure 15: Recurrent Neural Network (RNN) unrolled through time.



## 2.3 Graph Neural Networks and Attention Mechanisms

### 2.3.1 Graph Neural Networks

A *graph neural network* (GNN) is a type of neural network designed to handle graph-structured data. GNNs generalize neural networks to work directly on the graph domain by incorporating the graph’s structure into the learning process. Nodes in a GNN aggregate feature information from their neighbors through multiple layers, enabling the network to capture the dependencies and relationships inherent in the graph.

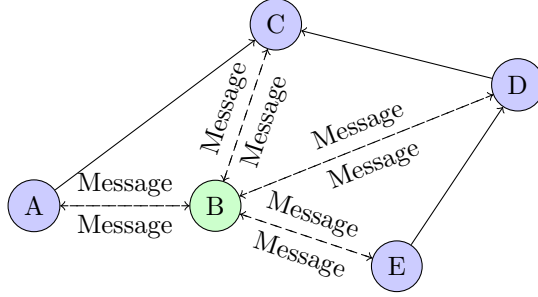


Figure 16: Message passing between node B and its neighbors in a GNN.

### 2.3.2 Attention Mechanisms

Attention mechanisms are techniques used to enhance the performance of neural networks by dynamically focusing on relevant parts of the input data while processing. The most famous use of attention mechanisms is in the *Transformer* model, which has revolutionized natural language processing tasks. The attention mechanism computes a weighted sum of values, where the weights are derived from the compatibility of the query with the corresponding keys, allowing the model to selectively focus on important parts of the input. The combination of GNNs and attention mechanisms has shown significant promise in various applications, such as molecular property prediction, social network analysis, and more.

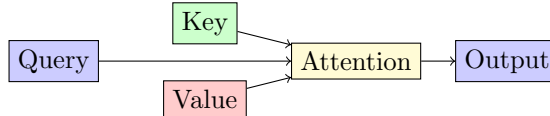


Figure 17: Attention mechanism in neural networks.

## 3 Graph Similarity Problem

The *graph similarity problem* involves determining the degree of similarity between two graphs. This problem has numerous applications in pattern recogni-

tion, computer vision, bioinformatics, and other fields. One common method to quantify graph similarity is through the *graph edit distance* (GED).

### 3.1 Graph Edit Distance (GED)

The *graph edit distance* between two graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  is defined as the minimum cost required to transform  $G_1$  into  $G_2$  using a sequence of atomic operations. Formally, let  $\Sigma$  be the set of all possible edit operations, and let  $c : \Sigma \rightarrow \mathbb{R}^+$  be a cost function that assigns a positive real number to each operation. The GED is then given by:

$$\text{GED}(G_1, G_2) = \min_{\sigma \in \Sigma^*} \sum_{o \in \sigma} c(o)$$

where  $\Sigma^*$  denotes the set of all finite sequences of operations from  $\Sigma$ , and  $o$  represents an individual operation within a sequence  $\sigma$ .

The computation of GED is known to be **NP-HARD** [1], indicating that finding the exact minimum edit distance between two graphs is computationally intensive.

### 3.2 Atomic Operations

The basic atomic operations typically includes:

- **Vertex Insertion:** Inserting a new vertex  $v$  into the graph.
- **Vertex Deletion:** Deleting an existing vertex  $v$  from the graph.
- **Edge Insertion:** Inserting a new edge  $e = \{u, v\}$  into the graph.
- **Edge Deletion:** Deleting an existing edge  $e = \{u, v\}$  from the graph.

To demonstrate the Graph Edit Distance (GED), consider pair of graphs represented in [Figure 18](#):

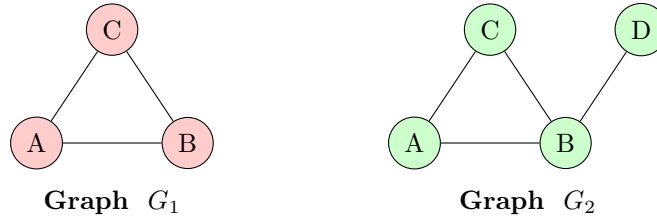


Figure 18: Pair of graphs to demonstrate Graph Edit Distance.

In this example, graph  $G_1$  has vertex set  $V_1 = \{A, B, C\}$  and edge set  $E_1 = \{\{A, B\}, \{A, C\}, \{B, C\}\}$ , while graph  $G_2$  has vertex set  $V_2 = \{A, B, C, D\}$  and edge set  $E_2 = \{\{A, B\}, \{A, C\}, \{B, C\}, \{B, D\}\}$ . The transformation (which

cost is lowest) from  $G_1$  to  $G_2$  involves: Inserting the vertex  $D$  and Inserting the edge  $\{B, D\}$ . If we assign a cost of 1 to each operation the total cost (GED) is:  $1 + 1 = 2$ .

### 3.3 SimGNN

SimGnn

## 4 Bottleneck Loading GPU Problem

## 5 Dataset Generation

## 6 State of the Art Review

## References

- [1] Wikipedia. Np-hard, 2024. <https://en.wikipedia.org/wiki/NP-hardness>.