SAPIENZA
UNIVERSITÀ DI ROMA

Parameterized Counterfactual Explanations for Node Classification in Graph Neural Networks

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di laurea magistrale in
Engineering in Computer Science - Ingegneria Informatica

Giammarco D'Alessandro
ID number 1753102

Advisor
Prof. Fabrizio Silvestri

Co-Advisor
Prof. Simone Scardapane

Academic Year 2022/2023

Thesis defended on 31 October 2023
in front of a Board of Examiners composed by:

Prof. Aristidis Anagnostopoulos,
Prof. Antonio Cianfrani,
Prof. Fabrizio D'Amore,
Prof. Luca Iocchi,
Prof. Massimo Panella,
Prof. Gabriele Proietti Mattia,
Prof. Leonardo Querzoni,
Prof. Simone Scardapane,
Prof. Fabrizio Silvestri,
Prof. Angelo Spognardi,
Prof. Andrea Vitaletti,
Prof. Massimo Mecella (chairman)

Parameterized Counterfactual Explanations for Node Classification in Graph Neural Networks
Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: dalessandro.1753102@studenti.uniroma1.it

"Writers foresee the inevitable,
and although catastrophes and problems
may be inevitable, solutions are not."
- Isaac Asimov

Abstract

As Graph Neural Networks (GNNs) gain traction in real-world applications like computational biology (Cheung and Moura, 2020), natural language processing (NLP, L. Wu et al., 2022), and computer security (J. Zhou et al., 2018), the need for explainability becomes apparent due to the inherent "black-box" nature of these deep learning models. Current approaches for interpreting GNN predictions have predominantly concentrated on individuating subgraphs that are especially relevant for a particular prediction, delving into crucial graph substructures and node features. As a result, the generated explanations are painstakingly customized for each instance.

A fresh take on this problem has emerged recently, counterfactual (CF) reasoning, where the goal of an explainer algorithm is to induce a change in the GNN prediction by minimal perturbation of the input structure. Existing methods for CF explanation of GNNs are limited to instance-specific local reasoning with the major drawback of not being capable of providing a global perspective over a natural set of instances (e.g. graphs of a given class).

In this study, to advance beyond the current limitation of existing counterfactual GNN explainers, we introduce a novel architecture to generate counterfactual explanation for GNNs, **C**ounter**F**actual **P**arameterized **G**raph neural network **E**xplainer (**CFPGE**xplainer), which aims to overcome inability to generalize the explanation to a model-level comprehension and the lack of efficient generation.

Contents

Chapter 1

Introduction

Over the past few decades, the field of Artificial Intelligence (AI) has undergone substantial transformation. With a considerable increase in available computational resources, AI algorithms have garnered substantial interest in both industry and research. Among the most interesting innovations within the AI panorama, Graph Neural Networks (GNNs) have emerged as a powerful tool for analyzing complex relational data represented in the form of graphs. Graphs are pervasive in various domains, including social networks, biological systems, transportation networks, and recommendation systems. For instance, a highly promising application of GNNs lies in the field of drug discovery. Recent research has demonstrated that, with an ample amount of data, GNNs are capable of achieving performance comparable to that of human-engineered fingerprints or descriptors in forecasting the molecular properties of potential antibiotics. This, in turn, contributes to guiding the creation of new molecules and enhancing our comprehension of molecular structures (Cheung and Moura, 2020; Xiong et al., 2020a).

Despite many GNN algorithms have achieved remarkable performances in many real-world application, the inner workings of their decision-making processes often remain obscure to the end users of these applications. More specifically, several Machine Learning (ML) based algorithms are frequently regarded as "black-box" algorithms due to their complexity, involving millions of parameters that are challenging to interpret and optimize during the training phase. This complexity renders the output of the algorithm difficult to explain. The inability to elucidate these automated decisions erodes users' trust, subsequently diminishing the usability of such systems (Ribeiro et al., 2016). As high-stakes AI applications become increasingly prevalent in daily life, the demand for their explanatory capabilities is correspondingly on the rise. A significant indicator of this requirement is the legal regulation within the European Union, where the General Data Protection Regulation (GDPR, Council of European Union, 2016) now mandates the need for

explaining the reasoning mechanisms behind such applications.

One promising approach to address the interpretability challenge in GNNs is the integration of counterfactual explanations in eXplainable Artificial Intelligence (XAI[1]). Counterfactual explanations provide meaningful insights into model predictions by identifying what changes in the input features would lead to a different prediction. In the context of node classification, counterfactual explanations can elucidate the necessary alterations to the node's attributes or connections that would result in a different predicted label.

This thesis focuses on the incorporation of counterfactual explanations in XAI techniques to enhance interpretability in GNNs for node classification. Understanding why a GNN predicts a specific label for a node is crucial for building trust, improving model robustness, and gaining insights into the underlying data. We explore how these explanations can shed light on the decision-making process of GNNs and provide actionable insights for stakeholders. The objective is to bridge the gap between predictive accuracy and model interpretability, facilitating the deployment of GNNs in real-world applications where transparency and trustworthiness are paramount. To this end, we introduce a novel exaplaining system, **C**ounterFactual **P**arameterized **G**raph neural network **E**xplainer (CFPGExplainer), which aims to overcome the current limitation of existing counterfactual GNN explainers, such as the inability to generalize the explanation to a model-level comprehension and the lack of efficient generation.

The subsequent chapters of this thesis are organized as follows: Chapter 2: "Background", provides an in-depth review of the related literature, covering topics related to deep learning and Graph Neural Networks (GNNs), exaplainable artificial intelligence and counterfactual explanations. Chapter 3: "Parameterized Counterfactual Explanations for Node Classification in Graph Neural Networks", outlines the proposed methodologies for generating counterfactual explanations for GNNs, including algorithmic details and rationales. Chapter 4: "Experimental setup" presents the experimental setup, datasets, evaluation metrics, and results obtained from assessing the effectiveness of the proposed methodologies. Chapter 5: "Conclusions", summarizes the contributions of the thesis, discusses limitations, and outlines potential directions for future research in this domain.

[1]The acronym was made popular by the USA Defense Advanced Research Projects Agency when launching to the research community the challenge of designing self-explanatory AI systems (https://www.darpa.mil/program/explainable-artificial-intelligence).

Chapter 2

Background

Understanding the context and rationale behind the research is crucial for appreciating the significance and relevance of the work presented herein. This background chapter provides an overview of the fundamental concepts, theories, and existing research relevant to Graph Neural Networks (GNNs) and eXplainable Artificial Intelligence (XAI). It sets the stage by presenting the existing knowledge landscape, highlighting key theories, models, and advancements in the field of study.

In section 2.1 an introduction of the background theory of GNNs is presented, starting from the basis of graph theory (section 2.1.1) and the foundation of deep learning (section 2.1.2), and proceeding with a more detailed description of GNNs, and their variations (section 2.1.3). In section 2.2 instead faces the other fundamental topic of this work, XAI. First an overview of the main concept in this field, explainablity and interpretability, is introduced (section 2.2.1) together with some example solutions (section 2.2.2), subsequently sections 2.2.3 and 2.2.4 deal with explainability focused on GNNs and counterfactual reasoning, respectively.

2.1 Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs, Scarselli et al., 2009) represent a revolutionary paradigm in the field of machine learning, particularly suited for analyzing and extracting insights from complex structured data, such as graphs. Traditional neural networks excel at processing signals like images, speech, videos or sequences in which there is an underlying Euclidean structure, but they struggle with non-Euclidean data, like graphs. Such kinds of data implies that there are no such familiar properties as global parameterization, common system of coordinates, vector space structure, or shift-invariance. Consequently, basic operations like convolution that are taken for granted in the Euclidean case are even not well defined on non-Euclidean domains (Bronstein et al., 2017).

GNNs, on the other hand, have emerged as a powerful tool for handling these intricate data structures by incorporating graph-related properties. Recent developments have increased their capabilities and expressive power. Practical applications have become popular in a really wide spectrum of different areas such as drug discovery (Xiong et al., 2020b), physics simulations (Sanchez-Gonzalez et al., 2020), fake news detection (Monti et al., 2019), recommendation systems (Eksombatchai et al., 2017) and many others (Hamilton, 2020).

In essence, GNNs are a class of deep learning models designed to operate on graphs, capturing both the node-level and the edge-level information within a graph. They are engineered to learn meaningful representations of nodes and edges by leveraging the graph's topology and associated features. These representations can be exploited in performing various tasks, such as node classification, link prediction, community detection, and graph generation. More generally GNNs have shown remarkable performance in domains where relationships and interactions between entities are crucial for understanding system dynamics and making informed predictions.

This introduction will delve deeper into the foundations, architectures, and applications of Graph Neural Networks, providing a comprehensive understanding of this cutting-edge approach to processing complex relational data. First it will go through some basics of graph theory and it will analyze how these concepts can be used to model real world scenarios, and how neural networks can exploit the information encoded in such models. Last it will added a brief presentation of different types of GNNs and their peculiarities.

2.1.1 From graphs to Graph Neural Networks

The theory of graphs (also graph theory) can be traced back at least to the illustrious swiss mathematician Leonhard Euler, who in a 1736 paper (N.L.Biggs et al., 1987) solved a puzzle about an optimal tour of the town of Königsberg (fig. 2.1). Some more developments were made in the $19th$ century, the very term "graph" was introduced in this period by Sylvester in a paper published in 1878 in Nature (Sylvester, 1878), where he draws an analogy between "quantic invariants" and "co-variants" of algebra and molecular diagrams. The field straight-up exploded during the $20th$ century, the first textbook on graph theory was published in 1936 by Dénes Kőnig, followed by many others (Tutte, 2001); currently graphs are one of the principal objects of study in discrete mathematics. Graphs, in the realm of mathematics and computer science, are powerful mathematical abstractions used to model relationships between

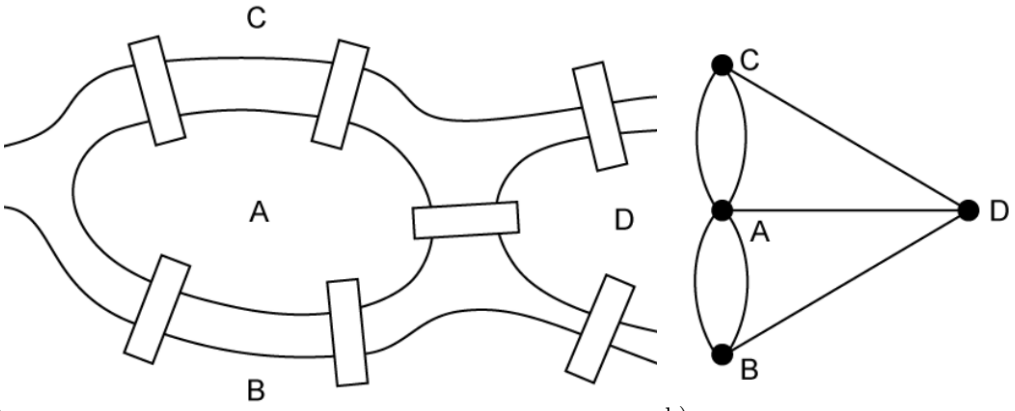


Figure 2.1. *A schematic representation of the seven bridges of Königsberg problem (fig.a), for which Euler (1707–1783) proposed a negative resolution in his famous 1736 paper, laying the foundation for modern graph theory. The question was if it is possible to traverse all the bridges only once and come back to the point where the trip was started. The answer was negative – it is not possible to get across all bridges exactly once. Mathematically the problem comes down to looking for a Eulerian cycle in a multigraph (multiple edges connecting two vertices) with four nodes and seven edges (fig.b).*

entities. In its basic type, a simple graph G can be defined as the couple:

$$G = (V, E) \tag{2.1}$$

where $V = \{v_1, v_2, ..., v_N\}$ is the set of *vertices* or *nodes*, the entities ($n = |V|$), and $E = \{(v_i, v_j)|v_i, v_j \in V; i \neq j\}$ is the set of *edges* ($m = |E|$), also called links, modeling the relationship between two entities. A basic distinction is made between *undirected* graphs, where edges link two vertices symmetrically, i.e. there is no difference between the edges (v_i, v_j) and (v_j, v_i); and *directed* graphs, where edges link two vertices asymmetrically, thus (v_i, v_j) and (v_j, v_i) would be two separated edges (Cormen et al., 2022). Moreover two other categorization are generally made: considering the types of its components and their behavior over time. A graph can be *homogeneous*, when nodes and edges of the graph have all the same type; or *heterogeneous* when nodes and edges are associated with different types. More specifically, in a heterogeneous graph (V, E, ϕ, ψ), each node v_i is associated with a type $\phi(v_i)$ and each edge e_j with a type $\psi(e_j)$. When input features or the topology of the graph vary with time, i.e. the existence of nodes and edges may change at different points in time, the graph is regarded as *dynamic*, otherwise is considered *static*. The time information should be carefully considered in dynamic graphs (Y. Wang et al., 2021; J. Zhou et al., 2021). Note that these categories are orthogonal, meaning that these properties can be combined in the same graph, e.g. one can deal with a dynamic directed heterogeneous graph.

One of the typical application of graphs is modeling the relationships inside social networks (Y. Wu et al., 2020), tools to study patterns in collective behaviour of people, institutions and organizations, where the set of social actors can be represented with the nodes of the graph, and their relative interactions are represented by the edges (e.g. the standard example for a social graph would be a "friendship graph"; here, V is again a set of people and E is the set of $(u, v) \in V$ such that u and v are friends). This perspective on social network provides a set of methods for analyzing the structure of whole social entities as well as a variety of theories explaining the dynamics and patterns observed in these structures (Wasserman and Faust, 1994). Even though realities such as social networks can be easily seen as graphs and modeled accordingly this data structures are an extremely powerful and general representation of data, many other domains can actually be modeled as graphs, like images and text. Although counterintuitive, one can learn more about the symmetries and structure of images and text by viewing them as graphs. Traditionally, we visualize images as rectangular grids with image channels, often representing them as arrays. However, an alternative way could be seeing images as graphs possessing a regular structure, where each pixel is a node connected to each neighboring pixels through an edge (fig. 2.2). Specifically, each non-border pixel will maintain precisely 8 neighbors, and the information stored at every node could be a 3-dimensional vector signifying the RGB value of the corresponding pixel. Also text can be easily converted into a graph format by assigning numerical indices to individual characters, words, or tokens, thus creating a sequence of these indices to represent the text. This process yields a straightforward directed graph, where each word or token is a node connected to the succeeding node through an edge. Of course, this is not usually how text and images are encoded: these graph representations are redundant since all images and all text will have very regular structures. In practice, images have a banded structure in their adjacency matrix because all nodes (pixels) are connected in a grid; and the adjacency matrix for text is just a diagonal line, because each word only connects to the prior word, and to the next one, but those trivial examples are supposed to serve as an intuition of how flexible graphs are and how wide can thus be the range of their applications (Sanchez-Lengeling et al., 2021).

Graph representation

Before going into the details of machine learning applied to graphs, we propose here an introduction on how to represent graphs, their components and the relative features. This problem is crucial to encode real world information into graph models and to be able to exploit the complete potentiality of such abstractions in real world

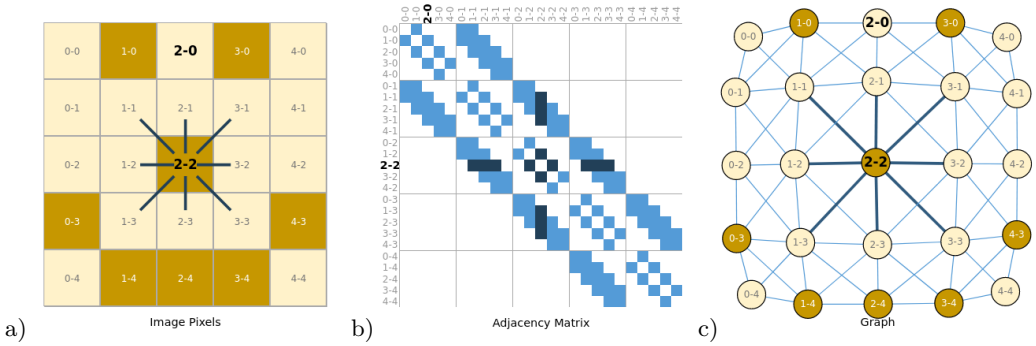


Figure 2.2. *A way of visualizing the connectivity of a graph is through its adjacency matrix. Picture a) shows a simple* 5×5 *image of a smiley face and each of the 25 pixels is indexed as the elements of a* 5×5 *matrix. In picture b) we can see the one of the possible adjacency matrices representing the graph of our smiley face image. This matrix has dimension* $n_{nodes} \times n_{nodes}$ *(*25×25*), given that each pixel is a node, connected to all its neighboring pixels through an edge (fig.a,b,c highlights pixel 2-2). Rightmost picture (fig.c), shows the actual graph encoding the smiley image. It is important noticing that each of these three representations below are different views of the very same piece of data (Sanchez-Lengeling et al., 2021).*

applications. Machine learning models typically take rectangular or grid-like arrays as input, more generally we talk about tensors. In mathematics, a tensor is an algebraic object that describes a multi-linear relationship between sets of algebraic objects related to a vector space. Tensors may map between different objects such as vectors, scalars, and even other tensors (Vasilescu, 2009). In machine learning, tensors have a broader definition and usually refers also to multidimensional arrays, often called informally "data tensors". The rationale behind this representation is that data stored in such a format can be readily fed into any artificial intelligence model based on neural networks, thereby opening up the possibility of analyzing them through a diverse range of algorithms and models. So, it's not immediately intuitive how to represent graph structured data in a format that is compatible with deep learning.

Graphs can have up to four types of elements that can be potentially useful to perform machine learning tasks: nodes, edges and connectivity. A solution to represent the first two is relatively straightforward. The most common choice to represent nodes is to assign to each node v an index i (e.g. an integer) as an identifier, and a d-dimensional vector to encode the node features, so that, in general, we will have a node feature matrix X of dimension $n \times d$ grouping all node data. For what concerns edges we can simply follow the definition (section 2.1.1) identifying each edge e by the couple of nodes it connects in the graph (e.g. if edge e connect nodes i and j, we have $e = (v_i, v_j)$) and follow the same idea, if necessary, to encode edge features (in this case the edge feature matrix will have dimension $m \times d$, being

m the number of edges) (Cormen et al., 2022). However, representing a graph's connectivity is more complicated. Several data structures have been proposed to tackle this problem, among them three have resulted to be the most efficient and thus the most used in practice: adjacency matrix, incidence matrix and adjacency list.

The *adjacency matrix* is a two-dimensional tensor ($n \times n$, being n the number of nodes, indicated as $A \in 0, 1^{n \times n}$), in which the rows represent source vertices and columns represent destination vertices. Data on edges and vertices must be stored externally. Only the cost for one edge can be stored between each pair of vertices. We note that any two vertices connected by an edges are called adjacent, thus the term *adjacency*. The *incidence matrix* is a two-dimensional tensor ($n \times m$, being $m = |E|$ the number of edges), in which the rows represent the vertices and columns represent the edges, so that the entries indicate the incidence relation between the vertex at a row and edge at a column (an edge e is said to be incident to a node v, if e is connected to v). *Adjacency lists* can be of two types. In one case nodes can be stored as records or objects, and every vertex stores the list of its adjacent vertices. This data structure allows the storage of additional data on the vertices, and even additional data on edges can be stored if they are also stored as objects, in which case each vertex stores its incident edges and each edge stores its incident vertices. The second case is the most common format of *adjacency list*, where all edges are stored together in one single list, which loses the explicit information of the vertices adjacent to a particular node, gaining on efficiency on space usage (Cormen et al., 2022).

Perhaps, since it can be easily tensorisable the most obvious choice would be to use an adjacency matrix to represent a graph for machine learning purposes. Yet, this particular representation comes with certain limitations. As illustrated by the example dataset table, the quantity of nodes within a real world graph can reach millions, with a highly fluctuating number of edges per node. Consequently, this frequently results in adjacency matrices with significant sparsity, rendering them highly inefficient in terms of space usage. Another big issue is that there can be many adjacency matrices encoding the same connectivity relations (i.e. the same graph), and there could be no guarantee that these different matrices would produce the same result in a deep neural network, that is to say, they are not permutation invariant. The permutation invariance property is a fundamental inductive bias of graph-structured data, resulting in the fact that for a graph with n nodes, there are up to $n!$ different adjacency matrices that are equivalent representations of the same graph. Learning permutation invariant operations is an independent area of recent research, we refer the reader to (Mena et al., 2018; Murphy et al., 2019) for further

details.

There are now dozens (if not hundreds) textbooks available on the subject (Berge and Minieka, 1976; Bondy and Murty, 2011; Cormen et al., 2022; Diestel, 2017; Griffin, 2023; only to cite few relevant ones); we refer the reader to the mentioned books to delve further into the details of all aspects of graph theory given that such a discussion deviates from the main purpose of this background introduction, which is to show the importance of graphs in mathematics, computational sciences and various fields, and why recently a great interest in applying deep learning to these concepts arose.

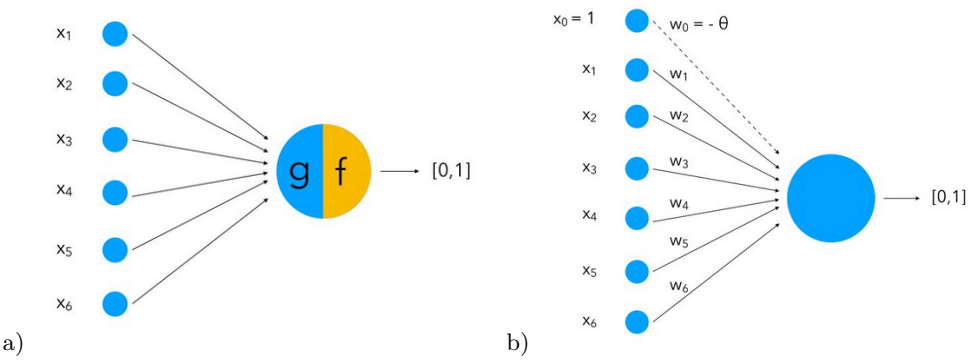
2.1.2 Deep Learning on graphs

Deep learning is part of a broader family of machine learning (ML) methods, mainly based on extracting patterns from raw data using Artificial Neural Networks (ANNs) and representation learning. The adjective "deep" in deep learning refers to the process of learning complicated concepts by building them from simpler ones in a hierarchical or multi-layer manner (LeCun et al., 2015). Artificial neural networks are popular realizations of such deep multi-layer hierarchies. In the past few years, the growing computational power of modern GPU-based computers and the availability of large training datasets have allowed successfully training neural networks with many layers and degrees of freedom (Bronstein et al., 2017).

history

The history of deep learning can be traced back several decades, with its roots in ANNs and machine learning. In 1943, drawing inspiration from the structure of a biological neuron, Warren McCulloch (a neuroscientist) and Walter Pitts (a logician) proposed a mathematical model of an artificial neuron, providing a foundational concept for the birth of artificial neural networks (McCulloch and Pitts, 1943).

Basically, a neuron in our brain takes an input signal (through the *dendrites*), processes it in its nucleus (*soma*), passes the output through a cable like structure to other connected neurons (*axon* to synapse to another neuron's dendrites). Although this oversimplification may not align with the biological intricacies of neurons, it offers a high-level representation of how a neuron in our brain functions: receiving input, processing it, and producing an output. This first computational model can be broken down in two parts (fig. 2.3a): the first part, g takes one or more inputs, performs an aggregation and based on the obtained value the second part, f produces an output, i.e. makes a decision. It is following the idea that the interconnection of many simple units, carrying out a simple task, can create the astonishing complexity of human brain that the first artificial neuron was born.



a) b)

Figure 2.3. *On the left (fig.a) a schema of the first artificial neuron introduced by McCulloch and Pitts in 1943. On the right (fig.b) the schema of the perceptron model, introduced by Rosenblatt in 1957. The crucial differences introduced by Rosenblatt are that a weight (w_i) is attached to each input and an input (x_0) of value 1 and weight $-\theta$ is added (this is called bias). Instead of having only the inputs and the weight and compare them to a threshold, the perceptron also learn the threshold as a weight for a standard input of value 1. This produces sort of a weighted sum of inputs, to which we apply an activation function, resulting in the final output.*

Later on, in 1957, Frank Rosenblatt developed the perceptron, a single-layer neural network capable of binary classification (Rosenblatt, 1958). It was designed to overcome most issues of the McCulloch-Pitts neuron: it can process non-boolean inputs and it can assign different weights to each input automatically; the threshold Θ is computed automatically (fig. 2.3b). It gained attention for its potential in pattern recognition tasks. The version of Perceptron we use nowadays as standard artificial neuron was introduced in 1969 by Minsky and Papert. Their book criticized the limitations of perceptrons and brought a major improvement to the previous model adding an activation function after the in put aggregation step. The activation function might take several forms and should restrict the weighted sum into a smaller set of possible values that allows a better classification of the output Minsky and Papert, 1969. It's a smoother version than the thresholding applied before. The first comprehensive learning algorithm for supervised, deep, feedforward, multilayer perceptrons was introduced in 1967 by Alexey Ivakhnenko and Lapa (Ivakhnenko et al., 1967). In a subsequent 1971 paper, they described a deep network with eight layers trained by the group method of data handling.

In 1989, the book "Parallel Distributed Processing" by Rumelhart, Hinton, and Williams popularized the backpropagation algorithm reviving the interest in the possibility of an efficient training for deep neural networks, and posed the basis for all modern neural network based ML algorithms (Rumelhart et al., 1988). One of the first big result that draw attention to deep networks was AlexNet (Krizhevsky et al., 2012), a deep Convolutional Neural Network (CNN) developed by Alex

Krizhevsky, that won the ImageNet competition, marking a significant breakthrough and initiating a surge of interest in deep learning. AlexNet competed in the ILSVRC (Olga et al., 2015) on September 30, 2012. The deep network achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up.

In recent years, since the growing availability of GPU processing, deep neural networks have been able to achieve outstanding results in many areas related to Machine Learning. In 2014 Ian Goodfellow introduced Generative Adversarial Networks (GANs), a powerful tool for deep generation of data (I. J. Goodfellow et al., 2014); in the same year DL also made strides in reinforcement learning (a sub-field of ML), with models like Deep Q Networks (DQN) (Volodymyr et al., 2015), showing more of the potentiality of deep networks with respect to shallow ones. Another important step forward was made with the introduction of the Transformer architecture, in the 2017 paper "Attention is All You Need" (Vaswani et al., 2017) revolutionized natural language processing (NLP), the interdisciplinary sub-field of computer science and linguistics primarily concerned with giving computers the ability to support and manipulate natural language and speech. This led to an important breakthrough in NLP, and in the whole AI spectrum, when the language model BERT (Bidirectional Encoder Representations from Transformers) came out. Being one of the first language model based on transformers, BERT achieved state-of-the-art results in various NLP tasks thanks to the fact that its the pre-trained model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of applications, such as question answering and language inference, without substantial task-specific architecture modifications. (Devlin et al., 2018).

Geometric Deep Learning

For almost two thousand years the word geometry has been a synonym of Euclidean geometry, until this hegemony was disrupted, during the nineteenth century, when Lobachevsky, Bolyai, Gauss, and Riemann independently constructed instances of non-Euclidean geometries, signaling the end of Euclid's monopoly (Coxeter, 1998). As that century concluded, these inquiries had fragmented into distinct domains. Mathematicians and philosophers engaged in debates regarding the authenticity and interconnections of these geometries, alongside discussions on the essence of the "one true geometry". A wide array of neural network architectures tailored for diverse data types exists, yet few overarching principles tie them together. As in the past, this lack of cohesion complicates comprehension regarding the interplay among different methodologies, consequently leading to redundant rediscovery and re-branding of identical concepts across distinct application domains. For anyone

who is a novice trying to learn the field, absorbing the sheer volume of redundant ideas can be a true nightmare (Bronstein et al., 2021).

Geometric Deep Learning (GDL) was born as a "geometrisation" attempt, with the mindset of bringing order into the chaos of deep learning, obtaining a systematization of this field. The key of this approach is to derive different inductive biases and network architectures implementing them from first principles of symmetry and invariance. GDL is thus an umbrella term for emerging techniques attempting to extend (structured) deep neural models to non-Euclidean spaces, such as graphs and manifolds. Deep learning models have excelled notably in handling signal-based data like speech, images, or video, where an inherent Euclidean structure is present. However, in recent years, there has been an increasing fascination with the endeavor to employ learning approaches for non-Euclidean geometric data, given that such kinds of data arise in numerous real-life applications like social networks and many others (see section 2.1.1 for more examples).

In this section on GDL we propose an overview of prevalent deep learning architectures, even though we acknowledge the continuous emergence of new neural network variants on a daily basis. Therefore, the objective is not to provide an exhaustive coverage of every potential architecture, but we aspire for the forthcoming sections to be sufficiently illustrative. The most common deep learning architectures are thus shown here by utilizing the perspective of invariances and symmetries, that can be a useful lens for an easy categorization of any future GDL developments. There are mainly three kind of architectures that cover the geometric data domains: Convolutional Neural Networks (CNNs), group-equivariant CNNs, and Graph Neural Networks (GNNs).

Convolutional Neural Networks CNNs are perhaps the earliest, and currently among the most successful, deep learning architectures in a variety of tasks, in particular, in computer vision. They are also known as shift invariant or space invariant artificial neural networks (SIANN, Chaman and Dokmanić, 2021), based on their shared-weights architecture and translation invariance characteristics. CNN have a broad range of applications such as in image and video recognition (Valueva et al., 2020), recommender systems (Oord et al., 2013), medical image analysis (Yu et al., 2021), and also natural language processing (Collobert and Weston, 2008), in the beginning of this field. A typical CNN used in computer vision applications consists of multiple convolutional layers, passing the input image through a set of filters followed by point-wise non-linearity ξ (typically, half-rectifiers $\xi(z) = max(0, z)$ or ReLU (Nair and Hinton, 2010) are used, although practitioners have experimented with a diverse range of choices for the activation function), and additional layers, including pooling layers, fully connected layers, and normalization layers.

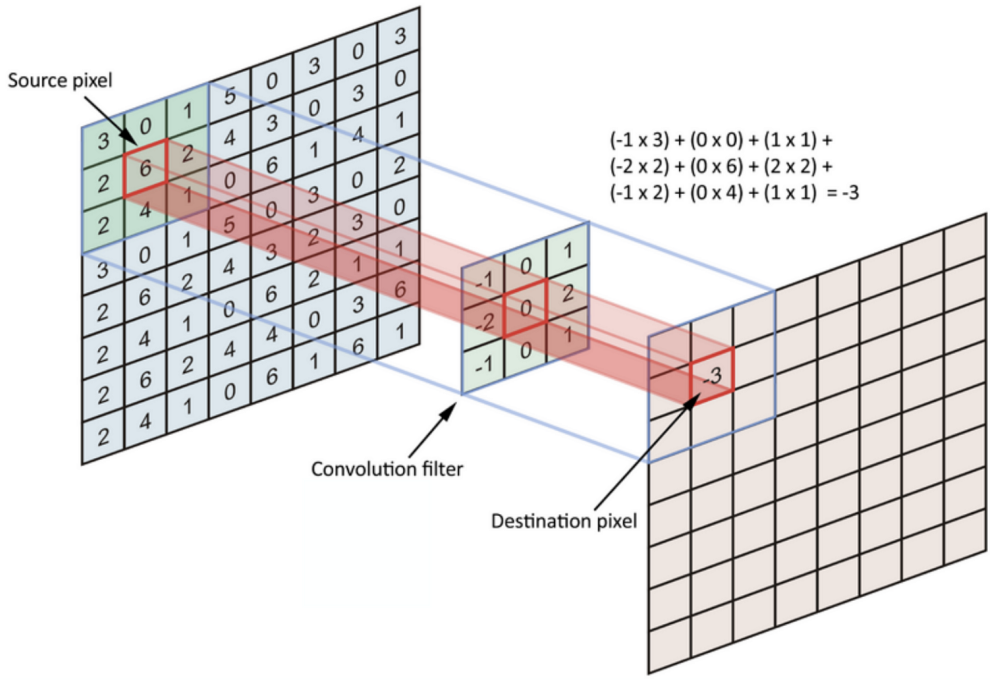


Figure 2.4. *The process of convolution on an input image (8×8), assuming each pixel has an integer p as value ($p \in \{0, 1\}$). In this 2-dimensional case the filter (or kernel) is a 2D matrix (3×3). In the convolution operation the kernel is strided on the input signal to compute each destination pixel of the output. The actual mathematical convolution is done by computing the dot-product between the kernel and a region of the same dimension of the input image, moving the filter all over the input to obtain each of the output components.*

The convolutional layer is the core building block of a CNN, it is defined by its parameters, primarily a collection of learnable filters (or kernels) that possess a relatively small receptive field but span the entire depth of the input volume. During the forward pass, each filter is "convolved" across the width and height of the input volume, computing the dot product between the filter entries and the input, producing a (when dealing with images, 2-dimensional) activation map of that filter (fig. 2.4). Consequently, the network learns filters that trigger upon detecting certain distinctive features at specific spatial positions within the input. Following this, the feature map obtain after the convolution operation can be used as input to another layer of the same type (but with a different kernel) or it can be fed to pooling, to reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer, and/or normalization layers. The combination of convolution, pooling and/or normalization is so common in CNN architectures, that the last two can be considered as a building component of a fully-functioning convolutional layer (LeCun et al., 2015). After one or many convolutional steps, in a typical CNN, there are some fully connected layers arranged

in a multi-layer perceptron (MLP) fashion, that aim to classify the encoded deep representation of the input data obtained with the convolution.

Group equivariant CNN Cohen and Welling, 2016 showed that it is possible generalize the convolution operation from signals on a Euclidean space to signals on any homogeneous space Ω acted upon by a group G, i.e. convolutional networks can be generalized to exploit larger groups of symmetries, including rotations and reflections. By analogy to the Euclidean convolution where a translated filter is matched with the signal, the idea of group convolution is to move the filter around the domain using the group action, e.g. by rotating and translating. By virtue of the transitivity of the group action, we can move the filter to any position on Ω.

Convolutional layers are highly efficient within a deep network due to the translation equivariance present in all its layers. This property implies that shifting an image and subsequently passing it through a series of layers is equivalent to initially processing the original image through the same layers and then shifting the resulting feature maps (while considering edge-effects). Essentially, each layer maintains the symmetry of translation, enabling its utilization not only in the initial layers but also in the higher layers of the network. The key concept behind Group-equivariant CNNs (G-CNN) is that convolutional networks can be generalized to exploit larger groups of symmetries, including rotations and reflections. This signifies that within the representation space, every vector possesses an associated pose that can be altered by the elements within a specific transformation group, denoted as G. This supplementary structure enhances our ability to model data with greater efficiency. In a G-CNN, a filter identifies co-occurrences of features exhibiting the desired relative pose and can recognize such a feature configuration in any global pose using an operation known as the G-convolution. For further details on this type of deep networks we suggest the following material, the already cited (Cohen and Welling, 2016) and (Bronstein et al., 2021).

Representation Learning Another interesting research are of Deep Learning is Representation Learning, which is the task of learning representations for complex structured data. In the last decade, many successful models have been developed structured data, defined on a discretized Euclidean domain. For instance, sequential data, such as text or videos, can be modeled via recurrent neural networks (Tealab, 2018), which can capture sequential information, yielding efficient representations as measured on machine translation and speech recognition tasks. Another example are Convolutional Neural Networks (CNNs)(Lecun et al., 1998) have been successfully applied to tackle problems such as image classification (Venkatesan and Li, 2017), semantic segmentation (Jégou et al., 2017) or machine translation (Gehring et al.,

2017), where the underlying data representation has a grid-like structure. These architectures efficiently reuse their local filters, with learnable parameters, by applying them to all the input positions (LeCun et al., 2015). However these major successes have been restricted to particular types of data that have a simple relational structure (e.g. sequential data, or data following regular patterns), many interesting tasks involve data that can not be represented in a grid-like structure and that instead lies in an irregular domain. This is the case of a large number of systems across various areas including natural science (physical systems (Sanchez-Gonzalez et al., 2018), social science (social networks as in the example in section 2.1.1), protein-protein interaction networks (Fout et al., 2017), knowledge graphs (Hamaguchi et al., 2017) and many other research areas (Khalil et al., 2017), to name a few. Such data can usually be represented in the form of graphs (Petar et al., 2018).

2.1.3 GNNs

Graph Neural Networks (GNNs) have been introduced in (Gori et al., 2005; Scarselli et al., 2009) as a generalization of recursive neural networks that can directly deal with the more general class of graphs and their impressive performance has propelled these deep learning models to the forefront of widely utilized methods in graph analysis today. The fundamental principle behind GNNs is to produce representations of graph (and/or its components) that can be used in downstream tasks such as graph or node classification. GNNs are among the most general class of deep learning architectures currently in existence, and it is a fact, most other deep learning architectures can be understood as a special case of the GNN with additional geometric structure. It is important to remind the reader that GNNs are not the exclusive tools for modeling graph-structured data; for example graph kernels (Vishwanathan et al., 2010) and random-walk methods (Grover and Leskovec, 2016; Perozzi et al., 2014) were previously among the most prevalent. However, in current times, GNNs have predominantly supplanted these approaches due to their inherent adaptability, allowing for superior modeling of the underlying systems.

The first motivation of GNNs is rooted in the long-standing history of neural networks for graphs. In the nineties, Recursive Neural Networks are first utilized on directed acyclic graphs (Frasconi et al., 1998; Sperduti and Starita, 1997). Afterwards, Recurrent Neural Networks and Feed-forward Neural Networks are introduced into this literature respectively in (Scarselli et al., 2009) and (Micheli, 2009) to tackle cycles. Although being successful, the universal idea behind these methods is building state transition systems on graphs and iterate until convergence, which constrained the extendability and representation ability. Recent advancement of deep neural networks, especially convolutional neural networks (CNNs) resulted

in the rediscovery of GNNs.

A Graph Neural Network (GNN) is a transformation that can be optimized across all attributes of a graph, including nodes, edges, and global-context, while preserving the symmetries of the graph (permutation invariances). In this section a basic approach to constructing GNNs is introduced, following the “message passing neural network” framework as proposed in (Justin et al., 2017), and employing the architectural schematics of Graph Nets introduced by (Battaglia et al., 2018). GNNs follow a "graph-in, graph-out" architecture, meaning they take a graph as input, with information embedded in its nodes, edges, and global-context. They then progressively enrich these embeddings without altering the connectivity of the initial graph.

Having introduced the basic concepts around graphs, to better better comprehend their success we present an overview of the principal tasks that can be performed through graph representation. As a unique non-Euclidean data structure for machine learning, graph analysis focuses on tasks such as graph classification, node classification, link prediction, and clustering. Despite this variety, there are three general types:

- **Graph-level** In a graph-level task, our goal is to predict the property of an entire graph. For instance, when dealing with a molecular representation structured as a graph, a common objective could be predicting the fragrance of the molecule or its likelihood to bind to a receptor associated with a particular disease. This can be likened to image classification tasks like MNIST (LeCun et al., 1998) and CIFAR (Krizhevsky and et al., 2009), where the goal is to assign a label to an entire image. In the realm of text analysis, a comparable task is sentiment analysis, where the aim is to determine the overall mood or emotion of an entire sentence in one go.

- **Node-level** For a node-level task, we predict some property for each node in a graph. An example could be a GNN used for a node prediction task on the Cora dataset (McCallum et al., 2000) to predict the subject of a paper given its words and citations network. The Cora dataset consists of 2708 scientific publications classified into one of seven classes, resulting in a citation network composed of 5429 links. Each article in the network is represented by a node, that can be classified into one of the seven classes available (i.e. the subjects of the each paper).

- **Edge-level** For an edge-level task, we want to predict the property or presence of edges in a graph. An instance of edge-level inference can be observed in image scene comprehension (fig. 2.5). Beyond merely recognizing objects within

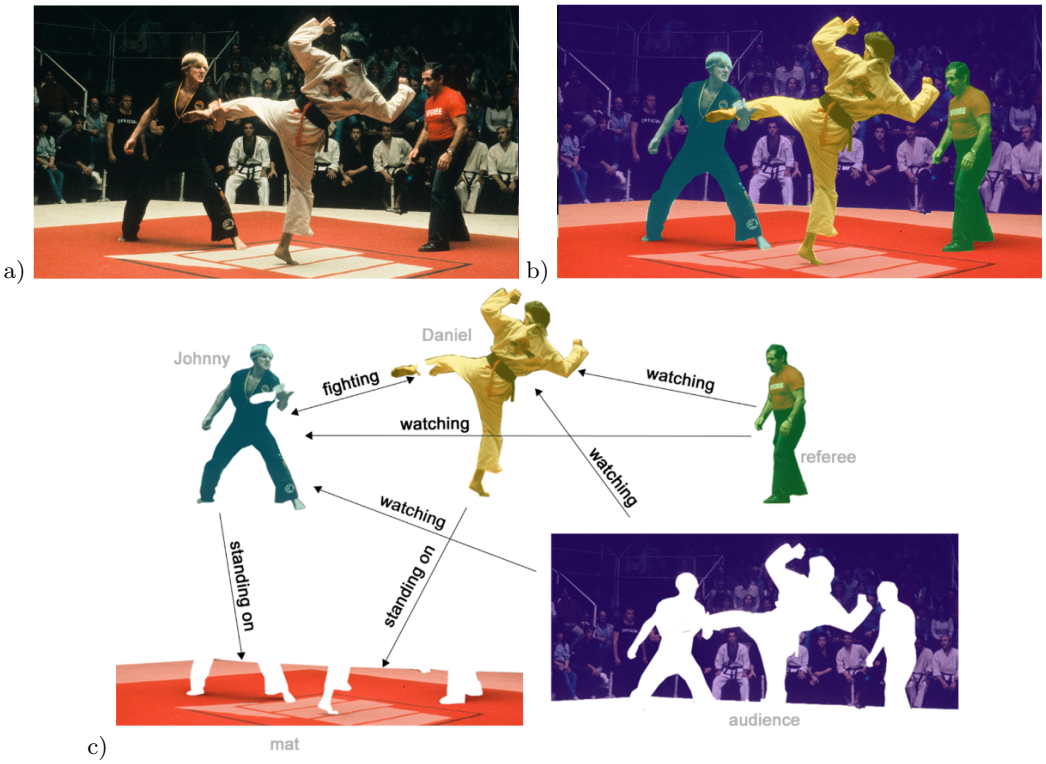


Figure 2.5. *An example of a scene understanding task. In (fig.b), above, the original image (fig.a) has been segmented into five entities: each of the fighters, the referee, the audience and the mat. (fig.c) shows the relationships between these entities as edges of a graph connecting the elements (nodes) in the scene. Note that in this example the extracted representation is a directed-graph with three edges attributes, the actions performed by the entities in the scene (fighting, watching, standing on). Image courtesy of (Sanchez-Lengeling et al., 2021).*

an image, deep learning models can also forecast the associations between these objects. This can be framed as edge-level classification: given nodes representing the image's objects, our goal is to anticipate which nodes are linked by an edge and determine the value of that edge. If our objective is to uncover relationships between entities, we might envision the graph as fully interconnected. From there, we can prune edges based on their predicted values to derive a sparser graph that highlights meaningful connections.

The three levels of prediction problems described above (graph-level, node-level, and edge-level), can be solved with the Graph Neural Network model class.

Pooling The following example are going to implicitly focus on binary classification, although this framework can be seamlessly expanded to handle multi-class or regression scenarios. When the objective is binary prediction for nodes and the graph already includes node information, the process is uncomplicated: for each

node embedding, utilize a linear classifier (Daigavane et al., 2021).

However, it is not always so simple. For instance, you might have information in the graph stored in edges, but no information in nodes, but still need to make predictions on nodes. We need a way to collect information from edges and give them to nodes for prediction. We can do this by pooling. Pooling proceeds in two steps: First for each item to be pooled, gather each of their embeddings and concatenate them into a matrix; second the gathered embeddings are then aggregated, usually via a sum operation. If we only have node-level features, and need to predict a binary global property, we need to gather all available node information together and aggregate them. This is similar to Global Average Pooling layers in CNNs. The same can be done for edges. Having successfully showcased the creation of a basic GNN model and its ability to make binary predictions by transmitting information between various graph segments, this pooling technique should be kept in mind as a fundamental component for crafting more advanced GNN models.

Message Passing

We could make more sophisticated predictions by using pooling within the GNN layer, in order to make our learned embeddings aware of graph connectivity. We can do this using message passing, where neighboring nodes or edges exchange information and influence each other's updated embeddings (Justin et al., 2017).

Message-passing on GNNs operates through a three-step process: *i*) for every node within the graph, it collects all the embeddings from neighboring nodes; *ii*) it aggregates all these messages, typically using a function like summation; *iii*) the combined messages are then passed through an update function, typically implemented as a learned neural network. Just as pooling can be applied to either nodes or edges, message passing can occur between either nodes or edges. When executed in a single iteration, this sequence of operations constitutes the most basic form of a message-passing GNN layer. This bears a resemblance to a standard convolution in CNNs (fig. 2.4): fundamentally, both message passing and convolution involve the aggregation and processing of information from neighboring elements to update the value of the element. In the context of graphs, this element is a node, while in images, it corresponds to a pixel. Nonetheless, it's worth noting that the number of neighboring nodes in a graph can vary, unlike in an image where each pixel has a fixed number of neighboring elements. The key achievement of this formulation is that stacking multiple message-passing GNN layers consecutively, a node can eventually assimilate information from the entire graph. At every step, the resulting representation of a node in the graph is influenced by the its neighbors, i.e. the nodes at one-edge distance. This progressively results in gathering information

from nodes one-edge further at each layer of the GNN (i.e. each message-passing iteration), so that after two layers, a node has information about the nodes two hops (edges) away from it. It is trivial to notice that with a sufficient number of iteration, this mechanism will cover the whole graph, making the embedding of a node aware of the entire topology of the graph. Lastly, following the notation introduced in section 2.1.1, if we consider a graph G with a feature matrix X with size $n \times d$ (being $n = |V|$ and d the number of node features for each node), whose topology is encoded by adjacency matrix $A \in \{0,1\}^{n \times n}$, a two-layer Graph Convolutional Network (GCN, Kipf and Welling, 2016) f acting on G is generally defined as:

$$f_\theta(G) = f_\theta(X, A) = \text{softmax}(D\phi(DX\theta_1)\theta_2) \tag{2.2}$$

where ϕ is an element-wise nonlinearity (such as the ReLU $\phi(\cdot) = \max(0, \cdot)$), D is a diffusion operator like the normalized Laplacian or any appropriate shift operator defined on the graph.

GNNs architectures

The design and study of GNN layers is one of the most active areas of deep learning at the time of writing, making it a landscape that is challenging to navigate. Fortunately, we find that the vast majority of the literature may be derived from only three "flavours" of GNN layers (Bronstein et al., 2021), which we will present here. In all three flavours, permutation invariance is ensured by aggregating features from X_{N_u} (potentially transformed, by means of some function ψ) with some permutation-invariant function $\oplus$, and then updating the features of node u, by means of some function function ϕ. Typically, ϕ and ψ are the learnable, features; with e.g. whereas $\oplus$ is realized as a non-parametric operation such as recurrent neural networks (Murphy et al., 2019). These flavours govern the extent to which ϕ transforms the neighbourhood features, allowing for varying degrees of complexity when modeling interactions across the graph.

Convolutional flavour In the convolutional flavour (Defferrard et al., 2017; Kipf and Welling, 2016; F. Wu et al., 2019), the features of the neighbourhood nodes are directly aggregated with fixed weights

$$h_u = \phi\left(X_u, \bigoplus_{v \in N_u} C_{uv}\psi(X_v)\right) \tag{2.3}$$

Here, C_{uv} specifies the importance of node v to node u's representation. It is a constant that often directly depends on the entries in A representing the structure of

the graph. Note that when the aggregator operator $\oplus$ is chose to be the summation, it can be considered as linear diffusion of position-dependent linear filtering, a generalization of convolution. Even though most of the GNNs used in practice are based on convolution (and thus graph convolutional networks, GCNs), this definition covers most of such approaches, in the sense of commuting with the graph structure, but not all of them.

Attentional falvour The attentional approach to GNNs is one of the most recent and most interesting. Attention mechanisms have become really popular, almost a *de facto* standard, in many sequence-based tasks (Bahdanau et al., 2015; Gehring et al., 2016). The key advantage that attention mechanisms offer is handling inputs with varying sizes, focusing on the most relevant parts of the input to facilitate informed decision-making. The attentional "flavour" is characterized by implicit interactions (Monti et al., 2016; Petar et al., 2018; J. Zhang et al., 2018),

$$h_u = \phi\left(X_u, \bigoplus_{v \in N_u} a(X_u, X_v)\psi(X_v)\right) \tag{2.4}$$

Here, a is a learnable self-attention mechanism that computes the importance coefficients $\alpha_{uv} = a(X_u, X_v)$ implicitly. They are often softmax-normalized across all neighbors. When $\oplus$ is the summation, the aggregation is still a combination of the neighborhood features, but now the weights are feature-dependent.

Message-passing flavour Finally, another really common approach to GNNs, the message-passing "flavour" (Battaglia et al., 2018; Justin et al., 2017) amounts to computing arbitrary vectors ("messages") between adjacent nodes across the edges connecting them, simulating an exchange of information between the graph components,

$$h_u = \phi\left(X_u, \bigoplus_{v \in N_u} \psi(X_u, X_v)\right) \tag{2.5}$$

Here, ψ is a learnable message function, computing v's vector sent to u, and the aggregation can be considered as a form of message passing on the graph. One important thing to note is a representational containment between these approaches: *convolution* $\subseteq$ *attention* $\subseteq$ *message-passing*. Indeed, attentional GNNs can represent convolutional GNNs by an attention mechanism implemented as a look-up table $a(X_u, X_v) = C_{uv}$, and both convolutional and attentional GNNs are special cases of message-passing where the messages are only the sender nodes features: $\psi(X_u, X_v) = C_{uv}\psi(X_v)$ for convolutional and $\psi(X_u, X_v) = a(X_u, X_v)\psi(X_v)$ for attentional GNNs.

2.2 eXplainable Artificial Intelligence (XAI)

Explainable Artificial Intelligence (XAI) stands at the forefront of the evolving field of artificial intelligence (AI) and seeks to bridge the gap between advanced AI models and human comprehension. As AI systems become increasingly sophisticated and pervasive in our lives, understanding their decision-making processes and outcomes is crucial for fostering trust, transparency, and usability. XAI aims to unravel the 'black box' nature of complex AI algorithms, providing insights into how these models arrive at specific conclusions or predictions.

The realm of artificial intelligence (AI) is extensive, intricate, and in a perpetual state of evolution. With the progress in computational capabilities and the ever-expanding pool of data, AI algorithms are being extensively researched and developed for diverse applications, catering to a wide array of potential users and associated risks. Within the AI community, there is a concerted effort to prioritize explainability as a fundamental trait for building trustworthy AI systems. Collaborating with this community, NIST (National Institute of Standards and Technology) has identified additional technical attributes essential for fostering trust in AI (Phillips et al., 2021). In addition to explainability and interpretability, the two big branches of the explaining ML research area, various other characteristics are proposed within AI systems to enhance their trustworthiness. These include accuracy, privacy, reliability, robustness, safety, security (resilience), mitigation of harmful bias, transparency, fairness, and accountability. The interaction of explainability and these AI system attributes may occurs at different stages throughout the AI life-cycle, but this work is focused mainly on the explainability.

NIST identified four main principle, that are significantly shaped by taking into account how the AI system interacts with the human recipient of the information. The specific demands of the situation, the task in progress, and the end-user all play a role in determining the suitable type of explanation for the circumstance. A more precise definitions for three fundamental terms in this definition, *explanation*, *output*, and *process*, should be established before introducing and discussing the principles. An *explanation* refers to the evidence, support, or reasoning associated with a system's output or process. The *output* of a system is defined as either i) the result or ii) the action executed by a machine or system while performing a task. It's important to note that the system's output varies based on the specific task (e.g. for a loan application, the output is a decision: approved or denied; for a movie recommendation system, the output could be a list of recommended movies). The term *process* encompasses the procedures, design, and overall workflow of the system (as referenced in Leslie and Briggs, 2021). This encompasses the system's documentation, details about the data employed during system development or

stored data, and relevant knowledge pertaining to the system. Briefly, according to the NIST definition (Phillips et al., 2021), the four principles of explainable AI are:

- **Explanation** A system provides or includes supporting evidence or rationale for its outputs and/or processes. In its essence, the principle of explanation is detached from whether the explanation is accurate, informative, or understandable, and it does not enforce any quality metric on these explanations. These factors are components of the meaningfulness and explanation accuracy.
- **Meanignful** A system offers explanations that are clear and comprehensible to the designated audience. There are commonalities across explanations which can make them more meaningful. For example, stating why the system did behave a certain way can be more understandable than describing why it did not behave a certain other (Lim et al., 2009).
- **Accuracy (of the explanation)** An explanation accurately represents the cause behind generating the output and/or faithfully mirrors the system's procedure. Accuracy in explanation is a separate concept from decision accuracy. Decision accuracy pertains to whether the system's judgment is right or wrong. Irrespective of the system's decision accuracy, the related explanation may or may not precisely elucidate the system's rationale for its conclusion or action.
- **Knowledge limits** A system functions exclusively within the conditions for which it was designed and proceeds only when it attains a satisfactory level of confidence in its output. By recognizing and articulating the bounds of its knowledge, this approach ensures that a judgment is not given when it might be unfitting. This principle can enhance trust in a system by averting misleading, hazardous, or unjust outputs.

In this introduction, the differences between the backbone concepts of interpretability and explainability are explored, and is shown a brief introduction on the most common XAI method for Machine Learning, namely LIME (Ribeiro et al., 2016) and SHAP (S. Lundberg and Lee, 2017). Understanding XAI not only empowers AI practitioners to design more transparent and accountable models but also enables users, stakeholders, and decision-makers to make informed judgments and establish a harmonious coexistence with AI technologies.

2.2.1 Explainability vs Interpretability

The rise of interpretable and explainable ML methods stems from the necessity to create machine learning systems that are understandable to the human mind, more critically to the likely non-expert users that will end up using those systems on a

daily. It also addresses the challenge of comprehending and justifying predictions made by complex models like deep neural networks or gradient boosting machines (Friedman, 2001; Mason et al., 1999). Early research on interpretable machine learning traces back to the 1990s (Rudin, 2019), often without explicit reference to terms like "interpretability" or "explainability". Additionally, many traditional statistical models are inherently interpretable, like Decision Trees, that offers an output that his human intelligible, despite their simplicity.

These concepts are often use interchangeably. In certain studies, the terms "interpretable" and "explainable" are used interchangeably (Molnar, 2022), while in others, subtle distinctions are acknowledged. Within this research, a clear differentiation between these terms is recognized, aligning with prior work (Rudin, 2019). Generally ML model can be defined as "interpretable" if it can, independently, offer humanly understandable interpretations of its own predictions. It's important to note that such a model, to some extent, should not be considered entirely a black box. For instance, a decision tree model is considered "interpretable". Conversely, an "explainable" model suggests that the inner functioning is obscure (i.e. a black box), but there exists potential for understanding its predictions through *post-hoc* explanation techniques.

It is to be noted that in the work presented in this text, explainability and interpretability are considered as two different things, as initially proposed by Rudin, 2019. Z. C. Lipton, 2017 try to summarize the difference in the research questions the two groups of techniques try to answer: interpretability, they claim, raises the question "How does the model work?"; whereas explanation methods try to answer "What else can the model tell me?" (Marcinkevičs and Vogt, 2023).

Interpretability Generally, the ML community lacks consensus regarding the precise definitions of interpretability and the specific task of interpretation (Doshi-Velez and Kim, 2017). For example, Doshi-Velez and Kim define interpretability of ML systems as "the ability to explain or to present in understandable terms to a human" (Z. C. Lipton, 2017); although this definition clearly lacks mathematical rigor. Nevertheless, the notion of interpretability often depends on the domain of application and the target *explainee* (i.e. the model to be explained), i.e. the recipient of interpretations and explanations, therefore, an all-purpose definition might be infeasible or unnecessary. Other terms that are synonymous to interpretability and also appear in the ML literature are "intelligibility" (Caruana et al., 2015; Vilone and Longo, 2021) and "understandability" (Z. C. Lipton, 2017).

Explainability Yet another term prevalent in the literature is "explainability", giving rise to the direction of explainable artificial intelligence (XAI) (Turek et al.,

2021). This concept is closely tied with interpretability; and many authors do not differentiate between the two (Carvalho et al., 2019). Doshi-Velez and Kim, 2017 provide a definition of explanation that originates from psychology: "explanations are... the currency in which we exchange beliefs". Rudin, 2019 draws a clear line between interpretable and explainable ML: interpretable ML focuses on designing models that are inherently interpretable; whereas explainable ML tries to provide *post-hoc* explanations for existing black box models, i.e. models that are inherently incomprehensible to human beings or that are proprietary, i.e. those AI models owned by company which are not accessible to the public, and thus behave as black boxes even in the case they are interpretable by design (e.g. a very well know example of proprietary model is the transformer based Large Language Model (LLM) ChatGPT, owned by OpenAI, (Brown et al., 2020)).

On the other hand, an "explainable" model suggests that the model itself remains a black box, and understanding its predictions is facilitated through *post-hoc* explanation techniques. We emphasize this distinction, aspiring for a standardized use of terminology as this field advances.

2.2.2 LIME and SHAP

Several approaches have been proposed as XAI methods dealing with a variety of data and model types, aiming to locally and/or globally explain the models outputs, from which SHAP (S. Lundberg and Lee, 2017) and LIME (Ribeiro et al., 2016) form the most popular XAI methods. Both these techniques lies inside the explainability category, given that both approaches act *a-posteriori* trying to produce a justification (the explanation) of the model output that can give useful insights on the model behavior and the "motivations" behind a specific prediction.

Local Interpretable Model-agnostic Explanations (LIME) Local Interpretable Model-agnostic Explanations (LIME) serves as a model-agnostic technique for local explanations (Ribeiro et al., 2016). It elucidates how each feature influences the outcome for a specific instance. In the case of classification models, it provides the probability of the instance belonging to each class. Moreover, it presents the contribution of each feature to each class through visualized plots. However, LIME transforms any model into a local linear model, revealing coefficient values that symbolize feature weights in the model. In simpler terms, if the user applies models that consider the non-linearity between features and the outcome, this aspect may not be fully captured in the explanation generated by LIME. This is because the non-linearity is lost in the linear model generated by LIME. In addition, LIME is a model-dependent method, meaning the used model will affect the outcome of

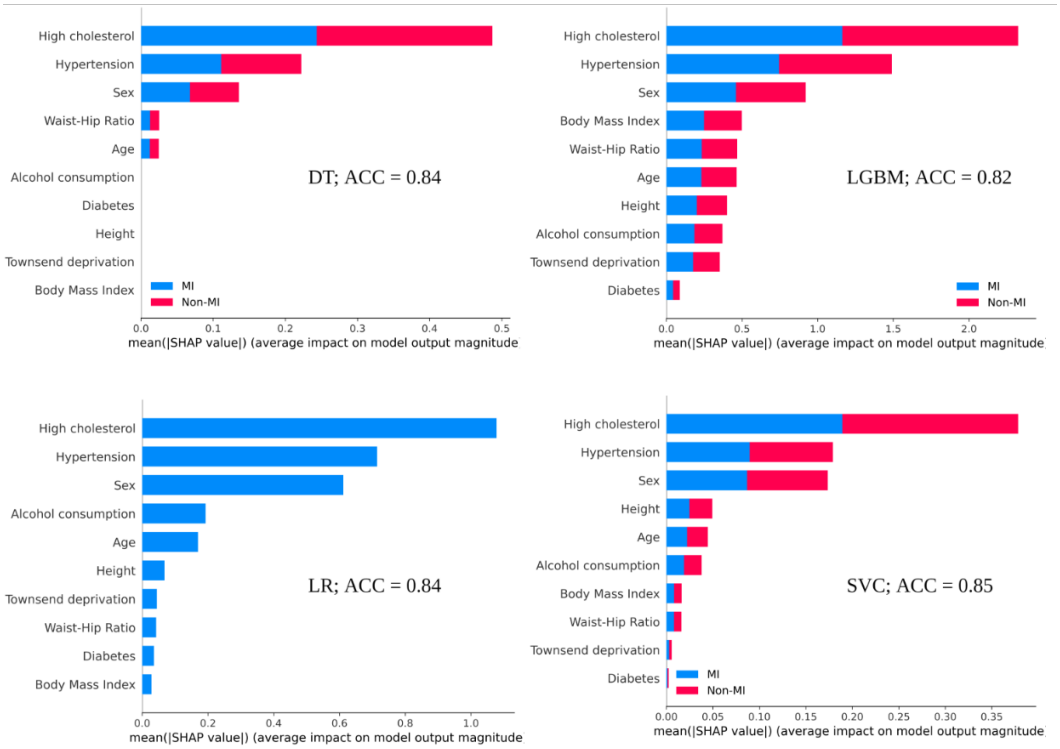


Figure 2.6. *SHAP output to explain four models globally. Models were applied to 1500 subjects (80%/20% train/test split) to classify them as cases (myocardial infarction, MI) or controls (No MI). DT: decision tree; LGBM: light gradient-boosting machine; LR: logistic regression; SVC: support vector machines classifier; ACC: accuracy; MI: myocardial infarction.*

LIME (i.e. the outcome of LIME is strictly related to the output of the model to be explained).

SHapely Additive exPlanation (SHAP) SHapely Additive exPlanation (SHAP) stands as a post-hoc, model-agnostic technique applicable to any machine learning model S. Lundberg and Lee, 2017. Grounded in game theory, it assesses the contribution of each "player" to the overall "payout". In the context of machine learning models, these "players" and the "payout" correspond to features and the model outcome, respectively. SHAP computes a score for each feature in the model, denoting its influence on the model's output. The calculation involves evaluating all possible combinations of features to cover scenarios where all features or a subset of features are in the model (fig. 2.6). As the number of features escalates, SHAP's computational complexity increases. To address this, an approximate SHAP method, Kernel SHAP, has been proposed. SHAP finds extensive applications across diverse domains to elucidate model outputs, whether on a local or global scale (García and Aznarte, 2020; KIM and Kim, 2022; Ullah et al., 2023). Nonetheless, researchers

and end-users should take note of certain aspects when employing SHAP. Primarily, SHAP is contingent on the specific model in use, i.e. model-dependent like LIME. This implies that the results obtained from SHAP can vary based on the model utilized, potentially resulting in differing explainability scores when employing different machine learning models.

2.2.3 Explainability in GNNs

After an overview of the interpretability/explainability sub-field of Machine Learning, a more detailed look on explainability techniques on graph neural network is presented, endorsing the main topic on which this text is focused on. Such an in-depth analysis will be needed regardless, given that the two explainability frameworks introduced in the previous section (section 2.2.2) have not been designed for graph data or deep models for graph.

In recent times, the popularity of Graph Neural Networks (GNNs) has surged due to the prevalent representation of real-world data in graph formats. This includes data from various domains like social networks, chemical molecules, and financial data (see section 2.1.2 for more examples). Several classification tasks on graph data tasks are widely studied, such as node classification (Gao and Ji, 2019; Henaff et al., 2015), graph classification (Xu et al., 2019; M. Zhang et al., 2018), link predictions (Cai and Ji, 2020; M. Zhang and Chen, 2018). Furthermore, numerous sophisticated operations for Graph Neural Networks (GNNs) have been introduced to enhance their performance. These encompass operations like graph convolution Kipf and Welling, 2016, graph attention Petar et al., 2018, and graph pooling Yuan and Ji, 2020, that increase the overall complexity and illegibility.

However, compared with more common domains in deep learning, like images and text, the explainability of graph models is relatively less explored, critically damaging the overall understanding of deep graph neural networks. Recently, following the growing popularity of GNNs, several approaches have been proposed to explain their predictions. To name the most successful (Xie et al., 2022), XGNN (Yuan, Tang, et al., 2020), GNNExplainer (Ying et al., 2019), PGExplainer (Luo et al., 2020), and SubgraphX (Yuan et al., 2021), etc. These approaches have emerged from diverse perspectives, offering varying levels of explanations. Moreover, there is a current deficiency of standardized datasets and metrics to evaluate the results of explanations (Hao et al., 2022).

To better understand these methods, the taxonomy of different explanation techniques for GNNs introduced in (Hao et al., 2022) can be followed. Based on what types of explanations are provided in output, the different methods can be classified into two main groups: instance-level methods and model-level methods.

Overall, these two categories elucidate deep graph models from distinct perspectives. Instance-level methods furnish explanations specific to each example, whereas model-level methods offer overarching insights and a more general understanding. To validate and trust in most deep graph models, human oversight is still essential to review the explanations and give the correct interpretation of the explanation provided with those methods.

Instance-level methods According to the criterion used to obtain the explanations, Instance-level techniques can be categorized into four different branches: gradients/features-based methods, perturbation-based methods, decomposition methods, and surrogate methods.

- **Gradient/feature based methods** Utilizing gradients or features for explaining deep models is a common and direct approach, extensively applied in image and text tasks. The fundamental concept involves using gradients or values from hidden feature maps as approximations of input importance. In gradients-based methods (Simonyan et al., 2014; Smilkov et al., 2017), the gradients of the target prediction concerning input features are computed through back-propagation. Conversely, in features-based methods (Selvaraju et al., 2019; B. Zhou et al., 2015), hidden features are mapped back to the input space using interpolation to gauge importance scores. Given their simplicity and versatility, these methods can be straightforwardly extended to the graph domain. In recent times, numerous techniques have been employed to elucidate graph neural networks, such as SA, Guided BP (Baldassarre and Azizpour, 2019), CAM and Grad-CAM (Pope et al., 2019). The principal distinction among these methods lies in the gradient backpropagation process and the manner in which diverse hidden feature maps are amalgamated.

- **Perturbation-based methods** These techniques find extensive use in explaining deep image models. The fundamental idea is to analyze how the outputs vary in response to diverse input perturbations. Essentially, if crucial input information is preserved, the predictions should remain akin to the original predictions. Current methods (Chen et al., 2018; Dabkowski and Gal, 2017; Yuan, Cai, et al., 2020), involve training a generator to produce a mask that selects significant input pixels to expound deep image models. However, applying such techniques directly to graph models is not feasible. Unlike images, graphs are comprised of nodes and edges, and altering their dimensions to maintain uniform node and edge numbers is not possible. Generally perturbation-based methods are employed to derive masks from the input graph, highlighting crucial input features. It's important to note that varying types of masks are

generated based on the explanation tasks, including node masks, edge masks, and node feature masks. Subsequently, the generated masks are integrated with the input graph, resulting in a modified graph that encapsulates essential input information. Finally, this updated graph is inputted into the trained GNNs for evaluating the masks and refining the mask generation algorithms. Many perturbation-based methods have been proposed over the last years, including GNNExplainer (Ying et al., 2019), ZORRO (Funke et al., 2021), GraphMask (Schlichtkrull et al., 2022), Causal Screening (X. Wang et al., 2021), and SubgraphX (Yuan et al., 2021).

- **Surrogate methods** The fundamental concept involves utilizing a straightforward and easy-to-understand surrogate model to approximate the predictions made by the intricate deep model for neighboring regions surrounding the input example. It's worth noting that these approaches operate under the assumption that relationships within the neighboring areas of the input example are less intricate and can be effectively represented by a simpler surrogate model. The explanations derived from this interpretable surrogate model are subsequently employed to elucidate the initial prediction. However, extending these surrogate methods to the graph domain presents challenges due to the discrete nature of graph data and the inclusion of topology information. In order to elucidate the prediction made for a particular input graph, the process begins by acquiring a local dataset that encompasses numerous neighboring data objects alongside their respective predictions. Subsequently, an interpretable model is trained on this local dataset to grasp the underlying patterns. The explanations derived from this interpretable model are then considered as explanations for the original model's prediction concerning the input graph. Some surrogate methods have been proposed recently to explain deep graph models, including GraphLime (Q. Huang et al., 2020), RelEx (Y. Zhang et al., 2021), and PGM-Explainer (Vu and Thai, 2020).

- **Decomposition methods** Another prevalent approach for explaining deep image classifiers involves decomposition methods. These techniques quantify the significance of input features by breaking down the original model predictions into distinct terms. These terms are subsequently considered as the importance scores corresponding to the respective input features. In this approach, a direct analysis of the model parameters is conducted to unveil the relationships between features in the input space and the resulting output predictions. It's important to emphasize that the conservative nature of these methods mandates that the sum of the decomposed terms equals the original prediction score. However, it is challenging to adapt such methods to the

graph domain since graphs contain many components (e.g. nodes, edges, and node features), and it is non-trivial to distribute scores to different edges that may contain important structural information that should not be ignored. In these methodologies, the prediction score is propagated layer by layer using a backpropagation approach, extending all the way to the input layer. The process commences from the output layer, where the model's prediction serves as the initial target score. This score is then deconstructed and allocated to neurons in the preceding layer according to predefined decomposition rules. Through iteratively applying these steps until reaching the input space, significance scores for node features are acquired. These scores can be aggregated to signify edge importance, node importance, and walk importance. It's important to highlight that within these algorithms, the activation functions within deep graph models are overlooked. Common implementation of decomposition methods proposed to explain the deep graph neural networks include Layer-wise Relevance Propagation (LRP) (Baldassarre and Azizpour, 2019), (Schwarzenberg et al., 2019), Excitation BP (Pope et al., 2019) and GNN-LRP (Schnake et al., 2022).

Model-level methods Diverging from instance-level methods, model-level methods strive to offer broader insights and a higher-level comprehension to elucidate deep graph models. In particular, they investigate the input graph patterns that influence specific behaviors in GNNs, such as optimizing for a particular prediction target. An established avenue within this realm is input optimization (Olah et al., 2017), which is widely used to derive model-level explanations for image classifiers. Although generalizing the explanations to a model-level perspective is a quite important and interesting improvement, at the time of writing this text only few works have tried to tackle this challenge. The few existing model-level method for explaining graph neural networks are XGNN (Yuan, Tang, et al., 2020), PGExplainer (Luo et al., 2020) which can be considered with perturbation-based methods (section 2.2.3), and GCFExplainer (Z. Huang et al., 2023) which employs counterfactual reasoning.

2.2.4 Contrastive and counterfactual reasoning

The pursuit of human-centric AI has prompted extensive research into the essence of explanation. Nevertheless, a consensus regarding the definition of explanation remains elusive, even though explanation has garnered considerable attention, particularly within fields such as the philosophy of science (Pitt, 1989; Schurz, 2000). In the context of AI, the broadest form of explanation typically relies on assessments of why an AI algorithm predicts a specific outcome and conjectures regarding the

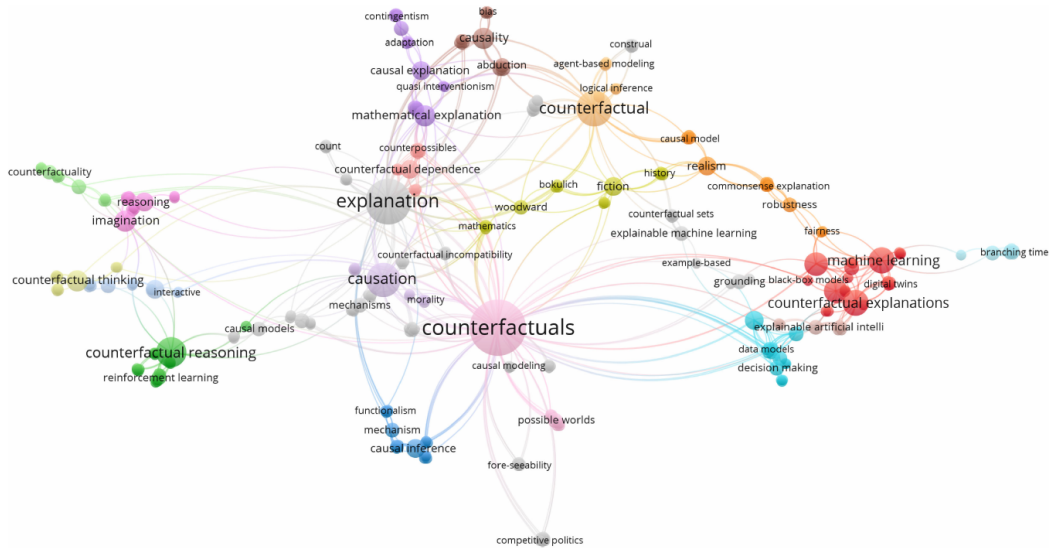


Figure 2.7. *Graph of countefactual XAI publications in literature from the Scopus*[1] *and Web of Science (WoS*[2]*) databases. The graph contains the most popular author keywords for counterfactual explanation (Stepin et al., 2021). This topic is often investigated in the context of causation (pay attention to such keywords as "causation", "causal inference" or "causal models") as well as cognitive science (as reflected by the keywords "imagination", "reasoning", etc.) and AI ("machine learning", "data models", "black-box models").*

causes in relation to the observed effects (Lombrozo, 2012). The necessity to produce explanations resembling those "generated" by humans has captured the interest of AI researchers, focusing on specific aspects of explanation and its sub-categories (Miller, 2019). Consequently, elucidating an algorithm's output in the context of plausible yet uncommon alternatives proves to be notably challenging, considering the potentially limitless array of such possibilities. Moreover, this complexity is compounded by the capacity to propose pertinent modifications to the input, resulting in a divergent decision by the algorithm, w.r.t. the original one.

With a growing focus on these forms of explanation, specifically referred to as *contrastive* and *counterfactual* within the XAI community, it becomes significantly important to examine the current theoretical explanations frameworks following this approach for their automated generation (Stepin et al., 2021). The interest here is even greater, considering that on such kind of XAI technique the research work presented in this text is based on. Thus, the aim of this section is to analyze relevant theoretical work on the contrastive and counterfactual accounts of explanation, and to discuss a degree of synergy between the revised theories and their related up-to-date implementations.

[1] https://www.elsevier.com/en-gb/solutions/scopus
[2] https://clarivate.libguides.com/webofscienceplatform/coverage

It's crucial to emphasize that contrastive explanations highlight the distinction between the actual decision and a hypothetical one. Conversely, counterfactual explanations outline the essential minimal alterations in the input required to produce a contrasting output. However, within the domain of XAI, these terms are at times used interchangeably (Le et al., 2020; Poyiadzi et al., 2020). Diverse categories of techniques have been put forth to create both contrastive and counterfactual explanations for AI algorithm outputs. In the realm of XAI, when regarding an explanation for an automated decision or prediction as an observation, it can be abductively derived by engaging in a search process across the known information pertaining to that observation (Bergadano et al., 2000).

Whilst the generation of counterfactual explanations poses numerous technical challenges, it is imperative to consider various ethical dimensions. Primarily, there is an expectation that their usage is secure, as exposing the model's internal workings through counterfactuals could potentially facilitate model theft (Sokol and Flach, 2019). Additionally, counterfactual explanations should strive to be fair, ensuring that discriminatory explanations are avoided at all costs (Kusner et al., 2018). Moreover, these explanations should be actionable, meaning that the proposed alterations to the input should be practical and feasible (Lucic et al., 2020). Lastly, accountability is a crucial aspect, emphasizing the need to ensure responsibility for the explanations furnished (Barredo Arrieta et al., 2020).

Contrastive explanations

Accumulated insights from the humanities and social sciences regarding explanation demonstrate its inherent contrastive nature (Miller, 2019). This characteristic implies that an explanation addresses a "why" question concerning the cause of a specific event ("Why did P happen"), considering hypothetical non-occurring alternatives ("Why did P happen rather than Q?") (P. Lipton, 1990). Therefore, proponents of the pragmatic explanation approach assert that the ability to distinguish the answer to an explanatory question from a set of contrastive hypothesized alternatives is what furnishes the explainee with sufficiently comprehensive insights into the rationale behind the question. Additionally, this approach establishes a fundamental criterion that an explanation must meet: it should assign higher probability to the observed event P compared to all hypothetical alternatives $(Q_1, Q_2, ..., Q_n)$ (Sørmo et al., 2005).

The domain of cognitive science has been significantly impacted by the exploration of contrastive explanation (Chin-Parker and Bradner, 2017; Roese, 1997). It is contended that contrastive explanations are an intrinsic part of human cognition (R. M. Byrne, 2002). We naturally tend to question decisions we've previously made,

particularly if these decisions or the accompanying circumstances led to tragic events (Wenzlhuemer, 2009). Moreover, contrastive reasoning serves as the foundation for abductive inference (Folger and Stein, 2016), where the objective is to infer specific facts that make certain observations plausible. Simply put, a particular observation can be elucidated by the most probable hypothesis among a set of competing alternatives (Rappaport, 1996).

Counterfactual explanations

Considering the inherent contrastive nature, one can envision alternative explanations for how things might have been if a different decision had been made at a certain point. This perspective can serve to explain potential consequences of such contrastive non-taken alternative decisions (Stepin et al., 2021). In this scenario, it is assumed that the mind generates and juxtaposes mental representations of an event that actually occurred with those of an alternative event (R. Byrne, 2015). Cognitive scientists term these mental representations of alternatives to past events as counterfactuals ("contrary-to-fact") (Roese, 1997). Consequently, the process of contemplating 'past possibilities and present or past impossibilities' is referred to as counterfactual thinking (R. M. J. Byrne, 1997). Furthermore, it is argued that counterfactual reasoning constitutes a fundamental mechanism for elucidating adaptive behavior in an evolving environment (Paik et al., 2014; Y. Zhang et al., 2015; B. Zhou et al., 2015).

Counterfactuals describe events or states of the world that did not occur and implicitly, or explicitly, contradict factual world knowledge. One definition of counterfactual (Grahne, 1991) defines a counterfactual to be a conditional statement where the antecedent "can contradict the current state of affairs, or our current knowledge thereof". Another interesting perspective is introduced by (Ginsberg, 1986), they define counterfactual as a conditional statement of the form "If P, then Q" where P is "expected" to be false. In the broader computer science context, the veracity of a counterfactually deduced statement is determined by: (i) creating a scenario where the most minimal alteration in the characteristics of the real world (as defined in the antecedent) results in a distinct (potentially desired) state of affairs (referred to as the "closest" or "nearest" conceivable world); and (ii) gauging the accuracy of statements in that particular scenario (Tomberlin, 1989).

There have been many some attempt to apply the counterfactual to explain GNNs (Bajaj et al., 2022; Z. Huang et al., 2023; Lucic et al., 2022) in pursuit of the minimal perturbation of the graph structure that leads to a change in the GNN prediction outcome, and we are gonna cover this topic in detail later in this work (section 3.1).

Chapter 3

CFPGExplainer: Parameterized Counterfactual Explanations for Node Classification in Graph Neural Networks

In this chapter we focus on the eXplainable Artificial Intelligence for Graph Neural Networks method that is the novelty introduce in this work: Counterfactual Parameterized Explaniner for graph neural network (CFPGExplainer). In section 3.1 we present an in-depth analysis of the XAI models that have mainly inspired our work, consequently we propose a formal definition (section 3.2.1) for the research problem faced by CFPGExplainer. In the end of the chapter (section 3.2) we illustrate the architectural choices and the proposed approach implemented in the realization of our model.

3.1 Related works

As introduced in the background section of this text (section 2.2.3), despite the impressive success of GNNs on many predictive tasks, these models, coming as a direct evolution of deep networks on graph data, are black-box machine learning models. This means, as stated earlier, that is non-trivial to explain the rationale behind particular prediction made by a GNN, even though the ability to explain a model is critical towards making it trustworthy and thus suitable for many real-world applications. Owing to this limitation of GNNs, there has been significant efforts in recent times towards explanation approaches.

In the next section we provide an in-depth look on the existing work on explaining

GNN predictions that have mostly influenced this work: first of all GNNExplainer (Ying et al., 2019) for introducing the concept of "explainer" for graph neural networks and for dealing with the lack of benchmark datasets for explanation tasks; PGExplainer (Luo et al., 2020), being one of the first work trying to overcome the instance-level restrictions of graph network explainers; finally we take a closer look to CF-GNNExplainer (Lucic et al., 2022) and GCFExplainer (Z. Huang et al., 2023), two of the most recent and interesting approaches, according to the authors of this text, to counterfactual explanations for GNNs.

Lastly, an overview of the other works that have mostly influenced this thesis is provided. Especially over the graph generation methods (Du et al., 2022; Liu et al., 2019), which provided many useful insights on the explanation generation step of this work, considering the similarity between the two tasks (Zhu et al., 2022), and the Graph Adversarial Attacks framework (Bose et al., 2020), taking inspiration from the analogies between counterfactual explanation and adversarial attacks on graph (additional details are provided in sections 3.1.5 and 3.1.6).

Notation

Before exploring the details of the explainer models of interest, here we briefly remind the notation to that is going to be used in the following paragraphs (sections 3.1.1 to 3.1.4). Let G denote a graph on edges E and nodes V ($G = (V, E)$) that are associated with d-dimensional node features $X = \{x_1, ..., x_n\}, x_i \in \mathbb{R}^d$ thus $X \in \mathbb{R}^{n \times d}$. The topology of the graph is encoded by the adjacency matrix $A \in \{0, 1\}^{n \times n}$. And let $f(A, X; W) \rightarrow y$ be any GNN, where y is the set of possible predicted classes, and W is the learned weight matrix of f. In other words, A and X are the inputs of f, and f is parameterized by W. We note that later in this text input graphs and explanation subgraph are indicated directly with the symbol of the adjacency matrix A, given that it is the representation used for graph in practice in all the works analyzed in this section. Without loss of generality, we focus our examples on the problem of explaining a GNN model performing node classification task. Given a node $v \in V$ whose prediction we wish to explain, we denote by $G_v^c = (X_v^c, A_v^c)$ the computational subgraph of $f_\theta(G) = f(A, X; \theta)$. For example, for a three-layer Graph Convolutional Network (GCN, section 2.1.3), G_v^c contains all the neighbors up to order three described with the associated adjacency matrix $A_v^c \in \{0, 1\}^{n \times n}$ and their set of features $X_v^c = \{x_j | j \in X_v^c\}$.

3.1.1 GNNExplainer: Generating Explanations for Graph Neural Networks

(Ying et al., 2019) employs soft masks for edges and node features to elucidate predictions through mask optimization. The process involves initializing soft masks randomly, treating them as trainable variables, and then combining these masks with the original graph using element-wise multiplications. Subsequently, the masks undergo optimization by maximizing mutual information between the predictions of the original graph and the newly obtained graph. Despite using various regularization terms, such as element-wise entropy, to guide the masks towards discreteness, they remain in a soft state, leading to the "introduced evidence" issue that GNNExplainer encounters. Moreover, the optimization of masks is specific to each input graph, potentially resulting in explanations lacking a holistic perspective.

A key insight in their work is the observation that the computation subgraph (considering that the adjacency matrix A is the representation of the graph G) of node v (denoted as A_v^c, or A_v for the sake of simplicity), which is defined by the GNN's neighborhood-based aggregation, fully determines all the information the GNN uses to generate prediction $\hat{y}$ at node v. The neighborhood-based aggregation considers that the node's final representation only includes neighbors that are at most k hops away from that node in the graph G, being k the numbers of layers in the GNN. The rest of the nodes in G are not relevant for the computation of the node's final representation.

Moreover, (Ying et al., 2019) introduced another important concept for this subfield of XAI. GNNExplainer provides the explanation of the prediction of a node v, as the subgraph $\hat{A}_v \subseteq A_v^c$, and the relative features $\hat{X}_v = \{x_j | v_j \in \hat{A}_v\}$, that are "important" for the GNN prediction. They formalize the notion of importance according to the mutual information (MI) criterion, formulated with conditional and joint entropy, i.e. the proposed method finds the explanatory subgraph and sub-features by maximizing the mutual information between the original prediction and the prediction based on the subgraph and sub-features. We can thus express GNNExplainer optimization framework as:

$$\max_{A_s} MI(Y_o, (\hat{A}_v, \hat{X}_v)) = H(Y_o) - H(Y_o | A = \hat{A}_v, X = \hat{X}_v) \tag{3.1}$$

where Y_o is the prediction of the GNN model with A_o (the original unperturbed graph) as the input ($Y_o = f(A_o, X)$). For node v, MI quantifies the change in the probability of prediction $\hat{y} = f(\hat{A}_v, \hat{X}_v)$ when v's computation graph is limited to explanation subgraph $\hat{A}_v$ and its node features are limited to $\hat{X}_v$. It is worth noticing that MI can serve, more generally, as a viable metric to evaluate the quality

of internal representations within deep learning models; the information plane, in particular, offers valuable insights into whether the model effectively utilizes the accessible information present in the dataset (Landsverk and Riemer-Sørensen, 2022).

3.1.2 PGExplainer: Parameterized Explainer for Graph Neural Networks

The method, proposed in (Luo et al., 2020), learns approximated discrete masks over the edges to explain the GNN predictions. PGExplainer, together with GNNExplainer, belongs to the perturbation-based methods for XAI on graph neural networks (section 2.2.3), but act on different components of the graph to induce the perturbation. Indeed PGExplainer produce the explanations perturbing only the structure of the graph, whereas GNNExplainer provides explanations on both structure and features (theoretically both edge and node features). This focus on structure stems from the similarity between feature explanation in GNNs and that in non-graph neural networks, a topic extensively explored in existing literature (Guo et al., 2018; S. M. Lundberg and Lee, 2017; Ribeiro et al., 2016).

For obtaining these edge masks, it adopts a deep neural network to parameterize the generation process of explanations, predicting the probability of each edge being selected (i.e. the probability of being in $\hat{A}_v$), which serves as the importance score. The process involves acquiring edge embeddings or, when edge feature are not available, concatenating embeddings of node incident to an edge, and then using these embeddings to make predictions. Subsequently, the approximated discrete masks are sampled using the reparameterization trick (Niculae et al., 2023). The training of the mask predictor network is carried out by maximizing the mutual information (MI) between the original predictions ($y = f(A, X)$) and the new predictions ($\hat{y} = f(\hat{A}_v, X_v)$), following the importance criterion introduced by (Ying et al., 2019, section 3.1.1). The optimization objective is thus comparable to what we showed in the previous paragraph for GNNExplainer, except for the absence of node features (X_v), given that PGExplainer strategy concerns only edge perturbation:

$$\max_{A_s} MI(Y_o, \hat{A}_v) = H(Y_o) - H(Y_o|A = \hat{A}_v) \tag{3.2}$$

Although the reparameterization trick is utilized, resulting masks are not strictly discrete; however, they significantly mitigate the "introduced evidence" problem. Additionally, since a common predictor is shared by all edges in the dataset, the explanations can offer a comprehensive understanding of the trained GNNs. Indeed, compared to the existing work, PGExplainer has better generalization ability and can be utilized in an inductive setting easily. Such characteristic make this explainer

one of the first, together with (Yuan, Tang, et al., 2020), facing the challenge of producing model-level explanations.

3.1.3 CF-GNNxplainer: Counterfactual Explanations for Graph Neural Networks

Recently, as introduced in section 2.2.3,there have been several attempts to extract explanations of graph neural networks (GNNs) via counterfactual reasoning (Abrate and Bonchi, 2021; Bajaj et al., 2022; Lucic et al., 2022; Tan et al., 2022). In (Abrate and Bonchi, 2021), the authors explore counterfactual explanations within a more specialized category of graphs, namely brain networks (Riedl et al., 2015), which consist of the same set of nodes. They employ a heuristic approach to iteratively add or remove edges in a greedy fashion. RCExplainer (Bajaj et al., 2022) aims to find a robust subset of edges whose removal changes the prediction of the remaining graph by modeling the implicit decision regions (considering the linear decision boundaries (LDBs), Chu et al., 2018) based on GNN graph embeddings. More recently, the authors of CFF (Tan et al., 2022) argue that a good explanation for GNNs should consider both factual and counterfactual reasoning and they explicitly incorporate those objective functions when searching for the best explanatory subgraphs and sub-features.

We focus this supplementary analysis on one of the earlier methods considering counterfactual reasoning for GNN explanation, CF-GNNExplainer (Lucic et al., 2022), that provides counterfactual explanations in terms of a learnable perturbed adjacency matrix that leads to the flipping of classifier prediction for a node. The greatest improvement introduced by the authors with respect to existing methods for explaining the predictions of GNNs, that have primarily focused on generating subgraphs that are relevant for a particular prediction (Baldassarre and Azizpour, 2019; Lin et al., 2021; Luo et al., 2020; Pope et al., 2019; Schlichtkrull et al., 2022; Vu and Thai, 2020; Ying et al., 2019; Yuan, Tang, et al., 2020; Yuan et al., 2021), is that CF-GNNExplainer is able to identify the minimal relevant subgraph automatically, whilst they all require the user to specify the size of the subgraph (i.e. the size of the explanation), S, in advance. The authors reveals that, even when adapting established methods for the counterfactual (CF) explanation challenge and experiment with different values for S, these methods fall short of generating valid, precise CF explanations. Consequently, they are not apt solutions for addressing the CF explanation problem.

The first intuition in their work is that we may want to change the relationships between instances (i.e. edges), rather than change the instances themselves (nodes), given that node feature may not be available. Therefore, a graph data CF example,

for the prediction on node v, should have the form $\hat{v} = (\hat{A}_v, x)$, where x is the feature vector of v and $\hat{A}_v$ is a perturbed version of A_v, the adjacency matrix of the subgraph neighborhood of a node v. $\hat{A}_v$ is obtained by removing some edges from A_v , such that $f(v) \neq f(\hat{v})$. Inspired by (Wachter et al., 2018), (Lucic et al., 2022), they generate countefactcual examples by minimizing a loss function of the form:

$$\mathcal{L} = \mathcal{L}_{pred}(v, \hat{v}|f, g) + \beta \mathcal{L}_{dist}(v, \hat{v}|d) \tag{3.3}$$

where f is the original GNN model, g is the CF model, introduced in the article, that generates the CF example $\hat{v}$, and $\mathcal{L}_{pred}$ is a prediction loss (also referred to as miscalssification loss) that encourages the countefactual property ($f(v) \neq f(\hat{v})$). $\mathcal{L}_{dist}$ is a distance loss that encourages $\hat{v}$ to be "close" to v, and β is a regularization coefficient that controls how important $\mathcal{L}_{dist}$ is compared to $\mathcal{L}_{pred}$. The two sub losses components are defined as:

$$\mathcal{L}_{pred}(v, \hat{v}|f, g) = -\mathbb{1}[f(v) \neq f(\hat{v})] \cdot \mathcal{L}_{NLL}(f(v), g(\hat{v})) \tag{3.4}$$

$$\mathcal{L}_{dist}(v, \hat{v}|d) = dist(v, \hat{v}) \tag{3.5}$$

where the negative sign in front of $\mathcal{L}_{pred}$ and the indicator function ensures the loss is active as long as $f(\hat{v}) = f(v)$. For the *dist* function in $\mathcal{L}_{dist}$ any differentiable distance function can be considered. In (Lucic et al., 2022), they take *dist* to be the element-wise difference between v and $\hat{v}$ ($v \ominus \hat{v}$), corresponding to the difference $A_v \ominus \hat{A}_v$, i.e. the number of edges removed by the perturbation. This loss is tied on the CF model g, a GNN model trained to learn the perturbation matrix P with the explanation objective described. Despite of the novelties introduced by this first approach to counterfactual explanations for GNNs, CF-GNNExplainer is still an instance-based method, lacking the ability to generalize the CF examples to the whole dataset and provide an higher-level comprehension of deep graph models.

3.1.4 GCFExplainer

In (Z. Huang et al., 2023) the authors propose the first method for "global" counterfactual explanations starting from the observation that most existing explainers for graphs (like Vu and Thai, 2020; Ying et al., 2019; Yuan, Tang, et al., 2020), both factual and counterfactual ones, provide instance-level explanation, considering a too narrow local perspective, generating counterfactual examples for individual input graphs. However, this kind of approach poses two key limitations: lacks global insights when working with multiple of data graphs; there can be an information overload owing to the number of counterfactual graphs growing linearly with the

graph dataset size, significantly reducing the human-intelligible property highly desirable in XAI.

Specifically, GCFExplainer aim to find a small set of representative counterfactual graphs that explains all input graphs, proposing an algorithm powered by vertex-reinforced random walks (VRRW, Pemantle, 2004) on an edit map of graphs with a greedy summary. The edit map ($\mathcal{G}$) is a meta-graph whose nodes are graphs in the same domain as the input graphs and edges connect graphs that differ by a single graph edit. $\mathcal{G}$ is used to organize the search space for counterfactual graphs in GCFExplainer, on which the random walks are performed. As an example, each graph in represents a node in the edit map, and the arrows denote edges between graphs (nodes) that are one edit away. VRRW has the nice property of converging to a set of nodes that are both "important" (i.e., cover many input graphs) and "diverse" (i.e., non-overlapping coverage), which will form a small set of counterfactual candidates for further processing.

In this very peculiar setting, although the vertex-reinforcement mechanism promotes diversity among heavily visited nodes, (Z. Huang et al., 2023) introduce another formulation of the importance function, by assigning a large-edge[1]weight $w(\bar{v}, \bar{u})$ to good counterfactual candidates via an importance function $I(\bar{v})$:

$$w(\bar{v}, \bar{u}) = I(\bar{v}) \tag{3.6}$$

The importance function $I(\bar{v})$ is designed to capture the quality of a subgraph as a counterfactual candidate. It has the following components: a counterfactual probability $p(\bar{v})$, that large-node $\bar{v}$ is a CF grah; an individial coverage measure (**coverage**$(\bar{v})$)that computes the proportion of input graphs that are close to $\bar{v}$; and a measure of the coverage gain (**gain**$(v; \mathbb{S})$), when adding graph $\bar{v}$ to the current set of counterfactual candidates $\mathbb{S}$ (i.e., the n most frequently visited large nodes).

$$I(v) = p(\bar{v})(\alpha \textbf{coverage}(\bar{v}) + (1 - \alpha)\textbf{gain}(\bar{v}; \mathbb{S})) \tag{3.7}$$

where α is a hyperparameter between 0 and 1. Led by the above importance function, the VRRW method converges to a set of diverse nodes that have high counterfactual probability and collectively cover a large number of input graphs.

[1]With the prefix "large-" the authors of GCExplainer indicates the components of the edit map $\mathcal{G}$ that the random-walk explore (Z. Huang et al., 2023). Being meta-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ a graph in turn, they talk about large-nodes ($\bar{v} \in \mathcal{V}$) and large-edges ($\bar{e} \in \mathcal{E}$).

3.1.5 Graph generation

In general, the task of graph generation is focused on computing hand-crafted statistical features of existing graphs with the purpose of generating new graphs with similar features. There are many real-world application for this task, for example, in drug discovery and chemical science, a fundamental yet challenging task is to generate novel, realistic molecular graphs with desired properties (e.g., high drug-likeness and synthesis accessibility, Jin et al., 2019; Zang and Wang, 2020). Traditionally graph generation models assume that real-world graphs obey certain statistical rules. However, this assumption oversimplifies the underlying distributions of graphs and is thus too strong to satisfy. As an illustration, the Barabási-Albert model (Albert and Barabási, 2002) posits that similar graphs adhere to the same empirical degree distribution. However, this model falls short in capturing other essential elements, such as community structures, present in real-world graphs. Consequently, there is a growing interest in the development of deep models tailored for graph-structured data, as they facilitate the effective generation of complex graphs.

In the vast literature about graph generation (Zhu et al., 2022), our attention was attracted by two methods: Constrained Graph Variational Autoencoder (CGVAE, Liu et al., 2019) and Monotonically Disentangled VAE (MD-VAE, Du et al., 2022). The author of CGVAE were among the first to propose a probabilistic model for graph generation that builds gated graph neural networks (GGNNs, Li et al., 2017) into the encoder and decoder of a variational autoencoder (VAE, Kingma and Welling, 2013) architecture. The graph generation framework of CGVAE follows a sequential approach to generation, starting from random nodes (*focus* step) and generating the edge between them with the most probable class (*expand*) step, in a graph-walk fashion order. The fundamental novelty in this work is that CGVAE is variational autoencoder model in which both encoder and decoder are graph-structured (i.e. graph neural network, GGNNs in this specific case). They also demonstrate how to incorporate hard domain-specific constraints to adapt the model for the molecule generation task. MD-VAE framework shares with CGVAE the intuition of using GNN models in the encoder and decoder roles of the VAE architecture. The main differences lie in how the molecule graph is generated, in (Du et al., 2022) the generation is not sequential and is carried out through inference, and in the property control mechanism employed. In the work by (Liu et al., 2019), the property control is performed at generation time, ensuring an higher validity for output molecules, but slowing down the whole process; whilst MD-VAE adopts the *post-hoc* strategy of generating latent variables monotonically increasing, directly related to the specific characteristics desired for the output molecule. This is achieved with monotonic regularization, penalizing the violation of constraints via additional regularization

terms together with the original objective.

3.1.6 Generalizable Adversarial Attacks

Another interesting point of view on explainable artificial intelligence comes from the cybersecurity perspective. It has been demonstrated in various settings, that the lack of interpretability and robustness of DNNs makes them vulnerable to adversarial attacks (Sun et al., 2023). For instance, (Szegedy et al., 2014) have pointed out the susceptibility of DNNs in image classification, proving how the performance of a well-trained DNN can be significantly degraded by adversarial examples, which are carefully crafted inputs with a small magnitude of perturbations added designed to fool prediction models. On the other hand it is also true that paying attention special attention to the security of deep network model can help satisfy the increasing need of developing trustworthy deep learning models. Thus, ensuring that a model has not been tampered with, and that its behavior is correct and faithful with respect to its design it is crucial to the proper application of explainability techniques.

Other than a mutual benefit in their recent exponential growth, the fields of XAI and Cybersecurity share some more characteristics. In machine learning, adversarial examples are inputs to ML models that an attacker has intentionally crafted by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence; they can be considered as optical illusions for machines (I. J. Goodfellow et al., 2015). Taking a closer look inside the realm of XAI, the concept of counterfactual example tightly resembles that of adversarial example. If we consider perturbation-based approaches (section 2.2.3) to explain deep graph models, the goal of those methods is to craft a small perturbation over input features (nodes) or structure (edges) that will result induce a change in final model prediction. Basically adversarial and counterfactual example can be seen as two sides of the same coin.

Building on this intuition, when designing the explainer proposed in this text, we took inspiration from the literature concerning adversarial attacks and defense techniques (Sun et al., 2023), specifically from the work of Bose et al., 2020. GAALV (Generelizable Adversarial Attacks with Latent Variable perturbation mod- eling) model offers a unified generative framework for whitebox adversarial attacks, which is easily adaptable to different input domains and can efficiently generate multiple adversarial perturbations. GAALV leverages advancements in deep generative modeling to learn an encoder-decoder based model with a stochastic latent variable that can be used to craft minimal perturbations. This framework brings forth several key benefits, main ones are: its domain agnostic nature, GAALV can be easily

deployed in different domains (e.g., images, text, or graphs are tested in the original paper) by simply selecting the appropriate encoder, decoder, and similarity function for that domain; an efficient generalization, achieved employing a parametric model with a stochastic latent variable allows us to amortize the inference process when crafting adversarial examples; and finally the possibility to generate diverse by resampling attacks, when an initial attack fails.

Without loss of generality, they define a generic adversarial generator network as a combination of four main components: a probabilistic encoder network, that defines a conditional distribution over latent encodings; probabilistic decoder network, that maps a sampled latent encoding to a perturbation; a combination function, that takes an input x and perturbation as input and outputs a perturbed example; and a similarity function, used to restrict the space of adversarial examples;

Even though this method is focused on perturbing node features, the generated perturbation influences j nodes inside the neighborhood of the node v whose prediction we want to taint (i.e. its computational subgraph A_v^c), contrasting with the subgraph (edge) perturbation proposed in our method (section 3.2.1), and considered the previously mentioned similarity between CF and adversarial example, we took inspiration from GAALV framework to build our explainer model. Finally, given the different focus of this thesis, we refer the reader to the already cited, and the most comprehensive, at the time of writing this text, survey on adversarial attacks on graph data by Sun et al., 2023.

3.2 Proposed Approach

In this section (3.2) we introduced the core of this thesis, the novel deep graph network explianer CFPGExplainer (**C**ounter**F**actual **P**arameterized **G**raph neural network **E**xplainer), a model-agnostic framework to generate counterfactual explanation for graph neural networks. With this method we aim to address some issues still present in the literature regarding explanation techniques for GNNs. First of all, the need to model-level CF explanations. To the best of our knowledge, the only works facing this challenge are GCExplainer (Z. Huang et al., 2023) and RCExplainer (Bajaj et al., 2022), that are capable of generalizing explanation at a model-level, but lack the ability to identify the minimal relevant subgraph automatically, forcing the user to specify the size of the explanation in advance, introducing an important bias on the final output and resulting in poorer performances. As stated in section 3.1.3, CF-GNNExplainer (Lucic et al., 2022) instead resolve this issue, providing a framework that can identify the optimal explanation size, but that it is still bounded by its instance-level approach.

Moreover, the efficiency in producing the counterfactual examples, and explanation more in general, should be considered. Instance-level methods (section 2.2.3), like CF-GNNExplainer and GNNExplainer (Ying et al., 2019), need to carry out a whole inference process for each node prediction one wish to explain, making the explaining process, overall, slower, other than tailoring the explanation to a very specific case. Furthermore, it is straightforward that the higher the number of nodes in a graph, and the higher its connectivity, the scalability of such methods could become a significant issue knowing that various GNN based models have been applied to analyze graph data with millions of instances (Ying et al., 2018). A solution to tackle this problem is offered by PGExplainer (Luo et al., 2020), which adopting a deep neural network to parameterize the generation process of explanations (section 3.1.2), enables a natural approach to explaining multiple instances collectively, and can be utilized in an inductive setting more easily than other strategies. This technique, although very promising, does not contemplate counterfactual reasoning, thus not enforcing all the benefits (section 2.2.4) that such a perspective on the XAI context has to offer.

To briefly summarize, the following are the advantages of CFPGExplainer, the proposed explainer system:

- **model-level explanations** we aim at providing the general insights and high-level understanding to explain deep graph models, with explanation not tailored to specific instances that can grasp a broader view of XAI tasks.

- **efficient generalization** employing a parametric model with a stochastic

latent variable allows us to amortize the inference process when crafting CF examples. In doing so, after training, we can efficiently generalize without any further optimization to unseen test examples with only a single pass through the trained explainer network.

- **counterfactual explanations** in many real-world applications producing an explanation in the form of the minimal input perturbation that result in an alternative output is way more meaningful than other approaches. For instance in drug discovery (Stokes et al., 2020; Xie et al., 2021), knowing why a molecule is predicted to not have a desirable property and what it lacks to achieve it, can help inform the design of new molecules and improve the understanding of molecular structures.

3.2.1 Problem form definition/formulation

Following the notation in section 3.1, we introduce in this section a more formal definition for the counterfactual explanation problem that the proposed CFPGExplainer aims to resolve.

In its most general form, a CF example $\hat{x}$ for an instance x according to a trained classifier f' is found by perturbing the features of x such that $f'(\hat{x}) \neq f'(x)$ (Wachter et al., 2018). For graph data, as seen in section 2.2.3, it may not be enough to simply perturb node features, especially since they might not be always available. This is why we focus our attention in generating CF examples by perturbing the graph structure instead, considering also that the unique network structure of GNNs is the greatest peculiarity of these models. Therefore, in our setting, a counterfactual example $\hat{v}$ for the node prediction task performed by a GNN model $f_\theta(A, X)$ on a node v is represented by the perturbed computational subgraph of v, $\hat{A}_v$. $\hat{A}_v$ is obtained by perturbing, i.e. removing some edges, from A_v such that $f_\theta(\hat{A}_v, X_v) \neq f_\theta(A_v, X_v)$. The CFPGExplainer model computes the perturbation mask P, that when applied to A_v produces $\hat{A}_v = P \otimes A_v$ (where $\otimes$ indicates the operation of removing the edges individuated by perturbation P). For the sake of simplicity, later in this section we denote $f_\theta(\hat{G}_v) = f(\hat{A}_v, X_v; \theta)$ and $f_\theta(G_v) = f(A_v, X_v; \theta)$, being $\hat{G}_v = (\hat{A}_v, X_v)$ and $G_v = (A_v, X_v)$ respectively. Following (Lucic et al., 2022) we generate the counterfactual examples by minimizing a loss function of the form:

$$\mathcal{L} = \xi \cdot \mathcal{L}_{pred}(G_v, \hat{G}_v | f) + \beta \cdot \mathcal{L}_{size}(A_v, \hat{A}_v) + \gamma \cdot \mathcal{L}_{ent}(P) \tag{3.8}$$

where $\mathcal{L}_{pred}$ is a mis-classification loss that encourages the final perturbation to produce a counterfactual example ($f(G_v) \neq f(\hat{G}_v)$). $\mathcal{L}_{size}$ is a term to enforces the

the perturbation to be minimal, penalizing the loss when the explanation size grows. $\mathcal{L}_{ent}$ serves to reduce the entropy in the values of the perturbation mask. Finally ξ, β and γ are regularization coefficient to control the importance of each term in $\mathcal{L}$.

More in detail we do not want $f(G_v)$ to match $f(\hat{G}_v)$ to enforce the counterfactual property, thus we put a negative sign in front of $\mathcal{L}_{pred}$ and include an indicator function to ensure the loss is active as long as $f(G_v) \neq f(\hat{G}_v)$:

$$\mathcal{L}_{pred}(G_v, \hat{G}_v | f) = -\mathbb{1}[f(G_v) = f(\hat{G}_v)] \cdot \mathcal{L}_{NLL}(f(G_v), f(\hat{G}_v))) \tag{3.9}$$

$\mathcal{L}_{size}$ is the number of edges removed by the perturbation, essentially the element-wise difference between A_v and $\hat{A}_v$:

$$\mathcal{L}_{size}(A_v, \hat{A}_v) = dist(A_v, \hat{A}_v) = A_v \ominus \hat{A}_v \tag{3.10}$$

The last term $\mathcal{L}_{ent}$ is a binary entropy loss (I. Goodfellow et al., 2016) computed over all the elements of P, since each $\rho \in P$ represents the probability of each edge to be part of the perturbation (note that $N_P = |P|$):

$$\mathcal{L}_{ent}(P) = -\frac{1}{N_P} \sum_{\rho \in P} [\rho \cdot log(\rho) + (1 - \rho) \cdot log(1 - \rho)] \tag{3.11}$$

$$P \text{ s.t. } f(G_v) \neq f(\hat{G}_v), \text{ where } \hat{G}_v = (A_v \otimes P, X_v) \tag{3.12}$$

Reparametrization trick Nonetheless, directly optimizing the aforementioned objective function becomes infeasible due to the presence of 2^m (we remind that $m = |E|$) potential candidates for $\hat{A}_v$. Thus, we need to consider a relaxation by assuming that the explanatory graph is a Gilbert random graph (Gilbert, 1959), where selections of edges from the original input graph A_o are conditionally independent to each other.

Because $\hat{A}_v$ is discrete in nature, we relax edge weights from binary variables to continuous variables in the range $(0, 1)$ and adopt the reparameterization trick to efficiently optimize the objective function with gradient-based methods (Jang et al., 2017). Categorical variables are a natural choice for representing discrete structure in the world, for this reason, following (Luo et al., 2020) we adopt the Gumbel-Softmax, a continuous distribution on the simplex that can approximate categorical samples, and whose parameter gradients can be easily computed via the

reparameterization trick, and thus integrated into neural networks and trained using standard backpropagation (Niculae et al., 2023). The problem of discrete latent structures classification in neural networks can be tackled with various techniques, and in this work some test have been performed using sparsemax (A. F. T. Martins and Astudillo, 2016). Sparsemax is a sparse alternative to softmax that corresponds to regularizing the argmax with the Gini entropy (Ross, 2007). Unlike the softmax function, sparsemax does not necessarily eliminate sparse points, and it can even yield a vertex, especially when the disparity between the highest and second-highest scores is sufficiently substantial. This behavior mirrors the margin property found in max-margin losses when applied in a latent context (Blondel et al., 2020; Niculae et al., 2023).

3.2.2 Explainer architecture

We now introduce our method CFPGxplainer, as stated in the beginning of this chapter (chapter 3), its workflow and architecture, and summarize its details in fig. 3.1. Note that in the following description we are assuming that both the explainer model and the GNN model have been already trained on the tasks of generating the perturbation mask and node classification respectively. Given a node v in the test set, we first obtain its original prediction y_o from GNN model f, as $y_o = f_\theta(A_v, X_v)$, next we run CFPGExplainer to compute the perturbation mask P and obtain the perturbed computation subgraph of v, $\hat{A}_v = P \otimes A_v$. Having $\hat{A}_v$ we execute again the GNN to be explained to see if the perturbation is sufficient to induce a change in the final prediction, using the perturbed adjacency we compute $\hat{y} = f_\theta(\hat{A}_v, X_v)$. At this point, if $\hat{y} \neq y_o$, we have produced a valid counterfactual example for v, $\hat{v} = \hat{A}_v$. The following picture illustrates the general process, just described, to generate counterfactual examples.

Explainer inner working The explainer module (fig. 3.2) processes the input data executing three main steps: an encoding phase, a decoding phase and a sampling phase. First the input data fed through a three-layer GCN model to get an encoded representation that takes into account the structure of the graph, inspired by the many works in graph generation literature that exploit this strategy (Zhu et al., 2022), especially those works focused on deep generative models based on Variational AutoEncoders (VAEs). The key intuition in this step is that this "parallel" GCN model is trained following the counterfactual learning objective (eq. (3.8)). We chose to implement three layer of graph convolution so that the "parallel" model can be aware of the same computational subgraph A_v^c) the primary GNN model is

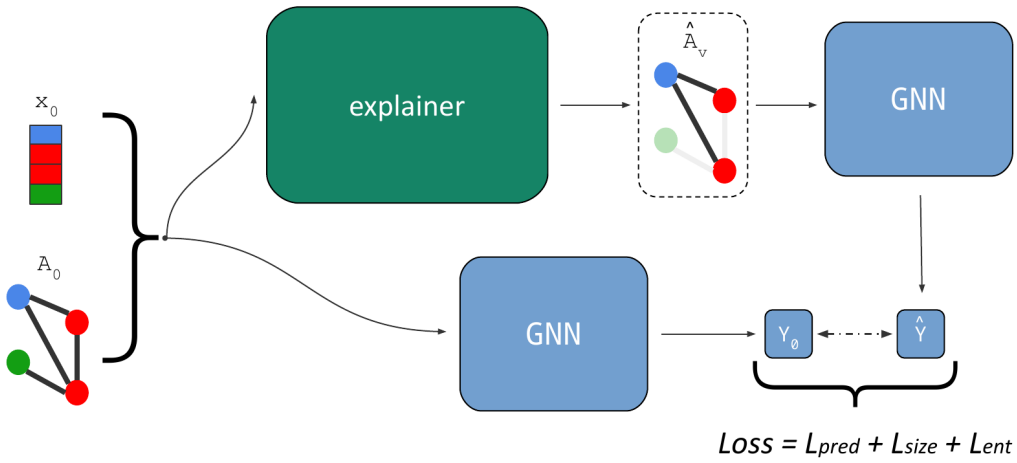


Figure 3.1. *The CFPGExplainer method general pipeline. The raw input data A_o, X_o, node feature and adjacency matrix respectively, are initially fed both to the explainer and to the GNN model.*

aware, as in (Ying et al., 2019), that is to say the 3-hop neighborhood in this precise instantiation.

The second phase consists of crafting edge representations (a sort of inner embeddings) for those edges that constitute the computational subgraph of a node, the ones involved by the explanation analysis, and subsequently feeding them into two fully connected layers that are supposed to learn to assign to each edge the probability of being part of the explanation. Following (Luo et al., 2020) edge embeddings are built concatenating the embeddings of the nodes connected by each edge, and the embedding of the node whose prediction we are explaining, together. Thus, when explaining the prediction of v, to craft the embedding of edge $e = (u, w), e \in A_v^c$, we concatenate the embeddings of v, u, w, namely $x_v, x_u, x_w \in X$, and obtain $e' = (x_v \cup x_u \cup x_w)$.

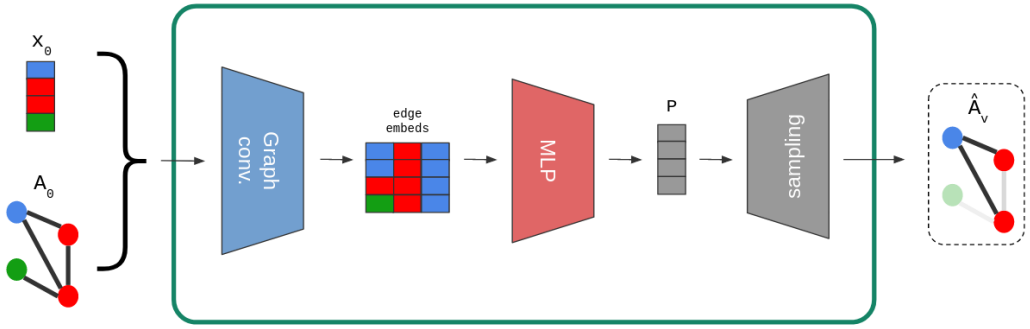


Figure 3.2. *A close-up look to CFPGExplainer inner workings. For the final sampling phase we have tested two strategies: Gumblel Softmax and Sparsemax. Note that the actual model implements one sampling module at a time.*

Lastly, in the third sampling phase, we sample a random graph, the perturbation P, from edge distributions acquired in the second phase, that is then combined to A_v^c to produce its perturbed version $\hat{A}_v$ (i.e. the CF example) that can thus be fed again to the trained GNN model to get the masked prediction. To conclude this final step, we employ categorical reparameterization with gumbel-softmax (section 3.2.1, Jang et al., 2017). It is noteworthy that in order to seamlessly conclude a single training step the primary GNN model must be previously trained and the original prediction for each node in the training and testing portion of the dataset should be computed in advance, as the model solely requires only the final prediction value.

Chapter 4

Experimental setup

In this chapter we illustrate the experimental settings in which our proposed explainer model has been tested. First we dive in a small introduction of the synthetic dataset developed for specifically for explanation tasks (section 4.1), and real-world data that are suitable for this purpose (section 4.1.1). In section 4.2 we provide an overview of the metrics that should be adopted in a counterfactual XAI context, and finally in section 4.3 we present the results of our experiments with CFPGExplainer.

Software and hardware setup The piece of software written to test empirically CFPGExplainer has been developed in python, version 3.10 (Van Rossum and Drake, 2009) relying on the Pytorch[1] machine learning framework (Paszke et al., 2019), on the PYtorch-Geometric[2] libraries specifically designed for geometric deep learning (PyG, Fey and Lenssen, 2019); and on NVIDIA CUDA libraries (version 11.7) for GPU processing (Vingelmann et al., 2020). All experiments have been executed on a laptop machine running a Linux operating system (Ubuntu[3] 22.04) with 16GB of RAM and 2GB of VRAM.

4.1 Synthetic datasets

In recent times, there has been an emergence of synthetic datasets tailored for evaluating explanation techniques (Luo et al., 2020; Ying et al., 2019). In such datasets, different graph motifs are included and can determine the node labels or graph labels. Furthermore, the relationships between data examples and their respective labels are meticulously defined by human experts. Even though the trained GNNs may not perfectly capture such relationships, the graph motifs can be

[1] https://pytorch.org/
[2] https://pytorch-geometric.readthedocs.io/en/latest/index.html
[3] https://www.ubuntu-it.org/

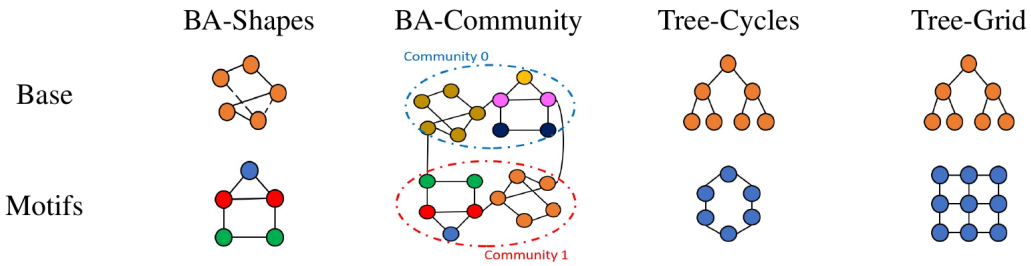


Figure 4.1. *The picture shows an overview of the synthetic datasets. The first row (base) shows the overall outline of each graph in each dataset, whilst the second (motifs) display the shape of the ground truth motifs inside the respective datasets that act as explanation.*

employed as reasonable approximations of the ground truths of explanation results. Here we introduce the most widely used synthetic datasets in GNN explanation tasks.

BA-shapes (syn1): It is a node classification dataset with 4 different node labels. It contains a base graph (300 nodes) and a house-like five-node motif (one for the top of the house, two in the middle, and two on the bottom). Note that the base graph is obtained by the Barabási-Albert (BA) model, which can generate random scale-free networks with a preferential attachment mechanism (Albert and Barabási, 2002). The motif is attached to the base graph while random edges are added. Each node is labeled based on whether it belongs to the base graph or different spatial locations of the motif: there are four possible classes (not in-motif, in-motif top, in-motif middle, in-motif bottom)

BA-community (syn2): a node classification dataset with 8 different labels. Each graph is obtained by combining two BA-shapes graphs with randomly added edges. Node labels are determined by the memberships of BA-shapes graphs and their structural locations. The memberships of the BA-shapes graphs and the structural location determine the labels.

Tree-Cycles (syn3): It is a node classification dataset with 2 different labels. Each graph it consists of a base balanced tree graph with the depth equal to 8 and a six-node cycle motif. These two components are randomly connected. The label for the nodes in based graphs is 0 otherwise 1.

Tree-Grids (syn4): It is a node classification dataset with 2 different labels. It is the same as the Tree-Cycle dataset, except that the Tree-Grids dataset employs nine-node (3×3) grid motifs instead of cycle motifs.

BA-2Motifs It is a graph classification dataset with 2 different graph labels. There are 800 graphs and each of them is obtained by attaching different motifs, such as the house-like motif and the five-node (BA-shapes, section 4.1) cycle motif (Tree-Cycle, section 4.1), to the base BA graph. Different graphs are labeled based on the type of motif.

4.1.1 Real-world datasets

Molecule datasets Molecular datasets are also widely used in explanation tasks, such as MUTAG (Debnath et al., 1991), BBBP (I. F. Martins et al., 2012), and Tox21 (Z. Wu et al., 2018). Each graph within these datasets is a representation of a molecule, with nodes symbolizing atoms and edges denoting the chemical bonds. The labeling of molecular graphs is typically derived from the chemical functionalities or properties inherent to the molecules. Utilizing these datasets for explanation tasks demands domain-specific knowledge, such as an understanding of which chemical groups serve as discriminative factors for their functionalities.

MUTAG molecular dataset is the most common dataset for XAI tasks. It contains several molecules represented as graphs where nodes represent atoms and edges chemical bonds. The molecules are labeled based on their mutagenic effect on a specific bacterium (Debnath et al., 1991). Carbon rings with chemical groups NH_2 or NO_2 are present in mutagenic molecules, and are known to lead to mutagenic effects. The task performed here is binary graph classification (mutagenic,non-mutagenic) and good explainer should identify such patterns for the corresponding class. Obviously MUTAG, as well as all the molecular dataset cited here, is not a synthetic dataset, but is reported here due to many work in XAI for GNNs (Luo et al., 2020; Spinelli et al., 2022; Ying et al., 2019) using it as one of the few real-world dataset containing explanation ground truth values.

Text datasets Text data proves to be a suitable choice for graph explanation tasks because it comprises words and phrases with readily understandable semantic meanings. Consequently, the results of explanations can be easily assessed by humans. Thus, we have constructed three sentiment graph datasets based on text sentiment analysis data, encompassing the SST2, SST5 (Socher et al., 2013), and Twitter (Dong et al., 2014) datasets.

Initially, for each text sequence, we transform it into a graph in which each node represents a word, and the edges signify relationships between different words. To accomplish this, we leverage the Biaffine parser (Gardner et al., 2018) for extracting word dependencies. In Figure 5, we provide an example of one of the sentiment graphs we've generated. It's worth noting that these generated graphs are directed,

although we omit edge labels, as most Graph Neural Networks (GNNs) are incapable of capturing edge label information. Subsequently, we employ BERT (Devlin et al., 2018) to acquire word embeddings, and these embeddings serve as the initial representations for the graph nodes. Specifically, we utilize a pre-trained twelve-layer base BERT model to extract a 768-dimensional feature vector for each word.

4.2 Countefactual metrics

Although visualization results can offer valuable insights into the human perspective on the reasonableness of explanations, these assessments are not entirely reliable due to the absence of established ground truths. Moreover, comparing various explanation methods necessitates human examination of results for each input example, which is a time-consuming process. Furthermore, human evaluations tend to be highly subjective and may lack fairness. Hence, the significance of evaluation metrics in the study of explanation methods becomes evident. Effective metrics should appraise the results from the model's standpoint, gauging factors like the faithfulness of explanations to the model (Jacovi and Goldberg, 2020; Wiegreffe and Pinter, 2019). This section will introduce several recently developed evaluation metrics tailored for explanation tasks, as introduce in (Barredo Arrieta et al., 2020).

Fidelity In tasks involving factual explanations, it is imperative that the explanations faithfully represent the model's perspective. They ought to pinpoint the input features that hold significance for the model, rather than being tailored to human preferences or understanding. In literature, Fidelity is defined as the proportion of nodes where the original predictions match the prediction for the explanations (Molnar, 2022; Ribeiro et al., 2016). Since we are dealing with the counterfactual task of generating counterfactual (CF) examples, our objective is to ensure that the original prediction does not align with the prediction made for the explanation. Therefore, we aim for a low fidelity value in this context. As shown by (Amara et al., 2022), this definition of fidelity can be extended by considering in addition the explanation focus, making some adjustments: when it comes to the phenomenon focus, fidelity is assessed in relation to the ground-truth node label. On the other hand, for the model focus, it is evaluated with regard to the GNN model's output.

Explanation Size In this setting the size of the explanation is the number of removed edges. This metric corresponds to the $\mathcal{L}_{size}$ term in eq. (3.8), which represents the disparity between the original A_v and the counterfactual $\hat{A}_v$. In our quest for concise explanations, our aim is to minimize this metric, seeking a smaller value. It's important to note that we cannot apply this metric to evaluate GNNExplainer like models (Bajaj et al., 2022; Luo et al., 2020; Vu and Thai, 2020; Yuan, Tang, et al., 2020), since they need the user to indicate the explanation size in advance, making this perspective useless.

Sparsity Effective explanations should also exhibit sparsity, signifying their ability to encompass the most critical input features while disregarding irrelevant ones. The

Sparsity metric quantifies this characteristic by gauging the proportion of features identified as important by explanation methods (Pope et al., 2019). More specifically, sparsity measures the proportion of edges in A_v^c that are removed (Yuan et al., 2020b). A value of 0 indicates all edges in A_v^c were removed. Given our preference for succinct explanations, we aim for a value approaching 1 for this metric. Note that, also in this case, we cannot evaluate this metric for GNNExplainer like models as for explanation size (section 4.2), for the same problem in their approach to generating explanation.

Accuracy Additionally, the Accuracy metric has been introduced for synthetic datasets (Sanchez-Lengeling et al., 2020; Ying et al., 2019). In the case of synthetic datasets, even if it remains uncertain whether GNNs make predictions as expected, the principles underlying the construction of these datasets, including graph motifs, can serve as reasonably close approximations of ground truths. Consequently, for any given input graph, we can assess its explanations in relation to these ground truths. For instance, when examining significant edges, we can evaluate the concordance rate between the important edges in explanations and those present in the ground truths.

In the context of this study, accuracy is defined as the average proportion of explanations that are deemed "correct". In line with the methodology established by (Luo et al., 2020; Ying et al., 2019), accuracy is computed exclusively for nodes that were originally predicted to be part of the motifs. This restriction is imposed because accuracy can only be reliably calculated for instances where we have knowledge of the ground truth explanations. As we strive for concise explanations, we classify an explanation as "correct" if it exclusively encompasses edges that are contained within the motifs, meaning it only removes edges that are situated within the motifs.

4.3 Results

In this section we present the results of the experiments executed in this study. First, the focus of our model, CFPGExplainer (chapter 3), is to explain semi-supervised node classification task in GNNs (Kipf and Welling, 2016), thus all the following results have been obtained considering the synthetic datasets (section 4.1) for node classification. Following (Luo et al., 2020) and(Spinelli et al., 2022)), for the sake of simplicity, we have adopted some aliases in developing the code and crafting the results tables; later in this text we refer to the datasets with: `syn1` as BA-shapes, `syn2` as BA-community, `syn3` as Tree-Cycles and `syn4` as Tree-Grids.

The following table (4.1) reports the dataset statistics in the configuration used for this study. Note that since these data are synthetically generated (Ying et al., 2019), the number of nodes, edges and thus the info on the average nodes/edges in A_v^c ($\forall v \in V$) may vary across different works on this very same topic. We followed the same dataset generation as (Spinelli et al., 2022), resulting in this configuration:

	BAshapes (syn1)	BAcommunity (syn2)	TreeCycles (syn3)	TreeGrids (syn4)
#node classes	4	8	2	2
#nodes in motif	5	5	6	9
#edges in motif	6	6	6	12
#node features	10	10	10	10
#tot. nodes	700	1400	871	1231
#tot. edges	4110	8920	1942	3130
avg. nodes in A_v^c	110.17	115.80	13.55	11.10
avg. edges in A_v^c	739.03	1014.10	27.30	27.35

Table 4.1. *Synthetic datasets statistics. The number of edges in motif row in this table also indicates the ground truth size for each dataset.*

The target GNN model implemented in these experiments follows the same architecture of (Ying et al., 2019). For the sake of completeness we have re-executed this model on the node classification task for each of the synthetic datasets (section 4.1) with the results shown in table 4.2.

accuracy	BAshapes (syn1)	BAcommunity (syn2)	TreeCycles (syn3)	TreeGrids (syn4)
train	0.9571	0.9527	0.9282	0.9187
eval	1.0	0.8286	0.9540	0.9187
test	0.9714	0.8714	0.9432	0.9355

Table 4.2. *GNN model performance for the node classification task over the four synthetic datasets tested.*

	BAshapes (syn1)				BAcommunity (syn2)			
	fid.↓	*spars.*↑	*acc.*↑	*size*↓	*fid.*↓	*spars.*↑	*acc.*↑	*size*↓
random	0.65	0.50	0.03	302.20	0.75	0.50	0.02	498.80
1hop	0.0	0.97	0.79	5.05	0.15	0.98	0.68	5.90
CFPGbase	0.05	0.0	0.02	604.20	0.90	0.24	0.02	616.00
CFPG-GCN	0.74	0.99	0.41	2.16	0.92	0.99	0.34	1.73
CFPG-GAT	0.26	0.99	0.78	3.0	0.87	0.99	0.20	1.47

Table 4.3. *Counterfactual metrics for tested explainer configuration on* `syn1` *(BAshapes) and* `syn2` *(BAcommunity). Note that accuracy and size of* `perfect` *explainer are trivial and thus not reported. The column (acc.) refers to the explanation accuracy.*

	TreeCycles (syn3)				TreeGrids (syn4)			
	fid.↓	*spars.*↑	*acc.*↑	*size*↓	*fid.*↓	*spars.*↑	*acc.*↑	*size*↓
random	0.65	0.52	0.51	10.90	0.62	0.50	0.85	13.07
1hop	0.93	0.73	0.66	3.0	0.86	0.79	0.95	2.28
CFPGbase	0.95	0.0	0.53	22.70	0.86	0.0	0.85	26.54
CFPG-GCN	0.45	0.86	0.60	3.56	0.70	0.92	0.88	2.07
CFPG-GAT	0.64	0.93	0.46	1.93	0.37	0.84	0.73	4.09

Table 4.4. *Counterfactual metrics for tested explainer configuration on* `syn3` *(TreeCycles) and* `syn4` *(TreeGrids). Note that accuracy and size of* `perfect` *explainer are trivial and thus not reported. The column (acc.) refers to the explanation accuracy.*

Experiments results Table 4.3 and 4.4 display the results for the tests on node classification explanation over the four synthetic datasets considered (BAshapes and BAcommunity in table 4.3, TreeCycles and TreeGrids in table 4.4 for a better visualization). For all the experiments we have employed the same target GNN model (that achieved the peformance in table 4.2). We adopt the standard metrics for CF explanation tasks on GNNs as elucidated in section 4.2. Four methods are reported: "1hop", "perfect", CFPGbase and CPFG. The first two, "1hop" and "perfect", are two baseline explainers. The former is a trivial explainer that just consider the 1-hop-neighborhood of a node as its explanation (Lucic et al., 2022) and the latter is just a control analysis on the generated datasets to check the inherent *fidelity* and *sparsity* of the ground truth explanation. We remind that these synthetic dataset are randomly generated graphs and the GT motifs are juxtaposed to serve as explanation, thus the correspondence between good values of *fidelity* and *sparsity* with respect to *accuracy* is not completely guaranteed by design. In a real-world context the presence of this relationship must be a characteristics of the data we would like to explain in order to employ such an explanation technique (e.g. in case of molecule datasets, section 4.1.1, we know exactly whether or an edge, a chemical bond, connecting two molecules is needed or not).

The other two are the tested configuration for the architecture proposed in this

study. CFPGbase is the first version of our explainer, it implements the architecture introduced in 3.2.2 without the 3-layer graph convolution encoder. This model serve, in turn, as a baseline to the actual CFPGExplainer, to check if there is an advantage in adopting such a strategy. Finally CFPG represents the complete implementation of our explainer proposal (chapter 3). The two variants of CFPG reported, CFPG-GCN and CFPG-GAT, stands for the tests with a 3-layer GCN encoder module and a 3-layer GAT encoder module respectively. All values shown in the results table (tables 4.3 and 4.4) have been obtained taking the average value after 4 runs with freezed parameters and different seeds, for each of the explainer model.

Lastly we report the optimal configuration for the hyper-parameters for each explainer-dataset couple. We remind that, as in eq. (3.8), ξ, β and γ are the regularization coefficients for the prediction loss ($\mathcal{L}_{pred}$), the size loss ($\mathcal{L}_{size}$) and ($\mathcal{L}_{ent}$) respectively. In addition we report the temperature (τ) values used to control the sampling approximation (when employing Gumbel-Softmax). A couple of values ($\tau = (\tau_1, \tau_2)$) is displayed in tables 4.5 and 4.6 because we adopted a temperature schedule across the epochs (N_e). Thus the temperature τ_e at epoch e is computed with a `lambda` expression as:

$$\tau_e = \text{lambda}_{\tau_1,\tau_2}(e) = \tau_1 \cdot \left(\frac{\tau_2}{\tau_1}\right)^{\left(\frac{e}{N_e}\right)} \tag{4.1}$$

Moreover a value of dropout $\delta = 0.5$ has been used to reduce overfitting on both the encoder GNN model and on the MLP decoder.

CFPGbase	optimizer	learning rate	ξ	β	γ	(τ_1, τ_2)
(syn1) BAshapes	SGD	0.003	5.0	0.001	1.0	(5.0,2.0)
(syn2) BAcommunity	Adam	0.005	2.0	0.001	1.0	(2.0,2.0)
(syn3) TreeCycles	SGD	0.001	5.0	0.05	0.1	(1.0,1.0)
(syn4) TreeGrids	SGD	0.005	5.0	0.01	1.0	(5.0,2.0)

Table 4.5. *CFPGbase hyperparameters for each synthetic dataset.*

CFPG	optimizer	learning rate	ξ	β	γ	(τ_1, τ_2)
(syn1) BAshapes	Adam	0.005	5.0	0.001	0.1	(5.0,1.0)
(syn2) BAcommunity	Adam	0.01	5.0	0.01	0.1	(1.0,1.0)
(syn3) TreeCycles	Adam	0.01	5.0	0.001	0.1	(1.0,0.5)
(syn4) TreeGrids	SGD	0.005	5.0	0.001	0.5	(1.0,1.0)

Table 4.6. *CFPGExplainer hyperparameters for each synthetic dataset.*

Chapter 5

Conclusions

In this thesis, we have explored the critical domain of counterfactual explanations for Graph Neural Networks (GNNs). The ability to understand and interpret the predictions of complex machine learning models is of paramount importance, especially in domains where transparency, accountability, and trust are crucial. GNNs have emerged as a powerful tool for graph-structured data, with applications ranging from social networks to molecular biology. However, their inherent black-box nature poses challenges when it comes to understanding their decision-making processes. Our investigation began by highlighting the significance of GNNs in real-world applications and the need for transparent explanations of their predictions. We emphasized the limitations of existing methods that predominantly focused on generating instance-specific explanations, which lack generalizability and struggle to elucidate patterns across a set of instances.

To address these challenges, we introduced a novel framework, the Counterfactual Parameterized Graph Neural Network Explainer (CFPGExplainer). Our framework aims to enhance the interpretability of GNNs for node classification tasks by generating counterfactual explanations. These explanations offer insights into why a GNN predicts a specific label for a node, empowering stakeholders with actionable insights. While our framework shows promise, there are areas for future exploration. As the field of GNNs and XAI continues to evolve, new challenges and opportunities will emerge. Research into efficient explanation generation and integration with existing GNN models remains a promising avenue. Furthermore, interesting developments stand in the subtle but evident connection between counterfactual and adversarial examples, and in the growing concern regarding the security of AI models, especially GNNs, that are not exempted form the increasing cyber-threats around this research area (Sun et al., 2023).

In closing this thesis, we reflect on the broader significance of our work. The adoption of counterfactual explanations in GNNs can bridge the gap between predictive

accuracy and interpretability. This facilitates the deployment of GNNs in high-stakes, real-world applications, where transparency and trust are paramount. Our research contributes to the evolving field of eXplainable Artificial Intelligence (XAI), enabling stakeholders to gain deeper insights into the decision-making processes of GNNs.

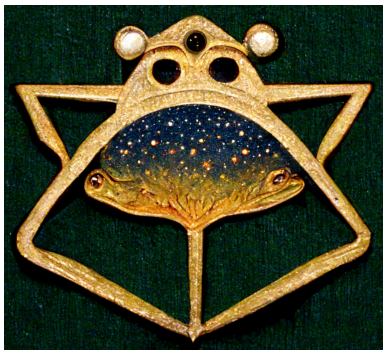
Our thesis corroborates the potential of counterfactual explanations for enhancing the interpretability of Graph Neural Networks. We hope that our work inspires further research and practical applications in this important and evolving field.

Acknowledgments

Infine desidero esprimere la mia profonda gratitudine a tutte le persone che hanno contribuito, direttamente e non, al completamento di questa mia laurea magistrale. Sebbene ci sia solo il mio nome fra gli autori, senza il vostro sostegno non sarei mai riuscito portare a termine il mio percorso accademico.

Vorrei ringraziare tutta la mia famiglia per il supporto e il costante sostegno emotivo (e non solo!), senza i quali non avrei mai potuto raggiungere questo traguardo e nulla di tutto questo sarebbe stato possibile. Ringrazio anche tutti i miei amici, vicini e lontani, che sono stati una costante fonte di ispirazione. I momenti passati insieme sono state preziosi e il vostro appoggio è stato fondamentale per realizzare questo mio obiettivo.

Grazie a tutti di cuore.



Bibliography

Articles

Abrate, C., & Bonchi, F. (2021). Counterfactual graphs for explainable classification of brain networks. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2495–2504. https://doi.org/10.1145/3447548.3467154

Albert, R., & Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, *74*(1), 47–97. https://doi.org/10.1103/revmodphys.74.47

Amara, K., Ying, R., Zhang, Z., Han, Z., Shan, Y., Brandes, U., Schemm, S., & Zhang, C. (2022). Graphframex: Towards systematic evaluation of explainability methods for graph neural networks.

Bajaj, M., Chu, L., Xue, Z. Y., Pei, J., Wang, L., Lam, P. C.-H., & Zhang, Y. (2022). Robust counterfactual explanations on graph neural networks.

Baldassarre, F., & Azizpour, H. (2019). Explainability techniques for graph convolutional networks.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., & Herrera, F. (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, *58*, 82–115. https://doi.org/https://doi.org/10.1016/j.inffus.2019.12.012

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., … Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks.

Bergadano, F., Cutello, V., & Gunetti, D. (2000). Abduction in machine learning. https://doi.org/10.1007/978-94-017-1733-5_5

Blondel, M., Martins, A. F. T., & Niculae, V. (2020). Learning with fenchel-young losses.

Bose, A. J., Cianflone, A., & Hamilton, W. L. (2020). Generalizable adversarial attacks with latent variable perturbation modelling.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, *34*(4), 18–42. https://doi.org/10.1109/msp.2017.2693418

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners.

Byrne, R. (2015). Counterfactual thought. *Annual review of psychology*, *67*. https://doi.org/10.1146/annurev-psych-122414-033249

Byrne, R. M. J. (1997). Cognitive processes in counterfactual thinking about what might have been. *Psychology of Learning and Motivation*, *37*, 105–154. https://api.semanticscholar.org/CorpusID:140755925

Byrne, R. M. (2002). Mental models and counterfactual thoughts about what might have been. *Trends in Cognitive Sciences*, *6*(10), 426–431. https://doi.org/https://doi.org/10.1016/S1364-6613(02)01974-5

Cai, L., & Ji, S. (2020). A multi-scale approach for graph link prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*, 3308–3315. https://doi.org/10.1609/aaai.v34i04.5731

Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., & Elhadad, N. (2015). Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1721–1730. https://doi.org/10.1145/2783258.2788613

Carvalho, D., Pereira, E., & Cardoso, J. (2019). Machine learning interpretability: A survey on methods and metrics. *Electronics*, *8*, 832. https://doi.org/10.3390/electronics8080832

Chaman, A., & Dokmanić, I. (2021). Truly shift-invariant convolutional neural networks. https://arxiv.org/abs/2011.14214

Chen, J., Song, L., Wainwright, M. J., & Jordan, M. I. (2018). Learning to explain: An information-theoretic perspective on model interpretation.

Cheung, M., & Moura, J. M. F. (2020). Graph neural networks for covid-19 drug discovery. *2020 IEEE International Conference on Big Data (Big Data)*, 5646–5648. https://doi.org/10.1109/BigData50022.2020.9378164

Chin-Parker, S., & Bradner, A. (2017). A contrastive account of explanation generation. *Psychonomic Bulletin & Review*, *24*, 1387–1397. https://api.semanticscholar.org/CorpusID:256206491

Chu, L., Hu, X., Hu, J., Wang, L., & Pei, J. (2018). Exact and consistent interpretation for piecewise linear neural networks: A closed form solution. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1244–1253. https://doi.org/10.1145/3219819.3220063

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning, 160–167. https://doi.org/10.1145/1390156.1390177

Dabkowski, P., & Gal, Y. (2017). Real time image saliency for black box classifiers.

Daigavane, A., Ravindran, B., & Aggarwal, G. (2021). Understanding convolutions on graphs [https://distill.pub/2021/understanding-gnns]. *Distill*. https://doi.org/10.23915/distill.00032

Debnath, A. K., Compadre, R. L., Debnath, G., Shusterman, A. J., & Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, *34 2*, 786–97. https://api.semanticscholar.org/CorpusID:19990980

Defferrard, M., Bresson, X., & Vandergheynst, P. (2017). Convolutional neural networks on graphs with fast localized spectral filtering.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

Dong, L., Wei, F., Tan, C., Tang, D., Zhou, M., & Xu, K. (2014). Adaptive recursive neural network for target-dependent Twitter sentiment classification. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 49–54. https://doi.org/10.3115/v1/P14-2009

Doshi-Velez, F., & Kim, B. (2017). Towards a rigorous science of interpretable machine learning.

Du, Y., Guo, X., Shehu, A., & Zhao, L. (2022). Interpretable molecular graph generation via monotonic constraints.

Eksombatchai, C., Jindal, P., Liu, J. Z., Liu, Y., Sharma, R., Sugnet, C., Ulrich, M., & Leskovec, J. (2017). Pixie: A system for recommending 3+ billion items to 200+ million users in real-time.

Folger, R., & Stein, C. (2016). Abduction 101: Reasoning processes to aid discovery. *Human Resource Management Review*. https://doi.org/10.1016/j.hrmr.2016.08.007

Frasconi, P., Gori, M., & Sperduti, A. (1998). A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, *9*(5), 768–786. https://doi.org/10.1109/72.712151

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*, 1189–1232. https://doi.org/10.2307/2699986

Funke, T., Khosla, M., & Anand, A. (2021). Hard masking for explaining graph neural networks. https://openreview.net/forum?id=uDN8pRAdsoC

García, M., & Aznarte, J. (2020). Shapley additive explanations for no2 forecasting. *Ecological Informatics*, *56*, 101039. https://doi.org/10.1016/j.ecoinf.2019.101039

Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M., Schmitz, M., & Zettlemoyer, L. (2018). AllenNLP: A deep semantic natural language processing platform. *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, 1–6. https://doi.org/10.18653/v1/W18-2501

Gehring, J., Auli, M., Grangier, D., & Dauphin, Y. N. (2017). A convolutional encoder model for neural machine translation.

Gilbert, E. N. (1959). Random Graphs. *The Annals of Mathematical Statistics*, *30*(4), 1141–1144. https://doi.org/10.1214/aoms/1177706098

Ginsberg, M. L. (1986). Counterfactuals. *Artificial Intelligence*, *30*(1), 35–79. https://doi.org/https://doi.org/10.1016/0004-3702(86)90067-6

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, 2672–2680.

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples.

Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, *2*, 729–734 vol. 2. https://doi.org/10.1109/IJCNN.2005.1555942

Grahne, G. (1991). Updates and counterfactuals. *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, 269–276.

Grover, A., & Leskovec, J. (2016). Node2vec: Scalable feature learning for networks. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 855–864. https://doi.org/10.1145/2939672.2939754

Hamaguchi, T., Oiwa, H., Shimbo, M., & Matsumoto, Y. (2017). Knowledge transfer for out-of-knowledge-base entities : A graph neural network approach. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 1802–1808. https://doi.org/10.24963/ijcai.2017/250

Hao, Y., Haiyang, Y., Shurui, G., & Shuiwang, J. (2022). Explainability in graph neural networks: A taxonomic survey.

Henaff, M., Bruna, J., & LeCun, Y. (2015). Deep convolutional networks on graph-structured data.

Huang, Q., Yamada, M., Tian, Y., Singh, D., Yin, D., & Chang, Y. (2020). Graphlime: Local interpretable model explanations for graph neural networks.

Huang, Z., Kosan, M., Medya, S., Ranu, S., & Singh, A. (2023). Global counterfactual explainer for graph neural networks. *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, 141–149. https://doi.org/10.1145/3539597.3570376

Ivakhnenko, A. G., Lapa, V. G., & Mcdonough, R. N. (1967). Cybernetics and forecasting techniques. https://api.semanticscholar.org/CorpusID:60378835

Jacovi, A., & Goldberg, Y. (2020). Towards faithfully interpretable nlp systems: How should we define and evaluate faithfulness?

Jang, E., Gu, S., & Poole, B. (2017). Categorical reparameterization with gumbel-softmax.

Jégou, S., Drozdzal, M., Vazquez, D., Romero, A., & Bengio, Y. (2017). The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation.

Jin, W., Barzilay, R., & Jaakkola, T. (2019). Junction tree variational autoencoder for molecular graph generation.

Justin, G., Samuel S., S., Patrick F., R., Oriol, V., & George E., D. (2017). Neural message passing for quantum chemistry.

KIM, Y., & Kim, Y. (2022). Explainable heat-related mortality with random forest and shapley additive explanations (shap) models. *Sustainable Cities and Society*, *79*, 103677. https://doi.org/10.1016/j.scs.2022.103677

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *CoRR*, *abs/1312.6114*. https://api.semanticscholar.org/CorpusID:216078090

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks.

Kusner, M. J., Loftus, J. R., Russell, C., & Silva, R. (2018). Counterfactual fairness.

Landsverk, M. C., & Riemer-Sørensen, S. (2022). Mutual information estimation for graph convolutional neural networks. *Proceedings of the Northern Lights Deep Learning Workshop*, *3*. https://doi.org/10.7557/18.6257

Le, T., Wang, S., & Lee, D. (2020). Grace: Generating concise and informative contrastive sample to explain neural network model's prediction. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 238–248. https://doi.org/10.1145/3394486.3403066

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

LeCun, Y., Bengio, Y., & Hinton, G. E. (2015). Deep learning. *Nature*, *521*, 436–444. https://api.semanticscholar.org/CorpusID:3074096

Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2017). Gated graph sequence neural networks.

Lim, B. Y., Dey, A. K., & Avrahami, D. (2009). Why and why not explanations improve the intelligibility of context-aware intelligent systems. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2119–2128. https://doi.org/10.1145/1518701.1519023

Lipton, P. (1990). Contrastive explanation. *Royal Institute of Philosophy Supplement*, *27*, 247–266. https://doi.org/10.1017/s1358246100005130

Lipton, Z. C. (2017). The mythos of model interpretability.

Liu, Q., Allamanis, M., Brockschmidt, M., & Gaunt, A. L. (2019). Constrained graph variational autoencoders for molecule design.

Lucic, A., Haned, H., & de Rijke, M. (2020). Why does my model fail? contrastive local explanations for retail forecasting. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 90–98. https://doi.org/10.1145/3351095.3372824

Lucic, A., ter Hoeve, M., Tolomei, G., de Rijke, M., & Silvestri, F. (2022). Cf-gnnexplainer: Counterfactual explanations for graph neural networks.

Lundberg, S., & Lee, S.-I. (2017). A unified approach to interpreting model predictions.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *ArXiv*, *abs/1705.07874*. https://api.semanticscholar.org/CorpusID:21889700

Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., & Zhang, X. (2020). Parameterized explainer for graph neural network.

Marcinkevičs, R., & Vogt, J. E. (2023). Interpretability and explainability: A machine learning zoo mini-tour.

Martins, A. F. T., & Astudillo, R. F. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification.

Martins, I. F., Teixeira, A. L., Pinheiro, L., & Falcão, A. O. (2012). A bayesian approach to in silico blood-brain barrier penetration modeling. *Journal of chemical information and modeling*, *52 6*, 1686–97. https://api.semanticscholar.org/CorpusID:28029842

McCallum, A., Nigam, K., Rennie, J. D. M., & Seymore, K. (2000). Automating the construction of internet portals with machine learning. *Information Retrieval*, *3*, 127–163. https://api.semanticscholar.org/CorpusID:349242

Mena, G., Belanger, D., Linderman, S., & Snoek, J. (2018). Learning latent permutations with gumbel-sinkhorn networks.

Micheli, A. (2009). Neural network for graphs: A contextual constructive approach. *IEEE Triansactions on Neural Networks*, *20*(3), 498–511. https://doi.org/10.1109/TNN.2008.2010350

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, *267*, 1–38. https://doi.org/https://doi.org/10.1016/j.artint.2018.07.007

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., & Bronstein, M. M. (2016). Geometric deep learning on graphs and manifolds using mixture model cnns.

Monti, F., Frasca, F., Eynard, D., Mannion, D., & Bronstein, M. M. (2019). Fake news detection on social media using geometric deep learning.

Murphy, R. L., Srinivasan, B., Rao, V., & Ribeiro, B. (2019). Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 807–814.

Niculae, V., Corro, C. F., Nangia, N., Mihaylova, T., & Martins, A. F. T. (2023). Discrete latent structure in neural networks.

N.L.Biggs, E.K.Lloyd, & R.J.Wilson. (1987). Graph theory 1736-1936, (oxford university press). *The Mathematical Gazette*, *71*(456), 177–177. https://doi.org/10.1017/S0025557200108976

Olah, C., Mordvintsev, A., & Schubert, L. (2017). Feature visualization. *Distill*, *2*. https://doi.org/10.23915/distill.00007

Olga, R., Jia, D., Hao, S., Jonathan, K., Sanjeev, S., Sean, M., Zhiheng, H., Andrej, K., Aditya, K., Michael, B., Alexander C., B., & Li, F.-F. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Oord, A. v. d., Dieleman, S., & Schrauwen, B. (2013). Deep content-based music recommendation. *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, 2643–2651.

Paik, J., Zhang, Y., & Pirolli, P. (2014). Counterfactual reasoning as a key for explaining adaptive behavior in a changing environment. *Biologically Inspired Cognitive Architectures*, *10*. https://doi.org/10.1016/j.bica.2014.11.004

Pemantle, R. (2004). Vertex-reinforced random walk.

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 701–710. https://doi.org/10.1145/2623330.2623732

Petar, V., Guillem, C., Arantxa, C., Adriana, R., Pietro, L., & Yoshua, B. (2018). Graph attention networks.

Phillips, P. J., Hahn, C., Fontana, P., Yates, A., Greene, K. K., Broniatowski, D., & Przybocki, M. A. (2021). Four principles of explainable artificial intelligence. https://doi.org/https://doi.org/10.6028/NIST.IR.8312

Pitt, J. C. (1989). Theories of explanation. *Revue Philosophique de la France Et de l'Etranger*, *179*(4), 654–655.

Pope, P. E., Kolouri, S., Rostami, M., Martin, C. E., & Hoffmann, H. (2019). Explainability methods for graph convolutional neural networks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Poyiadzi, R., Sokol, K., Santos-Rodriguez, R., Bie, T. D., & Flach, P. (2020). Face: Feasible and actionable counterfactual explanations. *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. https://doi.org/10.1145/3375627.3375850

Rappaport, S. (1996). Inference to the best explanation: Is it really different from mill's methods? *Philosophy of Science*, *63*(1), 65–80. https://doi.org/10.1086/289894

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "why should i trust you?": Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144. https://doi.org/10.1145/2939672.2939778

Riedl, V., Utz, L., Castrillon, G., Grimmer, T., Rauschecker, J. P., Ploner, M., Friston, K. J., Drzezga, A., & Sorg, C. (2015). Metabolic connectivity mapping reveals effective connectivity in the resting human brain. *Proceedings of the National Academy of Sciences*, *113*, 428–433. https://api.semanticscholar.org/CorpusID:11614278

Roese, N. (1997). Counterfactual thinking. *Psychological Bulletin*, *121*, 133–148. https://doi.org/10.1037/0033-2909.121.1.133

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, v65, No. 6*, 386–408.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, *1*, 206–215. https://doi.org/10.1038/s42256-019-0048-x

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., & Battaglia, P. W. (2020). Learning to simulate complex physics with graph networks.

Sanchez-Lengeling, B., Reif, E., Pearce, A., & Wiltschko, A. B. (2021). A gentle introduction to graph neural networks [https://distill.pub/2021/gnn-intro]. *Distill*. https://doi.org/10.23915/distill.00033

Sanchez-Lengeling, B., Wei, J., Lee, B., Reif, E., Wang, P. Y., Qian, W. W., McCloskey, K., Colwell, L., & Wiltschko, A. (2020). Evaluating attribution for graph neural networks. *Proceedings of the 34th International Conference on Neural Information Processing Systems*.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80. https://doi.org/10.1109/TNN.2008.2005605

Schlichtkrull, M. S., Cao, N. D., & Titov, I. (2022). Interpreting graph neural networks for nlp with differentiable edge masking.

Schnake, T., Eberle, O., Lederer, J., Nakajima, S., Schütt, K. T., Müller, K.-R., & Montavon, G. (2022). Higher-order explanations of graph neural networks via relevant walks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(11), 7581–7596. https://doi.org/10.1109/TPAMI.2021.3115452

Schurz, G. (2000). Scientific explanation: A critical survey. *Foundations of Science*, *1*. https://doi.org/10.1007/BF00145406

Schwarzenberg, R., Hübner, M., Harbecke, D., Alt, C., & Hennig, L. (2019). Layerwise relevance visualization in convolutional text graph classifiers. *Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-13)*, 58–62. https://doi.org/10.18653/v1/D19-5308

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2019). Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, *128*(2), 336–359. https://doi.org/10.1007/s11263-019-01228-7

Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps.

Smilkov, D., Thorat, N., Kim, B., Viégas, F., & Wattenberg, M. (2017). Smoothgrad: Removing noise by adding noise.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., & Potts, C. (2013). Recursive deep models -for semantic compositionality over a sentiment treebank. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1631–1642. https://aclanthology.org/D13-1170

Sokol, K., & Flach, P. A. (2019). Counterfactual explanations of machine learning predictions: Opportunities and challenges for ai safety. *SafeAI@AAAI*. https://api.semanticscholar.org/CorpusID:67865737

Sørmo, F., Cassens, J., & Aamodt, A. (2005). Explanation in case-based reasoning—perspectives and goals. *Artif. Intell. Rev.*, *24*(2), 109–143. https://doi.org/10.1007/s10462-005-4607-7

Sperduti, A., & Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, *8*(3), 714–735. https://doi.org/10.1109/72.572108

Spinelli, I., Scardapane, S., & Uncini, A. (2022). A meta-learning approach for training explainable graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 1–9. https://doi.org/10.1109/tnnls.2022.3171398

Stepin, I., Alonso, J. M., Catala, A., & Pereira-Fariña, M. (2021). A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence. *IEEE Access*, *9*, 11974–12001. https://doi.org/10.1109/ACCESS.2021.3051315

Stokes, J. M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., MacNair, C. R., French, S., Carfrae, L. A., Bloom-Ackermann, Z., Tran, V. M., Chiappino-Pepe, A., Badran, A. H., Andrews, I. W., Chory, E. J., Church, G. M., Brown, E. D., Jaakkola, T. S., Barzilay, R., & Collins, J. J. (2020). A deep learning approach to antibiotic discovery. *Cell*, *180*(4), 688–702.e13. https://doi.org/https://doi.org/10.1016/j.cell.2020.01.021

Sun, L., Dou, Y., Yang, C., Zhang, K., Wang, J., Yu, P. S., He, L., & Li, B. (2023). Adversarial attack and defense on graph data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *35*(8), 7693–7711. https://doi.org/10.1109/TKDE.2022.3201243

Sylvester, J. (1878). Chemistry and algebra. *Nature*, *17*, 284–284. https://api.semanticscholar.org/CorpusID:26451061

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2014). Intriguing properties of neural networks.

Tan, J., Geng, S., Fu, Z., Ge, Y., Xu, S., Li, Y., & Zhang, Y. (2022). Learning and evaluating graph neural network explanations based on counterfactual and factual reasoning. *Proceedings of the ACM Web Conference 2022*, 1018–1027. https://doi.org/10.1145/3485447.3511948

Tealab, A. (2018). Time series forecasting using artificial neural networks methodologies: A systematic review. *Future Computing and Informatics Journal*, *3*(2), 334–340. https://doi.org/https://doi.org/10.1016/j.fcij.2018.10.003

Tomberlin, J. E. (1989). On the plurality of worlds. *Noûs*, *23*(1), 117–125. Retrieved October 18, 2023, from http://www.jstor.org/stable/2215836

Turek, M., Gunning, D., Vorm, E., & Wang, J. Y. (2021). Explainable artificial intelligence (xai). https://doi.org/https://doi.org/10.1002/ail2.61

Ullah, I., Liu, K., Yamamoto, T., Shafiullah, M., & Jamal, A. (2023). Grey wolf optimizer-based machine learning algorithm to predict electric vehicle charging duration time. *Transportation Letters*. https://doi.org/https://doi.org/10.1080/19427867.2022.2111902

Valueva, M. V., Nagornov, N. N., Lyakhov, P. A., Valuev, G. V., & Chervyakov, N. I. (2020). Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Math. Comput. Simul.*, *177*, 232–243. https://api.semanticscholar.org/CorpusID:218955622

Vasilescu, M. A. O. (2009). A multilinear (tensor) algebraic framework for computer graphics, computer vision and machine learning. https://api.semanticscholar.org/CorpusID:125442900

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010.

Vilone, G., & Longo, L. (2021). Notions of explainability and evaluation approaches for explainable artificial intelligence. *Inf. Fusion*, *76*(100), 89–106. https://doi.org/10.1016/j.inffus.2021.05.009

Vishwanathan, S., Schraudolph, N. N., Kondor, R., & Borgwardt, K. M. (2010). Graph kernels. *Journal of Machine Learning Research*, *11*(40), 1201–1242. http://jmlr.org/papers/v11/vishwanathan10a.html

Volodymyr, M., Koray, K., David, S., Andrei, A. R., Joel, V., Marc G., B., Alex, G., Martin A., R., Andreas, K. F., Georg, O., Stig, P., Charlie, B., Amir, S., Ioannis, A., Helen, K., Dharshan, K., Daan, W., Shane, L., & Demis, H. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–533. https://api.semanticscholar.org/CorpusID:205242740

Vu, M. N., & Thai, M. T. (2020). Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Proceedings of the 34th International Conference on Neural Information Processing Systems*.

Wachter, S., Mittelstadt, B., & Russell, C. (2018). Counterfactual explanations without opening the black box: Automated decisions and the gdpr.

Wang, X., Wu, Y., Zhang, A., He, X., & Chua, T.-s. (2021). Causal screening to interpret graph neural networks. https://openreview.net/forum?id=nzKv5vxZfge

Wang, Y., Duan, Z., Huang, Y., Xu, H., Feng, J., & Ren, A. (2021). Mthetgnn: A heterogeneous graph embedding framework for multivariate time series forecasting.

Wenzlhuemer, R. (2009). Counterfactual thinking as a scientific method. *Historical Social Research / Historische Sozialforschung*, *34*(2 (128)), 27–54. Retrieved October 18, 2023, from http://www.jstor.org/stable/20762353

Wiegreffe, S., & Pinter, Y. (2019). Attention is not not explanation.

Wu, F., Zhang, T., de Souza Jr. au2, A. H., Fifty, C., Yu, T., & Weinberger, K. Q. (2019). Simplifying graph convolutional networks.

Wu, L., Chen, Y., Shen, K., Guo, X., Gao, H., Li, S., Pei, J., & Long, B. (2022). Graph neural networks for natural language processing: A survey.

Wu, Y., Lian, D., Xu, Y., Wu, L., & Chen, E. (2020). Graph convolutional networks with markov random field reasoning for social spammer detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(01), 1054–1061. https://doi.org/10.1609/aaai.v34i01.5455

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., & Pande, V. (2018). Moleculenet: A benchmark for molecular machine learning.

Xie, Y., Katariya, S., Tang, X., Huang, E., Rao, N., Subbian, K., & Ji, S. (2022). Task-agnostic graph explanations.

Xie, Y., Shi, C., Zhou, H., Yang, Y., Zhang, W., Yu, Y., & Li, L. (2021). Mars: Markov molecular sampling for multi-objective drug discovery.

Xiong, Z., Wang, D., Liu, X., Zhong, F., Wan, X., Li, X., Li, Z., Luo, X., Chen, K., Jiang, H., & Zheng, M. (2020a). Pushing the boundaries of molecular representation for drug discovery with graph attention mechanism. *Journal of medicinal chemistry*. https://api.semanticscholar.org/CorpusID:199572628

Xiong, Z., Wang, D., Liu, X., Zhong, F., Wan, X., Li, X., Li, Z., Luo, X., Chen, K., Jiang, H., & Zheng, M. (2020b). Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism [PMID:

31408336]. *Journal of Medicinal Chemistry*, *63*(16), 8749–8760. https://doi.org/10.1021/acs.jmedchem.9b00959

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks?

Ying, R., Bourgeois, D., You, J., Zitnik, M., & Leskovec, J. (2019). Gnnexplainer: Generating explanations for graph neural networks.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. https://api.semanticscholar.org/CorpusID:46949657

Yu, H., Yang, L. T., Zhang, Q., Armstrong, D., & Deen, M. J. (2021). Convolutional neural networks for medical image analysis: State-of-the-art, comparisons, improvement and perspectives. *Neurocomputing*, *444*, 92–110. https://doi.org/https://doi.org/10.1016/j.neucom.2020.04.157

Yuan, H., Cai, L., Hu, X., Wang, J., & Ji, S. (2020). Interpreting image classifiers by generating discrete masks. *IEEE transactions on pattern analysis and machine intelligence*, *PP*. https://doi.org/10.1109/TPAMI.2020.3028783

Yuan, H., Tang, J., Hu, X., & Ji, S. (2020). XGNN: Towards model-level explanations of graph neural networks. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &amp, Data Mining*. https://doi.org/10.1145/3394486.3403085

Yuan, H., & Ji, S. (2020). Structpool: Structured graph pooling via conditional random fields. *International Conference on Learning Representations*. https://api.semanticscholar.org/CorpusID:211031192

Zang, C., & Wang, F. (2020). MoFlow: An invertible flow model for generating molecular graphs. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &amp, Data Mining*. https://doi.org/10.1145/3394486.3403104

Zhang, J., Shi, X., Xie, J., Ma, H., King, I., & Yeung, D.-Y. (2018). Gaan: Gated attention networks for learning on large and spatiotemporal graphs.

Zhang, M., & Chen, Y. (2018). Link prediction based on graph neural networks.

Zhang, M., Cui, Z., Neumann, M., & Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1). https://doi.org/10.1609/aaai.v32i1.11782

Zhang, Y., Defazio, D., & Ramesh, A. (2021). Relex: A model-agnostic relational model explainer. *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, 1042–1049. https://doi.org/10.1145/3461702.3462562

Zhang, Y., Paik, J., & Pirolli, P. (2015). Reinforcement learning and counterfactual reasoning explain adaptive behavior in a changing environment. *Topics in cognitive science*, *7*. https://doi.org/10.1111/tops.12143

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2015). Learning deep features for discriminative localization.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2021). Graph neural networks: A review of methods and applications.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., & Sun, M. (2018). Graph neural networks: A review of methods and applications. *ArXiv*, *abs/1812.08434*. https://api.semanticscholar.org/CorpusID:56517517

Zhu, Y., Du, Y., Wang, Y., Xu, Y., Zhang, J., Liu, Q., & Wu, S. (2022). A survey on deep graph generation: Methods and applications.

Inproceedings

Cohen, T., & Welling, M. (2016, June). Group equivariant convolutional networks. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning* (pp. 2990–2999, Vol. 48). PMLR. https://proceedings.mlr.press/v48/cohenc16.html

Fout, A., Byrd, J., Shariat, B., & Ben-Hur, A. (2017). Protein interface prediction using graph convolutional networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/f507783927f2ec2737ba40afbd17efb5-Paper.pdf

Gao, H., & Ji, S. (2019, June). Graph u-nets. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 2083–2092, Vol. 97). PMLR. https://proceedings.mlr.press/v97/gao19a.html

Guo, W., Huang, S., Tao, Y., Xing, X., & Lin, L. (2018). Explaining deep learning models – a bayesian non-parametric approach. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 31). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/4b4edc2630fe75800ddc29a7b4070add-Paper.pdf

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., & Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances*

in neural information processing systems (Vol. 30). Curran Associates, Inc. https://dl.acm.org/doi/10.5555/3295222.3295382

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 25). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Lin, W., Lan, H., & Li, B. (2021, July). Generative causal explanations for graph neural networks. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (pp. 6666–6679, Vol. 139). PMLR. https://proceedings.mlr.press/v139/lin21d.html

Mason, L., Baxter, J., Bartlett, P., & Frean, M. (1999). Boosting algorithms as gradient descent. In S. Solla, T. Leen, & K. Müller (Eds.), *Advances in neural information processing systems* (Vol. 12). MIT Press. https://proceedings.neurips.cc/paper_files/paper/1999/file/96a93ba89a5b5c6c226e49b88973f46e-Paper.pdf

McCulloch, W., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. https://doi.org/10.7551/mitpress/12274.003.0011

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. In *Neurocomputing: Foundations of research* (pp. 696–699). MIT Press.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., & Battaglia, P. (2018, July). Graph networks as learnable physics engines for inference and control. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (pp. 4470–4479, Vol. 80). PMLR. https://proceedings.mlr.press/v80/sanchez-gonzalez18a.html

Yuan, H., Yu, H., Wang, J., Li, K., & Ji, S. (2021, July). On explainability of graph neural networks via subgraph explorations. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (pp. 12241–12252, Vol. 139). PMLR. https://proceedings.mlr.press/v139/yuan21c.html

Books

Berge, C., & Minieka, E. (1976). *Graphs and hypergraphs*. North-Holland Publishing Company. https://books.google.it/books?id=S59iNAEACAAJ

Bondy & Murty. (2011). *Graph theory*. Springer London. https://books.google.it/books?id=HuDFMwZOwcsC

Bronstein, M. M., Bruna, J., Cohen, T., & Veličković, P. (2021). *Geometric deep learning: Grids, groups, graphs, geodesics, and gauges*.

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2022). *Introduction to algorithms, fourth edition*. MIT Press. https://books.google.it/books?id=RSMuEAAAQBAJ

Coxeter, H. S. M. (1998). *Non-euclidean geometry*. Mathematical Association of America. https://doi.org/10.5948/9781614445166

Diestel, R. (2017). *Graph theory*. Springer Berlin Heidelberg. https://books.google.it/books?id=FY5BvgAACAAJ

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* [http://www.deeplearningbook.org]. MIT Press.

Griffin, C. (2023). *Applied graph theory: An introduction with graph optimization and algebraic graph theory*. World Scientific Publishing Company. https://books.google.it/books?id=wIrVEAAAQBAJ

Hamilton, W. (2020). *Graph representation learning*. Morgan & Claypool Publishers. https://books.google.it/books?id=Csj-DwAAQBAJ

Leslie, D., & Briggs, M. (2021). *Explaining decisions made with ai: A workbook (use case 1: Ai-assisted recruitment tool)*. Zenodo. https://doi.org/10.5281/ZENODO.4624711

Lombrozo, T. (2012, March). *260 Explanation and Abductive Inference*. Oxford University Press. https://doi.org/10.1093/oxfordhb/9780199734689.013.0014

Minsky, M., & Papert, S. A. (1969). *Perceptrons: An Introduction to Computational Geometry*. The MIT Press. https://doi.org/10.7551/mitpress/11301.001.0001

Molnar, C. (2022). *Interpretable machine learning: A guide for making black box models explainable* (2nd ed.). https://christophm.github.io/interpretable-ml-book

Ross, S. M. (2007). *Introduction to probability models* (9th). Academic Press.

Tutte, W. (2001). *Graph theory*. Cambridge University Press. https://books.google.it/books?id=uTGhooU37h4C

Venkatesan, R., & Li, B. (2017). *Convolutional neural networks in visual computing: A concise guide*. CRC Press. https://books.google.it/books?id=bAM7DwAAQBAJ

Wasserman, S., & Faust, K. (1994). *Social network analysis in the social and behavioral sciences*. Cambridge University Press. https://doi.org/10.1017/CBO9780511815478.002

Miscellaneous

Council of European Union. (2016). Regulation (eu) 2016/679 of the european parliament (GDPR) [Official Journal of the European Union, 2016]. https://eur-lex.europa.eu/eli/reg/2016/679/oj

Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric. https://pytorch-geometric.readthedocs.io/en/latest/install/installation.html

Krizhevsky, A., & et al. (2009). Cifar10 and cifar100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Van Rossum, G., & Drake, F. L. (2009). Python 3 reference manual. https://dl.acm.org/doi/book/10.5555/1593511 (ii): https://www.python.org/.

Vingelmann, P., Fitzek, F. H., & NVIDIA. (2020). Cuda, release: 11.7, driver version: 515.105.01. https://developer.nvidia.com/cuda-toolkit