



HAL
open science

Explaining machine learning models on graphs by identifying hidden structures built by GNNs

Luca Veyrin-Forrer

► **To cite this version:**

Luca Veyrin-Forrer. Explaining machine learning models on graphs by identifying hidden structures built by GNNs. Artificial Intelligence [cs.AI]. INSA de Lyon, 2023. English. NNT : 2023ISAL0025 . tel-04214965v2

HAL Id: tel-04214965

<https://hal.science/tel-04214965v2>

Submitted on 7 Dec 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



INSA

INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
LYON

N° d'ordre NNT : 2023ISAL0025

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
L'INSA LYON

ECOLE DOCTORALE N° 512
MATHÉMATIQUES ET INFORMATIQUE (INFOMATHS)

SPÉCIALITÉ / DISCIPLINE DE DOCTORAT : INFORMATIQUE

À soutenir publiquement le 30/03/2023 par
LUCA VEYRIN-FORRER

**Explaining machine learning models on graphs by identifying
hidden structures built by GNNs**

Devant le jury composé de:

Pr. Bruno Crémilleux	Université de Caen	Rapporteur
Dr. Marie-Jeanne Lesot	Sciences Sorbonne Université	Rapporteuse
Dr. Luis Galarraga	INRIA	Examineur
Pr. Pascal Poncelet	Université de Montpellier	Examineur
Pr. Christine Solnon	INSA Lyon	Examinatrice
Dr. Stefan Duffner	INSA Lyon	Co-directeur de thèse
Pr. Marc Plantevit	EPITA	Co-directeur de thèse
Pr. Céline Robardet	INSA Lyon	Directrice de thèse

Table of contents

Table of contents	i
1 Introduction	3
1.1 Context	3
1.2 Problems addressed in this thesis	4
1.3 Contributions	5
1.4 Structure of the thesis	6
1.5 List of publications	6
2 State of the art	9
2.1 Graphs	9
2.2 Machine learning on graphs	10
2.2.1 Machine learning tasks	10
2.2.2 From graph to vectors	13
2.2.3 Computing distances between graphs	13
2.2.4 Graph Neural Networks	15
2.2.5 Connections with Weisfeiler-Lehman algorithm	18
2.3 Explaining machine learning models	19
2.3.1 Why do we need explainable artificial intelligence?	19
2.3.2 Evaluate an explanation method	25
2.4 Explaining Graph Neural Networks	26
2.4.1 Gradient-based methods	27
2.4.2 Perturbation-based methods	28
2.4.3 Decomposition-based methods	31
2.4.4 Surrogate methods	34
2.4.5 Global model explanation methods	35
2.5 Discussion	40
2.5.1 Using masks as GNN explanations	40
2.5.2 Further explanation needs at the model-level	41
3 Mining activation rules in the hidden layers of GNNs	43
3.1 Inside GNN motivations and desirata	43
3.2 Graph Neural Network setup	43
3.3 Activation matrix and activation rules	44
3.4 Activation rules discovery	45

3.4.1	Measuring the interest of an activation rule	47
3.4.2	Computing the background model	49
3.4.3	Iterative extraction of subjective activation subgroups	49
3.5	Characterization of activation rules with subgroups	53
3.5.1	Numerical subgroups	53
3.5.2	Graph subgroups	54
3.5.3	Quality measure and algorithms	54
3.6	GNN explanations with activation rules	55
3.7	Experimental study	56
3.7.1	Datasets and experimental setup	56
3.7.2	Quantitative study of activation rules	58
3.7.3	Comparison with competitors for explainability of GNN output	60
3.7.4	Model insights via the (re)description of activation patterns	64
3.8	Discussion and conclusion	68
4	Characterizing activation rules with representative graphs	75
4.1	Problem definition and desiderata	75
4.2	Characterizing activation rules with subgraphs	76
4.2.1	Measuring the proximity between a graph and an activation rule	76
4.2.2	Realism factor	77
4.3	DISCERN method	77
4.4	Experiments	81
4.4.1	Datasets and experimental setup	82
4.4.2	Studying DISCERN behavior	83
4.4.3	Comparison to instance-level methods	88
4.4.4	Comparison to model-level baselines	90
4.4.5	Examples	91
4.4.6	Discussion	92
4.5	Conclusion	93
5	Conclusion and Future Directions	95
5.1	Conclusion	95
5.2	Perspectives	96
	Bibliography	99

Abstract

The last decade has witnessed a huge growth in the development of deep neural network-based techniques for graphs and Graph Neural Networks (GNNs) have proven to be the most effective for many graph machine learning problems. These powerful models are based on node representation learning which avoids the tedious and time-consuming task of hand-crafted feature engineering. However, the internal working of GNN models remains opaque which is the major obstacle to their deployment, raising some issues on societal acceptability and trustworthiness, requirements which enjoin making explicit the internal functioning of such models.

In this thesis, we study the problem of GNN explainability. Our main contribution, INSIDE-GNN, aims at mining activation rules in the hidden layers to understand how the GNNs perceive the world. The problem is not to discover activation rules that are individually highly discriminating for an output of the model. Instead, the challenge is to provide a small set of rules that cover all input graphs. We introduce a subjective activation pattern domain to solve this task. INSIDE-GNN is thus an effective and principled algorithm to enumerate activation rules in each hidden layer. The proposed approach for quantifying the interest of these rules is rooted in information theory and can account for background knowledge on the input graph data. Activation rules can subsequently be used for explaining GNN decisions. Experiments on both synthetic and real-life datasets show highly competitive performance, with up to 200% improvement in fidelity on explaining graph classification model over the state-of-the-art methods.

Activation rules are not interpretable by themselves since they only define internal representations having a strong impact on the classification process. They are not sufficient to examine what the GNN actually captures and shed light on the hidden features built by the GNN. We propose to interpret these rules by identifying a graph that is fully embedded in the related subspace identified by the rule. The devised method, named DISCERN, is based on a Monte Carlo Tree Search controlled by a proximity measure between the graph embedding and the internal representation of the rule, as well as a realism factor that constrains the distribution of the labels of the graph to be similar to that observed on the dataset. The obtained graphs are realistic and fully understandable by the end user.

Résumé

La dernière décennie a vu une énorme croissance dans le développement de techniques basées sur les réseaux de neurones profonds pour les graphes. Les réseaux de neurones sur graphes (GNNs) se sont avérés les plus efficaces pour de nombreux problèmes d'apprentissage automatique de graphes. Ces modèles puissants sont basés sur l'apprentissage de la représentation des nœuds, ce qui évite la tâche fastidieuse de construction artisanale de descripteurs de graphes. Cependant, le fonctionnement interne des modèles GNN reste opaque, ce qui constitue un obstacle majeur à leur déploiement, soulevant des questions d'acceptabilité sociale et de fiabilité, limites qui peuvent être surmontées par l'explication du fonctionnement interne de tels modèles.

Dans cette thèse, nous étudions le problème de l'explicabilité des GNNs. Notre contribution principale, *INSIDE-GNN*, vise à extraire les règles d'activation dans les couches cachées du modèle pour comprendre quels descripteurs et caractéristiques de graphes ont été automatiquement extraits des graphes. Le problème n'est pas de découvrir des règles d'activation individuellement très discriminantes pour une classe du modèle, mais le défi consiste à fournir un petit ensemble de règles qui couvrent tous les graphes d'entrée. Nous proposons un domaine de motifs subjectif pour résoudre cette tâche. Nous proposons l'algorithme *INSIDE-GNN* qui est efficace pour énumérer les règles d'activation dans chaque couche cachée. L'approche proposée pour quantifier l'intérêt de ces règles repose sur la théorie de l'information pour construire un modèle des connaissances apportées par l'ensemble des règles. Les règles d'activation peuvent ensuite être utilisées pour expliquer les décisions du GNN. Les expériences sur des ensembles de données synthétiques et réels montrent des performances très compétitives, avec jusqu'à 200 % d'amélioration de la fidélité sur l'explication du modèle de classification des graphes par rapport aux méthodes de l'état de l'art.

Cependant, les règles d'activation ne sont pas interprétables en elles-mêmes puisqu'elles reposent sur les représentations internes du GNN qui ont un fort impact dans le processus de classification. Elles ne permettent pas d'examiner ce que le GNN capture réellement et à faire la lumière sur les descripteurs cachés construits par le GNN. Nous proposons d'interpréter ces règles en identifiant un graphe entièrement plongé dans le sous-espace associé à chaque règle. La méthode *DISCERN* que nous avons mise au point est basée sur une recherche arborescente de type Monte Carlo dirigée par une mesure de proximité entre le plongement du graphe et la représentation interne de la règle. Les graphes ainsi obtenus sont réalistes et pleinement compréhensibles par l'utilisateur final.

Chapter 1

Introduction

1.1 Context

This thesis has been implemented in the Data Mining and Machine Learning (DM2L) group thanks to a Phd grant from the IDEX project "mACHine LeArning & Data sciEnce for coMplex and dynamICAL models" (ACADEMICS). This collaborative project aims to combine data science and machine learning, two areas of excellence for the involved laboratories (Laboratoire de Physique (LP) and Laboratoire Informatique du Parallélisme (LIP) both from ENS Lyon, Laboratoire Hubert Curien (LabHC) from Université Jean Monnet, Laboratoire d'Informatique en Images et Systèmes d'information (LIRIS)). The challenge is to jointly develop formal frameworks and learning algorithms adapted to difficult scientific contexts involving heterogeneous, irregular, dynamic and complex data, particularly in the form of graphs and networks.

The DM2L group has a strong and long time expertise in Data Mining (Calders et al., 2006), an interdisciplinary subfield of computer science and statistics whose main goal is to extract relevant information from data. Especially, DM2L members have devised many generic algorithms to discover patterns from datasets with a particular attention to the consideration of the prior knowledge, the expressive power and the actionability of the patterns.

Most of this research – and more generally, research in pattern mining – can be easily summarized from an Inductive Database perspective (Imielinski and Mannila, 1996) as the computation of a theory Th defined as:

$$Th(\mathcal{L}, \mathcal{D}, C) = \{\psi \in \mathcal{L} \mid C(\psi, \mathcal{D}) \text{ is true}\}.$$

Given a pattern language \mathcal{L} , some constraints C and a database \mathcal{D} , a pattern mining algorithm aims at enumerating the elements of the language that fulfill the constraints within the data. In practice, the users define their interests in a declarative way and do not specify how to compute the solution. This is an elegant way to define pattern mining. Notice that this is not just a theoretical abstraction and that there have been Inductive Database prototypes proposed in the literature, e.g., the mining view system (Blockeel et al., 2012).

Obviously, many variants are possible. For instance, the pattern mining algorithms can be required to be complete (as in the above formalization) or they can look for a subset of all the patterns (top- k , representative samples) for which the constraints hold. Similarly, on the

constraint part, the users are supposed to express their interests in term of constraints (maximal patterns, threshold-based constraints) so that the pattern mining task is a satisfaction problem. Expressing an interest with constraints is difficult for the users, but alternatively they can emit explicit preferences or have implicit preferences that have to be acquired within the mining process. In this case, the pattern mining task turns out to be an optimization problem.

Based on the inductive database formalization, we can easily summarize via two adjectives the challenges researchers in data mining aim to tackle: *better* and *faster*. Indeed, data mining is always guided by the search for patterns of better quality while taking into account scalability issues.

The major added value of pattern mining is the interpretability of the results, which is a highly demanded feature in Machine Learning, especially since the advent of neural networks.

1.2 Problems addressed in this thesis

In this thesis, we are interested in the explainability of machine learning on graphs methods. Traditional machine learning methods for graphs do not suffer from explainability issues since they require handcrafted features based on graph statistics that are inherently interpretable. Explainability issues have actually appeared with the advent of Graph Neural Networks (GNNs) which have proven to be the most effective for many graph machine learning problems. These powerful models are based on node representation learning which avoids the tedious and time-consuming task of hand-crafted feature engineering. However, the internal working of GNN models remains opaque which is the major obstacle to the deployment of GNNs, raising some issues on societal acceptability and trustworthiness, properties which require making explicit the internal functioning of such models.

The last five years have witnessed a huge growth in the definition of techniques for explaining deep neural networks (Burkart and Huber, 2021, Molnar, 2020), especially for image and text data. However, these methods cannot be directly used for explaining GNN due to the none grid-like format of graphs (Yuan et al., 2020b). Nevertheless, a few proposals have been made to explain GNNs according to two distinct approaches and have gained visibility:

Instance-level methods. These approaches aim to learn a mask seen as an explanation of the model decision for a graph instance (Baldassarre and Azizpour, 2019, Duval and Malliaros, 2021, Luo et al., 2020, Pope et al., 2019, Schnake et al., 2020, Ying et al., 2019). These methods provide input-dependent explanations specific to an input graph by identifying its important characteristics on which the model builds its prediction. We can identify four different families of methods.

The *gradient/feature-based methods* (Baldassarre and Azizpour, 2019, Pope et al., 2019) – directly adapted from dedicated image and text solutions – use the gradients or hidden feature map values to compute the importance of the input features.

The *perturbation-based methods* (Luo et al., 2020, Ying et al., 2019) aim at learning a graph mask by studying the prediction changes when perturbing the input graphs.

The *surrogate methods* (Huang et al., 2020, Vu and Thai, 2020) explain an input graph by sampling its neighborhood and learning an interpretable model.

The *decomposition-based methods* (Pope et al., 2019, Schnake et al., 2020) start by decomposing the prediction score to the neurons in the last hidden layer. Then, they back-propagate these scores layer by layer until reaching the input space.

On top of that, GraphSVX (Duval and Malliaros, 2021) falls into these four categories by learning a surrogate explanation model on a perturbed dataset that decomposes the explained prediction among input nodes and features based on their contribution.

These methods perform well on metrics evaluating the relationships between explanations and model decisions (i.e., fidelity and infidelity metrics). However, it appears that these masks can lead to unreliable explanations, and most importantly, can lead to misleading interpretations for the end-user. One can be tempted to interpret all the nodes or features of the mask as responsible for the prediction leading to wrong assumptions. For instance, a node feature may be perceived as important for the GNN prediction, whereas there is no difference between its distribution within and outside the graphs validating the mask.

Model-level methods. The only existing model-level method is XGNN (Yuan et al., 2020a) which aims at training a graph generator to maximize the predicted probability for a class and uses such graphs to explain this class. However, it is based on the strong assumption that each class can be explained by a single graph, which is unrealistic when considering complex phenomena.

Most of the aforementioned methods aim at either explaining the final decision of a GNN or generating a representative graph for a given decision. We believe that focusing only on the model decision does not allow to fully understand how the model behaves and builds its decision. One can provide additional insights about the GNN by not only looking at the output of the model, but also by trying to characterize some representation subspaces that the model has built in the different layers.

Imagine you are a scientist who has a GNN that accurately predicts the graphs you are studying. This means that the GNN is able to capture interesting features and combine them. Therefore, understanding how the GNN constructs its internal representation would shed new light on your field of research. This is our goal and this is why we introduce, in this thesis, new knowledge discovery from GNN models.

1.3 Contributions

This thesis addresses the problem of explaining GNNs from the perspective of knowledge discovery from models. It requires the definition of new pattern domains adapted to the inner representation of the GNNs. Furthermore, the discovered patterns must be interpretable by a human user which demands the redescription of the patterns discovered in the hidden layers with an interpretable language.

The proposed approaches can be structured in two contributions:

Mining activation rule sets in GNNs: Problem and method. We introduce the novel problem of mining activation rule sets in GNNs. We consider GNNs for graph binary

classification. We introduce a new method, called *INSIDE*, that aims at discovering activation rules in each hidden layer of the GNN. An activation rule captures a specific configuration in the embedding space of a given layer that is considered important in the GNN decision, i.e., discriminant for an output label. The problem is therefore not only to discover highly discriminant activation rules but also to provide a pattern set that covers all GNN decisions on the input graphs. To this end, we define a measure, rooted in the *FORSIED* framework (De Bie, 2011) to quantify the information provided by a rule relative to that supplied by the rules already extracted. We devise a branch-and-bound algorithm that exploits upper-bound-based pruning properties to discover such rules. We report an empirical evaluation which studies the performance and the potential of the proposed approach for providing instance-level explanations or insights about the model. *INSIDE* is compared against SOTA explanation methods and outperforms them by up to 200%. We also study the characterization of activation rules thanks to interpretable pattern languages. We demonstrate that this allows to obtain good summaries of the hidden features captured by the GNN. Based on this, we eventually compare our approach against a model-level explanation method.

Characterizing activation rules with representative graphs. We introduce the novel problem of characterizing internal representations of GNNs as well as our method *DISCERN* to generate realistic graphs that are representative of the activation rules. This method is based on a Monte Carlo Tree Search which relies on a proximity measure between a graph and an activation rule. There are different ways to construct such measures and we propose three different ones. We report an empirical evaluation on several real-world datasets where we study the ability of *DISCERN* to provide good explanations with realistic graphs and compare the three metrics we introduce. *DISCERN* is also compared to six state of the art baselines. These experiments demonstrate that our method provides better and more realistic explanations.

1.4 Structure of the thesis

This thesis is organized as follows. Chapter 2 presents the state-of-the-art on the related topics: (1) Graph Machine Learning, (2) explainable AI and (3) explaining GNNs. This chapter also discusses the limitations we propose to address in this thesis. We then present our first contribution in Chapter 3. We define the activation rules in the hidden layers of the GNN as well as an algorithm, *INSIDE*, to mine a relevant set of activation rules. In Chapter 4, we introduce *DISCERN*, a method to generate an interpretable redescription of activation rules under the form of graphs. We conclude and discuss the future directions in Chapter 5.

1.5 List of publications

Peer-reviewed French national conferences with proceedings:

- Luca Veyrin-Forrer, Ataollah Kamal, Stefan Duffner, Marc Plantevit, Céline Robardet: Qu'est-ce que mon GNN capture vraiment ? Exploration des représentations internes d'un GNN. EGC 2022: 159-170 (best paper award).

Peer-reviewed international conferences with proceedings:

- Luca Veyrin-Forrer, Ataollah Kamal, Stefan Duffner, Marc Plantevit, Céline Robardet: What Does My GNN Really Capture? On Exploring Internal GNN Representations. IJCAI 2022: 747-752

International journals:

- Luca Veyrin-Forrer, Ataollah Kamal, Stefan Duffner, Marc Plantevit, Céline Robardet: In pursuit of the hidden features of GNN's internal representations. Data & Knowledge Engineering, Volume 142, 2022, 102097.
- Luca Veyrin-Forrer, Ataollah Kamal, Stefan Duffner, Marc Plantevit, Céline Robardet: On GNN explainability with activation rules. Data Min Knowl Disc (2022). <https://doi.org/10.1007/s10618-022-00870-z>

Chapter 2

State of the art

Graphs are ubiquitous and natural data structures for representing interactions in social science, electronics, communication, biology, transport, chemistry, linguistics. They are a convenient way to represent all kinds of data. Such structures are studied since Leonhard Euler in 1736. The number of problems considered on graphs is very large and covers classical algorithmic questions, such as the computation of shortest paths or maximum flows, but also supervised and unsupervised classification problems (prediction of graph properties, discovery of communities). Due to the success of numerical approaches on complex data such as texts or images, an approach that is currently widely studied for machine learning on graphs is embedding, a method used to represent discrete variables as continuous vectors. Once the graph represented in a Euclidean space, standard methods can then be used. This numerically very powerful approach, however, loses one of the essential qualities of graphs, which is to be understandable by the end-user. Methods for explaining machine learning models are then developed in order to meet this need for interpretation and validation of methods.

2.1 Graphs

A graph is set of objects (nodes), some of which are interconnected (edges). The *nodes* can store different kind of information: simply an index, a single value, categorical information or a data structure. Similarly, the edges can be directed or un-directed and depict complex interactions with additional information. Some examples of graphs are given below:

- A road graph has nodes that represent road intersections and directed edges stand for streets with a value for the length of the road.
- A molecule is depicted by a graph whose nodes are associated to labels and act for the atoms, and labeled edges represent typed bounds.
- A social network can be seen as a graph whose nodes represent individuals connected by different types of edges, such as friendship (undirected) or following (directed).
- A relational database can also be seen as a graph, with nodes as entities and edges as relations.

More formally, a graph is described as a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with \mathcal{V} the set of nodes and \mathcal{E} the set of edges seen as pairs of nodes. There are two main ways to represent a graph: adjacency matrices and adjacency lists. With the adjacency matrix, the edges are described by a matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ where $A_{i,j} = 1$ if the edge $(i, j) \in \mathcal{E}$ and $A_{i,j} = 0$ otherwise. In the case of undirected graph, the matrix \mathbf{A} is symmetric, and when the edges are weighted, it is common to store the weights directly in the matrix. Adjacency lists $\mathcal{N}(u)$, $u \in \mathcal{V}$ are lists of nodes v such that $(u, v) \in \mathcal{E}$: $\mathcal{N}(u) = \{v \in \mathcal{V} \text{ such that } (u, v) \in \mathcal{E}\}$.

2.2 Machine learning on graphs

As previously mentioned, a lot of data can be naturally expressed as graphs or can be enriched with relational information. If we want to perform classification tasks on this type of data, it is important to define appropriate machine learning models. The main difficulty is the intrinsically discrete nature of these data even though the most powerful machine learning methods require numerical representations. As we have seen, graphs are defined by two sets on which there is no natural order. Thus, we can have two different representations of the same graph just by swapping the indices of the nodes. Testing whether two graph representations correspond to the same graph, named graph isomorphism test, is not known to be solvable in polynomial time¹. This is also the case for most problems derived from the isomorphism test, such as:

- **Sub-graph isomorphism test:** Given two graphs G and H , test if G contains a sub-graph isomorphic to H .
- **Graph Edit Distance (GED):** given G and H , compute the smallest number of modifications to do to transform G into H . The elementary modifications are node or edge addition, suppression or substitution. The GED is probably the most natural and general distance notion between graphs.

Graph mining techniques based on the search for isomorphic subgraphs, such as gSpan (Yan and Han, 2002), have been proposed to identify the frequent subgraphs of a graph. By representing graphs by binary vectors indicating the presence or absence of such subgraphs, classical machine learning models were used 15 years ago to solve classification problems (Borgwardt et al., 2008, Deshpande et al., 2005). More recently, following the path paved by the successes of neural networks to perform classification tasks on complex objects such as texts or images, neural networks dedicated to the classification of graphs have been proposed. As for images and texts, they use convolution mechanisms to directly identify features allowing to discriminate graphs with respect to the considered classification task. The peculiarity here is that the convolution must be independent of a predefined order.

2.2.1 Machine learning tasks

We are interested in machine learning techniques for classifying data in a supervised way. We can distinguish the tasks of regression, where the variable to be predicted is numeric, from

¹Babai (2016) proposes an algorithm to solve graph isomorphism problem in quasipolynomial ($\exp((\log n)^{O(1)})$) time.

that of classification, where we seek to predict categories. On complex data, more specific tasks are considered. For example, on images, classic tasks are object detection, description of images called transcription, or the synthesis of new images. On text data, question answering applications, sentiment analysis, or translation have been considered (LeCun et al., 2015).

In machine learning on graphs, there is a specific set of tasks that encompass link prediction (predicting the existence of an edge between two arbitrary nodes in a graph), node classification (predicting the category of a node in a graph), and graph classification (predicting the category of a graph in a set of graphs) (Xia et al., 2021). Due to the discrete structure and the large dimension of the graphs, these tasks are now carried out using the construction of numerical vectors which embed the graphs (see Figure 2.1) or the nodes (see Figure 2.2) in a lower dimensional and denser Euclidean space. These vectors are built in such a way that the distances between them *express* the topological proximities observed on the graphs.

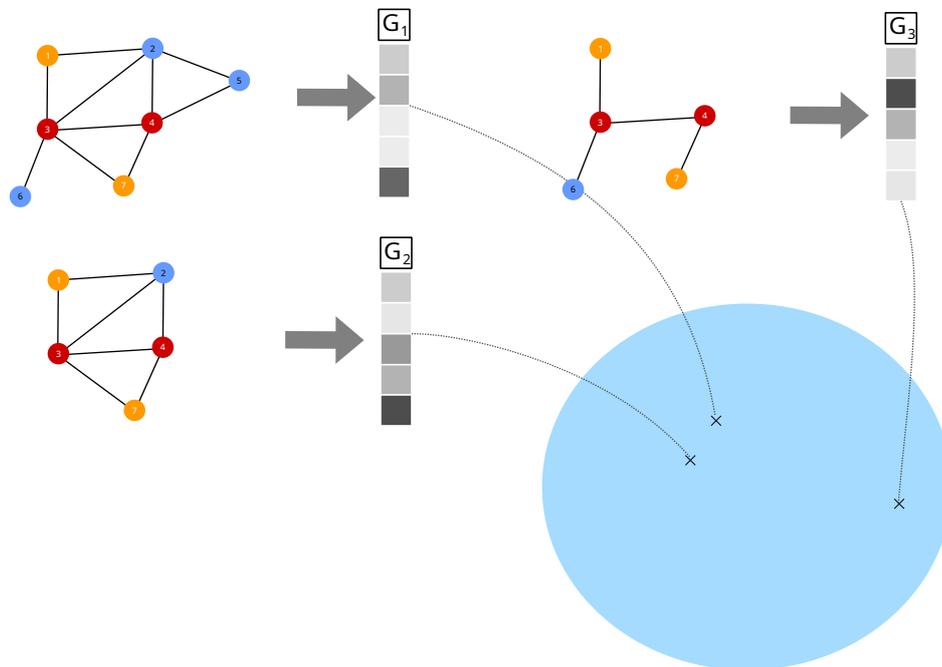


Figure 2.1: Construction of graph embeddings that numerically express the topological structure of the graphs into a low-dimensional space. Distances between vectors represent the topological proximities between graphs.

Link prediction

Link prediction consists of predicting the existence of an edge between two nodes. In the typical use case, we consider a large graph in which we want to discover new relationships between nodes. For example, in biochemical applications where the nodes in the graph represent a set of drugs and the links describe known interferences when used together, the machine learning task is to predict new, yet untested side effects (Zitnik et al., 2018). We can also consider regression tasks, such as for example to predict the electrical flow on a line of an electrical network (Tschora et al., 2022). In many cases, this task is semi-supervised: We

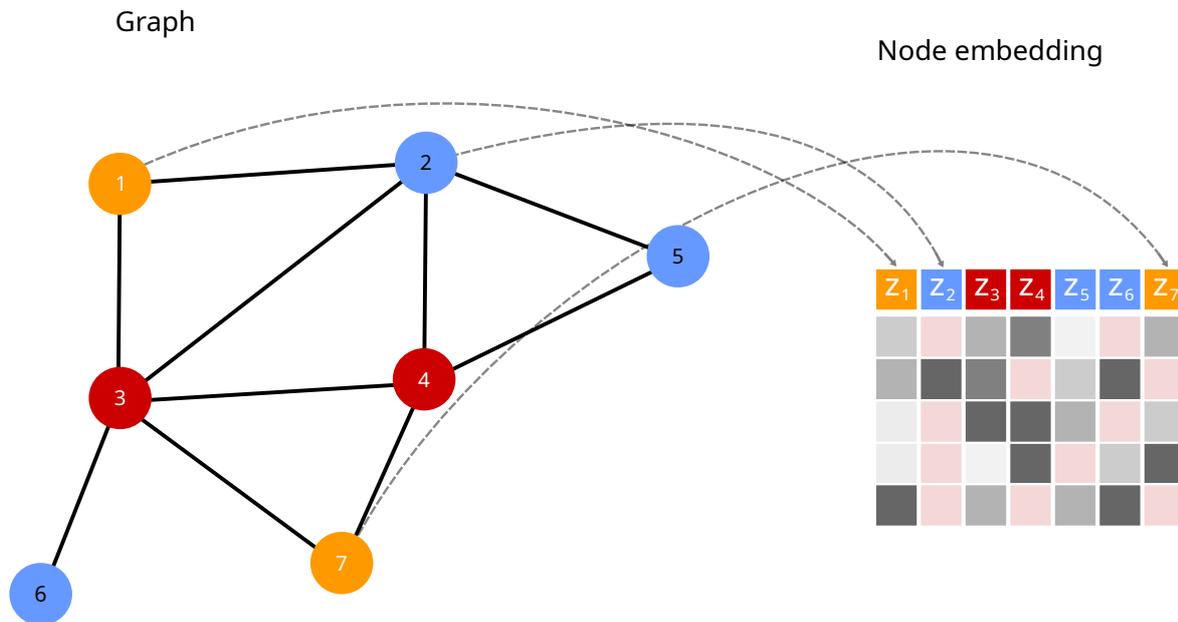


Figure 2.2: Construction of node embeddings that numerically express the topological structure of the node neighborhoods into a low-dimensional space. Distances between nodes represent the topological proximities between the vicinity of the nodes.

know the presence of some edges, the absence of others and we have some pairs of nodes for which we do not know if they are connected by an edge or not. The underlying assumption is that the presence or absence of edges in the graph is not independent and identically distributed on the graph but depends on topological structures in the neighborhood of the nodes adjacent to the edges.

Node classification

Node classification tasks take place on large graphs for which we want to predict the category of a node. For some nodes this target property is known and for others it is not. For example, considering Wikipedia seen as a set of linked web pages, the objective is to predict the topics of the pages (de Melo, 2017). On the IMDb data (Internet Movie Database), we can consider the graph where the nodes are the movies linked by the actors, directors or producers they share. The goal is to predict the movie genre (Yanardag and Vishwanathan, 2015).

Graph classification

The graph classification task requires having a set of graphs for which we want to predict a category. Several applications are in biochemistry, for example the prediction of the toxicity of a molecule or other properties such as mutagenicity (Borgwardt et al., 2005, Dobson and Doig, 2003, Errica et al., 2019, Morris et al., 2020, Wu et al., 2017).

These are the three main tasks of machine learning on graphs. The most recent techniques are based on the computation of an embedding of the nodes or the graphs that can then be

used for downstream processes using other machine learning techniques. In the next section, we review several methods to produce such an embedding.

2.2.2 From graph to vectors

Data analysis methods generally deal with tabular and numerical data. Many studies have thus aimed at associating graphs with numerical vectors. For example, Deshpande et al. (2005) propose a classification algorithm based on identifying frequent subgraphs with gSpan, constructing Boolean vectors expressing the presence or absence of such subgraphs, and then learning a classical classification model from these vectors. Other works, such as (Prado et al., 2013), have focused on the description of each node of a graph using topological properties. They describe the direct neighborhood of a node using measures such as degree, clustering coefficient or number of quasi-cliques (Liu and Wong, 2008) that involve the node. They also consider other properties to characterize a node while taking into account its connectivity to all other nodes of the graph, by computing graph communities and associating to each node the size of its community, by evaluating its closeness centrality, that is to say the inverse of the average distances between the node and all other nodes reachable from it, by measuring the betweenness centrality equal to the number of times a vertex appears on a shortest path in the graph, by calculating the eigenvector centrality measure, that favors vertices connected to nodes with high eigenvector centrality, or by estimating the Page rank index (Brin and Page, 1998). Graphlets, small networks of size up to 5 or 6, have also been used to exhibit fine-grained structural properties that make graphs similar (Charbey and Prieur, 2018).

The problem with such methods is that the properties used to describe the graphs or nodes are determined a priori using human expertise. Moreover, the calculation of some of these properties requires enumeration methods which are costly in terms of computational time. These pitfalls can be avoided by the use of algorithms which automatically construct graph descriptors, either explicitly with embedding methods, or implicitly with graph kernels based methods.

2.2.3 Computing distances between graphs

Many machine learning methods use numerical representations of data to calculate distances. However, distances between graphs can also be computed directly, using kernel methods. The kernel corresponds to a dot product in a high-dimensional feature space. Thus, in this space, estimation methods are linear as long as the computation can be formulated in terms of kernel evaluations, and thus the computation is not done in the high-dimensional space which remains implicit (Hofmann et al., 2008).

Kernel methods are a family of symmetric positive definite functions such as

$$K(g_1, g_2) = \langle \phi(g_1), \phi(g_2) \rangle$$

with ϕ a feature map from graph space to \mathbb{R}^d and $\langle \cdot, \cdot \rangle$ the inner product in a Hilbert space. The goal of kernel methods is to find a kernel function K that does not require to explicitly define the feature map ϕ to compute the similarity between two graphs. They make possible the use of many machine learning techniques such as support vector machines, k-means,

kernel-PCA. Here are some examples of kernels used in graph machine learning (Borgwardt et al., 2020):

- **The random walk kernel** $K_{RW}(g_1, g_2)$ is based on the count of the number of common walks between the graphs g_1 and g_2 . From the product graph g_\times whose nodes (u, v) are such that v has the same label as a node of g_1 and v has the same label as a node of g_2 , there is an edge between (u, v) and (w, z) if u and w are connected by an edge in g_1 and that v and z are also connected by an edge in g_2 (the two edges having the same labels if the graphs have labels on the edges). Paths in graph g_\times are common paths of g_1 and g_2 :

$$K_{RW}(g_1, g_2) = \sum_{(i,j) \in g_\times} \left(\sum_l \lambda^l A_\times^l \right)_{i,j} = e^T (I - \lambda A_\times)^{-1} e$$

with A_\times the adjacency matrix of g_\times , $\lambda < \frac{1}{d_{max}}$ (d_{max} the maximum degree of g_\times), I the identity matrix and e the all-ones vector. However, this method has a high computational cost $O(n^6)$. Some techniques reduce it to $O(n^3)$ (Vishwanathan et al., 2010). Several biases may occur on such a kernel such as the so-called *tottering* bias (Mahé et al., 2004) which is the fact that some paths can visit the same vertices indefinitely, inflating the similarity measure. Moreover, the λ parameter can lead to only consider paths of length 1.

- **The shortest path kernel** compares distances between pairs of nodes in both graphs:

$$K_{SP}(g_1, g_2) = \sum_{e \in \mathcal{E}_1^s} \sum_{e' \in \mathcal{E}_2^s} K_{path}(e, e')$$

where $S_g = (\mathcal{V}^s, \mathcal{E}^s)$ is the shortest path graph of g . The nodes \mathcal{V}^s are the same as those of g and each edge between two nodes u, v is weighted by w_{uv} , the length of the shortest path between them. This graph can be obtained via Floyd Warshall's algorithm. K_{path} is a kernel over paths which can be a dirac function taking the value 1 if the paths have the same length. It can also compare the endpoint labels of paths, or be built on other sub-kernels on edges.

- **The Weisfeiler-Lehman kernel** is based on the Weisfeiler-Lehman algorithm (Weisfeiler and Leman, 1968) designed in the late 60's to approximate isomorphism tests. Its key idea is to assign to each node a tuple with node's old compressed label and a multiset of the node's neighbors' compressed labels. The resultant tuple is then compressed into a new short label. This relabeling procedure is then repeated until reaching a fix-point:

1. To begin, C_u^0 is initialized with the node type for each node u .
2. At iteration i , for each node u , we set L_u^i to be a tuple containing the node's old label C_u^{i-1} and the multiset of compressed node labels from the previous iteration of all neighboring nodes of u : $\{C_v^{i-1}, v \in \mathcal{N}(u)\}$.
3. Then, C_u^i is set to the hash of $L_u^i = \left(C_u^{i-1}, \{C_v^{i-1}, v \in \mathcal{N}(u)\} \right)$. Any two nodes with the same label get the same compressed label.

4. Partition the nodes in the graph by their compressed label. Repeat 2 and 3 until reaching a fix-point.

This method is applied in parallel to the two graphs. The algorithm may be terminated early if the sizes of node's partition diverges between the two graphs. In this case, the graphs are not isomorphic. But it may happen that non isomorphic graphs appear as isomorphic.

The final value C_u^i can be used as graph kernel (Shervashidze et al., 2011). We can also go further and build the Weisfeiler-Lehman-Subtree kernel (Shervashidze and Borgwardt, 2009) by seeing each C_u^i value as defining a graphlet. We then construct a vector v_g such that $(v_g)_k$ represents the number of nodes having the k^{th} value in the set $\{C_u^i, u \in \mathcal{V}\}$. Kernel K_{Tree} is then defined by:

$$K_{Tree}(g_1, g_2) = \langle v_{g_1}, v_{g_2} \rangle$$

This method makes it possible the rapid computation of kernels on a set of graphs. It offers a good compromise to graphlet method by considering only a subset of graphlets, while being fast to compute. One of the main disadvantages here is that we have no connection between the different computed labels, some coming from similar sub-trees may not be taken into account.

Even if it is possible to calculate high-quality distances between graphs, which makes it possible to use classic machine learning techniques, the resulting data has as many dimensions as there are graphs in the dataset and the representation space is not compact. One way to represent graphs in a compact numerical space is to use deep learning methods that provide embeddings, a representation of discrete data as continuous vectors. An embedding is a low-dimensional representation of high-dimensional data. Typically, an embedding does not capture all of the information contained in the original data, but a sufficient part to solve the problem at hand. Deep learning models for graphs are called Graph Neural Networks.

2.2.4 Graph Neural Networks

Graph neural networks are a family of machine learning algorithms that aim to embed nodes or graphs in Euclidean space. Tasks such as graph classification, node classification, link prediction are based on these embeddings like any standard deep learning tasks. The main idea of GNN is to apply the framework of neural networks on graphs and to take advantage of all the theories and methods of this field. However, some specificity of graphs complicates the use of neural networks. On data with a given vicinity such as texts, images or sequences, the use of convolution or recurrent networks has shown very high efficiency. These techniques require aggregating neighborhood data. However, in graphs, the neighborhood of a node is a set whose size varies from one node to another and which has no intrinsic order relation. Thus, the convolution operation must be redefined so as to be invariant of the neighborhood order and size. An option could be to use the hash function defined by the Weisfeiler-Lehman algorithm because it respects such conditions. However, this operation is not differentiable and therefore prevents optimization by gradient descent.

GNNs are designed to embed each node of the graph in Euclidean space \mathbb{R}^d , d being an hyper-parameter. They operate layer by layer by aggregating the previous layer embedding

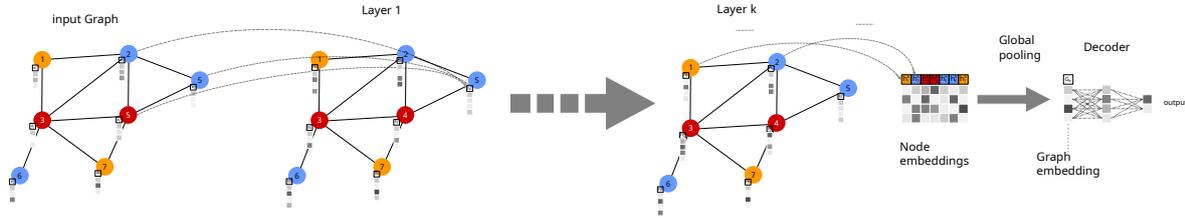


Figure 2.3: GNN embedding process: each hidden vector associated with a node is updated based on the vectors of its neighbors. The node embedded vector can then be directly used for node classification. Otherwise, for graph classification, all the vectors associated to the nodes of the graph are aggregated before the classification step.

vectors of neighboring nodes. GNNs work in such a way that their k -th layer maps the k -hop neighborhood of each node u , also called ego-network of u of diameter k , to a vector h_u^k (see Figure 2.3). The GraphSAGE GNN model alternates three operations (Hamilton et al., 2017):

- **Message passing or aggregation** is the function that combines neighbor vectors. This function maps a set of vectors in \mathbb{R}^d to a single vector in \mathbb{R}^d such that the new vector is invariant to the order of the input vectors and that it can take a variable number of inputs. The sum function is often a good candidate. The aggregate function is defined by:

$$h_{\mathcal{N}(u)}^{(k+1)} = \text{AGGREGATE} \left(\{h_v^{(k)}, \forall v \in \mathcal{N}(u)\} \right)$$

- **The linear application** is the application of the parameters of the machine learning model on the result of the aggregation function concatenated to the vector $h_u^{(k)}$ associated to u at the previous layer. It generates a vector of dimension d :

$$m_u^{(k+1)} = \mathbf{W}^{(k+1)} \cdot \text{CONCAT} \left(h_{\mathcal{N}(u)}^{(k+1)}, h_u^{(k)} \right)$$

with \mathbf{W} the weight matrix of dimension $(d \times 2d)$.

- **The non linear activation function** applies a nonlinear transformation to the node representations. Generally, the Rectified Linear Units (ReLU) is used as it speed up training:

$$h_u^{(k+1)} = \sigma \left(m_u^{(k+1)} \right)$$

with $\sigma(x) = \max(0, x)$, the ReLU function.

The combination of these operations corresponds to the following matrix calculation, adding a self-loop in the adjacency matrix and using a weight matrix of dimension $(d \times d)$ with the *sum* as the aggregation function:

$$\mathbf{H}^{(k+1)} = \sigma \left(\mathbf{A} \mathbf{H}^{(k)} \mathbf{W}^{(k+1)} \right)$$

Choice of aggregation function

The choice of aggregation function is quite crucial in GNN. Generally, we choose an aggregation function that is invariant to the permutation of neighbors. But it is possible to use a function that does not satisfy this property, like LSTM (Hamilton et al., 2017) – that processes the inputs in a sequential manner – by simply applying the LSTMs to a random permutation of the node’s neighbors.

In addition, each aggregation may lead to symmetries or undesirable side effects. For example, in a graph containing nodes with high degree and others with low degree, the former may become too important while the latter become insignificant. We can contain this bias by introducing degree normalization in the mean function, or using a pooling aggregation function. Other biases may appear (Xu et al., 2019).

Stacking layers

Layers are stacked either recursively or convolutionally. The recursive way (Gallicchio and Micheli, 2010, Scarselli et al., 2009) consists to apply the same weight matrix for each layer:

$$\mathbf{H}^{(k+1)} = \sigma \left(\mathbf{A} \mathbf{H}^{(k)} \mathbf{W} \right)$$

Banach’s theorem states that when $\sigma \left(\mathbf{A} \mathbf{H}^{(k-1)} \mathbf{W} \right)$ is contractible then the above equation reaches a fixed point. This recurrent approach is historical but has severe drawbacks such as its expressiveness which is limited and its computational cost that is high. Moreover, even if Banach’s theorem applies individually for each vector, all vectors may also converge all together towards a common fixed point.

In Graph convolutional Networks (GCNs) (Kipf and Welling, 2017), vectors are aggregated using convolution that differ from one layer to another (and therefore for each size of ego-networks). This is done at the cost of additional weights to be estimated, but with a well-defined inference calculation time. In GCN, $h_u^{(k+1)}$ is defined as

$$h_u^{(k+1)} = \sigma \left(\mathbf{W}^{(k+1)} \sum_{v \in \mathcal{N}(u)} \frac{h_v^{(k-1)}}{\sqrt{|d_u| |d_v|}} \right)$$

with $|d_u|$ the degree of u . This convolution allows an explicit matrix formulation as well:

$$\mathbf{H}^{(k+1)} = \mathbf{D}^{\frac{1}{2}} \hat{\mathbf{A}} \mathbf{D}^{\frac{1}{2}} \mathbf{H}^{(k)} \mathbf{W}^{(k+1)}$$

with D the diagonal degree matrix and $\hat{A} = A + I$ the adjacency matrix with self loops added. With this architecture, the embeddings of each layer are clearly linked to the corresponding ego-networks.

After several layers of this architecture, around three in general, we have for each node an embedding which corresponds either to the last hidden vector, or to the concatenation of all the hidden vectors. The vectors associated with the nodes are used as input to a decoder, a multilayer perceptron (MLP), which makes it possible to perform the task of classifying nodes, graphs, or predicting edges.

Node classification

For the classification of nodes, the GNN is followed by a simple decoder, a Multi Layer Perception (MLP), which takes the embedding vector of the node built by the GNN and associates a vector corresponding to the classes to be learned. This step can be formalized by:

$$z_u = MLP(h_u^K)$$

with h_u^K the node embedding produced by the GNN on the last layer. The parameters of the MLP are adjusted using the following loss function on the training set \mathcal{V}_{train} :

$$\mathcal{L} = \sum_{u \in \mathcal{V}_{train}} -\log(\text{softmax}(z_u, y_u))$$

with y_u the ground truth class scalar value of node u , and softmax defined by

$$\text{softmax}(z, c) = \frac{e^{(z_c)}}{\sum_{i=0}^{|z|} e^{(z_i)}}$$

where all the z_i values are the elements of the input vector and can take any real value. The term on the bottom of the formula is the normalization term which ensures that all the output values of the function will sum to 1.

Graph classification

When GNNs are used for graph classification, an aggregation step is needed to go from a set of node representations to a graph embedding. The function described above can be used. Other strategies exist such as the combination of max or mean pooling, or also clustering pooling (Ying et al., 2018) where nodes are clustered to form a smaller graph on which new embeddings are learned until the entire graph is reduced to a single node (Zhang et al., 2018).

A decoder of the same type as the one used for node classification is then used, taking as input the vector representing the graph and producing a vector indicating the predicted class. Other machine learning techniques can be applied directly from the vectors representing the graphs.

2.2.5 Connections with Weisfeiler-Lehman algorithm

Graph Neural Networks and the Weisfeiler-Lehman algorithm (WL-algorithm) are similar (Balcilar et al., 2021), and it has been proven that a GNN is no more expressive than WL-algorithm: by considering each layer of the GNN as a hash function, we obtain the same algorithm. However, for the GNN, this function is not necessarily injective depending on the aggregation function used. This has multiple consequences:

- The GNN are computationally less expressive than WL-algorithm, but the theoretical framework holds.
- Different input graphs may have the same embedding through the GNN, that can be due to equivalent classes described in WL-algorithm, or due to some loss of information in the aggregation. If such a GNN-isomorphic graph occurs in real life, it may be a limitation in the use of GNN and WL-algorithm.

- Due to the Neural network architecture, GNN can take continuous data as input features whereas WL-algorithm can only have discrete node attributes.

GNN cannot be more expressive than WL-algorithm, nevertheless they are similar machines. GNNs are much more convenient for learning on a specific task and benefits from all the neural network framework and tools. WL-algorithm cannot work on subgraphs that have not been seen before: if a new ego graph is considered, a new symbol has to be created and the kernel has to be rebuilt. On the other hand, a GNN will build a new embedding, close to the similar ego graphs previously found.

GNN has been a popular model for machine learning on graphs for a few years, providing state-of-the-art results in many tasks. They bring neural networks to graph machine learning tasks, with its advantages – efficiency, performance, research framework – but also its pitfalls: such models are mostly black boxes that are un-interpretable. Older kernel methods are much more interpretable because their inner workings come from common graph operations such as shortest path computation or subtree decomposition.

The need for explainability for graph machine learning tasks also comes from the use cases. For example, in drug discovery, explaining why two molecules interact is even more important than predicting it. GNNs are also used in flow problems such as power grid calculation, and guarantees are needed for this kind of applications. Also, in social media, the detection of fake accounts can benefit from explanations in order to support the decisions.

2.3 Explaining machine learning models

Although Graph Neural Networks achieve state-of-the-art performance in many tasks, this new deep learning approach relies on complex data transformation flows that involve new questions not yet considered by other explainability methods for learning algorithms. Examples of such questions are: Is the structure of the graph responsible for the decision? Or is it the attributes of the nodes which are preponderant in the classification process? Before considering the related work specific to the explanation of GNNs, we consider more generic methods that aim to explain machine learning models.

2.3.1 Why do we need explainable artificial intelligence?

Explainable AI is a broad, cross-cutting subfield of machine learning that has grown in popularity with the emergence in recent years of deep models. Deep neural networks are inherently opaque models and explainability methods aim to give more confidence to the model beyond empirical risk. Older machine learning models, such as decision trees, linear regressions, K-Nearest Neighbors, or rule-based learner are transparent models by design. They require little or no work to understand. Most new models are much harder to understand and are considered black box models: we cannot directly understand their inner workings.

Black box models

When we try to explain a black box model, we only look at the input and output of the model, refraining from working on the internal algebra and operation of the model. However, we gain in generalization, since such a method can be applied to any model.

Several families of machine learning algorithms are widely recognized as being black boxes (Burkart and Huber, 2021):

- Perceptron: Whereas perceptrons composed of a single layer are transparent models, with multiple stacked layers they become black boxes as their inner workings become much harder to understand. Multilayer neural networks, deep convolutional and recurrent neural networks are considered black boxes as well.
- Tree ensembles, and generally boosting techniques, tend to transform white box algorithms (as decision trees) into black box ones.
- Kernel methods, even applied to a transparent model such as k-nearest-neighbors, can be opaque especially when some learned parameters are involved.

A black box model is generally chosen for its better prediction performance. So, in the case of a real application, we might face a trade-off between better performance or transparent design. Would you let an AI model decide on a patient's prescription, even knowing it would be the correct one 99.99% of the time? Would you let an AI system manage the shunting of trains with only 0.0001% risk of having trains collide²? Or managing the power grid where one mistake can lead to blackouts across much of the country, with dire safety and economic consequences. We need to know more about these models than empirical risk.

Explainable AI provides taxonomies of tools, definitions, and ultimately algorithms to dig deeper and better understand how machine learning models work. The approaches differ according to the issues addressed: What is the use case? What do we want to know and when? Is this in a production environment or only in the development phase for debugging? Do we just want to check the overall behavior of the model? Who do we want to bring this knowledge to, doctors, expert engineers, the general public or data scientists? Second, what properties of our model do we want to verify by our approach? Do we just want to get confidence of our model or do we want to know more about the causalities between input and output? Do we want to ensure fairness when it comes to data bias? Thus, one can choose a model and a post hoc explainer or directly use a transparent model. Some adaptations may also be necessary, or in some cases come up with a new method from scratch.

Use cases

The need for explainability differs between use cases. If the model is built for a domain expert such as a doctor, an engineer, a bank advisor, then the machine learning model can be used as a decision aid. Even though the model is better and faster than the human decision, the decision cannot be taken on an oracle. We need evidence, explanatory elements of the decision. The experts in the loop can bring their knowledge of the domain. An explainable system should provide to the experts the important elements and leave them responsible for the decision.

In other cases, we want to design a machine learning model that ensures certain properties defined by an expert. It is therefore necessary to create models that can integrate specific knowledge. The type of properties may vary from one application to another.

²This would lead to 1 collision every two months in France.

Next to domain experts providing knowledge, we have statisticians and machine learning experts in general, who might need tools to debug or validate their models, while having little external knowledge. These experts should be more familiar with mathematics, statistics, algorithms, or bias (Tversky and Kahneman, 2008).

Then, the general public has now a right to explainability of processing and decisions with the GDPR. It requires that any automated decision-making about personal data be explained, based on machine learning or not. End users need explanations that they can understand, regardless of their education and social background. Explainability is also important for users to accept decisions. For example, an electric car can advise the user to recharge the battery at a certain time and provide information that it has learned user's driving habits and taken into account the price of electricity throughout the road.

Regulatory entities may also impose explanatory requirements. It can be the law as in the GDPR, but also an ISO standard to follow. The aim is to certify the conformity of the model to the law or to a standard. For example, an AI-enhanced flight assistant pilot must maintain the aircraft within the flight envelope to be validated by the FAA³. In such cases, the explainability system must provide strong evidence about the system and not just sufficient confidence.

Finally, there are derivative uses of the methods of explanation, which apply explainability methods for different purposes. For example, single instance explanations applied on image models can be used as object detection, especially if they are computationally inexpensive. A global explainer can also be used by scientists to learn patterns on a research task. Black box explanation methods can be used to learn and simulate a hidden model for reverse engineering purposes. In this case, we take advantage of the links between explainability and transfer learning, learning a surrogate model being a kind of transfer learning.

Identifying the users of the explainability system, their needs and constraints, and the type of expected result is the first step in an XAI process. The second step is to find the desired properties of the explanations.

Expected properties

The various works of XAI have shown that several different objectives can be achieved for different purposes, approaches and target audiences. In general, we want to provide additional guarantees beyond the statistical risk. We can define a set of desired qualities that we want to achieve from the machine learning workflow with the explainability methods.

- **Trust:** This is perhaps the most general goal of XAI. It represents the confidence we can have in the model to perform the task. This quality identifies the conditions under which the model is strong or weak.
- **Causality:** Most machine learning algorithms look for correlations but not causalities. Can the model express some causality between input and output, taking into account prior knowledge about the problem? For example, given the number of persons on a train platform and the presence of a train, one might conclude that the more people there are on the platform, the higher the likelihood that a train will arrive. But from common knowledge, we know that the reality is different. For testing causality

³Federal Aviation Administration – Agence européenne de la sécurité aérienne.

relationships, we either want to be able to highlight certain causal relations identified by the model and to confront them with prior knowledge, or to test or apply some known causal relations on the model.

- **Transferability:** This property evaluates a model's ability to generalize its learning, its ability to handle more recent data when deployed in production. This property is sought by any model designer, even without considering the problem of explainability. A common problem in the industry is that the machine learning algorithm is learned on local data. Then, when deployed in production, the environment differs and there is a loss of performance. Having a better knowledge of the domain learned helps transferability by giving, for example, bounds on the input data.
- **Fairness:** The data is often biased or unbalanced, a property that transfers to the model. We want to highlight and visualize these biases in the model or in the data, and be able to correct them based on prior knowledge or a target goal. This can be very important when making decisions about people in order to ensure ethical and fair use of algorithm outputs.
- **Informativeness:** The method should give information about the task because the use case of the model can be broader than that considered during the training. The goal is therefore to extract more information about the problem and the internal relationship of the data inside the model, and to give the user a broader representation of the problem.
- **Privacy:** While the previous qualities tend to give insight into the inner workings of the model, in many critical applications, such as for medical data, we don't want to provide personal information while still providing enough evidence about the behavior of the model. Moreover, while guaranteeing the GDPR, we might have to give indications of the decision of the models, without disclosing too much information about the model in order to keep the trade secret.

While other qualities of XAI methods have been considered, we focus here on the most important ones. When designing an XAI method, we need to be aware of these qualities and assess how well we fulfill them.

Type of methods

Once the need for explainability is known as well as the desired properties, we can choose an explainability method. Several families of methods have been designed for different purposes and use cases. There are single instance explainer or model explainer methods, ad-hoc and post-hoc explainers, as well as model-specific or model-agnostic methods. Visualization techniques can also be used for this purpose.

- **Transparent models:** Some models are inherently interpretable (Freitas, 2013), such as decision trees, linear models, rule-based systems or Bayesian networks (Charniak, 1991). Their interpretability is based on their size and the simplicity of understanding their inner workings. However, the interpretability of these models is guaranteed only under certain conditions of their learning procedure. Some learned models will be

easier to interpret than others. It can then be necessary to add external knowledge to such models, in particular for decision trees or rule-based models, to obtain guarantees on their behavior. Unfortunately, transparent models are often outperformed by the deep neural networks, especially for image and text classification. Transparent models are of interest in critical applications, such as medicine or civil engineering, where life is at risk.

- **Global model explanations:** Explaining a global model consists in considering the already trained black box model, and aim to analyze its behavior over the entire input space. Several strategies have emerged for this task with different goals in mind, but the main objective is to build a simpler and more interpretable model that approximates the output of the first model. This second model should perform better than if it was built using the target variable, otherwise we would have to directly learn the interpretable model on the dataset. As a by-product, it should behave like the original model also in its errors. The construction of the explanatory model can be done in two ways, first by rewriting the model in the formalism of the other, in the same way that a compiler would do it (Andrews et al., 1995, Craven and Shavlik, 1994). The main disadvantage is that these methods are very specific to the type of model considered. Other methods train the interpretable model by sampling data in the input space (Frost and Hinton, 2017), either from labeled or unlabeled data, or using data generated by a GAN or other technical (Goodfellow et al., 2014, Hinton et al., 2015). In such cases, the surrogate model is meant to be used in production, its decision may look like the black box model, the decision process is on another paradigm, so the explanation would not make the same sense. In many cases, constructing a surrogate model is not really possible. For example, the search for structured patterns, as performed by convolutional neural networks, cannot be emulated by a small set of rules or a decision tree that sees a complex piece of data as a set of independent data.
- **Local explanations:** Global model explanation may be a problem too difficult to solve, either because surrogate models are not expressive enough for the task, or because they are too model-specific. Of course, the global explanation allows a fortiori an explanation for a single instance, but it can be imprecise. Single instance explanations seek to produce an explanation specific to an input example. Several paradigms offer different types of explanations and approaches.
 - *Gradient and backpropagation methods* are based on computing the gradient of the model on the input example. Such methods require the model to be differentiable. Fortunately, this is often the case. Such methods give insights into how the model behaves locally, but still require some work to be truly usable: Many input features have little impact and should be ignored. Non-convexity of models, such as deep neural networks, can cause them to fail (Selvaraju et al., 2017, Simonyan et al., 2014, Zhang et al., 2016). Other methods such as DEEPLift (Shrikumar et al., 2017) and LPR (Bach et al., 2015) are similar in that they use backpropagation, but differ in that they do not use the gradient but the difference with a reference input.
 - *Surrogate methods* build a model that approximate the original model in the vicinity of the example. The model is easier to fit locally and can provide much detailed

- explanations specific to the instance. To build the surrogate model, a new dataset is required. First, some instances are selected in the dataset such that they are close to the instance to be explained. However this sub-dataset might be too small or cover a too small subspace of the data to build a high quality model. Such dataset can thus be augmented with new examples generated by adding noise (Ribeiro et al., 2016).
- *Feature mask methods* determine how much each input feature contributes to the model’s output decision. We can also know if this contribution is positive or negative. The most successful method in this respect are the Shapley values (Lundberg and Lee, 2017, Shapley, 1953). It is a tool built on the framework of game theory which allows to decompose the contributions of the input characteristics. The L2X method (Chen et al., 2018b) proposes to build a mask on the input which optimizes the mutual information between the original input and the masked input. Moreover it proposes to learn a neural network which allows to reconstruct a mask because the sampling process can be expensive. In this case, the neural network is trained on the model but is only useful for explaining the instance.

Single instance explanations can be more precise than global model explanation. However, this usually has a higher computational cost, as producing a single explanation can be relatively expensive. For example, LIME needs to draw a dataset and learn a model on it. The cost of gradient back-propagation is always higher than looking at a decision tree. But in many use cases, these costs will still be negligible compared to the cost of the human looking at them. Different methods will cover different needs. Gradient-based methods are likely to be very good at finding a counterfactual of an input – a close input that is classified differently – and will be relatively fast. However, this entry might not be realistic. Other methods have been devised to achieve this goal (Dandl et al., 2020, Mothilal et al., 2020). The LIME explainer will be useful to give a local view of the model, helping to see the limit of the model’s decision. The L2X method will provide sets of features that act together to achieve a decision.

Model-agnostic versus black-box models explanation methods

In the field of explainability of machine learning models, we generally distinguish between explanation methods for black box models and so-called agnostic explanation methods, which can be applied to any model. The black box concept refers to the internal complexity of the model, which obscures the decision-making mechanisms. The need for explainability comes from the non-understanding of these internal mechanics. On this type of models, we can obviously use agnostic methods, which only use the output of the model to construct the explanations. There are also methods that use internal mechanics, such as computing the gradient (Selvaraju et al., 2017) to produce explanations. The notion of agnosticity of the method of explanation with respect to the machine learning model can be seen as a gradual scale, which encompasses more or less models. For example, requiring the model to be differentiable encompasses many models. The requirement to be able to perform gradient descent restricts the application to neural networks.

2.3.2 Evaluate an explanation method

How to evaluate an explainer? Should an explainer be trusted (Camburu et al., 2019)? Ensuring the interest of an explainer depends on qualitative and subjective arguments. We discuss in this subsection on what points an explainer can be evaluated and what criteria ensure that a new explainer is better than another.

For any explainability task, there is no definite optimal solution: different elements can explain a model. There is no fundamental truth. For example, for a model trained to recognize a cat in a photo, the expected result of an explainer should not be the area where the cat is located, because this is not an invariant of the photos. The model is expected to have identified the cat's ears, paws or fur. But what should the explainer show when there is no cat in the picture? A model explainer could identify features that the model identifies as belonging to a cat and not show any of them, when the image does not contain a cat.

Most explainability methods seek to define a metric to be optimized and provide qualitative arguments to assess that the metric provides good explanations. They also show experimentally that the optimization scheme is efficient. For example LIME (Ribeiro et al., 2016) defines a lasso model whose optimization provides a local approximation of the model around the instance to be explained. The weights of the lasso model are used as an explanation. For a single instance explanation, this schema is one of the most used (Chen et al., 2018b, Lundberg and Lee, 2017, Ribeiro et al., 2018, Shrikumar et al., 2017, Zhang et al., 2016). This is the minimal explanation format we can work on: a weight associated with each input feature. However, for different methods these weights can mean different things. Sometimes they are independent probabilities – each weight is an independent measure with a value between 0 and 1 – or they form an overall probability distribution where all the weights sum up to 1. Sometimes they are considered as weights in a local model. To compare these different approaches, there is unfortunately no universal measure.

Comparison of explanations with ground truth

Some datasets come with a selection of features that can be considered as ground truth (Debnath et al., 1991). Others are synthetically generated to hide a ground truth (Ying et al., 2019). In such dataset, explaining why an instance is in class *A* rather than class *B* is known. However, this is an explanation of class membership and not an explanation of why the model made this decision. Sometimes, in a synthetic dataset, the way of generating the examples induces a certain bias which can be identified by the model and also by the explainer. Synthetic datasets are usually too simple and do not correspond to real use cases.

Experimental validation by crowdsourcing

Explainability is about providing information to humans. Therefore, crowdsourcing validation is a good way to gauge an explainer's interest. Even though this is a subjective assessment, a large number of responses from different people tends to minimize the subjectivity of such a validation scheme. However, this approach has several pitfalls (Fürnkranz et al., 2020). It measures how well a human will accept the explanation, not directly the performance of the explanation. It is necessary to design experiments such that these two objectives have the maximum common points. Variations in results with different use cases and audiences can be investigated. For example, on molecular data, biochemists will be

much more demanding than ordinary people. Finally, these experiments are relatively expensive, and can be used to validate a method. However, it is less straightforward to compare different methods with this approach.

Fidelity, Infidelity and Sparsity metrics

The most common way to evaluate an explainer is a set of specific metrics that measure the impact of features/areas identified by the explainer from just the original model and input. There are three main metrics, named Fidelity, Infidelity and Sparsity, to assess the quality of an explanation. Fidelity assesses how the model changes its prediction when we remove the features identified as explaining the decision. Infidelity assesses whether the instance, reduced to the characteristics identified as explaining the decision, is classified in the same way as the original example by the machine learning model. The Sparsity measure assesses the relative size of the explanation to the original input:

- **Fidelity** measures the variation in the output value of the machine learning model when removing the features identified as explaining the decision. A high fidelity value means that removing the features explaining the decision changes the prediction, and thus these features are important for the model. However, considering only Fidelity without taking into account the Sparsity can be misleading: removing all features can lead to change the decision without providing any explanation.
- **Infidelity** measures the variation in the output value of the machine learning model by considering only the features identified as explaining the decision as input to the classifier. Having a low value in Infidelity ensures that the explanation is relevant. For example, for the prediction of a disease, if the explanation contains only the age of the patient, it can have a high score in fidelity but a bad score in infidelity, as age alone not being decisive in predicting the disease.
- **Sparsity**: When evaluating two explainers, it is important to verify that they have similar sparsity values. It can even be used as a parameter, especially when binarizing continuous masks. Sparsity can help adjusting masks for a fair comparison.

2.4 Explaining Graph Neural Networks

Explaining graph neural networks presents multiple challenges. The most important one is probably to consider both node information and structural information. For example, the gradient method tends to confuse this information. Another challenge is to have methods capable to handle multiple types of graph data and classification tasks.

We can classify GNN explanation methods according to the classification task for which they were designed. Indeed, some methods are designed for the classification of nodes, edges or graphs. But, in most cases, the GNN consists of a node encoder followed by a pooling layer or a decoder, depending on the classification task. Thus, a large part of the models are common to all GNNs. Yuan et al. (2020b) suggest classifying explainable methods as follows. They first consider whether the method is designed to explain a single instance or a model. Next, they consider the type of explanation produced: Gradient Methods,

Perturbation-Based, Surrogate models or Decomposition approaches. In most cases, state-of-the-art methods for explaining GNNs are adaptations of existing work dedicated to the analysis of neural network models for complex data such as images or texts.

What differs between instant-level methods is the family of questions answered by the explainer. Gradient-based methods identify which part of the examples can profoundly influence the decision. Perturbation-based methods aim to find the parts of the examples that are classified as the original input. Surrogate methods attempt to express the model in a classifier that is more interpretable than the original model in the vicinity of a given example. Decomposition methods seek to quantify the contribution of each input feature. Since there are very few model-level explanatory methods, no further subdivisions are relevant yet.

2.4.1 Gradient-based methods

Gradient-based methods can be used for explaining any differentiable model. They consist in understanding the behavior of the classifier by looking at the input gradient which is the derivative of the output with respect to the inputs. This means that for any small change in the input, the gradient tells us how the output would change. If the gradient is large, a small change in input dimension has a large effect on the output. If the gradient is small, the effect is small. In the case of linear classifiers, the gradient corresponds to the feature weights. These methods are popular for analyzing image classifiers, offering easy-to-understand explanations. The main difference between the methods in this category lies in the way the gradient is back-propagated. The main interest of such methods is their simplicity to implement and understand, especially for people with scientific and mathematical background.

Some works have directly reused methods developed for the analysis of image classifiers on GNNs using the conceptual proximity between CNN and graph convolutions. Pope et al. (2019) propose to compare several methods such as GRAD, GRAD-CAM and Excitation Back-Propagation (Selvaraju et al., 2017, Zhang et al., 2016, Zhou et al., 2016). These methods create heatmaps on the input features and visualize the importance of the features when classifying the input example. Heatmaps can be thought of as continuous masks. The methods are used with little or no adaptations from the original articles. Fidelity, contrastivity and sparsity are used as evaluation measures. The contrastivity is specific to this type of methods and measures the difference between the masks.

This work provides a good basis for GNN explanatory work. The methods are nevertheless quite naive and have several limitations:

- The main weakness of CAM and GRAD-CAM is the necessity to use an average pooling layer after the convolutional layer for node embedding, designed with a fixed graph structure.
- Even if experiments show good contrastivity results, demonstrating the identification of distinct components, the fidelity and sparsity are low. This is partly due to several choices in the experimental setup.
- There is no consideration of the structural information of graphs as the gradient only considers the features associated to nodes. The end-user may guess the structural influence on the decision.

- There is no experiment with node classification tasks, while in such cases the pooling layer become trivial.

Gradient-based methods are a very economical way to produce explanations. However, these explanations are more appropriated to generate adversarial perturbations. Indeed, the gradient gives the minimum modification on the input which leads to the maximum modification of the classification (Ancona et al., 2018, Szegedy et al., 2014). These methods were also used for global-model explanation in order to evaluate the marginal influence of individual characteristics in the classification. However, in the experiments, no feature was found to influence the classification. This can be due to the fact that these methods cannot see the structure of the graph. This is a major limit in the interest of these methods for GNN explanation (Simonyan et al., 2014).

2.4.2 Perturbation-based methods

Perturbation-based methods work the opposite of previous methods by finding areas where the prediction does not change. Such methods typically work top-down by learning a mask generator to select important features for the decision. For example, Chen et al. (2018b) optimize the mutual information between input and output features. The search space for maximizing this function is exponentially large and smart heuristics are needed to get good performance in an acceptable time. Several works have been done in that direction (Luo et al., 2020, Schlichtkrull et al., 2021), the most popular explainability method for GNNs being probably GNNExplainer (Ying et al., 2019).

GNNExplainer

GNNExplainer (Ying et al., 2019) is one of the main works for explaining GNNs. It is based on the model explanation principle developed by (Chen et al., 2018b), but adapted to the architecture of a GNN in order to produce single instance explanations. It uses mutual information as main optimization metric. GNNExplainer generates a mask on edges and nodes features.

Considering G the input graph and X the associated features, the objective is to identify a subgraph $G_s \subseteq G$ and a subset of features X_s which are important for the GNN decision Y . Let denote by Φ the GNN model that learns the conditional distribution $P_\Phi(Y|G, X)$.

First, G_s has to be found, and for that the mutual information measure is used:

$$\max_{G_s, X_s} MI(Y, (G_s, X_s)) = H(Y) - H(Y|G = G_s, X = X_s)$$

with $H(X) = -\mathbb{E}[\log P_\Phi(X)]$ the entropy measure. MI quantifies the change in the probability of prediction $\hat{Y} = \Phi(G, X)$ when the graph is limited to the explanation subgraph G_s and the node features are limited to X_s . We can observe that $H(Y)$ is a constant because Φ is fixed for a trained GNN. Therefore maximizing the Mutual Information is equivalent to minimizing

$$-\mathbb{E}_{Y|G_s, X_s} [\log P_\Phi(Y|G = G_s, X = X_s)]$$

As we might expect, finding the best G_s, X_s for MI is not tractable in general. GNNExplainer finds an approximation of this optimal by a continuous relaxation of the problem for the adjacency matrix: $A_s \in [0, 1]^{n \times n}$ instead of $\mathbb{B}^{n \times n}$ and enforce $A_s[j, k] \leq A[j, k]$. It is considered

as a random graph that follows a multivariate Bernoulli distribution $P_{\mathcal{G}}(G_s) = \prod_{j,k \in G} A_s[j, k]$. Thus, GNNExplainer optimizes the following objective using gradient descent:

$$\min_M - \sum_{c=1}^C \mathbb{1}[y = c] \log P_{\Phi}(Y = y | G = A \odot \sigma(M), X = X_S)$$

where $A \odot \sigma(M)$ represents the masking of the computed graph of adjacency matrix, with $M \in \mathbb{R}^{n \times n}$, \odot denotes element-wise multiplication, and σ denotes the sigmoid that maps the mask to $[0, 1]^{n \times n}$.

The method adds several constraints such as a limit on the number of edges and features selected and also constrains the explanation graph to be connected.

The experiments first show good results on synthetic datasets for the explanation of nodes and graphs, outperforming naive methods like attention methods (Velickovic et al., 2018) and a simple gradient-based method. Then two real datasets are considered: Mutag (Debnath et al., 1991) and reddit-Binary (Yanardag and Vishwanathan, 2015).

However, this method has several limitations. Like most optimization methods in Deep Neural Networks, the convexity assumption is violated, leading to a lower bound on the performance explanation, and results in local optima in the explanations. For example, explanations should be connected and not too large, these constraints may be fine for node classification tasks, but for graph classification, several disconnected parts of the graph may constitute a good explanation. Also, continuous masks can lead the GNN to unexplored areas during training. This unstable behavior is especially true when labels and features are discrete on graph nodes.

It is regrettable that the authors did not show a useful use case for feature selection. The experiments deal with molecular data where the nodes are atoms whose type is encoded by a one-hot vector, so that the feature selector retains only the component corresponding to the atoms found in the explained molecule. More experiments with richer feature vectors might yield more insight into the performance of the feature selector. The explanation language is only made of masks and only handles the absence of elements.

PGExplainer

PGExplainer (Luo et al., 2020) is quite similar to GNNExplainer but offers some improvements and changes. First, it does not consider node feature information arguing that there are many methods on this type of tabular data that should be used. They seek to optimize the same mutual information function. Let G be the original input graph and PGExplainer looks for the following decomposition $G = G_S + \Delta G$ where G_S is the subgraph explaining the decision and ΔG be the remaining task-irrelevant edges of G .

$$\max_{G_S} MI(Y, G_S) = H(Y) - H(Y | G = G_S)$$

PGExplainer uses a multi-layered perceptron to construct its explanations. This perceptron takes the node embedding generated by the hidden layer of the GNN from both ends of each edge and generates a weight $w_{i,j}$. This makes possible to learn the parameters on the whole dataset and not just on single instances. Let $Z = (z_1, \dots, z_n)$ be the hidden embedding of the input graph produced by the GNN. The weight associated with the edge (i, j) is calculated by $MLP_{\Psi}(z_i, z_j) = \omega_{i,j}$. The explanatory graph is then drawn randomly from the weights $\omega_{i,j}$ which make it possible to sample a graph and then evaluate it with the mutual information.

Continuous relaxation is done using the Gumbel softmax trick (Jang et al., 2017, Maddison et al., 2017) to learn MLP_{Ψ} . Then edges are drawn proportionally to

$$e_{i,j} \sim \sigma((\log \epsilon - \log(1 - \epsilon) + \omega_{i,j})/\tau)$$

where ϵ is a uniform random variable in $[0, 1]$, σ is the sigmoid function, $\omega_{i,j}$ the learned parameters and τ a temperature parameter. When $\tau \rightarrow 0$, $e_{i,j}$ acts as a Bernoulli random variable while having a well defined gradient. The authors also propose to use some constraints as a limit size and to be connected for the explanation graph G_S to be a satisfactory explanation.

In experiments, the authors show that PGExplainer outperforms GNNExplainer on most datasets and is faster. The pre-learned MLP helps a lot in this regard. However, some strange behavior may occur on particular graphs. For example, when the best explanation is a subgraph that appears multiple times in the graph, MLP tends to give high weights to the edges of these subgraphs. But if they are the same, the weights will be the same, and in the end, if selected, the budget and connectivity constraints will give a low score to the edges of this subgraph. In fact, GNNExplainer can work around this problem and find at least one of the subgraphs. In general, graphs with multiple symmetries or isomorphic subgraphs will not perform well in PGExplainer. Also, as in GNNExplainer, the authors test their method only on a single real dataset and compare their results to the ground truth values and not to the model output.

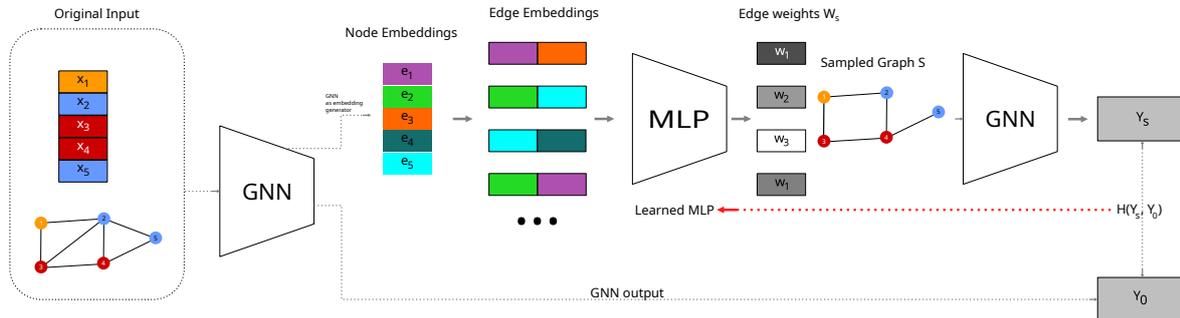


Figure 2.4: PGExplainer uses the GNN to construct edge embedding obtained by concatenating the embeddings of the two end-nodes of the edge. Then an MLP learns the edge weights used to randomly draw a graph that maximizes the mutual information between the sampled graph and the original graph.

SubgraphX

The SubgraphX method (Yuan et al., 2021) offers to find interesting subgraphs using Monte Carlo Tree search and Shapley Values (see GraphSVX presentation in Section 2.4.3 for more details). The MCTS algorithm is a well-known reinforcement learning algorithm that structures the search space into a tree. Here, the root of the tree is the graph to explain \mathcal{G} and the children of each node are the graphs obtained by pruning a single node. At each step, the search tree is expanded following the upper confidence bound. At MCTS node s_t

representing the graph \mathcal{G}_i we use :

$$a^* = \operatorname{argmax}_a \mathcal{Q}(s_t, a) + \mathcal{N}(s_t, a)$$

$$\mathcal{U}(s_t, a) = \lambda \mathcal{R}(s_t, a) \frac{\sqrt{\sum_k \mathcal{C}(s_t, a)}}{1 + \mathcal{C}(s_t, a)}$$

- $\mathcal{Q}(s_t, a)$ is the mean reward from node t .
- $\mathcal{R}(s_t, a)$ is the instantaneous reward.
- $\mathcal{C}(s_t, a)$ is the number of times we selected action a from s_t .
- $\mathcal{W}(s_t, a)$ is the sum off reward from all (s_t, a) visits.

The reward could simply be the call to Φ to evaluate \mathcal{G}_i but to have a more consistent value, the authors use Shapley values. Let $V = \{v_1, \dots, v_n\}$ be the nodes of \mathcal{G} , $V = \{v_1, \dots, v_k\}$ the nodes of \mathcal{G}_i and $\{v_{k+1}, \dots, v_n\}$ the nodes in $\mathcal{G} \setminus \mathcal{G}_i$. The players in the game are $\{\mathcal{G}_i, v_{k+1}, \dots, v_n\}$. As calculating the Shapley can be expensive, the authors propose several optimizations, such as restricting the nodes to the L-hop neighborhood of the deleted node, corresponding to the depth of the GNN.

The authors test this model on MUTAG, BBBP, BA-2 and BA datasets, on various GNN models (GC, GAT, GIN) and compare their results to GNNExplainer, PGExplainer and MCTS_GNN (a variant of their algorithm which does not use Shapley values). They show that the fidelity/sparsity trade-off outperforms the other methods on most experiments, to the detriment of computation time: on BBBP the method is 5 times slower than GNNExplainer and 4000 times slower than PGExplainer, while having a fidelity of 0.55 over 0.19 and 0.18 respectively. In many use cases, especially when the human is involved, the time to produce an explanation is not the most important but here, more than a minute per explanation can make it unusable. In fact, the use of Shapley values and MCTS as two expensive algorithms. Unfortunately, the authors have not shown the timing performance of MCTS.GNN for comparison. The calculation of Shapley values may have redundant information between nodes of the same neighborhood, but this information is not reused to speed up the algorithm. The usage of an MCTS to search graphs can lead to isomorphisms and ultimately search through a lattice, making the search intractable. SubgraphX proposes to use a pruning strategy on the edge degree. However on graph data some other strategies can be implemented to reduce the size of the search space.

2.4.3 Decomposition-based methods

Decomposition methods aim to assign an importance score to individual features of the input space. In most cases, these methods examine the hidden parameters to decompose the input features. What are considered input features may vary between different methods as well as how to combine the importance of these different types of information between features, nodes and edges (Baldassarre and Azizpour, 2019, Duval and Malliaros, 2021, Lundberg and Lee, 2017, Pope et al., 2019, Schwarzenberg et al., 2019).

GNN-LRP

The Layer-wise Relevance Propagation (LRP) method aims to propagate the output to the input layer. This method showed better results than gradient methods, especially in deep models. Instead of using gradient calculation as a measure of importance, LRP methods use a different equation. For a standard neural network, the relevance of a layer's neuron j is calculated from the relevance of neurons from the previous layers:

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_i a_i w_{ik}} R_k$$

with a_i the activation value of the neuron i and w_{ik} the weight of the connection between i and j . This equation is based on the first order Taylor expansion. With the conservative property $\sum_j R_j = \sum_k R_k = f(X)$, the sum of values of the relevance matrix of a layer should be the same as that of the relevance matrix of the following layer and the output of the original input.

One could use the LRP method for GNN by simply rewriting the above equation for the aggregation layer. But as with the gradient methods, it wouldn't help to get the structural information captured by the GNN. GNN-LRP (Schwarzenberg et al., 2019) provides a walk-based procedure for obtaining structural information. Walks are common in learning graph representations. The relevance score of neuron j at step K of the walk is defined by:

$$R_{jKL\dots} = \sum_{k \in K} \frac{e_{JK} h_j w_{jk}^*}{\sum_J \sum_{j \in J} e_{JK} h_j w_{jk}^*} R_{kL\dots}$$

$w_{jk}^* = w_{jk} + \gamma \max(0, w_{jk})$ is the weight component of the matrix \mathbf{W} at coordinates jk and γ an hyperparameter described in (Bach et al., 2015).

It is then possible to rewrite $R_{jKL\dots} = h_j c_{jKL\dots}$, with $p_k = \sum_J \sum_{j \in J} e_{JK} h_j w_{jk}$, $c_{jKL\dots}$ can be formulated :

$$c_{jKL\dots} = \sum_{k \in K} e_{JK} w_{jk}^* \frac{h_k}{p_k} c_{jL\dots} = \sum_{k \in K} \frac{\partial p_k}{\partial h_j} \frac{h_k}{p_k} c_{jL\dots}$$

This allows using the automatic differentiation algorithm of any deep learning library to calculate the relevance score of a $R_{jKL\dots}$ walk. In the end, we can see this as a Taylor expansion of the same order as the number of layers of the GNN.

The authors propose a wide range of experiments, ranging from the syntactic tree, molecules, synthetic graphs and even to the image Convolutional network with VGG-16 (Simonyan and Zisserman, 2015), seen as a particular graph neural network. Experiments show a better but close performance of GNNExplainer.

GNN-LRP provides an importance score on paths, so post-processing is required to provide usable output. This is also a strong point in favor of this method: it is possible to adapt certain prior knowledge to the explanation. For example, this allows certain paths or nodes to be deleted. However the calculation of the paths can be expensive: if each node is connected to two nodes (no isolated node and no automatic loop is authorized) there are more than $n * 2^T$ paths with n the number of nodes and T the depth of the GNN. Hopefully GNNs in general have 3 or 4 layers, but on large and dense graphs this can be a serious limitation. Unfortunately, the authors give no computation time to produce an explanation.

As gradient methods, this method does not work on any type of models. Although this

should work on any locally differentiable function, rewriting the propagation equation on other GNN models is necessary.

Graph-SVX

Graph-SVX (Duval and Malliaros, 2021) is another approach to explain GNNs by weighting the importance of elements in the graph using Shapley's value. Shapley's values come from game theory: In a collaborative environment, there are multiple players with a common goal. The game can be played multiple times with different subsets of players. The algorithm breaks down the output score into each player's individual contributions. The Shapley value are obtained by the average marginal contribution of each player participating in each possible coalition (Shapley, 1953, Strumbelj and Kononenko, 2010).

This method uses additive feature attribution by creating a linear approximation function around the explained point. In the GNN environment, the players of the game are the nodes, edges and features of the input graph (denoted X) and the method evaluates the average marginal contribution of each of them. Let X be the player set with $M = |X|$. Let S be a subset of players and $\Phi : X \rightarrow \mathbb{R}$ be the function we want to explain by decomposing its value in a vector of values corresponding to the players individual contributions: $\alpha(\Phi) = (\alpha_1, \dots, \alpha_M)$. Under the following axioms, the solution α is unique.

- Efficiency: the sum of the contributions $\sum_{i=0}^M \alpha_i$ must match the difference between the model prediction ($\Phi(X)$) on X and the average prediction α_0 :

$$\Phi(X) = \alpha_0 + \sum_{i=0}^M \alpha_i, \quad \alpha_0 = \mathbb{E}[\Phi]$$

- Symmetry: the contribution of two players is equivalent if they contribute equally to each possible coalition:

$$\forall S \subseteq X, \text{ and } j, k \notin S, \text{ if } \Phi(S \cup \{j\}) = \Phi(S \cup \{k\}) \text{ then } \alpha_j = \alpha_k$$

- Dummy: a player that does not influence the predicted value has a contribution of 0:

$$\forall S \subseteq X, \text{ an } j \notin S, \text{ if } \Phi(S \cup \{j\}) = \Phi(S) \text{ then } \alpha_j = 0$$

- Additivity: for every pair of games Φ_1 and Φ_2

$$\alpha(\Phi_1 + \Phi_2) = \alpha(\Phi_1) + \alpha(\Phi_2) \text{ where } (\Phi_1 + \Phi_2)(S) = \Phi_1(S) + \Phi_2(S)$$

This axiom constraints the values to be consistent along the space of predictions.

There exists a unique solution α that satisfies the above constraints: It is the Shapley values:

$$Sh_j(\Phi) = \sum_{S \subseteq X \setminus \{j\}} \frac{|S|!(M - |S| - 1)!}{M!} [\Phi(S \cup \{j\}) - \Phi(S)]$$

In practice, the sum becomes impossible to compute because the number of possible coalitions (2^{M-1}) increases exponentially by adding more features. Shapley values can thus be approximated using sampling. To find the values α , pairs of coalition and prediction are generated: $\mathcal{D} = (z, \Phi(z'))$ where z' is the graph generated from the mask. The equation

$\alpha = \arg \min \mathcal{L}_\Phi(g)$ is solved using a weighted linear regression model:

$$\mathcal{L}_{\Phi,\pi}(g) = \sum_z (\Phi(z') - (\alpha_0 + \sum_{j \in Z} \alpha_j)^2 \pi_z)$$

where $\pi_z = \exp(-D(x, z)^2/\sigma^2)$ be an exponential kernel defined on some distance function D (e.g. cosine distance for text, $L2$ distance for images) with width σ . π_z is a kernel that assigns weights to masks of small or large dimensions around the explained point x .

The authors propose a method to produce masks for the graph by adding specific constraints for the masks. For example, in node classification, a mask with a node w disconnected to the center v makes no sense so they add the shortest path from w to v while removing information from characteristics of intermediate nodes by assigning them a random vector of characteristics.

In terms of taxonomy, Graph-SVX fits in several categories. It is a decomposition-based method as it tries to associate each input feature with a contributing score. The score adds up to the final prediction and the outcome of the method is of the same type as the other decomposition models. But in its process, it is also a perturbation-based method as it runs the model multiple times around the original input. Also, the output is clearly a linear approximation of the original model and can be considered a surrogate method.

2.4.4 Surrogate methods

Surrogate methods find a simpler and more explainable model to approximate the original model in the vicinity of the instance to be explained. Such methods assume a simpler behavior on a bounded subspace, for example a linear approximation. In most cases, surrogate methods generate a dataset of instances around the example to explain, and then build the surrogate model on the generated data. Again the difficulty lies in the structural generation. Here, the work focuses on the choice of the surrogate model and the characteristics considered by this model. When considering other DNN models than GNN, LIME (Ribeiro et al., 2016) is probably the best known method. Any interpretable model can be used as a surrogate method, however the problem is building a model that works on graphs and a direct lime implementation would not work and some adaptations are needed (Huang et al., 2020). Surrogate models can also be used as a model-level explanation, again the lack of interpretable models for the graph data is the limitation.

PGM-Explainer

PGM-Explainer (Vu and Thai, 2020) proposes to use Probabilistic Graphical Models (PGM) Jordan (2004) to construct explanations of the GNN. The principle is to determine the dependencies between the nodes of the graph with respect to the decision. It first builds a dataset of graphs by perturbing the nodes and storing them whenever this perturbation changes the decision. Then it builds a probabilistic graphical model by selecting top dependent variables to reduce the size of the local dataset via the Grow-Shrink algorithm (Gámez et al., 2011, Margaritis and Thrun, 1999). Finally, the Bayesian network is fit to the local dataset and used to explain the predictions of the original GNN model.

In experiments, PGM-Explainer is compared to SHAP (Lundberg and Lee, 2017) and GNNExplainer on several datasets (synthetic datasets, bitcoin-OTC (Kumar et al., 2016),

Mnist superPixel-Graph (Wang and Vinel, 2021)). This model only builds its interpretation on node features and looks vaguely at structure, whereas most competitors work equally or exclusively on edges. The authors compare this method to other methods such as GNNExplainer, however some adaptations are necessary: GNNExplainer builds the explanation on the edges and not on the nodes, this means that the comparison is not necessary just. Moreover the proposed experiment does not show any real dataset where GNNExplainer was initially tested. The real-world datasets used rely weakly on structural information. But, on the positive side, this is one of the few articles that uses crowd sourcing as a validation tool and offers an interesting reflection on this subject.

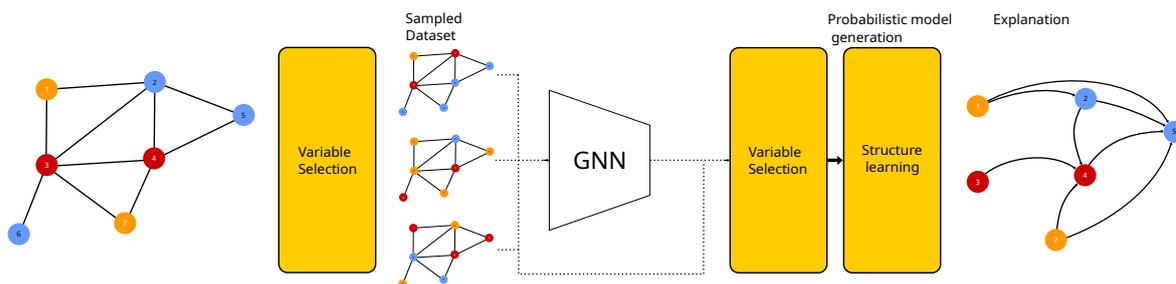


Figure 2.5: PGM-Explainer builds a dataset by modifying the node features and feeds this dataset to the GNN. Then, it removes the least important node features and learns a probabilistic graphical model that expresses the dependencies between the nodes of the input graph.

2.4.5 Global model explanation methods

Model explanation techniques try to find an explanation for the model rather than single instances. When considering graphs, the main difficulty is probably to explore the search space. For single-instance explanation methods, subgraphs are considered as explainable domain. This is still a vast search space but, with smart heuristic algorithms, it is possible to find explanations in limited time. It is also easier to provide meaningful subgraphs. For model explanation tasks, the search starts with a blank page and must come up with a graph, or set of graphs. This approach is more difficult. The realism of the explanations must be taken into account. For example, a random graph is hardly similar to a real molecule. For some data, domain knowledge can be introduced.

XGNN

XGNN (Yuan et al., 2020a) is the first work that considers the model explanation problem. It relies on a generative model based on reinforcement learning to produce graphs that activate one of the output classes. The generative model works iteratively. It takes a graph G and proposes the next edge to add, either between two existing nodes, or between an existing node and a node of the candidate set C . The generative model starts by computing the embedding of each node of G and of candidate nodes with a GCN. The embedding matrix is denoted \hat{X} . Then, with two MLPs (MLP_s , MLP_e), it chooses the next edge to add to G . The first neural network $MLP_s : emb \rightarrow \mathbb{R}$ generates a weight for each embedded node and

selects the best node to start an edge. Technically, two values are calculated at this step: the probability of choosing each node as the starting node, and the starting node sampled from it,

$$p_{t,start} = \text{Softmax}(MLP_s(\hat{X}))$$

$$a_{t,start} \sim p_{t,start} \cdot m_{t,start}$$

with $m_{t,start}$ a binary mask that removes the candidates nodes. The starting node have to be in G .

The selection of the end node is done similarity but, knowing the starting node, the second neural network takes as input the embedding matrix and the starting node's embedding ($MLP_e : emb \times emb \rightarrow \mathbb{R}$):

$$p_{t,end} = \text{Softmax}(MLP_e(\hat{X}, \hat{x}_{start}))$$

$$a_{t,end} \sim p_{t,end} \cdot m_{t,end}$$

where $m_{t,end}$ masks the starting node. This generative model is used in a reinforcement framework. The model starts with an initial graph G_1 , generally made of a single node. At each step t , the generative model builds the graph G_{t+1} by adding an edge to G_t . Then G_{t+1} is evaluated to calculate the reward R_t . If $R_t < 0$, G_{t+1} is rolled-back to G_t . This process is iterated until the end condition is satisfied, usually a combination of a maximal number of steps, and a size limit on G in number of edges or nodes. The evaluation of G_t is constructed as follows:

$$R(G_t) = R_c(G_t) + \lambda \frac{\sum_m R_c(\text{Rollout}(G_t))}{M}$$

$$R_c(G_t) = p(\Phi(G_t) = c) - 1/2$$

The reward $R(G_t)$ is used in the loss function used to learn the MLPs:

$$\mathcal{L}_g = -R(G_t)(\mathcal{L}_{CE}(p_{t,start}, a_{t,start}) + \mathcal{L}_{CE}(p_{t,end}, a_{t,end}))$$

where \mathcal{L}_{CE} is the cross entropy loss. XGNN also offers a graph rules mechanism to disallow an edge between two existing nodes, or limit the number of nodes. This is done by adding a penalty on the reward when the proposed graph does not respect the rules of the graph.

Due to the nature of the task, XGNN is assessed using new measures that assess the quality of explanations. XGNN assumes that there is only one phenomenon involved in a class. This may be true in synthetic datasets (where, for example, graphs contain one cycle), but is likely false in Mutag dataset (where the graphs may have multiple mutagenic elements). XGNN can probably find a structure responsible for the target property, but no guarantee is given on how the model can find the other structures.

The graph generator is size invariant (this is usually an interesting property in graph generation), but this implies that the way the first edge is selected is the same as the last edge. There is therefore no apparent structure in the generation of the graph. We can see that the reward function is quite rich but is transmitted as a single weight to the loss. This consists of passing very little information through back-propagation. In the end, the learning is unidirectional and does not separate the predictive part from the roll-out part and the graph rule part.

Finally, graph rules are underdeveloped. Assessing realism is necessary at some point, otherwise the graph search space is too large and XGNN would only generate completely

irrelevant contradictory graphs. In this algorithm, the graph rules are a binary function giving almost no clue to why the proposed edge is wrong. With roll-out, it becomes very difficult to obtain a valid graph, and it is very difficult for the model to have an informative roll-out: when the rules of the graph are not respected, a roll-out penalty is applied. In fact, in our tests, the generative model has great difficulty to learn how to create a valid graph and tends to generate adversarial graph rather than a global explanation.

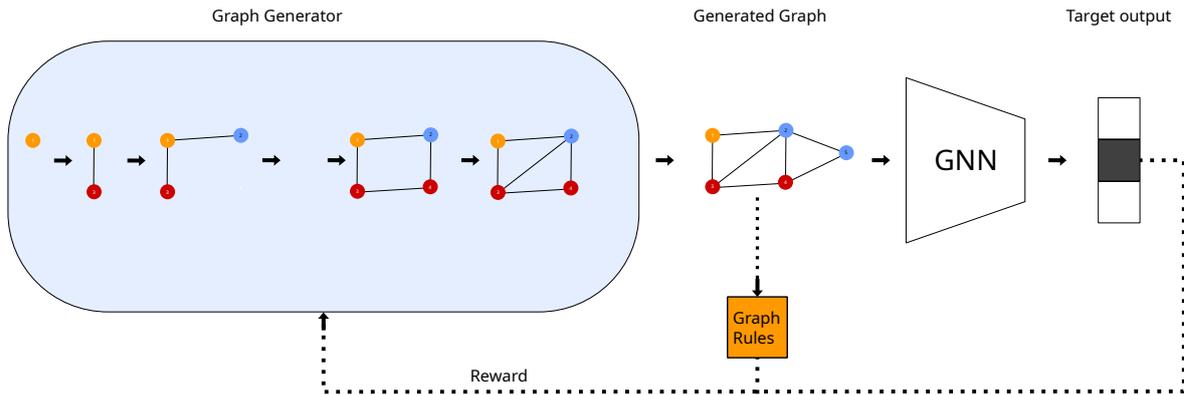


Figure 2.6: XGNN attempts to replicate a specific output class by generating graphs. If a generated graph follows a set of specific rules and if the GNN classifies the graph in the target class, then a reward helps the generator to learn how to generate subsequent graphs.

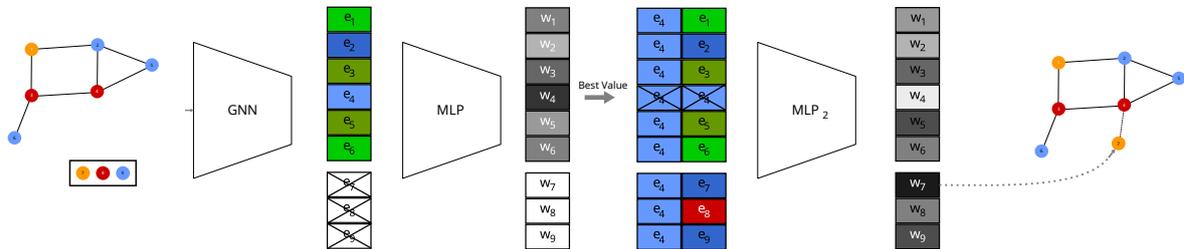


Figure 2.7: XGNN Graph Generator: The graph generator takes a graph as input and chooses an edge to add. The GNN is used to create node embeddings. A first MLP is used to choose the first node of the new edge. With this information, the second MLP uses the embedding of the candidate node to choose the termination node of the edge.

ProtGNN

ProtGNN (Zhang et al., 2022) is a transparently designed GNN that aims to learn representative subgraphs based on a prototypical vector for each instance. It provides a self-explanatory GNN. Its result consists of the prediction and a set of sub-graphs supporting the prediction. Like the previous methods, ProtGNN is inspired by other methods on other types of neural networks such as ProSeNet (Ming et al., 2019) and ProtoPNet (Chen et al., 2019).

ProtGNN uses a GNN as node graph encoder, named $emb(\cdot)$. g_p is the component that learns a set of typical graph embedding vectors, $(p_k)_{k \leq 2m}$, with m prototypes for the positive

class and m prototypes for the negative class. It calculates a similarity sim between the embedding of an input graph $h = emb(g)$ and a prototype: $sim(p_k, h) = \log(\frac{\|p_k - h\|_2^2 + 1}{\|p_k - h\|_2^2 + \epsilon})$ and ϵ is a small value to avoid 0 division. The Conditional Subgraph Sampling module generates for an instance a set of graphs that correspond to the embeddings of the prototypes.

The conditional subgraph sampling module is designed to transform uninterpretable prototype vectors into subgraphs of the input graph to provide instance-level interpretation. This sub-graph is generated by pruning the input graph via a Monte Carlo Tree search which aims to minimize the similarity between the embedding of the graphs generated by the GNN and the embedding of the target prototype. This MCTS algorithm is used repeatedly for each prototype.

To learn the model, the authors use cross-entropy as the main loss, but the model also needs another mechanism to force properties of interest into prototypes, and enforced them to be sufficiently different to bring informativeness:

- Each graph embedding should be close to at least one prototype of the same class, P_{y_i} . This enforce specific prototypes

$$Clst = \frac{1}{n} \sum_{i=1}^n \min_{p_j \in P_{y_i}} \|emb(x_i) - p_j\|_2^2$$

- The embedding should be as far as possible to the prototypes of the other class

$$Sep = -\frac{1}{n} \sum_{i=1}^n \min_{p_j \notin P_{y_i}} \|emb(x_i) - p_j\|_2^2$$

- The embeddings should be diverse:

$$Div = \sum_{k=1}^C \sum_{\substack{i \neq j \\ p_i, p_j \in P_k}} \max(0, \cos(p_i, p_j) - s_{max})$$

with s_{max} the cosine threshold.

The global loss is thus:

$$loss = \frac{1}{n} \sum_{i=1}^n CrsEnt(c \circ g_p \circ emb(x_i), y_i) + \lambda_1 Clst + \lambda_2 Sep + \lambda_3 Div$$

with c the fully connected layer used for the final decision.

Moreover, at each stage, ProtGNN uses MCTS to generate new prototypes. The prototype vector is replaced by the closest embedding generated by the MCTS. This helps make the prototype vectors realistic by coming from subgraph of the dataset rather than just an average value in a cluster:

$$p_j \leftarrow \arg \min_{h \in MCTS(p_j)} \|h - p_j\|_2^2$$

In such an architecture, the MCTS is the slowest part. So the authors proposed another approach for this section: Use an MLP to generate graph prototypes similar to PGExplainer (Luo et al., 2020). An MLP is learned to predict the edge weights of a random Gilbert graph.

$e_{i,j} = \sigma(\text{MLP}_\theta(z_i, z_j, p_k))$ where z_i and z_j are the nodes integrated from the GNN encoder. This MLP is trained to minimize the similarity of the generated graph and the prototype vector. MLP is used in training.

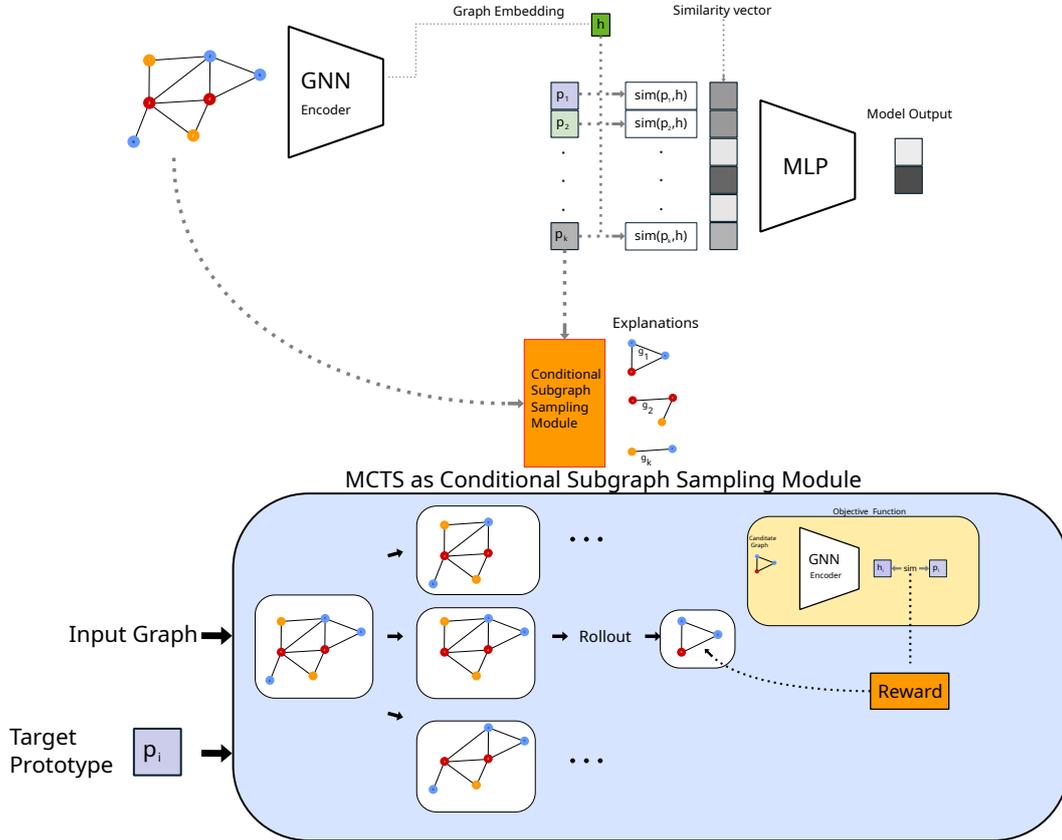


Figure 2.8: For classification, ProtGNN uses a GNN to construct a graph embedding. Then, this vector is compared to each of the prototype vectors p_i to construct the similarity vector. An MLP is used as a decoder for classification tasks. For the sake of explanation and for each prototype vector p_i , the conditional sampling module constructs a sub-graph whose embedding is close to the prototype vector p_i .

ProtGNN is transparent by design and can be used as an instance-level explainer in the same way as other post-hoc explainers. However, the authors offer only a superficial comparison of ProtGNN with GNNExplainer and PGExplainer. The choice of MLP changes the pruning paradigm: MCTS was about pruning nodes (with adjacent edges) while MLP only prunes edges. Such changes should have an impact on the prototypes. Finally, the optimization path goes in several directions: the prototypes are optimized according to the classification error and the other prototypes. The subgraph generator is also used to optimize prototypes. Finally, the GNN encoder must be learned. Such a complex optimization path may raise questions about the reproducibility and convergence of such a model.

2.5 Discussion

Explainability methods for Graph Neural Networks is a very active field as we have just seen. Several additional methods have been proposed with more or less success (Funke et al., 2021, Huang et al., 2020, Schlichtkrull et al., 2021, Wang et al., 2021) in the last two or three years. Most of the methods work on single instance explanations and provide a mask or are compatible with masks. This notion of mask comes from explainability tasks in image or text neural networks. However, if the masks are a good support of explanation, they have weaknesses in their expressiveness.

2.5.1 Using masks as GNN explanations

In graph domain, a mask is a subset of edges, nodes, and node features. Some methods focus only on edges while others also consider node features. Hiding nodes with a discrete mask also hides their adjacent edges. The same goes for the edge mask: if a node is isolated, we need to mask it too. One of the main advantages of such masks is that it can be lattice-structured with the original input at the top and the empty graph at the bottom, even though the number of elements in such a lattice is exponential in size of the considered graph. Sparse masks and nearly full masks are more tractable. These masks are the most interesting ones. A sparse mask can be an explanation on which part is important for the decision: removing it changes the decision. A nearly full mask will only select a portion of the input that is largely correlated with the output. The first type of mask is interesting and perform well on fidelity measure, the latter on the infidelity measure.

However, this explanation language excludes the addition of nodes or edges. Satisfactory explanations could be obtained by adding nodes or edges to the graph. It would also be interesting to explain the absence of a pattern: proving a pattern that is not present in an input requires considering all the possibilities and in graph this is equivalent to a subgraph inclusion test. Probably the best way to explain that a pattern is not present is to present a minimal set of edits (additions) to make the pattern appear. Adding nodes and edges to provide an explanation can be interesting, but it is also a very expensive procedure, especially when the input graph is sparse. We must consider the validity of the explanation provided. For example, in the molecule we have the limitation of the number of edges, but the graph must also be able to fit in space with specific rules for the distance between nodes and the angle between edges. Deleting elements can also break the validity of the graph. However, it is much easier to understand it as a subgraph of one or more valid graphs.

The choice in the domain of explanation, the types of explanation provided, is limited by the combinatorics and the methods chosen: most of the methods we have seen cannot have the additions of edges and nodes in their search space in a way efficient. For gradient methods, we need to test each edit to see if it's relevant with the gradient, and the same goes for GNNExplainer. Graph-SVX cannot calculate the contribution to a non-existing edge or node. This language limitation is more of a limitation of the single instance explanation process than other types of methods.

2.5.2 Further explanation needs at the model-level

Model-level approaches can overcome these limitations. Global approaches seek to identify the most important patterns. The transparent model by design ProtGNN creates prototypes that can help to have a more global view of what's going on. However, GNNs are relatively new as machine learning tasks and it is highly likely that efficient architectures will continue to evolve. This method takes some freedom from standard GNN structures. On the other hand, XGNN offers a fully model independent method. However, generative approaches have several limitations in application. While it can help find a few typical instances of a class, it can't find them all. It can behave more like an adversarial generator that finds examples that maximize the output value and not directly what the model has identified. Global explainability tasks are in a more daunting task. Therefore, it is a much less studied topic in graph work on global explainability. On other data types, like tabular data, a human can infer how a model works overall and its limitations by several well-selected single instance explanations. With graphs, the structural aspect is much more difficult to navigate in, and the space explored by single-instance explanation methods is too limited in that regard.

Graph Neural Networks construct a local embedding of each node of the graphs. Even for the classification of graphs, there is the number of nodes multiplied by the number of layers as data points that represent structural information. For the Aids dataset, this represents 120,000 data points that carry the structures identified by the GNN. In case we are able to identify features in the embedding, we do not know how they relate to graph structures. The GNN is a one-way application of the graph to embedding, the other way is much more difficult. There are many difficulties in this task. The first GNNs are surjective functions. Second, it is not about finding a specific embedding and inverse image, but finding a subgraph with an integration that matches the highlighted part of the integration. Few very recent approaches (Duval and Malliaros, 2021, Zhang et al., 2022) have considered this problem, just as we do in this thesis.

Chapter 3

Mining activation rules in the hidden layers of GNNs

3.1 Inside GNN motivations and desirata

As the literature review shows, there is a lack of methods to explain GNN models. Most existing techniques focus on single explanations that are easier to create and understand. However, they provide a limited understanding of what is really going on. They make possible to understand specific instances and also to approximate the global behavior of the GNN based on multiple instances. But, such approaches have low evidentiary value. The global understanding of GNNs is much more interesting and only a few methods have tried to solve this problem.

The main leverage we will use is that in a GNN classification task we have many more data points to work with than other models. The GNN is an embedding machine. When running it on a full dataset, we have as many embeds as there are nodes in the graph. Considering the multiple layers, we can multiply this number by the number of layers. Each node embedding is a characteristic vector of an ego-graph of fixed diameter. It contains structural and nodal information of this ego-graph. A GNN extracts information locally. To understand its behavior, we focus on local models. The developed method (Veyrin-Forrer et al., 2022) works in two main steps: Identifying patterns that are associated with a GNN class and finding prototype graphs that embed themselves in such patterns.

In the following work, we want to be as time resistant as possible. We are working on Graph Neural Networks, but this computational model is quite new and constantly evolving. We want the developed algorithm to be always valid with new versions of GNNs. We want to take the best position in the compromise between building a general agnostic algorithm and a sufficiently specific method that takes advantage of the GNN architecture and specificity.

3.2 Graph Neural Network setup

We now need to more formally define the type of GNN we are working on. In all our experimental setup, we use Graph Convolutional Networks (GCN), but any type of Graph Neural Network with a fixed number of layers can be explained with our method. On the

contrary, the recurring architecture should require adaptations and new experiments. We consider a set of graphs \mathcal{G} with labels on nodes: $G = (V, E, L)$ with V a set of nodes, E a set of edges in $V \times V$, and L a mapping between the nodes and the labels, $L \subseteq V \times T$, with T the set of labels. The graphs of \mathcal{G} are classified in two categories $\{c^0, c^1\}$ by a GNN: $\text{GNN} : \mathcal{G} \rightarrow \{c^0, c^1\}$. The GNN takes decisions at graph level. More precisely, we consider Graph Convolutional Networks (GCN) (Kipf and Welling, 2017) with L layers. GCNs compute vectors $\mathbf{h}_v^\ell, l = 1 \dots L$ of dimension K , an hyperparameter of the method. \mathbf{h}_v^ℓ represents the ego-graph centered at node v with radius ℓ . This ego-graph is induced by the nodes that are less than a distance ℓ (in number of edges) from v . Such vectors are recursively computed by the following formula:

$$\mathbf{h}_v^\ell = \text{ReLU} \left(\mathbf{W}_\ell \cdot \sum_{w \in \mathcal{N}(v)} \frac{e_{w,v}}{\sqrt{d_v d_w}} \mathbf{h}_v^{\ell-1} \right), \text{ with } d_v = \sum_{w \in \mathcal{N}(v)} e_{v,w}.$$

$e_{v,w}$ is the weight of the edge between nodes v and w , $\mathcal{N}(v)$ is the set of neighboring nodes of v including v , ReLU is the rectified linear activation function, and \mathbf{W}_ℓ are the parameters learned during the model training phase. Finally, \mathbf{h}_v^0 is the initial vector for node v with the one-hot encoding of its labels from set T .

Each vector is of size $K = 20$ and ℓ ranges from 0 to $L = 3$ the number of layers in the GNN, two hyperparameters of the GNN, $\ell = 0$ refers to the input layer¹. Considering graph classification task, we concatenate the embedding of each layer for each node : $emb_v = (h_v^1, h_v^2, h_v^3)$ then we aggregate all nodes embedding $(emb_v)_{v \in V}$ with a max and a mean pool. The decision is taken using a single decoder layer to a binary output. For a trained GNN, the vectors \mathbf{h}_v^ℓ capture the key characteristics of the corresponding ego-graphs on which the classification is made. When one of the vector components is of high value, it plays a role in the decision process, actually it is the combination of multiple values in the embedding vector that is determinant for the decision process, and even early in the inference process important patterns, seen as a subgraph or a family of subgraph that are similar, are identified by the GNN and are useful for the classification. An identified pattern is either specific to one or the other class, otherwise no learning stress is put to the GNN to keep track of such pattern. We want to identify such structures through the hidden vector.

3.3 Activation matrix and activation rules

A hidden vector h_v^k is the embedding of the ego network of radius k rooted in node v . We want to build a family of functions $r_a(h) : \mathbb{R}^K \rightarrow \{0, 1\}$ that output 1 if the embedding h of an ego-graph activates the pattern a . This family of functions are called activation rules and we want to build them to be fast to compute so that we can compute (a) patterns in tractable time. For this, we build a set of positive rules, associated with patterns belonging to the positive class and with negative rules for the negative class, or class 0:

$$r_a(h) = \prod_{i \in a} \mathbb{1}(h_i > 0)$$

with a a subset of indices. The rule is activated if all component with indices in a have a value over 0. This means that the component was on the right side of the ReLU function and

¹There is no \mathbf{W}_0 .

we consider the component as activated. We can see a as a binary vector A where $A_i := 1$ if $i \in a, A_i = 0$. In order to search through the space of activation rules, we consider a binarized version of the hidden vectors. For the next step, we pre-compute all hidden vectors, layer by layer and build the activation matrix as defined below.

Definition 1 (Activation matrix). *The activation matrix \widehat{H}^ℓ has dimensions $(n \times K)$, with $n = \sum_{g_i \in \mathcal{G}} |V_i|$.*

$$\widehat{H}^\ell[v, k] = \begin{cases} 1 & \text{if } (\mathbf{h}_v^\ell)_k > 0 \\ 0 & \text{otherwise} \end{cases}$$

For a given layer ℓ , the activated components of \mathbf{h}_v^ℓ correspond to the part of the ego-graph centered at v and of radius ℓ that triggers the decision. With this representation, the activation rule test is done through a scalar product $\mathbf{h}_i \cdot A > 0$.

Other languages can be used to define activation rules. For example, we could use non-zero numeric triggers, or linear relationships. In the next step, we see a way to leverage activation rules with other languages. The search space then becomes much larger and even more intractable. On the other hand, the rule with the threshold 0 stated above corresponds to a computational reality.

3.4 Activation rules discovery

We want two sets of activation rules: one that identifies embeddings of the positive class c^1 and another that identifies embeddings of the negative class. We use a subgroup discovery approach to identify sets of vector components that are mostly activated in graphs having the same GNN decision to construct our rules. In the activation matrix, each line corresponding to an embedding node of a graph G is associated with the decision of the GNN on this embedding in order to define the support of the rule:

Definition 2 (Activation rule and support). *An activation rule $A^\ell \rightarrow c$ is composed of a binary vector A^ℓ of size K and $c \in \{c^0, c^1\}$ a decision class of the GNN. A graph $g_i = (V_i, E_i, L_i) \in \mathcal{G}$ activates the rule if there is a node v in V_i such that $\widehat{H}^\ell[v, k] = (A^\ell)_k, \forall k = 1 \dots K$. It is denoted $\mathbf{Activate}(A^\ell \rightarrow c, v)$. The activated graphs with GNN decision c form the support of the rule:*

$$\mathbf{Supp}(A^\ell \rightarrow c, \mathcal{G}) = \{g_i \in \mathcal{G} \mid \exists v \in V_i, \mathbf{Activate}(A^\ell \rightarrow c, v) \text{ and } \text{GNN}(g_i) = c\}$$

Thus, the activated rules are more interesting if their supports are largely homogeneous in terms of GNN decisions, i.e. the graphs of the support are mainly classified either in the class c^0 , or in the class c^1 . Most rules have support in both classes in terms of GNN decision, and to be more precise, it is possible to have strictly identical embeddings of nodes which belong to differently classified graphs. This phenomenon is more common in the lower layers where the embedded ego-networks are small. Using the GNN decision instead of the real class is motivated by the same argument in Section 2.3.2. We want to explain the decision of the GNN and explain why it made a certain decision regarding its internal state and not try to explain why an instance is of a certain class. Regarding the internal state, it does not change anything when the GNN has correctly predicted the class. But when the

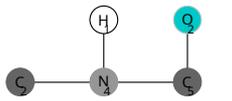
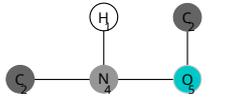
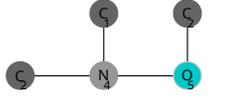
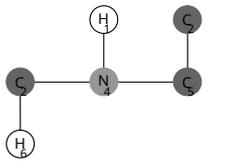
	Graph	Output class	Component						
			Node	component 1	component 2	component 3	component 4	component 5	component 6
G_1		c^1	1	0.1	0.2	0.0	0.0	0.0	0.3
			2	0.0	0.0	0.2	0.2	0.4	0.0
			3	0.2	0.1	0.0	0.0	0.0	0.2
			4	0.0	0.1	0.0	0.0	0.0	0.2
			5	0.0	0.0	0.2	0.0	0.3	0.0
G_2		c^1	1	0.1	0.0	0.2	0.2	0.3	0.0
			2	0.1	0.0	0.1	0.0	0.1	0.4
			3	0.1	0.0	0.2	0.2	0.3	0.0
			4	0.0	0.0	0.2	0.0	0.3	0.0
			5	0.0	0.0	0.1	0.0	0.1	0.3
G_3		c^0	1	0.0	0.0	0.0	0.0	0.0	0.0
			2	0.0	0.3	0.1	0.0	0.1	0.1
			3	0.0	0.0	0.0	0.4	0.2	0.0
			4	0.0	0.0	0.0	0.0	0.1	0.0
			5	0.0	0.2	0.0	0.0	0.1	0.1
G_4		c^0	1	0.0	0.2	0.0	0.0	0.0	0.3
			2	0.0	0.0	0.2	0.2	0.3	0.0
			3	0.0	0.1	0.0	0.0	0.0	0.1
			4	0.0	0.1	0.0	0.0	0.0	0.1
			5	0.0	0.0	0.1	0.0	0.2	0.0
			6	0.0	0.2	0.0	0.0	0.0	0.0

Figure 3.1: Toy example: The internal GNN representation of 4 graphs on the third layer with $K = 6$. Non-null components (grey cells) are considered as activated and encoded by a '1' value in the binary activation matrix \widehat{H}^3 . The pattern $A^3 = (1, 0, 0, 0, 0, 1)$ is activated by nodes 1 and 3 of G_1 and node 2 of G_2 . Thus, $\text{Activate}(A^3 \rightarrow 1, v_1) = \text{True}$, $\text{Activate}(A^3 \rightarrow 1, v_3) = \text{True}$ for G_1 and $\text{Activate}(A^3 \rightarrow 1, v_2) = \text{True}$ for G_2 . $\text{Supp}(A^3 \rightarrow 1, \mathcal{G}) = \{G_1, G_2\}$.

GNN has misclassified an instance, we keep the correlation between the internal state and the output.

The support is the first measure to evaluate the interest of an activation rule. It is useful to build a first rule. However, given that we have found the most interesting rule for a given target class, what is the use of the second most interesting rule if it shares 90% of the support? We want to increasingly cover our activation matrix with the following rules. The goal here is to add *a priori* knowledge about the interest of our activation rules in order to be able to build them iteratively and build the next activation rule knowing the previous ones.

3.4.1 Measuring the interest of an activation rule

The question now is how to evaluate the interest of the activation rules so as to obtain a set of non-redundant rules. One way to achieve this is to model the knowledge extracted from the activation matrix into a background model and to evaluate the interest of a rule by the knowledge it brings in relation to it. This is what the FORSIED framework (De Bie, 2011) does. It proposes an operational way to define the background model and to evaluate the subjective²

We consider the discrete random variable $H^\ell[v, k]$ associated to the activation matrix $\widehat{H}^\ell[v, k]$ ³, and we model the background knowledge by the probability $P(H^\ell[v, k] = 1)$. Intuitively, the information content (IC) of an activation rule should increase when its components are unusually activated for the nodes in the graphs of its support, it is unlikely that these components are activated when considering a random node, while this probability increases when considering graphs supporting the pattern).

Thus, given the probabilities $P(H^\ell[v, k] = 1)$ and with the assumption that all $H^\ell[v, k]$ are independent of each other, we can evaluate the interest of a rule by the product of $P(H^\ell[v, k] = 1)$ for v activated by the rule and k such that $(A^\ell)_k = 1$. Equivalently, we use the negative log-probability.

The more probable the pattern – and therefore the less interesting – the smaller this value. As there may exist several nodes activated in a single graph, we choose the one maximizing the measure.

Definition 3 (Rule information content). *Given a probabilistic background model P , the information provided by a rule $R = A^\ell \rightarrow c$ to characterize a set of graphs \mathcal{G} is measured by*

$$IC(R, \mathcal{G}) = \sum_{g: \in \text{Supp}(R, \mathcal{G})} \max_{\substack{v \in V_i \text{ and} \\ \text{Activate}(R, v)}} - \sum_{k \text{ s.t. } (A^\ell)_k = 1} \log(P(H^\ell[v, k] = 1))$$

Example 1. Considering the example of Fig. 3.1 and the rule $A^3 \leftarrow 1$ with $A^3 = (1, 0, 0, 0, 0, 1)$ and the probabilistic background model P given in Table 3.1, $IC(A^3 \rightarrow 1, \mathcal{G}) = -\log(0.72) - \log(0.34) - \log(0.99) - \log(0.47) = 3.13$.

The measure of a pattern's interestingness is founded on several choices and mainly the questions around do we count a rule activated on multiple nodes on a graph? Here the max limits this to only one node maximum per graph, and avoid larger graph in the datasets or frequent but not very specific patterns to have a too high IC. We have a preference here toward patterns that appears once on graphs. Also this policy helps to avoid the mining of a rule that capture multiple patterns: if we replace the max by a \sum and if multiple ego network happens to have embedding with the same activated components the IC of a rule targeting such component could be high while not capturing only one phenomenon. This choice leads to sparser explanations with less nodes and edges, that are easier to understand and to justify for the user.

The IC alone is not enough, but it may be more difficult for the user to assimilate it, especially when its description is complex. To avoid this drawback, the IC value is contrasted

²Subjective means that the measure of pattern interest is relative to the background knowledge model. It uses information theory to quantify both its informativeness and complexity.

³We use hats to signify the empirical values.

by its description length which measures the complexity of communicating the pattern to the user. The higher the number of components in A^ℓ , the more difficult to communicate it to the user.

Definition 4 (Description length of a rule). *The description length of a rule is evaluated by*

$$DL(A^\ell \rightarrow c) = \alpha(|A^\ell|) + \beta$$

with $|A^\ell|$ the L1 norm of A^ℓ , α the cost for the user to assimilate each component and β a fixed cost for the pattern. We set $\beta = 1$ and $\alpha = 0.6$, as the constant parameter β does not influence the relative ranking of the patterns, and with a value of 1, it ensures that the DL value is greater than 1. With $\alpha = 0.6$, we express a slight preference toward shorter patterns.

Example 2. $DL(A^3 \rightarrow 1) = 2\alpha + \beta = 2.2$.

The subjective interestingness measure is defined as the trade-off between IC and DL.

Definition 5 (Subjective interestingness of a rule). *The subjective interestingness of a rule on the whole set of graphs \mathcal{G} is defined by*

$$SI(A^\ell \rightarrow c, \mathcal{G}) = \frac{IC(A^\ell \rightarrow c, \mathcal{G})}{DL(A^\ell \rightarrow c)}$$

Up to this point we can find interesting subset of components, however they are not yet linked to a specific class, we want to find interesting patterns for each of the classes. So we split the subjective interestingness for the different class. With a negative argument for the activated node on the other class.

Definition 6 (Differential measure of subjective interest). *If we denote by \mathcal{G}^0 (resp. \mathcal{G}^1) the graphs $g_i \in \mathcal{G}$ such that $GNN(g_i) = c^0$ (resp. $GNN(g_i) = c^1$), the subjective interest of the rule $A^\ell \rightarrow c$ with respect to the classes is evaluated by*

$$SI_SG(A^\ell \rightarrow c) = \omega_c SI(A^\ell \rightarrow c, \mathcal{G}^c) - \omega_{1-c} SI(A^\ell \rightarrow c, \mathcal{G}^{1-c}).$$

The weights ω_0 and ω_1 are used to counterbalance the measure in unbalanced decision problems. The rational is to reduce the SI values of the majority class.

Example 3. $SI_SG(A^3 \rightarrow 1) = SI(A^3 \rightarrow 1, \mathcal{G}^1) - SI(A^3 \rightarrow 1, \mathcal{G}^0) = \frac{IC(A^3 \rightarrow 1, \mathcal{G}^1)}{DL(A^3 \rightarrow 1)} - \frac{IC(A^3 \rightarrow 1, \mathcal{G}^0)}{DL(A^3 \rightarrow 1)} = \frac{3.13}{2.2} - 0$.

We set $\omega_0 = \max(1, \frac{|\mathcal{G}^1|}{|\mathcal{G}^0|})$ and $\omega_1 = \max(1, \frac{|\mathcal{G}^0|}{|\mathcal{G}^1|})$. The choice of weights is important when the dataset is unbalanced, and can be used to leverage

Example 4. Let see what happens if a rule activates the same number of graphs in both classes. Is such rule interesting? If we have a ratio of 70 – 30 positive over negative graphs, and a rule A with the same support for the positive in both classes, and making the assumption that the SI is the same, we will have $\omega_0 = 2.33$ and $\omega_1 = 1$ leading to a $SI_SG(A^\ell \rightarrow c_0) = 1.33SI$ and $SI_SG(A^\ell \rightarrow c_1) = -0.66SI$. Here the number of activated graphs impacts linearly the interestingness of the rule.

Or if the percentage p of coverage is the same on each class, with once again a similar interest per graph, we have $SI(A^\ell \rightarrow c_0, \mathcal{G}^0) = 0.3p\alpha$ and $SI(A^\ell \rightarrow c_1, \mathcal{G}^1) = 0.7p\alpha$. In the end, we have $SI_SG(A^\ell \rightarrow c_0) = \frac{7}{3} \cdot 0.3p\alpha - 0.7p\alpha = 0 = SI_SG(A^\ell \rightarrow c_1)$. A rule activating the same proportion of graphs among both classes is as uninteresting for both classes.

3.4.2 Computing the background model

The background model P is initialized with basic assumptions about the activation matrix and updated as rules are extracted.

Definition 7 (Initial background model). *Some components can be activated more than others on all the graphs, or some nodes can activate a variable number of components. We assume that this information is known and use it to constrain the initial background distribution P :*

$$\sum_v P(H^\ell[v, k] = 1) = \sum_v P(\widehat{H}^\ell[v, k] = 1),$$

$$\sum_k P(H^\ell[v, k] = 1) = \sum_k P(\widehat{H}^\ell[v, k] = 1).$$

However, these constraints do not completely specify the probability matrix. Among all the probability distributions satisfying these constraints, we choose the one with the maximum entropy. Indeed, any distribution P with an entropy lower than the maximum entropy distribution effectively injects additional knowledge, reducing uncertainty unduly. The explicit mathematical MaxEnt model solution can be found in (De Bie, 2009).

The corresponding initial background model of example of Fig. 3.1 is given in Table 3.1.

Once a rule $A^\ell \rightarrow c$ has been extracted, it brings some information about the activation matrix that can be integrated into P . The model must integrate the knowledge carried by this rule, that is to say that all the components with value 1 of A^ℓ are activated by the vertices activating the rule.

Definition 8 (Updating the background model). *The model P integrates the rule $A^\ell \rightarrow c$ as follows: $\forall k$ such that $(A^\ell)_k = 1$ and v such that $\widehat{H}^\ell[v, k] = (A^\ell)_k$, $P(H^\ell[v, k] = 1)$ is set to 1.*

3.4.3 Iterative extraction of subjective activation subgroups

We propose to compute the subjective activation rules with an enumerate-and-rank approach. It consists to compute the rule $A^\ell \rightarrow c$ with the largest SI_SG value and to integrate it in the background distribution P to take into account this newly learnt piece of information.

Algorithm 1 sketches the method. First, it sets the output set equal to the empty set (line 1) and the minSI value to the smallest value (line 2). A stack of size K is initialized line 3. The first considered rule A^ℓ is initialized as a bit-vector of size K containing only 0's. It corresponds to the rule with no activated components. It has an associate attribute Pot that encodes the components that could still be activated for A^ℓ , as it leads a yet unconsidered combination of activated components. Rule A^ℓ is then staked in $Stack$ (lines 4 to 6). Line 7, it computes the background model P from the activation matrix \widehat{H}^ℓ as defined in Definition 7. Then, in a loop (lines 8 to 11), it computes iteratively the rule $A^\ell \rightarrow c$ having the best $SI_SG(A^\ell \rightarrow c)$ value. Then, this best rule is used to update the model P (line 11). Indeed,

Graph	Node	component 1	component 2	component 3	component 4	component 5	component 6
G_1	1	0.729	0.556	0.556	0.507	0.346	0.346
	2	0.729	0.556	0.556	0.507	0.346	0.346
	3	0.729	0.556	0.556	0.507	0.346	0.346
	4	0.527	0.402	0.402	0.366	0.250	0.250
	5	0.527	0.402	0.402	0.366	0.250	0.250
G_2	1	0.999	0.762	0.762	0.695	0.474	0.474
	2	0.999	0.762	0.762	0.695	0.474	0.474
	3	0.999	0.762	0.762	0.695	0.474	0.474
	4	0.527	0.402	0.402	0.366	0.250	0.251
	5	0.729	0.556	0.556	0.507	0.346	0.346
G_3	1	0.256	0.195	0.195	0.178	0.122	0.122
	2	0.999	0.762	0.762	0.695	0.474	0.474
	3	0.527	0.402	0.402	0.366	0.250	0.250
	4	0.374	0.285	0.285	0.259	0.177	0.177
	5	0.730	0.556	0.556	0.507	0.346	0.346
G_4	1	0.527	0.402	0.402	0.366	0.250	0.250
	2	0.729	0.556	0.556	0.507	0.346	0.346
	3	0.527	0.402	0.402	0.366	0.250	0.250
	4	0.527	0.402	0.402	0.366	0.250	0.250
	5	0.527	0.402	0.402	0.366	0.250	0.250
	6	0.374	0.285	0.285	0.259	0.177	0.177

Table 3.1: Initial background model $P(H^\ell[v, k] = 1)$ of example of Fig. 3.1.

once the rule A^ℓ is known, its subjective interest falls down to 0. This consists in setting the corresponding probabilities to 1.

Algorithm 2 presents `INSIDE-SI` that computes the best rule with as activated components the one's values of the vector stored in `Stack[depth]`, and even more, depending on the recursive process. It considers a pattern A stored in the stack at depth `depth`. A has 5 attributes:

- $A.Pot$, a vector whose one's values represent the activated components that can be further added to A during the enumeration process,
- $A.G^c$ (resp. $A.G^{1-c}$) the set of graphs from \mathcal{G}^c (resp. \mathcal{G}^{1-c}) that support A ,
- and $A.TG^c$ (resp. $A.TG^{1-c}$) the set of graphs that are supporting A and all its descendants in the enumeration process (there is a node in these graphs that activates all the components of A and $A.Pot$).

Algorithm 1 $\text{INSIDE-GNN}(\widehat{H}^\ell, c, nbPatt)$

Require: \widehat{H}^ℓ the activation matrix (see Definition 1), c the class to be characterized and $nbPatt$ the number of patterns.

Ensure: $output$, the up to $nbPatt$ best activation rules $A^\ell \rightarrow c$ w.r.t. SI_SG .

```

 $output \leftarrow \emptyset$ 
1:  $minSI \leftarrow -\infty$ 
2:  $Stack.maxsize \leftarrow K$ 
3:  $A^\ell \leftarrow$  a size  $K$  bit-vector initialized at 0
4:  $A^\ell.Pot \leftarrow$  a size  $K$  bit-vector full of 1's
5:  $Stack[0] \leftarrow A^\ell$ 
6:  $P \leftarrow Compute\_Model(\widehat{H}^\ell)$ 
7:
8: while ( $|output| < nbPatt$ ) and ( $minSI > 0$ ) do  $A^\ell, minSI \leftarrow \text{INSIDE-SI}(Stack, P, c, minSI, 0)$ 
9:  $output \leftarrow output \cup A^\ell$ 
10:  $Update\_Model(P, A^\ell)$ 
11: end while

```

Algorithm 2 $\text{INSIDE-SI}(Stack, P, c, minSI, depth)$

Require: $Stack$ a stack of recursively enumerated patterns at depth $depth$, P the background distribution, c the class to be characterized, $minSI$ a dynamic threshold on $SI_SG(A^\ell \rightarrow c)$.

Ensure: $Best$, the best rule w.r.t. SI_SG .

```

 $A \leftarrow Stack[depth]$ 
1:
2: if ( $(\phi(A) = False)$  or ( $UB\_SI(A, P, c) < minSI$ )) then return
3: end if
4: if ( $A.Pot = \emptyset$ ) then
5:   if ( $SI\_SG(A \rightarrow c) > SI\_SG(Best \rightarrow c)$ ) then  $Best \leftarrow A$ 
6:  $minSI \leftarrow SI\_SG(Best \rightarrow c)$ 
7:
8:   end if
9: else  $x \leftarrow$  first bit of  $A.Pot$  set to 1
10:  $A.Pot[x] \leftarrow 0$ 
11:  $A[x] \leftarrow 1$ 
12:  $Stack[depth + 1] \leftarrow A$ 
13:  $\text{INSIDE-SI}(Stack, P, c, minSI, depth+1)$ 
14:  $A[x] \leftarrow 0$ 
15:  $Stack[depth + 1] \leftarrow A$ 
16:  $\text{INSIDE-SI}(Stack, P, c, minSI, depth+1)$ 
17:
18: end if return  $Best, minSI$ 

```

The algorithm computes the closure of A using the function ϕ . It consists in adding activated components to A (set some components of A to 1) as long as the set $A.G^c$ of supporting graphs stays unchanged. Furthermore, if a component has been removed from A (on line 14) but can be added later to A (i.e. $\phi(A) \& A.pot \neq A$ with $\&$ the bitwise **and** operation), A is not closed, the function returns $False$ and the recursion stops.

Line 2, a second criterion based on an upper bound UB_SI makes the recursion stop if its value is less than the one of the current best found rule. It relies on the following property.

Property 1. Let A and B be two binary vectors of size K . The components that are activated for A are also activated for B (i.e., $A \& B = A$, with $\&$ the bitwise **and** operation). We can upper bound the value $SI_SG(B \rightarrow c)$ and have

$SI_SG(B \rightarrow c) \leq UB_SI(A, P, c)$ with

$$UB_SI(A, P, c) = w_c \frac{\sum_{g \in A.G^c} \max_{v \in V_g} - \sum_{k \text{ s.t. } (A \& A.Pot)_k=1} \log(P(H^\ell[v, k] = 1))}{\alpha(|A|) + \beta} - w_{1-c} \frac{\sum_{g \in A.TG^{1-c}} \max_{v \in V_g} - \sum_{k \text{ s.t. } (A)_k=1} \log(P(H^\ell[v, k] = 1))}{\alpha(|A \& A.Pot|) + \beta}$$

with $|A|$ the L1 norm of A .

Proof. To upper bound the measure $SI_SG(B \rightarrow c)$, we follow the strategy explained in (Cerf et al., 2009). Let

$$SI_SG(B \rightarrow c) = w_c \frac{X}{Y_1} - w_{1-c} \frac{Z}{Y_2}$$

with

$$\begin{aligned} X &= IC(B, \mathcal{G}^c) = \sum_{g_i \in \text{Supp}(B, \mathcal{G}^c)} \max_{v \in V_i} - \sum_{k \text{ s.t. } (B)_k=1} \log(P(H^\ell[v, k] = 1)) \\ Z &= IC(B, \mathcal{G}^{1-c}) = \sum_{g_i \in \text{Supp}(B, \mathcal{G}^{1-c})} \max_{v \in V_i} - \sum_{k \text{ s.t. } (B)_k=1} \log(P(H^\ell[v, k] = 1)) \\ Y_1 &= Y_2 = DL(B) = \alpha(|B|) + \beta \end{aligned}$$

Similarly, we denote the upper bound function by

$$UB_SG(A, P, c) = w_c \frac{\gamma}{\delta} - w_{1-c} \frac{\epsilon}{\eta}.$$

Therefore, the largest value of $SI_SG(B \rightarrow c)$ is obtained if:

- X has the maximal possible value, that is to say $B = A \& A.Pot$ and all the graphs of \mathcal{G}^c that support A , also support $A \& A.Pot$. In that case, we have

$$\gamma = \sum_{g \in A.G^c} \max_{v \in V_g} - \sum_{k \text{ s.t. } (A \& A.Pot)_k=1} \log(P(H^\ell[v, k] = 1))$$

- Y_1 has the smallest possible value $\alpha(|A|) + \beta$ (more elements in B will decrease the value of the fraction)

$$\delta = \alpha(|A|) + \beta$$

- Z has the smallest possible value and is computed over A , and on the graphs from \mathcal{G}^c that support A and all its descendants ($A.TG^{1-c}$)

$$\epsilon = \sum_{g \in A.TG^{1-c}} \max_{v \in V_g} - \sum_{k \text{ s.t. } (A)_k=1} \log(P(H^\ell[v, k] = 1))$$

- Y_2 has the value $\alpha(|A \& A.Pot|) + \beta$ (less elements in B will decrease the value of the function)

$$\eta = \alpha(|A \& A.Pot|) + \beta$$

It results in the upper bound definition. □

Line 4 of Algorithm 2, *Best* is updated as well as *minSI* if there are no more component to enumerate, and if the *SI_SG* value of the current rule is better than the one already found. Otherwise (lines 9 to 16), the enumeration continues either 1) by adding a component from *A.Pot* to *A* (lines 9-12) and recursively call the function (line 13), or 2) by not adding the component while still removing from *A.Pot* (line 14) and recursively continue the process (line 16).

3.5 Characterization of activation rules with subgroups

Once the activation patterns are found, we aim to describe them in an intelligible and accurate way. We believe that each activation pattern can be linked to hidden features of the graphs, that are captured by the model as being related to the class to be predicted. The objective here is to make these features explicit. For this, we seek to characterize the nodes that support the activation pattern, and more precisely to describe the singular elements of their neighborhoods. Many pattern domains can be used to that end. In the following, we consider two of them: one based on numerical descriptions and the other one based on common subgraphs. In order to characterize the subgraphs centered on the nodes of the activation pattern support (called ego-graphs) in a discriminating way compared to the other subgraphs, we extend the well-known *gSpan* algorithm (Yan and Han, 2002) so that it takes into account subgroup discovery quality measure.

3.5.1 Numerical subgroups

In this approach, we propose to describe each node that supports a given activation pattern by some topological properties⁴. We choose to consider its degree, its betweenness centrality value, its clustering-coefficient measure, and the number of triangles it is involved in, as characteristic features. These properties can be extended to the whole ego-graph by aggregating the values of the neighbors. We consider two aggregation functions: the sum and the mean. Thanks to these properties, we make a propositionalization of the nodes of the graphs and we consider as target value the fact that the node belongs to the support of the activation rule (labeled as a positive example) or not (labeled as a negative example). Therefore, we have a matrix \mathcal{D} whose rows denote graph nodes and columns correspond to numerical attributes describing the position of the node in its graph: $\mathcal{D}[v] \in \mathbb{R}^p$, with p the number of attributes. \mathcal{D} is split in two parts \mathcal{D}^0 and \mathcal{D}^1 with $v \in \mathcal{D}^c$ iff $\text{Activate}(\mathbf{A}^\ell \rightarrow c, v)$.

To identify the specific descriptions of the support nodes, we propose to use a subgroup discovery method in numerical data. It makes it possible to find restrictions on numerical attributes (less or greater than a numerical value) that characterize the presence of a node

⁴These attributes are computed with Networkx Python Library <https://networkx.org/>.

within the support of the activation rule. A numerical pattern has the form $\times_{i=1}^p [a_i, b_i]$ (i.e. the pattern language) and a graph node supports the pattern if $\forall i = 1 \dots p, a_i \leq \mathcal{D}[v, i] \leq b_i$.

To discover such subgroups, we use the `pysubgroup` library (Lemmerich and Becker, 2018).

3.5.2 Graph subgroups

Another approach consists to characterize activation rules by subgraphs that are common among positive examples in contrast to the negative ones. To this end, we consider as positive examples the ego-graphs (with a radius equal to the layer) of nodes that support the activation pattern of interest. By taking the radius into account, we are not going beyond what the model can actually capture at this layer. The negative examples are the graphs in \mathcal{G} for which none of their vertices support the activation pattern. Hence, \mathcal{D} is a set of graph nodes v associated to ego-graphs $\mathcal{E}_g = (V_g, E_g, L_g)$. \mathcal{D} is split into \mathcal{D}^0 and \mathcal{D}^1 with $v \in \mathcal{D}^c$ iff $\text{Activate}(\mathbf{A}^\ell \rightarrow c, v)$.

A graph pattern has the form $G = (V, E, L)$ (i.e. the pattern language) and a graph node supports the pattern if there exists a graph isomorphism between $\mathcal{E}_g = (V_g, E_g, L_g)$ and its ego-graph $\mathcal{E}_g = (V_g, E_g, L_g)$.

3.5.3 Quality measure and algorithms

As for the identification of activation patterns, we could have used subjective interestingness measure to characterize the supporting ego-graphs of the activation patterns. However, we opt for a more usual measure, the Weighted Relative Accuracy (Lavrač et al., 1999). Given a pattern P of a given language, a dataset \mathcal{D} split into \mathcal{D}^0 and \mathcal{D}^1 and a $\text{Supp}(P, \mathcal{D})$ measure that gives all the graph nodes supporting the pattern P in the data \mathcal{D} , the WRAcc measure

$$\text{WRAcc}(P, c) = \frac{|\text{Supp}(P, \mathcal{D})|}{|\mathcal{D}|} \left(\frac{|\text{Supp}(P, \mathcal{D}^c)|}{|\text{Supp}(P, \mathcal{D})|} - \frac{|\mathcal{D}^c|}{|\mathcal{D}|} \right)$$

gives high values to patterns that are mainly supported by nodes of \mathcal{D}^c compared to the whole dataset \mathcal{D} . Then, we use off-the-shelf algorithms to discover the best subgroups. We compute patterns P such that

$$(3.1) \quad \text{WRAcc}(P, c) \geq \min_WRAcc \text{ and } |\text{Supp}(P, \mathcal{D})| \geq \min_sup$$

or just the subgroup with the highest WRAcc value.

For the numerical subgroups, we use `Pysubgroup` library (Lemmerich and Becker, 2018). For graph subgroup discovery, we integrate the WRAcc measure into the `gSpan` algorithm (Yan and Han, 2002). As WRAcc measure is not anti-monotone, we use the following upper-bound instead of the WRAcc for pruning:

$$UB(P, c) = \frac{|\text{Supp}(P, \mathcal{D})|}{|\mathcal{D}|} \left(1 - \frac{\max(\min_sup, |\mathcal{D}^c|)}{|\mathcal{D}|} \right)$$

If $\min_sup < |\mathcal{D}^c|$, then we have $UB(P, c) = \frac{|\text{Supp}(P, \mathcal{D})|}{|\mathcal{D}|} \left(1 - \frac{|\mathcal{D}^c|}{|\mathcal{D}|} \right)$. Since $\frac{|\text{Supp}(P, \mathcal{D}^c)|}{|\text{Supp}(P, \mathcal{D})|} \leq 1$,

$WRAcc(P, c) \leq UB(P, c)$. In the other case, we have:

$$\begin{aligned} \frac{|\mathbf{Supp}(P, \mathcal{D}^c)|}{|\mathbf{Supp}(P, \mathcal{D})|} - \frac{|\mathcal{D}^c|}{|\mathcal{D}|} &\leq \frac{|\mathbf{Supp}(P, \mathcal{D})|}{|\mathbf{Supp}(P, \mathcal{D})|} - \frac{min_sup}{|\mathcal{D}|} \Leftrightarrow \\ \frac{min_sup}{|\mathcal{D}|} - \frac{|\mathcal{D}^c|}{|\mathcal{D}|} &\leq \frac{|\mathbf{Supp}(P, \mathcal{D})|}{|\mathbf{Supp}(P, \mathcal{D})|} - \frac{|\mathbf{Supp}(P, \mathcal{D}^c)|}{|\mathbf{Supp}(P, \mathcal{D})|} \Leftrightarrow \\ \frac{1}{|\mathcal{D}|}(min_sup - |\mathcal{D}^c|) &\leq \frac{1}{|\mathbf{Supp}(P, \mathcal{D})|} (|\mathbf{Supp}(P, \mathcal{D})| - |\mathbf{Supp}(P, \mathcal{D}^c)|) \end{aligned}$$

The last inequality holds since $\frac{1}{|\mathcal{D}|} \leq \frac{1}{|\mathbf{Supp}(P, \mathcal{D})|}$, $min_sup \leq |\mathbf{Supp}(P, \mathcal{D})|$, and finally $|\mathcal{D}^c| \geq |\mathbf{Supp}(P, \mathcal{D}^c)|$.

Since UB is not dependent to the $\mathbf{Supp}(P, \mathcal{D}^c)$, when $|\mathcal{D}^c|$ is much lower than the $|\mathcal{D}|$, this upper bound is not tight. We can use another upper bound which is dependent to the $|\mathcal{D}^c|$. Let us call this upper bound $UB2$:

$$UB2(P, c) = \frac{|\mathbf{Supp}(P, \mathcal{D}^c)|}{|\mathcal{D}|} - \frac{min_sup}{|\mathcal{D}|} \times \frac{|\mathcal{D}^c|}{|\mathcal{D}|}$$

Since except $\mathbf{Supp}(P, \mathcal{D}^c)$ everything is constant, and $\mathbf{Supp}(P, \mathcal{D}^c)$ is anti-monotone, $UB2$ is anti-monotone too. To show that $UB2$ is an upper bound for $WRAcc$, note that $\frac{min_sup}{|\mathcal{D}|} \times \frac{|\mathcal{D}^c|}{|\mathcal{D}|} \leq \frac{|\mathbf{Supp}(P, \mathcal{D})|}{|\mathcal{D}|} \times \frac{|\mathcal{D}^c|}{|\mathcal{D}|}$ and the first terms of $WRAcc$ and $UB2$ are equal. In our algorithm we use $UB3(P, c) = \min\{UB2(P, c), UB(P, c)\}$ as upper bound for the $WRAcc$.

3.6 GNN explanations with activation rules

INSIDE-GNN builds a set of activation rules. Each rule describes a subset of node embedding and is associated with one or the other class of the GNN, and this for each GNN's layer. These rules have been mined iteratively to describe the activation matrix obtained by running the GNN on the graphs. Now, for a new input we can easily know the activated rules and which nodes trigger them. Is this information enough to produce any kind of explanation? Or how should it be refined to be presented to a human?

An activation rule has a support made of nodes associated to the target class or the other class. The activation rule itself is made of components whose activation is correlated with the target output, meaning that something has been captured by the part of the embedding vector corresponding to the rule. The layer of the GNN that computed the rule-based embedding indicates the radius of the ego-network to consider. The information not yet available is: Does all nodes and edges of this ego network are important for the decision? Also smaller phenomenons should be identified in rules that occur in the first layer. For instance the presence of a certain label on a node can be related to a certain output. And the opposite is also possible in the later layers: Some rules can be the union of multiple patterns. In this case the neighborhood identification is still valid but a user might see two radically different ego-networks that activate the same rule, and lead to confusion.

The rules are mined iteratively in order to each rule bring new information. The question posed by this statement is: should explanations take into account the order in which the rules are extracted? The best option is to use rules support, which is much easier to understand. Considering activation rules as a set rather than a sequence is a better level of abstraction. The method generates a set of rules, each is given with metadata: target class, $SI.SG$, positive

support, negative support allowing us to create a quick single instance explanation by calling the GNN and seeing which embedded nodes are activated. However, post-processing is needed to produce a solid explanation. Do we want to show every activated node? In the end, we prefer to produce a mask: a sub-graph which is important. This way, another abstraction is made between the rules and the output. We also want to express the locality properties of an embedding: a rule is not only a description of a node but a description of an ego-network. We will see later several policies to implement this question.

3.7 Experimental study

In this section, we evaluate `INSIDE-GNN` through several experiments in order to verify its interest and validate its performance against other methods. We first see synthetic and real-world datasets and the experimental setup. Then a quantitative study of the patterns provided by `INSIDE-GNN`. Next, we show the experimental results on explanations of graph classification against several state of the art methods. Finally, we report results on the characterization of activation rules by human understandable descriptions of what GNN models capture.

`INSIDE-GNN` has been implemented in Python and the experiments have been performed on a machine equipped with 8 Intel(R) Xeon(R) W-2125 CPU @ 4.00GHz cores 126GB main memory, running Debian GNU/Linux. The code and the data are available⁵.

3.7.1 Datasets and experimental setup

Experiments are performed on six graph classification datasets whose main characteristics are given in Table 3.2. BA2 (Ying et al., 2019) is a synthetic dataset, the other datasets (Aids (Morris et al., 2020), BBBP (Wu et al., 2017), Mutagenicity (Morris et al., 2020), DD (Dobson and Doig, 2003), Proteins (Borgwardt et al., 2005)) depict real molecules and the class identifies important properties in Chemistry or Drug Discovery (i.e., possible activity against HIV, permeability and mutagenicity).

BA2

BA2 is a synthetic dataset. Each graph is generated from a Barabasi-Albert model, and a substructure is added: if the structure is a 5 node cycle, the graph belong to the negative class, if the structure is a small house the graph belongs to the positive class. Each graph contains 25 nodes and each node feature vector is of size 10 and each of its component contain the value 0.1, Finally this dataset is perfectly balanced with 500 graphs for each class.

The main interest of a synthetic dataset such as BA-2 is to control graph parameters such as the degree distribution. And only one phenomenon is involved in the difference between the two classes, On a well trained GNN, a good explainer should provide easy interpretable outputs. We see that the main drawback of this dataset is that only two patterns exist for the model to identify the classes. Actually a simple graph search algorithm can differentiate both classes and machine learning is not necessary for this problem. The test result for this dataset should be taken with care and be used as simple debugging tool.

⁵<https://www.dropbox.com/sh/jsri7jbhmkw6c8h/AACKHwcM3GmaPC8iBPMiFehCa?dl=0>

AIDS

AIDS (Riesen and Bunke, 2008) is a real-world dataset extracted from drug discovery problems. On this problem we want to predict if a molecule has inhibited the replication of HIV. Being able to predict if a molecule might have an impact on a disease can save a lot of time and money in research and testing. Each graph represents a molecule, a node represents an atom identified by a one-hot vector whose components correspond to a certain atom. The edges represent the atomic bounds where the type of bound is not used. This dataset is not well balanced with 80% of positive instances. AIDS is a quite popular dataset on which GNNs perform well.

BBBP

BBBP, for Blood Brain Barrier Penetration, is also a drug discovery dataset but this time to predict if a molecule can reach the brain. For most vertebrates, the brain is a vital organ. The brain is separated from the rest of the body by a membrane which blocks the passage of blood. This membrane makes possible the transport of some elements such as nutrients and oxygen, and blocks components such as drugs. The challenge is to find drugs that can get through this membrane to cure different brain diseases. The structure of this dataset is similar to the previous one, with each example representing a molecule.

Mutagenicity

The mutagenicity dataset (Kazius et al., 2005, Riesen and Bunke, 2008) is an undesired property of a drug that must be avoided. This sub-task is described in this dataset, and each molecule is described like in the above datasets. AIDS, BBBP, and Mutagenicity correspond to problems of increasing complexity.

DD

DD (Dobson and Doig, 2003) does not stand for Dungeons and Dragons but for the name of their authors Dobson and Doig. This dataset is not a molecular dataset but a protein one. The proteins are not represented in their molecular form because the way a protein is wrapped is a key component difficult to represent by molecular form. Also, each protein would be too large to handle. Here the protein is represented by a graph where each amino acid is a node and two nodes are connected when they are less than 6 Ångströms apart. The goal is to predict if a protein is an enzyme or not. Proteins are a key component in living beings, some carry information, other energy, or help with the structure of the cell. There are also enzymes that help chemical reactions to happen by catalysis and lowering reaction temperature.

Proteins

Proteins (Borgwardt et al., 2005) correspond to real molecules, and the class identifies important properties in Chemistry or Drug Discovery (i.e., possible activity against HIV, permeability and mutagenicity).

These datasets describe molecular data, which is a quite narrow subset of the graph problem spectrum. Such graphs have several common properties. Molecules are almost planar graphs, with a similar distribution in node labels: mostly Carbon, then Hydrogen, Oxygen, Nitrogen. Also we do not consider the bound type of edges. The use of such datasets is motivated by their accessibility. A lot of graph Datasets are designed for node classification. We discuss later on how to work with node classification problems. The main difficulty with adding a dataset type is the code overhead that needs to be created. For example some have values on the nodes and not a label, some have different types of nodes. Each different case must be dealt with several times throughout the process. Finally, the selected datasets are among the most used in the literature on GNN explainability.

A 3-convolutional layer GNN (with $K = 20$) is trained on each dataset using 80% of the data (train set). The hyperparameters are chosen using a grid-search on other 10% of the data (validation set). The learned GNN are tested on the remaining 10% of the data (test set). The corresponding accuracy values are reported in Table 3.2. *INSIDE-GNN* mines the corresponding GNN activation matrices to discover subjective activation pattern set. We extracted at most ten patterns per layer and for each output value, with a *SI_SG* value greater than 10.

Table 3.2: Main characteristics of the datasets.

Dataset	#Graphs	(#neg,#pos)	Avg. Nodes	Avg. Edges	Acc. (train)	Acc. (test)	Acc. (val)
BA2(syn)	1000	(500, 500)	25	25.46	0.995	0.97	1.0
Aids	2000	(400, 1600)	15.69	16.2	0.989	0.99	0.975
BBBP	1640	(389, 1251)	24.08	25.98	0.855	0.787	0.848
Mutagenicity	4337	(2401, 1936)	30.32	30.32	0.815	0.786	0.804
DD	1168	(681, 487)	268	715.7	0.932	0.692	0.760
Proteins	1113	(663, 450)	39	72.82	0.754	0.768	0.784

3.7.2 Quantitative study of activation rules

Table 3.3 reports general indicators about the discovery of activation rules by *INSIDE-GNN*. The execution time ranges from few minutes for simple task (i.e., synthetic graphs) to two days for more complex ones (i.e., DD). It shows the feasibility of the proposed method. Notice that this process is performed only once for each model.

To assess whether the set of extracted rules represent the GNN well and in its entirety, we used the rules to describe the input graphs (i.e. the graphs (in row) are described by the rules (in columns) and the data matrix contains the number of graph nodes supporting the corresponding rule). We then learned the simple and interpretable model that is the decision tree. Thus, from only the knowledge of the number of nodes of a graph supporting each of the rules, we can see, in Table 3.3 last column, that the decision tree can mimic the GNN decision output with high accuracy. Obviously, we do not provide an interpretable model yet, since the decision tree is based on the patterns that capture sets of activated components of the GNN. Nevertheless, the results demonstrate that the pattern set returned by *INSIDE-GNN* captures the inner workings of GNNs well.

The general characteristics of the activation rules for each dataset are provided in Figs. 3.2–3.5. One can observe – in Fig. 3.2 – that a rule is usually supported by more

Table 3.3: Execution time, number of discovered patterns by INSIDE-GNN and the ability of the pattern set to mimic GNN: the accuracy of a decision tree with activation rules as features and measured on a test set of 20% of the data. The class variable is the GNN output y_i . The closer $Acc(DT^P, y_i)$ to 1, the better the mimicry.

Dataset	Time (s)	# Act. Rules	$Acc(DT^P, y_i)$
BA2(syn)	180	20	0.98
Aids	5160	60	0.96
BBBP	6000	60	0.89
Mutagen	41940	60	0.87
DD	212400	47	0.86
Proteins	8220	29	0.87

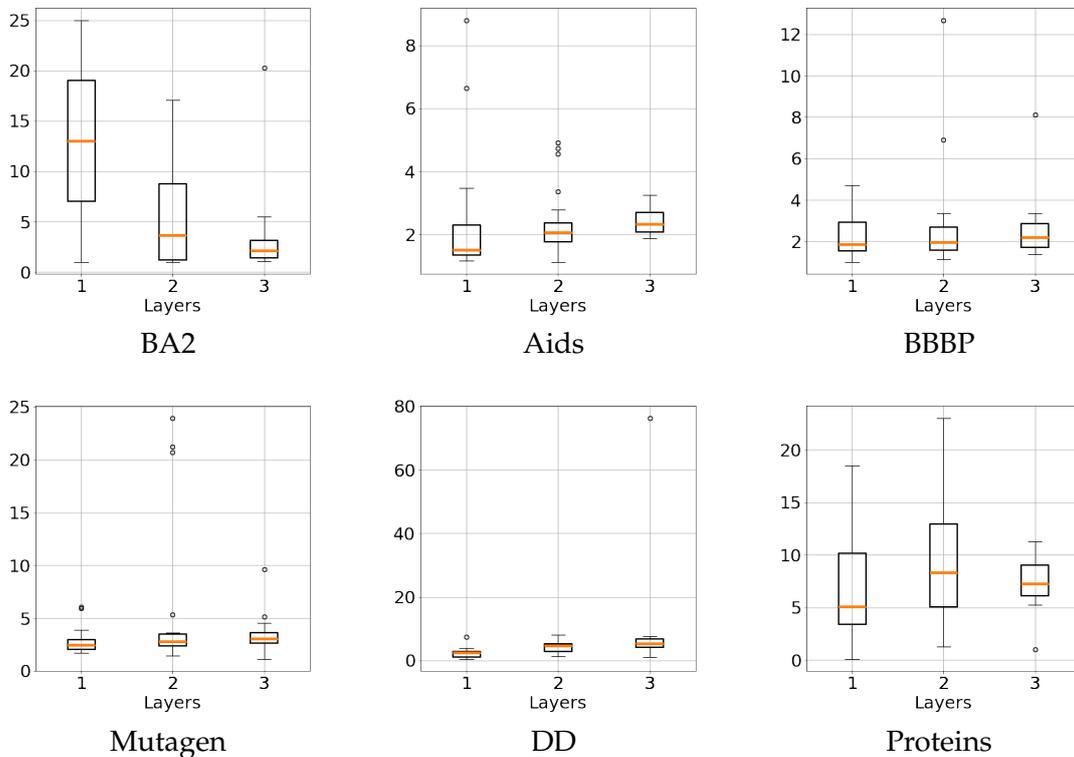


Figure 3.2: Boxplot of the number of supporting vertices per graph for layers 1, 2 and 3.

than one node within a graph. Rules from the first layer of the GNN tend to involve a higher number of vertices than those in the following layers. It may be due to the fact that the first layer captures some hidden common features about the direct neighborhood of the vertices. The features captured by the GNN become more discriminant with layer indexes, as evidenced by the increasing SLSG score with layers in Fig. 3.4. For some datasets (e.g., BA2, AIDS, DD, Proteins), some rules have high discriminative power for the positive class

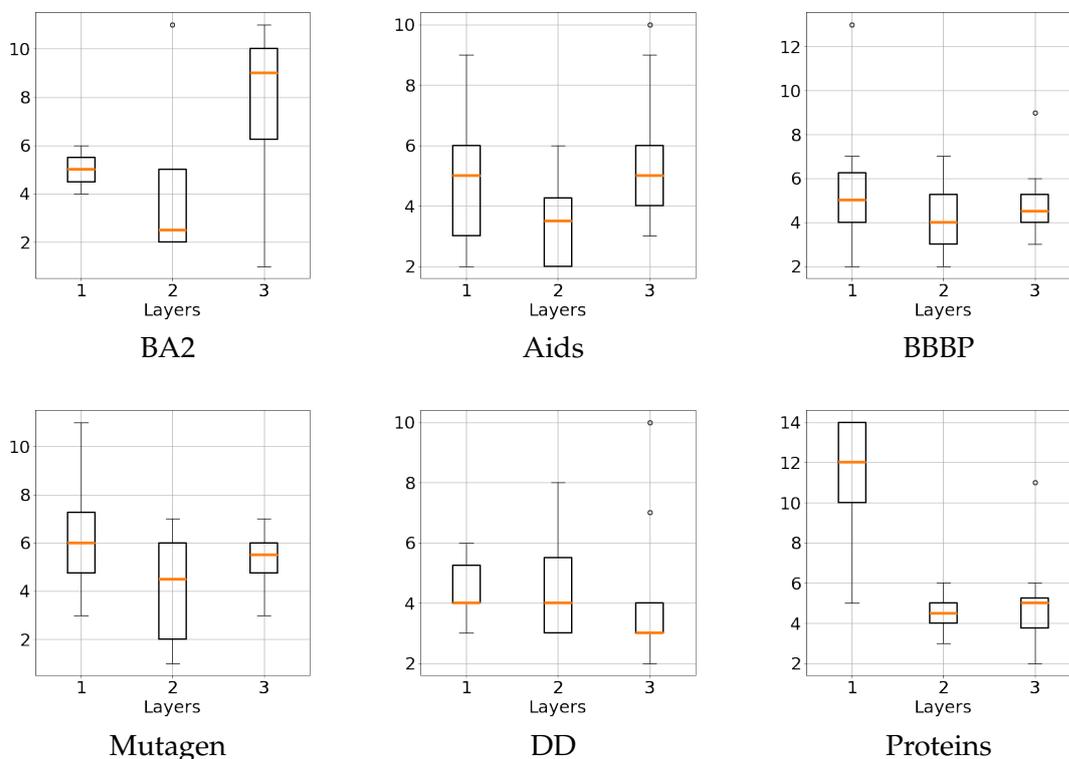


Figure 3.3: Boxplot of the number of components per pattern for layers 1, 2 and 3.

(bottom right corner in Fig. 3.5) or the negative class (top left corner). Their discriminative power is less effective for Mutagen and BBBP datasets. The most discriminant rules come from the last layer of the GNN. Some rules are not discriminant (i.e., around the diagonal) but remains subjectively interesting. These rules uncover activated components that capture general properties of the studied graphs. It is important to note that we study here the discriminative power of a rule according to its presence in graphs. These rules can be more discriminant if we take into account the number of occurrences of the rules in the graphs. For instance, a rule that is not discriminant can becomes highly discriminant if we add a condition on its number of occurrences in graph, as we did when learning the decision trees in Table 3.3.

3.7.3 Comparison with competitors for explainability of GNN output

We now assess the ability of activation rules to provide good explanations for the GNN decisions. According to the literature, the best competitors are GNNExplainer (Ying et al., 2019), PGExplainer (Luo et al., 2020) and PGM-Explainer (Vu and Thai, 2020). We consider all of them as baseline methods. Furthermore, we also consider a gradient-based method (Pope et al., 2019), denoted Grad, even if it has been shown that such method is outperformed by the three others. Therefore, we compare INSIDE-GNN against these 4 single-instance-explanation methods in our experiments.

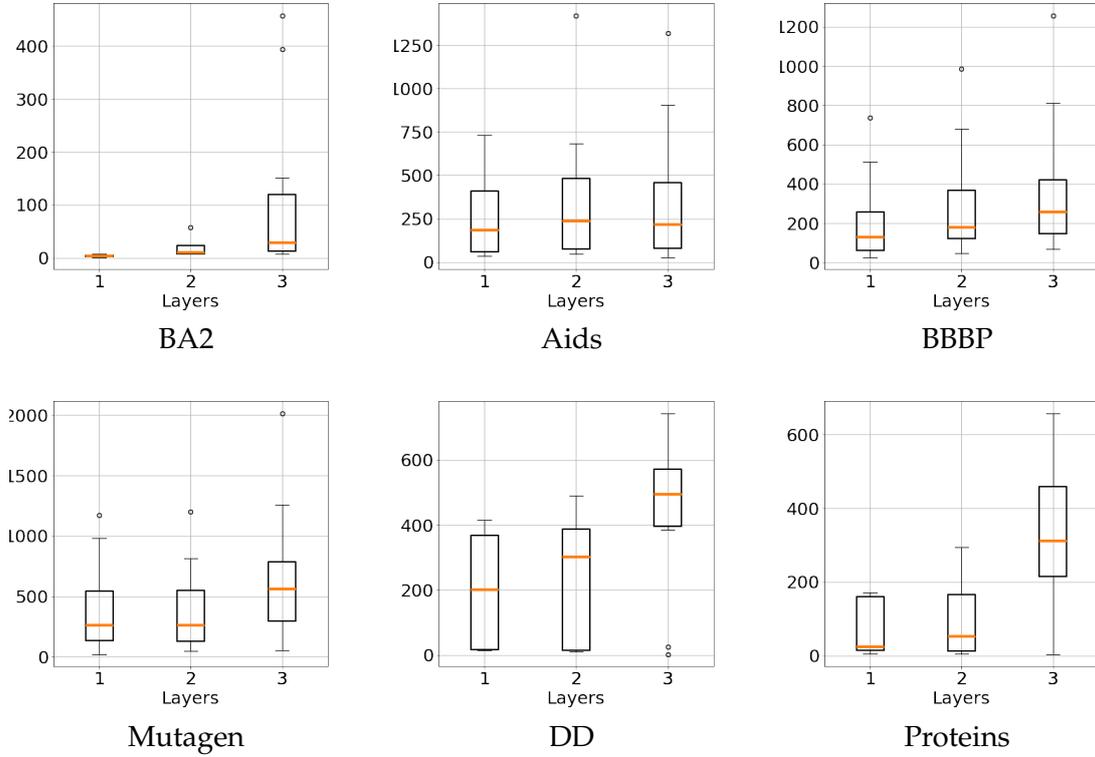


Figure 3.4: Boxplot of the SLSG score of patterns for layers 1, 2 and 3.

Evaluating the reliability of an explanation is not trivial due to the lack of ground truths. In our case, only BA2 is provided with ground truths by construction. When we have ground truths, we expect a good explanation to match it perfectly, but sometimes the model captures a different explanation that is just as discriminating. Moreover, if fully present, ground truths contain only simple relationships (e.g., BA2) which are not sufficient for a full assessment. Therefore, to be able to consider synthetic and real-world datasets, we consider a ground truth free metric. We opt for *Fidelity* (Pope et al., 2019) which is defined as the difference of accuracy (or predicted probability) between the predictions on the original graph and the one obtained when masking part of the graph based on the explanations:

$$Fid^{acc} = \frac{1}{N} \times \sum_{i=1}^N (1 - \delta_{(\hat{y}_i^{g_i \setminus m_i} = y_i)}),$$

where y_i is the original prediction of graph g_i , m_i is the mask and $g_i \setminus m_i$ is the complementary mask, $\hat{y}_i^{g_i \setminus m_i}$ is the prediction for the complementary mask and $\delta_{(\hat{y}_i^{g_i \setminus m_i} = y_i)}$ equals 1 if both predictions are equal.

The fidelity can also be measured by studying the raw probability score given by the

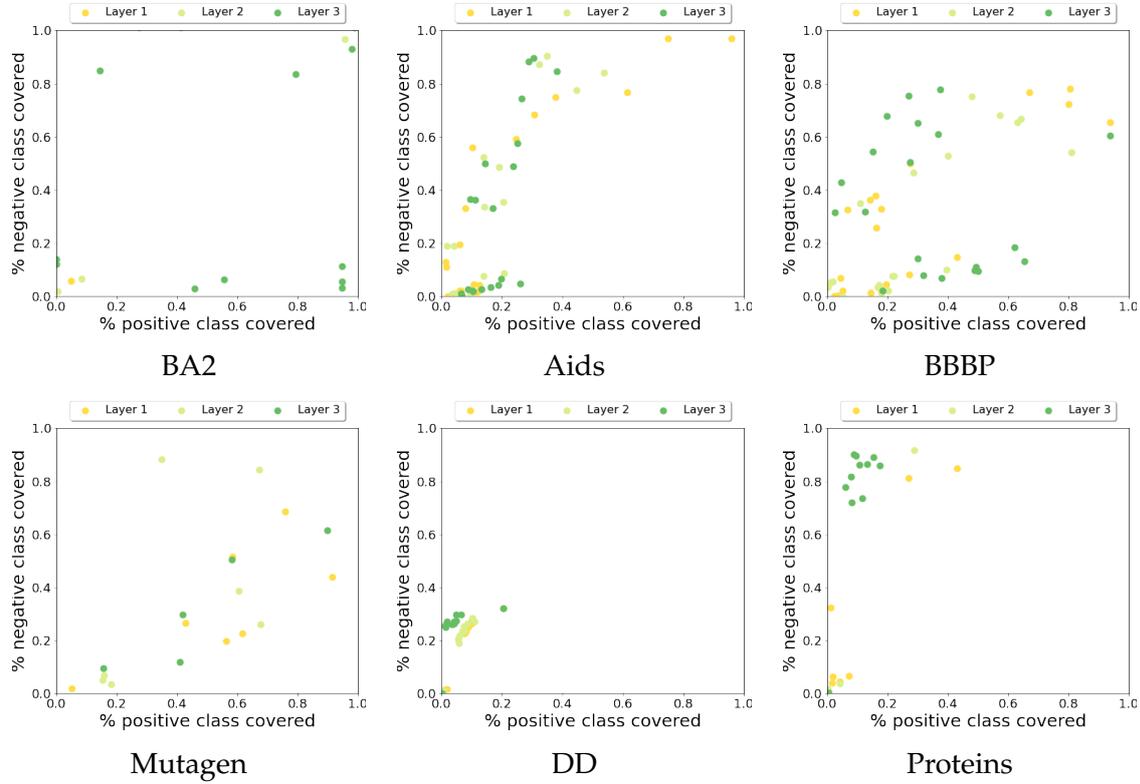


Figure 3.5: Coverage of positive and negative classes, coloured according to the layers. A “perfect” discriminating pattern for the positive class (resp. negative class) would be projected to the lower right corner (resp. upper left corner).

model for each class instead of the accuracy:

$$Fid^{prob} = \frac{1}{N} \times \sum_{i=1}^N (f(g_i)_{y_i} - f(g_i \setminus m_i)_{y_i}),$$

with $f(g)_{y_i}$ is the prediction score for class y_i .

Similarly, we can study the prediction change by keeping important features (i.e., the mask) and removing the others as Infidelity measures do:

$$Infid^{acc} = \frac{1}{N} \times \sum_{i=1}^N (1 - \delta_{(y_i^{m_i}=y_i)})$$

$$Infid^{prob} = \frac{1}{N} \times \sum_{i=1}^N (f(g_i)_{y_i} - f(m_i)_{y_i}).$$

The higher the fidelity, the lower the infidelity, the better the explainer.

Obviously, masking all the input graph would have important impact to the model prediction. Therefore, the former measures should not be studied without considering the

Sparsity metric that aims to measure the fraction of graph selected as mask by the explainer:

$$Sparsity = \frac{1}{N} \sum_{i=1}^N \left(1 - \frac{|m_i|}{|g_i|} \right),$$

where $|m_i|$ denotes the size of the mask m_i and $|g_i|$ is the size of g_i (the size includes the number of nodes, of edges and the attributes associated to them). Based on these measures, a better explainability method achieves higher fidelity, lower infidelity while keeping a sparsity close to 1.

We devise four policies to build a mask from an activation rules:

- (1) **node**: the simplest policy which takes only the nodes that are covered by the activation rule and the edges adjacent to these nodes.
- (2) **ego**: the ego-graphs of radius ℓ centered on activated nodes, with ℓ the layer associated to the pattern.
- (3) **decay**: a continuous mask with a weight associated to the edges that depends on the distance of its end-points to the activated nodes:

$$w_v = \sum_{a \in V_{\mathcal{A}}} \frac{1}{2^{1+d(v,a)}} \text{ if } d(v,a) \leq \ell, 0 \text{ otherwise}$$

with $V_{\mathcal{A}}$ the set of activated nodes, $d(v,a)$ the geodesic distance between nodes v and a and $w_{(u,v)} = w_u + w_v$.

- (4) **top k** : a discrete mask containing only the k edges from **decay** mask with the highest weights ($k = 5$ or $k = 10$ in our experiments).

For each policy, we select the mask (and the related pattern) that maximises the fidelity. As GNNExplainer and PGExplainer provide continuous masks, we report, for fair comparisons, the performance with both continuous and discrete masks built with the k best edges. Note that the average time to provide an explanation ranges from 8ms to 84ms for INSIDE-GNN. This is faster than PGM-Explainer (about 5s), GNNExplainer (80ms to 240ms) and Grad (300ms). It remains slightly slower than PGExplainer (6ms to 20ms). Table 3.4(a) summarises the performance of the explainers based on the Fidelity measures. Results show that INSIDE-GNN outperforms the baselines regardless of policy. On average, the gain of our method against the best baseline is 231% for Fid^{prob} and 207% for Fid^{acc} . These results must be analysed while considering the sparsity (see Table 3.4(c)). In most of the cases, INSIDE-GNN provides sparser explanation than the baselines. Furthermore, at equal sparsity (top k), INSIDE-GNN obtains higher fidelity values than both competitors. Notice that PGM-Explainer fails on BA2 because this dataset does not have labeled nodes and this method investigate only the nodes of the graphs.

We provide additional information on the Fidelity in Table 3.5. The Fidelity aims to measure the percentage of times that a model decision is changed when the input graphs is obfuscated by the mask m . In Table 3.5, we report a polarized version of the Fidelity for which we count the number of changes between the two possible decisions of the model. For instance, $F^{0 \rightarrow 1}$ measures the percentage of graphs initially classified as 'false' by the model that become classified as 'true' when obfuscating the graph with a mask. We can observe a

dissymmetry between the class changes. As an example, INSIDE-GNN has a perfect fidelity on BA2 and DD when considering only the positive examples, i.e., the mask provided by INSIDE-GNN makes the model change its decision. When dealing with the negative examples, we obtain much lower score. Intuitively, some class changes cannot be done by only removing some vertices or edges. Regarding BA2, it is impossible to obtain a house motif from a cycle without adding an edge to form a triangle.

The quality of the explanations are also assessed with the Infidelity metrics in Table 3.4(b). INSIDE-GNN achieves excellent performance on BA2. On the other datasets, INSIDE-GNN is outperformed by GNNExplainer. INSIDE-GNN obtain similar scores or outperforms the other competitors (i.e., PGExplainer, PGM-Explainer, Grad) at equal sparsity on most of the datasets. Notice that, in these experiments, we made the choice to build mask based on a single activation rule which is not enough to obtain fully discriminant mask for complex datasets. This is in agreement with what we observed in Fig. 3.5. We have no fully discriminant activation rule for the positive and negative classes. Hence, it would be necessary to combine activation rules to build a more discriminant mask and thus better optimise the Infidelity.

3.7.4 Model insights via the (re)description of activation patterns

We argue that activation rules also help provide insight into the model, especially what the GNN model captures. As discussed in Section 3.5, this requires characterizing the nodes (and their neighborhood) that support a given activation rule. In this experimental study, we investigate the obtained numerical subgroups for BA2 and the subgraph characterizing the activation rules retrieved for Mutagen, BBBP and Aids datasets.

Numerical subgroups

Each node can be easily described with some topological properties (e.g., its degree, the number of triangles it is involved in). Similarly, we can describe its neighborhood by aggregating the values of the neighbors. Thanks to such properties, we make a propositionalization of the nodes of the graphs. Considering the two most discriminant activation rules⁶, we use a subgroup discovery algorithm to find the discriminating conditions of the nodes supporting these two patterns. Fig. 3.6 reports a visualisation of two graphs with activated nodes in red. The best description based on WRacc measure of pattern p^1 (Fig. 3.6 left) and p^0 (Fig. 3.6 right) are given below.

For the House motif (positive class of BA2), the nodes that support activation rules are almost perfectly described (the WRacc equals to 0.24 while maximum value is 0.25) with the following conditions: *Nodes connected to two neighbors (degree=2) that are not connected between them (clustering coefficient=0), not involved in a triangle and one of its neighbors is involved in a triangle (triangle2=1)*. In other words, the activation rule captures one node of the floor of the "house motif". We have similar conditions to identify some nodes of the 5-node cycle (negative class of BA2): *nodes without triangle in their direct neighborhood (clustering2=0) and whose sum of neighbors' degree (including itself) equal 7 (degree2 ∈ [7:8])*.

⁶ $p^1 = \{a_3, a_6, a_7, a_9, a_{10}, a_{15}\}$ (where a_i are the activated components of the rule and p is the set representation of the bitset A^ℓ of Definition 2), $|\text{Supp}(p^1, \mathcal{G}^1)| = 474$, $|\text{Supp}(p^1, \mathcal{G}^0)| = 16$ and $p^0 = \{a_0, a_1, a_2, a_4, a_5, a_8, a_{11}, a_{17}, a_{18}, a_{19}\}$, $|\text{Supp}(p^0, \mathcal{G}^1)| = 137$, $|\text{Supp}(p^0, \mathcal{G}^0)| = 506$.

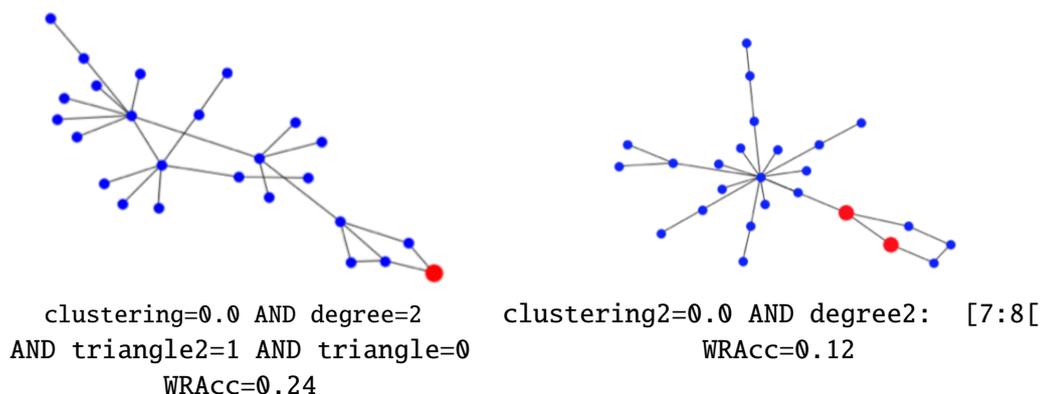


Figure 3.6: Nodes (in red) in the support of two activation rules that are discriminant for p^1 support, related to the positive target (left), and for p^0 support, related to the negative target (right).

We report the description in terms of numerical subgroups of the activation rules in Table 3.6. It is important to note that even if some activation rules were found as subjectively interesting according to a specific output of the model, they may capture some general properties of the BA2 graph that are not so specific of one of the classes. For instance, the second subgroup is related to the positive class (i.e., house motif) but what it captured is not specific to house motif (degree=2, absence of triangle).

Graph subgroups

Similarly, we can characterize activation rules with graph subgroups. We investigate the interest of such pattern language for three datasets: Aids, BBBP and Mutagen. For each activation rule, we compute the graph that has the maximum WRAcc value, using $\min_sup = 10$ (see Equation 3.1). In other words, this graph has an important number of isomorphisms with ego-graphs that support the rule and that correspond to the class of the target of the rule. In Fig. 3.7, we report the WRAcc values of the discovered graphs that aim to characterize the activation rules. We can observe that the WRAcc values are rather high (the WRAcc belongs to $[-1, 0.25]$) which demonstrates that these graphs well describe the parts of the GNN identified by the activation rules.

The subgraphs obtained for Mutagen dataset are summarised in Fig. 3.8. For each layer and decision, we display the subgraphs whose WRAcc is greater than 0.1 layer by layer. The negative class is related to mutagenic molecules. Several things can be observed from this figure. First, some subgraphs are known as toxicophores or fragment of toxicophores in the literature (Kazius et al., 2005). For instance, the subgraph with two hydrogen and one azote atoms is a part of an aromatic amine. Similarly, the subgraph with one azote and two oxygen atoms is an aromatic nitro. The subgraph involving 6 carbon atoms is a fragment of a bay region or a k-region. Second, some subgraphs appear several times. It means that several activation rules are described with the same subgraphs. This can be explained in several ways. Neural networks are known to have a lot of redundant information, as evidenced by the numerous papers in the domain that aim to compress or simplify deep neural networks

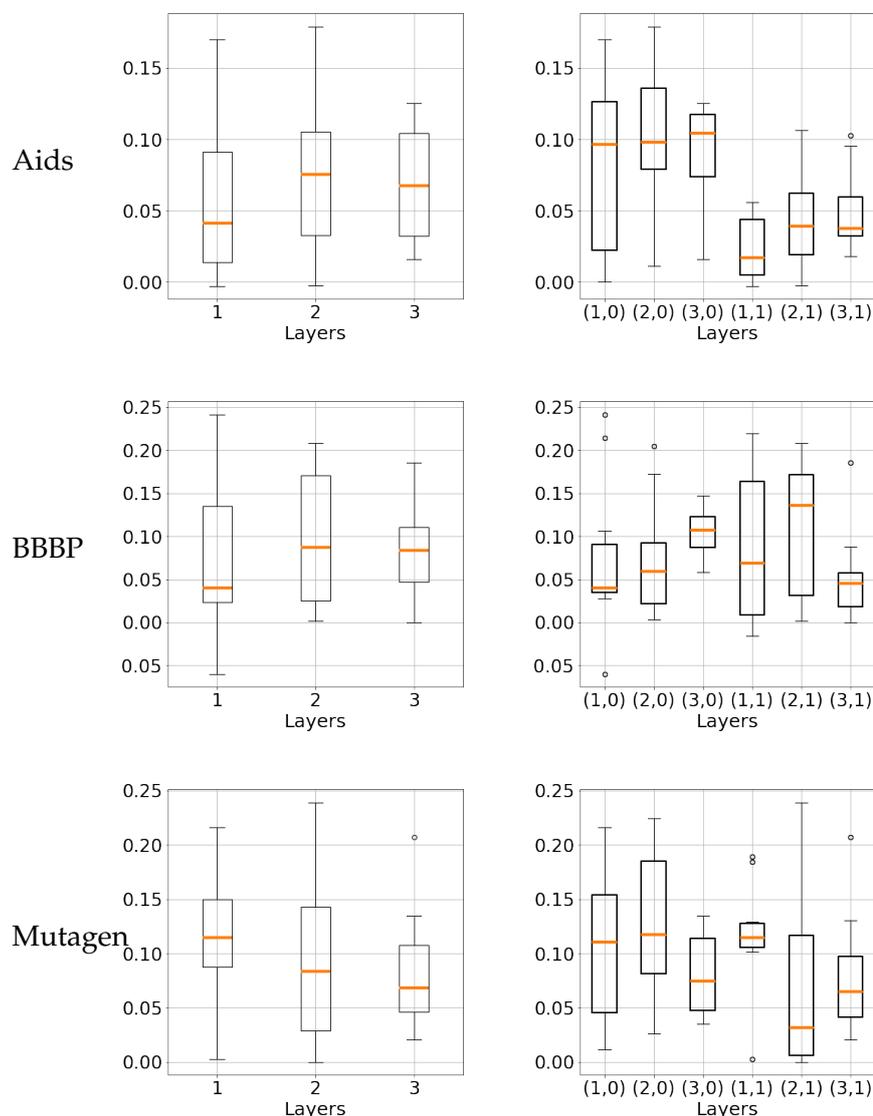


Figure 3.7: Boxplot of the WRAcc values of graph subgroups related to activation rules by layer (left column) or by both layer and model decision (right column) for Aids (first row), BBBP (second row) and Mutagen (third row).

(Chen et al., 2018a, Pan et al., 2016, Pasandi et al., 2020, Xu et al., 2018). Accordingly, this is not surprising to have several parts of the GNN that are similar and described by the same subgraphs. Notice that this problem could be an interesting perspective for our work. Another explanation is that the subgraphs well describe the hidden features captured by the GNN but from different perspective, i.e., the center is different. For instance, for a simple chemical bond C-N, one may have the same graph with one centered in C and the other in N. A last explanation could be that the subgraph language is not enough powerful to

capture the subtle differences between the activation rules. Once again, the definition of more sophisticated and appropriate languages to describe the hidden features captured by the GNN is a promising perspective of research.

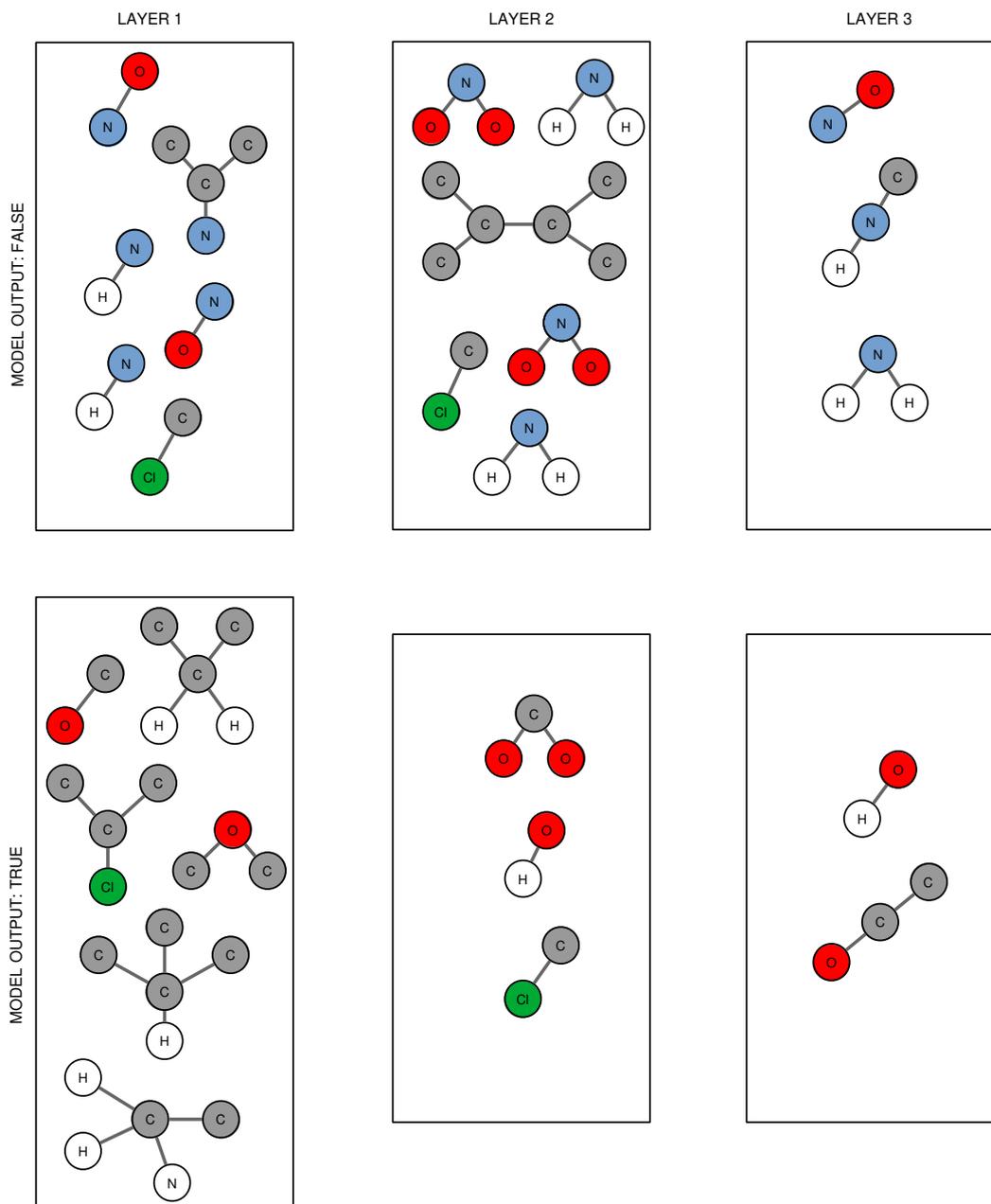


Figure 3.8: Characterization of activation rules for Mutagen with discriminant subgraphs. We retain only the subgraphs with a WRacc value greater than 0.1. Mutagenic chemicals are classified as False.

These latter experiments show that INSIDE-GNN represents a valuable alternative to GNN

explainability methods. In addition to providing single instance explanations, *INSIDE-GNN* can provide insights about what the GNN perceives. Especially, it allows to build a summary of the hidden features captured by the model (e.g., Fig. 3.8). In relation to this, our method is quite analogous to model explanation methods such as *XGNN* (Yuan et al., 2020a). This deserves a discussion and a comparison with *XGNN*.

Comparison to *XGNN*

XGNN (Yuan et al., 2020a) is a method rooted in reinforcement learning that generates graphs that maximise the model decision for a given class. For *Mutagen*, we generate 20 graphs for each class with a maximum size equal to 6. Considering the 40 generated graphs, we observe that only one of them is a subgraph of at least one graph of the dataset. The other graphs have on average 60% of partial inclusion: the maximum common subgraph with molecules from *Mutagen* uncovers 60% of a generated graph. Therefore, we can conclude that *XGNN* generates graphs that are not enough realistic. The only graph that appears within the dataset involves a carbon atom bonded to 2 others carbon atoms and one hydrogen atom. With *INSIDE-GNN*, we obtained two subgraphs characterizing some activation rules that are super-graphs of this one (see Fig. 3.8). Notice that, we also found this subgraph for some activation rules. We did not report it in Fig. 3.8 because its *WRAcc* value is lower than 0.1. Nevertheless, this graph appears in 21100 ego-graphs in the dataset. It describes a fragment of molecule that is very common. One can wonder if such a fragment can be mutagenic or if *XGNN* has just captured it a biased of the GNN. Furthermore, *XGNN* has generated graphs that are not planar, which is not common in Chemistry. Based on these evidences, we argue that *XGNN* does not return realistic graphs while our approach – by construction – provides subgraphs from the dataset.

We search for each pattern produced by *INSIDE-GNN* the closest pattern in *XGNN* according to the Graph Edit Distance (GED) and vice versa. We note that the previously described prototype graph (i.e., 3 carbons and 1 hydrogen) is found in most of the cases as being the closest to the patterns produced by *INSIDE-GNN*. In average, the distance between each *XGNN* prototype and the closest pattern of *INSIDE-GNN* is 4.6 while the mean distance between *INSIDE-GNN* subgraphs and the closest from *XGNN* is 3.7. This is rather important since the graphs provided by *XGNN* have at most 6 nodes.

We believe that a model decision for a class cannot be summarized into a single prototype. Several different phenomena can lead to the same class. Furthermore, as we observed, this can lead to unrealistic prototype even if domain knowledge is integrated within the graph generation. *INSIDE-GNN* allows to have deeper insights from the GNN by considering each hidden feature separately.

3.8 Discussion and conclusion

We have introduced a novel method for the explainability of GNNs. *INSIDE-GNN* is based on the discovery of relevant activation rules in each hidden layer of the GNN. Prior beliefs are used to assess how contrastive a rule is. We have proposed an algorithm that efficiently and iteratively builds a set of activation rules, limiting the redundancy between them. Extensive empirical results on several real-world datasets confirm that the activation rules capture interesting insights about how the internal representations are built by the GNN. Based

on these rules, *INSIDE-GNN* outperforms the SOTA methods for GNN explainability when considering Fidelity metric. Furthermore, the consideration of pattern languages involving interpretable features (e.g., numerical subgroups on node topological properties, graph subgroups) is promising since it makes possible to summarize the hidden features built by the GNN through its different layers.

We believe that such method can support knowledge discovery from powerful GNNs and provide insights on object of study for scientists or more generally for any user. However, a number of potential limitations need to be considered for future research to make this knowledge discovery from GNNs effective in practice.

First, assessing explanations without ground truth is not trivial. Our experimental evaluation relies on Fidelity, Infidelity and Sparsity metrics. Fidelity assumes that the GNN decision would change if key part of the graphs are removed. However, it is not always the case in practice. For instance, it is difficult to obtain a toxic molecule from a non-toxic one by only removing some atoms. That would be interesting to investigate other evaluation measures that take into account the negation (i.e., absence of important features) and evaluation measures based on the addition of subgraphs.

In this section, we have devised an exhaustive algorithm for discovering the activation rules. Even if pruning based on upper bound is featured, the execution time remains a problem. It ranges from few minutes to two days. This shows only the feasibility of the proposed method, not its practical application. To overcome this limitation, the completeness must be relaxed and some heuristic-based algorithms have to be defined. Beam-search or Monte-Carlo Tree Search-based algorithms are good alternatives to the one we propose to speed up this process with a minimal quality penalty.

Activation rule patterns are the simplest pattern language to deal with activation matrices since such patterns involve only conjunction of activated components. Even simple, these activation rules are able to capture the hidden features built by the GNN as witnessed by the experiments. We believe that more sophisticated pattern languages are possible for GNNs. For instance, we observed that taking into account the number of occurrences within a graph leads to better characterisations. This can be integrated to the pattern language. Considering the negation (i.e., the absence of activations) is also promising and would offer a deeper description of the internal mechanism of the GNNs. But such changes increase the computation time of *INSIDE-GNN*.

With *INSIDE-GNN*, the activation rules are mined for each layer independently. As a consequence, the relations between layers are not taken into account in the discovery of activation rules. This may lead to redundant results when considering all the layers. To avoid such redundancy, it is necessary to take into account as prior knowledge the previous layers of a given layer.

Finally, activation rules capture specific configurations in the embedding space of a given layer that is discriminant for the GNN decision. Experiments demonstrate that these rules can be directly used to support instance-level model explanation. However, activation rules cannot be easily interpreted by human beings because of the pattern language itself (i.e., conjunction of activated components of the hidden layers). The consideration of pattern languages with interpretable features makes it possible to characterize them. However, this second step can be improved by query the model itself. Indeed, the current characterization methods investigate a dataset generated from the support of the activation rules. The model should be considered in this step to have guarantee that the interpretable pattern that

describes a rule well embeds in the subspace related to this rule.

Table 3.4: Assessing the explanations with Fidelity, Infidelity and Sparsity metrics.

(a) Fidelity Model	DD		Proteins		BA2		Aids		BBBB		Mutagen	
	Fid ^{prob}	Fid ^{acc}										
INSIDE-GNN(ego)	0.540	0.663	0.362	0.651	0.342	0.494	0.165	0.097	0.344	0.295	0.492	0.647
INSIDE-GNN(node)	0.490	0.567	0.359	0.634	0.342	0.494	0.175	0.076	0.362	0.336	0.582	0.833
INSIDE-GNN(decay)	0.447	0.485	0.344	0.576	0.342	0.494	0.145	0.055	0.316	0.276	0.554	0.781
INSIDE-GNN(top 5)	0.276	0.421	0.069	0.086	0.353	0.917	0.160	0.058	0.271	0.260	0.450	0.629
INSIDE-GNN(top 10)	0.296	0.445	0.092	0.127	0.220	0.496	0.160	0.058	0.304	0.270	0.458	0.600
Grad	0.083	0.089	0.060	0.084	0.195	0.494	0.078	0.018	0.171	0.132	0.223	0.254
GnnExplainer	0.077	0.086	0.021	0.037	0.093	0.198	0.036	0.009	0.100	0.101	0.177	0.227
PGExplainer	0.070	0.082	0.019	0.034	0.004	0.000	0.032	0.010	0.098	0.099	0.157	0.179
Grad(top 5)	0.080	0.085	0.042	0.081	0.087	0.175	0.059	0.013	0.126	0.107	0.222	0.263
GnnExplainer(top 5)	0.020	0.027	0.026	0.053	0.183	0.461	0.060	0.018	0.086	0.079	0.226	0.305
PGExplainer(top 5)	0.021	0.027	0.038	0.058	0.182	0.516	0.066	0.019	0.148	0.125	0.199	0.236
Grad(top 10)	0.083	0.089	0.060	0.084	0.195	0.494	0.078	0.018	0.171	0.132	0.223	0.254
GnnExplainer(top 10)	0.034	0.042	0.043	0.088	0.200	0.491	0.074	0.018	0.125	0.104	0.293	0.400
PGExplainer(top 10)	0.032	0.036	0.046	0.072	0.206	0.517	0.083	0.030	0.165	0.117	0.206	0.258
PGM-Explainer	0.233	0.339	0.096	0.207	0.000	0.000	0.089	0.028	0.212	0.198	0.260	0.338
(b) Infidelity	Infid ^{prob}	Infid ^{acc}										
INSIDE-GNN(ego)	0.133	0.062	0.163	0.188	0.000	0.000	0.766	0.806	0.369	0.452	0.273	0.349
INSIDE-GNN(node)	0.133	0.048	0.160	0.196	0.000	0.000	0.767	0.806	0.374	0.464	0.237	0.288
INSIDE-GNN(decay)	0.140	0.097	0.162	0.202	0.000	0.000	0.767	0.806	0.362	0.454	0.233	0.272
INSIDE-GNN(top 5)	0.341	0.340	0.287	0.355	0.323	0.494	0.770	0.806	0.441	0.574	0.341	0.460
INSIDE-GNN(top 10)	0.341	0.340	0.297	0.355	0.310	0.494	0.768	0.806	0.405	0.524	0.329	0.435
Grad	0.344	0.340	0.326	0.355	0.334	0.494	0.769	0.806	0.447	0.623	0.357	0.489
GnnExplainer	0.075	0.084	0.021	0.036	0.223	0.494	0.036	0.012	0.099	0.098	0.140	0.141
PGExplainer	0.082	0.086	0.024	0.039	0.353	0.494	0.038	0.012	0.098	0.096	0.157	0.185
Grad(top 5)	0.343	0.340	0.312	0.355	0.327	0.494	0.770	0.806	0.471	0.651	0.356	0.485
GnnExplainer(top 5)	0.348	0.498	0.228	0.599	0.321	0.494	0.101	0.057	0.216	0.179	0.297	0.354
PGExplainer(top 5)	0.343	0.340	0.296	0.355	0.332	0.494	0.769	0.806	0.510	0.695	0.353	0.490
Grad(top 10)	0.344	0.340	0.326	0.355	0.334	0.494	0.769	0.806	0.447	0.623	0.357	0.489
GnnExplainer(top 10)	0.343	0.474	0.197	0.491	0.308	0.494	0.105	0.054	0.206	0.180	0.282	0.343
PGM-Explainer	0.345	0.340	0.341	0.355	0.342	0.494	0.765	0.806	0.392	0.514	0.354	0.498
(c) Sparsity	DD	Proteins		BA2		Aids		BBBB		Mutagen		
INSIDE-GNN(ego)	0.544	0.410	0.011	0.822	0.805	0.822	0.805	0.822	0.805	0.717	0.717	
INSIDE-GNN(node)	0.769	0.429	0.002	0.897	0.870	0.897	0.870	0.897	0.870	0.731	0.731	
INSIDE-GNN(decay)	0.717	0.394	0.010	0.870	0.860	0.870	0.860	0.870	0.860	0.697	0.697	
INSIDE-GNN(top 5)	0.997	0.993	0.902	0.955	0.969	0.955	0.969	0.955	0.969	0.989	0.989	
INSIDE-GNN(top 10)	0.994	0.986	0.804	0.915	0.939	0.915	0.939	0.915	0.939	0.978	0.978	
Grad	0.994	0.986	0.804	0.910	0.938	0.910	0.938	0.910	0.938	0.978	0.978	
GnnExplainer	0.502	0.501	0.619	0.501	0.501	0.501	0.501	0.501	0.501	0.505	0.505	
PGExplainer	0.529	0.545	0.955	0.547	0.534	0.547	0.534	0.547	0.534	0.515	0.515	
PGM-Explainer	0.973	0.955	nan	0.855	0.884	0.855	0.884	0.855	0.884	0.956	0.956	

Table 3.5: Polarized fidelity.

(a) Fidelity Model	DD		Proteins		BA2		Aids		BBBB		Mutagen	
	$F^{0 \rightarrow 1}$	$F^{1 \rightarrow 0}$	$F^{0 \rightarrow 1}$	$F^{1 \rightarrow 0}$	$F^{0 \rightarrow 1}$	$F^{1 \rightarrow 1}$	$F^{0 \rightarrow 1}$	$F^{1 \rightarrow 0}$	$F^{0 \rightarrow 1}$	$F^{1 \rightarrow 0}$	$F^{0 \rightarrow 1}$	$F^{1 \rightarrow 0}$
INSIDE-GNN(ego)	0.489	1.000	0.572	0.795	0.000	1.000	0.353	0.035	0.965	0.137	0.543	0.809
INSIDE-GNN(node)	0.344	1.000	0.546	0.795	0.000	1.000	0.198	0.047	0.981	0.184	0.795	0.891
INSIDE-GNN(decay)	0.219	1.000	0.455	0.795	0.000	1.000	0.180	0.025	0.933	0.121	0.744	0.838
INSIDE-GNN(top 5)	0.135	0.977	0.029	0.190	0.836	1.000	0.080	0.053	0.808	0.130	0.500	0.830
INSIDE-GNN(top 10)	0.167	0.985	0.045	0.276	0.004	1.000	0.098	0.047	0.869	0.129	0.398	0.913
Grad	0.091	0.086	0.022	0.195	0.000	1.000	0.046	0.011	0.569	0.029	0.053	0.567
GnnExplainer	0.031	0.191	0.018	0.071	0.000	0.401	0.026	0.005	0.495	0.008	0.097	0.429
PGExplainer	0.029	0.186	0.017	0.066	0.000	0.000	0.023	0.007	0.511	0.002	0.072	0.345
Grad(top 5)	0.087	0.081	0.040	0.154	0.002	0.352	0.039	0.007	0.454	0.026	0.240	0.299
GnnExplainer(top 5)	0.013	0.055	0.026	0.101	0.049	0.883	0.034	0.014	0.265	0.035	0.265	0.368
PGExplainer(top 5)	0.010	0.060	0.053	0.068	0.423	0.611	0.098	0.001	0.581	0.017	0.251	0.214
Grad(top 10)	0.091	0.086	0.022	0.195	0.000	1.000	0.046	0.011	0.569	0.029	0.053	0.567
GnnExplainer(top 10)	0.023	0.078	0.033	0.187	0.000	0.994	0.067	0.006	0.419	0.030	0.317	0.527
PGExplainer(top 10)	0.012	0.083	0.070	0.076	0.077	0.968	0.155	0.000	0.556	0.014	0.212	0.331
PGM-Explainer	0.198	0.612	0.109	0.385	0.000	0.000	0.111	0.008	0.645	0.093	0.130	0.662

Table 3.6: Characterization of activation rules with numerical subgroups on BA2. We only report the subgroup whose WRAcc value is greater than 0.1.

Layer	Class	Description	WRAcc
2	0	degree=3	0.2475
2	1	clustering2=0 AND degree=2 AND triangle2_avg=0	0.207
2	1	betweenness: [0.0:0.00[AND clustering2=0.0	0.127
3	0	clustering2=0.0 AND degree2: [7:8[AND degree2_avg: [3.50:3.57[0.114
3	0	clustering2=0.0 AND degree=2 AND triangle2=0	0.101
3	0	betweenness2: [0.37:0.38[AND betweenness2_avg: [0.19:0.20[AND clustering2=0.0	0.202
3	0	betweenness2: [0.37:0.39[AND betweenness2_avg: [0.19:0.21[AND betweenness=0.07608695652173914	0.209
3	0	betweenness: [0.29:0.30[AND clustering2=0.0 AND degree==3	0.147
3	0	betweenness: [0.0:0.00[AND clustering2=0.0 AND degree2_avg: [4.0:4.17[0.162
3	1	clustering=0.0 AND degree=2 AND triangle2_avg=0.5	0.227
3	1	degree2: [7:8[AND degree2_avg: [3.50:3.60[AND degree=2 AND triangle=0	0.224
3	1	degree=2 AND triangle2=1	0.238
3	1	clustering==0.0 AND degree==2 AND triangle2==1 AND triangle==0	0.240
3	1	degree=2	0.125
3	1	clustering=0.0 AND degree=2 AND triangle2=1 AND triangle2_avg=0.5	0.232

Chapter 4

Characterizing activation rules with representative graphs

Activation rules, as defined, implemented and studied in the previous chapter, are not sufficient to fully explain Graph Neural Networks (GNNs). Even if unlike existing methods, these rules are not simply linked to the decision of the model but capture and shed light on the hidden features built by the GNN, they do not constitute an explanation understandable by a human being. To achieve this goal, we seek to identify the closest subgraph to an activation rule. That is, we are looking for the most specific graph that triggers the rule without triggering another one. The graph search is based on a Monte Carlo Tree Search directed by a proximity measure between the graph embedding and the internal representation of the rule, as well as a realism factor that constrains the distribution of the labels of the graph to be similar to that observed on the dataset. Experiments demonstrate that our method DISCERN generates realistic graphs of high quality which allows providing new insights into the respective GNN models (Veyrin-Forrer et al., 2022a,b,c).

4.1 Problem definition and desiderata

For each layer of the GNN we have a set of activation rules generated with the method INSIDE-GNN. Each of them comes with statistics from the dataset. But we do not know what information is captured by each of them. Extracting a graph or a small set of graphs that exhibit the typical structure captured by the rule would be of interest. In this process the first question to ask is what is the best language able to express this type of explanations.

The smallest representation space is probably that of the subgraphs of the dataset. We want to find the best subgraph of data instances to explain a rule. This space is finite and with other constraints such as connectedness, or taking ego-graphs, it becomes easily controllable. Moreover, all the graphs in this space come from a real example and do not contain unrealistic examples that are irrelevant in our use case. For example, an explanation in the form of a dense graph in a family of mainly planar graphs, such as a molecular graph, is useless in our case. The major problem with this space is that it is perhaps a bit too limited and might not show the generalizability of GNN.

We can then search in the same graph space as the input graph space. In this case, we should need a good heuristic to access the realism of the graph to limit the search space. We

will focus on this type of search space.

Representation spaces like PGMExplainer’s probabilistic graphs (Vu and Thai, 2020) can also be used. They have increased expressive power because they are primarily weighted directed graphs. We also want the graph produced to be a relevant structure of what is captured by the activation rule. This can mean several things. This may be the typical structure captured by the GNN. But also, it can be the ability of the GNN to resist data attacks. The metrics and method selected will impact the question to be answered. We later propose several metrics evaluating the similarities between graphs and rules.

4.2 Characterizing activation rules with subgraphs

Activation rules correspond to a part of the GNN (i.e. part of the matrix \widehat{H}^ℓ) specifically activated for a given decision (for the graphs g such that $GNN(g) = c$). However, these rules are not intelligible, and do not allow highlighting the parts of the graphs which are used for the classification. To make the rule humanly understandable, we propose to associate to each rule a subgraph. To that end, we are looking for a subgraph whose GNN embedding in layer ℓ is as close as possible to a given activation rule. This requires defining a measure of proximity between embedding and activation rule, ensuring that the graphs are realistic and defining a procedure to determine the subgraph that maximizes the proximity measure while being realistic.

4.2.1 Measuring the proximity between a graph and an activation rule

To measure the proximity between a graph and a rule, we compute the embedding of the graph by the GNN and then we compare the embedding with the rule. We propose three different measures to evaluate the proximity between a graph embedding, centered at node v , \mathbf{h}_v^ℓ and an activation rule $\mathbf{A}^\ell \rightarrow c$, with $\mathbf{A}^\ell = \{a_1, \dots, a_K\}$, $a_i \in \{0, 1\}$. These measures use $\mathbf{E}_g = \{\epsilon_1, \dots, \epsilon_K\}$ the ego-graph embedding truncated to fit the interval $[0, 1]$: $\epsilon_k = \min\left(\max(0, (\mathbf{h}_v^\ell)_k), 1\right)$. The truncation avoids distorting our measurements by extreme values.

The first measure is the cosine similarity measure:

$$(4.1) \quad \text{Cosine}(\mathbf{E}_g, \mathbf{A}^\ell) = \frac{\mathbf{E}_g \cdot \mathbf{A}^\ell}{\|\mathbf{E}_g\| \|\mathbf{A}^\ell\|} = \frac{\sum a_i \epsilon_i}{\sqrt{\sum a_i^2} \sqrt{\sum \epsilon_i^2}}$$

It is defined to equal the cosine of the angle between the two vectors, or identically be the inner product of the vectors normalized to length 1.

We can also consider to use the cross-entropy measure (equivalently the log-likelihood):

$$(4.2) \quad \text{Cross-entropy}(\mathbf{E}_g, \mathbf{A}^\ell) = \sum a_i \log(\epsilon_i)$$

that increases with the number of components that have large values in the embedding of the ego-graph for the components activated in the activation rule.

We can also consider a set of activation rules R and search for an ego-graph that specifically activates a rule \mathbf{A}^ℓ but not the other rules, i.e. this ego-graph activates components

outside the rule only if it does not trigger another rule. We propose the following expression to measure this:

$$(4.3) \quad \text{Relative-CE}(\mathbf{E}_g, \mathbf{A}^\ell, R) = \text{Cross-entropy}(\mathbf{E}_g, \mathbf{A}^\ell) - \max_{\mathbf{r} \in R} \text{Cross-entropy}(\mathbf{E}_g, \mathbf{r})$$

4.2.2 Realism factor

Graphs can have an embedding close to an activation rule without being realistic and be very different from graphs in the dataset. To avoid this, we associate a realism score with each graph. This factor depends both on the probability that two vertices are connected according to their type, and on the degree distribution for each vertex type.

Let $P_{i,j}$ be the probability of having an edge with endpoints of type i and j ($i, j \in T$). Let $P_{deg}(k|i)$ be the probability for a node of type i of being of degree k . As we are considering subgraphs of the graphs of the dataset, we do not want to penalize graphs whose degree is smaller than expected. Thus, we propose to use the value $d_{k|i} = P_{deg}(k|i) + \sum_{x>k} \frac{P_{deg}(x|i)}{2^k}$ to increase the value with the probabilities associated with higher degrees. We transform this value in a probability measure with $D_{k|i} = \frac{d_{k|i}}{\sum_x d_{x|i}}$. Thus, the realism score is calculated by

$$(4.4) \quad \text{Realism}(g = (V, E, L)) = \frac{\sum_{(u,v) \in E} \log(P_{L(u),L(v)})}{\#E} + \frac{\sum_{u \in V} \log(D_{d_u|L(u)})}{\#V}$$

with $L(u)$ the type of node u and d_u its degree. This realism value is added to the similarity measure between the activation rule and the graph embedding to form the score used to evaluate a graph quality:

$$(4.5) \quad \text{Score}(g, \mathbf{A}^\ell, R) = \beta \times m(\mathbf{E}_g, \mathbf{A}^\ell, R) + (1 - \beta) \times \text{Realism}(g)$$

with β a hyperparameter whose value is fixed empirically (see Section 4.4) and m one of the tree measures defined by equations (1)–(3).

4.3 DISCERN method

We propose to use Monte Carlo Tree Search (MCTS) to find a realistic subgraph whose embedding is similar to an activation rule. The objective is to generate an ego-graph g_t that maximizes $\text{Score}(\mathbf{E}_t, \mathbf{A}^\ell, R)$. Each node t of the search tree represents an ego-graph g_t centered at v_t whose GNN embedding is \mathbf{E}_t . A value M_t is also associated to t : it is the sum of $\text{Score}(\mathbf{E}_t, \mathbf{A}^\ell, R)$ values evaluated on the terminal nodes of the subtree rooted in t .

Each node of the tree is obtained by adding an edge to the graph of its parent node. This process stops when a terminal condition is satisfied. In our algorithm, we consider three terminal conditions:

1. The diameter of the graph is greater than ℓ ,
2. The number of edges is greater than *min-edges*,
3. The number of vertices is greater than *min-vertices*.

If one of the three conditions mentioned above is satisfied, the graph is considered terminal (see method **isTerminal** in function **findChild** line 3, and in Function **rollout** line 2).

The tree is partially explored favoring the parts most likely to lead to high-scoring graphs. From an intern node t , terminal nodes (satisfying one of the terminal conditions) are randomly generated in order to be able to evaluate their score. This process is called rollout. The obtained score is then back-propagated to all ancestors of t in their variable M_t . The rationale behind the value M_t is to estimate the quality of the descendants of t explored so far. This value is then adjusted to trade-off exploitation of the current examined graphs, and exploration of new ones. We use the classical the Upper Confidence Bound UCB1 to guide the selection of the tree node to be expanded: We select the one that maximizes UCB1.

$$UCB1(t) = \frac{M_t}{n_t} + c \sqrt{\frac{\log(N(t))}{n_t}},$$

where c is a constant number, M_t is the sum of the values **Score** for all terminal nodes descendant of t that have been explored so far, n_t is the number of terminal nodes expanded from node t during previous iterations of the algorithm, and $N(t)$ is the number of times the parent of t has been visited so far. It should be mentioned that the constant c in UCB1 plays an important role to make a balance between exploration and exploitation. If c is too large, the algorithm acts like a pure random algorithm (i.e. more exploration) and if c is too small, then exploitation rate will be increased and may get stuck in local maxima. In our experiments, c is set to 0.5.

Algorithm 3 DISCERN

Require: N : Number of epochs, **Score** ($\cdot, \mathbf{A}^\ell, R$): the measure to maximize, T : the set of node labels.

Ensure: $best_graph$ the graph that maximizes **Score**(E_g, \mathbf{A}^ℓ, R) among all explored graphs.

```

1:  $root.leaf \leftarrow \mathbf{True}$ 
2:  $root.visit \leftarrow 0$ 
3:  $root.value \leftarrow 0$ 
4:  $best\_value \leftarrow -\infty$ 
5: for epoch = 1 to  $N$  do
6:    $t \leftarrow exploreChild(root)$ 
7:   if ( $t.leaf = \mathbf{True}$ ) then
8:     if ( $t.visit = 0$ ) then
9:       [ $value, explored\_graph$ ]  $\leftarrow rollout(t)$ 
10:       $backPropagate(t, value)$ 
11:     else
12:        $first\_child \leftarrow generateChildren(t)$ 
13:       [ $value, explored\_graph$ ]  $\leftarrow rollout(first\_child)$ 
14:        $backPropagate(first\_child, value)$ 
15:     end if
16:   end if
17:   if ( $value > best\_value$ ) then
18:      $best\_value \leftarrow value$ 
19:      $best\_graph \leftarrow explored\_graph$ 
20:   end if
21: end for
22: return  $best\_graph$ 

```

Let's consider how DISCERN works in detail. Algorithm 3 starts by building the tree root node, that is a leaf, its number of visits n_t equals 0 as well as its value M_t . DISCERN consists

Algorithm 4 exploreChild(*t*)

```

1: if (t.leaf = False) then
2:   best_child ← findChild(t)
3:   if (best_child ≠ None) then
4:     exploreChild(best_child)
5:   else
6:     exploreChild(t.parent)           ▷ the subtree rooted in t has been completely explored
7:   end if
8: else
9:   return t
10: end if

```

Algorithm 5 findChild(*t*)

```

1: best_child ← None
2: for s ∈ children(t) do
3:   if (isTerminal(s.g) = False) then
4:     if  $UCB1(\textit{best\_child}) \leq UCB1(\textit{s})$  then
5:       best_child ← s
6:     end if
7:   end if
8: end for
9: return best_child

```

of some iterations (epochs) whose number is an input parameter N (line 6 to 16 of DISCERN). Each iteration starts by calling the method **ExploreChild** (line 5 of DISCERN) on the root node. This method returns the next tree node that had to be explored, that is to say the one 1) whose path from root is made of nodes with maximal UCB1 values among their brothers, 2) that is a leaf and 3) that has still to be explored (it is not a terminal node). If the current node is not a leaf (line 1 of **exploreChild**), **findChild** is called to take among the children that are not terminal, the one with the best UCB1 value. If such a node exists, **exploreChild** is recursively called on it (line 4). Otherwise, the subtree has completely been explored and the exploration goes up to the parent node to examine another branch (line 6).

Then, DISCERN explores the identified node. If the node has never being expanded ($t.visit = 0$ at line 8), then a rollout is performed and the obtained value is back-propagated to the tree root. Otherwise, the node is expanded with children and a rollout is performed on the first whose value is back-propagated to the root. These three functions (**generateChildren**, **rollout**, **backPropagate**) are as follows. **generateChildren** creates as many children then there are possible graphs with one edge in addition to the current graph. Those edges can be between two nodes of the graph ($v \in V$ line 6) or between a node of the graph and a new node with one of the possible labels from T ($v \in W$ line 6). These edges have to be valid (see Figure 4.1 and explanation below). The (arbitrary) first child is return by the function. The function **rollout** simulates a new graph created by taking repeatedly uniformly at random a valid edges and adding to the current graph until the graph is terminal. It returns the graph and its score. If the score is better than the one encountered so far, the *best_graph* is updated (see lines 17 to 20 of DISCERN). Finally, **backPropagate** updates the M_t and n_t values of the tree nodes ($t.value$ and $t.visit$) until reaching the tree root.

Only valid edges can be added to a current graph g centered in node v with radius r .

Algorithm 6 Additional sub-functions

```

generateChildren(t):
1: first_child ← None
2:  $g \leftarrow$  the graph associated to  $t$ 
3:  $V \leftarrow$  the vertices of the graph  $g$ 
4:  $W \leftarrow$  a set of new vertices with label in  $T$ 
5: for  $u \in V$  do
6:   for  $v \in V \cup W$  do
7:     if (isValid( $u,v$ )) then
8:        $g'$  is the graph  $g$  with edge  $(u, v)$ 
9:        $t' \leftarrow t.add\_child(g')$ 
10:       $t'.leaf \leftarrow \mathbf{True}$ ,  $t'.value \leftarrow 0$ ,  $t'.visit \leftarrow 0$ 
11:      if (first_child = None) then
12:        first_child  $\leftarrow t'$ 
13:      end if
14:    end if
15:  end for
16: end for
17: return first_child

```

```

rollout(t):
1:  $g, g_2 \leftarrow$  the graph associated to  $t$ 
2: while (isTerminal( $g$ ) = False) do
3:    $g_2 \leftarrow g$ 
4:   Take a random edge  $e$  among the valid edges that can be added to  $g$ 
5:    $g \leftarrow$  Simulate a new graph  $g$  with edge  $e$  added
6: end while
7: return [Score( $E_g, A^\ell, R$ ),  $g_2$ ]

```

```

backPropagate(t,value):
1: while ( $t \neq \mathbf{None}$ ) do
2:    $t.visit \leftarrow t.visit + 1$ 
3:    $t.value \leftarrow t.value + value$ 
4:    $t \leftarrow t.parent$ 
5: end while

```

An edge is valid (see **isValid** method in **rollout** and **generateChildren**) if adding it does not change the graph embedding in layer $r - 1$. That is to say, for each node $x \in g$, $dist(v, x)$ does not change when adding the edge. To that end, three conditions have to be met:

- Adding an edge cannot reduce the shortest distance of a node to v . The counter example (a) in Figure 4.1 illustrates this condition.
- An edge can be added between two nodes of g , if their distance to v is greater or equal to $radius - 1$. The counter example (b) in Figure 4.1 illustrates this condition as well as

examples (2) and (3).

- An edge between a node x of the graph and a new node can be added only if $\text{dist}(v, x) \geq \text{radius} - 1$. The counter example (c) in Figure 4.1 illustrates this condition, as well as examples (1) and (4).

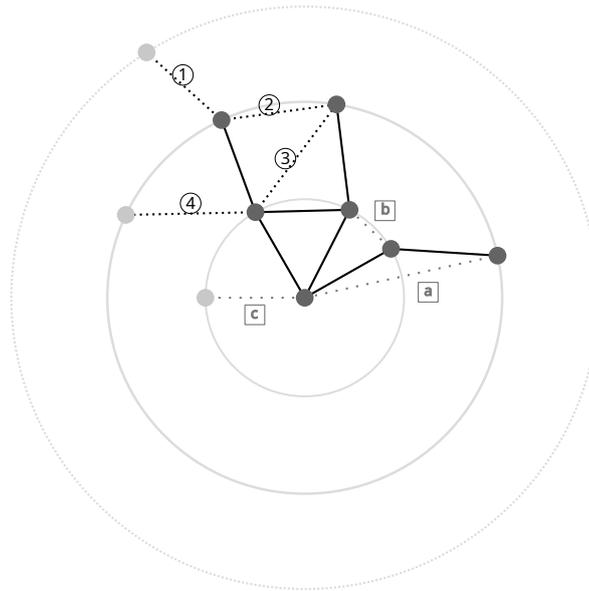


Figure 4.1: Considering a graph (solid lines), some edges can be added (dark dot lines), while other cannot (light gray lines). **Examples – (1) and (4):** edge with a new node and a node at distance from center $\geq \text{radius} - 1$; (2) edge between two nodes at a distance to the center equal to the radius of the graph; (3) edge between a node at distance radius to the center, and a node at distance $\text{radius} - 1$. **Counter examples – (a):** a new edge cannot create a shortcut between existing nodes; (b): no edge can be added between nodes that both are at a distance $< \text{radius}$ from the center of the graph; (c): a new node cannot be connected to a node at distance $< \text{radius} - 1$.

4.4 Experiments

In this section, we evaluate DISCERN through several experiments. We first describe synthetic and real-world datasets and the experimental setup. Then, we discuss some examples of subgraphs generated by our method and the baselines. Eventually, we present a quantitative study of our method as well as some comparisons against several baselines. DISCERN has been implemented in Python and the experiments have been done on a machine equipped with 8 Intel(R) Xeon(R) W-2125 CPU @ 4.00GHz cores 126GB main memory, running Debian GNU/Linux. The code and the data are available ¹.

¹<https://doi.org/10.5281/zenodo.7208320>

4.4.1 Datasets and experimental setup

Experiments are performed on six graph classification datasets whose main characteristics are given in Table 4.1. BA2 (Ying et al., 2019) is a synthetic dataset generated with Barabasi-Albert graphs and hiding either a 5-cycle (negative class) or a “house” pattern (positive class). The other datasets (Aids (Morris et al., 2020), BBBP (Wu et al., 2017), Mutagen (Morris et al., 2020), DD (Dobson and Doig, 2003), Proteins (Borgwardt et al., 2005)) correspond to real molecules, and the class identifies important properties in Chemistry or Drug Discovery (i.e., possible activity against HIV, permeability and mutagenicity). Table 4.1 shows that the datasets are diverse, each having its own specificity. BA2 is synthetic and simple to interpret. BBBP and Aids are very unbalanced. DD has a large number of node attributes and is made of large graphs. Mutagen and Proteins are similar datasets, with Proteins graphs denser than Mutagen ones. All these characteristics witness that the benchmarks we consider are diverse. This supports thorough and systematic experimental study.

A 3-convolutional layer GNN (with $K = 20$) is trained on each dataset. Accuracy measures obtained on test sets are given in Table 4.1 column Acc. We use the method INSIDE-GNN to mine the GNN activation vectors \mathbf{h}_v^ℓ to discover the activation rules \mathbf{A}^ℓ . We extract at most ten rules per layer and for each class $\{0, 1\}$ as explained in Veyrin-Forrer et al. (2022). On some datasets, less than 10 rules per layer and per class are needed to describe the inner workings of a GNN (see Table 4.1 column #Rules). Our goal is to provide a representative graph for each rule with DISCERN. We generate graphs with labels appearing in at least 100 nodes in the dataset (see Table 4.1 column #Freq_T).

Table 4.1: Main characteristics of the datasets.

Dataset	# \mathcal{G}	(# 0,#1)	#T	\bar{V}	\bar{E}	Acc.	#Rules	#Freq_T
DD	1168	(681, 487)	90	268	1352	0.692	47	21
Aids	2000	(400, 1600)	38	15.69	32.39	0.99	60	7
Mutagen	4337	(2401, 1936)	14	30.32	61.54	0.786	60	10
BBBP	1640	(389, 1251)	13	24.08	51.96	0.787	60	6
Proteins	1113	(663, 450)	3	39	145	0.768	29	3
BA2(syn)	1000	(500, 500)	-	25	50.92	0.97	20	-

This experimental study aims to answer the following questions:

- How does DISCERN behave?
- Are the activation rules good?
- Are the generated graphs representative and realistic?
- Which is the best metric?
- How does DISCERN behave against baselines?

To that end, especially the latter question, we compare our method against to both instance-level and model-level explanation methods. For instance-level methods, we consider four state of the art methods: **GNNExplainer** (Ying et al., 2019), **PGExplainer** (Luo

et al., 2020), **PGM-Explainer** (Vu and Thai, 2020), and **GraphSVX** Duval and Malliaros (2021).

We also examine 3 model-level baselines:

- **Random** generates graphs randomly by calling the Roll-out function 250x.
- **XGNN++** is an extension of XGNN (Yuan et al., 2020a) to our problem. We integrate Cos, CE and Relative-CE metrics as function optimized by XGNN. We set a budget that corresponds to 5000 calls to the GNN to do a fair comparison with DISCERN so that both methods do exactly the same number of calls to the GNN.
- **DISC-GSPAN** is a sound and complete method that aims at discovering discriminant subgraphs within a collection based on GSPAN enumeration (Yan and Han, 2002) while exploiting some tight upper bounds on the WRAcc measure Lavrac et al. (2004). The input dataset contains the set of ego-network of nodes that support the activation rule as the “positive class” and the ego-network of nodes not involved in the support of the rule as the negative class. Then, DISC-GSPAN consists in computing the top-k subgraphs that are discriminant for the positive class.

By default, we empirically set $min_edges=10$ and $min_vertices=6$ as terminal conditions for DISCERN in the following experiments. We also empirically set the hyper-parameter β to 0.5 (see Figure 4.4).

4.4.2 Studying DISCERN behavior

For an activation rule, DISCERN takes between 20 and 50 seconds for 5000 epochs. The deeper the GNN layer, the larger the ego-graph to be investigated, the higher the execution time.

Quality with respect to the number of epochs

Figure 4.2 (a–c) shows the maximum value of measure m (Cosine, Cross-entropy and Relative-CE) obtained on the graphs generated by DISCERN for each dataset with respect to the number of epochs. The value on the y-axis is the maximum value of the measure m evaluated on the explored graphs in the MCTS (*explored_graph* lines 9 and 13 in DISCERN function) at the corresponding x-axis epoch. For each dataset, the values are aggregated over all activation rules. The graphs show an asymptotic convergence for all the curves. Yet, the convergence is faster for some combinations of metric and dataset (e.g., Cosine on Mutagen) than others (e.g., Cosine on DD). This experiment demonstrates that DISCERN correctly learns which parts of the MCTS search space hold promise for generating high value graphs.

Are the results actually good?

We observe in the latter experiment that the graphs generated by DISCERN become better when increasing the number of epochs. However, one can wonder if the obtained graphs g_{mcts} are actually good compared to the ego-graphs $g_{support}$ that support the activation rules. Especially, we want to answer the following question:

Are the obtained scores higher than those of the ego-graph taken randomly in the dataset?

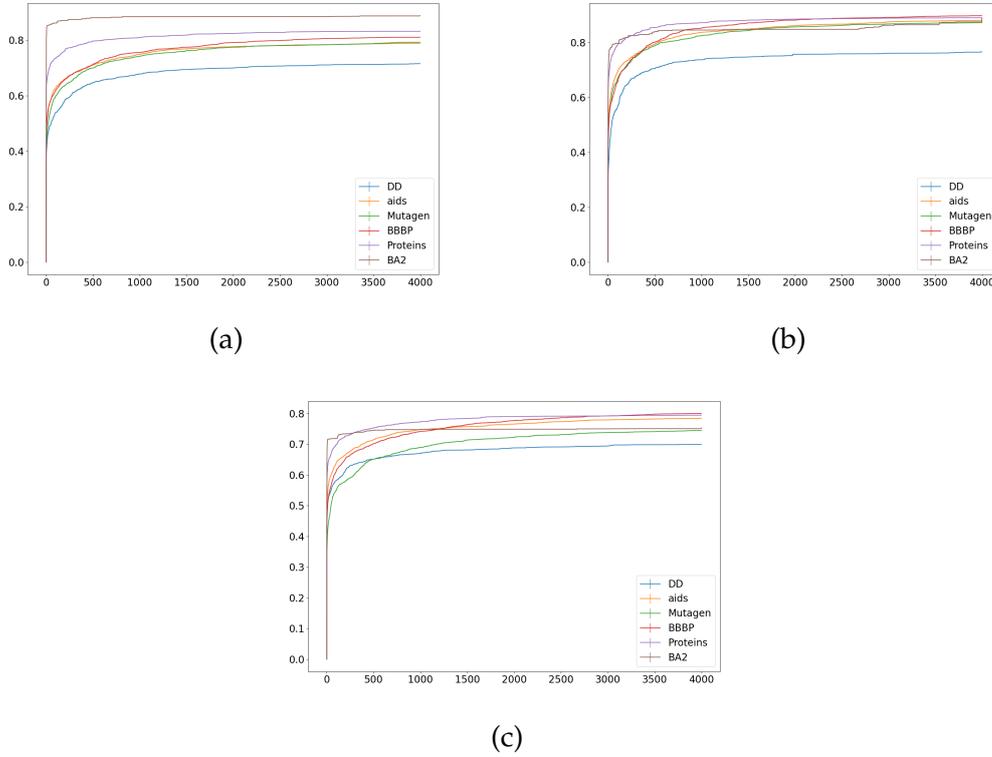


Figure 4.2: Maximum m values on the graphs generated by DISCERN for the metrics m (a) Cosine, (b) Cross-entropy, (c) Relative-CE on each dataset when varying the number of epochs.

To this end, we consider a sample of nodes known to be embedded in the targeted subspace as they support the activation rules. Similarly, we take a sample of random nodes. For both samples respectively denoted $E_{g_{sample}}$ and $E_{g_{rand.}}$, we consider the ego-graphs whose size equals the layer of the corresponding activation rule. For each sample, we compute the score values for each quartile (Q1, Q2, Q3) that partition the sample ordered by values into 4 subsets of equal sizes. We also consider the maximal value Q_{max} . We report in Table 4.2 the improvement factor of the graphs produced by DISCERN compared to each quartile, i.e. the ratio $\mathbf{Score}(E_{g_{mcts}}, A^\ell, R)/Q_x$. Results are aggregated over all activation rules for each dataset.

Interestingly, we observe that, in most of the cases we obtain values greater than 1, even when we compare the generated graphs to Q_{max} (i.e., max value of the sample). This demonstrates the high quality of the graphs generated by DISCERN. Having values greater than 1 for Q_{max} of the supporting node sample means that the graphs we generate embed well in the targeted subspace, with less activated components outside this space than for supporting nodes. Indeed, the metrics we consider penalize activated components of the ego-graph that are not activated in the activation rules. Obviously, the improvement factor is always better when considering the random sample than the node support sample. Nevertheless, this gives interesting insights showing the ability of metrics to separate support nodes from

random nodes well. It is important to notice that the difference between improvement factors between the samples $E_{g_{sample}}$ and $E_{g_{rand}}$ is much larger for Cosine and Cross-Entropy than Relative-CE, especially for Q_3 and Q_{max} . Based on these observations, we can conclude that Cosine and Cross-Entropy metrics has a better ability to separate better than Relative-CE.

Table 4.2: Avg. improvement factor of the score provided by DISCERN against the score of the quartiles of two distributions: (1) nodes from rule support (supp) and (2) some random nodes (rand).

Dataset	Measure	Samp.	Q_1	Q_2	Q_3	Q_{max}
DD	Cosine	supp	1.19×10^8	2.72	1.56	1.28
	Cosine	rand	8.12×10^9	5.04×10^9	2.83×10^9	1.03×10^9
	Cross-Entropy	supp	1.26	1.16	1.14	1.12
	Cross-Entropy	rand	9.57×10^{11}	4.21×10^{11}	2.33×10^{11}	1.09×10^{11}
	Relative-CE	supp	1.35	1.05	0.96	0.89
	Relative-CE	rand	3.79×10^{11}	3.14	2.37	1.68
Aids	Cosine	supp	1×10^8	1.94	1.63	1.46
	Cosine	rand	7.34×10^9	1.52×10^9	7.50×10^8	2.30×10^8
	Cross-Entropy	supp	1.25	1.17	1.15	1.14
	Cross-Entropy	rand	9.48×10^{11}	1.44×10^{11}	6.42×10^{10}	7.82×10^9
	Relative-CE	supp	1.35	1.10	1.04	0.99
	Relative-CE	rand	4.02×10^{11}	3.66	2.37	1.74
Mutagen	Cosine	supp	12.59	2.69	2.14	1.66
	Cosine	rand	7.33×10^9	1.42×10^9	1.01×10^9	3.84×10^8
	Cross-Entropy	supp	1.22	1.16	1.15	1.13
	Cross-Entropy	rand	1.02×10^{12}	1.28×10^{11}	1.01×10^{11}	3.51×10^{10}
	Relative-CE	supp	1.32	1.17	1.12	1.06
	Relative-CE	rand	4.38×10^{11}	6.71	3.76	2.17
BBBP	Cosine	supp	5.27	2.31	1.78	1.58
	Cosine	rand	7.61×10^9	1.16×10^9	5.19×10^8	1.37×10^8
	Cross-Entropy	supp	1.21	1.15	1.14	1.13
	Cross-Entropy	rand	9.86×10^{11}	1.16×10^{11}	6.22×10^{10}	1.17×10^{10}
	Relative-CE	supp	1.27	1.09	1.03	0.99
	Relative-CE	rand	4.20×10^{11}	11.33	2.18	1.61
Proteins	Cosine	supp	4.49	1.67	1.45	1.29
	Cosine	rand	5.61×10^9	3.1×10^8	3.80	1.71
	Cross-Entropy	supp	1.32	1.22	1.20	1.19
	Cross-Entropy	rand	1.01×10^{12}	1.68×10^{10}	10.72	3.82
	Relative-CE	supp	1.31	1.06	1.00	0.96
	Relative-CE	rand	3.72×10^{11}	5.16	2.11	1.44
BA2	Cosine	supp	1.84×10^7	4.54	2.82	1.78
	Cosine	rand	7.25×10^9	1.77×10^9	7.71×10^8	43.69
	Cross-Entropy	supp	5.01×10^9	2.48	2.02	1.75
	Cross-Entropy	rand	1.29×10^{12}	1.23×10^{11}	8.59×10^{10}	1.26×10^{10}
	Relative-CE	supp	1.24	1.00	0.98	0.99
	Relative-CE	rand	5.69×10^{11}	3.74×10^5	2.74	1.63

Which is the best metric?

To deeper investigate the quality of the generated graph, we now study in Figure 4.3 the L2 norm between $E_{g_{mcts}}$ and, $E_{g_{sample}}$ or $E_{g_{rand.}}$. We observe that both CE and Relative-CE tend to provide graphs whose embeddings are more similar of the support sample than Cosine. Cosine reports greater L2 norm to $E_{g_{rand.}}$ than the two other measures but this is not significant.

Similarly, we investigate the distances between the graphs generated by DISCERN with the three metrics, the graphs from the support sample and the ones from the random sample, for each activation rule and dataset. We consider the graph edit distance Gao et al. (2010), an error tolerant matching technique between graphs that is computed directly from the graphs and not their embedding. Results are reported in Table 4.3. We consider, for each rule and each measure, the set G_m of graphs produced by 10 runs of DISCERN, or a subset of 10 graphs from the support or the random samples. For set $m1$ in row and set $m2$ in column, we report the value $\text{mean}_{g \in G_{m1}} \max_{h \in G_{m2}} \frac{GED(g,h)}{\#V_g + \#E_g}$, with GED, the graph edit distance. Results are similar to what we observe when studying the L2 norm between vectors: Graphs from the support sample are closer to graphs generated by DISCERN with Cross-Entropy than those generated with Relative-CE and Cosine metrics (see supp related rows in Table 4.3). Furthermore, graphs generated by DISCERN – particularly with Cross-Entropy metric – are far away from the graphs of the random samples (see rand related rows in Table 4.3).

Table 4.3: Graph edit distances on Aids, Mutagen and BBBP. Distances are normalized by ($\#vertices + \#edges$) of the row graphs.

Aids	Cosine	Cross-Entropy	Relative-CE	supp	rand
Cosine	0.	0.751	0.755	0.582	0.647
Cross-Ent.	0.746	0.	0.643	0.499	0.654
Relative-CE	0.749	0.658	0.	0.517	0.661
Supp.	0.747	0.725	0.724	0.	0.526
Rand.	0.753	0.758	0.734	0.391	0.
Mutagen	Cosine	Cross-Ent.	Rel.-CE	Supp.	Rand.
Cosine	0.	0.755	0.712	0.653	0.66
Cross-Ent.	0.73	0.	0.622	0.61	0.678
Relative-CE	0.726	0.683	0.	0.599	0.672
Supp.	0.696	0.654	0.643	0.	0.419
Rand.	0.731	0.712	0.689	0.376	0.
BBBP	Cosine	Cross-Ent.	Rel.-CE	Supp.	Rand.
Cosine	0.	0.701	0.713	0.609	0.633
Cross-Ent.	0.689	0.	0.630	0.559	0.625
Relative-CE	0.718	0.657	0.	0.559	0.622
Supp.	0.694	0.645	0.636	0.	0.4
Rand.	0.716	0.685	0.663	0.352	0.

These experiments demonstrate that the ability of DISCERN to generate representative

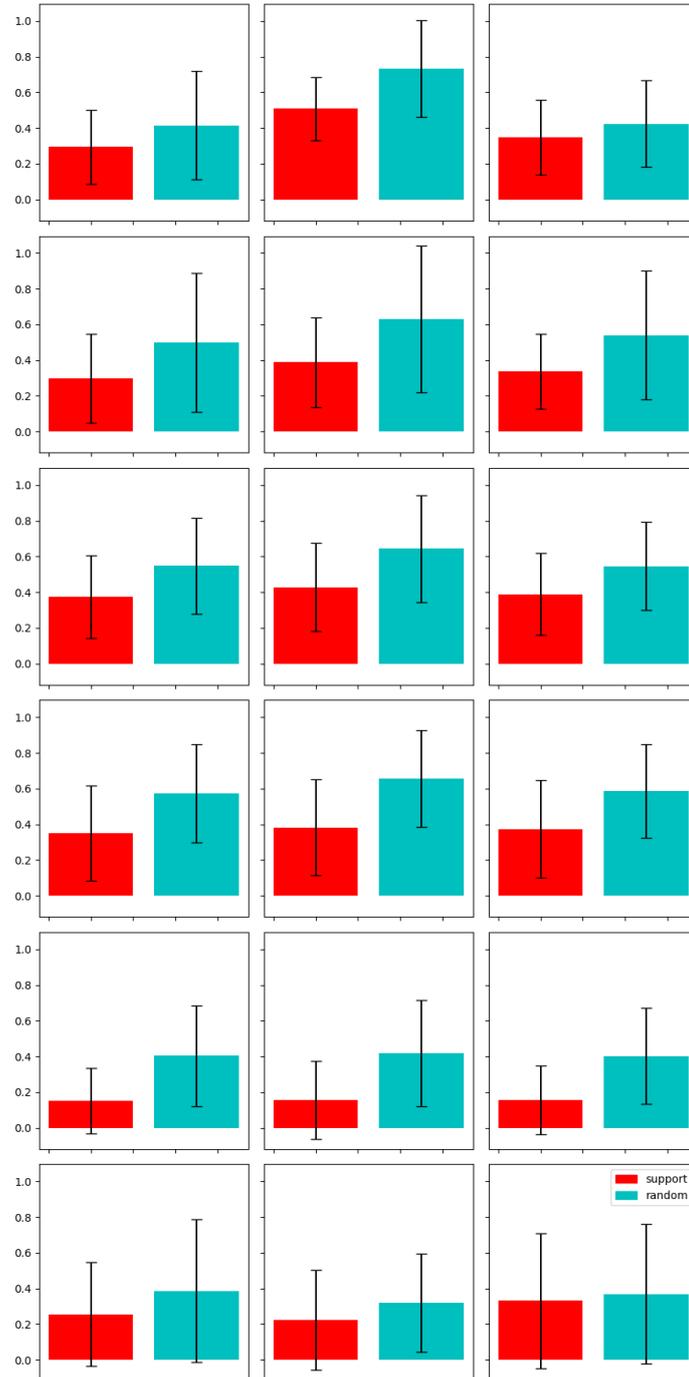


Figure 4.3: Average L2 norm between the generated graphs and support (red) and random (cyan) nodes for all activation rules. Dataset are (from top to bottom): DD, Aids, Mutagen, BBBP, Proteins, BA2. Left, middle and right columns depict Cosine, Cross-Entropy and Relative-CE measures. The lower the red histogram, the higher the cyan one, the better.

graphs for the activation rules regardless of the three measures Cosine, Cross-Entropy, and Relative-CE. These experiments suggest that Cross-Entropy is slightly better than the two other measures. Nevertheless, the difference are not significant.

4.4.3 Comparison to instance-level methods

We consider a ground-truth free metric to compare the methods. We opt for the *Fidelity* (Pope et al., 2019) which is defined as the difference of predicted probability between the predictions on the original graph and the one obtained when masking part of the graph based on the explanations:

$$\text{Fidelity} = \frac{1}{N} \times \sum_{i=1}^N (f(g_i)_{y_i} - f(g_i \setminus m_i)_{y_i})$$

where m_i is the mask, $g_i \setminus m_i$ is the complementary mask and $f(g)_{y_i}$ is the prediction score for class y_i . Similarly, we can study the prediction change by keeping important features (i.e., the mask) and removing the others as Infidelity measure does:

$$\text{Infidelity} = \frac{1}{N} \times \sum_{i=1}^N (f(g_i)_{y_i} - f(m_i)_{y_i})$$

The higher the fidelity, the lower the infidelity, the better the explanation.

Obviously, masking all the input graph would have important impact to the model prediction. Therefore, the former measures should not be studied without considering the *Sparsity* metric that aims to measure the fraction of graph selected as mask by the explainer:

$$\text{Sparsity} = \frac{1}{N} \sum_{i=1}^N \left(1 - \frac{|m_i|}{|g_i|} \right),$$

where $|m_i|$ denotes the size of the mask m_i and $|g_i|$ is the size of g_i (the size includes the number of nodes, of edges and the attributes associated to them). Based on these measures, a better explainability method achieves higher fidelity, lower infidelity while keeping a sparsity close to 1.

Several policies to build a mask directly from an activation rule are possible. We opt for the simplest policy **AR(node)** which takes as a mask only the nodes that are covered by the activation rule and the edges adjacent to these nodes. This policy allows assessing how relevant are the activation rules. We also consider the graphs generated by DISCERN as masks. To this end, for an instance graph, we select among all the rules that are activated, the related generated graph that maximizes the trade-off between Fidelity and Infidelity. The selected subgraph is then used as a mask for explanation.

The average time to provide an explanation ranges from 8ms to 30ms for AR(node). This is faster than PGM-Explainer (about 5s), GNNExplainer (80ms to 240ms) and SVXexplainer (40ms to 60ms). It remains slightly slower than PGExplainer (6ms to 20ms). DISCERN is slower than AR(node) (about 1s). Even if the graph is already built for each activation rule, it requires several graph inclusion computations to provide a mask for an instance.

Table 4.4(a) outlines the performance of the explainers based on the Fidelity measure. Results show that **AR(node)** outperforms the baselines. These results must be analysed while considering the sparsity (see Table 4.4(c)). Except for Proteins and BA2, **AR(node)**

Table 4.4: Assessing the explanations with several metrics. A better explainer achieves higher fidelity, lower infidelity while keeping a sparsity close to 1.

ModelsDatasets	DD	Aids	Mutagen	BBBP	Proteins	BA2
(a) Fidelity						
AR(node)	0.490	0.175	0.582	0.362	0.359	0.342
DISCERN(Cosine)	0.124	0.031	0.319	0.172	0.086	0.471
DISCERN(Cross-Entropy)	0.130	0.026	0.360	0.188	0.072	0.460
DISCERN(Relative-CE)	0.113	0.054	0.366	0.198	0.070	0.446
GnnExplainer	0.077	0.036	0.177	0.100	0.021	0.093
PGExplainer	0.070	0.032	0.157	0.098	0.019	0.004
PGM-Explainer	0.059	0.080	0.123	0.212	0.073	0.222
SVXexplainer	0.010	0.003	0.039	0.008	0.006	0.004
(b) Infidelity						
AR(node)	0.133	0.767	0.237	0.374	0.160	0.000
DISCERN(Cosine)	0.239	0.119	0.290	0.193	0.069	0.149
DISCERN(Cross-Entropy)	0.228	0.125	0.196	0.411	0.056	0.124
DISCERN(Relative-CE)	0.271	0.039	0.195	0.134	0.076	0.246
GnnExplainer	0.075	0.036	0.140	0.099	0.021	0.223
PGExplainer	0.082	0.038	0.157	0.098	0.024	0.353
PGM-Explainer	0.343	0.766	0.347	0.482	0.324	0.296
SVXexplainer	0.343	0.771	0.356	0.489	0.292	0.341
(c) Sparsity						
AR(node)	0.769	0.897	0.731	0.870	0.249	0.002
DISCERN(Cosine)	0.983	0.664	0.741	0.630	0.649	0.641
DISCERN(Cross-Entropy)	0.977	0.721	0.742	0.623	0.645	0.674
DISCERN(Relative-CE)	0.979	0.478	0.195	0.707	0.669	0.676
GnnExplainer	0.502	0.501	0.505	0.501	0.986	0.804
PGExplainer	0.529	0.547	0.515	0.534	0.545	0.955
PGM-Explainer	0.976	0.862	0.900	0.973	0.957	0.746
SVXexplainer	0.965	0.988	0.931	0.940	0.991	0.943

provides explanations which have a comparable sparsity to the baselines. The quality of the explanations are also assessed with the Infidelity metrics in Table 4.4(b). **AR(node)** is outperformed by GnnExplainer and PGExplainer. This suggests that a rule taken in isolation does not allow a correct classification of a graph. It is undoubtedly necessary to consider combinations of graphs to explain a decision. Interestingly, DISCERN that builds on activation rules often provides better results in term of Infidelity than **AR(node)**, achieving score that are similar to the state of the art methods while having better fidelity and sparsity scores than these methods in most of the cases. Finally, the generated graph brings further interpretability on activation rules without altering too much the performance of the explainer directly built from these rules. All together, these results suggest that the activation rules allow identifying relevant representation space within the GNNs.

4.4.4 Comparison to model-level baselines

We now assess DISCERN against model-level explanation baselines. Notice that Random and DISC-GSPAN are not directed by the measures we introduce. On the contrary, XGNN++ optimizes either Cosine, Cross-Entropy or Relative-CE measures. We compare these baselines against DISCERN with a similar experimental protocol as in Section 4.4.2. For each activation rule, we study the L2 norm between the best graph generated by each method and the activation rule A^ℓ . We assume that each rule is represented by a vector whose values equal to 1 for components inside the rule, 0 otherwise. Results are reported in Table 4.5. Note that DISC-GSPAN fails for BA2 and Proteins because of the small number or the absence of labels on nodes, which makes extraction impossible. For all measures, DISCERN provides graphs that are better embedded within the target space than any other method. On average, $\text{DISCERN}_{\text{Cos}}$ outperforms the best solution based on either XGNN or DISC-GSPAN of about 12%.

Table 4.5: Average L2 norm between the graphs provided by each method aggregated over all activation rules. The lower the value, the better.

	DD	Aids	Mutagen	BBBP	Proteins	BA2
Random	4.18	4.12	4.63	4.34	5.32	6.59
DISC-GSPAN	3.52	3.62	3.90	3.47	•	•
XGNN++(Cosine)	4.19	4.04	4.61	4.26	5.39	5.58
XGNN++(Cross-Entropy)	4.09	3.97	4.46	4.15	5.23	6.57
XGNN++(Relative-CE)	4.16	4.01	4.50	4.22	5.15	6.56
$\text{DISCERN}_{\text{Cosine}}$	3.11	3.18	3.47	3.15	4.42	4.61
$\text{DISCERN}_{\text{Cross-Entropy}}$	3.16	3.47	3.76	3.45	4.54	4.64
$\text{DISCERN}_{\text{Relative-CE}}$	3.34	3.47	3.75	3.48	4.55	4.59

We study the importance of hyperparameter β (see section 4.2.2) for generating realistic graphs. To this end, we assess how realistic the generated graphs are compared to those from the related dataset. We compute the maximum common subgraph (MCSG) between the graph returned by DISCERN and each graph from the dataset. In Fig 4.4, we report the size of the MCSG normalized by the size of the generated graph when β varies. When this value reaches one, it means that the generated graph is a subgraph of a graph from the dataset. Therefore, the closer the value is to 1, the more realistic the generated graphs are. We observe that the graphs generated by DISCERN become more realistic when the hyperparameter β increases. Interestingly, Cross-Entropy and Relative-CE metrics allow obtaining a more realistic graph than Cosine metric for the same value of β . For information, the realism factor is also reported for the three baselines. By definition, DISC-GSPAN provides more realistic graphs since it mines frequent subgraphs. Hence, the realism factor is equal to 1. XGNN++ fails to generate realistic graphs whatever the metrics with a value ranging between 0.5 and 0.7 which still remains better than random.

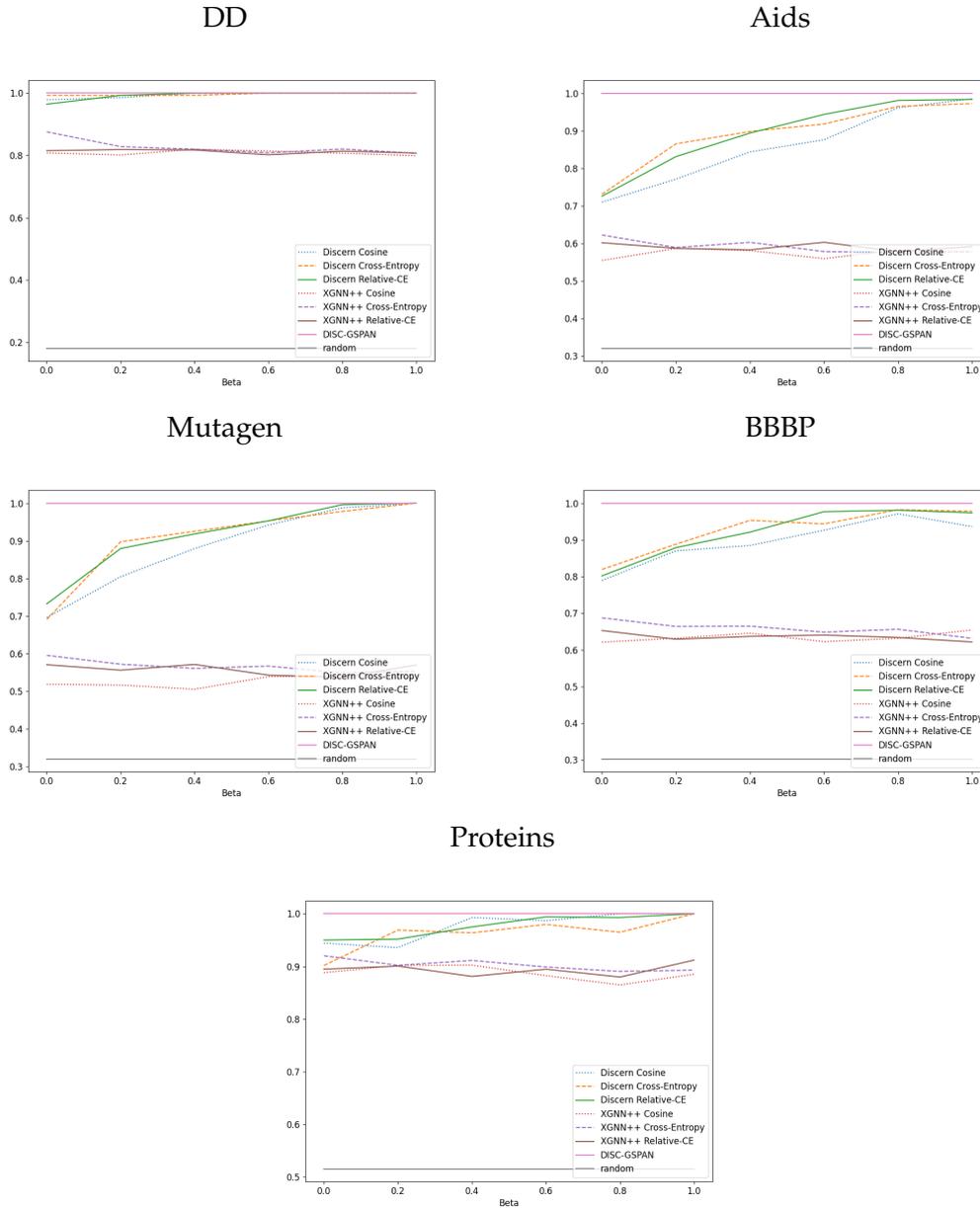


Figure 4.4: Normalized maximum common subgraph (MCSG) between DISCERN and generated graphs (y-axis) with respect to β for all datasets except BA2. The closer to 1, the more realistic the graph.

4.4.5 Examples

We report in Figure 4.5 the best graphs provided by each method for two activation rules on Mutagen. These rules are highly correlated to the decision “Mutagenicity”. For the first rule (top row), DISC-GSPAN and DISCERN identify parts of toxycophores (Bay-region, K-region) (Kazius et al., 2005). XGNN++ provides either unrealistic (i.e., Cosine) or too

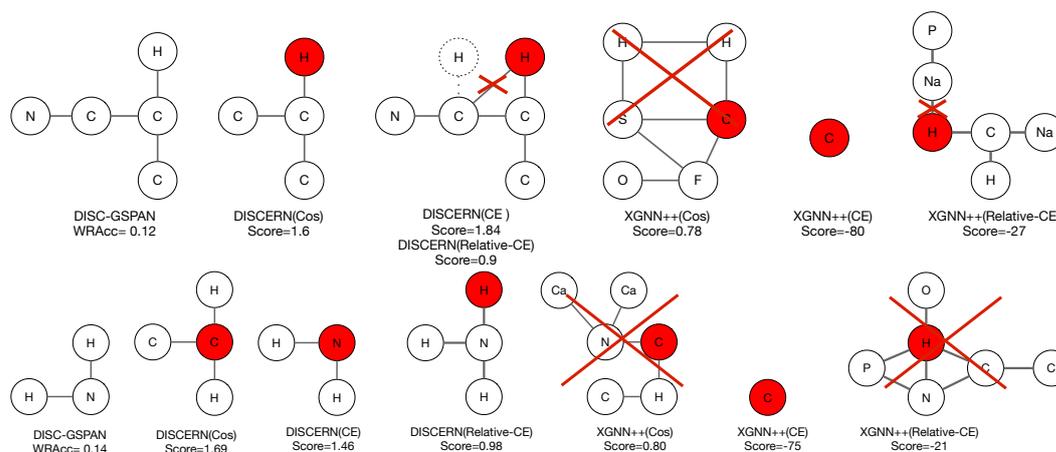


Figure 4.5: Graphs generated by each method for two activation rules (rows) that are highly correlated to mutagenicity. Red cross highlights unrealistic bonds or molecules. Red nodes are those that activate the rule.

general graphs (i.e., only one carbon). Note that the graph generated by DISCERN for both Cross-Entropy and Relative-CE is not entirely realistic since a hydrogen atom cannot have two bonds. Nevertheless, duplicating this atom and binding it to another carbon (dashed node and edge) leads to a similar representation which is realistic.

For the second activation rule (bottom), both DISC-GSPAN and DISCERN_{Cross-Entropy} depict a part of amine group (NH_2) while DISCERN_{Relative-CE} outputs Ammonia. These molecules are known to be toxicophore. DISCERN_{Cosine} generates a Vinylidene group also known to be toxic. It is interesting to note that the center nodes (red filled nodes) do not depict the same atom. Once again, graphs generated by XGNN++ are either unrealistic or too general (carbon atom). Graphs generated by Random are not shown as they are too unrealistic as shown in Figure 4.4.

4.4.6 Discussion

We state here the main conclusion we can draw from these experiments. Our method DISCERN makes it possible to generate representative graphs considering three metrics (i.e., Cross-Entropy, Cosine and Relative-CE). Experiments give evidence that these three metrics are well optimized through the MCTS-based generation. The quality of the representative graphs is statistically significant as reported in Table 4.2. Results suggest that Cross-Entropy is slightly better than two others but this is not significant. Experiments demonstrate that interest of activation rules. Building mask directly from activation rules allows us outperforming the state of the art instance-level explainers. Nevertheless, such rules are not interpretable. This motivates DISCERN whose instance explanation performance is comparable to competitors. The second series of experiments demonstrates that DISCERN outperforms the baselines by providing more realistic graphs. Nevertheless, we do not have theoretical guarantee of generating fully realistic graphs and DISCERN can generate graph with unrealistic configuration as shown in Figure 4.5.

4.5 Conclusion

We have tackled the problem of explaining GNNs with an original angle of attack. Instead of just assessing the importance of some input graph feature to the model decision, our goal is to study the GNN internal representation, i.e., to identify and highlight the features the GNN built through its different layers. To this end, we have introduced a novel method for explaining internal representations of GNNs. Given some activation rules that define internal representations having a strong impact on the classification process, DISCERN generates, with a MCTS approach, realistic graphs that fully embed in the related subspace identified by the rules. Our method relies on a proximity measure between a graph and an activation rule to assess how the generated graph embeds in the subspace defined by the activation rule. There are different ways to construct such a measure and we have proposed three different ones. We have reported an extensive empirical study on six real-world datasets. We have obtained comprehensive results proving that the activation rules allow identifying relevant representation spaces built by the GNN. Masks directly built from the activation rules allow obtaining an instance-level model explanation method that outperforms four state of the art methods while explanations directly based on the graph generated by DISCERN achieve performance comparable to state of the art methods. In terms of combinatorics we only need a subset of ego graphs to support any single instance explanation, while other methods need the full expressivity of masks. With the added benefit of having coherent explanations through the dataset, for instance GNN explainer provide independent explanations for each instance. We have provided further evidence that DISCERN characterizes well each rule with realistic graphs. This makes it possible to capture interesting insights about how the internal representations are built by the GNN.

We believe that such method can support knowledge discovery from powerful GNNs and provide insights on object of study for scientists. Finally, a number of potential limitations need to be considered for future research to make this knowledge discovery from GNNs effective in practice. First, due to its intrinsic nature, the generated graphs pay attention to the graph structure. As a consequence, some findings may be over-specified especially in the case where the GNN builds only-content-related features. To overcome this limitation, we need to provide some additional assessment and/or to introduce a “wild-card label”. Second, the relations between layers are not taken into account in the discovery of activation rules. This may lead to redundant results when considering all the layers. To avoid such redundancy, it is necessary to take into account as prior knowledge the previous layers of a given layer.

The present study has only investigated the discovery of activation rules and then their characterization based on a representative generated graph. We believe that this allows to identify hidden features built by the GNN. However, the current study does not take into account how these features are combined to lead to the model decision. A next step toward interpretability is the investigation of how the model combine these features.

Chapter 5

Conclusion and Future Directions

We first summarize the contributions developed in this manuscript before discussing their perspectives.

5.1 Conclusion

In this thesis, we have tackled the problem of explaining GNNs with an original angle of attack. Instead of just assessing the importance of some input graph feature to the model decision, we have endeavored to study the GNN internal representation, i.e., to identify and highlight the features the GNN builds through its different layers.

To this end, we have introduced a novel method rooted in subgroup discovery for the explainability of GNNs in Chapter 3. The so-called `INSIDE-GNN` is based on the discovery of relevant activation rules in each hidden layer of the GNN. Activation rules aim at depicting set of components that are commonly activated together for a given model decision. The interestingness of the rules is defined through the subjective interestingness framework: prior beliefs are used to assess how contrastive a rule is. We have devised an algorithm that efficiently and iteratively builds a set of activation rules. At each iteration, updating the model with the chosen rule allows to discard candidates rules that are too similar to those in the set. Therefore, this allows to reduce the redundancy of the set of activation rules. An experimental study confirms that activation rules capture interesting insights about how the internal representations are built by the GNN. It is possible to design an instance-level explainer based on these rules, which outperforms the state-of-the-art methods when considering fidelity metric. The main limitation of the proposed method is that activation rules are not interpretable themselves. They require the consideration of ad-hoc advanced treatments to depict them in an human interpretable way.

In Chapter 4, we have tackled the previous limitation. We have introduced a novel method for explaining internal representations of GNNs. Given some activation rules that define internal representations having a strong impact on the classification process, we have introduced a novel method called `DISCERN` that generates representative graph for each rule. With a MCTS approach, `DISCERN` generated realistic graphs that fully embed in the related subspace identified by the rules. Our method relies on a proximity measure between a graph and an activation rule to assess how the generated graph embeds in the subspace defined by the activation rule. There are different ways to construct such a measure and we

have proposed three different ones. Through an extensive empirical study on six real-world datasets, we have made the following observations. We have obtained comprehensive results proving that the activation rules allow identifying relevant representation spaces built by the GNN. Explanations directly based on the generated graph achieves performance comparable to state of the art methods. We have provided further evidence that DISCERN characterizes well each rule with realistic graphs. This makes it possible to capture interesting insights about how the internal representations are built by the GNN.

5.2 Perspectives

The modular aspect of this work makes it possible to improve each part independently. First of all, the computation of the activation rules is a slow process especially because each layer is considered independently, whereas the activation rules of later layers are correlated to the previous ones. This information could be used to speed up the search processes and increase the quality of the generated subgraphs, by adding multiple rules in the objective of the MCTS. Such improvements will require the definition of a metalanguage on the rules with conjunction, disjunction and negations of components and an algorithm to extract them. The FORSIED framework allows the production of high quality rules, but with a significant computational cost which could be reduced by using another algorithm and add different heuristics.

Regarding the association of a graph with an activation rule, the use of MCTS is mainly motivated by its stability and ability to give successful result much more important than the one of XGNN-like approaches. First, we can think of using Generative Adversarial Networks Degardin et al. (2021), Goodfellow et al. (2014) with an encoder that builds an ego network that triggers a rule, and a discriminator that learns to differentiate generated ego networks from real ego networks that activate the rule. Variational Auto-Encoders Bengio (2009), Kingma and Welling (2014), Kipf and Welling (2016), Vercheval et al. (2020) could also be used to reconstruct ego networks from embedding. Several works have already started to work on such Neural Network architecture with GNN. If such methods are successful the main benefit they provide over an MCTS is that they do not need a cumbersome realism check. One can also simply improve the MCTS: the search space is a directed acyclic graph whose order can be modified when certain edges are added. The MCTS algorithm can be adapted with the upper confidence limit for DAG Saffidine et al. (2012) instead of UCT. This approach requires graph isomorphism checking and is not compatible with the layered construction of ego networks. On the other hand, we could follow the decomposition of the Weisfeiler-Lehman tree which could build more mathematically understandable results because it would follow the computational flow of the GNN and build an entire equivalence class instead of a single graph. It would allow easier canonical ordering among graphs, but at the expense of realism checking. As we observe an equivalence class, most realism checks are not feasible, and reconstructing graphs from a Weisfeiler-Lehman-Tree is NP-hard. Such structural changes represent different compromises that are not worth it as they are. The second area of improvement comes from objective functions, that is to say realism and the rule activation objectives.

The class of models considered so far is limited: no experimentation has been done to support node classification explanations. The method could work perfectly fine as the

algorithm is defined, however there is one conceptual problem . Let us consider a rule on the first layer. The structure responsible for this activation rule should not mean the same if it is at distance 2 or 3 from the node we want to predict. The first adaptation is to separate the activation matrices into layer and respective distance from the central node, multiplying the computation time of the activation rules. A modification of the rule extraction algorithm could consider this information. Edge prediction and regression are not tested either and the addition of node attribute, especially numerical ones, requires some modification in the MCTS search as it would have to guess the numerical value of the node when it is added. The General approach is general to GNN with local message passing but not k-hop message passing without significant improvements. Deeper GNN, even if they are not common, are problematic as the rule extraction would take longer time as well as the MCTS search.

INSIDE-GNN and DISCERN extracts a set of 60 rules with associated subgraphs that is able to explain a GNN model in general. Our explainer is able to decompose the model into elementary bricks. With the hypothesis that the original model has good performance, the explanations can be used to extract knowledge on the task itself rather than the model, and even build science from it. In several tasks human can classify the data but scientists do not fully understand a natural phenomenon. GNN learning has its own limitation: the data structure of the graph which is a simplification of the real phenomenon and seen by an algorithm equivalent to WL. As a user, we need to be careful not to add bias to our conclusions, but the layered breakdown of each step should help in this regard. Apart from the extraction of scientific knowledge and if we move away from GNNs strictly, the explainer can be used in the extraction of multi-agent strategy (reinforcement learning). Depending on the configuration, several agents following the same strategy and communicating can define a graph, and if we record either an internal state of each agent, or the explicit communications, literally message passing, as a hidden feature vector, the whole system can be considered equivalent to a GNN. The explanations on such a system is a local strategy: "as a node v , if my neighbors are in the state x and y and the local structure is z I should act as D ". On an existing multi-agent setup, it would probably be possible to get some interesting results, however, if we train a GNN as multi-agent reinforcement learning, a reinforcement learning environment extracts a strategy for an application of the real world such as the IoT, where more explicit strategies than a neural network are required for security reasons for instance.

Bibliography

- M. Ancona, E. Ceolini, C. Öztireli, and M. Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.
- R. Andrews, J. Diederich, and A. B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowl. Based Syst.*, 8(6):373–389, 1995.
- L. Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In D. Wicks and Y. Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016.
- S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS one*, 10(7):e0130140, 2015.
- M. Balcilar, G. Renton, P. Héroux, B. Gaüzère, S. Adam, and P. Honeine. Analyzing the expressive power of graph neural networks in a spectral perspective. In *International Conference on Learning Representations*, 2021.
- F. Baldassarre and H. Azizpour. Explainability techniques for graph convolutional networks. *CoRR*, abs/1905.13686, 2019.
- Y. Bengio. Learning deep architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127, 2009.
- H. Blockeel, T. Calders, É. Fromont, B. Goethals, A. Prado, and C. Robardet. An inductive database system based on virtual mining views. *Data Min. Knowl. Discov.*, 24(1):247–287, 2012.
- K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H. Kriegel. Protein function prediction via graph kernels. In *Proceedings Thirteenth International Conference on Intelligent Systems for Molecular Biology 2005, Detroit, MI, USA, 25-29 June 2005*, pages 47–56, 2005.
- K. M. Borgwardt, X. Yan, M. Thoma, H. Cheng, A. Gretton, L. Song, A. Smola, J. Han, P. Hu, and H.-P. Kriegel. Combining near-optimal feature selection with gSpan. In *6th International Workshop on Mining and Learning with Graphs (MLG 2008)*, pages 1–3, 2008.
- K. M. Borgwardt, M. E. Ghisu, F. Llinares-López, L. O’Bray, and B. Rieck. Graph kernels: State-of-the-art and future challenges. *Found. Trends Mach. Learn.*, 13(5-6), 2020.

- S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comp. net. and ISDN systems*, 30(1-7):107–117, 1998.
- N. Burkart and M. F. Huber. A Survey on the Explainability of Supervised Machine Learning. *Journal of Artificial Intelligence Research*, 70:245–317, 2021.
- T. Calders, C. Rigotti, and J.-F. Boulicaut. A survey on condensed representations for frequent sets. In J.-F. Boulicaut, L. D. Raedt, and H. Mannila, editors, *Constraint-based mining and Inductive Databases : European Workshop on Inductive Databases and Constraint Based Mining, Hinterzarten, Germany, March 11-13, 2004, Revised Selected Papers*, pages 64–80. Springer, Feb. 2006.
- O. Camburu, E. Giunchiglia, J. N. Foerster, T. Lukasiewicz, and P. Blunsom. Can I trust the explainer? verifying post-hoc explanatory methods. *CoRR*, abs/1910.02065, 2019.
- L. Cerf, J. Besson, C. Robardet, and J. Boulicaut. Closed patterns meet n -ary relations. *ACM Trans. Knowl. Discov. Data*, 3(1):3:1–3:36, 2009.
- R. Charbey and C. Prieur. Graphlet-based characterization of many ego networks. working paper or preprint, Apr. 2018. URL <https://hal.archives-ouvertes.fr/hal-01764253>.
- E. Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50, Dec. 1991.
- C. Chen, F. Tung, N. Vedula, and G. Mori. Constraint-aware deep neural network compression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 400–415, 2018a.
- C. Chen, O. Li, D. Tao, A. Barnett, C. Rudin, and J. Su. This looks like that: Deep learning for interpretable image recognition. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8928–8939, 2019.
- J. Chen, L. Song, M. J. Wainwright, and M. I. Jordan. Learning to Explain: An Information-Theoretic Perspective on Model Interpretation. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, {ICML} 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 882–891. PMLR, 2018b.
- M. W. Craven and J. W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In W. W. Cohen and H. Hirsh, editors, *Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994*, pages 37–45. Morgan Kaufmann, 1994.
- S. Dandl, C. Molnar, M. Binder, and B. Bischl. Multi-objective counterfactual explanations. In T. Bäck, M. Preuss, A. H. Deutz, H. Wang, C. Doerr, M. T. M. Emmerich, and H. Trautmann, editors, *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I*, volume 12269 of *Lecture Notes in Computer Science*, pages 448–469. Springer, 2020.

- T. De Bie. Finding interesting itemsets using a probabilistic model for binary databases. Technical report, University of Bristol, 2009.
- T. De Bie. An information theoretic framework for data mining. In C. Apté, J. Ghosh, and P. Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 564–572. ACM, 2011.
- G. de Melo. Inducing conceptual embedding spaces from wikipedia. In *Proceedings of the 26th International Conference on World Wide Web Companion, WWW '17 Companion*, page 43–50, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.
- A. Debnath, R. Lopez de Compadre, G. Debnath, A. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, February 1991.
- B. Degardin, J. C. Neves, V. Lopes, J. Brito, E. Yaghoubi, and H. Proença. Generative adversarial graph convolutional networks for human action synthesis. *CoRR*, abs/2110.11191, 2021.
- M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050, 2005.
- P. D. Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.
- A. Duval and F. D. Malliaros. Graphsvx: Shapley value explanations for graph neural networks. In N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, and J. A. Lozano, editors, *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part II*, volume 12976 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2021.
- F. Errica, M. Podda, D. Bacciu, and A. Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.
- A. A. Freitas. Comprehensible classification models: a position paper. *SIGKDD Explor.*, 15(1):1–10, 2013.
- N. Frosst and G. E. Hinton. Distilling a neural network into a soft decision tree. In T. R. Besold and O. Kutz, editors, *Proceedings of the First International Workshop on Comprehensibility and Explanation in AI and ML 2017 co-located with 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017), Bari, Italy, November 16th and 17th, 2017*, volume 2071 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.
- T. Funke, M. Khosla, and A. Anand. Hard masking for explaining graph neural networks, 2021. URL <https://openreview.net/forum?id=uDN8pRAdsoC>.

- J. Fürnkranz, T. Kliegr, and H. Paulheim. On cognitive preferences and the plausibility of rule-based models. *Mach. Learn.*, 109(4):853–898, 2020.
- C. Gallicchio and A. Micheli. Graph echo state networks. In *International Joint Conference on Neural Networks, IJCNN 2010, Barcelona, Spain, 18-23 July, 2010*, pages 1–8. IEEE, 2010.
- J. A. Gámez, J. L. Mateo, and J. M. Puerta. Learning bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood. *Data Min. Knowl. Discov.*, 22(1-2):106–148, 2011.
- X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.
- I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.
- W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive Representation Learning on Large Graphs. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, {USA}*, pages 1024–1034, 2017.
- G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.
- T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The annals of statistics*, 36(3):1171–1220, 2008.
- Q. Huang, M. Yamada, Y. Tian, D. Singh, D. Yin, and Y. Chang. GraphLIME: Local interpretable model explanations for graph neural networks. *CoRR*, abs/2001.06216, 2020.
- T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Commun. ACM*, 39(11):58–64, nov 1996.
- E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.
- M. I. Jordan. Graphical models. *Statistical science*, 19(1):140–155, 2004.
- J. Kazius, R. McGuire, and R. Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of medicinal chemistry*, 48(1):312–320, 2005.
- D. P. Kingma and M. Welling. Auto-encoding variational bayes. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.
- T. N. Kipf and M. Welling. Variational graph auto-encoders. *CoRR*, abs/1611.07308, 2016.

- T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.
- S. Kumar, F. Spezzano, V. S. brahmanian, and C. Faloutsos. Edge weight prediction in weighted signed networks. In F. Bonchi, J. Domingo-Ferrer, R. Baeza-Yates, Z. Zhou, and X. Wu, editors, *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 221–230. IEEE Computer Society, 2016.
- N. Lavrač, P. Flach, and B. Zupan. Rule evaluation measures: A unifying view. In *International Conference on Inductive Logic Programming*, pages 174–185. Springer, 1999.
- N. Lavrac, B. Kavsek, P. A. Flach, and L. Todorovski. Subgroup discovery with CN2-SD. *J. Mach. Learn. Res.*, 5:153–188, 2004.
- Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- F. Lemmerich and M. Becker. pysubgroup: Easy-to-use subgroup discovery in python. In U. Brefeld, E. Curry, E. Daly, B. MacNamee, A. Marascu, F. Pinelli, M. Berlingerio, and N. Hurley, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2018, Dublin, Ireland, September 10-14, 2018, Proceedings, Part III*, volume 11053 of *Lecture Notes in Computer Science*, pages 658–662. Springer, 2018.
- G. Liu and L. Wong. Effective pruning techniques for mining quasi-cliques. In *ECML/PKDD*, pages 33–49, 2008.
- S. M. Lundberg and S. I. Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, {USA}*, pages 4765–4774, 2017.
- D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang. Parameterized Explainer for Graph Neural Network. In *NeurIPS 2020*, 2020.
- C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.
- P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Extensions of marginalized graph kernels. In *Proceedings of the twenty-first international conference on Machine learning*, page 70, 2004.
- D. Margaritis and S. Thrun. Bayesian network induction via local neighborhoods. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 505–511. The MIT Press, 1999.

- Y. Ming, P. Xu, H. Qu, and L. Ren. Interpretable and steerable sequence learning via prototypes. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 903–913. ACM, 2019.
- C. Molnar. *Interpretable machine learning*. Lulu. com, 2020.
- C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *CoRR*, abs/2007.08663, 2020.
- R. K. Mothilal, A. Sharma, and C. Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In M. Hildebrandt, C. Castillo, L. E. Celis, S. Ruggieri, L. Taylor, and G. Zanfir-Fortuna, editors, *FAT* '20: Conference on Fairness, Accountability, and Transparency, Barcelona, Spain, January 27-30, 2020*, pages 607–617. ACM, 2020.
- W. Pan, H. Dong, and Y. Guo. Dropneuron: Simplifying the structure of deep neural networks. *arXiv preprint arXiv:1606.07326*, 2016.
- M. M. Pasandi, M. Hajabdollahi, N. Karimi, and S. Samavi. Modeling of pruning techniques for deep neural networks simplification. *arXiv preprint arXiv:2001.04062*, 2020.
- P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann. Explainability methods for GCN. In *IEEE CVPR 2019*, pages 10772–10781, 2019.
- A. Prado, M. Plantevit, C. Robardet, and J. Boulicaut. Mining graph topological patterns: Finding covariations among vertex descriptors. *IEEE Trans. Knowl. Data Eng.*, 25(9):2090–2104, 2013.
- M. T. Ribeiro, S. Singh, and C. Guestrin. “Why Should {I} Trust You?”: Explaining the Predictions of Any Classifier. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd {ACM} {SIGKDD} International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144. ACM, 2016.
- M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-precision model-agnostic explanations. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1527–1535. AAAI Press, 2018.
- K. Riesen and H. Bunke. IAM graph database repository for graph based pattern recognition and machine learning. In N. da Vitoria Lobo, T. Kasparis, F. Roli, J. T. Kwok, M. Georgiopoulos, G. C. Anagnostopoulos, and M. Loog, editors, *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshop, SSPR & SPR 2008, Orlando, USA, December 4-6, 2008. Proceedings*, volume 5342 of *Lecture Notes in Computer Science*, pages 287–297. Springer, 2008.
- A. Saffidine, T. Cazenave, and J. Méhat. UCD : Upper confidence bound for rooted directed acyclic graphs. *Knowl. Based Syst.*, 34:26–33, 2012.

- F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.
- M. S. Schlichtkrull, N. D. Cao, and I. Titov. Interpreting graph neural networks for NLP with differentiable edge masking. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.
- T. Schnake, O. Eberle, J. Lederer, S. Nakajima, K. T. Schütt, K.-R. Müller, and G. Montavon. {XAI} for Graphs. *CoRR*, abs/2006.0, 2020.
- R. Schwarzenberg, M. Hübner, D. Harbecke, C. Alt, and L. Hennig. Layerwise relevance visualization in convolutional text graph classifiers. In D. Ustalov, S. Somasundaran, P. Jansen, G. Glavas, M. Riedl, M. Surdeanu, and M. Vazirgiannis, editors, *Proceedings of the Thirteenth Worksp on Graph-Based Metds for Natural Language Processing, TextGraphs@EMNLP 2019, ng Kong, November 4, 2019*, pages 58–62. Association for Computational Linguistics, 2019.
- R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 618–626. IEEE Computer Society, 2017.
- L. S. Shapley. 17. a value for n-person games. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games (AM-28), Volume II*, pages 307–318. Princeton University Press, 1953.
- N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada*, pages 1660–1668. Curran Associates, Inc., 2009.
- N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.
- A. Shrikumar, P. Greenside, and A. Kundaje. Learning Important Features Through Propagating Activation Differences. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, {ICML} 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 2017.
- K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014.

- E. Strumbelj and I. Kononenko. An efficient explanation of individual classifications using game theory. *J. Mach. Learn. Res.*, 11:1–18, 2010.
- C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.
- L. Tschora, E. Pierre, M. Plantevit, and C. Robardet. Electricity price forecasting on the day-ahead market using machine learning. *Applied Energy*, 313:118752, May 2022.
- A. Tversky and D. Kahneman. Extensional versus intuitive reasoning: The conjunction fallacy in probability judgment. *Reasoning: Studies of human inference and its foundations*, pages 114–135, 2008.
- P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.
- N. Vercheval, H. D. Bie, and A. Pizurica. Variational auto-encoders without graph coarsening for fine mesh learning. In *IEEE International Conference on Image Processing, ICIP 2020, Abu Dhabi, United Arab Emirates, October 25-28, 2020*, pages 2681–2685. IEEE, 2020.
- L. Veyrin-Forrer, A. Kamal, S. Duffner, M. Plantevit, and C. Robardet. Qu’est-ce que mon GNN capture vraiment ? exploration des représentations internes d’un GNN. In S. Amer-Yahia and A. Soulet, editors, *Extraction et Gestion des Connaissances, EGC 2022, Blois, France, 24 au 28 janvier 2022*, volume E-38 of RNTI, pages 159–170. Editions RNTI, 2022a.
- L. Veyrin-Forrer, A. Kamal, S. Duffner, M. Plantevit, and C. Robardet. What does my GNN really capture? on exploring internal GNN representations. In L. D. Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 747–752. ijcai.org, 2022b.
- L. Veyrin-Forrer, A. Kamal, S. Duffner, M. Plantevit, and C. Robardet. In pursuit of the hidden features of gnn’s internal representations. *Data Knowl. Eng.*, 142:102097, 2022c.
- L. Veyrin-Forrer, A. Kamal, S. Duffner, M. Plantevit, and C. Robardet. On GNN explainability with activation patterns. *Data Min. Knowl. Discov.*, 2022.
- S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.
- M. N. Vu and M. T. Thai. PGM-Explainer: Probabilistic Graphical Model Explanations for Graph Neural Networks. In *NeurIPS 2020*, 2020.
- X. Wang and A. Vinel. Benchmarking graph neural networks on link prediction. *CoRR*, abs/2102.12557, 2021.
- X. Wang, Y. Wu, A. Zhang, X. He, and T. seng Chua. Causal screening to interpret graph neural networks, 2021. URL <https://openreview.net/forum?id=nzKv5vxZfge>.

- B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968.
- Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. S. Pande. Moleculenet. *CoRR*, abs/1703.00564, 2017.
- F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence*, 2(2):109–127, 2021.
- K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How Powerful are {GNN}? In *ICLR*, 2019.
- Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong. Deep neural network compression with single and multiple level quantization. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4335–4342. AAAI Press, 2018.
- X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 721–724. IEEE Computer Society, 2002.
- P. Yanardag and S. V. N. Vishwanathan. Deep graph kernels. In L. Cao, C. Zhang, T. Joachims, G. I. Webb, D. D. Margineantu, and G. Williams, editors, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1365–1374. ACM, 2015.
- Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 4805–4815, 2018.
- Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec. GNNExplainer: Generating Explanations for Graph Neural Networks. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. D’Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 9240–9251, 2019.
- H. Yuan, J. Tang, X. Hu, and S. Ji. XGNN: towards model-level explanations of graph neural networks. In R. Gupta, Y. Liu, J. Tang, and B. A. Prakash, editors, *KDD ’20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 430–438. ACM, 2020a.
- H. Yuan, H. Yu, S. Gui, and S. Ji. Explainability in graph neural networks: A taxonomic survey. *CoRR*, abs/2012.15445, 2020b.
- H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji. On explainability of graph neural networks via subgraph explorations. In M. Meila and T. Zhang, editors, *Proceedings of the 38th*

- International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, volume 139 of Proceedings of Machine Learning Research, pages 12241–12252. PMLR, 2021.*
- J. Zhang, Z. L. Lin, J. Brandt, X. Shen, and S. Sclaroff. Top-down neural attention by excitation backprop. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 543–559. Springer, 2016.
- M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4438–4445. AAAI Press, 2018.
- Z. Zhang, Q. Liu, H. Wang, C. Lu, and C. Lee. ProtGNN: Towards self-explaining graph neural networks. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelfth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 9127–9135. AAAI Press, 2022.
- B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2921–2929. IEEE Computer Society, 2016.
- M. Zitnik, M. Agrawal, and J. Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinform.*, 34(13):i457–i466, 2018.