Qu'est-ce que mon GNN capture vraiment ? Exploration des représentations internes d'un GNN

Luca Veyrin-Forrer*, Ataollah Kamal*, Stefan Duffner*, Marc Plantevit**, Céline Robardet*

*Université de Lyon, INSA Lyon, CNRS, LIRIS UMR5205
**EPITA Research and Development Laboratory (LRDE), France

Résumé. Nous considérons l'explication de GNN. Alors que les travaux existants expliquent la décision du modèle en s'appuyant sur la couche de sortie, nous cherchons à analyser les couches cachées pour identifier les attributs construits par le GNN. Nous extrayons d'abord des règles d'activation qui identifient des ensembles de neurones co-activés pour une classe. Ces règles définissent des représentations internes ayant un impact fort sur la classification. Ensuite, nous associons à celles-ci un graphe dont le plongement produit par le GNN est très proche de celui identifié par la règle. Des expériences sur 6 jeux de données et 3 baselines démontrent que notre méthode génère des graphes réalistes de haute qualité.

1 Introduction

Les graphes sont une structure de données puissante très utilisée pour représenter des données relationnelles. L'une de leurs spécificités est que leur structure sous-jacente n'est pas dans un espace Euclidien et n'a pas une structure en forme de grille, caractéristiques facilitant l'utilisation directe de techniques génériques d'apprentissage automatique. En effet, chaque nœud d'un graphe est caractérisé par un label, ses nœuds voisins et récursivement leurs propriétés. Ces informations intrinsèquement discrètes ne peuvent pas être directement utilisées par des méthodes d'apprentissage automatique standard pour prédire une classe associée au graphe ou à un de ses nœuds. Ainsi, les réseaux de neurones pour graphes (GNN) apprennent des vecteurs $\mathbf{h}_v \in \mathbb{R}^K$ représentant chaque nœud v dans un espace métrique afin de permettre la comparaison entre nœuds. Les GNN utilisent une stratégie de propagation de message qui agrège récursivement les informations des nœuds vers leurs voisins afin de produire des représentations vectorielles de l'ego-graphe centrées sur un nœud v – avec un rayon égal à l'indice de récursivité – de telle sorte que la tâche de classification, basée sur ces vecteurs, soit optimisée.

Bien que les GNN aient atteint des performances remarquables dans de nombreuses tâches, un inconvénient majeur est leur manque d'interprétabilité. Les cinq dernières années ont vu un énorme effort de recherche pour définir des techniques d'explication de réseaux de neurones profonds (Burkart et Huber, 2021; Molnar, 2020), en particulier pour les données sous forme d'images ou de textes. Cependant, l'explicabilité des GNN a été bien moins explorée avec seulement deux types d'approches. D'une part, les algorithmes d'explication au niveau de l'instance (Baldassarre et Azizpour, 2019; Pope et al., 2019; Schnake et al., 2020; Duval et Malliaros, 2021; Luo et al., 2020; Ying et al., 2019) visent à apprendre un masque vu

comme une explication de la décision du modèle pour une instance de graphe. Ils obtiennent les meilleures performances pour l'explication d'exemples. Cependant, ces masques peuvent conduire à des explications peu fiables, ou pire, à des interprétations trompeuses pour l'utilisateur. On peut être tenté d'interpréter tous les nœuds ou caractéristiques du masque comme responsables de la prédiction conduisant à des hypothèses erronées. Un exemple d'interprétation trompeuse est lorsqu'une caractéristique de nœud est perçue comme importante pour la prédiction du GNN, alors qu'il n'y a pas de différence entre sa distribution sur et en dehors des exemples du masque. D'autre part, la méthode XGNN (Yuan et al., 2020a) vise à fournir des explications au niveau du modèle en générant un modèle de graphe qui maximise une étiquette de sortie. Or, cela suppose qu'il existe un modèle unique pour chaque classe, ce qui n'est pas le cas en pratique lorsque l'on fait face à des phénomènes complexes. La plupart de ces méthodes interrogent le GNN avec des graphes d'entrée perturbés pour évaluer l'impact de la perturbation sur la décision du GNN. Elles n'étudient pas les mécanismes internes des GNN, en particulier les différents espaces de plongement produits par les convolutions des graphes, alors que nous sommes convaincus que l'étude de ces vecteurs peut apporter de nouvelles informations sur la façon dont les GNN modélisent les données d'entrées.

Dans cet article, nous considérons les GNN pour la classification de graphes et introduisons la méthode DISCERN [1] afin de caractériser les représentations internes du GNN par des graphes. Dans chaque couche cachée, nous sommes en mesure d'identifier des ensembles de neurones activés, discriminants par rapport à la variable de sortie. De telles règles d'activation capturent des caractéristiques cachées des graphes d'entrée. Cependant, ces règles d'activation ne peuvent pas être facilement interprétées de manière compréhensible. Notre objectif est d'expliquer chaque règle d'activation en générant un graphe qui se plonge parfaitement dans le sous-espace identifié par la règle. Générer un tel graphe nécessite de définir une mesure de proximité pour évaluer à quel point le graphe est proche de la règle d'activation. Ensuite, le problème revient à optimiser cette mesure, c'est-à-dire à trouver le graphe qui maximise la mesure de proximité avec une règle d'activation donnée. Nous proposons une méthode basée sur une recherche arborescente de type Monte Carlo (MCTS) pour générer de tels graphes.

Nos principales contributions sont les suivantes. Nous dressons un panorama des travaux connexes dans la section 2. Nous introduisons le problème de caractérisation des représentations internes des GNN dans la section 3 ainsi que notre méthode DISCERN pour générer des graphes réalistes et représentatifs des règles d'activation. DISCERN repose sur une mesure de proximité entre les graphes et les règles d'activation. Il existe différentes façons de construire de telles mesures, nous en proposons trois. Nous présentons une évaluation empirique sur plusieurs jeux de données réels dans la section 4 où nous étudions la capacité de DISCERN à fournir de bonnes explications via des graphes réalistes et comparer les trois métriques introduites. Ces expériences montrent également que DISCERN fournit des explications meilleures et plus réalistes que les méthodes de l'état de l'art.

2 État de l'art

Les GNN sont très performants dans plusieurs tâches telles que la classification de nœuds, de liens ou de graphes (Wu et al., 2020). De nombreuses techniques permettent d'amélio-

1. DISClosing the IntERnal workings of gNns with graphs

rer les performances des modèles comme la convolution de graphes (Kipf et Welling, 2017), les mécanismes d'attention dans les graphes (Velickovic et al., 2018) et le regroupement de graphes (Wang et Ji, 2020). Cependant, l'explicabilité des GNN a été peu étudiée par rapport aux domaines de l'image et du texte où de nombreuses méthodes ont été proposées (Burkart et Huber, 2021; Molnar, 2020). Comme établi par (Yuan et al., 2020b), les méthodes existantes pour l'explication des réseaux de neurones pour la classification des images ne peuvent pas être directement utilisées sur des graphes. Par exemple, les méthodes qui calculent une image abstraite par rétro-propagation du gradient (Simonyan et al., 2014) fournissent des résultats non exploitables lorsqu'elles sont appliquées à des matrices d'adjacence discrètes. Quelques méthodes ont cependant été proposées pour expliquer les GNN très récemment.

Méthodes au niveau de l'instance. Ces méthodes visent à fournir des explications dépendantes des entrées en identifiant les caractéristiques importantes utilisées pour la prédiction. Quatre familles de méthodes existent. Les méthodes basées sur les *gradients* (Baldassarre et Azizpour, 2019; Pope et al., 2019) – directement adaptées des solutions dédiées aux images et aux textes – utilisent les gradients ou les valeurs de carte de caractéristiques cachées pour calculer l'importance des caractéristiques d'entrée. Les *méthodes basées sur les perturbations* (Ying et al., 2019; Luo et al., 2020) visent à apprendre un masque de graphe en étudiant le changement de prédiction lors de la perturbation des graphes d'entrée. Les *méthodes de substitution* (surogate) (Huang et al., 2020; Vu et Thai, 2020) expliquent un graphe d'entrée en échantillonnant son voisinage et en apprenant un modèle interprétable. Les *méthodes basées sur la décomposition* (Pope et al., 2019; Schnake et al., 2020) décomposent le score de prédiction des neurones de la dernière couche cachée, puis rétro-propagent ces scores couche par couche. Enfin, GraphSVX (Duval et Malliaros, 2021) entre dans ces 4 catégories en apprenant un modèle d'explication de substitution sur un ensemble de données perturbé, la prédiction expliquée est décomposée entre les nœuds d'entrée et leur label en fonction de leur contribution respective.

Méthodes au niveau du modèle. La seule méthode de ce type est XGNN (Yuan et al., 2020a). Elle consiste à entraîner un générateur de graphes de façon à maximiser la probabilité de prédire une certaine classe. Ces graphes sont utilisés pour expliquer cette classe. Cependant, la méthode repose sur l'hypothèse forte que chaque classe peut être expliquée par un seul graphe, ce qui est irréaliste lorsqu'on considère des phénomènes complexes.

Limitations et desiderata. La plupart des méthodes visent soit à expliquer la décision d'un GNN, soit à générer un graphe représentatif pour une décision donnée. Nous pensons que se concentrer uniquement sur la décision du modèle ne permet pas de bien comprendre comment le modèle se comporte et construit sa décision. On peut fournir des informations supplémentaires sur le GNN en examinant la sortie du modèle, mais aussi en essayant de caractériser certains sous-espaces de représentation que le modèle a construits dans les différentes couches.

3 Méthodologie

La méthode que nous proposons identifie des groupes de neurones qui fonctionnent ensemble dans le processus de prise de décision et associe un sous-graphe qui déclenche la règle de classification correspondante. Ce problème est formalisé ci-dessous.

3.1 Définition du problème.

On considère un ensemble de graphes $\mathcal{G}$ avec des labels : $G = (V, E, L)$ avec V un ensemble de sommets, E un ensemble d'arêtes dans $V \times V$, et L une correspondance entre les sommets et les labels : $L \subseteq V \times T$, avec T l'ensemble des labels. Un GNN classe chaque graphe de $\mathcal{G}$ en deux catégories $\{c^0, c^1\}$: GNN $\mathcal{G} \rightarrow \{c^0, c^1\}$. Le GNN prend des décisions au niveau de chaque graphe à partir de vecteurs, les neurones, calculés au niveau des nœuds de chaque graphe. Pour chaque nœud, des ego-graphes de rayons croissants sont plongés dans l'espace euclidien de telle sorte que des ego-graphes similaires soient associés à des vecteurs similaires. Plus précisément, on considère les Graph Convolutional Networks (GCN) (Kipf et Welling, 2017) qui calculent les vecteurs $\mathbf{h}_v^\ell$ associés à l'ego-graphe centré sur le sommet v de rayon ℓ, récursivement par la formule suivante : $\mathbf{h}_v^\ell = ReLU\left(\mathbf{W}_\ell \cdot \sum_{w \in \mathcal{N}(v)} \frac{e_{w,v}}{\sqrt{d_v d_w}} \mathbf{h}_v^{\ell-1}\right)$. $e_{v,w}$ est le poids de l'arête entre les nœuds v et w, $\mathcal{N}(v)$ est l'ensemble des nœuds voisins de v incluant v, ReLU est la fonction d'activation linéaire rectifiée, et $\mathbf{W}_\ell$ sont les paramètres appris lors de la phase d'apprentissage du modèle. Nous avons aussi $d_v = \sum_{w \in \mathcal{N}(v)} e_{v,w}$. Enfin, $\mathbf{h}_v^0$ est le vecteur initial pour le nœud v avec l'encodage one-hot de son label de l'ensemble T.

Chaque vecteur est de taille K et ℓ varie de 0 à L (le nombre maximum de couches dans le GNN), deux hyperparamètres du GNN. Une fois le GNN appris, les vecteurs $\mathbf{h}_v^\ell$ capturent les caractéristiques clés des graphes correspondants sur lesquels la décision de classification est apprise. Lorsqu'une des composantes vectorielles est de valeur élevée, elle joue un rôle dans le processus de décision. Plus précisément, les composantes activées des vecteurs (les indices k tels que $(\mathbf{h}_v^\ell)_k > 0$) sont combinés par le réseau de neurones dans un chemin menant à la décision , soit c^0 soit c^1. Pour une couche ℓ donnée, les composantes activées du plongement $\mathbf{h}_v^\ell$ correspondent à la partie de l'ego-graphe centrée en v et de rayon ℓ qui déclenche la décision.

Règles d'activation. Les règles d'activation sont des groupes de composantes vectorielles qui sont pour la plupart activées ensemble et ce dans des graphes ayant la même décision du GNN. On dit qu'une règle $\mathbf{A}^\ell \rightarrow c$, avec $\mathbf{A}^\ell$ un vecteur binaire de taille K et $c \in \{c^0, c^1\}$, est activé pour un graphe $g_i = (V_i, E_i, L_i) \in \mathcal{G}$ s'il existe un nœud v dans V_i dont les composantes dans la couche ℓ qui correspondent aux composantes activées de la règle sont également co-activées, c'est-à-dire ssi $\exists v \in V_i$ tel que si $\mathbf{A}_k^\ell = 1$ alors $(\mathbf{h}_v^\ell)_k > 0$. Les graphes de décision c pour lesquels $\mathbf{A}^\ell$ est activé forment le support de $\mathbf{A}^\ell$: $\mathrm{supp}(\mathbf{A}^\ell \rightarrow \mathrm{c}, \mathcal{G}) = \{\mathrm{g_i} = (\mathrm{V_i}, \mathrm{E_i}, \mathrm{L_i}) \in \mathcal{G} \mid \exists \mathrm{v} \in \mathrm{V_i} \text{ tel que } \forall \mathrm{k} = 1 \ldots \mathrm{K},\ \left(\mathbf{A}_\mathrm{k}^\ell = 1\right) \Rightarrow \left((\mathbf{h}_\mathrm{v}^\ell)_\mathrm{k} > 0\right) \text{ et } \mathrm{GNN}(\mathrm{g_i}) = \mathrm{c}\}$.

Calculer les règles d'activation. Les règles d'activation d'intérêt sont celles pour lesquelles $\mathrm{supp}(\mathbf{A}^\ell \rightarrow \mathrm{c}, \mathcal{G})$ est élevé et $\mathrm{supp}(\mathbf{A}^\ell \rightarrow \bar{\mathrm{c}}, \mathcal{G})$ est bas [2]. Il existe différentes manières de calculer les règles d'activation. Nous pouvons par exemple calculer des ensembles succincts de règles tirant parti du principe de longueur minimale de description (Fischer et al., 2021). Dans Veyrin-Forrer et al. (2021), nous avons proposé d'utiliser le cadre de l'intérêt subjectif (De Bie, 2011) pour les calculer. Dans la suite, nous nous appuyions sur cette méthode.

Caractérisation des règles d'activation avec des sous-graphes. Dans cet article, nous allons plus loin et expliquons les représentations internes de GNN par la présence de sous-

2. $\bar{c}$ est égal à $1 - c$ et correspond à l'autre classe.

graphes spécifiques dans les données d'entrée. Pour cela, nous recherchons un sous-graphe dont le plongement par le GNN dans la couche ℓ est le plus proche possible d'une règle d'activation donnée. Cela nécessite de définir une mesure de proximité entre les vecteurs de plongement et une règle d'activation (voir section 3.2) et de définir une procédure pour déterminer le sous-graphe qui maximise cette mesure (voir section 3.4).

3.2 Les mesures.

Nous proposons trois mesures différentes pour évaluer la proximité entre un ego-graphe de vecteur $\mathbf{h}_v^\ell$ centré sur le nœud v à la couche ℓ, et une règle d'activation $\mathbf{A}^\ell$. Soit la règle d'activation $\mathbf{A}^\ell \rightarrow c$ avec $\mathbf{A}^\ell = \{a_1, \ldots, a_K\}$, $a_i \in \{0, 1\}$. On note $\mathbf{E}_g = \{\epsilon_1, \ldots, \epsilon_K\}$ le plongement de l'ego-graphe tronqué sur l'intervalle $[0, 1]$: $\epsilon_k = \min\Big(\max(0, (\mathbf{h}_v^\ell)_k), 1\Big)$. La troncature évite de fausser nos mesures avec des valeurs extrêmes. La similarité entre $\mathbf{E}_g$ et $\mathbf{A}^\ell$ peut être évaluée par la mesure de similarité cosinus :

$$\text{Cosinus}(\mathbf{E}_g, \mathbf{A}^\ell) = \frac{\mathbf{E}_g \,.\, \mathbf{A}^\ell}{\parallel \mathbf{E}_g \parallel \parallel \mathbf{A}^\ell \parallel} = \frac{\sum a_i \epsilon_i}{\sqrt{\sum a_i^2}\sqrt{\sum \epsilon_i^2}} \tag{1}$$

Elle est égale au cosinus de l'angle entre les deux vecteurs. On peut aussi envisager d'utiliser la mesure d'entropie croisée (qui équivaut à la log-vraisemblance) :

$$\text{Cross-entropy}(\mathbf{E}_g, \mathbf{A}^\ell) = \sum a_i \log(\epsilon_i) \tag{2}$$

Elle augmente avec le nombre de composantes qui ont de grandes valeurs sur le plongement de l'ego-graphe pour les composantes activées par la règle d'activation. Comme nous avons généralement un ensemble de règles d'activation R, nous pouvons aussi rechercher un ego-graphe spécifique à une règle $\mathbf{A}^\ell$, c'est-à-dire qui n'active des composantes en dehors de la règle que si cela ne déclenche pas une autre règle. Nous proposons l'expression suivante pour mesurer cela :

$$\text{Relative-CE}(\mathbf{E}_g, \mathbf{A}^\ell, R) = \text{Cross-entropy}(\mathbf{E}_g, \mathbf{A}^\ell) + \max_{\mathbf{r} \in R} \text{Cross-entropy}(\mathbf{E}_g, \mathbf{r}) \tag{3}$$

3.3 Coefficient de réalisme.

Les graphes peuvent avoir un plongement proche d'une règle d'activation sans être réalistes et être très différents des graphes du jeu de données. Pour éviter cela, nous associons un score de réalisme à chaque graphe. Ce facteur dépend à la fois de la probabilité que deux sommets soient connectés selon leur type, et de la distribution des degrés pour chaque type de sommet.

Soit $P_{i,j}$ la probabilité d'avoir une arête avec des extrémités de type i et j. Soit $P_{deg}(k|i)$ la probabilité pour un nœud de type i d'être de degré k. Comme nous considérons des sous-graphes de graphes du jeu de données, nous ne voulons pas pénaliser les graphes dont le degré est plus petit que prévu. Ainsi, nous proposons d'utiliser la valeur $d_{k|i} = P_{deg}(k|i) + \sum_{x>k} \frac{P_{deg}(x|i)}{2k}$ pour augmenter la valeur avec les probabilités associées aux degrés supérieurs. Nous dérivons une mesure de probabilité avec $D_{k|i} = \frac{d_{k|i}}{\sum_x d_{x|i}}$. Ainsi, le score de réalisme est calculé par $Realism(g = (V, E, L)) = \frac{\sum_{(u,v)\in E} \log\left(P_{L(u),L(v)}\right)}{|E|} + \frac{\sum_{u \in V} \log(D_{d_u|L(u)})}{|V|}$ avec

$L(u)$ le type du nœud u et d_u son degré. Cette valeur de réalisme s'ajoute à l'une des mesures de similarité entre la règle d'activation et le plongement du graphe – l'une des trois mesures précédentes noté m – pour former le score utilisé pour évaluer la qualité d'un graphe : $Score(g, A^\ell, R) = m(\mathbf{E}_g, \mathbf{A}^\ell, R) + \beta \times Realism(g)$ avec β un hyperparamètre dont la valeur est fixée empiriquement (voir section 4).

3.4 Méthode DISCERN.

Nous proposons d'utiliser une recherche de type *Monte Carlo Tree Search* (MCTS) pour trouver un sous-graphe réaliste dont le plongement est similaire à une règle d'activation. L'objectif est de générer un ego-graphe g_t qui maximise $score(\mathbf{E}_t, \mathbf{A}^\ell, R)$. Chaque nœud t de l'arbre de recherche représente un ego-graphe g_t centré en v_t dont le plongement GNN est $\mathbf{E}_t$. Une valeur M_t est également associée à t : c'est la somme des valeurs $score(\mathbf{E}_t, \mathbf{A}^\ell, R)$ évaluées sur les nœuds terminaux du sous-arbre enraciné en t.

Chaque nœud de l'arbre est obtenu en ajoutant une arête au graphe de son nœud parent. Ce processus s'arrête lorsqu'une condition terminale est satisfaite. Dans notre algorithme, nous considérons trois conditions terminales : (1) le diamètre du graphe est supérieur à ℓ, (2) le nombre d'arêtes est supérieur à $min\text{-}edges$, (3) le nombre de sommets est supérieur à $min\text{-}vertices$. L'arbre est partiellement exploré. A partir d'un nœud t, des nœuds terminaux sont générés aléatoirement afin de pouvoir évaluer leur score (fonction de déploiement). Le score obtenu est ensuite rétro-propagé à tous les ancêtres de t dans leur variable M_t. La valeur M_t permet d'estimer la qualité des descendants de t explorés jusqu'à présent. Cette valeur est ensuite utilisée pour régler l'exploitation, des graphes explorés en cours, et l'exploration de nouveaux graphes. Nous utilisons le classique Upper Confidence Bound UCB1 pour guider la sélection du nœud à explorer : $UCB1(t) = \frac{M_t}{n_t} + c\sqrt{\frac{\log(N(t))}{n_t}}$, où c est un nombre constant, M_t est la somme des valeurs $score(t')$ pour tous les nœuds terminaux t' descendants de t qui ont été explorés jusqu'à présent, n_t est le nombre de nœuds terminaux étendus à partir du nœud t au cours des itérations précédentes de l'algorithme, et $N(t)$ est le nombre de fois où le parent de t a été visité jusqu'à présent. Il convient de mentionner que la constante c dans UCB1 joue un rôle important pour établir un équilibre entre l'exploration et l'exploitation. Si c est trop grande, l'algorithme agit comme un algorithme purement aléatoire (c'est-à-dire plus d'exploration) et si c est trop petite, alors le taux d'exploitation est augmenté et l'algorithme peut rester bloqué dans des maxima locaux. Dans nos expériences, c est fixé à 0,5.

L'algorithme 1 commence par construire un arbre composé d'un nœud racine avec autant d'enfants qu'il y a d'étiquettes dans le jeu de données. Les graphes des enfants de la racine sont constitués d'un seul nœud dont l'étiquette est i. DISCERN effectue plusieurs itérations (époques) dont le nombre est un paramètre d'entrée N. Chaque itération commence par l'appel de la méthode `Explore_child` sur le nœud racine. Cette méthode renvoie un ensemble de graphes terminaux parmi lesquels on prend celui maximisant la mesure.

Si le nœud actuel de `Explore_child` (lignes 1–5) n'est pas une feuille, la méthode trouve le meilleur enfant à considérer en premier. C'est celui avec la valeur UCB1 maximum et qui a au moins un enfant à explorer. Si un tel nœud non terminal existe, `Explore_child` est appelé récursivement dessus. Si le nœud courant est une feuille, soit c'est un nœud non encore exploré ($n_t = 0$) soit un graphe qui satisfait les conditions terminales. Dans le premier cas (lignes 7–11), un déploiement (roll-out) est effectué. Ensuite, le nœud est étendu avec les

Algorithm 1 DISCERN

Require: Number of epochs N, m, $term\_cond$, $\mathbf{A}^\ell$, R.
Ensure: Graph g maximizing $score(\mathbf{E}_g, \mathbf{A}^\ell, R)$.
1: **for** i = 1 to #T **do**
2: $t \leftarrow \{\ g_t \leftarrow (\{v\}, \emptyset, L(v) = i), v_t \leftarrow v, n_t \leftarrow 0, M_t \leftarrow 0\ \}$
3: add t as child of root
4: **end for**
5: **for** epoch = 1 to N **do** Explore_child($root, term\_graphs$)
6: **end for**
7: **return** argmax$_{g \in term_graphs} score(\mathbf{E}_g, \mathbf{A}^\ell, R)$

Explore_child($t, term\_graphs$)
1: **if** (t is not leaf) **then** ▷ selection
2: $bestChild \leftarrow$ Find_child(t)
3: **if** ($bestChild \neq$ **None**) **then** Explore_child($bestChild$)
4: **else** Explore_child(t.parent)
5: **end if**
6: **else**
7: **if** $n_t = 0$ **then** ▷ playout and expansion
8: $g \leftarrow$ roll-out(t)
9: **for** each available_action(g_t) a **do** add-child(t, a)
10: **end for**
11: **end if**
12: **if** $term\_cond(t)$ **then**
13: $g \leftarrow g_t, terminal(t) \leftarrow$ **True**, add(*term_graphs*, g)
14: **end if**
15: **if** ($n_t = 0$) **or** term_cond(t) **then** ▷ back-propagation
16: **for** $s \in$ path from root to g **do**
17: $M_s \leftarrow M_s + score(\mathbf{E}_g, \mathbf{A}^\ell, R), n_s \leftarrow n_s + 1$
18: **end for**
19: **end if**
20: **if** not terminal(t) **then** Explore_child($t, term\_graphs$)
21: **end if**
22: **end if**

Find_child(t)
1: $terminal(t) \leftarrow$ **True**, $bestChild \leftarrow$ **None**
2: **for** $s \in children(t)$ **do**
3: **if** (terminal(s) = **False**) **then**
4: $terminal(t) \leftarrow$ **False**
5: **if** $UCB1(bestChild) \leq UCB1(s)$ **then** $bestChild \leftarrow s$
6: **end if**
7: **end if**
8: **end for**
9: **return** $bestChild$

graphes résultant de l'ajout d'une arête de l'ensemble `available_action`. La fonction roll-out consiste à prendre de manière répétée uniformément au hasard une action parmi les `available_actions` jusqu'à ce que les conditions terminales soient remplies. Dans le second cas, le graphe est marqué comme terminal et stocké dans l'ensemble `term_graphs`. Dans les deux cas (lignes 15–19), la mesure évaluée sur le plongement $\mathbf{E}_g$ du nœud terminal g est rétro-propagée au nœud racine. Ensuite, `Explore_child` est appelé récursivement.

Actions disponibles. Les actions possibles pour étendre un ego-graphe non terminal g_t sont les différentes arêtes qu'il est possible d'ajouter au graphe pour qu'il reste connecté. Il existe deux types d'arêtes : (1) celles avec une extrémité dans V_t, l'autre extrémité étant un nouveau nœud ($v \notin V_t$) étiqueté avec une étiquette de T, ou (2) celles dont les deux extrémités sont dans V_t. Ainsi, il y a $\#V_t \times (\#V_t + \#T)$ actions possibles. Pour améliorer la recherche, nous avons réduit cet ensemble d'actions disponibles. Le but est de réduire la largeur de l'arbre, d'éviter de générer des graphes isomorphes, tout en obtenant des valeurs plus homogènes de la mesure $score$ sur les graphes du sous-arbre. Notons $dist(v, u)$ la distance

géodésique entre les nœuds u et v (la longueur du plus court chemin). On considère la valeur $k(u,w) = \min(dist(u,v_t), dist(w,v_t)) + 1$ qui correspond au nombre de convolutions nécessaires pour que les vecteurs de u et w modifient le plongement du centre de l'ego-graphe v_t. On permet l'ajout d'une arête $\{u,w\}$, avec u et w deux sommets du graphe, si cela ne diminue pas la valeur k des arêtes du graphe. Ainsi, $\{u,w\}$ peut être ajouté si $dist(v_t,u) \geq k-1$ et $dist(v_t,w) \geq k-1$. Une arête entre un sommet u du graphe et un nouveau sommet z peut être ajoutée si $dist(v_t,u) \geq k-1$.

4 Expérimentations

Dans cette section, nous évaluons DISCERN à travers plusieurs expériences. Nous décrivons d'abord les jeux de données et le plan expérimental. Ensuite, nous présentons quelques exemples de sous-graphes générés par notre méthode comparés à ceux obtenus par les compétiteurs. Puis, nous présentons une étude quantitative de notre méthode. DISCERN a été implémentée en Python et les expérimentations ont été faites sur une machine équipée de 8 processeurs Intel(R) Xeon(R) W-2125 @ 4,00GHz cœurs 126Go de mémoire principale, exécutant Debian GNU/Linux. Le code et les données sont disponibles [3].

4.1 Jeux de données et compétiteurs.

Les expérimentations sont réalisées sur six jeux de données de classification de graphes dont les principales caractéristiques sont données dans le Tab. 1 (gauche). BA2 (Ying et al., 2019) est un jeu de données synthétique généré avec des graphes de Barabasi-Albert et contenant soit un cycle de talle 5 (classe négative) soit une "maison" (classe positive). Les autres jeux de données (Aids (Morris et al., 2020), BBBP(Wu et al., 2017), Mutagen (Morris et al., 2020), DD (Dobson et Doig, 2003), Proteins (Borgwardt et al., 2005)) correspondent à de vraies molécules, et la classe identifie des propriétés importantes en chimie ou en découverte de médicaments (c'est-à-dire une activité possible contre le VIH, la perméabilité et la mutagénicité). Un GNN à 3 couches convolutives (avec $K = 20$) est appris sur chaque jeu de données. L'accuracy pour chaque jeu de tests est donnée dans Tab. 1. Nous utilisons la méthode INSIDE-GNN pour fouiller les vecteurs d'activation du GNN $\mathbf{h}_v^\ell$ et découvrir les règles d'activation $\mathbf{A}^\ell \rightarrow c$. Nous extrayons au maximum dix règles par couche et pour chaque classe $\{c^0, c^1\}$ comme expliqué dans Veyrin-Forrer et al. (2021). Sur certains jeux de données, moins de 10 règles par couche et par classe sont nécessaires pour décrire le fonctionnement interne de GNN (voir Tab. 1 – gauche). Notre objectif est de fournir un graphe représentatif pour chaque règle avec DISCERN.

Cette étude expérimentale vise à répondre aux questions suivantes : Comment se comporte DISCERN ? Les résultats sont-ils bons ? Les graphes générés sont-ils réalistes ? Quid des compétiteurs ? Nous considérons trois compétiteurs : **Random** génère des graphes aléatoirement. Pour cela, nous appelons 250 fois la fonction Roll-out. **XGNN++** est une extension de XGNN Yuan et al. (2020a) à notre problème. Nous intégrons à la fois les métriques Cosinus et Cross-Entropy en tant que fonction à optimiser dans XGNN. Nous fixons un budget qui correspond à 5000 appels au GNN pour faire une comparaison juste avec DISCERN. **DISC-GSPAN** est une

3. https://www.dropbox.com/sh/sbeimvh8p6vdhps/AAAo66eFt3vTy6M4s4OL2jmxa?dl=0

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

Score	Samp.	Q_1	Q_2	Q_3	Q_{max}
Cosine	supp	10.77	2.42	1.96.	1.82
Cosine	rand	$8.13\ 10^9$	$2.38\ 10^9$	$1.61\ 10^9$	2.96
Cross-Ent.	supp	1.17	1.08	1.07	1.06
Cross-Ent.	rand	$7.98\ 10^{11}$	$1.1\ 10^{11}$	$7.34\ 10^{10}$	2.15
Relative-CE	supp	1.73	1.31	1.28	1.24
Relative-CE	rand	$4.5\ 10^{11}$	4.41	3.25	1.93

TAB. 1 – *Caractéristiques des données (gauche)– Amélioration moyenne du score fourni par* DISCERN *par rapport aux scores des quartiles de deux distributions : (1) les nœuds du support de la règle (supp) et (2) certains nœuds aléatoires (rand) sur Aids.*

méthode correcte et complète qui vise à découvrir des sous-graphes discriminants avec une énumération de type GSPAN (Yan et Han, 2002) tout en exploitant des bornes supérieures sur la mesure WRAcc (Veyrin-Forrer et al., 2021). L'ensemble de données d'entrée contient l'ensemble des ego-graphes de nœuds qui supportent la règle d'activation en tant que « classe positive » et les ego-graphes des nœuds non impliqués dans le support de la règle en tant que classe négative. Ensuite, DISC-GSPAN fournit les top-k sous-graphes qui sont discriminants pour la classe positive correspondant à la règle d'activation. Nous définissons $min\text{-}edges$=10 et $min\text{-}vertices$=6, et nous fixons de manière empirique l'hyper-paramètre β à 1 (voir Fig. 2 (droite)). Notez que nous donnons uniquement les résultats pour Aids lorsque les observations sont similaires pour tous les jeux de données.

4.2 Résultats expérimentaux.

Pour une règle d'activation, DISCERN prend entre 20 et 50 secondes pour 5000 époques. Plus la couche du GNN est profonde, plus l'ego-graphe à étudier est grand, et plus le temps d'exécution est long.

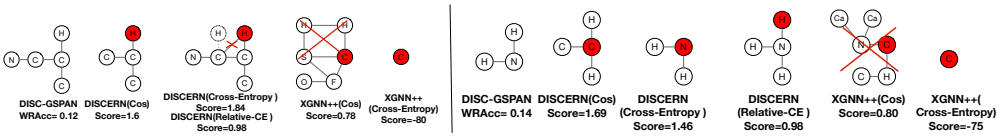

FIG. 1 – *Graphes générés par chaque méthode pour deux règles d'activation (gauche et droite) fortement corrélées à la mutagénicité. La croix rouge met en évidence des liaisons ou des molécules irréalistes. Les nœuds rouges sont ceux qui activent la règle.*

Exemples. La Fig. 1 présente les meilleurs graphes obtenus par chaque méthode pour deux règles d'activation sur Mutagen. Ces règles sont fortement corrélées à la décision « Mutagénicité ». Pour la première règle (gauche), DISC-GSPAN et DISCERN identifient des parties de toxycophores (bay-region, K region) (Kazius et al., 2005). XGNN++ fournit des graphes irréalistes (avec la mesure Cosine) ou trop généraux (i.e., carbone). Notons que le graphe généré par DISCERN pour Cross-Entropy et Relative-CE n'est pas totalement réaliste car un atome d'hydrogène ne peut pas avoir deux liaisons. Néanmoins, dupliquer cet atome et le lier à un autre carbone (nœud et arête en pointillés) conduit à une représentation similaire qui est réaliste.

Pour la deuxième règle d'activation (à droite), DISC-GSPAN et DISCERN$_{Cross-Entropy}$ capturent une partie du groupe amine (NH_2) tandis que DISCERN$_{Relative-CE}$ produit de l'ammoniac. Ces molécules sont connues pour être des toxicophores. DISCERN$_{Cosine}$ génère un groupe vinylidène également connu pour être toxique. Encore une fois, les graphes générés par XGNN++ sont soit irréalistes, soit trop généraux (atome de carbone). Les graphes générés par Random ne sont pas affichés car ils sont trop irréalistes.

Quid de la qualité des résultats ? Afin d'évaluer si les graphes g_{mcts} obtenus par DISCERN sont réellement bons par rapport aux ego-graphes $g_{support}$ qui supportent les règles d'activation, nous considérons un échantillon de nœuds appartenant au support de la règle. De même, nous prenons un échantillon de nœuds aléatoires. Pour les deux échantillons, nous considérons les ego-graphes dont la taille est égale à la couche de la règle d'activation correspondante. Pour chaque échantillon, nous calculons le score pour chaque quartile (Q1, Q2, Q3), les valeurs de score qui divisent l'échantillon ordonné par valeurs en 4 sous-ensembles de tailles égales. On considère également la valeur maximale Q_{max}. Nous rapportons dans Tab. 1 (droite) le ratio des scores des graphes produits par DISCERN par rapport à chaque quartile, soit le ratio $m(E_{g_{MCTS}}, A^{\ell}, R)/Q_x$. Les résultats sont agrégés sur toutes les règles d'activation de tous les jeux de données. Fait intéressant, nous observons que nous obtenons toujours des valeurs supérieures à 1, même lorsque nous comparons les graphes générés à Q_{max} (c'est-à-dire la valeur max de l'échantillon). Cela démontre la haute qualité des graphes générés par DISCERN. Avoir des valeurs supérieures à 1 pour Q_{max} de l'échantillon de nœuds de support signifie que les graphes que nous générons se plongent bien dans le sous-espace cible, avec moins de composants activés en dehors de cet espace que pour les nœuds du support. En effet, les métriques que nous considérons pénalisent les composantes activées de l'ego-graphe qui ne sont pas activées dans les règles d'activation.

	Mutag.	Aids	BBBP	DD	Prot.	BA2	Avg
Random	4.13	4.60	4.37	4.10	5.52	5.56	4.71
DISC-GSPAN	4.17	3.69	3.82	3.66	–	–	–
XGNN++$_{Cos}$	4.57	4.13	4.42	4.06	5.45	5.46	4.68
XGNN++$_{Cross-Ent}$	4.48	4.06	4.25	4.05	5.74	5.36	4.65
DISCERN$_{Cos}$	3.39	**3.12**	**3.08**	**3.23**	**4.56**	4.69	**3.68**
DISCERN$_{Cross-Ent}$	3.68	3.29	3.32	3.31	4.70	**4.58**	3.81
DISCERN$_{Relative-CE}$	**3.32**	3.62	3.35	3.43	4.72	4.63	3.84

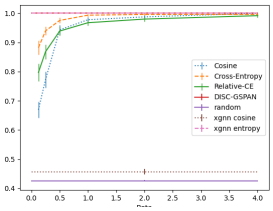

FIG. 2 – *(Gauche) Norme L2 moyenne entre les graphes fournis par chaque méthode et chaque règle d'activation. Plus la valeur est basse, mieux c'est. (Droite) MCSG entre les graphes de DISCERN et les graphes du jeu de données par rapport à β. Plus la valeur est proche de 1, plus le graphe est réaliste.*

Les graphes générés sont-ils réalistes ? Nous étudions l'impact de β pour générer des graphes réalistes. À cette fin, nous évaluons le réalisme des graphes générés par rapport à ceux du jeu de données. Nous calculons le sous-graphe commun maximum (MCSG) entre le graphe produit par DISCERN et chaque graphe de l'ensemble de données. Dans la Fig 2 (droite), nous donnons la taille du MCSG normalisée par la taille du graphe généré lorsque β varie.

Plus la valeur est proche de 1, plus les graphes générés sont réalistes. Nous observons que les graphes générés par DISCERN deviennent plus réalistes lorsque β augmente. Fait intéressant, la métrique Cross-Entropy permet d'obtenir un graphe plus réaliste que les autres métriques pour la même valeur de β. Pour information, le facteur de réalisme est également indiqué pour les trois lignes de base. Par définition, DISC-GSPAN fournit un graphe réaliste car il produit des sous-graphes fréquents. Notez que XGNN++(Cosinus) a également un facteur de réalisme égal à 1 car seuls les graphes avec un seul sommet sont générés. Les deux autres compétiteurs ne parviennent pas à générer des graphes réalistes avec une valeur inférieure à 0,5.

Comment DISCERN se comporte-t-elle par rapport aux compétiteurs ? Les trois compétiteurs optimisent les mesures de cosinus ou d'entropie croisée. Pour chaque règle d'activation, nous étudions la norme L2 entre le meilleur graphe généré par chaque méthode et la règle d'activation. Les résultats sont rapportés dans Fig. 2 (gauche). Notez que DISC-GSPAN échoue pour BA2 et Protéines en raison du petit nombre ou de l'absence de labels, ce qui rend l'extraction impossible. DISCERN fournit des graphes qui se plongent mieux dans l'espace cible que toute autre méthode. En moyenne, DISCERN$_{Cos}$ surpasse la meilleure solution basée sur XGNN d'environ 21%.

5 Conclusion et travaux futurs

Nous avons introduit une nouvelle méthode pour expliquer les représentations internes des GNN. Étant donné des règles d'activation qui définissent des représentations internes ayant un fort impact sur le processus de classification, DISCERN génère des graphes réalistes qui se plongent pleinement dans le sous-espace identifié par les règles. Des résultats empiriques sur six jeux de données avec trois compétiteurs démontrent la qualité des résultats. Dans des travaux futurs, nous souhaitons prendre en compte plusieurs couches simultanément.

Références

Baldassarre, F. et H. Azizpour (2019). Explainability for GCNs. *arXiv :1905.13686*.

Borgwardt, K. M., C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, et H. Kriegel (2005). Protein function prediction via graph kernels. In *Int. Sys. Mol. Bio.*, pp. 47–56.

Burkart, N. et M. F. Huber (2021). A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research 70*, 245–317.

De Bie, T. (2011). An information theoretic framework for data mining. In C. Apté, J. Ghosh, et P. Smyth (Eds.), *SIGKDD 2011*, pp. 564–572. ACM.

Dobson, P. D. et A. J. Doig (2003). Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology 330*(4), 771–783.

Duval, A. et F. D. Malliaros (2021). Graphsvx : Shapley value explanations for graph neural networks. In *ECMLPKDD'21*, pp. 302–318.

Fischer, J., A. Oláh, et J. Vreeken (2021). What's in the box ? exploring the inner life of neural networks with robust rules. In M. Meila et T. Zhang (Eds.), *ICML 2021*, pp. 3352–3362.

Huang, Q., M. Yamada, Y. Tian, D. Singh, D. Yin, et Y. Chang (2020). Graphlime : Local interpretable model explanations for GNNs. *arXiv :2001.06216*.

Kazius, J., R. McGuire, et R. Bursi (2005). Derivation and validation of toxicophores for mutagenicity prediction. *Journal of medicinal chemistry 48*(1), 312–320.

Kipf, T. et M. Welling (2017). Semi-supervised classification with GCN. In *ICLR*.

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

Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

Morris, C., N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, et M. Neumann (2020). Tudataset. *CoRR abs/2007.08663*.

Pope, P. E., S. Kolouri, M. Rostami, C. E. Martin, et H. Hoffmann (2019). Explainability methods for GCN. In *IEEE CVPR 2019*, pp. 10772–10781.

Schnake, T., O. Eberle, J. Lederer, S. Nakajima, K. T. Schütt, K. Müller, et G. Montavon (2020). XAI for graphs. *CoRR abs/2006.03589*.

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

Velickovic, P., G. Cucurull, A. Casanova, A. Romero, P. Liò, et Y. Bengio (2018). Graph attention networks. In *ICLR 2018*.

Veyrin-Forrer, L., A. Kamal, S. Duffner, M. Plantevit, et C. Robardet (2021). On GNN explanability with activation patterns. preprint – https ://hal.archives-ouvertes.fr/hal-03367714.

Vu, M. N. et M. T. Thai (2020). Pgm-explainer : Probabilistic graphical model explanations for graph neural networks. In *NeurIPS 2020*.

Wang, Z. et S. Ji (2020). Second-order pooling for graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Wu, Z., S. Pan, F. Chen, G. Long, C. Zhang, et S. Y. Philip (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*.

Wu, Z., B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, et V. S. Pande (2017). Moleculenet. *CoRR abs/1703.00564*.

Yan, X. et J. Han (2002). gspan : Graph-based substructure pattern mining. In *ICDM 2002*, pp. 721–724. IEEE Computer Society.

Ying, Z., D. Bourgeois, J. You, M. Zitnik, et J. Leskovec (2019). GNNExplainer : Generating explanations for GNNs. In *NeurIPS 2019*, pp. 9240–9251.

Yuan, H., J. Tang, X. Hu, et S. Ji (2020a). XGNN. In *KDD'20*, pp. 430–438.

Yuan, H., H. Yu, S. Gui, et S. Ji (2020b). Explainability in GNN. *arXiv :2012.15445*.

Summary

While existing GNN's explanation methods explain the decision by studying the output layer, we propose a method that analyzes the hidden layers to identify the neurons that are co-activated for a class. We associate to them a graph.