Projet 3A - Département GIMA - Parcours IM

Études des Réseaux de Neuronnes sur Graphes (GNN)

SPEISMANN Matthieu

Ecole Nationale Supérieure des Mines de Nancy

Année scolaire 2025-2026

Table des matières

Table des matières **2**

Introduction **3**
I Intérets de l'étude . . . 3
II Rappels sur les Graphes . . . 3
II-A Quelques notions utiles sur les graphes . . . 4

I Découverte des GNN **5**
I Choisir la bonne représentation pour les graphes . . . 6
II Construction d'un GNN simple . . . 8
II-A Le modèle de base . . . 8
II-B Les opérations de pooling . . . 9
II-C Utilisation de la représentation globale : Le Master Node . . . 11

II Les Graph Convolutional Networks (GCN) **12**
I Les Convolutions Polynomiales . . . 13
I-A Le filtre Laplacien . . . 13
I-B Les polynômes du Laplacien . . . 15
I-C Le ChebNet . . . 17
I-D Utilisation dans un Graph Convolution Network (GCN) . . . 18
II La Convolution Spectrale . . . 19
II-A Défintion de la représentation spectrale . . . 19
II-B La troncature en représentation spectrale . . . 20
II-C Utilisation dans un Graph Convolutional Network (GCN) . . . 21

Bibliographie **22**

Introduction

I Intérets de l'étude

Le but de ce projet est d'étudier, comprendre et utiliser des Graph Neural Networks (GNN). L'objectif des GNN est d'adapter les méthodes de réseaux de neurones pour réaliser des tâches de régression, de classification et/ou de prédiction sur des données structurées sous la forme de graphes.

Ce sujet est particulièrement intéressant car beaucoup de situations concrètes peuvent être représentées avec des données sous forme de graphes :

— Les réseaux sociaux

— Les molécules chimiques

— Les chaines logistiques d'approvisionnement, de production et de livraison.

II Rappels sur les Graphes

Défintion : Graphe

Un graphe G est défini par :

— Un ensemble de nœuds (ou sommets) $V = \{v_1, v_2, \ldots, v_n\}$

— Un ensemble d'arêtes (ou liens) $E \subseteq \{\{u, v\}, u \in E, v \in E\}$

On dira que le graphe est orienté si les arrêtes sont des couples ordonnés (u, v).

On pourra de plus pondérer les noeuds, les arrêtes et/ou le graphe entier en leur associant des valeurs scalaires ou vectorielles.

Cela peut représenter, par exemple :

— **Pour les noeuds** : des caractéristiques propres à chaque noeuds (âge, localisation, etc. dans le cas d'un réseau social)

— **Pour les arrêtes** : des caractéristiques propres aux relations entre deux noeuds (force de la relation, distance géographique, etc.)

— **Pour le graphe entier** : des caractéristiques globales du graphe (type de réseau social, secteur d'activité dans le cas d'une chaine logistique, etc.)

II-A Quelques notions utiles sur les graphes

Pour la suite de ce document, il est important de rappeler quelques notions de base sur les graphes.

Définition : Distance dans un graphe

La distance entre deux noeuds u et v d'un graphe G, notée $\text{dist}_G(u, v)$ est définie comme le nombre minimum d'arrêtes à parcourir pour aller de u à v.

Degré d'un noeud

Le degré d'un noeud v dans un graphe est défini comme le nombre d'arrêtes connectées à ce noeud.

Chapitre I

Découverte des GNN

L'ensemble de ce chapitre se base sur la lecture de l'article *A gentle Introduction to Graph Neural Networks* [1].

Les GNN ont pour objectif de généraliser les tâches de prédictions (classification et régression par exemple) des réseaux de neurones « standards » sur les graphes ou leurs composants. On classera donc les tâches selon qu'elles sont appliquées sur :

- **Les noeuds (node-level tasks)** : prédiction des valeurs d'un noeud, classification des noeuds, etc.
- **Les arrêtes (edge-level tasks)** : prédiction de la présence (ou non) d'une arrête, établir la relation entre les noeuds (classification d'arrêtes), etc.
- **Le graphe entier (global-level tasks)** : le plus souvent une classification du graphe entier.

Dans un réseau de neuronnes standard, les données en entrée n'ont pas de strucure particulière. Chaque données (i.e. la valeur à prédire et ses descripteurs) peuvent être lus et traités indépendament les uns des autres.

La structure des données sous forme de graphe est alors l'élément différentiant par rapport aux approches habituelles et représente donc la principale difficulté pour travailler avec un réseau de neurones. En effet, la connectivité des noeuds entre eux doit être prise en compte pour que le modèle puisse exploiter cette information structurelle, et fournir des résultats satisfaisant exploitants cette particularité.

I Choisir la bonne représentation pour les graphes

Pour représenter les données en graphes de manière satisfaisante pour un réseau de neurones, une solution naïve semble être d'utiliser **la matrice d'adjacence**.

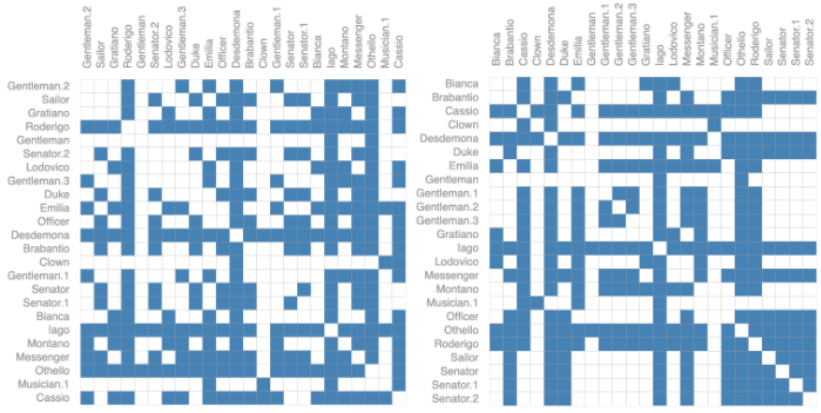
Définition : Matrice d'adjacence

Une matrice d'adjacence A d'un graphe $G = (V, E)$ est une matrice carrée de taille $|V| \times |V|$ telle que : *(|.| correspondant au cardinal)*

$$A_{ij} = \begin{cases} 1 & \text{si } (v_i, v_j) \in E \\ 0 & \text{sinon} \end{cases}$$

Cette représentation intuitive a cela dit deux défauts problématiques dans notre cas :

1. La matrice d'adjacence stocke des valeurs pour l'ensemble des arrêtes possibles, soit, dans un graphe à n noeuds, n^2 valeurs.
   Cependant, dans la majorité des cas, les graphes sont creux (sparse), c'est-à-dire qu'ils contiennent beaucoup moins d'arrêtes que le nombre maximum possible.
   Cela signifie que la matrice d'adjacence contiendra beaucoup de zéros, ce qui est inefficace en termes de mémoire et de calcul, car ce sont bien les arrêtes présentes qui portent l'information.
2. Dans une matrice d'adjacence, rien ne nous interdit de permuter les nœuds du graphe. Il existe donc en réalité $n!$ matrices d'adjacence équivalentes pour un même graphe.
   Le problème, c'est que toutes ces représentations équivalentes d'un même graphe, après passage dans un réseau de neuronnes, ne garantissent pas de fournir un résultat identique. Cela signifie qu'il serait possible d'obtenir des résultats différent pour un même graphe, simplement en permutant les oeuds dans la matrice d'adjacence.

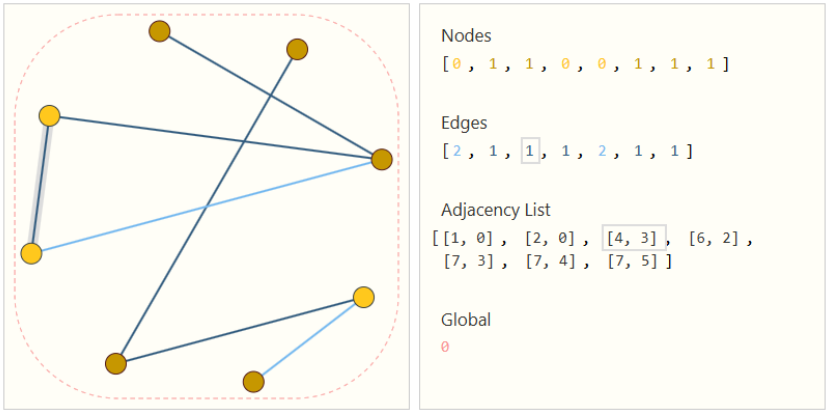


Exemple de deux matrices d'adjacence représentant un même graphe.

Pour résoudre ces deux problèmes, on utilise donc une autre représentation, basée sur la **liste d'adjacence**.

Représentation par liste d'adjacence

- La liste V_0 des valeurs prises par les noeuds
- La liste d'adjacence L, qui est la liste des arrêtes présente dans le graphe
- La liste E_0 des valeurs prises par les arrêtes
- Une valeur U_0 représentant globalement le graphe entier (*voir la sous-section II-C — Utilisation de la représentation globale : Le Master Node — pour son utilisation*)



Exemple de représentation d'un graphe par liste d'adjacence

Dans l'exemple ci-dessus, les valeurs prises par les noeuds, les arrêtes et le graphe global sont scalaires, mais la représentation se généralise aux valeurs vectorielles.

L'utilisation de la liste d'adjacence permet de ne stocker que les arrêtes présentes dans le graphe, ce qui est plus efficace en termes de mémoire et de calcul.

De plus, les conditions sur l'ordre des noeuds et des arrêtes permettent d'assurer une invariance de la représentation du graphe par permutation des noeuds.

II Construction d'un GNN simple

II-A Le modèle de base

En utilisant la représentation précédemment définie, on peut construire un 1er modèle de GNN simple.

En notant U_n, V_n et E_n les valeurs respectives du graphe global, des arrêtes et des noeuds à l'étape n du réseau de neurones, on a alors :

— $U_{n+1} = f(U_n)$

— $V_{n+1} = g(V_n)$

— $E_{n+1} = h(E_n)$

Où f, g et h sont des *update functions* qui peuvent être chacune un réseaux de neuronnes standard, ou toute autre fonction souhaitée (voir par exemple le Chapitre II à propos des convolutions sur graphes).

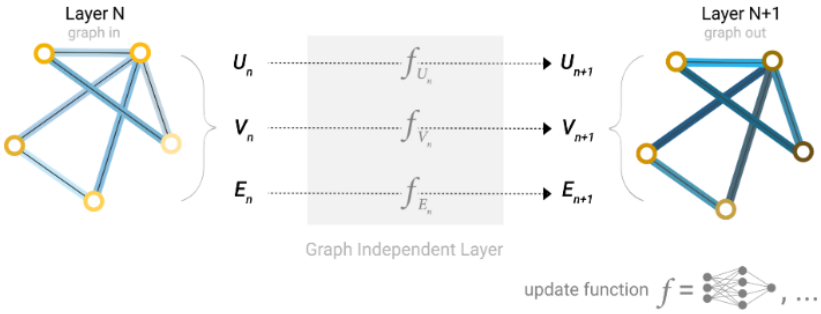


Schéma d'un bloc GNN

Cette démarche forme alors un bloc GNN que l'on peut alors répéter à partir de la représentation initiale du graphe (U_0, V_0, E_0) pour calculer des représentation (U_n, V_n, E_n).

Ensuite, on va chercher à faire des prédictions à partir de la sortie des blocs GNN. Pour cela, on applique simplement une fonction de prédiction C sur les valeurs de sortie du bloc GNN, en choisissant l'ensemble U_n, V_n ou E_n selon la tâche à réaliser :

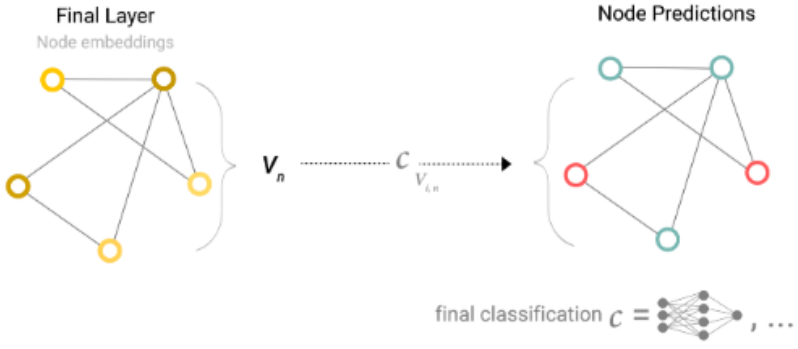


Schéma d'une tâche de classification de noeuds

À ce stade, il est important de noter que ce premier modèle de GNN ne prend pas du tout en compte la connectivité du graphe, puisque nous ne nous sommes pas servis de la liste d'adjacence.

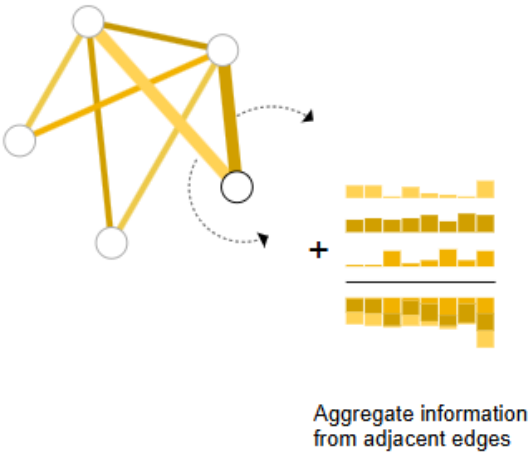
II-B Les opérations de pooling

Pour prendre en compte la connectivité du graphe, on va introduire une opération clé des GNN : le **pooling**.

Défintion : Le pooling

Le pooling est une opération qui agrège les informations des noeuds et/ou arrêtes voisins d'un noeud et/ou arrêtes donné pour mettre à jour la valeur de cet élément en prenant en compte son contexte local dans le graphe.

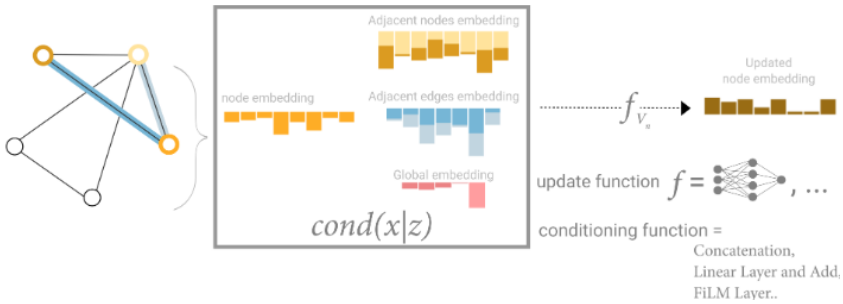
Cette opération peut être réalisée de différentes manières, telles que la somme, la moyenne, le maximum, ou des méthodes plus complexes, comme avec des réseaux d'attention.



Exemple de pooling sur un noeud

Sur l'exemple ci-contre, on effectue une opération de pooling sur le noeud séléctionné (en gras), en agrégeant les informations des arrêtes adjacentes pour mettre à jour la valeur du noeud.

Cela dit, les représentations des noeuds et des arrêtes, ne sont pas nécéssairement identiques, en particulier la taille du vecteur qui les représentent. Pour cette raison, il faut s'intérésser au conditionnement des pooling.



Le conditionnement des pooling

Pour conditionner les poolings, et donc gérer les différences taille de représentation entre noeuds et arrêtes, on utilise des fonctions de conditionnement qui permettent d'adapter les valeurs avant l'agrégation. Celle-ci peuvent être des concaténations, des Linear Mapping Layers (à apprendre avec un autre réseau par exemple) ou d'autres fonctions d'adaptation.

Interessons nous maintenant à deux usages du pooling dans les GNN :

Le pooling en fin de réseau, juste avant la prédiction :

Dans ce cas, à partir des informations finales (U_n, V_n, E_n) en sortie des blocs GNN, on applique une opération de pooling pour obtenir une nouvelle représentation agrégée du graphe avant de faire la prédiction.

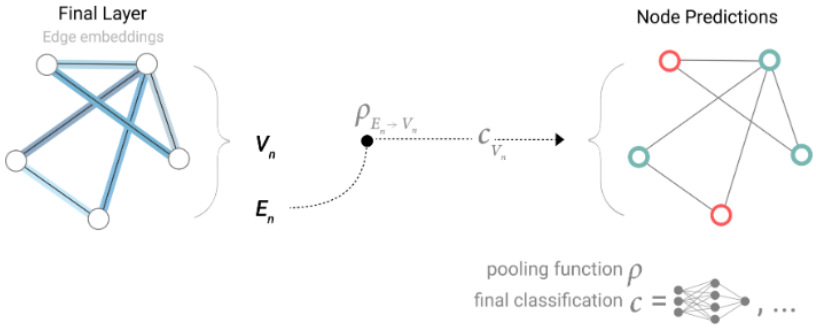


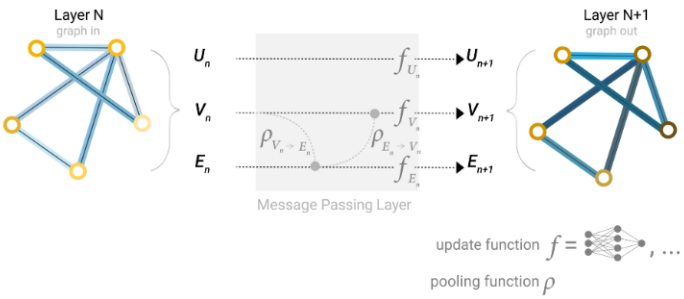
Schéma d'un exemple sur une tâche de classification de noeuds

Le pooling dans les blocs GNN - le message passing :

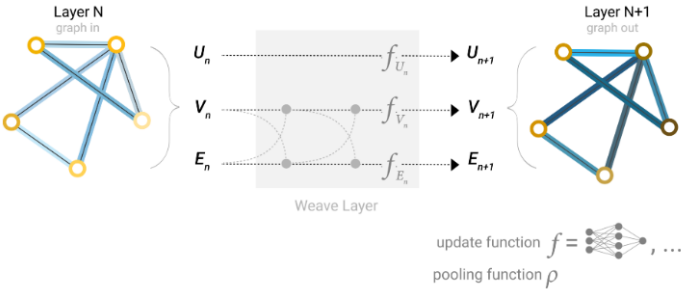
On peut aussi faire le pooling à l'intérieur des blocs GNN, pour prendre en compte la connectivité du graphe à chaque étape de mise à jour des valeurs (U_n, V_n, E_n). Cela permet de propager l'information à travers le graphe, en mettant à jour les valeurs des noeuds et des arrêtes en fonction de leurs voisins.

On voit sur le premier exemple ci-dessous que, pour effectuer le message passing pour atteindre les noeuds voisins, de part la représentation utilisée, on doit en fait d'abord faire un message passing des noeuds vers les arrêtes, puis des arrêtes vers les noeuds.

De la même manière, on peut faire un message passing sur les arrêtes en utilisant les noeuds comme intermédiaires, et en superposant les deux approches, on obtient un message passing local complet.



Exemple de message passing sur les noeuds



Exemple de message passing local complet

Distance de réceptivité du message passing

k étapes de message passing permettent à chaque noeud (ou arrête) d'incorporer l'information de son k-hop voisinage, c'est-à-dire des noeuds (ou arrêtes) situés à une distance maximale de k.

Cela signifie que, pour capturer des informations plus globales du graphe, il est nécessaire d'augmenter le nombre de message-passing effectués dans le GNN. Mais pour les très grands graphes (par exemple les réseaux sociaux), le nombre à considérer est bien trop important pour envisager de le faire de manière efficace.

II-C Utilisation de la représentation globale : Le Master Node

L'utilisation de la représentation globale du graphe, aussi appelée **Master Node**, permet de contourner la limitation du message passing pour capturer des informations globales du graphe.

Pour ce faire, on fait un pooling du Master Node vers les noeuds et les arrêtes, puis un pooling des noeuds et des arrêtes vers le Master Node. Cela permet au Master Node de collecter des informations de l'ensemble du graphe, et de les redistribuer aux noeuds et arrêtes à chaque étape, facilitant ainsi la propagation d'informations globales à travers le graphe.

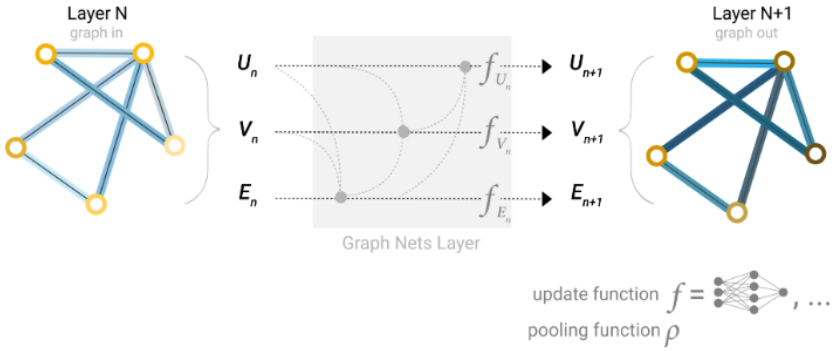


Schéma d'une utilisation du Master Node dans un bloc GNN

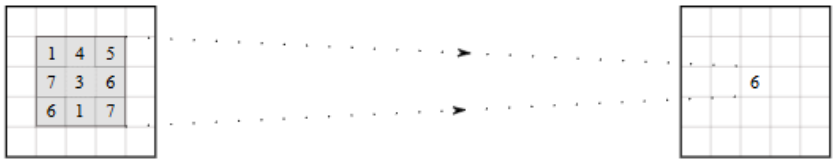
Chapitre II

Les Graph Convolutional Networks (GCN)

L'ensemble de ce chapitre se base sur la lecture de l'article *Understanding Convolutions on Graphs* [2].

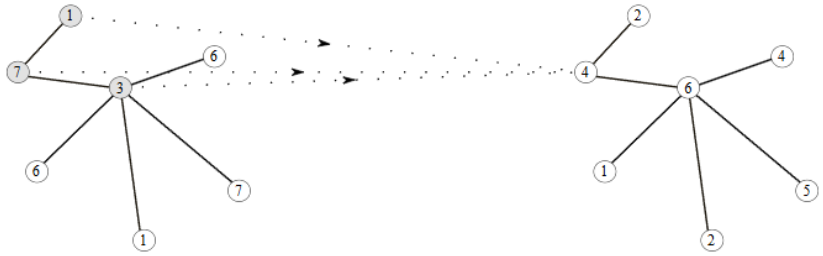
Un cas particulier de GNN très utilisé est celui des **Graph Convolutional Networks (GCN)**, dont l'objectif principal est de généraliser les réseaux de convolutions aux graphes.

On rappelle que les opérations de convolution sont des opérations locales, initialement pensées pour les images, qui permettent d'extraire des caractéristiques locales en appliquant des filtres sur des régions restreintes des données.



Exemple de filtre de convolution sur les images

L'idée serait donc de créer des filtres de convolution adaptés aux graphes, qui pourraient être utilisés dans des blocs GNN pour extraire des caractéristiques locales des graphes.



Exemple de filtre de convolution sur les graphes

I Les Convolutions Polynomiales

I-A Le filtre Laplacien

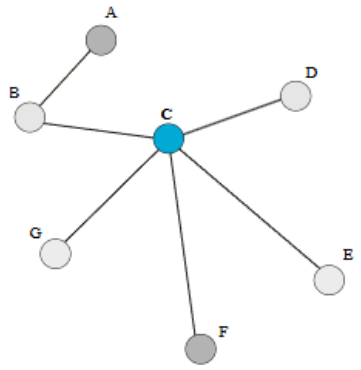
Une première approche pour définir des filtres de convolution sur les graphes est d'utiliser le Laplacien du graphe.

Définition : Le Laplacien d'un graphe

Le Laplacien L d'un graphe est défini comme :

$$L = D - A$$

Où D est la matrice de degré (matrice diagonale où chaque élément diagonal D_{ii} est égal au degré du noeud i) et A est la matrice d'adjacence du graphe.



$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{bmatrix} 1 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 5 & -1 & -1 & -1 & -1 \\ & & -1 & 1 & & & \\ & & -1 & & 1 & & \\ & & -1 & & & 1 & \\ & & -1 & & & & 1 \end{bmatrix} \end{array}$$

Exemple du Laplacien d'un graphe

En notant alors x les valeurs prises par les noeuds du graphe (dans l'ordre des noeuds utilisé dans A, D et L), on peut définir une opération de convolution sur le graphe en utilisant le Laplacien :

$$x' = LX$$

Cette opération permet de propager l'information locale à travers le graphe, en mettant à jour les valeurs des noeuds en fonction de leurs voisins.

La matrice L du Laplacien est **semie-définie positive** pour les graphes non orientés.

Preuve :

Soit $x \in \mathbb{R}^n$. $x^T L x = x^T(D - A)x = \sum_i d_i x_i^2 - \sum_{i,j} A_{ij} x_i x_j$

Or $d_i = \sum_j A_{ij}$, donc $\sum_i d_i x_i^2 = \sum_{i,j} A_{ij} x_i^2$.

Comme de plus, le graphe est non orienté, A est symétrique et donc $\sum_{i,j} A_{ij} x_i^2 = \frac{1}{2} \sum_{i,j} A_{ij}(x_i^2 + x_j^2)$.

Finalement $x^T L x = \frac{1}{2} \sum_{i,j} A_{i,j}(x_i^2 + x_j^2 - 2x_i x_j) = \frac{1}{2} \sum (x_i^- x_j)^2 \geqslant 0$

CQFD

I-B Les polynômes du Laplacien

On peut également élargir le noyau de convolution de cette approche. Pour cela, on utilisee des polynômes du Laplacien pour définir des filtres de convolution entre des nœuds qui ne sont pas voisins directs.

En notant $P_w(L) = \sum_{i=0}^{d} w_i L^i$ un polynôme de degré d du Laplacien, on peut définir l'opération de convolution :

$$x' = P_w(L)x$$

avec w_i les coefficients du polynome à apprendre.

Lien entre puissance de L et distance dans le graphe

Soit $i \in \mathbb{N}$. Soit u et v deux noeuds du graphe G.

Si u et v sont à une distance strictement supérieure a i dans le graphe G, alors la valeur $L^i_{uv} = 0$.

$$\text{dist}_G(v,u) > i \Rightarrow L^i_{uv} = 0$$

En particulier, la contraposée donne :

$$L^i_{uv} \neq 0 \Rightarrow \text{dist}_G(v,u) \leq i$$

Preuve :

Soit $i \in \mathbb{N}$ Soit u et v deux noeuds du graphe G tels que $\text{dist}_G(u,v) > i$.

Si $i = 1$, alors on a que $L_{uv} = 0$ car $u \neq v$ et qu'il n'existe donc pas d'arrête entre u et v.
Si $i > 1$, on a, par répétition de l'expression du produit natriciel :

$$L^i_{uv} = \sum_k (L_{uk_1})(L_{k_1k_2})(L_{k_2k_3})...(L_{k_{i-1}v})$$

avec la somme prise sur l'ensemble des séquences de longueur $i-1$: $(k_1, k_2, ..., k_{i-1})$.
Supposons maintenant par l'absurde que $L^i_{u,v} \neq 0$.
Alors, il existe une séquence $(k_1, k_2, ..., k_{i-1})$ telle que : $L_{uk_1} \neq 0, L_{k_1k_2} \neq 0, L_{k_2k_3} \neq 0, ..., L_{k_{i-1}v} \neq 0$
Cela implique l'existence d'un chemin de longueur inférieure ou égale à i entre u et v passant par les noeuds $k_1, k_2, ..., k_{i-1}$, ce qui contredit l'hypothèse que $\text{dist}_G(u,v) > i$.

CQFD.

Cela résultat signifie que que l'application d'un polynôme de degré d du Laplacien peut propager l'information jusqu'aux noeuds situés à une distance maximale de d dans le graphe.

Intéressons nous à présent à une propriété importante des convolutions polynomiales, **l'équivariance par permutation** :

Définition : Permutation des noeuds d'un graphe

Soit un ordre des noeuds du graphes fixé.
On peut représenter une permutation de cet ordre de noeuds par une matrice de permutation P, qui est une matrice carrée binaire orthogonale, où chaque ligne et chaque colonne contient exactement un élément égal à 1.

Définition : Équivariance par permutation

Une opération f sur les graphes est dite **équivariante par permutation** si, pour toute permutation des noeuds représentée par une matrice de permutation P :

$$f(Px) = Pf(x)$$

Où x est une représentation quelconque du graphe (par exemple les valeurs des noeuds).

L'equivariance par permutation

Les convolutions polynomiales du Laplacien sont équivariantes par permutation.

Preuve :

Soit P une matrice de permutation.

En appliquant cette permutation aux éléments suvants du graphe :

- Les valeurs des noeuds x deviennent Px
- La matrice d'adjacence A devient PAP^T
- La matrice de degré D devient PDP^T
- Le Laplacien $L = D - A$ devient (par les deux points précédents) PLP^T
- Les puissances du Laplacien L^i deviennent $(PLP^T)^i = PL^iP^T$ grâce à l'orthogonalité de P

Finalement, en utilisant donc ici $f(x) = P_w(L)x$.

$$f(Px) = \sum_{i=0}^{d} w_i(PL^iP^T)(Px) = P\sum_{i=0}^{d} w_iL^ix = Pf(x)$$

CQFD

Ce résultat permet d'assurer que, réaliser une permutation des noeuds dans le graphe ne change pas le résultat de la convolution, et donc, dans le cas d'une utilisation en réseau, assure un résultat unique pour les différentes représentation d'un même graphe.

I-C Le ChebNet

Cette fois-ci, on définit $P_w(L) = \sum_{i=1}^{d} w_i T_i(\tilde{L})$ où

$$\begin{cases} \tilde{L} = \dfrac{2L}{\lambda_{max}} \text{ est le Laplacien normalisé où } \lambda_{max} \text{ est la valeur propre maximale de } L \\ T_i \text{ est le polynôme de Tchebychev de la première espèce, de degré } i \end{cases}$$

L'opération de convolution est toujours définie comme :

$$x' = P_w(L)x$$

avec w_i les coefficients du polynome à apprendre.

Explicitons à présent les motivations derrière ces choix :

1. En notant que L est semi-définie positive, toutes ses valeurs propres sont positives. En particulier, si $\lambda_{\max} > 1$ les puissance de L vont exploser, ce qui est problématique pour la stabilité nunérique.
   Cette normalisation assure donc que toutes les valeurs propres de $\tilde{L}$ soient desormais dans l'intervalle $[-1, 1]$, et les puissances de L restent bornées.
2. L'utilisation des polynômes de Tchebychev permet d'obtenir une base orthogonale pour les polynômes, ce qui améliore la stabilité numérique.

Rappel sur les polynômes de Tchebychev de la première espèce

Les polynômes de Tchebychev de la première espèce sont les polynomes orthogonaux vérifiant la propriété suivante :

$$T_n(\cos\theta) = \cos(n\theta)$$

On les obtient donc avec la relation de récurrence :

$$\begin{cases} T_0 = 1 \\ T_1 = X \\ T_{n+1} = 2XT_n - T_{n-1} \end{cases}$$

Il est donc clair que $\forall n \in \mathbb{N}, T_n$ est un polynôme de degré n.

Propriétés du ChebNet

Le ChebNet, défini ici, hérite des propriétés des convolutions polynomiales :

— Il est capable de capturer des informations locales jusqu'à une distance de réceptivité égale au degré du polynôme utilisé.

— Il est équivariant par permutation des noeuds du graphe.

I-D Utilisation dans un Graph Convolution Network (GCN)

L'utilisation générale des convolutions polynomiales ou Chebnet dans un GCN est relativement simple, en calculant les représentation des valeurs des noeuds à chaque couche par :

On note h^0 le vecteur contenant les valeurs initiales des noeuds (scalaires ici)

Puis, $\forall k \in \mathbb{N}$:

1. Définition du polynôme de convolution utilisé à la couche k (représenté matriciellement) :
$$p^{(k)} = P_{w^{(k)}}(L)$$
2. Application de la convolution :
$$g^{(k)} = p^{(k)} h^{(k-1)}$$
3. Application d'une fonction d'activation σ pour la non-linéarité :
$$h^{(k)} = \sigma(g^{(k)})$$

L'approche qui suit est utilisée et détaillée dans l'article original de Kipf et Welling *Semi-Supervised Classification with Graph Convolutional Networks* (2017). [3]

— On dévellope le calcul qui précède, avec un polynone du Laplacien $P_w = \sum_{i=0}^{d} w_i L^i$, on obtient :

$$h^{(k)} = \sigma\left(\left(\sum_{i=0}^{d} w_i L^i\right) h^{(k-1)}\right)$$

Ici, les poids w_i impacte chacun une puissance différente du Laplacien, et donc une distance de réceptivité différente, sans mélange.

— Or, pour l'apprentissage, il peut être intéressant de mélanger les informations provenant de différentes distances de réceptivité. Pour cela, en fixant $w_i = w$ pour tout i, on obtient :

$$h^{(k)} = \sigma\left(w\left(\sum_{i=0}^{d} L^i\right) h^{(k-1)}\right)$$

— Enfin, on peut généraliser cette approche aux valeurs vectorielles des noeuds, en notant $H^{(k)} \in \mathbb{R}^{n \times d_k}$ la matrice des représentations des noeuds à la couche k, avec d_k le nombre de features (i.e. la taille du vecteur représentant chaque noeud) à la couche k. On a alors :

$$H^{(k)} = \sigma\left(\left(\sum_{i=0}^{d} L^i\right) H^{(k-1)} W^{(k)}\right)$$

Ce qui s'écrit de manière équivalente, pour chaque noeud v, en intrégrant les valeurs de L^i_{uv} dans W :

$$h_v^{(k)} = \sigma\left(W^{(k)}\left(\sum_{\mathrm{Dist}_G(u,v) \leq d} h_u^{(k-1)}\right)\right)$$

II La Convolution Spectrale

Toutes les méthodes de convolutions sur les graphes vues précédemment sont des méthodes locales. Pour faire le lien avec le chapitre I, ces convolutions peuvent être vues comme des messages passing locaux à chaque étape des bloc GNN.

Comme vu dans le chapitre précédent donc, ces méthodes ne permettent donc pas de capturer efficacement des informations globales du graphe. Pour cela, on va donc maintenant s'intéresser aux convolutions spectrales.

II-A Défintion de la représentation spectrale

Il faut commencer par normaliser x tel que $displaystyle \sum_{i=1}^{n} x_i^2 = 1$.

Ensuite, d'après le théorème spectral, le Laplacien L d'un graphe étant réel symétrique, il est diagonalisable.

Il existe donc U une matrice contenant les vecteurs propres, pris orthonormaux entre-eux ($UU^T = U^TU = I$)

et $\Lambda = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_n \end{pmatrix}_{\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n}$ la matrice diagonale des valeurs propres de L, telle que : $L = U\Lambda U^T$

La matrice U définit donc une base orthonornée qui peut être utilisée pour effectuer une changement de base des valeurs des noeuds x du graphe, appelé **transformée de Fourier sur les graphes** :

$$\hat{x} = U^T x = \sum_{i=1}^{n} x_i u_i^T$$

La transformée de Fourier inverse est elle donnée par :

$$x = U\hat{x} = \sum_{i=1}^{n} \hat{x}_i u_i$$

Le changement de base permet bien de prendre en compte la structure du graphe globalement, car les vecteurs propres u_i sont des fonctions définies sur l'ensemble des noeuds du graphe.

On définit le quotient de Rayleigh par $R_L(x) = \dfrac{x^T L x}{x^T x} = \sum_{(i,j)\in E} (x_i - x_j)^2$,
Par analogie avec la théorie de Fourier, on peut alors interpréter les valeurs propres les plus faibles comme celles associées aux basses fréquences dans le graphe, liées aux différences de valeurs "lisses" entre des nœuds éloignés dans le graphe, c'est à dire des tendances globales.

A l'inverse, les valeurs propres les plus élevées sont associées aux hautes fréquences, liées aux différences de valeurs entre des nœuds proches dans le graphe, captant alors des différences moins "lisses", des détails très locaux.

II-B La troncature en représentation spectrale

Cependant, cette approche présente un inconvénient majeur : le calcul de la transformée de Fourier nécessite la diagonalisation du Laplacien, ce qui est coûteux en temps de calcul pour les grands graphes.

Or, en notant $\hat{x}_m = \text{Tronquer}_m(\hat{x}) = \text{Tronquer}_m(U^T x)$, la troncature aux m premiers coefficients de Fourier (en fixant à 0 les suivants). Cette opération de troncature est en fait équivalente à utiliser uniquement les m premiers vecteurs propres U_m, associés aux m valeurs propres les plus faibles, ce qui représente alors surtout les informations globales du graphe.

On peut alors définir une convolution spectrale tronquée, en utilisant uniquement les m premiers vecteurs propres :

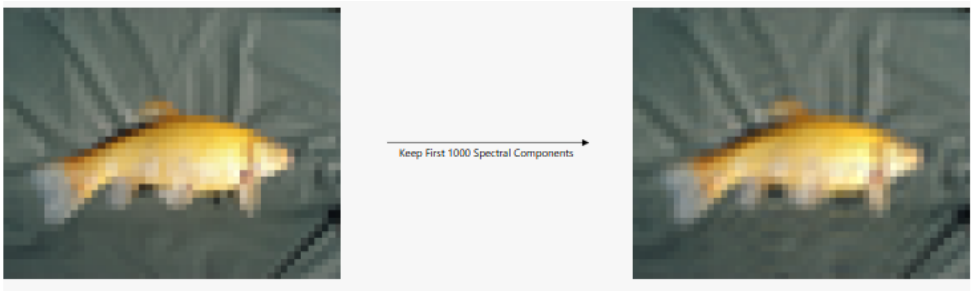
$$\hat{x} = U_m^T x$$

et son inverse, pour revenir en représentation spatiale :

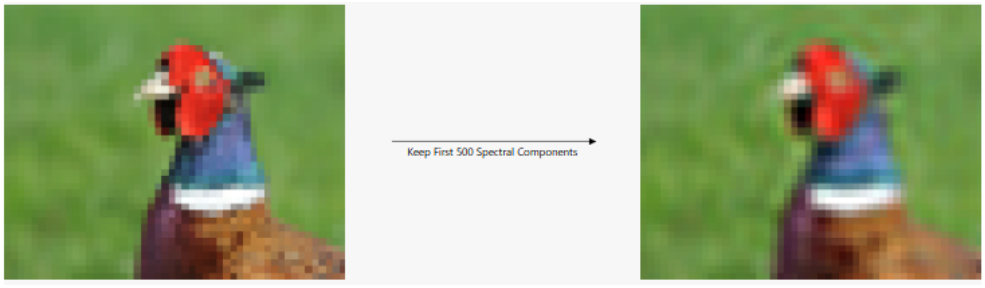
$$x_m = U_m \hat{x}_m$$

Cette approche permet de réduire considérablement le coût de calcul puisqu'il est plus rapide de déterminer m vecteurs propres que tous. De plus, on capture bien les caractéristiques globales du graphe en ne perdant que peu d'information sur les données.

Voici ci-dessous deux exemples, sur des images de 2000 pixels, traités commes des graphes reliants les pixels à leurs voisins, sur lesquelles on à appliquer une convolution spectrale tronquée.



Troncature avec 1000 vecteurs propres



Troncature avec 500 vecteurs propres

II-C Utilisation dans un Graph Convolutional Network (GCN)

L'utilisation de la convolution spectrale (éventuellement tronquée) dans un GCN est ensuite similaire à l'utilisation des convolutions polynomiales.

On note H^0 la matrice contenant les vecteurs de représentation de chaque noeuds

Puis, $\forall k \in \mathbb{N}$:

1. Passage en représentation spectrale, eventuellement tronquée
$$\hat{H}^{(k-1)} = U_m^T H^{(k-1)}$$
2. Ajout des poids d'apprentissage (appris par le réseau) par produit de Hadamard (Produit terme à terme des matrices) :
$$\hat{G}^{(k)} = W^{(k)} \odot H^{(k-1)}$$
C'st ici qu'a lieu la convolution puisque la transformée de Fourier effectuée à l'étape 1 transforme les produits de Hadamard en convolutions dans le domaine spatial.
3. Retour en représentation spatiale :
$$G^{(k)} = U_m \hat{G}^{(k)}$$
4. Application d'une fonction d'activation σ pour la non-linéarité :
$$H^{(k)} = \sigma(G^{(k)})$$

Finalement, tout cela s'écrit de manière compacte :

$$H^{(k)} = \sigma\left(U_m\left(W^{(k)} \odot \left(U_m^T H^{(k-1)}\right)\right)\right)$$

Bibliographie

[1] B. Sanchez-Lengleling, E. Reif, A. Pearce, A. Wiltschko (2021). A gentle Introduction to Graph Neural Networks. https://doi.org/10.23915/distill.00033

[2] A. Daigavane, B. Ravindran, G. Aggarwal (2021). Understanding Convolutions on Graphs. https://doi.org/10.23915/distill.00032

[3] T. N. Kipf, M. Welling (2017). Semi-Supervised Classification with Graph Convolutional Networks. https://arxiv.org/abs/1609.02907