Présentation d'article : Simple Yet Powerful Simplicial-aware Neural Networks

Présentation par Alexandre XIA Xinhao ZHAO

Sorbonne Université

Introduction

Les graphes sont des structures complexes à apprendre qui ne sont pas régulières. Afin de faire des tâches d'apprentissage supervisées, une façon de considérer les structures est de passer par des complexes simpliciels, une topologie qui permet de regrouper les noeuds du graphe.

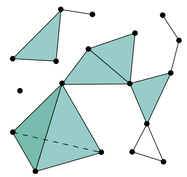

Figure 1. Exemple de complexe simpliciel (source : wikipédia)

Ainsi, les auteurs de l'article proposent une architecture de réseau de neurones qui prend en compte les propriétés de cette topologie pour pouvoir faire des représentations génériques s'adaptant à différentes tâches d'apprentissage supervisées sur des graphes.

Notions et contexte

Soit un graphe $G = (V, E)$, V l'ensemble des sommets, E les arêtes, on définit un complexe simpliciel K un ensemble de sous-ensembles de V (qu'on notera s) tel que tous les sous-ensembles de s sont inclus dans K. s de taille $k+1$ est appelé un k-simplexe. Les 0-simplexes sont les sous-ensembles ne contenant que des noeuds seuls par exemple. Les auteurs notent N_k le nombre de simplexes de taille $k+1$

Comparé aux graphes classiques, les simplexes on 4 types d'adjacences, qu'on définit informellement par:

- **upper** : 2 k-simplexes sont liés par un $k+1$-simplexe.
- **lower** : 2 k-simplexes sont liés par un $k-1$-simplexe
- **boundary** : 1 k-simplexe est lié à un $k+1$-simplexe
- **co-boundary** : 1 k-simplexe est lié à un $k-1$-simplexe

À partir de ces voisinages, on va de façon directe et sans apprentissage traiter des données de k-simplexe $X_k \in \mathbb{R}^{N_k \times D_k}$. Les différentes matrices de voisinages sont simplement définies à partir de la matrice d'incidence $B_k \in \mathbb{R}^{N_{k-1} \times N_k}$, avec des valeurs 1, 0 ou -1 dans le cas de simplexes orientés et 1 ou 0 dans le cas non-orienté. On définie respectivement les matrices de voisinages dites Laplacienne, 'upper' $A_{k,U} = B_{k+1}B_{k+1} \in \mathbb{R}^{N_k \times N_k}$ et 'lower', $A_{k,L} = B_k^T B_k \in \mathbb{R}^{N_k \times N_k}$ ainsi que les matrices 'boundary', $A_{k,B} = B_{k+1}^T \in \mathbb{R}^{N_{k+1} \times N_k}$ et 'co-boundary' $A_{k,C} = B_k \in \mathbb{R}^{N_{k-1} \times N_k}$.

Architecture et pré-traitement

Pré-traitement (partie sans apprentissage)

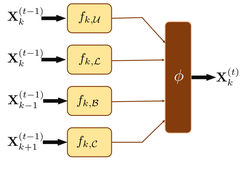

Figure 2. Pré-traitement des données selon le voisinage [3]

$X_k^{(t-1)}$ correspond aux informations des k-simplexes au 'hop' $(t-1)$, $X_{k+1}^{(t-1)}$ pour les $k+1$-simplexes et $X_{k-1}^{(t-1)}$ pour les $k-1$-simplexes. $X_k^{(0)} = X_k$

Les fonctions, $f_{k,n} : X_{k,n}^{(t)} \mapsto \mathbb{R}^{D_k}$ pour $n \in \{U, L, B, C\}$ prennent l'information sur les voisinages d'un k-simplexe. ϕ est une opération d'agrégation des informations sur les 4 types de voisinages.

Apprentissage des descripteurs de simplexes

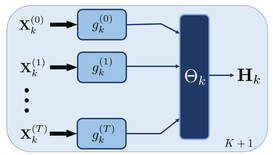

Figure 3. Blocs de transformation des descripteurs des k-simplexes [3]

Les descripteurs de chaque t-hop passent par des fonctions non-linéaires $g_k^{(t)}$ pour obtenir des 'embeddings'. On agrège les données de chaque k-simplexe et des "hop" par concaténation avec Θ_k et on obtient $H_k = \Theta_k(g_k^{(0)}(X_k^{(0)}), ..., g_k^{(T)}(X_k^{(T)}))$, T une limite arbitraire de 'hop'. On crée un bloc H_k par k-simplexe utilisé.

Exemple concret

Plus concrètement, un exemple faisant directement usage des matrices définies précédemment est fourni par les auteurs.

Pré-traitement

Les fonctions $f_{k,n}$ peuvent êtres directement obtenues par des multiplications matricielles, car la somme engendrée permet de garder la propriété injective. Les calculs sont définis par :

$$Y_{k,U}^{(t)} = A_{k,U}X_k^{(t-1)},\ Y_{k,L}^{(t)} = A_{k,L}X_k^{(t-1)},\ Y_{k,C}^{(t)} = A_{k+1,C}X_{k+1}^{(t-1)},\ Y_{k,B}^{(t)} = A_{k-1,B}X_{k-1}^{(t-1)} \quad (1)$$

Après agrégation, on obtient finalement, $X_k^{(t)} \in \mathbb{R}^{N_k \times D_k^{(t)}}$ avec $D_k^{(t)} = 2D_k^{(t-1)} + D_{k-1}^{(t-1)} + D_{k+1}^{(t-1)}$ avec

$$X_k^{(t)} = [Y_{k,U}^{(t)}, Y_{k,L}^{(t)}, Y_{k,C}^{(t)}, Y_{k,B}^{(t)}]^T \quad (2)$$

Apprentissage des descripteurs de simplexes

Les fonctions g_k sont des perceptrons multicouches et les auteurs précisent que pour préserver la propriété injective, il ne faut pas utiliser des réseaux de perceptrons à une seule couche.

Θ_k est une concaténation également et on a alors le résultat H_k obtenu dans la section "Architecture et pré-traitement".

Il est recommandé selon les auteurs de ne pas faire plus de 2 ou 3 "hop" pour limiter les dimensions des données et éviter des problèmes de "over smoothing" causant du sous-apprentissage.

Données et résultats

Pour les expériences, on a décidé de se concentrer sur la classification de graphe et la prédiction de chemin dans un graphe qui nous semblait êtres les plus intéressants.

Classification sur *proteins*

But : classifier 2 types de protéines avec comme donnée originale un one-hot encoding d'après l'article référencé [2] des auteurs et une matrice de 1 de même dimension pour les descripteurs que ceux des 0-simplexe (noeuds). On compare les résultats de certains de nos expériences à ceux des auteurs sur leur réseau et ajoute aussi le second meilleur auquel les auteurs se comparent :

	Auteurs		Nous		
Modèle	MSPN	SaNN	SaNN simplifié	SaNN	SaNN (MLP 128*128)
Accuracy (%)	76.5 ±3.4	77.6 ±2	74.3 ±1	75.6 ±5.3	73.4 ±3

Table 1. Résultats en performance des reproductions sur les données *proteins* avec validation croisée. SaNN simplifié est une version par nous où nous avons refait le pré-traitement à partir de l'exemple donné (sans validation croisée).

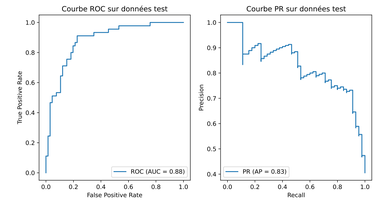

Figure 4. Courbes ROC AUC et ROC PR pour le SaNN simplifié

Nous montrons une courbe de ROC AUC et ROC PR pour le SaNN simplifié pour vérifier la cohérence avec nos pré-traitements.
On observe que l'exemple qu'on a refait est plutôt bon.
Pas de validation croisé dans notre cas, car il aurait été long de refaire l'expérience plusieurs fois en plus du pré-traitement.

Prédiction de trajectoire

But: Prédire le prochain nœud dans une séquence formée par une série de nœuds connectés par des arêtes, avec des flux orientés sur les arêtes.
Voici quelques points importants pour cette application et réalisation de l'expérience:

- On utilise des matrices d'incidence orientées pour l'agrégation.
- On applique un 5-fold validation croisée pour évaluer la performance de la modèle. Ainsi un early stoping notamment pour l'expérience avec les données Mesh.
- On utilise *Ocean*[4] et *Mesh*[1] comme les données pour tester le modèle.

On étudie l'effet des caractéristiques pré calculées agrégées à partir de différentes "hop" de voisinage. On obtient les résultats suivants qui ne contient que nos expériences:

	Ocean		Mesh	
Dataset	Accuracy (%)	Temps utilisé par epoch (s)	Accuracy (%)	Temps utilisé par epoch (s)
SaNN (1-hop)	**31 ± 0.6**	0.34	**75±3.0**	20.7
SaNN (2-hop)	30±0.7	0.42	70±2.8	22.6
SaNN (3-hop)	29±0.7	0.42	64±2.9	24.7

Table 2. Résultats prédiction de la trajectoire

On constate que les informations au voisinage de profondeur 1 sont généralement les plus informatives, ce qui implique que les informations locales sont cruciales pour les 2 expériences. Les informations au voisinage de profondeur 3 ne semblent pas si pertinentes pour ces expériences.

Conclusion et critique

- Article accepté pour la conférence ICLR 2024 et un séminaire a été fait par le directeur de thèse. [3]
- Cependant, l'article en lui même sur openreview présente une faute théorique sur les matrices utilisées, puis corrigé dans le séminaire.
- Le cas $k = 0$ qui n'est pas détaillé dans l'article ce qui rend les données pré-traitées plus difficile à comprendre.
- L'article si on retire la faute est relativement claire (surtout si on ajoute le séminaire).

References

[1] Jean-Baptiste Cordonnier and Andreas Loukas. Extrapolating paths with graph neural networks, 2019.

[2] F. Errica et al. A fair comparison of graph neural networks for graph classification. *In Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

[3] S. Gurugubelli and S. P. Chepuri. Sann : Simple yet powerful simplicial-aware neural networks. *ICLR 2024*.

[4] T. Mitchell Roddenberry, Nicholas Glaze, and Santiago Segarra. Principled simplicial neural networks for trajectory prediction, 2021.