Apprentissage sur graphes

GraphML – 2e partie

Raphaël Fournier-S'niehotta

CNAM Paris, fournier@cnam.fr

HTT-FOD
RCP217
2020-2021

le cnam

Plan

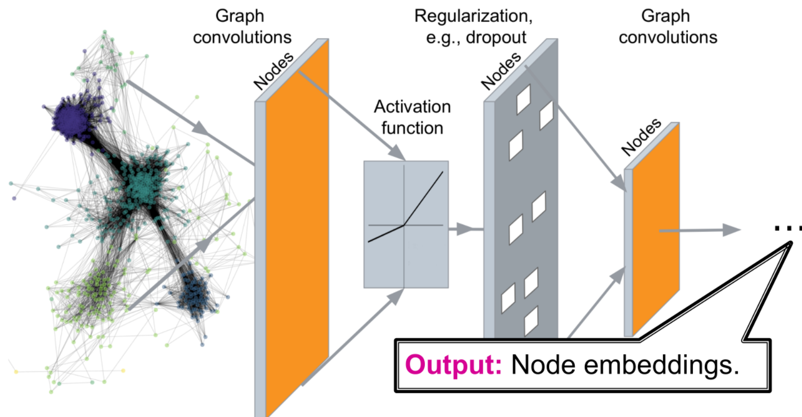
1 | Idées générales

2 | Les composants du framework GNN
1 – Messages et agrégation
2 – Quelques modèles classiques de couches GNN
3 – Connexions
4 – Modification du graphe
5 – Entraînement

3 | Sujets avancés
1 – Limites des GNNs
2 – Passage à l'échelle

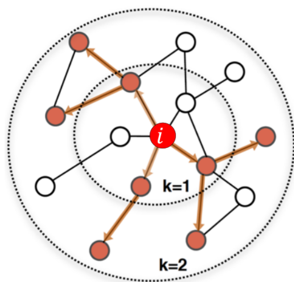
Idées générales

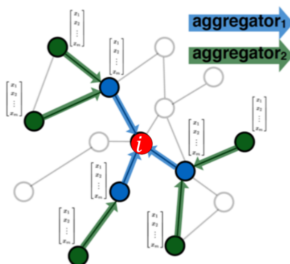
Ce qu'on veut



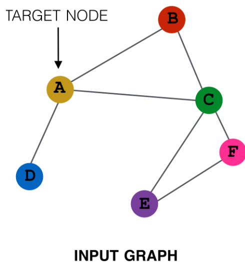
Idée

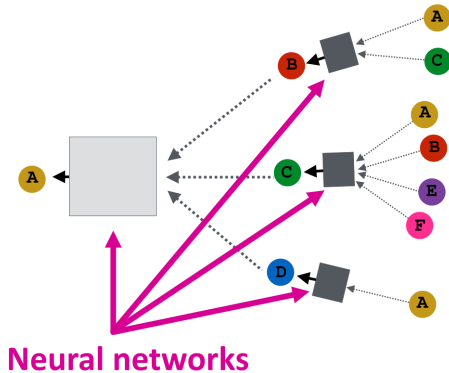
Propagation de l'information dans le graphe, localement, pour calculer des représentations vectorielles des nœuds





Agrégations

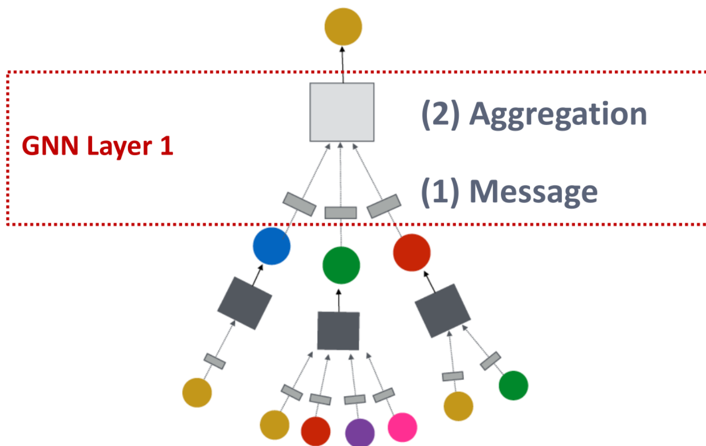




Les composants du framework GNN

Framework GNN

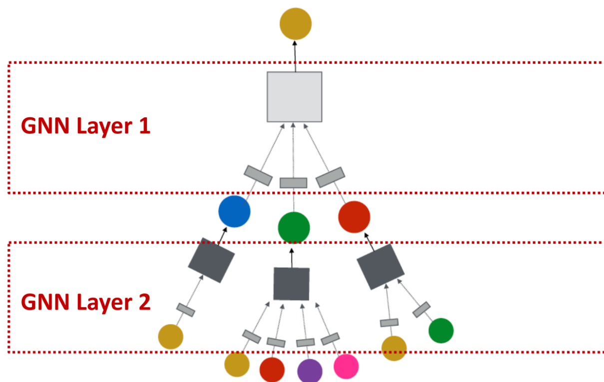
- Une couche GNN, c'est :
  - des "messages"
  - leur agrégation



Framework GNN

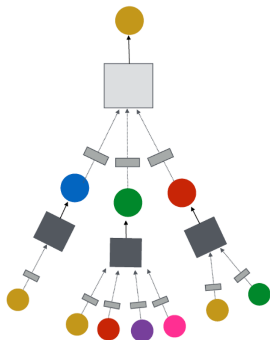
- Des connexions entre couches GNN
  - différentes manières de les empiler
  - des raccourcis (skip connections)

(3) Layer connectivity



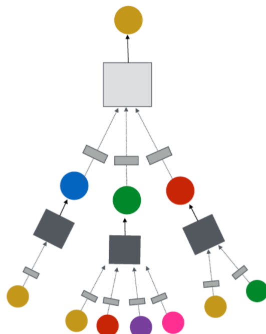
Framework GNN

- le ”graphe de calcul”, à partir duquel sont calculés les messages, peut être différent du graphe ”brut”
  - ajout d’attributs
  - modification de la structure

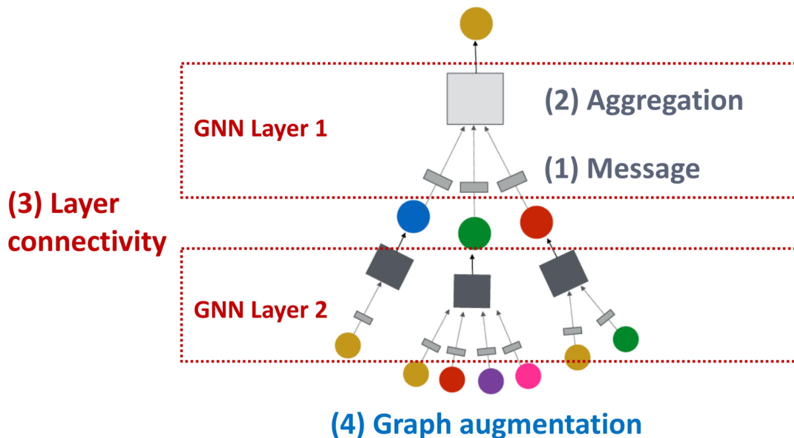


Framework GNN

- l'entraînement du GNN peut se faire
  - en supervisé/non supervisé
  - à plusieurs niveaux : Noeud/Lien/Graphe

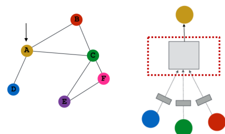


Framework GNN : synthèse



Message

- L'idée d'une couche GNN, c'est de compresser un ensemble de vecteurs (celui d'un nœud, de ses voisins) en un seul
- 2 étapes : Message + Agrégation



Message

- Un *message* est créé par chaque nœud, passés aux autres ensuite

$$m_u^{(l)} = MSG^{(l)}(h_u^{(l-1)})$$

- En linéaire : $m_u^{(l)} = W^{(l)} h_u^{(l-1)}$

Agrégation

- Chaque nœud agrège les messages de ses voisins

$$h_u^{(l)} = AGG^{(l)}(\{m_v^{(l)}, v \in N(u)\})$$

- Somme, Moyenne, Max

Information du nœud

Problème de l'agrégation

- Perte de l'information sur chaque nœud (si le calcul de $h_u^{(l)}$ ne dépend pas de $h_u^{(l-1)}$)

- Solution : on inclut $h_u^{(l-1)}$, dans le message et l'agrégation
- Dans le message, avec un message spécifique $m_u^{(l)}$
- Mais aussi dans l'agrégat, en concaténant avec l'agrégation des voisins :

$$h_u^{(l)} = \text{CONCAT}(AGG^{(l)}(\{m_v^{(l)}, v \in N(u)\}), m_u)$$

GNN classiques : GCN

- Graph Convolutional Networks (Kipf Welling 2017)

$$h_v^{(l)} = \sigma\big(W^l \sum_{u \in N(v)} \frac{h_u^{(l-1)}}{|N(v)|}\big)$$

- Message (chaque voisin) : $m_u^{(l)} = \frac{1}{|N(v)|} W^{(l)} h_u^{(l-1)}$
- Agrégation : Somme sur tous les voisins, puis activation

GraphSage

$$h_v^{(l)} = \sigma\left(W^{(l)} \qquad \text{CONCAT}\left(h_u^{(l-1)}, \text{AGG}\left(\left\{h_u^{(l-1)}, u \in N(v)\right\}\right)\right)\right)$$

- Message : AGG (variable)

- Agrégation en 2 phases :
  - on agrège le voisinage
  - on agrègre avec le nœud lui-même
- AGG peut être :
  - Moyenne pondérée : $\text{AGG} = \sum_{u \in N(v)} \frac{h_u^{(l-1)}}{|N(v)|}$
  - Pooling : $\text{AGG} = \text{Mean}(\text{MLP}(h_u^{(l-1)}, \forall u \in N(v)))$
- Optionnel : L2 normalisation à chaque couche

Graph Attention Networks

$$h_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu} W^l h_u^{(l-1)} \right)$$

- Dans GCN/GraphSage, $\alpha_{vu} = \frac{1}{|N(v)|}$
- Ces poids sont définis à partir du degré
- Tous les voisins de v sont également importants pour v

- L'idée est d'améliorer l'agrégation du voisinage, en allouant des poids variés (et appris) à différents voisins

Mécanisme attentionnel

- α_{uv} est le résultat d'un calcul de coefficients d'attention bruts e_{vu} :
  - $e_{vu} = \text{ATT}\left(W^l h_u^{(l-1)}, W^l h_v^{(l-1)}\right)$
  - les e_{vu} sont normalisés par softmax, pour que $\sum_{u \in N(v)} \alpha_{vu} = 1$

$$\alpha_{uv} = \frac{exp(e_{vu})}{\sum_{k \in N(v)} exp(e_{vk})}$$

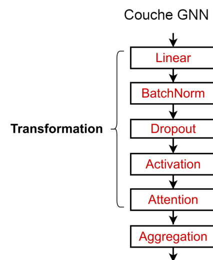
- le mécanisme d'attention ATT peut être un NN simple couche, dont les paramètres sont appris en même temps que W^l

Bénéfice de l'attention

- Efficacité computationnelle : parallélisation sur les liens / les nœuds
- Efficacité en espace : $\mathcal{O}(V + E)$ pour les matrices à stocker, nombre fixé de paramètres (ne dépend pas de n)
- Inductif : ne dépend que de l'arête (localité)

Améliorations : des couches GNN augmentées

- On a vu les blocs à la base d'une couche GNN
- On peut aussi inclure d'autres éléments dans les couches :
  - Batch Normalization (stabilise l'apprentissage). Centrage des embeddings pour avoir une moyenne 0 et une variance unitaire
  - Dropout (évite le sur-apprentissage). Probabilité de désactiver quelques neurones dans l'entraînement (couche linéaire)
  - Activation. ReLU, Sigmoïde, ReLU paramétrique ($y = ax, x < 0$ avec a appris)
  - Attention (contrôle l'importance des messages)



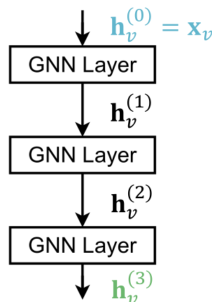
Empiler des couches GNN

Manière standard de construire un GNN :

- on place les couches les unes à la suite des autres
- entrée : les attributs du nœud dans le vecteur x_v
- sortie : les embeddings $h_v^{(L)}$ après L couches

Problème :

- les embeddings convergent vers des valeurs similaires
- gênant car on veut les utiliser pour différencier des nœuds !

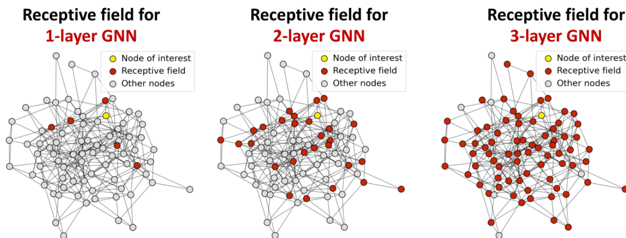


Champ réceptif

Que se passe-t-il ?

- On parle du champ réceptif pour désigner l'ensemble des nœuds qui contribuent à l'embedding d'un nœud donné
- Dans un GNN de K couches, chaque nœud a un champ réceptif correspondant à son voisinage à distance K (!)
- Les voisins en commun deviennent rapidement nombreux (quand on augmente le nombre de couches)
- À distance 3, presque tous les sommets

- Conséquence : ces voisinages avec beaucoup de recouvrement induisent des embeddings très similaires !



Champ réceptif

- Il faut donc être prudent dans l'ajout/empilement des couches GNN
- Contrairement aux images (CNN), ajouter des couches n'aide pas toujours
  - Il faut analyser le champ réceptif nécessaire au problème (par exemple, en analysant le diamètre du graphe)
  - Choisir L pas trop grand !

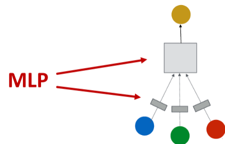
3 solutions possibles :

- Accroître l'expressivité de chaque couche (complexifier)
- Ajouter des couches qui ne font pas de passage de messages
- Ajouter des raccourcis entre couches (*skip-connections*)

Solutions

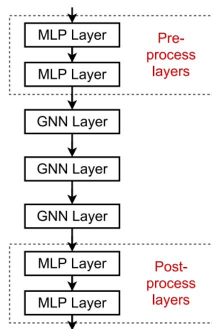
Expressivité

- dans nos exemples, on n'avait qu'une couche linéaire pour l'agrégation/transformation
- chaque agrégation/transformation peut être un réseau de neurones (par exemple : un MLP de 3 couches)



Autres couches

- on peut avoir des couches ”non GNN” dans un ”GNN”
- des couches de pré-traitement, par exemple pour encoder les features visuelles/textuelles associées à un nœud
- des couches de post-traitement, pour raisonner sur les embeddings eux-mêmes (classification de graphes)



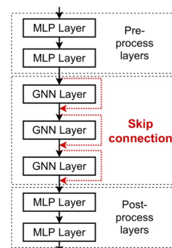
Raccourcis (*skip connections*)

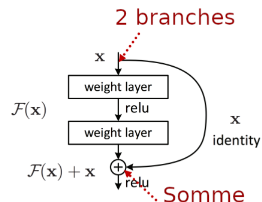
Si on a, malgré tout, besoin de nombreuses couches GNN, on peut ajouter des raccourcis.

- Pourquoi ? Les embeddings des premières couches différencient mieux les nœuds les uns des autres

$\Rightarrow$ On augmente leur influence avec des raccourcis dans le réseau

- Ça marche en créant, automatiquement, un mélange entre GNN "superficiel" et GNN profond (puisque l'information peut prendre divers chemins)
- N raccourcis $\Rightarrow 2^N$ chemins
- On peut aussi faire les raccourcis vers la dernière couche





Avant raccourci $\boldsymbol{F}(\mathbf{x})$
Après raccourci $\boldsymbol{F}(\mathbf{x}) + \mathbf{x}$

Ajouts

Jusqu'à présent, on a considéré que le graphe de calcul était basé sur le graphe brut. Pourtant :

- le graphe peut manquer d'attributs $\Rightarrow$ on peut en ajouter
- le graphe peut être trop creux $\Rightarrow$ passage de message peu efficace
- le graphe peut être trop dense $\Rightarrow$ passage de message trop coûteux
- le graphe peut être trop grand $\Rightarrow$ mémoire (GPU)

Le graphe brut est probablement pas optimal pour le calcul des embeddings.

Modifications du graphe

- ajout d'attributs
- Ajouts de nœuds/arêtes
- Échantillonnage de voisinage
- Échantillonnage par sous-graphes

Ajout d'attributs

- Pas d'attributs (ex. : on n'a que la matrice d'adjacence)
- 2 approches :

Valeur constante ajoutée aux nœuds

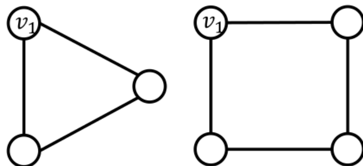
- les nœuds sont identiques, mais on peut tout de même apprendre de la structure
- généralisation aux nouveaux nœuds
- coût de calcul faible

Identifiants unique pour chaque nœud (1-hot encoding)

- on peut préserver l'information de chaque nœud
- généralise mal aux nouveaux nœuds
- taille élevée, donc coût de calcul plus important
- contexte : petits graphes, sans nouveaux nœuds

Autres ajouts d'attributs

- On peut aussi essayer de compléter les attributs existants, s'il y en a peu.
- Exemple : essayer de savoir si le nœud est dans un cycle, et quelle en est la longueur



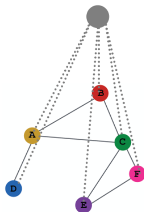
- D'autres caractéristiques classiques :
  - PageRank
  - Coefficient de clustering
  - Centralité

Modifications structurelles

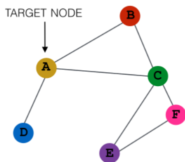
On peut souhaiter augmenter les graphes un peu creux

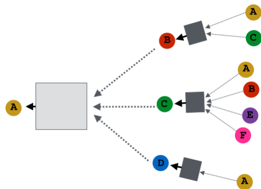
- Ajout de liens
  - Idée simple : ajout de liens directs pour les voisins à distance 2
  - traduction matricielle : au lieu de calculer avec la matrice d'adjacence A, on travaille avec $A + A^2$
- Ajouts d'arêtes
  - nœud virtuel connecté à tous les sommets
  - dans un graphe creux, où les plus courts chemins sont de taille 10 $\Rightarrow$ taille 2
  - A $\Rightarrow$ nœud virtuel $\Rightarrow$ B

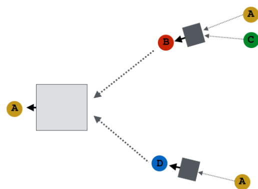
$\Rightarrow$ Améliore le passage de message



Échantillonnage de voisinage

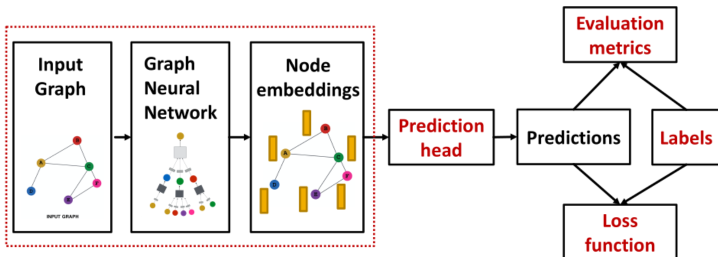






- On peut changer de voisins à chaque calcul
- En pratique, ça marche bien

Entraînement



- Différentes manières de faire des prédictions
  - Niveau nœud
  - Niveau lien
  - Niveau graphe

Niveau nœud

- Usage direct des embeddings possible
- On dispose d'embeddings de dimension d : $\{h_v^{(L)} \in \mathbb{R}^d, \forall v \in G\}$
- On veut prédire en k catégories
- Notre "tête de prédiction" est :

$$\hat{y} = \text{Head}_{node}(h_v^{(L)}) = W^{(H)} h_v^{(L)}$$

- $W^{(H)} \in \mathbb{R}^{k \times d}$

Niveau lien

- On utilise les embeddings de la paire de sommets qui sont liés par le lien
- 2 options principales pour $\text{Head}_{edge}(h_v^{(L)}, h_u^{(L)})$
  - Concaténation et couche linéaire :
  $$\hat{y}_{uv} = \text{Linear}(\text{Concat}(h_u^{(L)}, h_v^{(L)}))$$
  - Linear doit mapper de dimension 2d vers k.
  - Produit scalaire :
  $$\hat{y}_{uv} = {h_u^{(L)}}^T h_v^{(L)}$$
  - Marche pour la prédiction vers dimension 1 (link prediction)

Niveau graphe

- On veut utiliser tous les embeddings de notre graphe
- On veut

$$\hat{y} = \text{Head}_{graph}(\{h_v^{(L)} \in \mathbb{R}^d, \forall v \in G\})$$

- C'est comme l'agrégation dans une couche GNN, donc mêmes idées :
  - Mean, max, sum pooling
  - Attention, le pooling global perd de l'information
  - Pooling hiérarchique, par exemple (basé sur clusters)

Supervisé - non-supervisé

- Entraînement supervisé : les labels viennent de sources externes
- Entraînement non-supervisé : les signaux viennent du graphe lui-même
- parfois la différence est mince (si on entraîne pour prédire le coefficient de clustering)
- Idée : réduire la tâche à de la prédiction de label (ex. : du clustering)
- on parle de self-supervision. On peut utiliser des signaux classiques (stats de nœuds)

Classification, régression

- Classification : les labels que l'on prédit sont discrets
- Régression : "labels continus" (probabilité d'être une molécule viable)

- Les GNN peuvent s'utiliser dans les deux cas, la loss et la métrique d'évaluatuion changent !

Loss

- Classification : entropie croisée $CE = -\sum y_j \log(\hat{y}_j)$

E.g.

0	0	1	0	0

$\boldsymbol{y}^{(i)} \in \mathbb{R}^K$ = one-hot label encoding

$\hat{\boldsymbol{y}}^{(i)} \in \mathbb{R}^K$ = prediction after Softmax$(\cdot)$

E.g.

0.1	0.3	0.4	0.1	0.1

- MSE : $MSE = \sum(y_j - \hat{y}_j)$

Métriques

Régression

- RMSE
- MAE

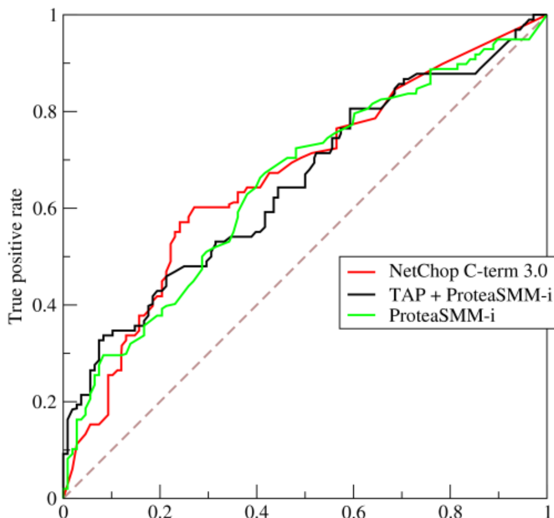
Classification

- Accuracy : $\frac{TP+TN}{TP+TN+FP+FN}$
- Précision $\frac{TP}{TP+FP}$
- Rappel $\frac{TP}{TP+FN}$ (Information Retrieval)

- ROC AUC

ROC AUC

- ROC : Receiver Operating Curve
- Affiche TPR ($\frac{TP}{TP+FN}$) comparé à FPR ($\frac{FP}{FP+TN}$)
- AUC : Area Under Roc. Probabilité qu'un classifieur classe une instance positive au-dessus d'une négative



Split de graphe

Pour entraîner notre GNN, il faut découper notre dataset.
On peut faire un split fixe :

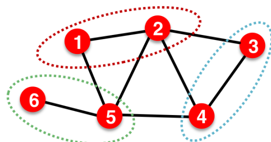
- Training set, pour optimiser les paramètres du GNN
- Validation set, pour améliorer le modèle
- Test set, pour la performance finale

- En imagerie, les instances sont indépendantes
- En graphe : non !

Training

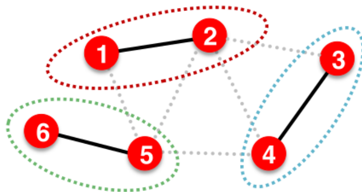
Validation

Test



Idées de solution

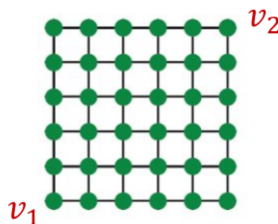
- Transductif : on garde le graphe entier dans tous les sous-ensembles
  - on splitte seulement les labels
  - on calcule donc les embeddings avec tout le graphe
  - mais on s'entraîne sur une partie des labels
  - à la validation, re-calcul sur tout le graphe, mais évaluation sur d'autres labels
- Inductif : on casse des arêtes, pour avoir des graphes indépendants



Sujets avancés

Limites des GNNs

- Deux nœuds avec le même voisinage ⇒ même embedding
- Pas le même voisinage ⇒ embedding différent
- Pourtant : on peut vouloir tenir compte de la position
  - *Position-aware* GNN



- Les nœuds dans des cycles ont les mêmes embeddings, malgré des longueurs variables
  - *Identity-aware* GNN

Expressivité des GNNs

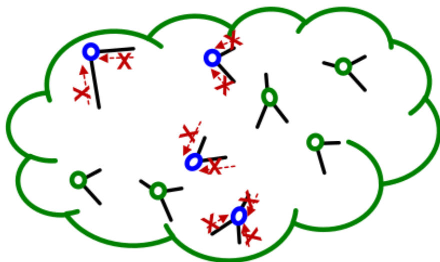
- L'expressivité des GNN, c'est leur capacité à distinguer des nœuds
- Elle dépend surtout de la manière dont sont agrégés les voisinages
- Les premiers GNN ne parviennent pas à distinguer des structures simples, on peut faire mieux
- Il existe un Graph Isomorphism Network (GIN, Xu et al. ICLR'19) dont on peut montrer que la fonction d'agréagtion est injective :
  - c'est le plus "expressif" des GNN
  - réussit à distinguer la plupart des graphes réels

Passage à l’échelle

- Les GNNs sont utilisés dans des contextes très grande échelle aujourd’hui
  - recommandation (10^8 utilisateurs, 10^7 produits)
  - réseaux sociaux (10^8 utilisateurs)
  - *drug discovery* (10^8 molécules)

Difficulté :

  - nœuds isolés si on échantillonne
  - SGD standard inefficace



Passage à l'échelle

- Idée naïve : full batch
- charger tout le graphe et les attributs
- chaque couche calcule des embeddings avec tous ceux de la couche d'avant
- Mais : une GPU, c'est 10/20 Go, pas 1 To (Ram)

Approches

- Pré-traitement d'attributs : Simplified GCN
- Passage de message sur des sous-graphes de taille réduite (Cluster-GCN, NeighborSampling)

Références

Remerciements

Ce cours doit beaucoup aux ressources suivantes :

- le cours de Jure Leskovec à Stanford
- le livre Graph Representation Learning de W. L. Hamilton

Version 1.0 du 7 juin 2021