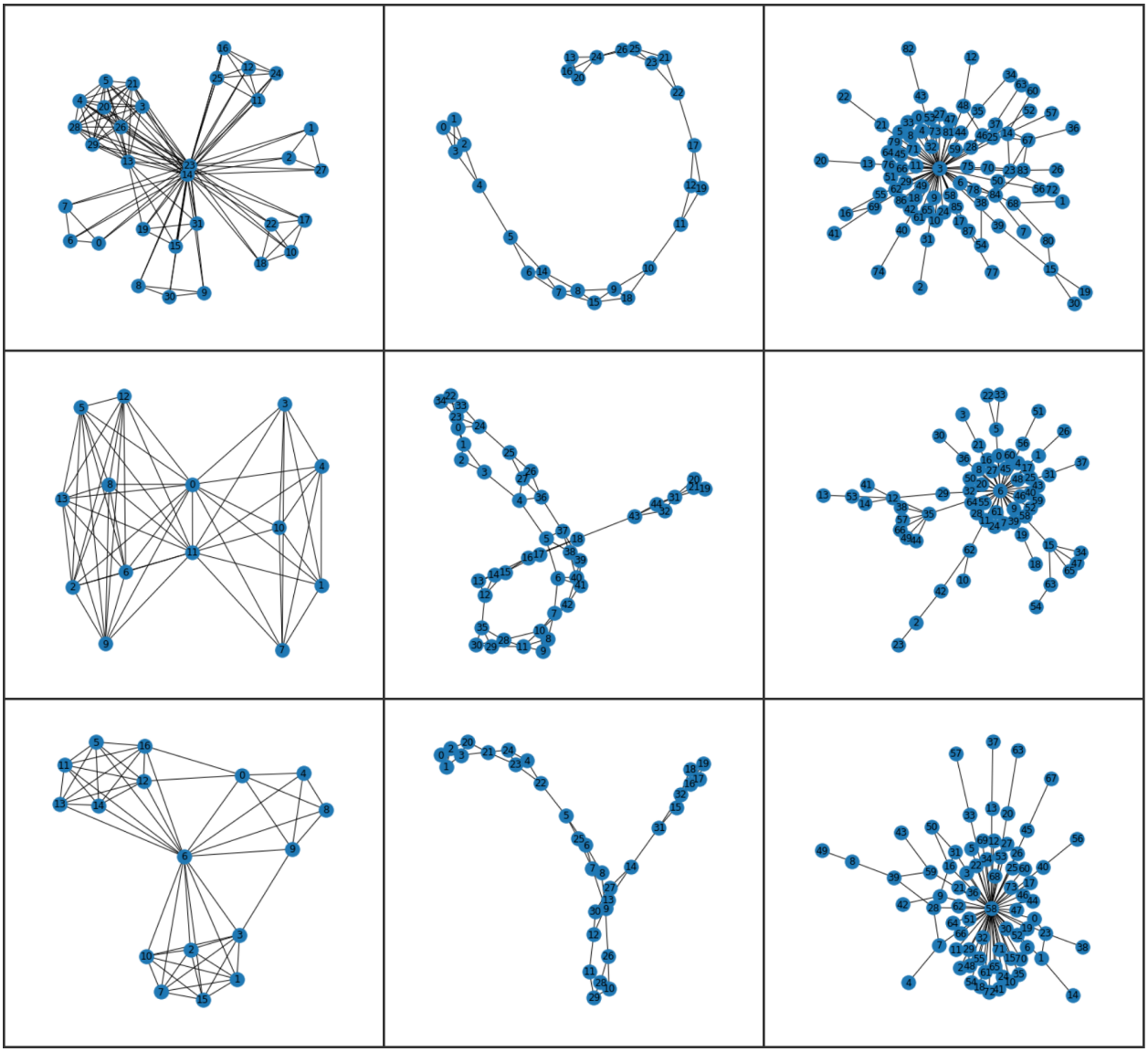
RAPPORT DE PROJET

Deep Graph Convolutional Neural Network



A partir de la publication de "An End-to-End Deep Learning Architecture for Graph Classification" écrit par Muhan Zhang, Zhicheng Cui, Marion Neumann, Yixin Chen, ce projet consiste à implémenter de cette méthode de classification de graph via une approche d'apprentissage profond.

Yans KHOJA - 2021/2022 (1er semestre) - CNAM – UE RCP27

Sommaire

Préambule

Dans le cadre de l'unité d'enseignement RCP217 enseigné au CNAM, ce projet porte sur l'implémentation et l'analyse d'une méthode de classification de graphs proposée par Muhan Zhang, Zhicheng Cui, Marion Neumann, Yixin Chen. Cette méthode consiste à classifier des graphs à l'aide d'un algorithme de deep learning.

Ce sujet me semblait intéressé car je suis issue d'une formation d'ingénieur-designer et je suis attiré par la vision par ordinateur notamment l'exploration des images de synthèses 2D et 3D.

J'ai réalisé ce projet individuellement par contrainte d'organisation et de temps. Il est joint à ce rapport le code commenté dont je détaillerai les grandes parties dans ce rapport. Je fournirai en annexe un READ.txt afin d'expliquer comment exécuter le code.

Introduction

Un graphe est simple à définir, c'est un ensemble de points reliés par des flèches ou des traits. Cependant, ce modèle transporte une quantité de données importantes en conservant une structure de l'information véhiculée par celle-ci. Ils répondent de façon simple à un besoin riche en informations.

Ces représentations sont relativement faciles à utiliser. On peut le constater dans notre quotidien avec, par exemple, le plan d'un réseau des transports en commun. En revanche, ils peuvent être beaucoup plus contraignant à apprendre car la connaissance transmise par leur configuration est plus riche que celle portée par leurs objets eux-mêmes.

Le recours au graphe comme support d'information se révèle de plus en plus utile car énormément de types de données se présentent naturellement avec un ordre établi. L'usage croissant des neural network a engendré un intérêt pour les graphes. Comprendre où réside l'information, comment se transmet-elle à travers un graphe et comment généraliser leur apprentissage sans perdre de l'information….

Dans ce rapport j'exposerai une des ses méthodes qui se veut "globale" car apprenant sur un quelconque ensemble de graphes et sur l'intégralité du graphe. Dans un premier temps, je présenterai ma compréhension de cette publication. La deuxième partie consistera à exposer mon implémentation de cette méthode. Pour finir, je présenterai les divers résultats obtenus.

Détails de la méthode “An End-to-End Deep Learning Architecture for Graph Classification”

La méthode de Graph Neural Network décrite dans l’article “An End-to-End Deep Learning Architecture for Graph Classification” consiste à utiliser une architecture d’apprentissage profond au niveau du graphe entier. Elle a pour objectif la classification de graphe quelconque. Cette architecture appelée DGCNN se diffère de l’état de l’art des graph kernels grâce à l'utilisation de graph convolutional network et d’un phase readout permettant de généraliser la classification à d’un large panel de forme de graphe. En d'autres mots, cette méthode permet de classifier efficacement n’importe quelle structure et taille de graphes.

Apprentissage sur le graphe entier

Dans un premier temps, il est intéressant de comprendre quels signaux de graph on va utiliser et comment on va "transformer" nos graphes afin de pouvoir les mettre à disposition de notre algorithme et d’effectuer un apprentissage supervisé. Les graphes sont donc transformés en 3 matrices : Une matrice d’adjacence avec les auto boule des noeuds $\tilde{A} = A + I$ (où A est la matrice d'adjacence et I celle des auto boucles), d’une matrice d’information du graphe X correspondant soit aux attributs multidimensionnel ou non des nœuds ou soit à une représentation des degrés des nœuds normalisée et la matrice des degrés de nœud.

Algorithme utilisé

À partir de ces matrices représentant le graphe, le modèle proposé est décomposé en 3 phases séquentielles :

1) Une couche GCN qui va extraire les caractéristiques des sous-structures locales et un ordre cohérent dans la configuration des sommets.
2) Une couche de Sortpooling qui trie les caractéristiques des sommets dans l'ordre défini au préalable et qui unifie la taille d'entrée du graphe.
3) Une couche convolutive et une couche dense qui lisent la configuration du graphe et font les prédictions.

Ce qui s'apparentera à l’illustration ci-dessous.

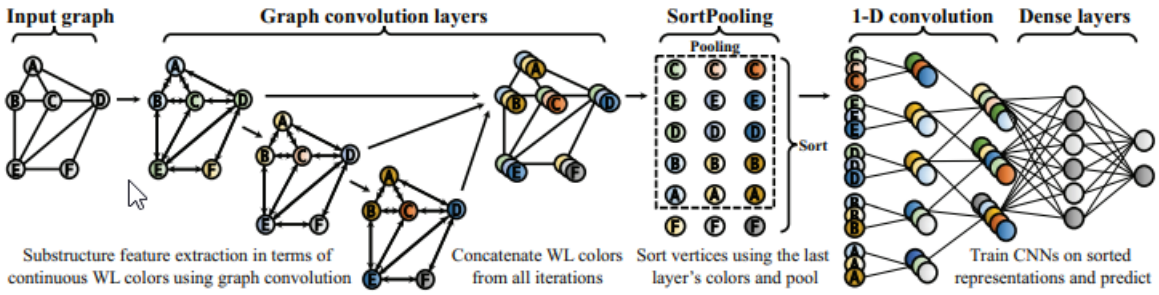


Illustration tirée de la publication "An End-to-End Deep Learning Architecture for Graph Classification"

Étape du traitement

Pré-traitement

Comme mentionné précédemment, on peut utiliser la matrice d'information X où chaque ligne correspondant aux features vector d'un sommet pour les graphes disposant ses étiquettes. Cette matrice X peut être encodé en utilisant un one-hot encoding mais elle peut aussi s'utiliser telle quelle (avec ses attributs potentiellement multidimensionnel). Dans le cas d'un ensemble de graphe non étiquetés, la matrice X peut être définie comme un vecteur colonne des degrés de chaque nœud avec ou sans boucle. Pour les matrices d'adjacence et de degrés, un pré-traitement consisterait à les construire. Ici, nous utiliserons des datasets déjà ayant déjà ces matrices exploitables (cf à la partie chargement des données pour plus de détails dans notre cas).

Chaque colonne de X est appelée "Feature channel".

La couche Graph convolutional layer

Cette couche est la plus importante car elle joue le rôle d'encodeur-décodeur. Elle est constituée de plusieurs Graph convolutional network.

La formalisation mathématique de ses réseaux est la suivante :

$$Z = f(D^{-1}\tilde{A}XW)$$

où D est la matrice diagonale des degrés, $\tilde{A} = A + I$ la matrice d'adjacence additionné à la matrice des auto boucles, $X \in \mathbb{R}^{n \times c}$ la matrice des attributs des noeuds, $W \in \mathbb{R}^{c \times c'}$ la matrice des paramètres du réseaux , f la fonction d'activation non linéaire et $Z \in \mathbb{R}^{n \times c'}$.

Ce mécanisme peut être décomposé en 4 étapes :

1) Une transformation linéaire des features (transformé chaque channel feature dans un channel feature pour les suivantes couches) -> XW.
2) Une propagation des informations des noeuds à leur voisins et à eux-même -> $\tilde{A}XW$
3) Une préservation de taille fixe de feature après chaque convolution -> $D^{-1}\tilde{A}XW$
4) Une convergence via la fonction d'activation non linéaire -> $f(D^{-1}\tilde{A}XW)$

Afin d'extraire plusieurs sous-structures caractérisantes à différentes échelle et d'étendre notre champ réceptif, on va empiler plusieurs GCN. La formulation mathématique de cette empilement est la suivante :

$$Z^{t+1} = f(D^{-1}\tilde{A}Z^{t}W^{t})$$

où $Z^{0} = X, Z^{t} \in \mathbb{R}^{n \times c}$ est la sortie au $t^{ème}$ GCN

Après les multiples couches, on va construire notre readout $Z^{1:h}$ en concaténant les embeddings de chaque couche GCN puis en appliquant un sortpooling.

<u>remarque :</u> Ce mécanisme s'apparente au modèle à noyau de Weisfeiler-Lehman pour extraire les sous-structures caractéristique du graphe suivi d'une propagation kernel pour mesurer efficacement la similarité entre graphe à partir des caractéristiques de ses sous-structures afin de pouvoir déterminer quelle sous-structure est pertinente pour la comparaison.

La couche Sortpooling

Cette couche est l'originalité de cette méthode. Elle suit une logique de tri lexicographique.

Cette couche va trier les sommets en fonction de leur rôle structurel dans le graphe. Pour capter le rôle structurel des sommets, on va utiliser ce qui s'apparente à leur signature WL color par analogie au noyau WL.

La sortie des GCN est un tensor rassemblant l'ensemble des informations des nœuds sur les features channels transformé par la propagation des connaissances du nœud dans la configuration du graphe. A partir de cette structure, l'ordre des nœuds est calculé en triant les sommets par rapport à la dernière variable "descriptive" de cette sortie.

Ensuite, la fonction sortpooling va permettre la taille du tensor de sortie en additionnant ces caractéristiques de trie. L'autre bénéfice de cette couche est de permettre de back

propager l'erreur au précedente couche.

Cette couche se finalise par un flattering la sortie $Z^{1:h}$ afin de jouer un réseau convolutif suivi d'une couche dense.

Les autres couche

Les couches convolutive et dense de ce modèle ont une fonction traditionnelle. La couche convolutive permet d'apprendre les patterns locaux sur les séquences de nœuds soit la configuration du graphe.

Quant à elle, la couche dense est utilisée pour réaliser les prédictions.

Résumé

En d'autre mot, cette méthodes permet d'extraire des éléments descriptifs pertinents appris sur les configurations structurelles des graphes dans un ensemble de graphe (critères isomorphique, topologie…) et d'utiliser ses critères pour projeter chaque graphe dans ce langage commun afin de permettant leur comparaison et ainsi les classifier.

Implémentation

Configuration et librairies utilisées

J'ai implémenté mon code en utilisant le langage python avec la bibliothèque pytorch (pytorch = 1.10.2).

Mon code est divisé en 3 fichiers : *train.py* contenant le programme principale, *model.py* contenant le code du modèle, *utils.py* contenant des fonctions utilisées dans le programme. Le code sera joint au dossier.

Chargement des graphs

Les ensembles de graphes. utilisés sont ceux proposés par TU https://chrsmrrs.github.io/datasets/docs/datasets/

Les dataset utilisées dans l'implémentation utilisent un format proposé par torch_geometric permettant d'accéder à des équivalences des matrices d'adjacence et de degrés. Les informations présentes dans chacun des datasets sont composés des informations ci-dessous :

- data.x : Node feature matrix with shape [num_nodes, num_node_features]
- data.edge_index: Graph connectivity in COO format with shape [2, num_edges] and type torch.long
- data.edge_attr: Edge feature matrix with shape [num_edges, num_edge_features]
- data.y: Target to train against (may have arbitrary shape), e.g., node-level targets of shape [num_nodes, *] or graph-level targets of shape [1, *]
- data.pos: Node position matrix with shape [num_nodes, num_dimensions]

On verra dans les résultats des représentations de ces graphes et des descriptions de ces ensemble.

Comme évoqué dans la partie précédente, une transformation pour obtenir un vecteur colonne de degrés nœud normalisé est nécessaire dans le cas de graphes non labellisés (ici le dataset IMDB-Binary).

```
dataset =
TUDataset(root='C:/Users/Utilisateur/Documents/GitHub/DGCNN/data',
name=DATASET_NAME, transform = TransFormGetDegree())
```

```
class TransFormGetDegree(object):

   def __call__(self, data):
       col, nodes, x = data.edge_index[1], data.num_nodes, data.x
       deg = degree(col, nodes)
       deg = deg / deg.max()
       if data.x is None:
           data.x = deg.view(-1, 1)
       return data
```

Le modèle

Pour le modèle, nous utiliserons 4 GCN de 32 hidden layers et la fonction d'activation tangente hyperbolique et 2 CNN.

model.py

```
class Dgcnn(torch.nn.Module):
   def __init__(self, num_features, num_classes):
       super(Dgcnn, self).__init__()
       self.conv1 = GCNConv(num_features, 32)
       self.conv2 = GCNConv(32, 32)
       self.conv3 = GCNConv(32, 32)
       self.conv4 = GCNConv(32, 1)
       self.conv5 = Conv1d(1, 16, 97, 97)
       self.conv6 = Conv1d(16, 32, 5, 1)
       self.pool = MaxPool1d(2, 2)
       self.classifier_1 = Linear(352, 128)
       self.drop_out = Dropout(0.5)
```

```
        self.classifier_2 = Linear(128, num_classes)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.batch
        edge_index, _ = remove_self_loops(edge_index)
        x_1 = torch.tanh(self.conv1(x, edge_index))
        x_2 = torch.tanh(self.conv2(x_1, edge_index))
        x_3 = torch.tanh(self.conv3(x_2, edge_index))
        x_4 = torch.tanh(self.conv4(x_3, edge_index))
        x = torch.cat([x_1, x_2, x_3, x_4], dim=-1)
        x = global_sort_pool(x, batch, k=30)
        x = x.view(x.size(0), 1, x.size(-1))
        x = self.relu(self.conv5(x))
        x = self.pool(x)
        x = self.relu(self.conv6(x))
        x = x.view(x.size(0), -1)
        out = self.relu(self.classifier_1(x))
        out = self.drop_out(out)
        classes = F.log_softmax(self.classifier_2(out), dim=-1)
        return classes
```

Entrainement

Pour l'entraînement, nous utilisons des mini-batch de 50 éléments avec un optimiseur Adam et une fonction de perte negative log likelihood.

main.py

```
optimizer = Adam(model.parameters())
loss_criterion = nn.NLLLoss()
print(f"Train start on {NUM_EPOCHS}")
for epoch in range(0, NUM_EPOCHS):
    train_loss = train(train_loader)
    test_acc = test(test_loader)
print(f"Train finished")
print(f"execution finish")
```

functions.py

```
# Fonction permettant l'entrainement du modèle sur un ensemble de données
```

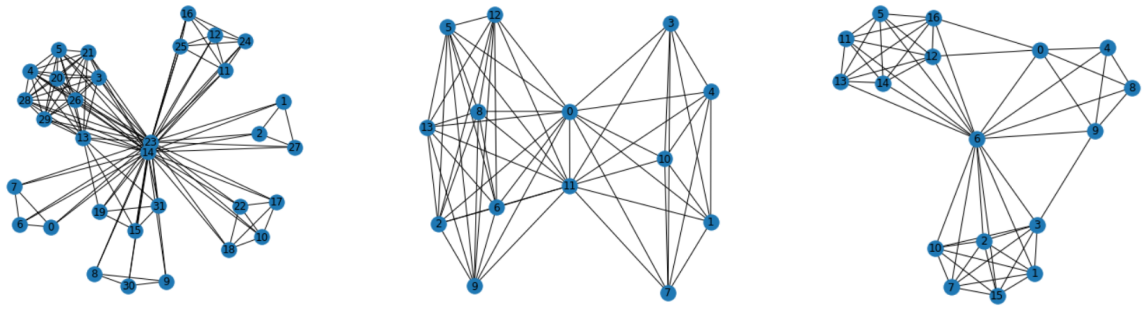
```
dédiées à l'entrainement
def train(loader):
    model.train()
    running_loss = 0
    for data in loader:
        classes = model(data)
        loss = loss_criterion(classes, data.y)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        running_loss += loss.item()
    return running_loss / len(train_loader)

# Fonction permettant d'évaluer le modèle sur un ensemble de données
dédiées aux tests
def test(loader):
    model.eval()
    correct = 0
    for data in loader:
        out = model(data)
        pred = out.argmax(dim=1)
        correct += int((pred == data.y).sum())
    # retourne le pourcentage de graph bien classé
    return correct / len(loader.dataset)
```

Résultats

Description des datasets

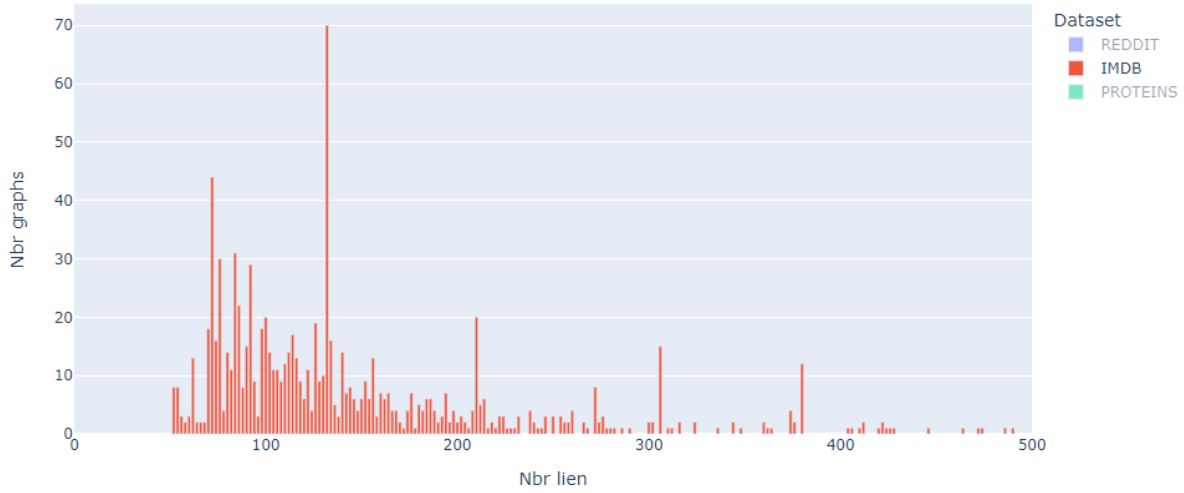
Dans un premier temps, je vais décrire les jeux de données utilisés.

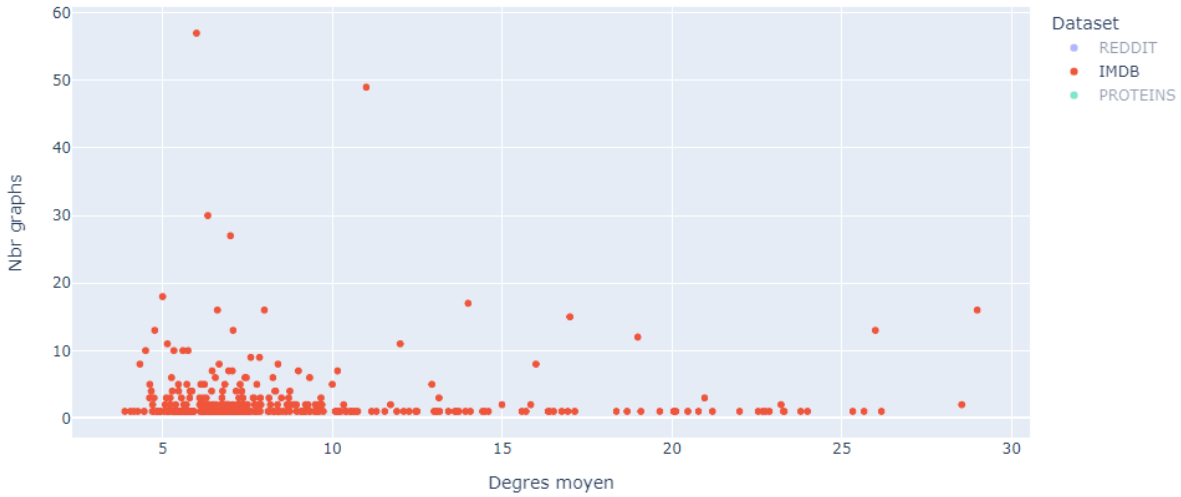
IMDB-BINARY



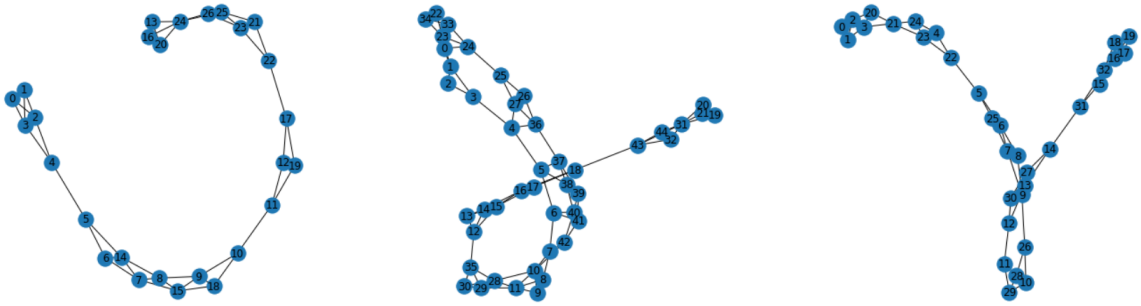
Dataset : IMDB-BINARY
==========
IMDB-BINARY is a movie collaboration dataset that consists of the ego-networks of 1,000 actors/actresses who played roles in movies in IMDB. In each graph, nodes represent actors/actress, and there is an edge between them if they appear in the same movie.
==========
Nombre de graphes: 1000
Nombre d'attributs: 0
Nombre de classes: 2
Nombre de graphes ayant des boucles : 0
Nombre de graphes ayant des noeuds isolés : 0
Nombre de graphes ayant des non-dirigé : 1000


Dataset
REDDIT
IMDB
PROTEINS
Nbr graphs
Nbr noeud
0
20
40
60
80
100
120
140
160
0
50
100
150
200

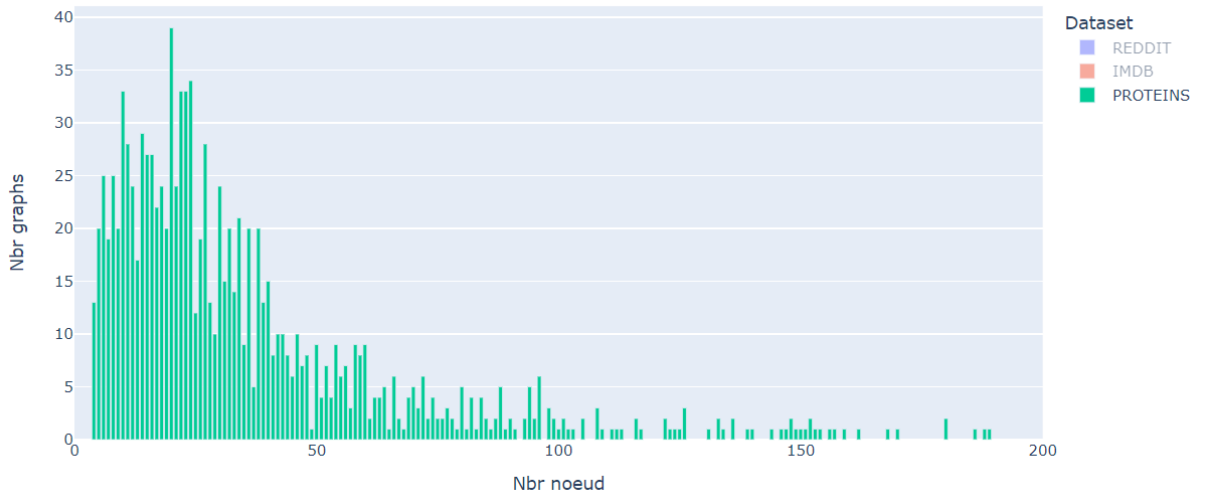

Dataset
REDDIT
IMDB
PROTEINS
Nbr graphs
Nbr lien
0
10
20
30
40
50
60
70
0
100
200
300
400
500

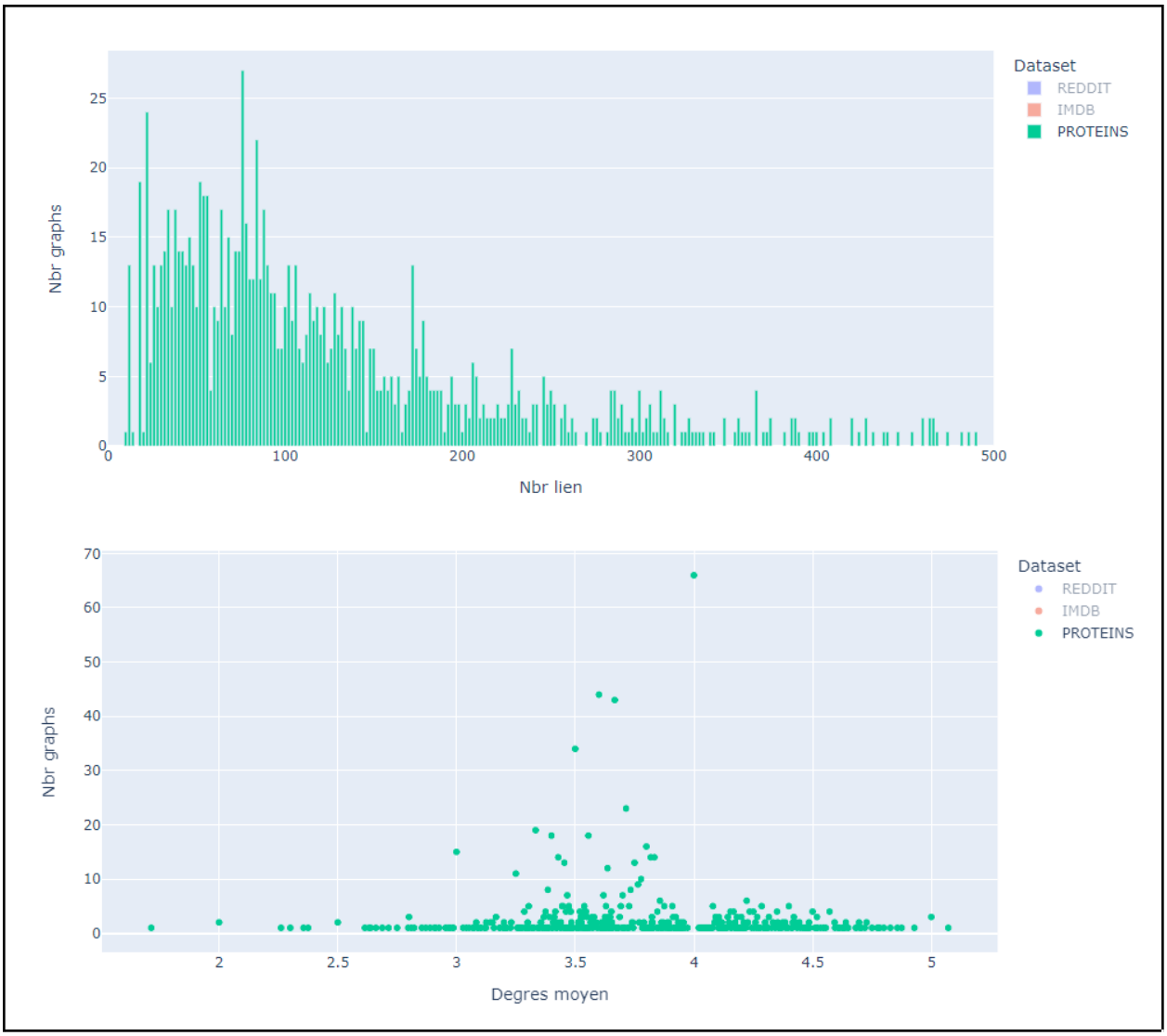

Dataset
REDDIT
IMDB
PROTEINS
Nbr graphs
Degres moyen
0
10
20
30
40
50
60
5
10
15
20
25
30

PROTEINS

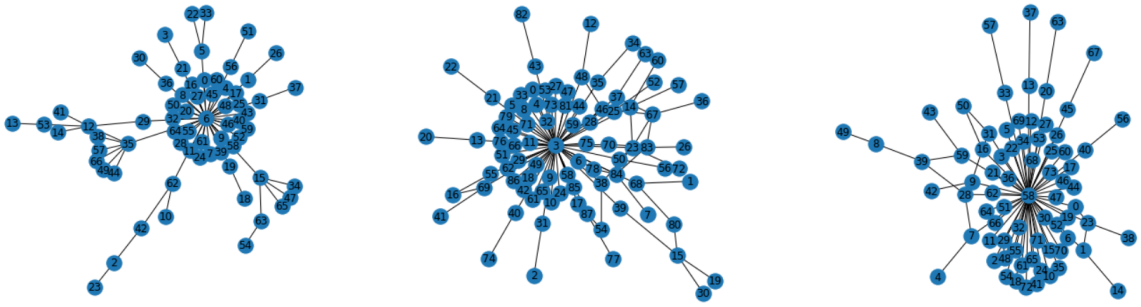


Dataset: PROTEINS
==========
PROTEINS is a dataset of proteins that are classified as enzymes or non-enzymes. Nodes represent the amino acids and two nodes are connected by an edge if they are less than 6 Angstroms apart.
==========
Nombre de graphes: 1113
Nombre d'attributs: 3
Nombre de classes: 2
Nombre de graphes ayant des boucles : 0
Nombre de graphes ayant des noeuds isolés : 5
Nombre de graphes ayant des non-dirigé : 1113

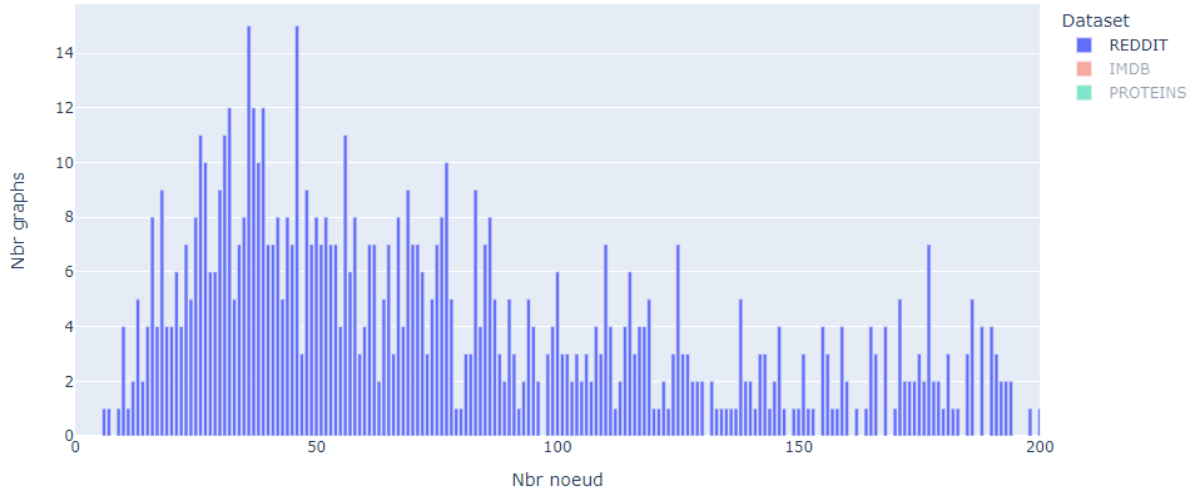


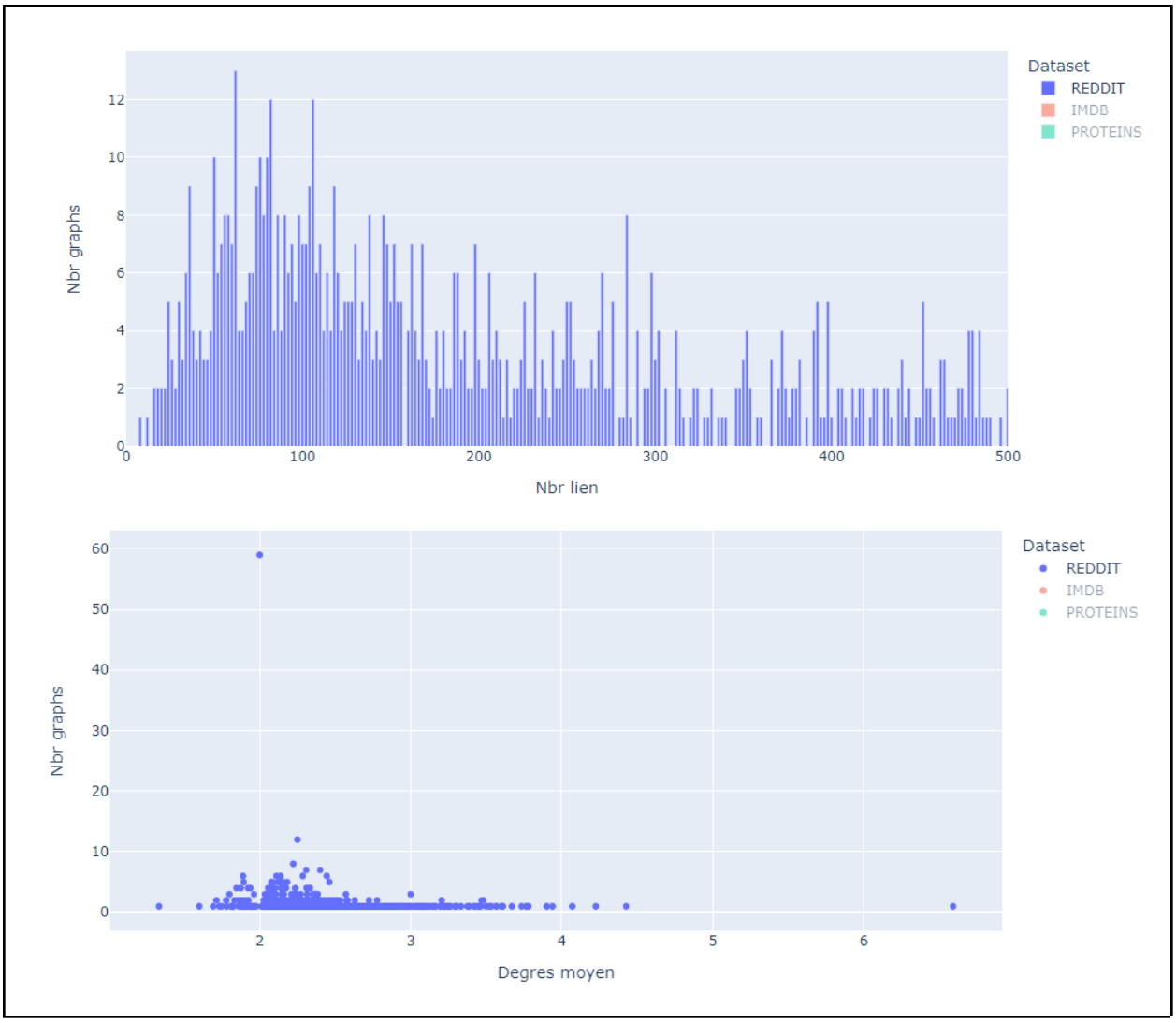

Dataset
REDDIT
IMDB
PROTEINS
Nbr graphs
Nbr lien
0
5
10
15
20
25
0
100
200
300
400
500
Dataset
REDDIT
IMDB
PROTEINS
Nbr graphs
Degres moyen
0
10
20
30
40
50
60
70
2
2.5
3
3.5
4
4.5
5

REDDIT-BINARY		



Dataset: REDDIT-BINARY
==========
REDDIT-BINARY consists of graphs corresponding to online discussions on Reddit. In each graph, nodes represent users, and there is an edge between them if at least one of them respond to the other's comment.
==========
Nombre de graphes: 2000
Nombre d'attributs: 0
Nombre de classes: 2
Nombre de graphes ayant des boucles : 0
Nombre de graphes ayant des noeuds isolés : 21
Nombre de graphes ayant des non-dirigé : 2000




Dataset
REDDIT
IMDB
PROTEINS
Nbr graphs
Nbr lien
0
2
4
6
8
10
12
0
100
200
300
400
500
Dataset
REDDIT
IMDB
PROTEINS
Nbr graphs
Degres moyen
0
10
20
30
40
50
60
2
3
4
5
6

Résultats finales

Dataset	**Accuracy (%)**
IMDB	72
PROTEINS	75
REDDIT	77

La méthode DGCNN fonctionne globalement bien au vu de la diversité des ensembles de graphes.
En se reportant au benchmark https://paperswithcode.com/area/graphs, on observe que cette approche donne d'assez bon résultat sur un large panel d'ensemble de données.
Tout de même, on distingue de meilleures performances lorsque le dataset présente des formes marquées.

Conclusion

Cette approche qu’on pourrait qualifier de classificateur universel de graphes présente d’assez bon résultats avec des avantages intéressants tels que la possibilité d’apprendre sur des graphes de différentes taille et forme, et aussi qu' elle est entrainable tout comme des CNN traditionnels.

En jouant sur les nombres de couches cachées dans les GCN, le nombre de GCN, l'hyper paramètre k de la couche Sort global pooling et d’autres paramètres sur les dernières couches, nous pourrions certainement améliorer sensiblement les performances.

Malgré de tels affinages, je pense qu'une réelle amélioration des performances ne peut être qu’au niveau des couches d’extraction qui pourrait être plus expressive. La couche de pooling pourrait être un sujet d’amélioration

Annexe

READ.txt

Guide d'installation et utilisation du code

SPEC

Operating System	Windows 10 Famille
GPU	GeForce GTX 1650 Ti
CUDA Version	10.2

Step 1 : Installation CUDA

https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html

lib : cuDNN = 7.6.4

Step 2 : Installation Anaconda

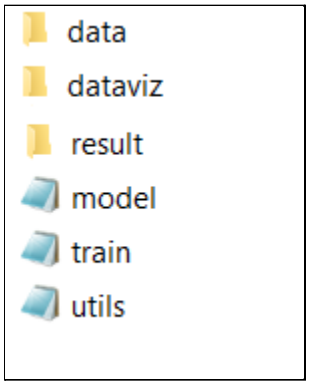
https://docs.anaconda.com/anaconda/install/

Step 3 : Installation environnement virtuel

Jouer les commandes suivantes en séquence dans Anaconda Prompt :

```
conda create -n pytorchgpu
conda activate pytorchgpu
conda install pyg -c pyg
conda install pandas
conda install plotly
conda install numpy
```

Step 4 : Répertoire et manipulation



- le répertoire *data* contient les dataset IMDB, REDDIT et PROTEINS
- le répertoire *result* contient les résultats
- Et *dataviz* contient les graphiques générés à l'exécution du programme + notebook d'analyse des datasets

Comment lancer le programme ?

- Exécuter la ligne de commande suivante :
-

```
python train.py IMDB-BINARY 50 20 NORMAL
```

- Un help est disponible

```
>python train.py -h
usage: DGCNN [-h] DATASET_NAME BATCH_SIZE NUM_EPOCHS MODE
========================
   Dgcnn : graph classifier
   ========================
   Launching Method
   ----------------------
   TRAIN : Launch this prog training mode need a valoriszation of different arguments.
   NORMAL : Launch normal mode need having runnning previous the training on the
three datasets (REDDIT-BINARY, PROTEINS, IMDB-BINARY)

positional arguments:
  DATASET_NAME  Dataset name (REDDIT-BINARY, PROTEINS, IMDB-BINARY)
  BATCH_SIZE    Batch size
  NUM_EPOCHS    Numbers epoch
  MODE          Executing mode TRAIN/NORMAL
optional arguments:
  -h, --help    show this help message and exit
ex : python train.py IMDB-BINARY 50 20 TRAIN
```