Graph Neural Networks

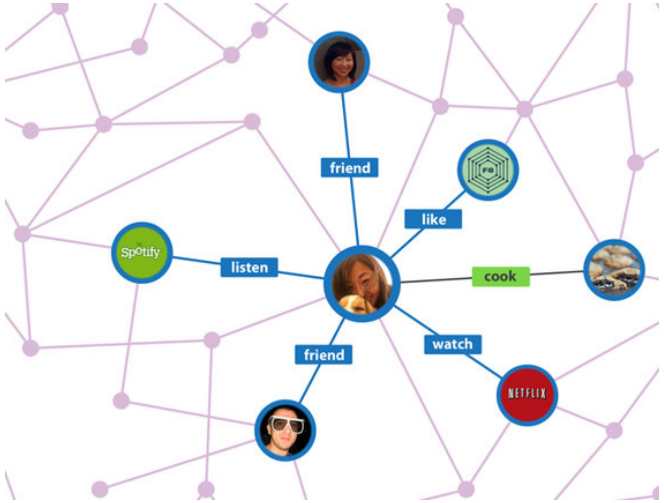
Jian Tang

HEC Montreal

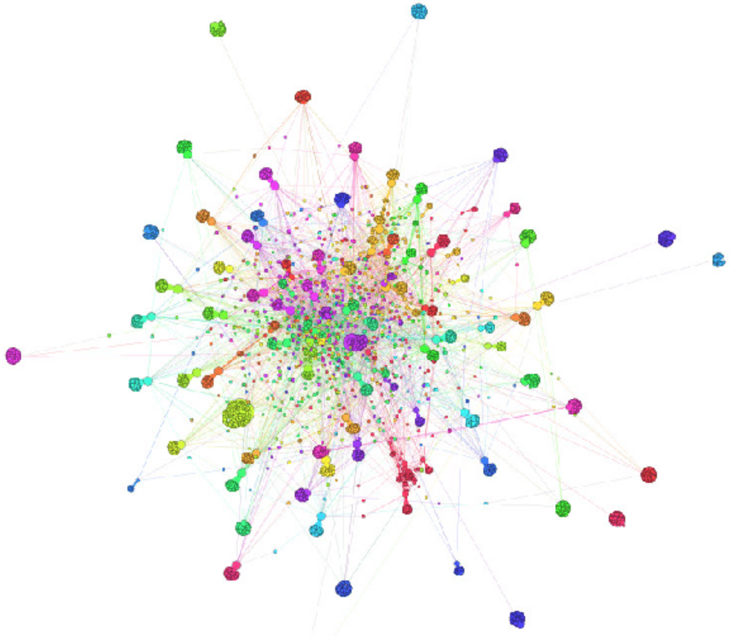
Mila-Quebec AI Institute

Email: jian.tang@hec.ca

Réseaux sociaux



Facebook



Twitter

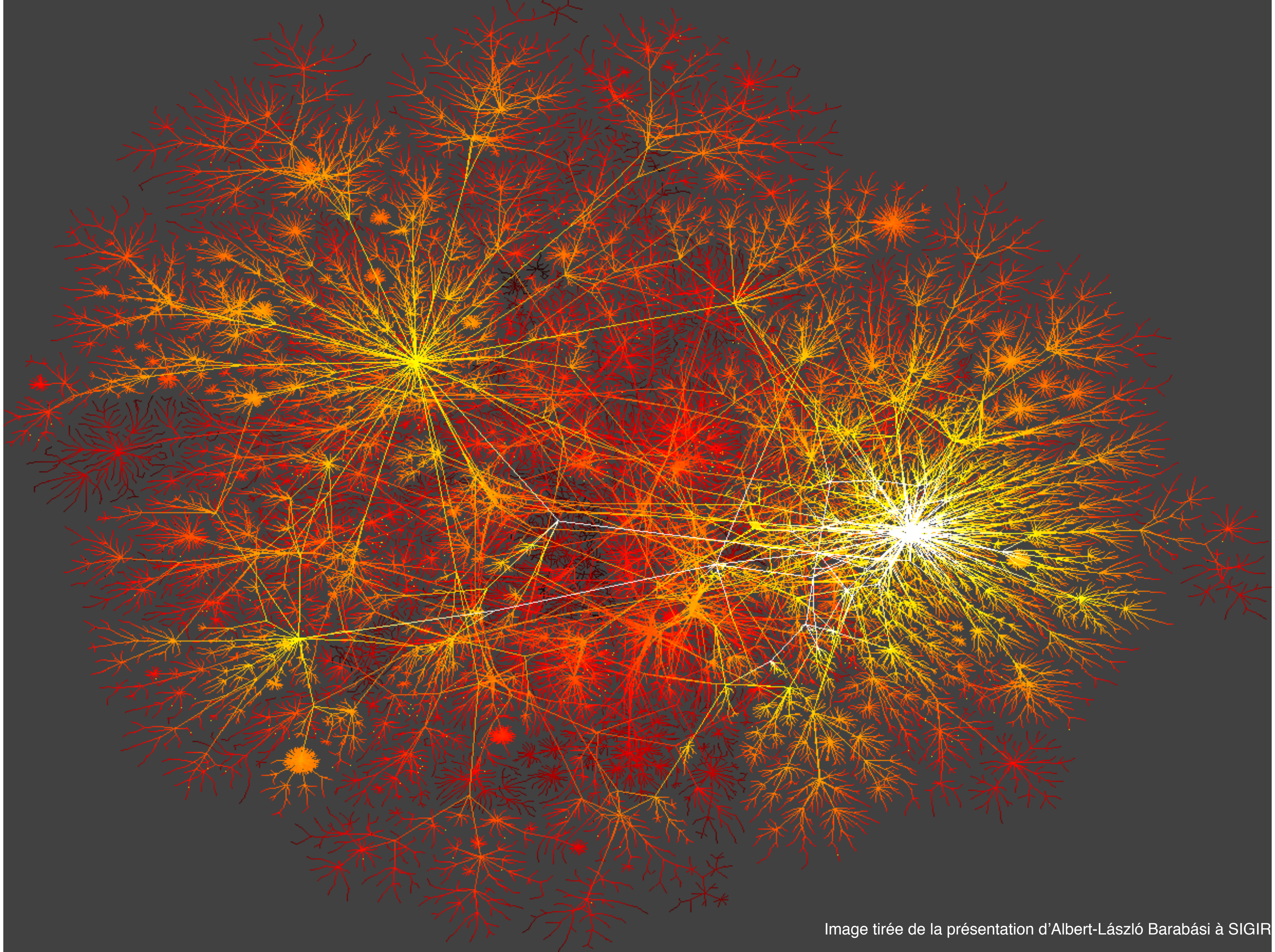
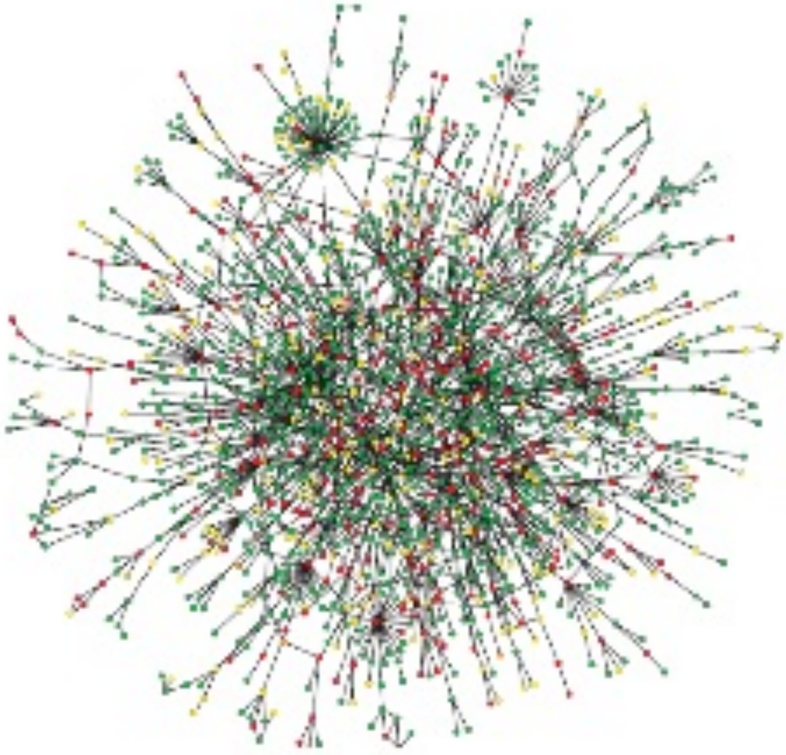
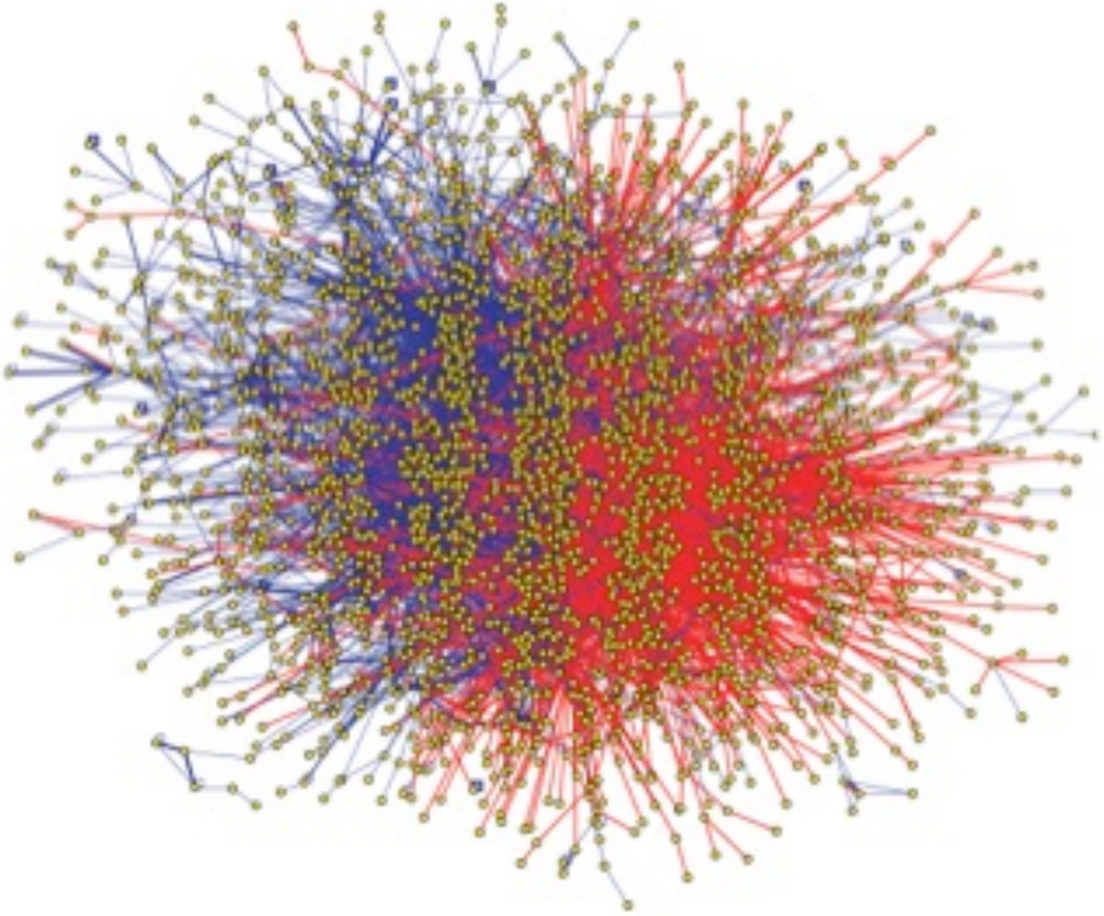


Image tirée de la présentation d’Albert-László Barabási à SIGIR

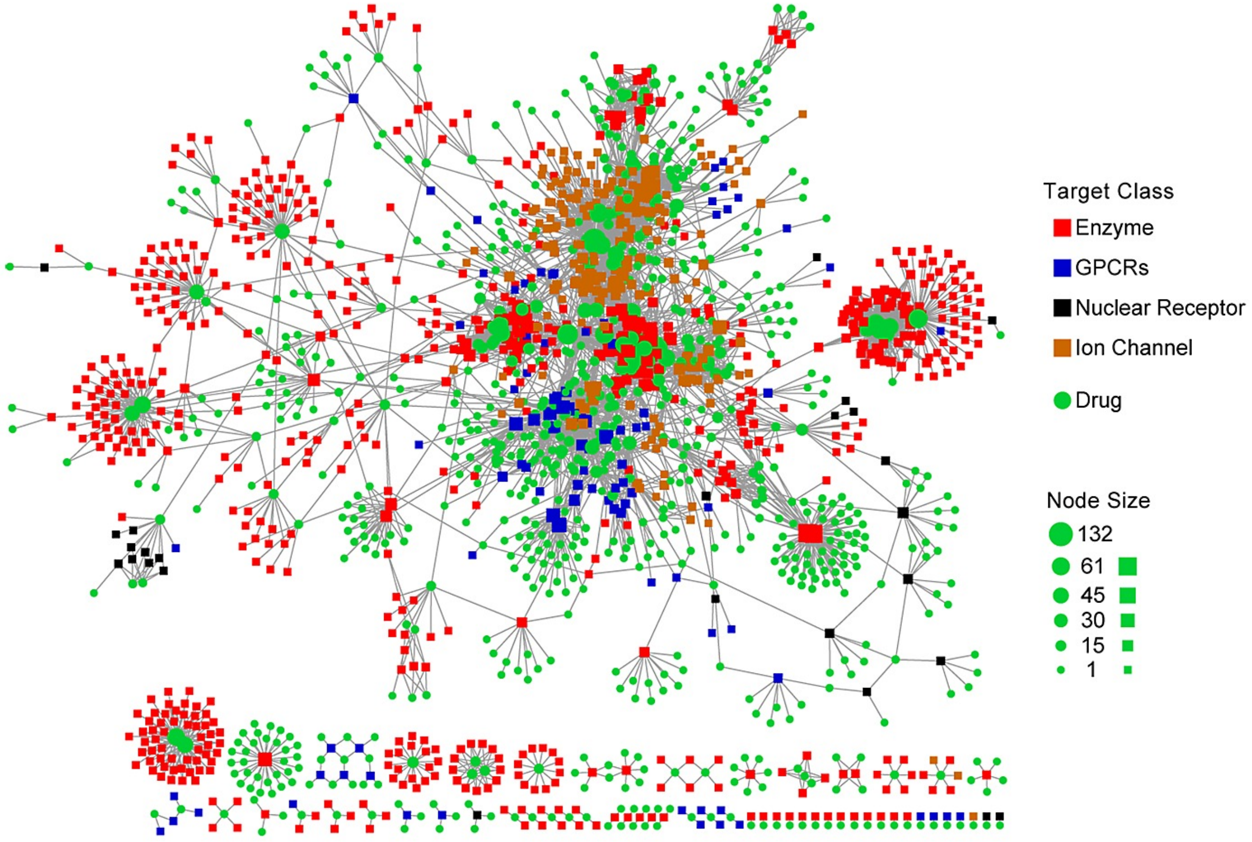
Graphe des interactions entre protéines





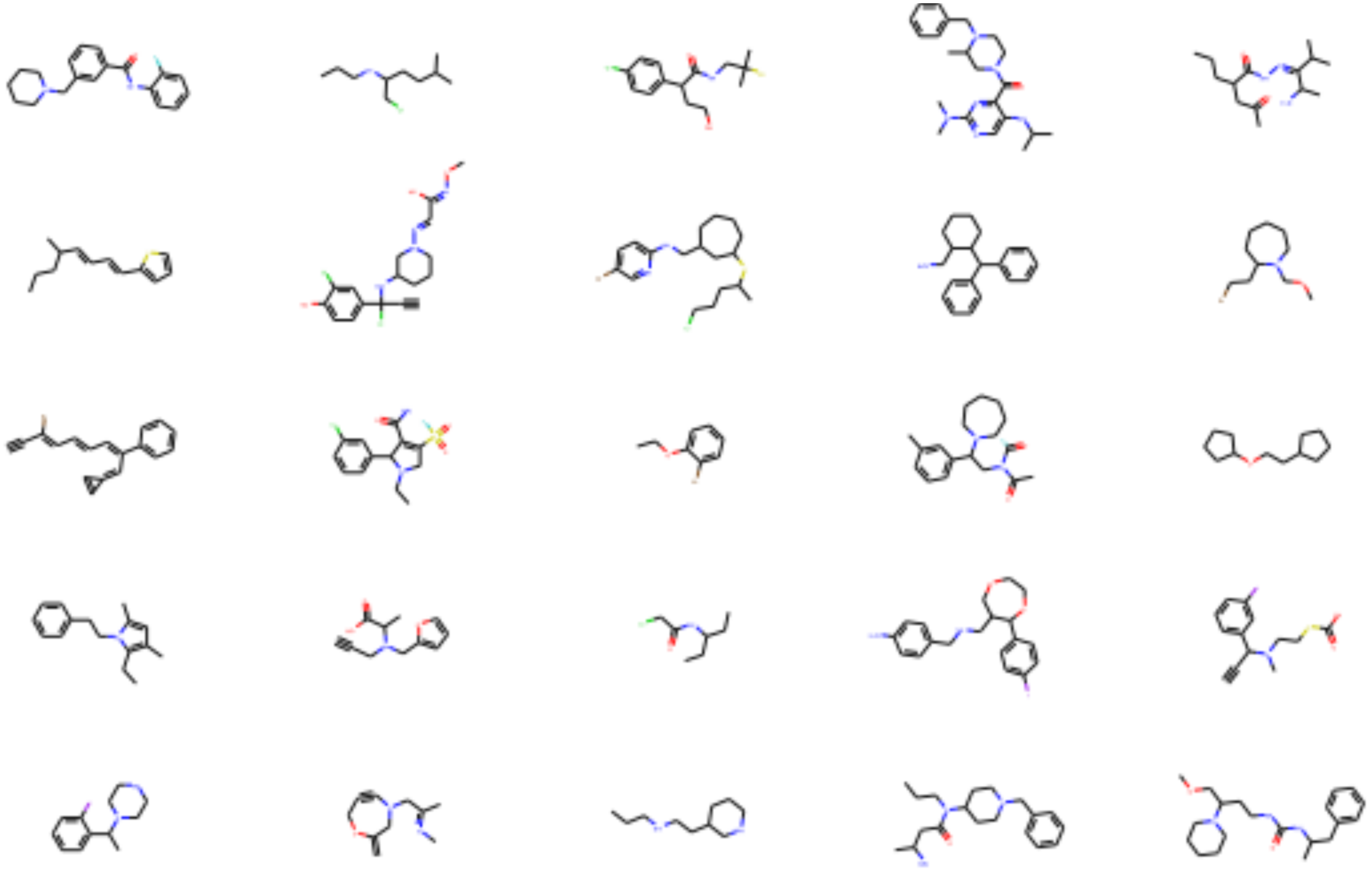
https://www.ebi.ac.uk/training/online/course/network-analysis-protein-interaction-data-introduction/protein-protein-interaction-networks

Graphe des interactions entre protéines et médicaments



https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002503

Molécules

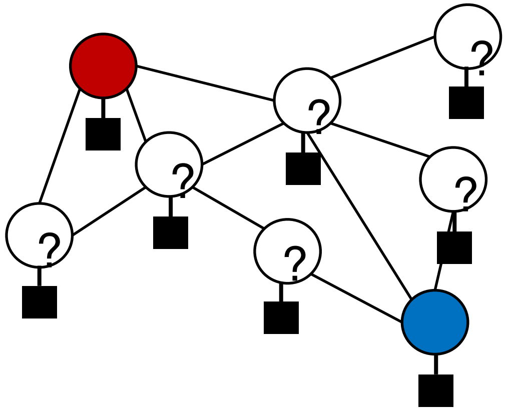


Quelques applications des graphes

- Recommandations d’amis sur les réseaux sociaux
- Prédiction de l’allégeance politique d’un utilisateur sur Facebook
- Prédiction de la diffusion d’informations sur les réseaux sociaux
- Prédiction du rôle des protéines dans un graphe des interactions entre protéines
- Prédiction des propriétés chimiques d’une molécule
- Etc.
- Ces applications nécessitent une bonne représentation du graphe!!

Apprentissage de graphes (semi) supervisé

- Au lieu de préserver la structure du graphe, des tâches supervisées sont données:
  - Classification des nœuds,
  - Présence d'une arête.
- Apprendre les représentations des nœuds pour une tâche spécifique

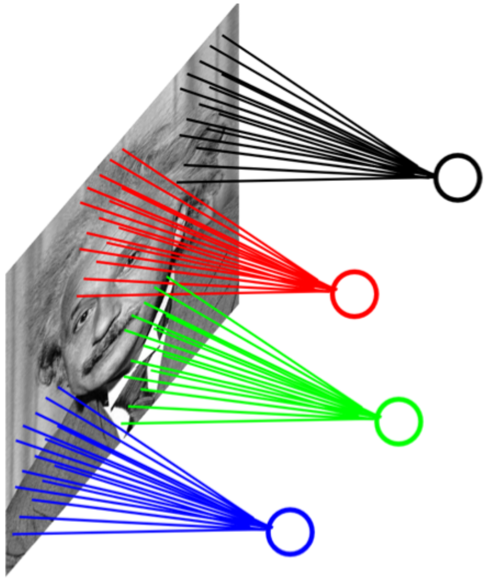


○ Type du noeud

■ Attribut du noeud

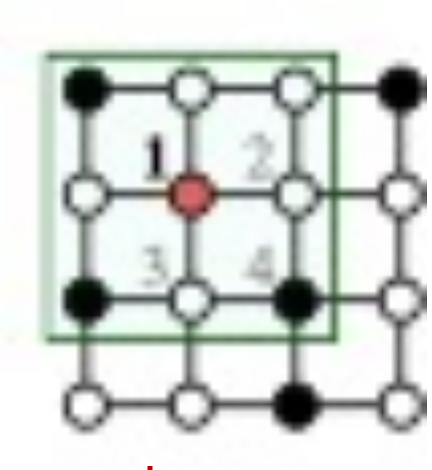
Rappel: Réseaux convolutifs (CNN) pour apprentissage de représentation

- Filtres convolutifs
  - Permet la reconnaissance d'attributs locaux.
  - Différents attributs peuvent être appris en fonction de leur emplacement sur l'image.

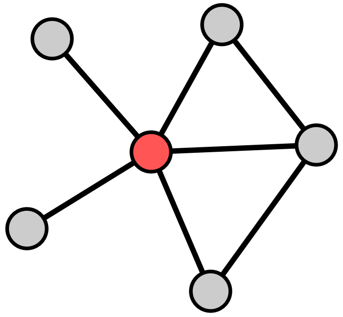


Champ récepteur local/ Local Receptive Field pour les graphes

- Comment peut-on définir des local receptive fields pour des graphes?
  - Sous-graphes locaux
- Par contre, il n’y pas d’ordre entre les voisins:
  - Avec une image, les voisins peuvent être ordonnés.



Imag



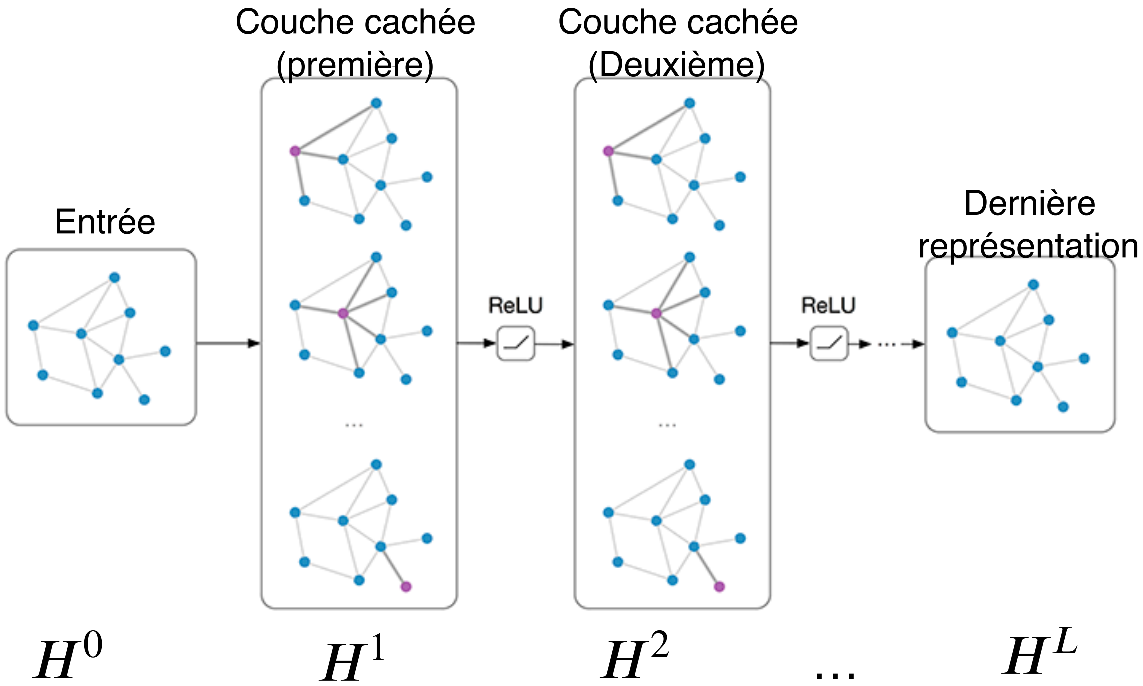
Graph

Formalisme

- Soit le graphe $G = (V,\ E)$, où V est l’ensemble des noeuds et E est l’ensemble des arêtes.
- Deux types d’informations sont présentées:
  - Un vecteur d’attribut $\boldsymbol{x}_i \in R^D$ pour chaque noeud v_i. L’ensemble des attributs pour V peut être représenté dans une matrice des attributs X de dimension $N\ \times\ D$.
  - La structure du graphe, généralement définie sous la forme d’une matrice adjacente A où A_{ij} est le poids associé à l’arête (i, j).
- But: obtenir une représentation des noeuds, définie par H (de dimension $N\ \times\ F$, où F est la dimension de chaque représentation).

Réseaux de neurones de graphe (formalisme)

- Réseaux de neurones de graphes (à plusieurs couches):
  - H^0 = X, la matrice des attributs des noeuds
  - De façon itérative, mettre à jour la représentation des noeuds
- La $k$ième couche cachée du réseau de neurones est la $k$ième représentation des noeuds, laquelle est symbolisée par H^k.
- Soit H^L la dernière représentation:
  - Peut être utilisée pour tâches spécifiques (classification de noeuds)



Entrée

Couche cachée (première)

Couche cachée (Deuxième)

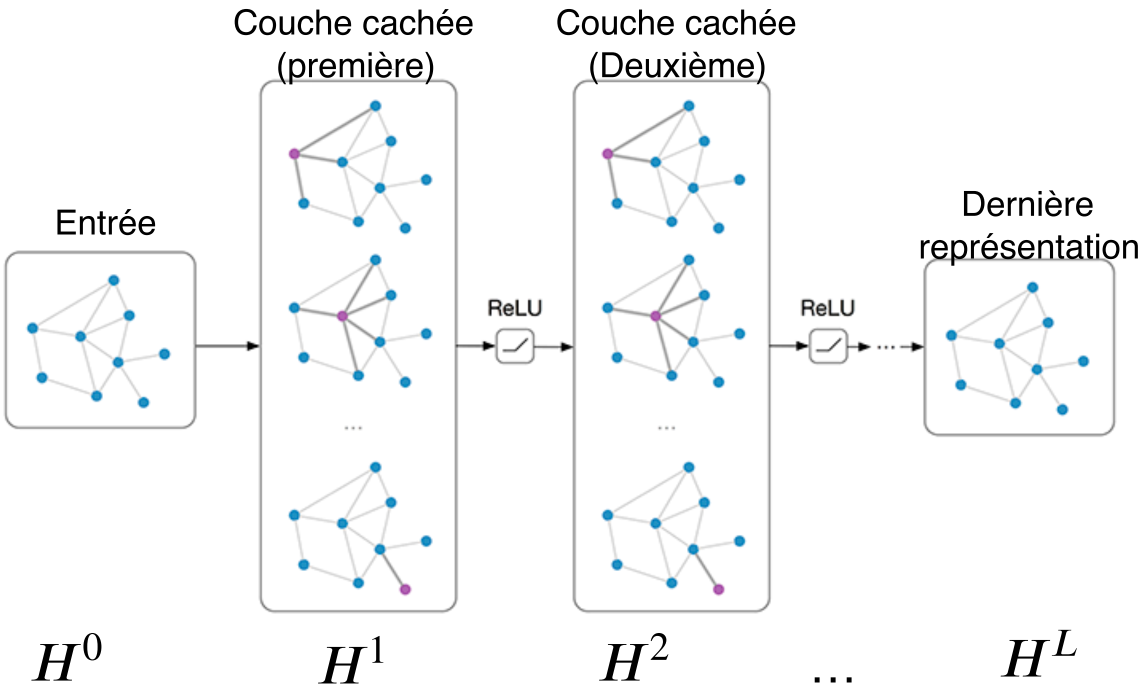
Dernière représentation

H^0 H^1 H^2 ... H^L

Apprentissage supervisé

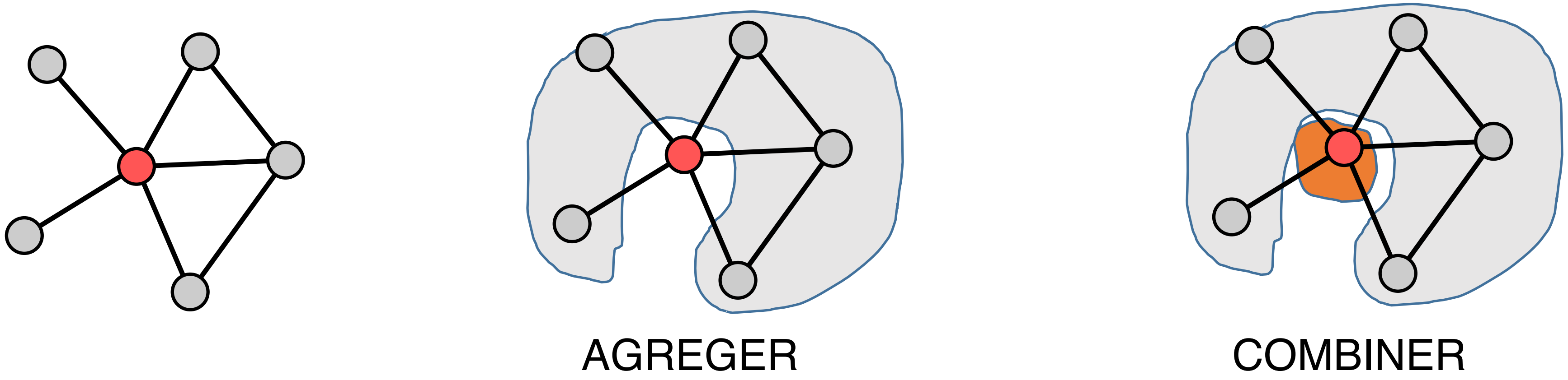
- Apprentissage d'un classificateur, f, à l'aide de la représentation finale H^L.
- Fonction de perte est de la forme:

$$O = \sum_{i \in \text{ exemples libellés}} loss\left(f(H_i^L), y_i\right)$$



Comment mettre à jour la représentation des noeuds?

- Pour chaque couche d’un GNN et pour chaque noeud:
    - AGREGER l’information associée aux voisins d’un noeud,
    - COMBINER cette information à celle du noeud d’intérêt.



$\mathrm{a}_{\mathrm{v}}^{k} = \mathbf{AGGREGATE}^{\mathbf{k}}(\{\mathrm{h}_{\mathrm{u}}^{k-1} : u \in N(v)\})$ $\mathrm{h}_{\mathrm{v}}^{k} = \mathbf{COMBINE}^{\mathbf{k}}(\mathrm{h}_{\mathrm{v}}^{k-1},\ a_{v}^{k})$

Réseaux convolutifs de graphes**

**Kipf et al. 2017. Semi-supervised Classification with Graph Convolutional Networks.

Réseaux convolutifs de graphes

- Soit A, la matrice adjacente
- On pose:

$$\hat{A} = A + I$$

$$\mathrm{h}_{\mathrm{i}}^{k} = \sigma(\sum_{j \in \{N(i) \cup i\}} \frac{\hat{a}_{ij}}{\sqrt{d_i d_j}} \mathrm{W}^{\mathrm{k}} \mathrm{h}_{\mathrm{j}}^{k-1})$$

$$\mathrm{h}_{\mathrm{i}}^{k} = \sigma(\sum_{j \in N(i)} \frac{a_{ij}}{\sqrt{d_i d_j}} \mathrm{W}^{\mathrm{k}} \mathrm{h}_{\mathrm{j}}^{k-1} + \frac{1}{d_i} \mathrm{W}^{\mathrm{k}} \mathrm{h}_{i}^{k-1})$$

d_i: degré du noeud i (pour $\hat{\mathbf{A}}$)

W^{k}: matrice des poids associée à la couche k

Graphe de computation

- Deux couches de GCN

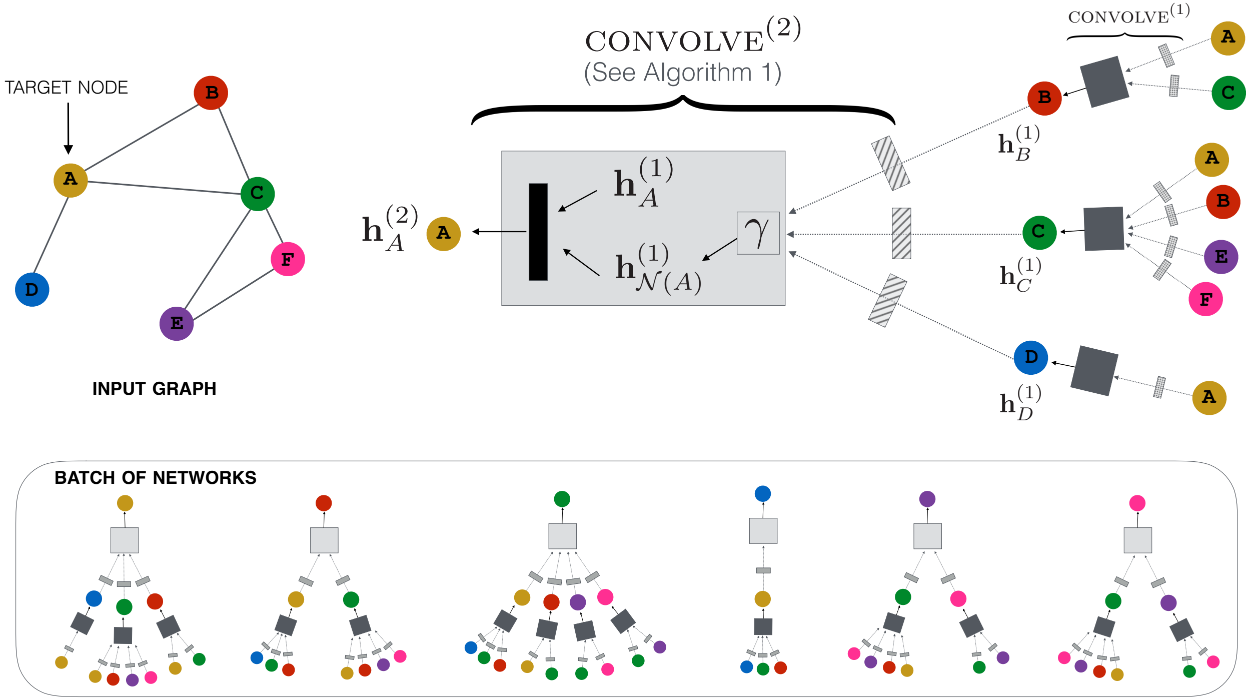


Figure from Ying et al. 2018

Images tirées de Ying et al. 201

Peut-on changer les poids des arêtes?

- Pour les GCN, l’influence d’un noeud j sur le noeud i est déterminée en fonction du poids de leur arête (i,j) de même que de leur degré respectif:

$$\frac{a_{ij}}{\sqrt{d_i d_j}}$$

- Par contre,
  - Les arêtes peuvent contenir beaucoup de bruit
  - Peut ne pas être optimal pour certaines tâches

Graph Attention Networks (Veličković et al. 2017)

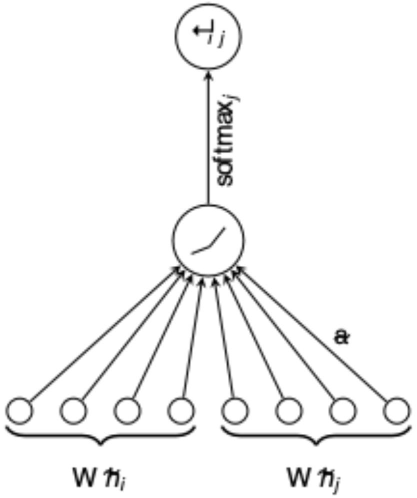
Graph Attention Networks (GAT)

- Un mécanisme d'attention est utilisé afin d'apprendre les poids des arêtes
  - Requête: noeud actuel
  - Mémoire: voisins (incluant le noeud actuel).
- L'attention entre les noeuds i et j se calcule ainsi:

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})}.$$

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$



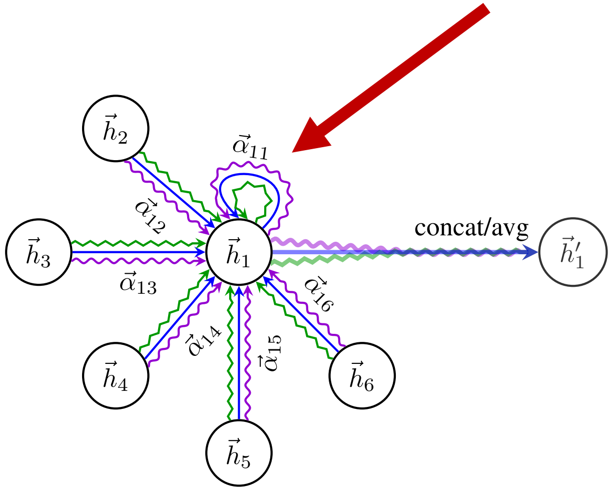
||: concaténation vectorielle

Graph Attention Networks (GAT)

- Agrège l'information du voisinage à l'aide de l'attention:

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \vec{h}_j\right)$$

- Notez que chaque nœud peut se connecter à lui-même :

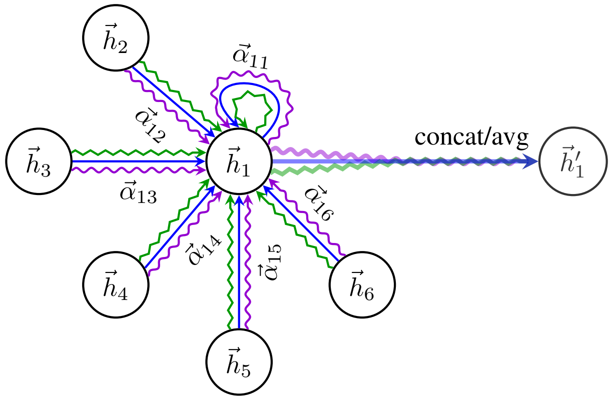


Attentions multiples (Multi-head attention)

- De façon analogue à l'attention multiple dans les modèles de transformers, l'attention multiple peut être mise à profit pour l'apprentissage de graphe.
- Cette représentation peut concatener, ou simplement faire une moyenne, des différents mécanismes d'attention.

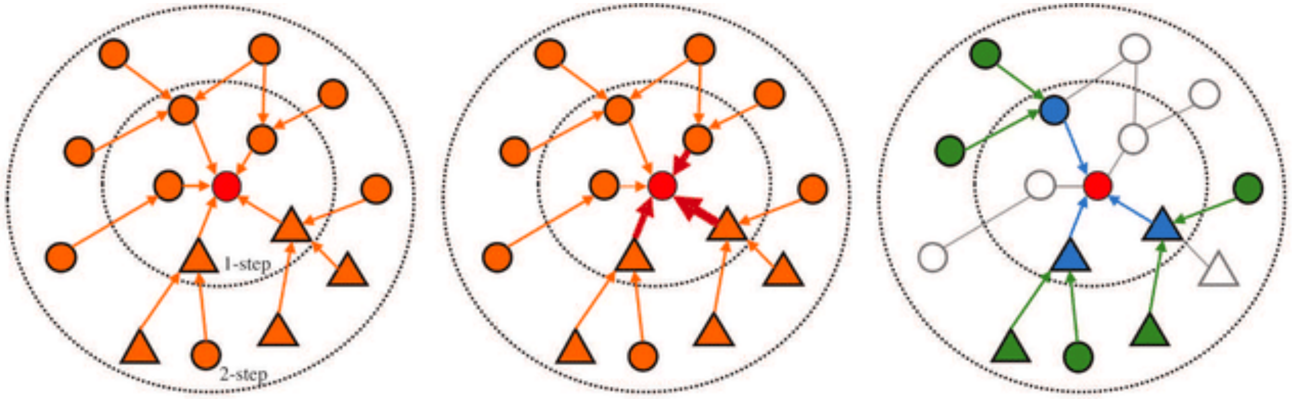
$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$



Quelques problèmes (en pratique)

- Certains noeuds possèdent beaucoup de voisins
- Randomly sample a fixed number of neighbors in each iteration of SGD (Hamilton et al. 2017).



(a) Normal Conv Aggregator (b) Attention Aggregator (c) GraphSAGE Aggregator

On peut échantillonner de façon aléatoire un nombre fixe de voisins pour chaque itération.

Image tirée de Wang et al. (201

Réseaux de neurones avec propagation de message

Gilmer et al. (2017). Neural Message Passing for Quantum Chemistry.

Réseaux de neurones avec propagation du message (MPNN)

- Tout graphe de réseaux de neurones peut être formalisé à l’aide du concept de propagation de message neural (neural message passing)
  - Le message (sous forme de vecteurs) est propagé de façon itérative à travers les noeuds du graphe

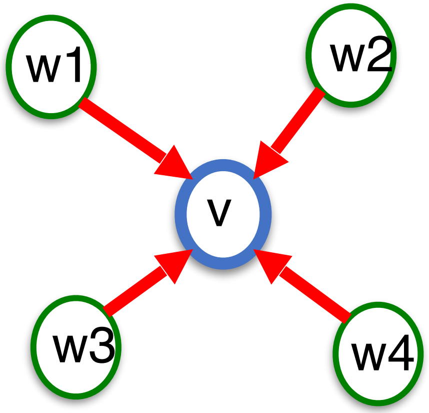
- Deux fonctions
  - Fonction de message
  - Fonction de la mise à jour du noeud

Gilmer et al. (2017). Neural Message Passing for Quantum Chemistry.

Phase de propagation

AGREGER: $$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

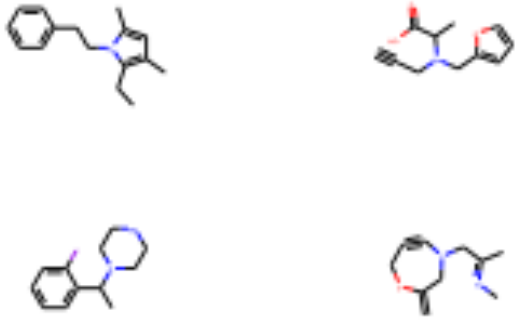
COMBINER: $$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$



M_t : message function

U_t : vertex update function

Comment procéder pour apprendre une représentation du graphe complète?



- Apprentissage de représentation pour un graphe
  - Afin de prédire les propriétés d'une molécule

- Ajouter une fonction de lecture R, laquelle considère la dernière représentation

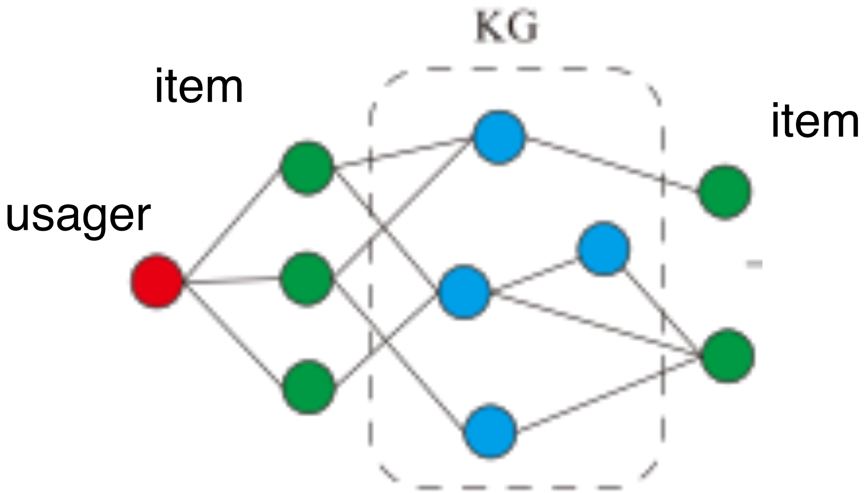
$$\hat{y} = R(\{h_v^T \mid v \in G\})$$

R : fonction de lecture

- $\hat{y}$ est la représentation complète du graphe
- R peut être une fonction très simple (somme, moyenne, etc…)

Applications
Système de recommandations**

- Prédire les items les plus pertinents pour un utilisateur
  - Graphe usager-item ou encore item-item



**Qu et al. An End-to-End Neighborhood-based Interaction Model for Knowledge-enhanced Recommendation.

Applications
Compréhension du langage naturel (NLP)

- Étiquetage de rôles sémantiques (*Semantic Role Labeling*)
  - Encode les phrases à l'aide de GCN

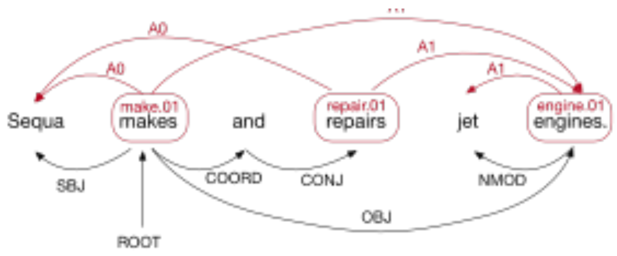


Figure 1: An example sentence annotated with semantic (top) and syntactic dependencies (bottom).

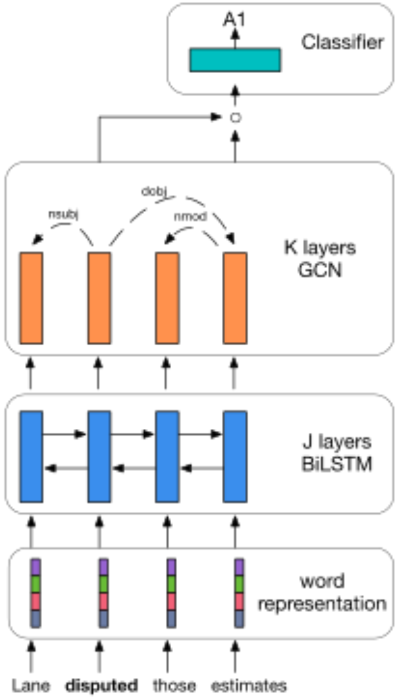
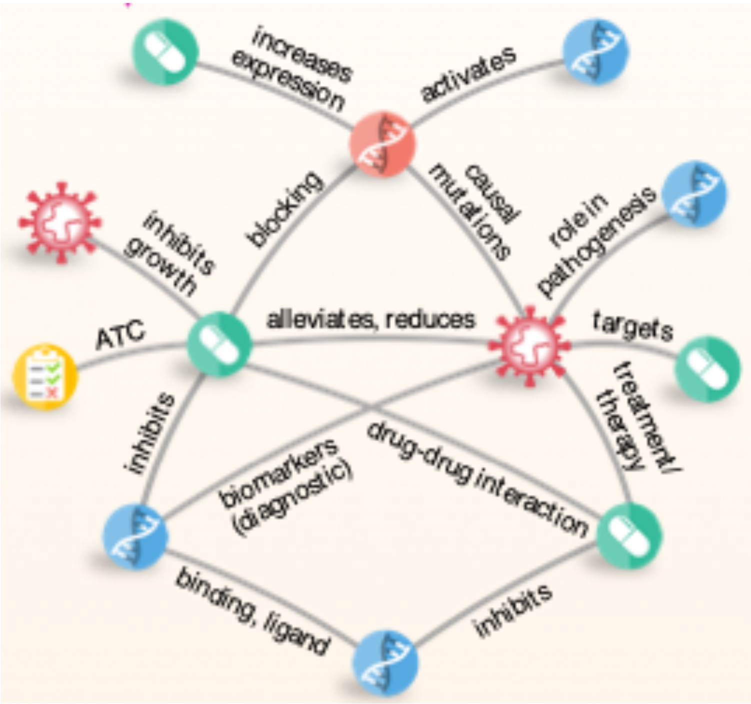
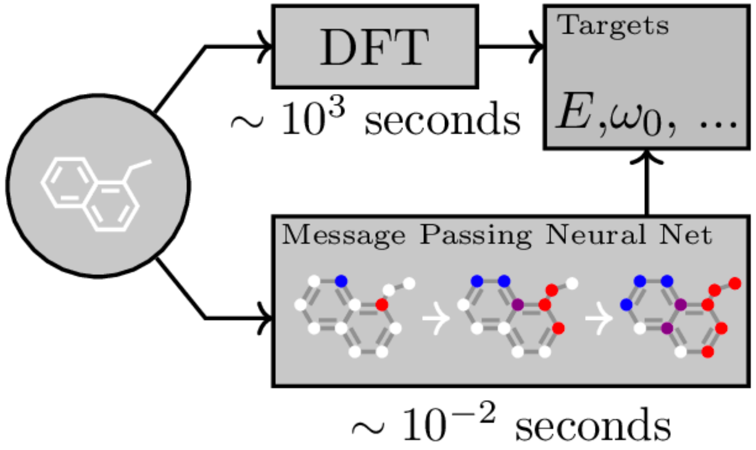


Image tirée de ??

Applications
Découverte de médicaments

- Réorientation de médicaments
    - Graphe protéines - médicaments - maladie
- Prédiction des propriétés d'une molécule

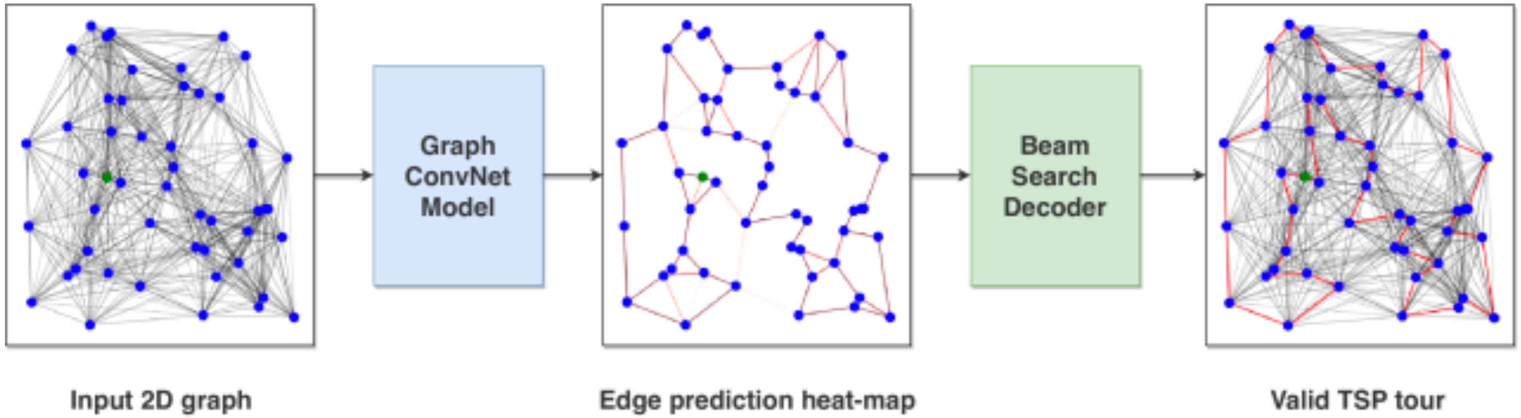




Images tirées de Zeng et al. 20

Applications
Optimisation combinatoire

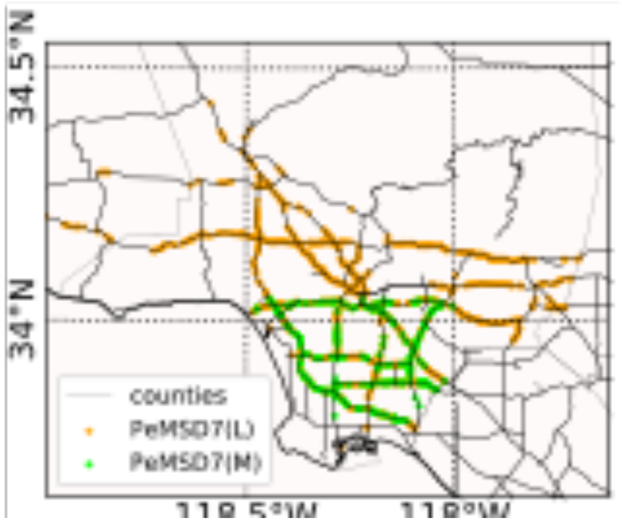
- Problème du commis voyageur

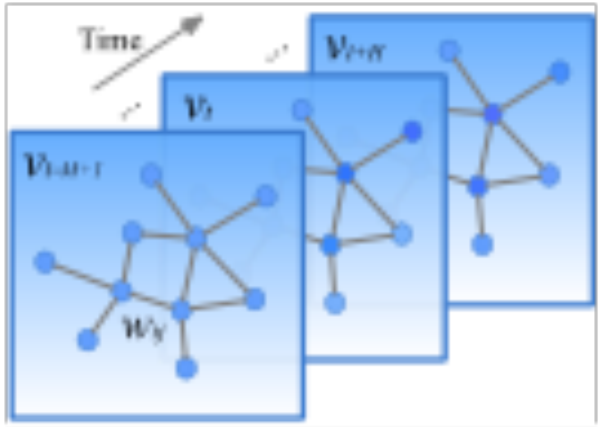


Joshi et al. An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem.

Applications
Transports

- Prédiction du trafic:
  - La carte routière comme un graphe

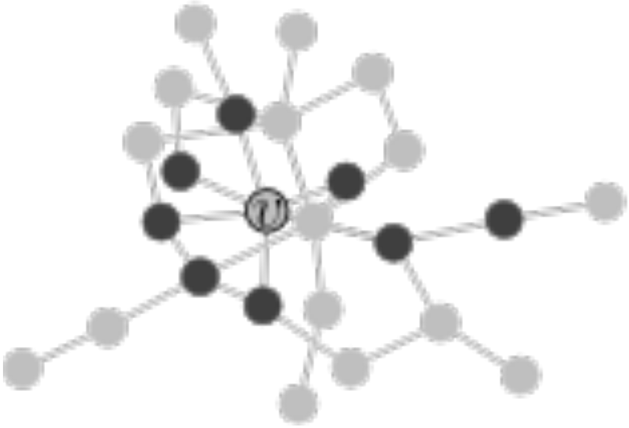




Yu et al. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting
IJCAI'18

Applications
Réseaux sociaux

- Prédiction d’influences
  - Prédit le statut d’un usager en fonction de ses voisins ou «ami.e.s»



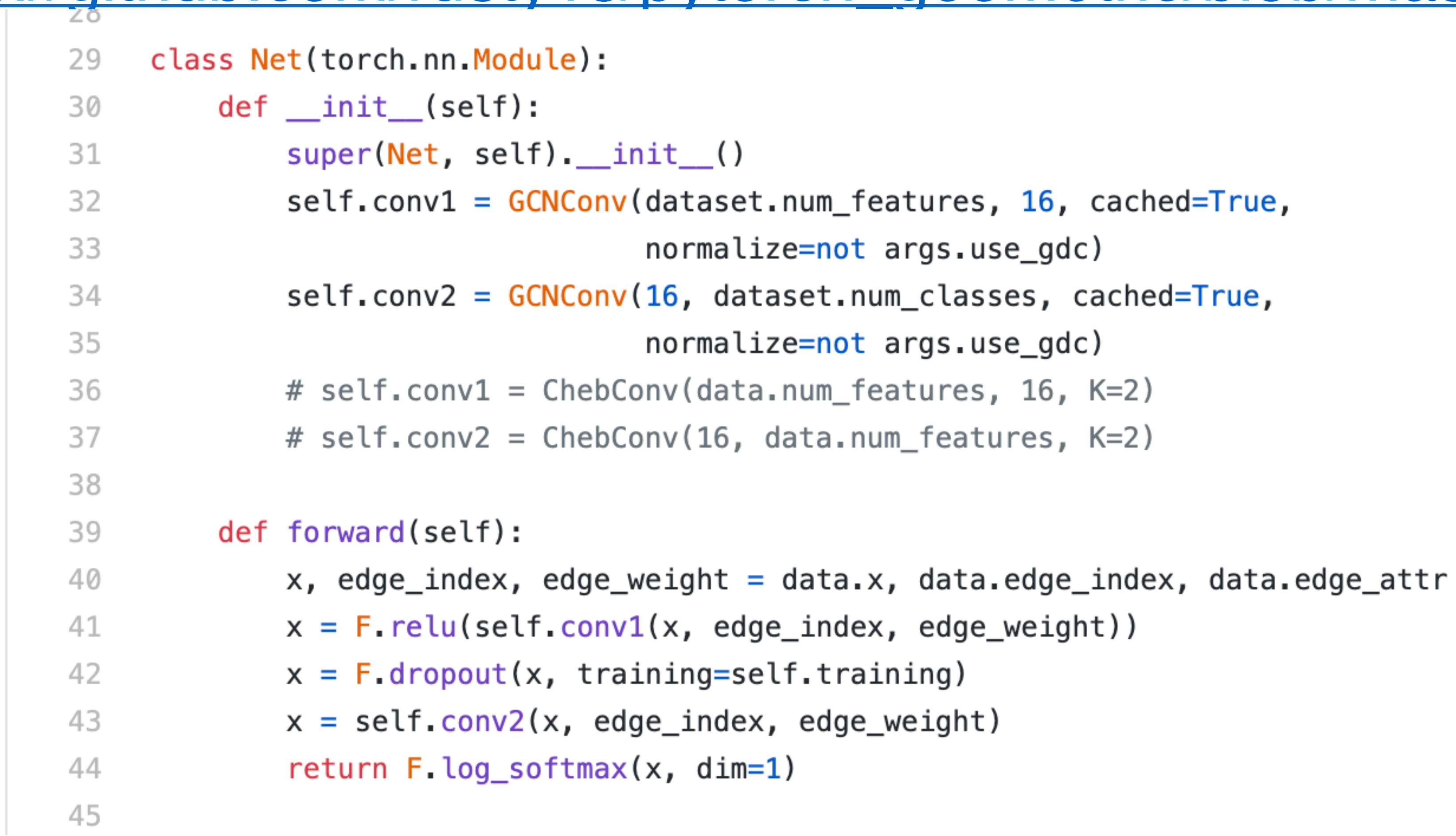
Qiu et al. DeepInf: Social Influence Prediction with Deep Learn

Quelques implémentations

- PyTorch Geometric: https://pytorchgeometric.readthedocs.io/en/latest/
- Deep Graph Learning: https://www.dgl.ai/

Exemple: GCN (Pytorch Geometric)

- https://github.com/rusty1s/pytorch_geometric/blob/master/examples/gcr



```python
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = GCNConv(dataset.num_features, 16, cached=True,
                             normalize=not args.use_gdc)
        self.conv2 = GCNConv(16, dataset.num_classes, cached=True,
                             normalize=not args.use_gdc)
        # self.conv1 = ChebConv(data.num_features, 16, K=2)
        # self.conv2 = ChebConv(16, data.num_features, K=2)

    def forward(self):
        x, edge_index, edge_weight = data.x, data.edge_index, data.edge_attr
        x = F.relu(self.conv1(x, edge_index, edge_weight))
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index, edge_weight)
        return F.log_softmax(x, dim=1)

```

Merci!