Apprentissage sur graphes

GraphML – 1e partie

Raphaël Fournier-S'niehotta

CNAM Paris, fournier@cnam.fr

HTT-FOD
RCP217
2020-2021

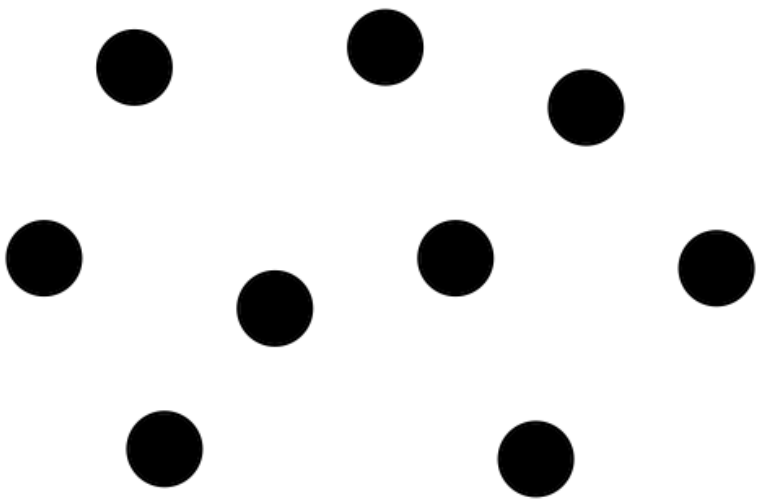
le cnam

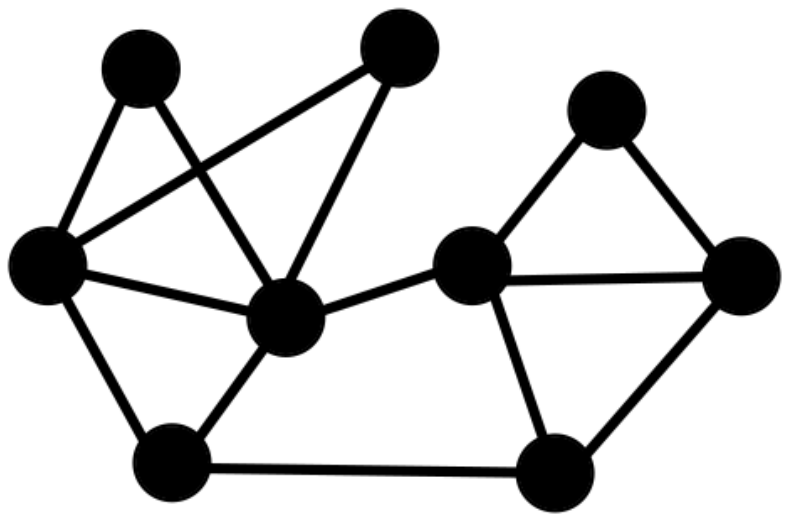
Plan

Introduction

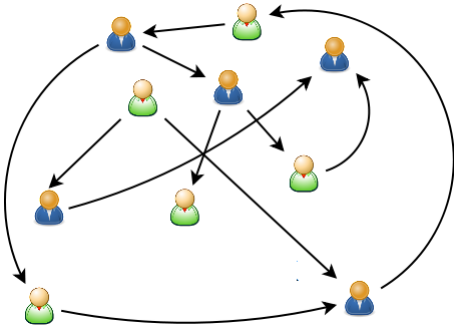
Les graphes

- Les graphes proposent un **langage général** pour décrire des **entités** qui ont des **interactions**



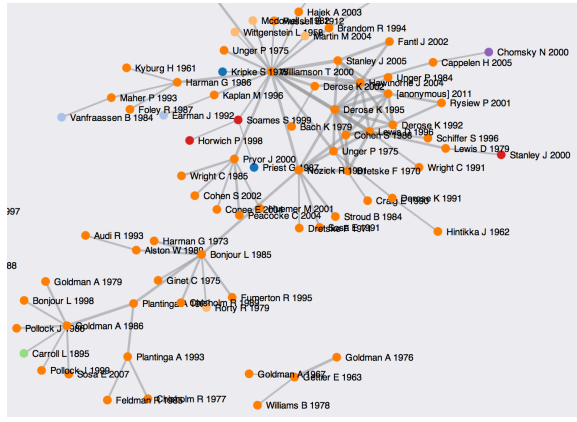


Types de graphes

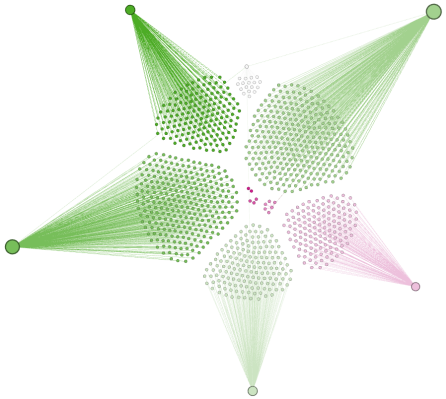


Réseau social

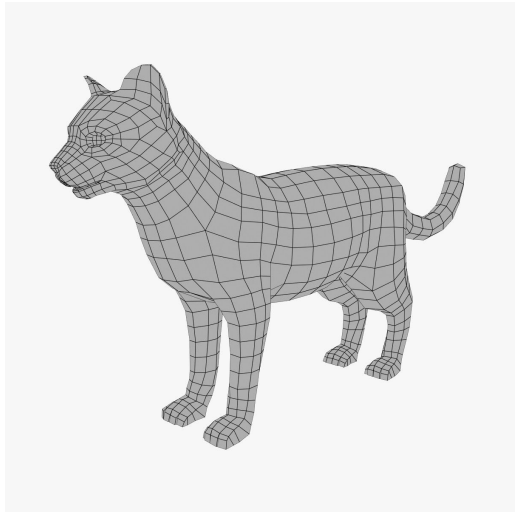
Réseau de transport



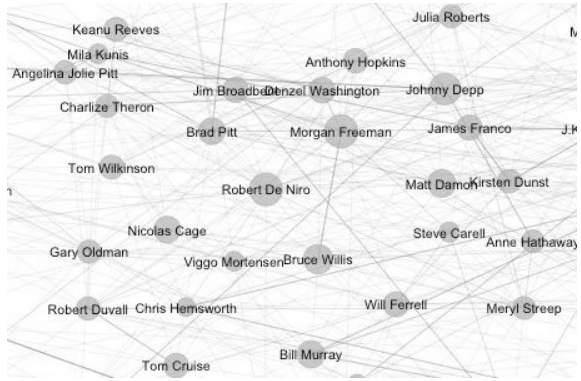
Réseau de citations



Réseau de communications



Formes 3D



Réseau de films

Domaines

- Réseaux sociaux
- Réseaux informatiques
- Communications
- Bio-informatique / médecine
- Cerveau

- Graphes de connaissances
- Dépendances logicielles
- Formes 3D

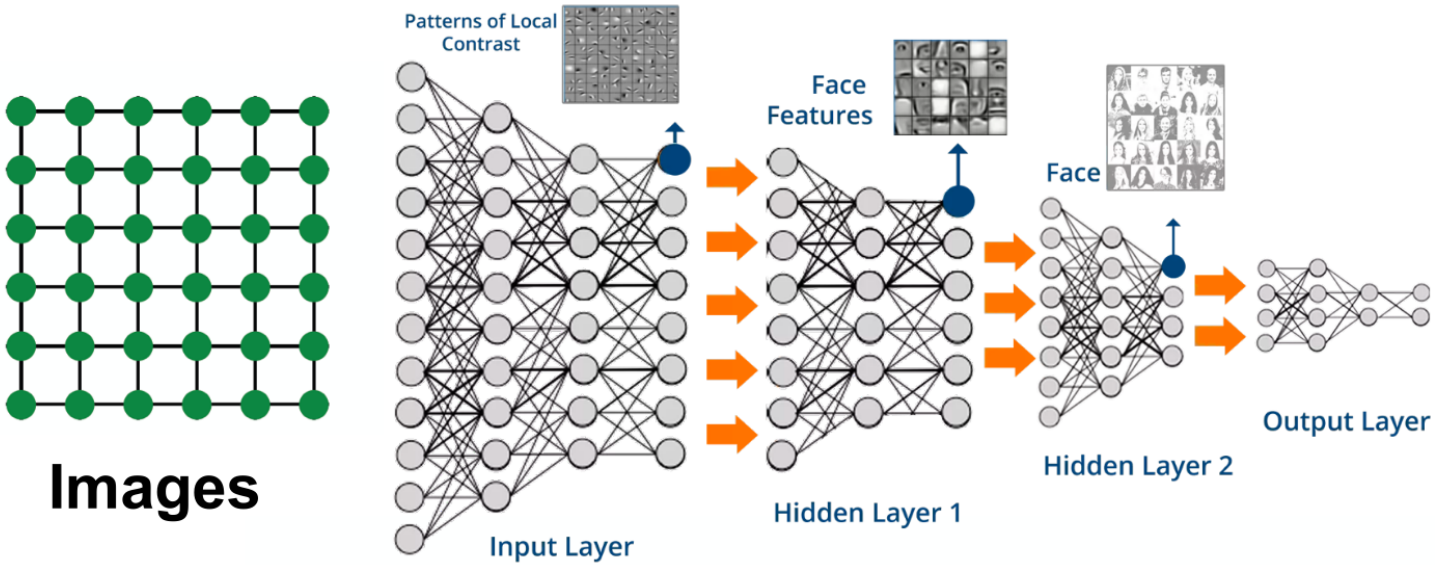
Objectif

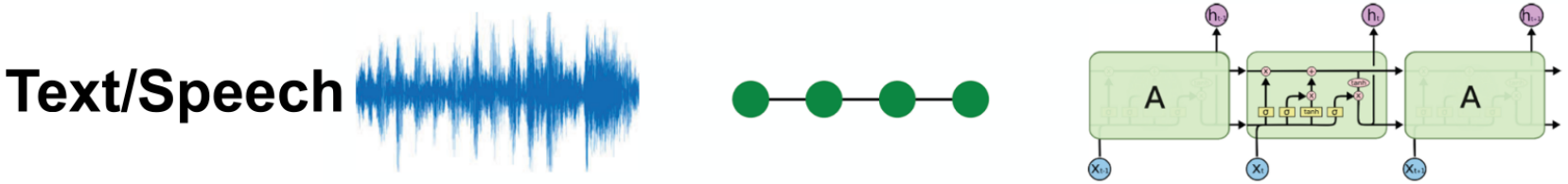
Tirer partie de la structure (relations) du graphe,
pour améliorer les prédictions.

Peut-on développer des réseaux de neurones
pour ces structures ?

Motivations, challenges

Machine learning classique :





Challenges :

- Complexité topologique : pas de régularité locale (grille)
- Pas d'ordre évident pour les nœuds, pas de points de référence
- Souvent dynamique

Tâches en Graph ML

- Classification de nœuds
- Prédiction de liens
- Classification de graphes
- Clustering

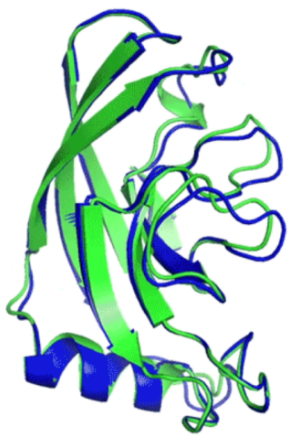
- Génération de graphes
- Évolution de graphes

Réussites

Tâches sur les nœuds

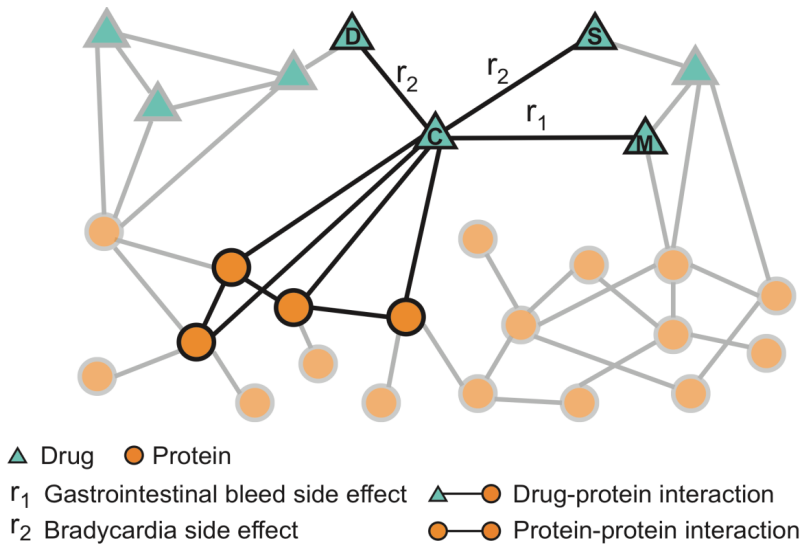
- Protein-folding (Alpha-Fold, Google)

T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)



T1049 / 6y4f
93.3 GDT
(adhesin tip)

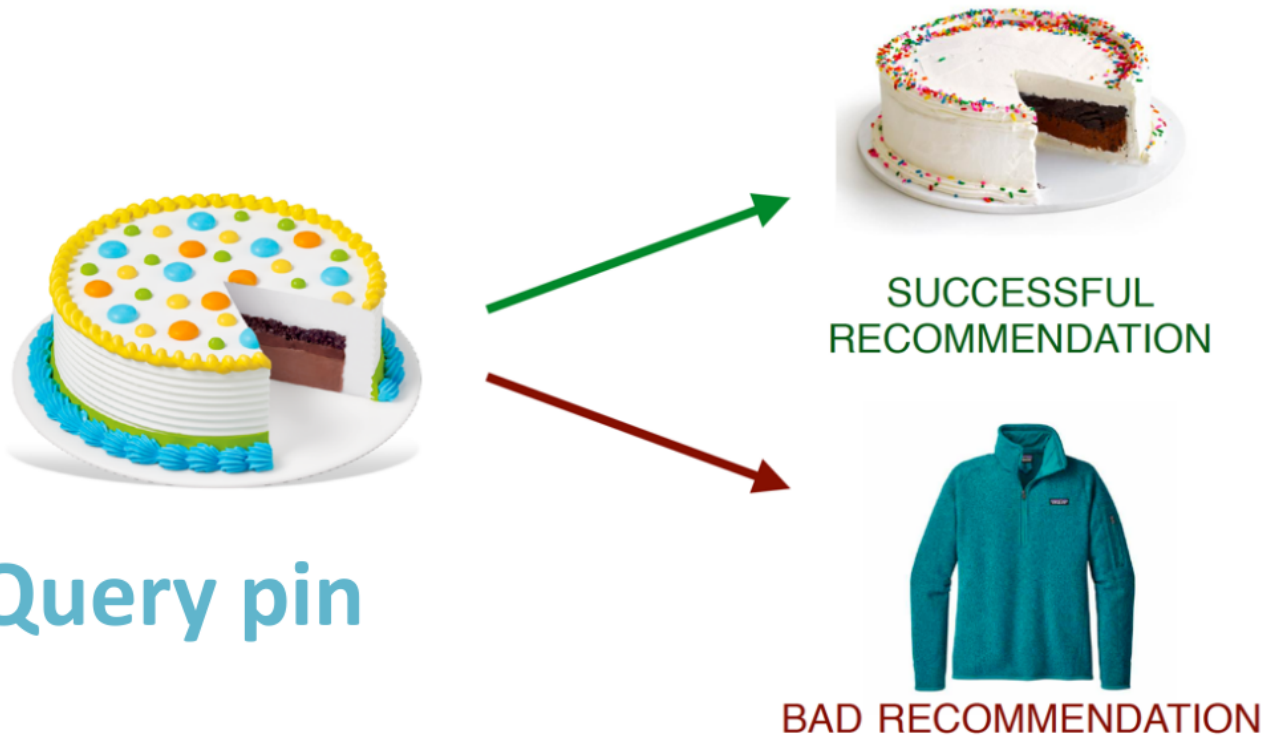
- Effets secondaires



Réussites : RecSys

Tâche sur les arêtes :

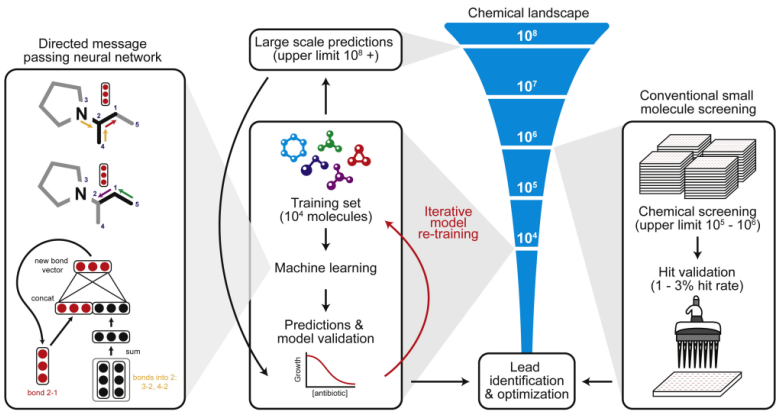
- Recommandation d'images chez Pinterest
- Très large échelle
- Proposer des images similaires à une autre
- Features multiples



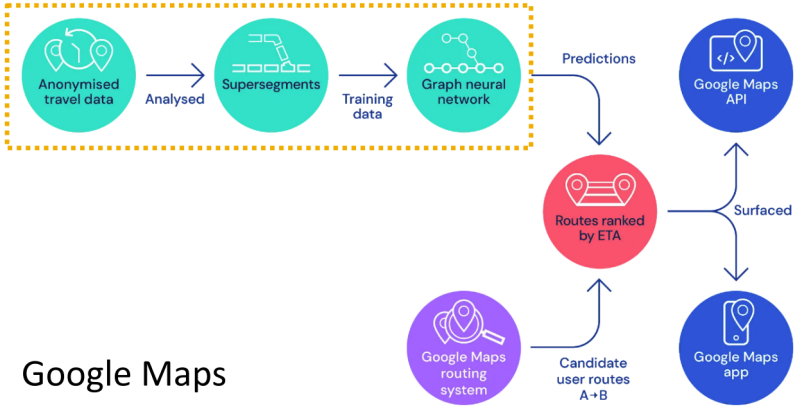
Réussites

Tâches sur les (sous-)graphes

- Sélection de molécule pour antibiotiques



- prédiction de trafic routier



Google Maps

Terminologie

Un graphe est défini par un couple G = (V,E) tel que :

- V (pour l'anglais *vertices*) est un ensemble fini de sommets
- E (pour l'anglais *edges*) est un ensemble fini d'arêtes

Un graphe peut être orienté, ou non :

- si oui, les couples $(v_i, v_j) \in E$ sont ordonnés, v_i est le sommet initial, v_j est le sommet terminal.
- on appelle alors le couple (v_i, v_j) un *arc*, représenté graphiquement par $v_i \rightarrow v_j$.
- si non, les couples ne sont pas orientés et (v_i, v_j) est équivalent à (v_j, v_i), et on l'appelle *arête*, représenté par $v_i - v_j$

Terminologie

- l'**ordre** d'un graphe, c'est son nombre de sommets (souvent désigné par n).
- une **boucle** est un arc/une arête reliant un sommet à lui-même
- un graphe dépourvu de boucle est dit **élémentaire**
- un graphe **simple** ne comporte pas de boucle et au plus une arête entre deux sommets
- un graphe **partiel** est le graphe obtenu en supprimant certains arcs ou arêtes
- un **sous-graphe** est le graphe obtenu en supprimant certains sommets et tous les arcs/arêtes incidents aux sommets supprimés.
- un sommet v_i est dit **adjacent** à un autre s'il existe une arête entre eux (on parle de **voisins**).
- le **degré** d'un sommet est le nombre d'arêtes incidentes à ce sommet.
- un graphe est dit **complet** s'il comporte une arête (v_i, v_j) pour toute paire de sommets $(v_i, v_j) \in E^2$.

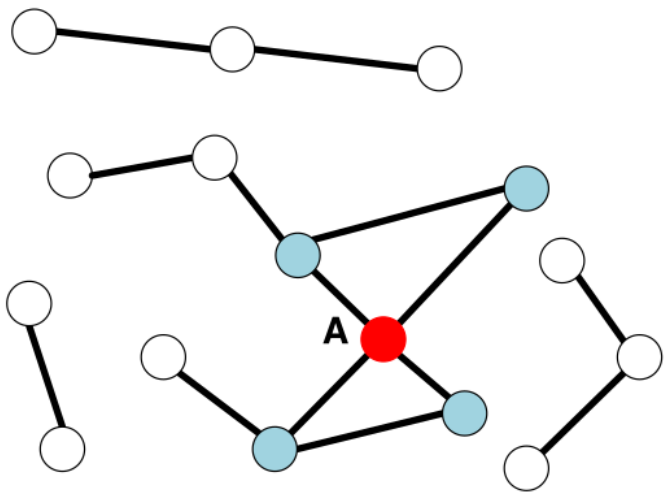
Représentation

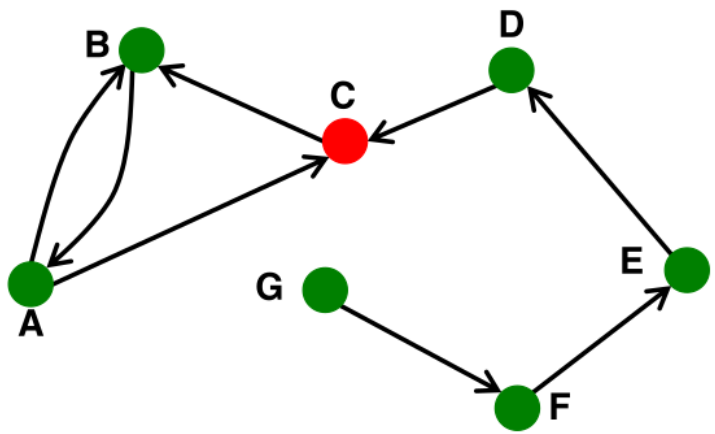
- Un graphe de gens qui travaillent ensemble, c'est un réseau professionnel
- Un graphe d'articles qui se citent, c'est un réseau de citations
- Un graphe de livres qui contiennent les mêmes mots dans leur titre : pas de nom, mais c'est un réseau...

- Importance de bien choisir nœuds et arêtes !
- Parfois, c'est évident, souvent pas
- Le choix des liens détermine la question à laquelle on répond

Degrés des nœuds

- Le degré d'un nœud, c'est le nombre de ses voisins qui lui sont directement liés
- Dans les graphes dirigés, on définit un degré entrant et un degré sortant (et le degré total est la somme des deux)

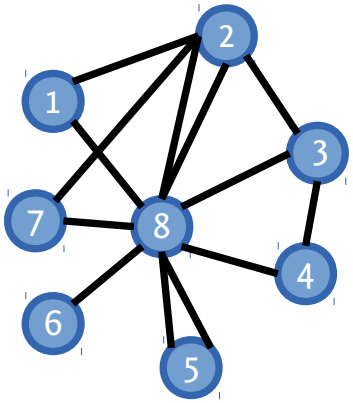


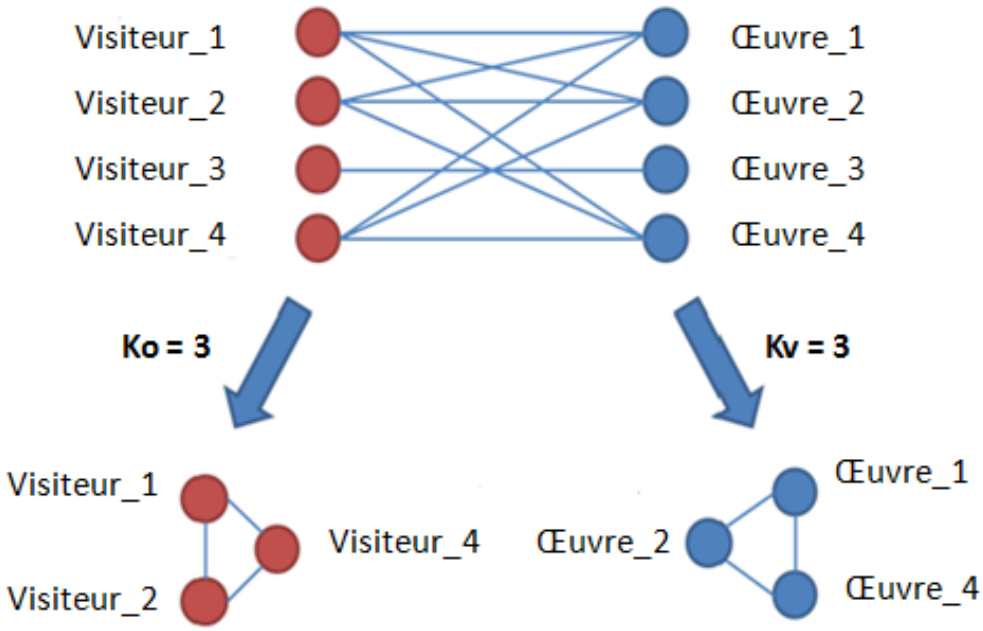


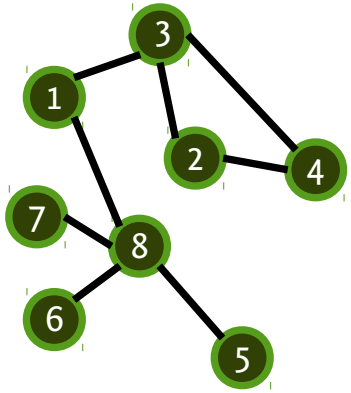
Graphe biparti

- Un graphe est dit biparti si l'on peut diviser en deux ensembles disjoints U et V ses sommets, de façon que chaque arête ait un sommet dans U et un dans V
- Exemples :
  - utilisateurs-articles (recommandation : films, titres, etc.)
  - auteurs-œuvres (recherche)
  - recettes-ingrédients
- on peut "projeter" ces réseaux, et analyser la projection (attention au seuil)

Projection de biparti

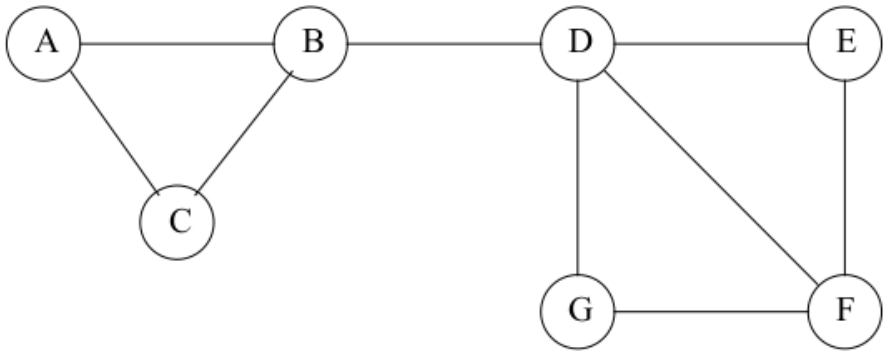






Représentation de graphes

- Matrice d'adjacence : $m_{ij} = 1$ si l'arête (v_i, v_j) existe, 0 sinon.



$$\begin{array}{c} \\ A \\ B \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & & & & & \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

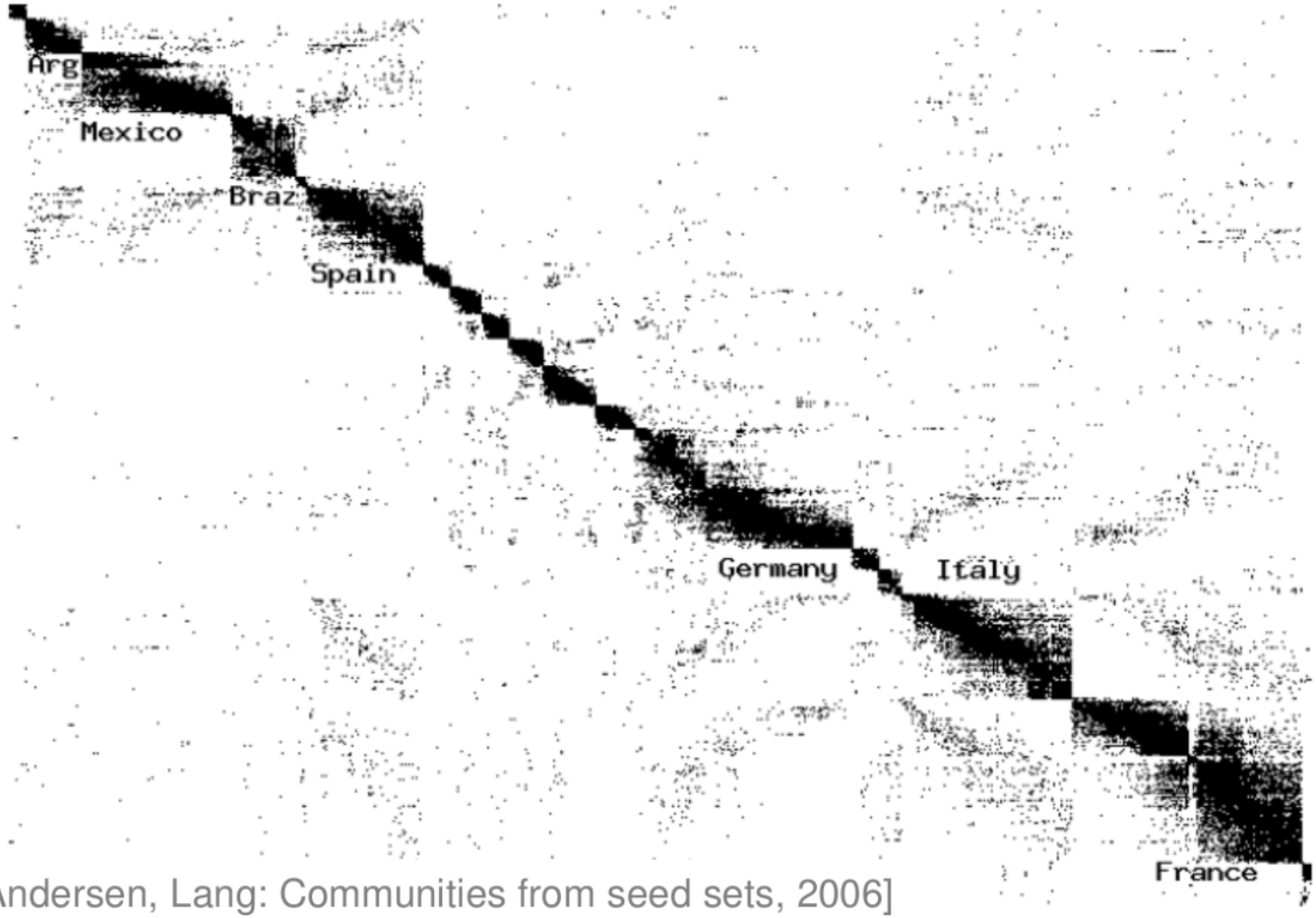
- Matrice des degrés : $m_{ij} = d(v_j)$ pour $i = j$, 0 sinon.

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

- Matrices de Laplace (ou "laplacienne") : L = D - A (attention : nombreuses variantes)

Sparsity

Attention : les matrices d'adjacence sont souvent très creuses.

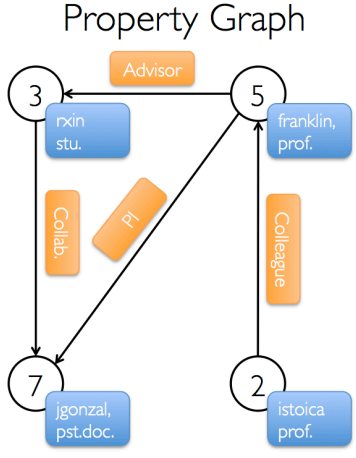


[Andersen, Lang: Communities from seed sets, 2006]

Attributs

Pour les nœuds et les liens :

- poids (fréquence de communication)
- rang (collaborateur préféré, 2e préféré)
- type (ami, collaborateur, famille)
- signé ou non : confiance, amitié/inimitié
- propriétés liées au graphe : nombre de voisins en commun



Vertex Table

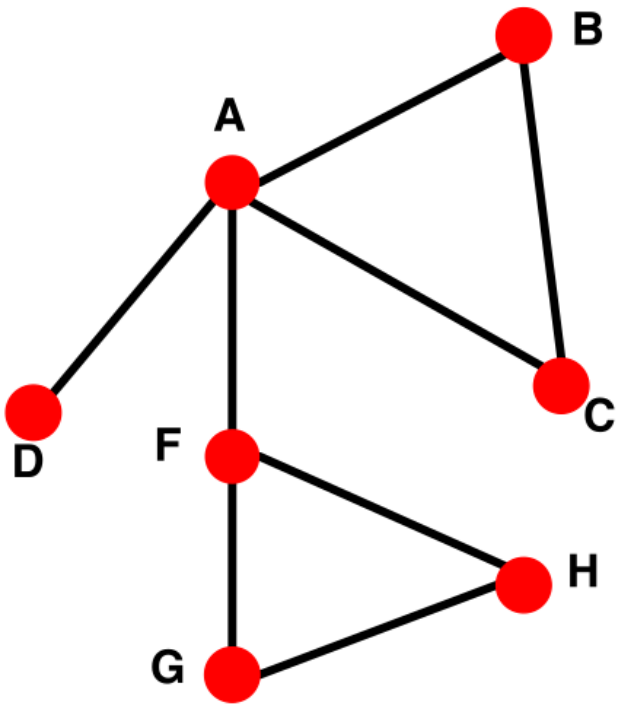
Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

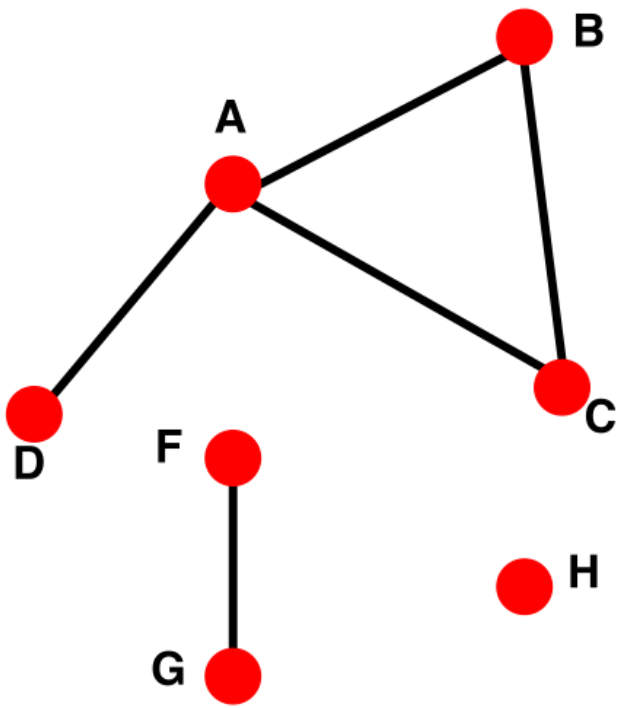
Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Connexité

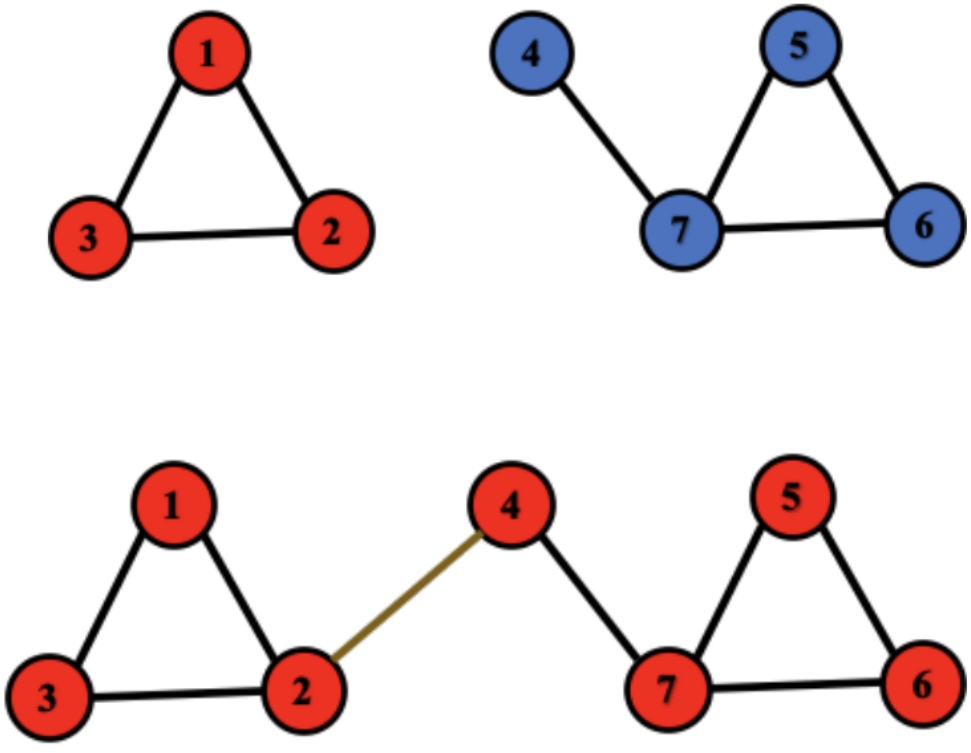
- Un graphe est connecté si deux sommets quelconque peuvent être liés par un chemin
- Un sous-graphe connecté est appelé une composante connexe
- Un graphe est déconnecté s'il a deux composantes connexes ou plus





Connexité et matrice d'adjacence

La matrice d'adjacence d'un graphe avec plusieurs composantes connexes peut être ré-écrite pour être diagonale par blocs.

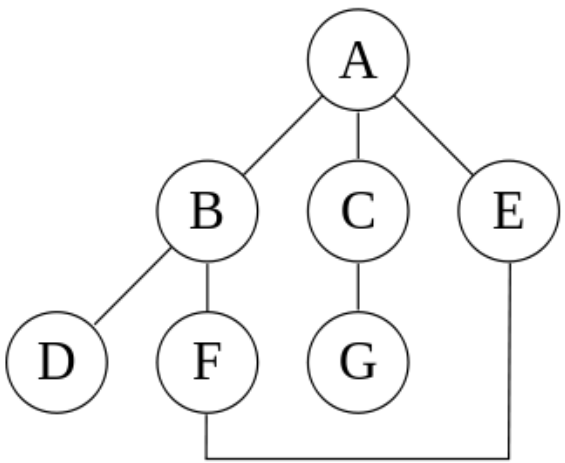


$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

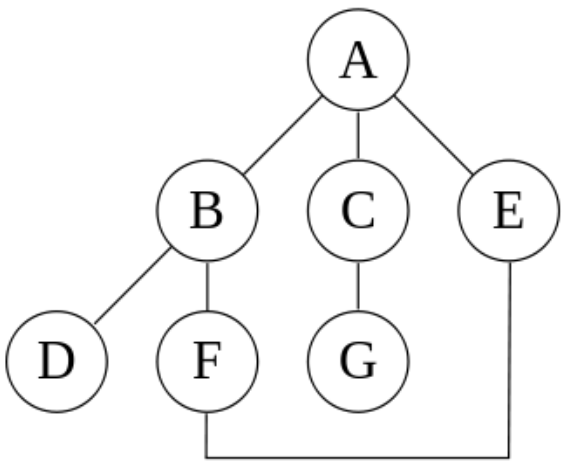
Parcours en profondeur

- En anglais, DFS, pour Depth First Search
- progresse à partir d'un sommet S en s'appelant récursivement pour chaque sommet voisin de S.
- pour chaque sommet, on prend le premier sommet voisin, on explore tous ses voisins (non marqués) avant de revenir au "père"
- Ordre de visite : A, B, D, F, E, C, G
- s'implémente avec une pile (LIFO)



Parcours en largeur

- En anglais, BFS, pour Breadth First Search
- pour chaque sommet, on repère tous ses voisins, on stocke ceux qui ne sont pas marqués dans une file (queue)
- Ordre de visite : A, B, C, E, D, F, G
- s'implémente avec une file (FIFO)
- on obtient les plus courts chemins à la racine

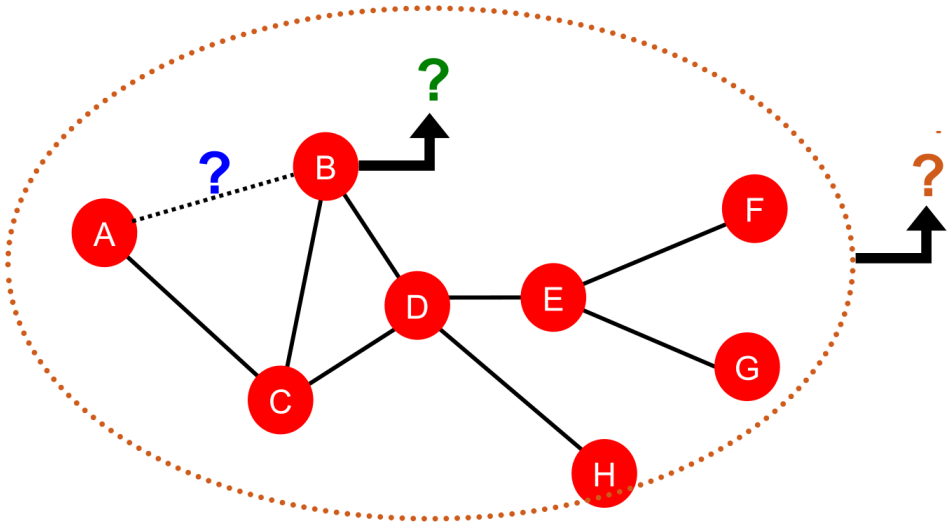


Apprentissage classique sur des graphes

Apprentissage classique sur graphes

Trois types de tâches :

- prédiction sur les nœuds
- prédiction sur les arêtes
- prédiction au niveau graphe entier

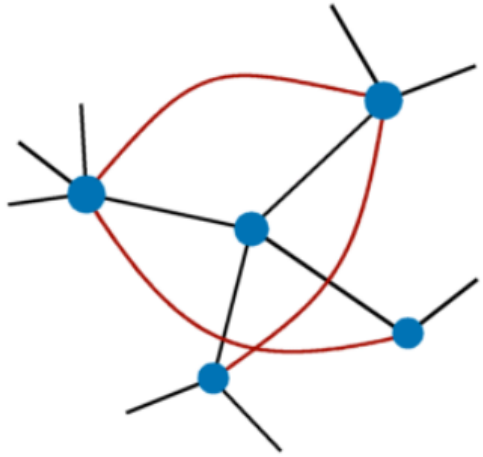


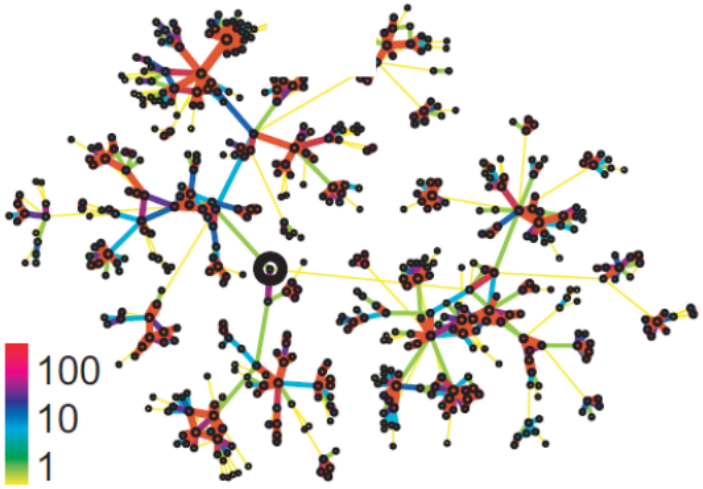
Schéma/pipeline classique

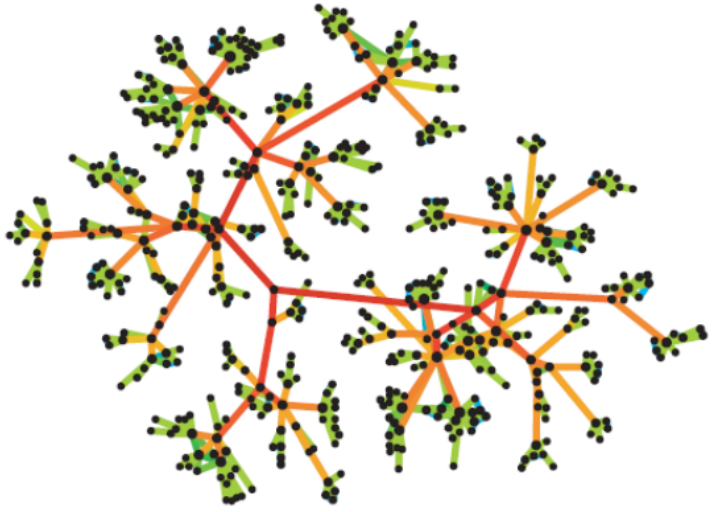
- On conçoit des caractéristiques pour les nœuds/arêtes/sommets
- On obtient les valeurs de ces caractéristiques pour toutes les données d'apprentissage
- On entraîne le modèle
- On l'applique : pour un nouveau nœud/lien/graphe, on utilise ses caractéristiques pour faire une prédiction

Attributs de nœuds

- degré (voisinage, sans l'importance)
- centralité (diversement calculée)
- coefficient de clustering (densité locale)
- graphlets

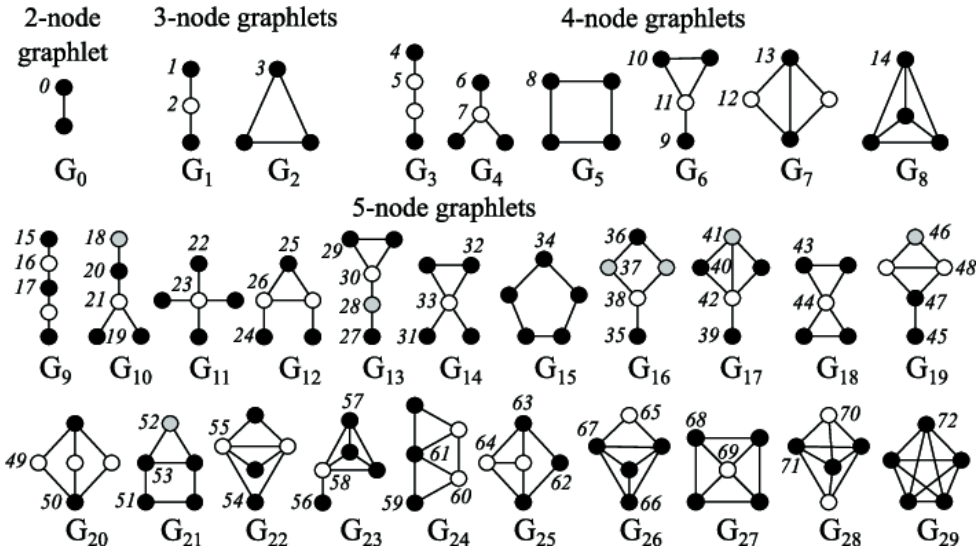




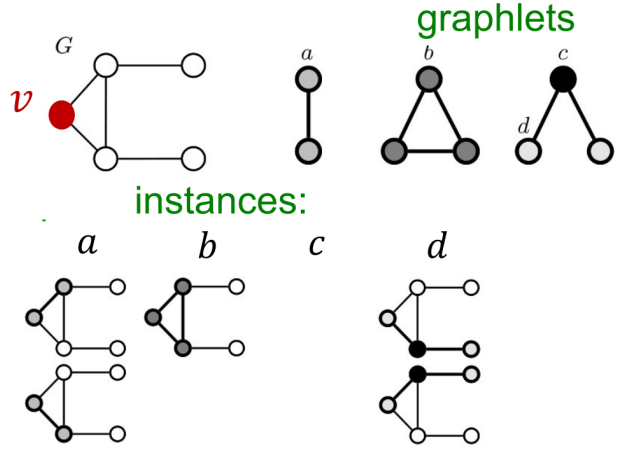


Graphlets

- sous-graphes connectés non isomorphes



- GDV de v : $[2, 1, 0, 2]$
- Graphlet Degree Vector : mesure de topologie locale (dim 73 !)



Attributs de liens

- Métrique reposant sur la distance
  - plus court chemin entre deux nœuds
- Voisinage local (cf cours Reco)
  - Common Neighbors
  - Indice de Jaccard
  - Adamic-Adar
- Structure globale
  - Indice/centralité de Katz (cf Reco)

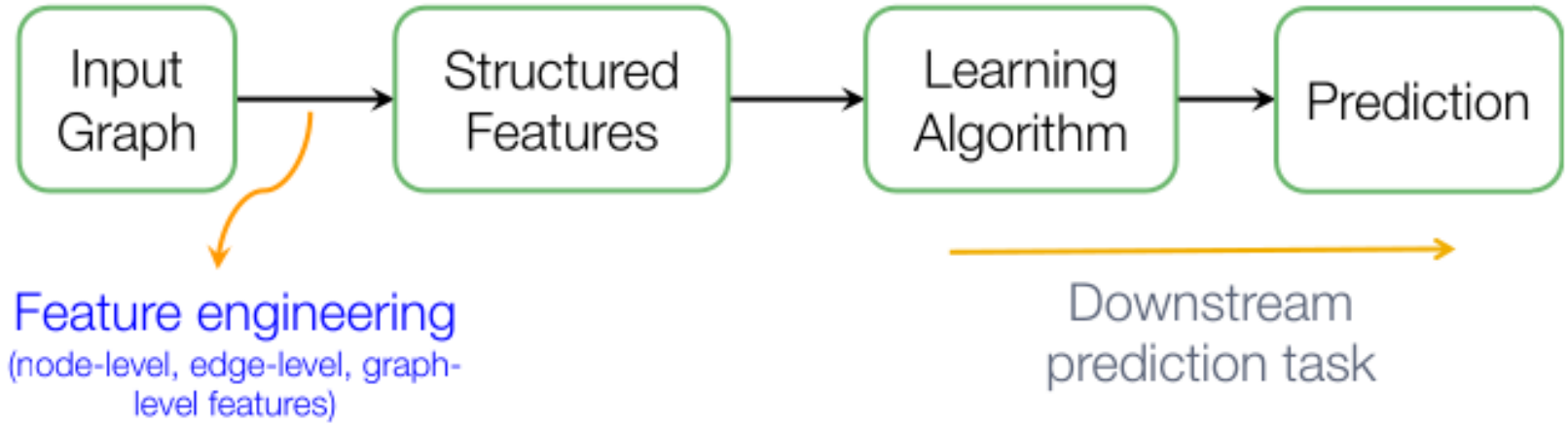
Attributs de graphes entiers

- Méthodes à noyaux.
  L'idée : trouver des noyaux plutôt que des vecteurs de caractéristiques. Une fois le noyau défini, on prend un modèle sur étagères (SVM) pour prédire
- Idée commune : généraliser les "bag-of-words" (IR/NLP) aux graphes
- Quelques modèles :
  - Noyau à Graphlets (généralisés, avec nœuds isolés)
  - Noyau de Weisfeiler-Lehman (Bag-of-degrees subtil)
  - Noyau à marche aléatoire, à plus court chemin, etc.

Résumé

Le ML classique sur les graphes consiste donc à :

- extraire les features (attributs, puis métriques) pour les nœuds, les liens, voire le graphe
- apprendre un modèle



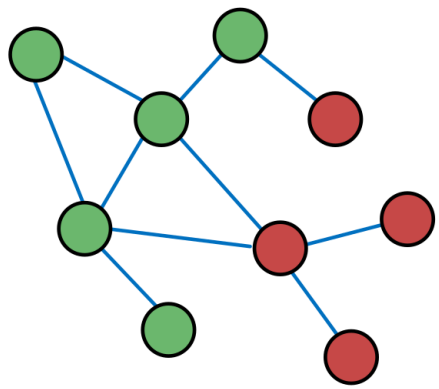
Passage de messages

Le passage de message est un framework pour faire de la classification de nœuds.

- Cela repose sur les corrélations dans le graphe : des nœuds similaires sont connectés (homophilie, cf RecSys)
- Classification collective : on cherche à affecter les étiquettes de tous les nœuds du graphe
- On suppose que le label Y_v d'un nœud v dépend des labels de ses voisins :

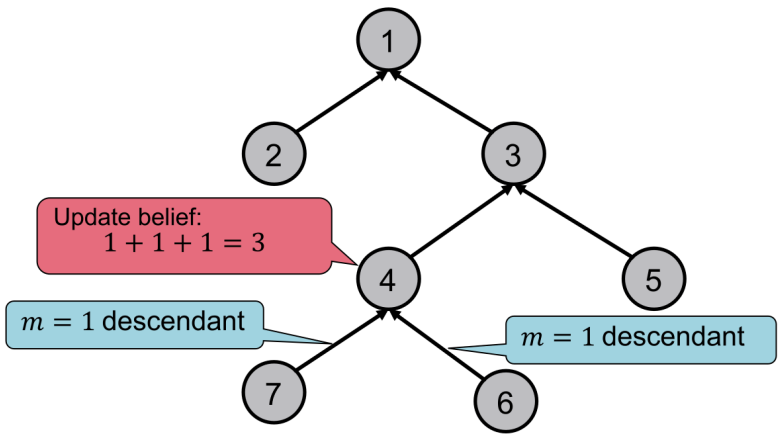
$$P(Y_v) = P(Y_v|N_v)$$

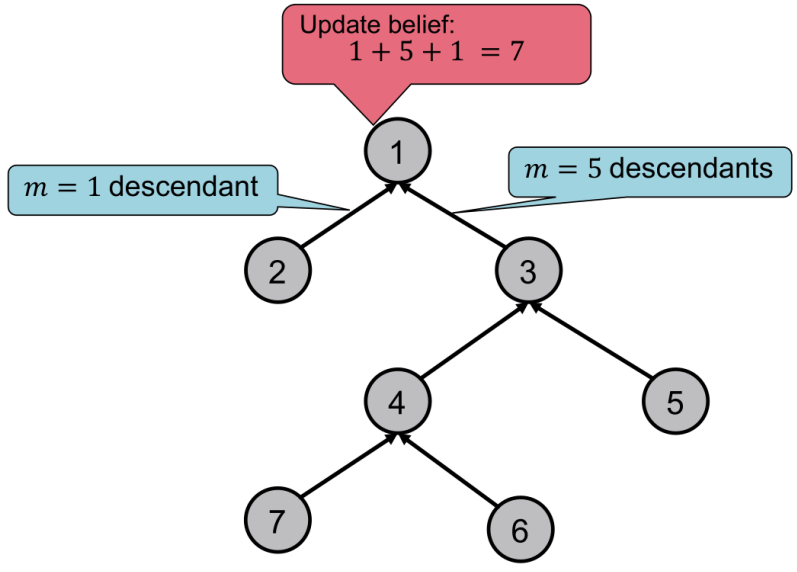
- 3 étapes :
  1. on affecte des étiquettes
  2. on capture les corrélations entre nœuds
  3. on propage ces corrélations dans le réseau



Techniques de passage de message

- Classification relationnelle
  - probabilité qu'un nœud ait une classe
  - mise-à-jour : moyenne (pondérée) du voisinage
  - risque de non convergence et pas d'usage des attributs
- Classification itérative (utilisation d'attributs)
  - résumé à partir des attributs les plus fréquents dans le voisinage
- Propagation de croyance (*belief propagation*, avec programmation dynamique)

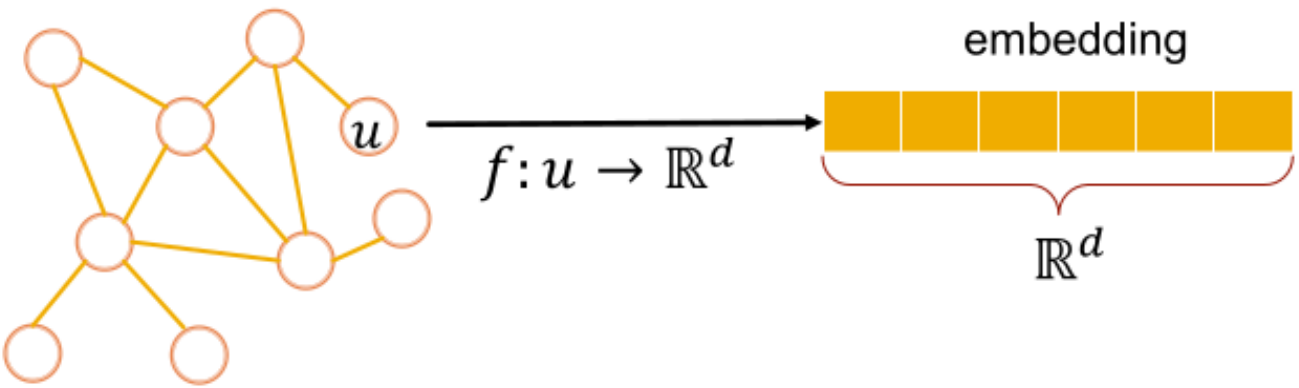




Embeddings

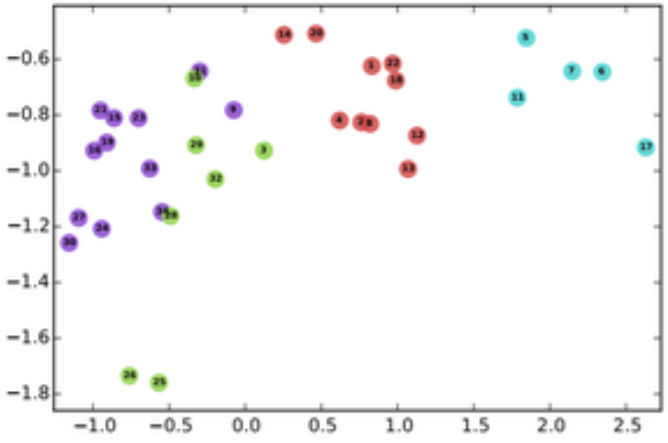
Embeddings

- Embeddings = plongements
- Objectif : apprendre des représentations indépendantes des tâches
- On veut une représentation vectorielle des nœuds
  - similarité d'embedding = similarité dans le réseau (proximité, symétrie)
  - encoder l'information structurelle



Exemple : Karaté-club

(a) Input: Karate Graph



(b) Output: Representation

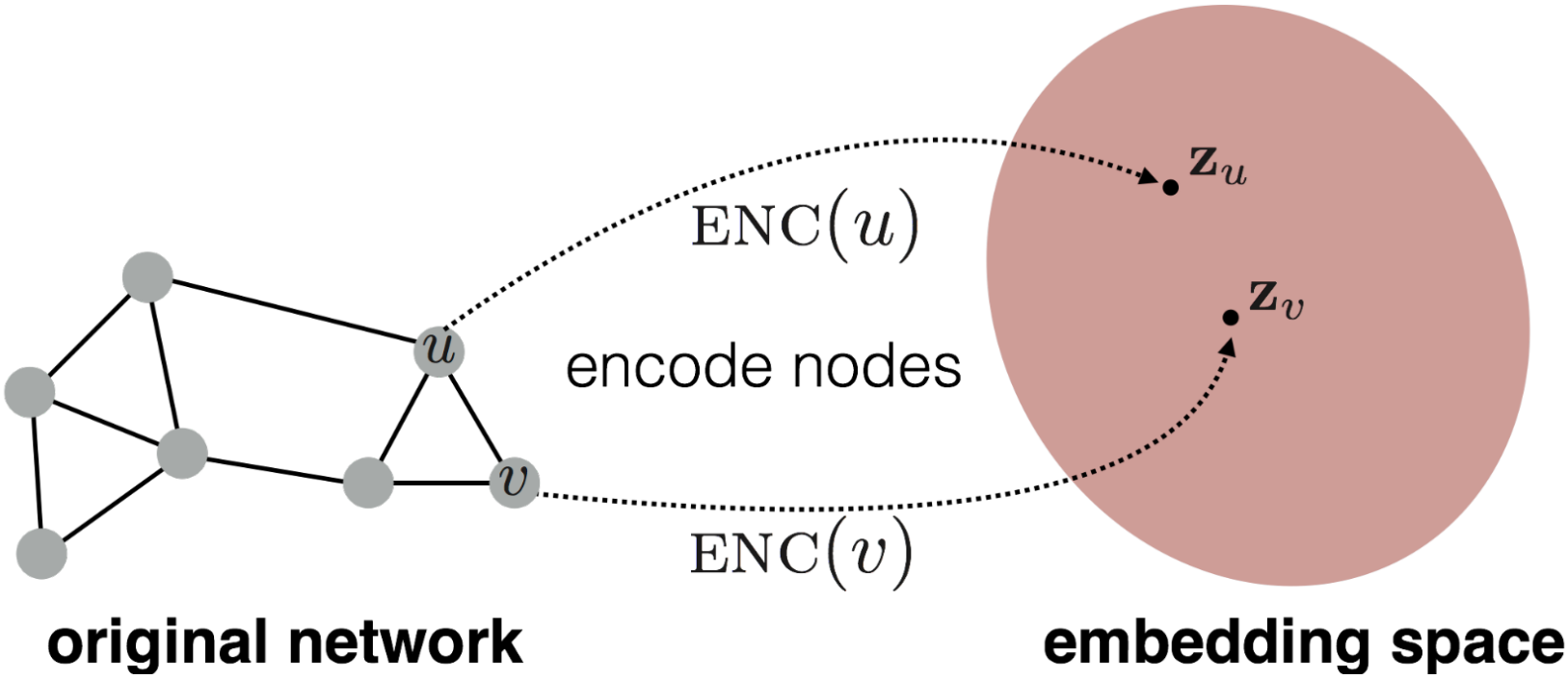
Perozzi et al, 2014. ”DeepWalk”

Objectif

On voudrait :

$$sim(u,v) \approx z_v^T z_u$$

- sim(u,v) concerne le graphe initial
- $DEC(z_v, z_u) = z_v^T z_u$ calcule la similarité d'embeddings



Apprendre les embeddings

- L'encodeur ENC fait la correspondance des nœuds vers les embeddings

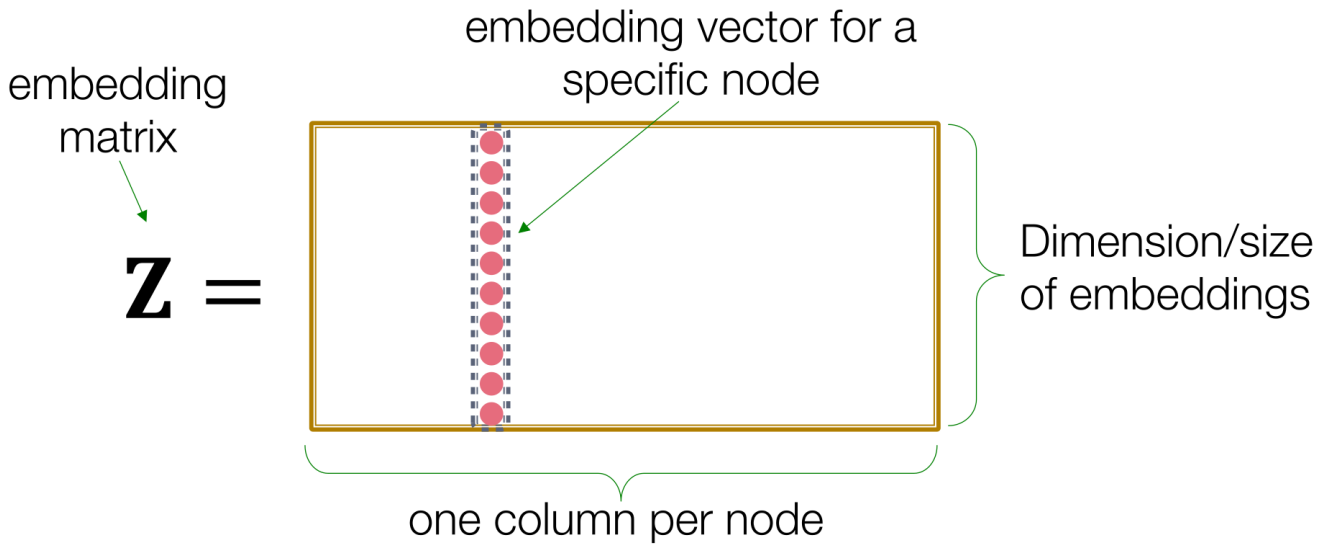
$$\text{ENC}(v) = z_v \qquad \text{(de dimension d)}$$

- On définit une similarité entre nœuds dans le réseau original
- Le décodeur DEC fait la correspondance entre embeddings et similarité
- On cherche à optimiser les paramètres (encodeur, décodeur) tels que :

$$\text{sim(u,v)} \approx DEC(z_v, z_u) = z_v^T z_u$$

Encodeur superficiel

- L'encodeur est juste un "lookup"
- $ENC(v) = z_v = Z \cdot v$ où
  - $Z \in \mathbb{R}^{d \times |V|}$ est une matrice (chaque colonne est un embedding (appris)
  - $v \in \mathbb{I}^{|V|}$ est un vecteur-indicateur, nul sauf sur la ligne correspondant à v



- En fait, on optimise/apprend "directement" les embeddings de chaque nœud (via la matrice Z).
- La fonction objectif doit maximiser $z_v^T z_u$ pour les paires de nœuds similaires

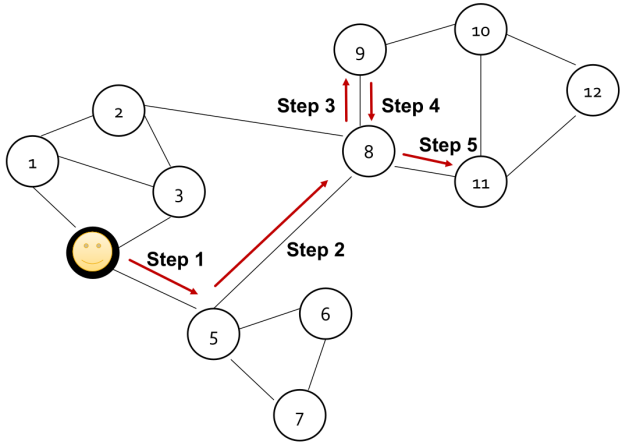
Similarités

On veut ”plonger” les nœuds dans un espace tel que les distances dans celui-ci reflètent les similarités dans le graphe original. Comment faire ?

- Naïvement : similarité si connexion entre nœuds
- Indicateurs de voisinage (Jaccard, Adamic-Adar)
- Approches par **marches aléatoires**
  - expressivité : incorpore de l’information locale et globale sur le graphe
  - plus efficace : on ne considère pas toutes les paires dans l’entraînement, seulement celles qui sont co-apparaissent dans une marche aléatoire
  - Méthodes DeepWalk, node2vec
- On peut aussi faire de l’embedding de graphes entiers (hors scope)

Similarités avec marches aléatoires

- Objectif : estimer directement des valeurs d'embedding pour un nœud de façon à préserver des aspects de la structure du graphe



On veut : $z_v^T z_v \approx$ P ("u,v" dans la même marche aléatoire)
On estime la probabilité de visiter le nœud v quand on fait une marche

aléatoire depuis u :

- on fait une marche aléatoire de longueur fixée, à partir de chaque nœud dans le graphe
- on collecte l'ensemble des nœuds visités
- on optimise avec une SGD :

$$L = \sum_{v \in V} \sum_{v \in N_R(u)} -log(P(v|z_u))$$

Similarités avec marches aléatoires

Comment faire la marche aléatoire :

- marche aléatoire non biaisée, depuis chaque nœud (DeepWalk. Perozzi et al., 2013)
- node2vec : marche aléatoire ”flexible”, avec 2 paramètres :
  - possibilité de retourner au nœud précédent
  - ratio d’examen par BFS ou DFS
  - linéaire en temps et parallélisable

Embeddings : à quoi ça sert

Une fois qu'on a les z_i, on peut faire :

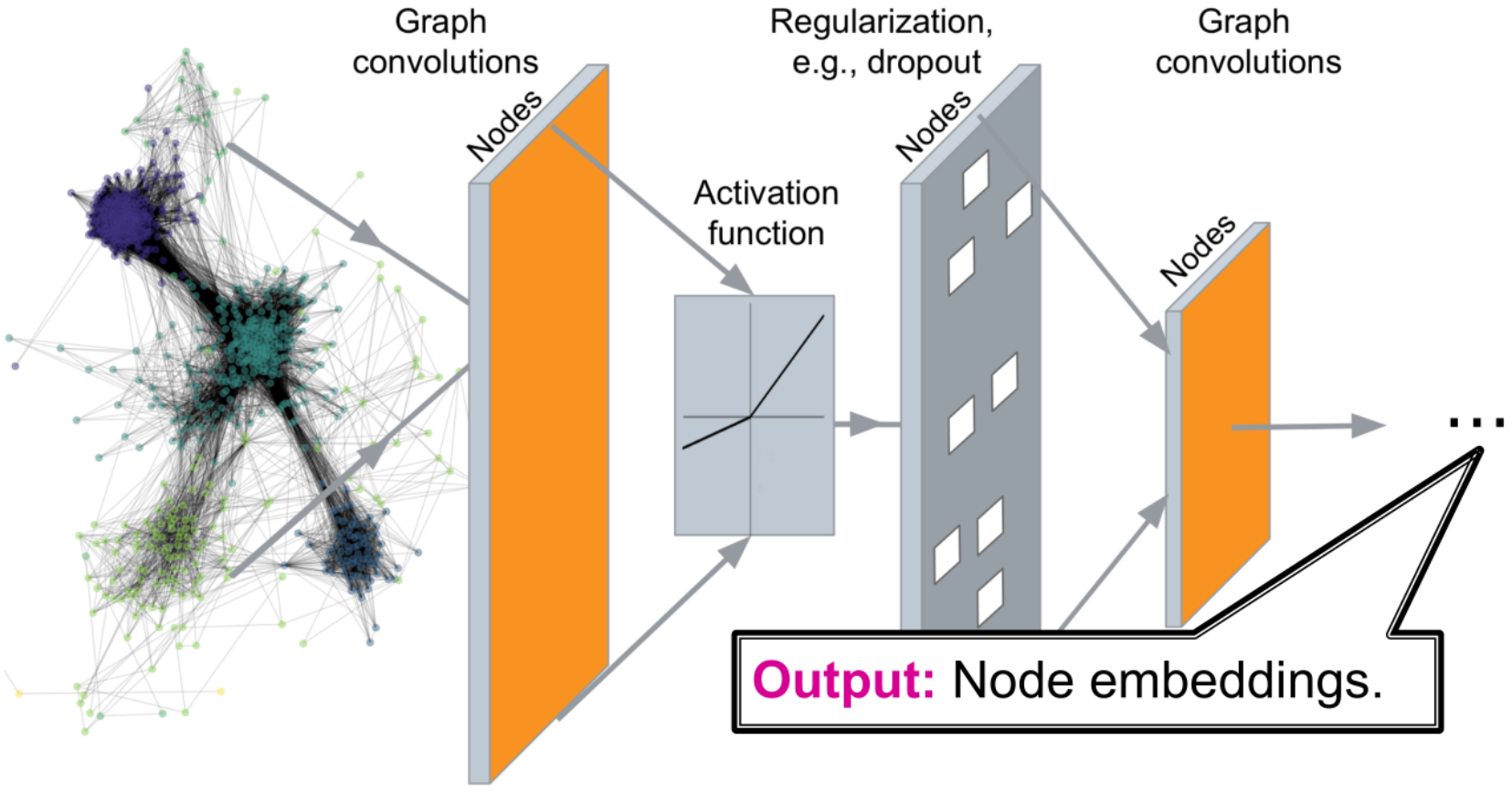
- du clustering/de la détection de communautés
- classer des nœuds, en prédisant une étiquette avec z_i
- prédire des liens (i, j) à l'aide de (z_i, z_j)
  - concaténation, somme, moyenne d'embeddings
- classer des graphes entiers, en agrégeant des z_i

Graph neural networks

Limite du Shallow encoding

- beaucoup de paramètres ($\mathcal{O}(|V|)$) sont nécessaires
  - pas de partage de paramètres entre nœuds
  - chaque nœud a un embedding unique
- transductif uniquement : impossible de générer des embeddings pour des nœuds non vus lors de l'entraînement
- pas d'utilisation des attributs des nœuds

Encodeurs profonds pour graphes

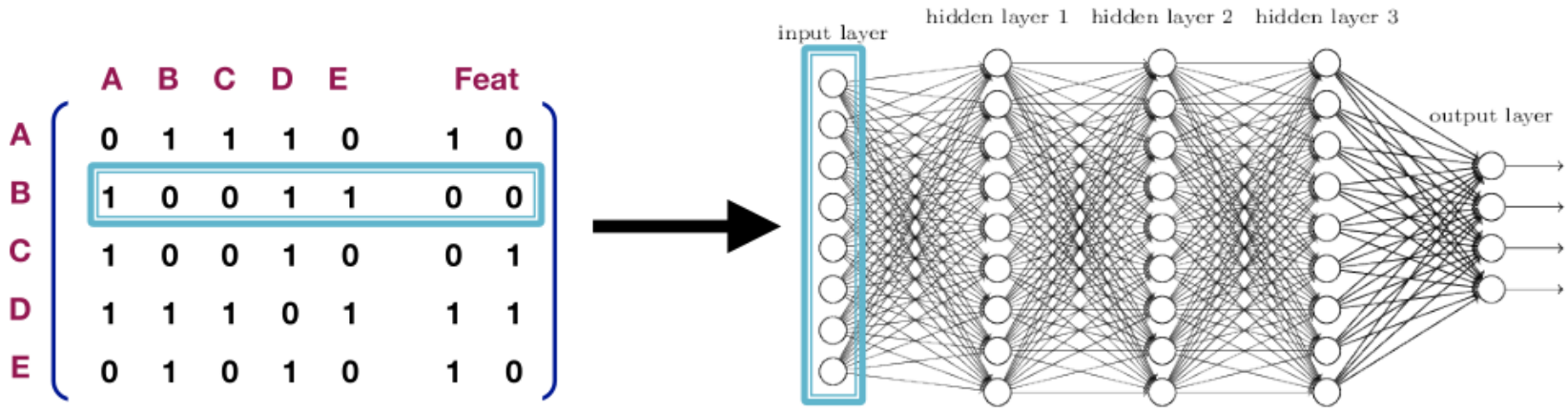


Apprentissage profond et graphes

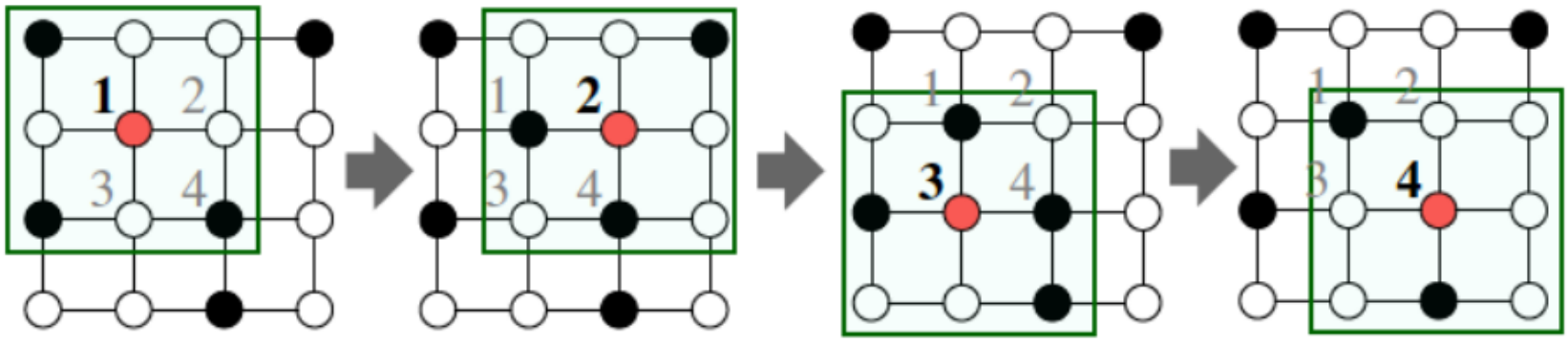
- Graphe G
- Matrice d'adjacence A (binaire)
- $v \in V$, $N(v)$ ses voisins
- $X \in \mathbb{R}^{m \times |V|}$ matrice d'attributs
  - profil de réseau social, image, etc.

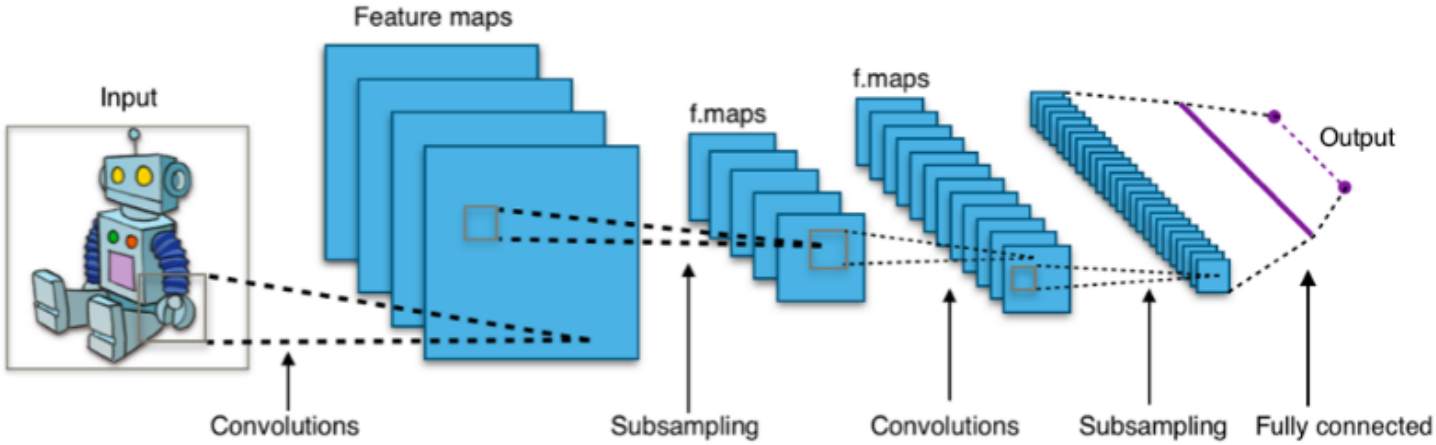
Approche naïve

- fusionner / concaténer la matrice d'attributs et celle d'adjacence, la passer dans un réseau de neurones classique
- problèmes :
  - toujours $O(|V|)$ paramètres
  - peu applicable aux graphes de différentes tailles
  - sensibilité à l'ordre des nœuds

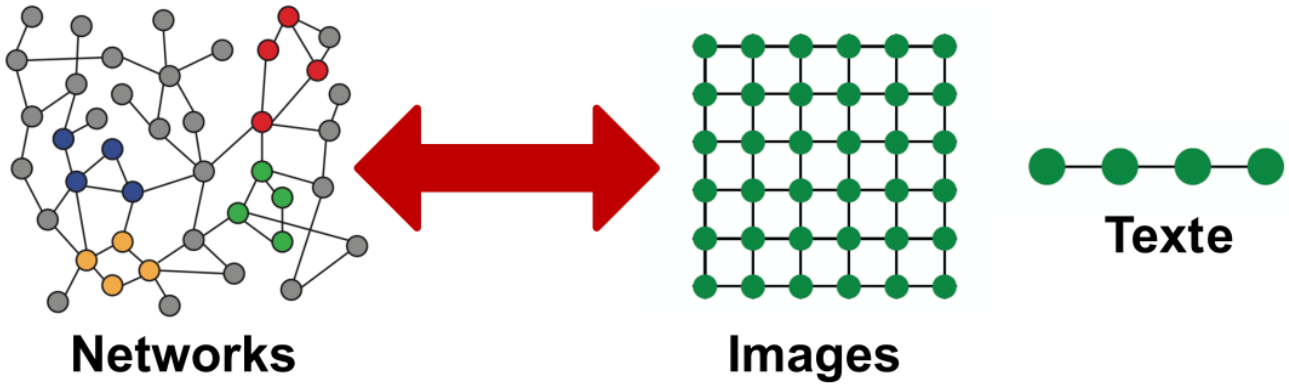


CNN et images





Localité et graphes

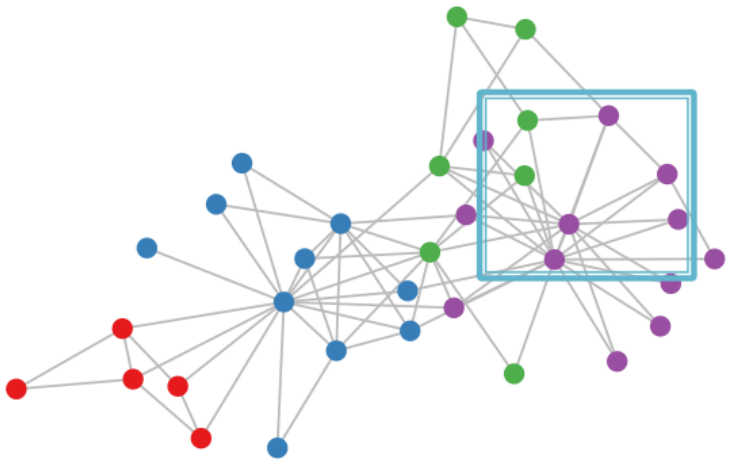


- Les graphes n'ont pas la notion classique de "localité", ou de fenêtre glissante

- Les graphes sont "invariants par permutation" :

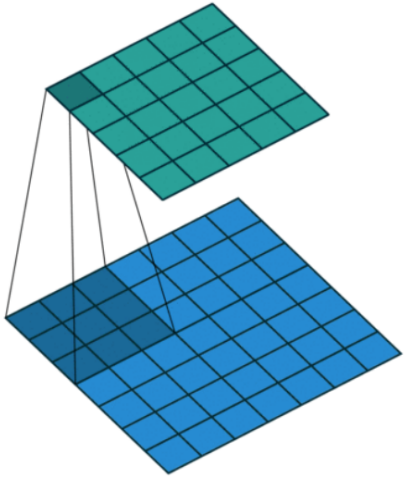
$$f(PAP^T) = f(A) \qquad \text{invariance}$$

$$f(PAP^T) = Pf(A) \qquad \text{equivariance}$$

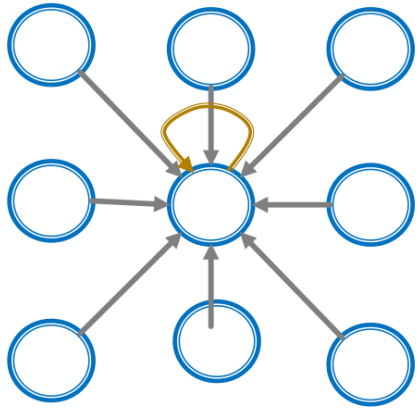


Des images aux graphes

- L'idée est de "transformer" l'information des voisins :
  - les embeddings h_i des voisins sont transformés $W_i \cdot h_i$
  - Puis sommés : $\sum_i W_i h_i$



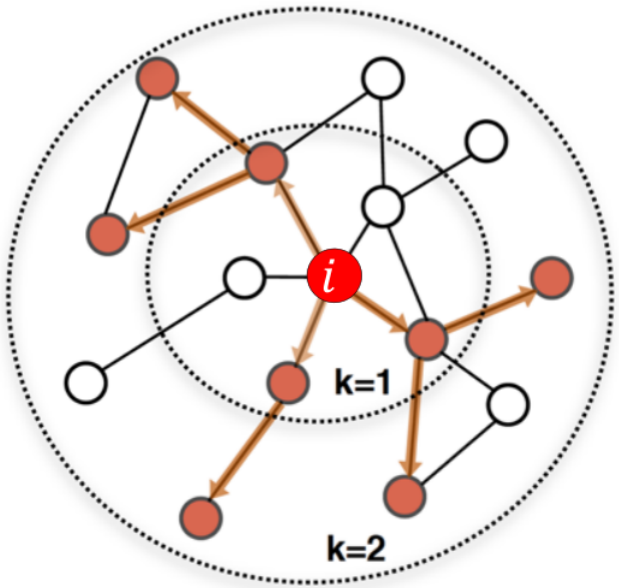
Image



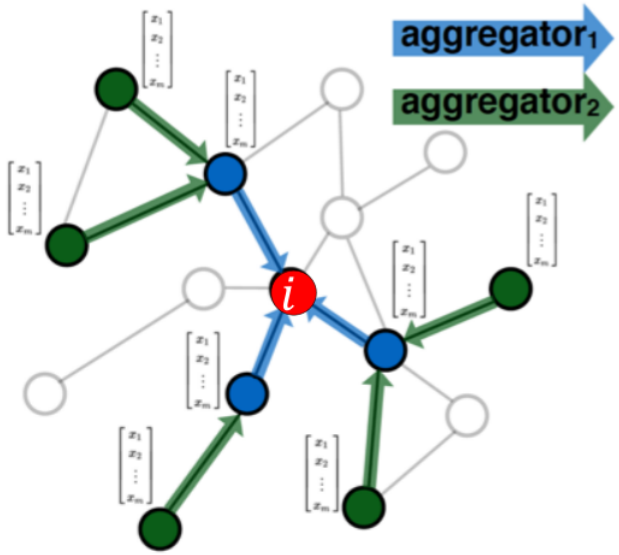
Graph

Graph Convolutional Networks

- Kipf & Welling, ICLR 2017
- le voisinage d'un nœud définit un "graphe de calcul", à l'aide duquel on va propager et transmettre l'information



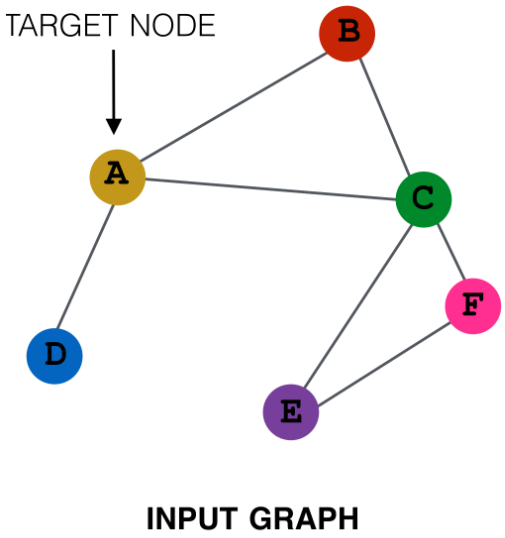
Determine node computation graph

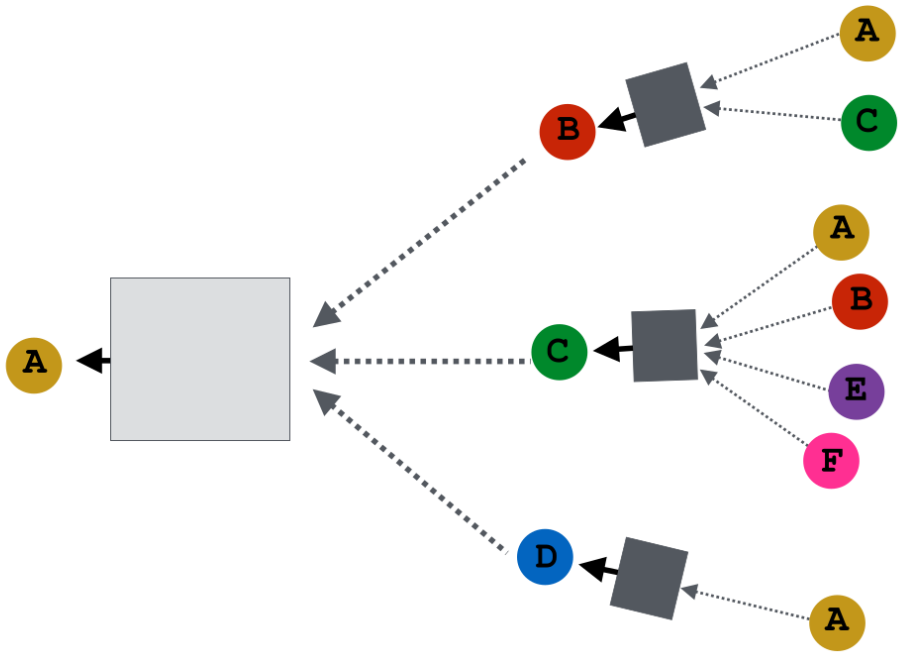


Propagate and transform information

GCN : agrégation de voisinage

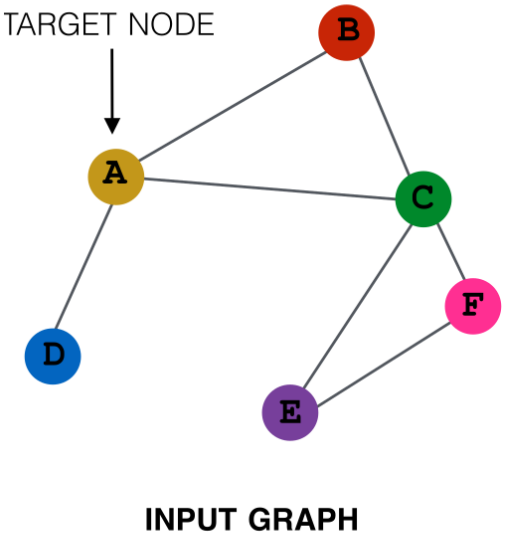
- Embeddings générés à partir des voisinages (localité)

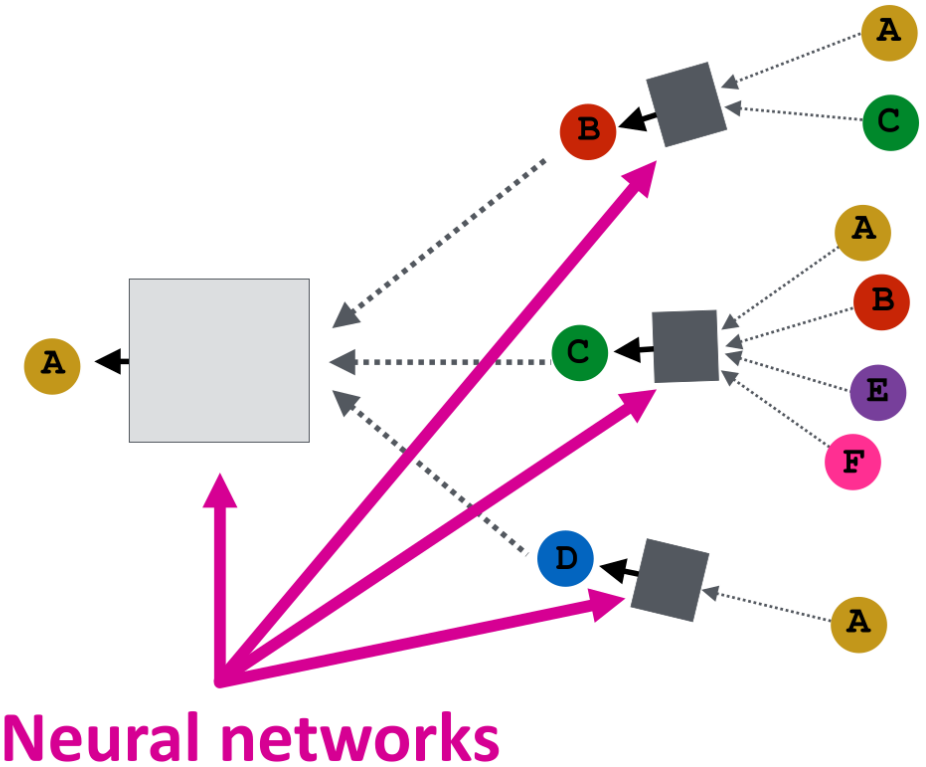




GCN : agrégation de voisinage

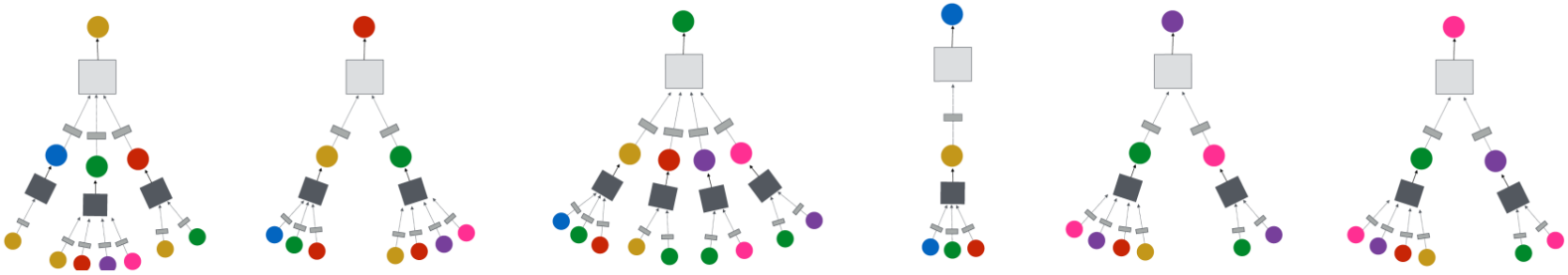
- Les nœuds agrègent l'information des voisins avec des réseaux de neurones





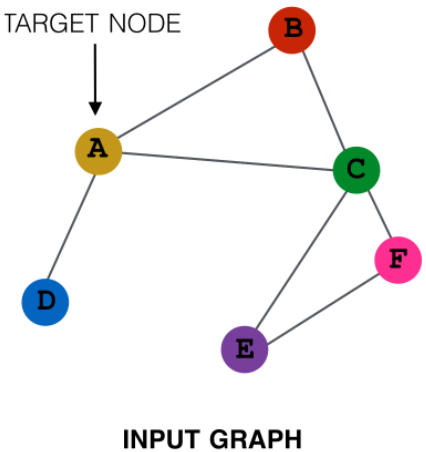
GCN : agrégation de voisinage

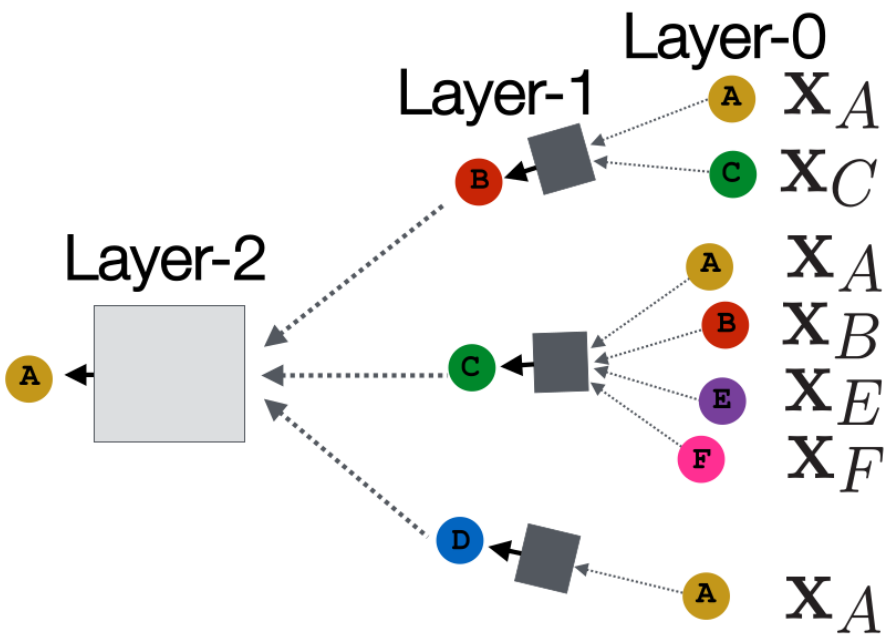
- Chaque nœud définit un ”graphe de calcul” (computation graph) en fonction de la structure de son voisinage



Nombre de couches

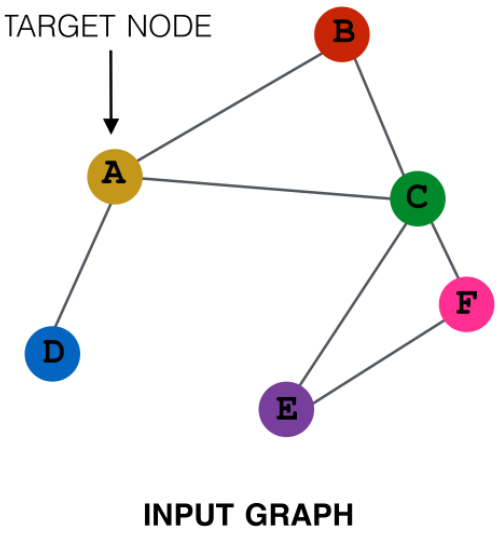
- Profondeur arbitraire (!)
  - les nœuds ont des embeddings à chaque couche
  - la couche 0 : les attributs d'entrée x_v
  - la k-ième couche : de l'information des sommets à distance k

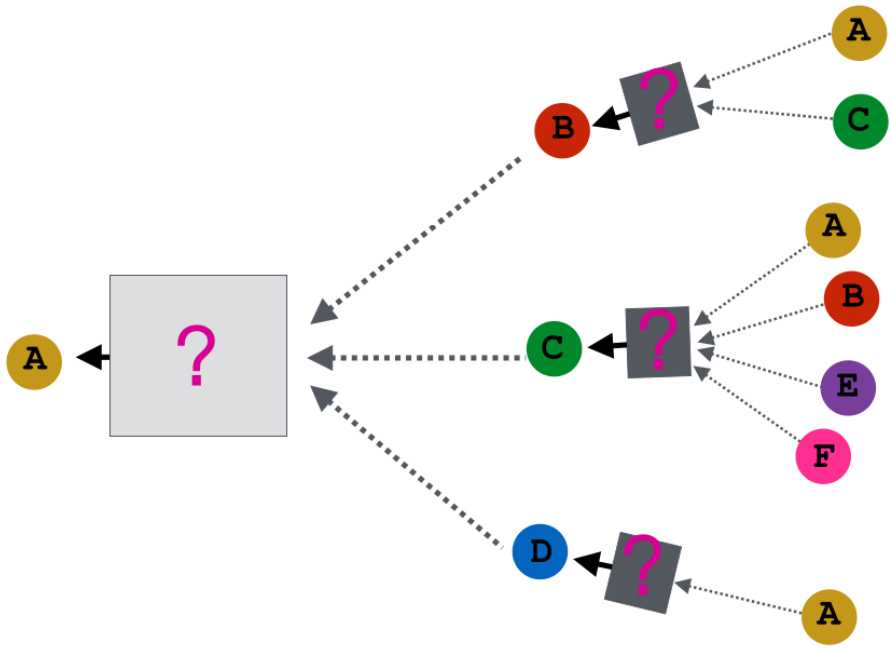




Quel réseau de neurones ?

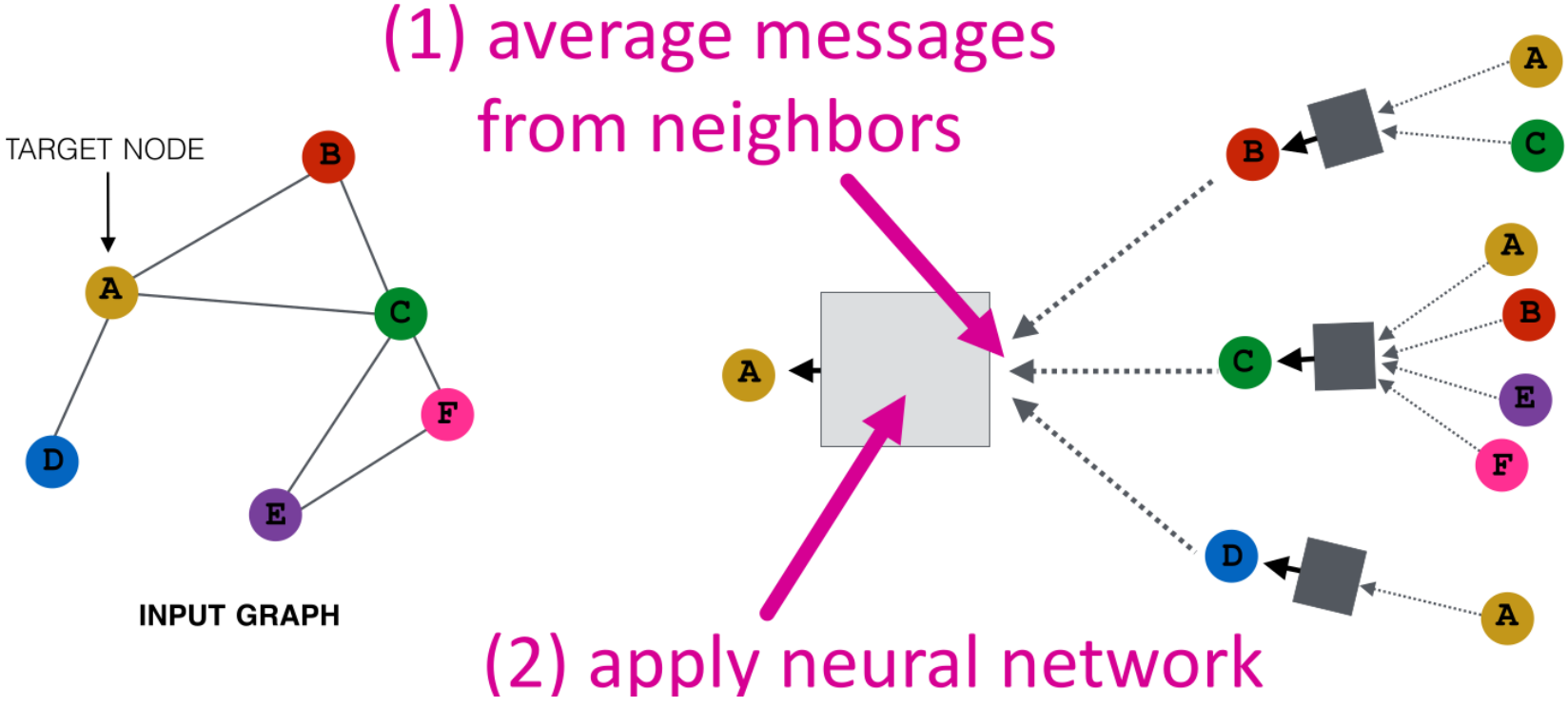
- Les approches diffèrent surtout dans la manière d'agréger l'information des différentes couches





Quel réseau de neurones ?

- Approche simple : une moyenne des voisins



Mathématisation

- $h_v^0 = x_v$
- $h_v^{l+1} = \sigma(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}), \qquad \forall l \in \{0, ..., L-1\}$
- $z_v = h_v^L$

- h_v^l est la représentation cachée de v pour la couche l
- W_l et B_l sont des matrices de poids (qu'on apprend). W pour l'agrégation du voisinage, B pour la transformation du vecteur lui-même
- on peut placer ces embeddings dans une fonction de loss et utiliser une SGD pour apprendre ces poids

Entraînement

Supervisé

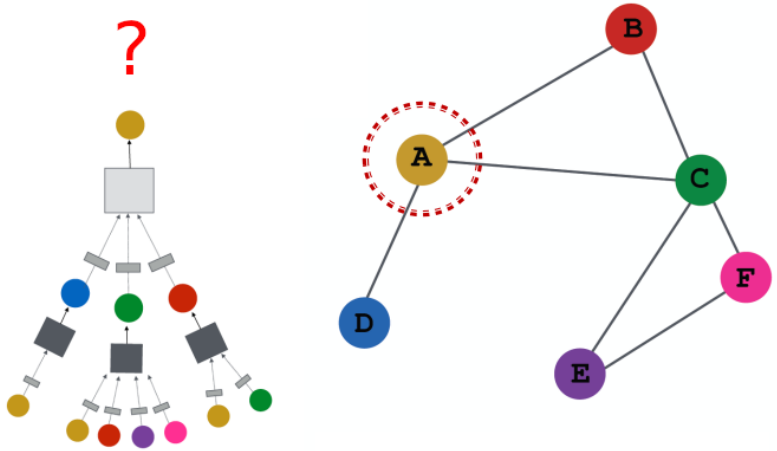
- on cherche à minimiser la loss $\mathscr{L}$:

$$\min_{\theta} \mathscr{L}(y, f(z_v))$$

- y est l'étiquette du nœud
- $\mathscr{L}$ peut être L2 si y est réel, Cross-entropy sinon :

$$\mathscr{L} = \sum_{v \in V} y_v log(\sigma(z_v^T \theta)) + (1 - y_v) log(1 - \sigma(z_v^T \theta))$$

On entraîne directement, par exemple pour de la classification



- on peut entraîner en non supervisé, en prenant une similarité structurelle comme label

Inductivité

- Les paramètres d'agrégation sont partagés entre les nœuds : le modèle a un nombre de paramètres sous-linéaire en $|V|$
- Capacité de généralisation aux nœuds non vus !
  - application : recommandation à des nouveaux utilisateurs !
- On peut même généraliser à des graphes entièrement "non vus"
  - à partir de graphes d'interaction de protéines d'un organisme A, on peut générer des embeddings à partir de données collectées pour l'organisme B

Résumé

- On définit une fonction d'agrégation de voisinage (on passe des "messages"
- On définit une loss sur les embeddings
- On s'entraîne sur un ensemble de sommets (ie : des graphes de calcul)
- On génère des embeddings pour les sommets... et même pour ceux que l'on n'a pas vu (à l'aide des embeddings/des valeurs dans les couches)

La suite

- la couche GNN en détail
- les connexions entre couches GNN
- l'entraînement de GNNs
- un peu de formalisme
- passage à l'échelle
- limites
- applications

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 31 mai 2021