Graph neural networks

GREYC – CNRS UMR 6072, University of Caen, ENSICAEN
Luc.Brun@ensicaen.fr

Sources

Sources used to build this lecture:

- My own work: [Brun, 2019]

  https://brunl01.users.greyc.fr/ARTICLES/presentationCAIP2019.pdf

- A survey by Zonghan Wu et al [Wu et al., 2019]:

  https://arxiv.org/abs/1901.00596

Applications

The same as structural pattern recognition:

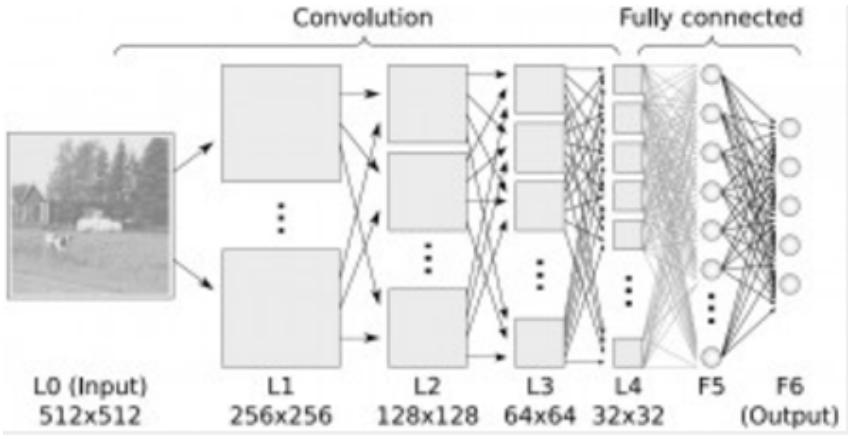
e-commerce : Find links between consumers and products (e recommandation),

chemistry : associate each graph describing a molecule to a property (chemical, physical, biological(drug discovery))

citation network : use citations between papers to classify them...

Why is it difficult ?

- A basic image neural network:



  Images of fixed sizes, fixed number of neighbors, unlabeled links, fixed neighborhoods

- We have thus to re define the following operations:
  1. Aggregation,
  2. Decimation,
  3. Pooling

Different types of GNN

Aggregation problem:

Recurrent graph neural networks (RecGNN):

Historically, the first type of GNN. Based on the idea of iterative message passing between neighbors.

Convolutional Graph neural networks (ConvGNN):

Generalizes the convolution operation defined on images. Each vertex aggregates information from its own features and the ones of its neighbors. Convolutional neural networks may be stacked (main diff. with RecGNN)

Different levels of predictions

Node prediction: predict a value/classify at a node level

- Semi supervised learning of nodes (Input: a graph with some labeled nodes, Output: A classification of unlabeled nodes).

Edge prediction: predict the existence/strength of an edge based on nodes hidden representations. (Semi supervised).

Graph level: Output related to the whole graph.

- Graph classification/regression. Usually requires a proper reduction of the input graphs(decimation,pooling).
- Graph embedding. No class required. May be based on auto encoders or on a regression of the edge strength.
- Graph metric: Learns a distance/similarity value between graphs. Should be combined with a classification/regression task.

Graph sequences: A sequence of graphs $G^{(t)} = (V, E, X^{(t)})$ with fixed topology but varying values has to be classified (gesture recognition, brain graphs, traffic speed forecasting,...)

Graph Agregation

- Associate to each vertex v an hidden variable h_v and aggregate local information.

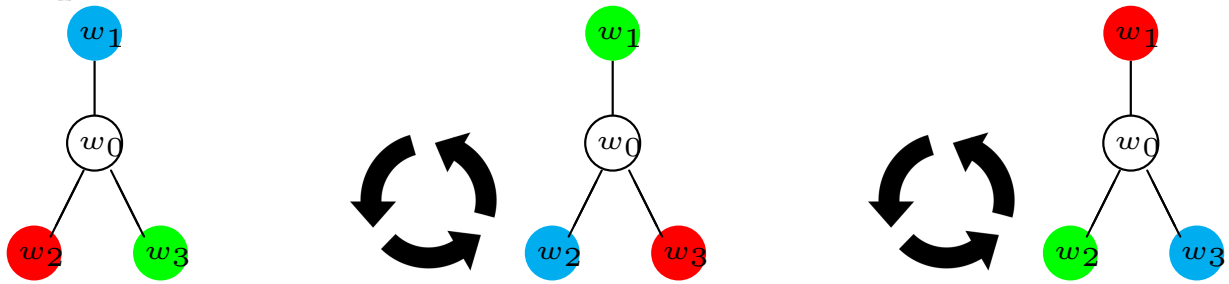
$$h_v^{(t)} = F(x_v, \{(x_u, x_{vu}^e, h_u^{(t-1)})\}_{u \in \mathcal{N}_v})$$

- Using images we learn $w_0 \ldots, w_8$:

w_5	w_1	w_6
w_3	w_0	w_4
w_7	w_2	w_8

  w_1 denotes the weigh of the pixel above the central pixel.

- Using graphs:



Without embedding nothing distinguishes the cyan,red and green neighbors.

How to become permutation invariant

- Message passing Framework (MPN): A framework to unify permutation invariant agregation functions:

$$\begin{cases} m_v^t = \sum_{w \in \mathcal{N}_v} M_t(h_v^{t-1}, h_w^{(t-1)}, e_{v,w}) \\ h_v^t = U_t(h_v^{(t-1)}, m_v^t) \\ y = R(\{h_v^T \mid v \in G\}). \end{cases}$$

  where $M_t(), U_t(), R()$ are learnable functions.
- May be summed up by a function f independent of the order of its arguments:

$$h_v^{(t)} = \sum_{u \in \mathcal{N}_v} f(x_v, x_u, x_{vu}^e, h_u^{(t-1)})$$

- A basic strategy common to Recurrent and convolutional graph neural networks.

Recurrent Graph Neural Networks

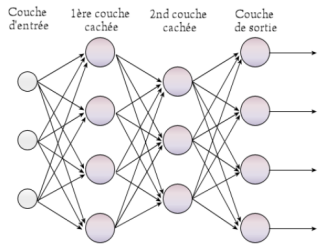
- Initially proposed for acyclic graphs:
- Basic idea: iterate an update of hidden values up to convergence:

$$h_v^{(t)} = \sum_{v' \in \mathcal{N}(v)} f(x_v, x_{v,v'}, x_{v'}, h_{v'}^{(t-1)})$$

- Each iteration should alternate:
  - Label propagation:
  - Gradient computation (using an appropriate loss function)
- function f may be:
  - An affine function [Scarselli et al., 2009],

$$f(l_v, l_{v,v'}, l_{v'}, h_{v'}^{(t-1)}) = A^{(l_v, l_{v,v'}, l_{v'})} h_{v'}^{(t-1)} + b^{(l_v, l_{v,v'}, l_{v'})}$$

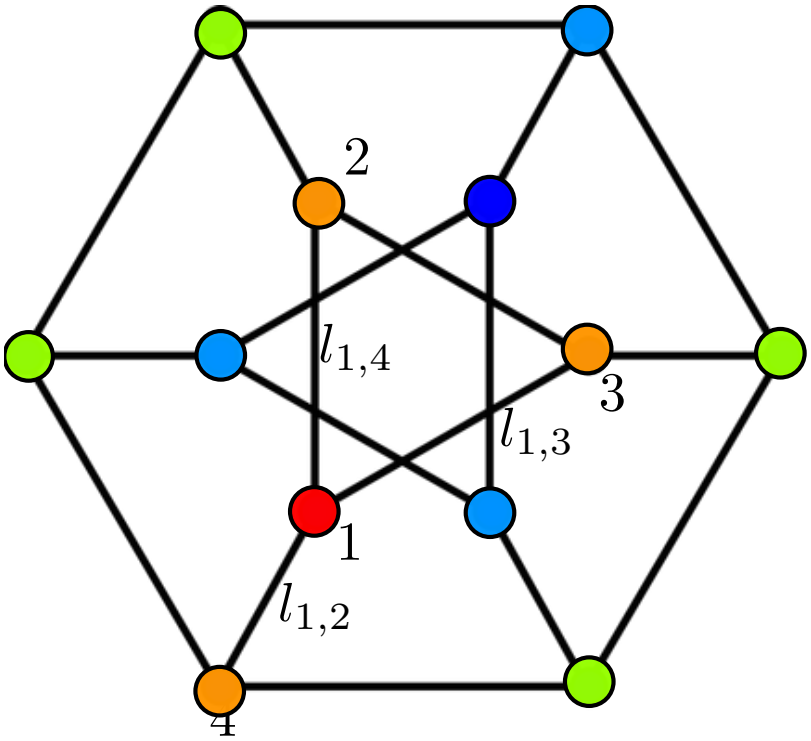
  - A MLP [Massa et al., 2006]



Par HRcommons — Travail personnel, Domaine public, https://commons.wikimedia.org/w/index.php?curid=11996182

- in any case f should be a contraction mapping to insure

Example



$t = 0$

$t = 1$

$t = 2$

$t = 3$

$t = 4$

$$\begin{cases} h_v^{(t)} & = & \sum_{u \in \mathcal{N}_v} f(x_v, x_u, x_{vu}^e, h_u^{t-1}) \\ o_v & = & g_w(h_v^T, l_v) \end{cases}$$

More complex agregation functions

- A long Short-term Memory [Hochreiter and Schmidhuber, 1997, Peng et al., 2017, Zayats and Ostendorf, 2018]
- A Gated Reccurent Unit [Li et al., 2016]

$$
\begin{aligned}
h_v^{(1)} &= [x_v^T, 0] && (1)\\
a_v^{(t)} &= A_v^T [h_1^{(t-1)\ T}, \ldots, h_{|V|}^{(t-1)\ T}]^T + b && (2)\\
z_v^t &= \sigma(W^z a_v^{(t)} + U^z h_v^{(t-1)}) && (3)\\
r_v^t &= \sigma(W^r a_v^{(t)} + U^r h_v^{(t-1)}) && (4)\\
\tilde{h}_v^{(t)} &= \tanh\left(W a_v^{(t)} + U\left(r_v^t \odot h_v^{(t-1)}\right)\right) && (5)\\
h_v^t &= (1 - z_v^t) \odot h_v^{(t-1)} + z_v^t \odot \tilde{h}_v^t && (6)
\end{aligned}
$$

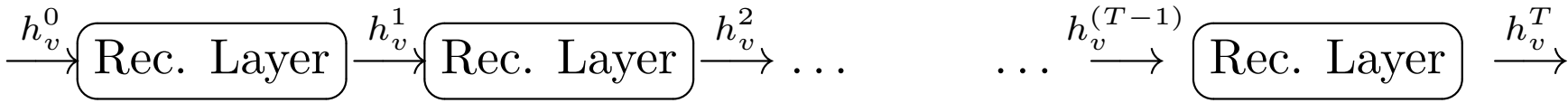
z_v^t: update gate, r_v^t: reset gate, A_v: weight by edges types.

- No more contraction constraint.
- uses the back-propagation through time (requires to store all intermediate states of all nodes).
- Learned weight by edge type:
  $a_v^{(t)} = \sum_{w \in \mathcal{N}(v)} A_{l_{v,w}} h_w^{(t-1)}$ [Gilmer et al., 2017]

Rec. and Conv. Graph neural networks

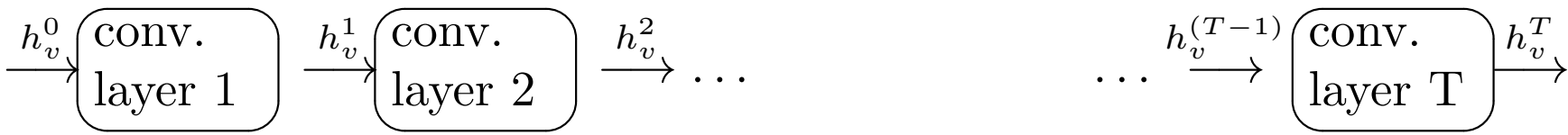
- Rec. Graph NN are iterative algorithms:

$$\begin{cases} m_v^t = \sum_{w \in \mathcal{N}_v} M(h_v^{t-1}, h_w^{(t-1)}, e_{v,w}) \\ h_v^t = U(h_v^{(t-1)}, m_v^t) \end{cases}$$



- Graph convolution is one shot:

$$\begin{cases} m_v^t = \sum_{w \in \mathcal{N}_v} M_t(h_v^{t-1}, h_w^{(t-1)}, e_{v,w}) \\ h_v^t = U_t(h_v^{(t-1)}, m_v^t) \end{cases}$$



Two famillies of methods

Spectral approach: Based on the Laplacian matrix. Transposes signal processing results onto the graph domain.

Spatial-based approach: Close from Rec. GNN. Based on "Hand made" learned filters.

The spectral approach

- Graph Laplacian:

$$L = D - A \text{ with } D_{ii} = \sum_{j=1}^{n} A_{ij}$$

A adjacency matrix of a graph G.

- Matrix L is real symmetric semi definite positive:

$$L = U \Lambda U^T$$

U orthogonal, Λ real(positive) diagonal matrix.

- A classical result from signal processing:

$$x * y = \mathcal{F}^{-1}(\hat{x}.\hat{y})$$

*: convolution operation, $\mathcal{F}^{-1}$ inverse Fourrier transform, $\hat{x}$ fourrier transform of x, '.' term by term multiplication.

The spectral approach

- If x is a signal on G, $\hat{x} = U^T x$ can be considered as its "Fourrier" transform. We have:

$$U\hat{x} = UU^T x = x$$

  U is thus the inverse Fourier transform.

- By analogy:

$$z * x = U(\hat{z} \odot \hat{x}) = U\left(U^T z \odot U^T x\right) = U\left(diag(U^T z)U^T x\right)$$

  $\odot$: Hadamard product.

- Let $g_\theta(\Lambda)$ be a diagonal matrix. The filtering of x by g_θ is:

$$y = U\left(g_\theta(\Lambda)U^T x\right) = \left(U g_\theta(\Lambda)U^T\right) x$$

The spectral approach

- If: $g_\theta(\Lambda) = \sum_{i=0}^{K-1} \theta_i \Lambda^i$. Then:

$$y = \left(U g_\theta(\Lambda) U^T\right) x = U \left(\sum_{d=0}^{K-1} \theta_d \Lambda^d \right) U^T x = \left(\sum_{d=0}^{K-1} \theta_d L^d \right) x$$

- One parameter per ring:
  - Lx : one step (direct) neighborhood,
  - $L^2 x$: two step neighborhood (idem for $L^3, L^4, \dots$)
- If multiple components:

$$y_j = \sigma \left(\sum_{i=1}^{f_{k-1}} \left(\sum_{d=0}^{K-1} \theta_d^{i,j} L^d \right) x_i \right) \quad \text{for } j = 1, \dots, f_k$$

- If $g_\theta = \Theta_{i,j}^{(k)}$ (layer k from component i to j) we obtain using previous notations:

$$h_j^{(k)} = \sigma \left(\sum_{i=1}^{f_{k-1}} U \Theta_{i,j}^{(k)} U^T h_i^{(k-1)} \right)$$

Chebyshev polynomials

- Problem: Computing L^i for $i \in \{0, \ldots, K-1\}$ is problematic for large matrices (SVD computation)
- Let us consider Chebyshev polynomial $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0 = 1$ and $T_1(x) = x$.

$$g_\theta(\Lambda) = \sum_{i=0}^{K-1} \theta_i \Lambda^i \rightarrow g_\theta(\Lambda) = \sum_{i=0}^{K-1} \theta_i T_i(\tilde{\Lambda})$$

  $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I \in [-1, 1:]$ the domain of Chebyshev polynomials.
- Considering $\tilde{L} = U\tilde{\Lambda}U^T = 2L/\lambda_{max} - I$ and $\tilde{x}_k = T_k(\tilde{L})x$ we have:

$$\tilde{x}_k = 2\tilde{L}\tilde{x}_{k-1} - \tilde{x}_{k-2}$$

  with $\tilde{x}_0 = x$ and $\tilde{x}_1 = \tilde{L}x$
- $\mathcal{O}(K|\mathcal{E}|)$ operations to get all $\tilde{x}_k$.

$$y = \sum_{d=0}^{K-1} \theta_d \tilde{x}_d$$

Cayley polynomials

- Chebyshef polynomials imposes to shrink the spectrum within $[-1, 1]$.
- If several eigenvalues are close it becomes difficult to explicit the influence of a given eigenvalue. More precisely, the number of Chebyshev coefficients required for approximating a filter having features in a given scale, is inverse proportional to the scale.
- Using Cayley transform $g(x) = \frac{x-i}{x+i}$ from $\mathbb{R}$ to $e^{i\mathbb{R}}/\{1\}$ we derive the Cayley polynomials:

$$g_{c,\gamma}(\lambda) = c_0 + 2Re\{\sum_{j=1}^{r} c_j \left(\frac{(\gamma\lambda - i)}{(\gamma\lambda + i)}\right)^j\}$$

  $\gamma > 0$ is the spectral zoom parameter, $i^2 = -1$.
- Given a graph with a Laplacian $L = U\Delta U^T$, and an input x, we have:

$$y = g_{c,\gamma}(\Delta)x = c_0 x + 2Re\{\sum_{j=1}^{r} c_j(\gamma\Delta - iI)^j(\gamma\Delta + iI)^{-j}x\}$$

First order approximation

- If $K = 2$, Chebyshev def. provides:

$$y = \sum_{d=0}^{K-1} \theta_d \tilde{x}_d = \theta_0 \tilde{x}_0 + \theta_1 \tilde{x}_1 = \theta_0 x + \theta_1 \tilde{L} x$$

- If we additionnaly suppose that $\lambda_{max} \approx 2$ then $\tilde{L} = L - I$:

$$y = \theta_0 x + \theta_1 (L - I)x = \theta_0 x + \theta_1 (I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} - I)x = \theta_0 x - \theta_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

- with $\theta_0 = -\theta_1 = \theta$ we obtain:

$$y = \theta \left(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x$$

First order approximation

- To improve stability $I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$. So:

$$y = \theta \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} x$$

  with $\tilde{A} = A + I, \tilde{D}_{i,i} = \sum_j \tilde{A}_{i,j}$.
- Generalization to multi-components:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$$

  where $X \in \mathbb{R}^{N \times C}, \Theta \in \mathbb{R}^{C \times F}$.
- Simpler (more efficient) convolution.
- Additional hops may be obtained by iterating convolutions.

Spatial approaches

- Mimics image convolution.
- Each vertex receives a weighted sum of the values of its neighbors (without any reference to Fourrier).
  - How to define weighs,
  - how to map a specific weight to a specific neighbors

  using spectral convolution these tasks are defined by the Laplacian.
- Let us note that GCN [Kipf and Welling, 2017] may be interpreted as a transition between spectral and spatial approaches. Indeed:

$$y = \theta \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} x \Rightarrow y_v = \theta \sum_{u \in \mathcal{N}_v \cup \{v\}} (\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}})_{v,u} x_u$$

First methods

- NN4G [Micheli, 2009]

$$\begin{aligned} h_v^1 &= f(W_1 x_v) \\ h_v^k &= f\left(W_1 x_v + W_2 \sum_{u \in \mathcal{N}_v} h_u^{(k-1)}\right) \end{aligned}$$

- Re formulated in terms of Graph Markov models [Bacciu et al., 2018]

- DCNN [Atwood and Towsley, 2016]: Let $P = D^{-1}A$ be a transition probability matrix :
  - Node features propagation:

$$h^k = W^k \odot P^k x \, ; \, y = f\left(||_{j=0}^{r} h^j\right)$$

    $||$: concatenation
  - An other possibility:

$$y = \sum_{j=0}^{r} f\left(P^k X W^{(k)}\right)$$

Different neighborhoods

- PGC-DGCNN [Tran et al., 2018]: Let S such that $S^{(j)}_{v,u} = 1$ iff there is a shortest path of length j between v and u.

$$h^k = ||_{j=0}^{r} f\left((\tilde{D}^{(j)})^{-1} S^{(j)} h^{(k-1)} W^{(j,k)}\right)$$

with $\tilde{D}^{(j)}_{ii} = \sum_l S^{(j)}_{i,l}$
Connect "easily" distant vertices.

- PGC [Yan et al., 2018]. Let us suppose that any neighborhood $\mathcal{N}_v$ is partitioned into K clusters $\mathcal{N}_v^0, \ldots, \mathcal{N}_v^{K-1}$:

$$h^k = \sum_{j=0}^{K-1} \overline{A}^{(j)} h^{(k-1)} W^{(j,k)},$$

with $\overline{A}^{(j)} = (\tilde{D}^{(j)})^{-\frac{1}{2}} \tilde{A}^{(j)} (\tilde{D}^{(j)})^{-\frac{1}{2}}$, $\tilde{A}^{(j)} = A^{(j)} + I, h^0 = x$.
Allows to insert *a priori* information in the convolution process.

Using MPN notations

- A familly of methods following the update rules:

$$\begin{cases} m_v^t = \sum_{u \in \mathcal{N}_v} M_t(h_v^{t-1}, h_w^{(t-1)}, e_{v,w}) \\ h_v^t = U_t(h_v^{(t-1)}, m_v^t) \\ y = R(\{h_v^T \mid v \in G\}). \end{cases}$$

- [Duvenaud et al., 2015]:
$M(h_v, h_w, e_{v,w}) = h_w || e_{v,w}, U_t(h_v^{(t-1)}, m_v^t) = \sigma(H_t^{deg(v)} m_v^t)$ where $H_t^{deg(v)}$ is a learned matrix. $R = f(\sum_{v,t} softmax(W_t h_v^t))$.
- [Li et al., 2016] (see slide 12), $M = A_{e_{v,w}} h_w^{(t-1)}$, $U = GRU(h_v^{(t-1)}, m_v^t)$ and $R = \sum_{v \in V} \sigma(i(h_v^{(T)}, h_v^0)) \odot (j(h_v^{(T)}))$ where i and j are neural networks and $\odot$ the Hadamar product.

Message Passing Networks

- [Battaglia et al., 2016]
  $M = i(h_v||h_w||e_{v,w}), U = j(h_v||x_v||m_v), R = f(\sum_v h_v^T)$ where $||$ is the concatenation, i, j, f are neural networks, x_v is an external feature of v
- [Kearnes et al., 2016] $M(h_v^{(t-1)}, h_w^{(t-1)}, e_{v,w}^{(t-1)}) = e_{v,w}^t = \alpha(W_4(\alpha(W_2 e_{v,w}^{(t-1)})||\alpha(W_3(h_v^{(t-1)}||h_w^{(t-1)}))))$,
  $U(h_v^{(t-1)}, m_v^t) = \alpha(W_1(\alpha(W_0 h_v^{(t-1)})||m_v^t))$. α ReLU function, $W_0, \ldots, W_4$ learned weighs matrices, $e_{v,w}^t$ learned edge representation.
- [Schütt et al., 2017]:

$$M = tanh\left(W^{fc}((W^{cf} h_w^{(t-1)} + b1) \odot (W^{df} e_{v,w} + b2))\right)$$

  W^{cf}, W^{df}, b_1, b_2 learned matrices and biases.
  $U(h_v^{(t-1}, m_v^t) = h_v^{(t-1)} + m_v^t, R = \sum_v NN(h_v^T)$.

- [Kipf and Welling, 2017] $M = \frac{A_{vw}}{\sqrt{deg(v)deg(w)}}, U = RELU(Wm_v^t)$,
- [Gilmer et al., 2017] (see slide 12) $M = A_{l_{v,w}} h_w^{(t-1)}$
- [Simonovsky and Komodakis, 2017]

$$M = \frac{1}{|\mathcal{N}_v|} \sum_{u \in \mathcal{N}_v} F_\theta(l_{v,u}) h^{(t-1)} + b$$

F: Parametric function of θ which associates one weigh to each edge label $l_{v,u}$.

- Laplacian based methods: $M = C_{v,w}^t h_w^{(t-1)}$ where C^t is a matrix based on the Laplacian, $U = \sigma(m_v^t)$.

Graph attention network

- [Verma et al., 2017]:

$$y_v = \frac{1}{|\mathcal{N}_v|} \sum_{m=1}^{M} \sum_{u \in \mathcal{N}_v} q_m(u, v) W_m u + b$$

$q_m(.,.)$ m^{th} learned soft-assignment function. W_m weight matrix:

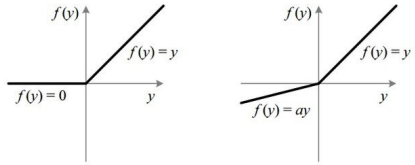
$$q_m(u, v) \propto exp(\alpha_m^T u + \beta_m^T v + c_m),$$

with $\sum_{m=1}^{M} q_m(u, v) = 1$

Graph attention Networks

- Not all neighbors have a same importance for update:

$$\alpha_{v,v'}^t = softmax_{v'}(e_{v,v'}^t) = \frac{exp(e_{v,v'}^t)}{\sum_{v'' \in \mathcal{N}_i} exp(e_{v,v''}^t)}$$



- With : $e_{v,v'}^t = LeakyReLU(a^T[W^t h_v || W^t h_{v'}])$
  a, W : learnable weight vector and matrix.
- Update rule:

$$h_v^t = \sigma(\sum_{u \in \mathcal{N}_v} \alpha_{v,u} W^t h_u^{(t-1)})$$

- With K features:

$$h_v^t = ||_{k=1}^K \sigma(\sum_{u \in \mathcal{N}_v} \alpha_{v,u}^k W^t h_u^{(t-1)})$$

Graph attention network

- Same basic idea than [Velickovic et al., 2018] but do not use a global attention for the edge.Uses instead the notion of (K) heads.

$$\begin{cases} h_v^t & = & FC_{\theta_0}(h_v^{(t-1)}|| \left(||_{k=1}^{K} \sum_{u \in \mathcal{N}_v} w_{u,v}^{(k)} FC_{\theta_V^{(k)}}^{h}(h_v^{(t-1)}) \right)), \\ w_{v,u}^{(k)} & = & \frac{e^{\varphi_w^{(k)}(v,u)}}{\sum_{u \in \mathcal{N}_v} e^{\varphi_w^{(k)}(v,u)}}, \\ \varphi_w^{(k)}(v,u) & = & < FC_{\theta_{xa}^{(k)}}(v), FC_{\theta_{za}^{(k)}}(u) > . \end{cases}$$

where $FC_{\theta_{xa}^{(k)}}, FC_{\theta_{za}^{(k)}}$ and FC_{θ_V} full connected layers for the query, the key and the value.

Spectral/Spatial approaches

Property	spectral	spatial
Localized filter	✓	✓
Theoretical background on signal processing on graphs	✓	
Individual node weighting	ring	✓
Can be applied to different graphs	Not really	✓
Efficiency/modularity		✓
Adatative to different types of graphs		✓

Generalization of Spectral/Spatial approaches

Definition

Graph Shift Operators

A matrix $S \in \mathbb{R}^{n \times n}$ is called a Graph shift operator (GSO) if it satisfies:

$$i \neq j \text{ and } (i,j) \notin E \Rightarrow S_{ij} = 0$$

Aggregation is then performed using:

$$h^t = \sigma(Sh^{t-1}W)$$

where S is a GSO, h our vector of features, W the projection matrix.

GSO cover all one hop spectral and spatial approaches

GSO: A proposal

Proposed by [Dasoulas et al., 2021].
Let $A \in \mathbb{R}^{n \times n}$ be the adjacency matrix of a graph G and $\mathcal{S}$ a set of parameter. The GSO $\gamma(A, \mathcal{S})$ is defined by:

$$\gamma(A, \mathcal{S}) = m_1 D_a^{e_1} + m_2 D_a^{e_2} A_a D_a^{e_3} + m3 I_n$$

where $A_a = A + aI_n$ and $D_a = diag(A_a 1_n)$ is the diagonal matrix of A_a. The set of parameters is defined by $\mathcal{S} = (m_1, m_2, m_3, e_1, e_2, e_3, a)$

	S							
Convolution	m_1	m_2	m_3	e_1	e_2	e_3	a	$\gamma(A, \mathcal{S})$
GCN	0	1	0	0	$-\frac{1}{2}$	$-\frac{1}{2}$	1	$D_1^{-\frac{1}{2}} A_1 D_1^{-\frac{1}{2}}$
GIN	0	1	0	0	0	0	$1+\epsilon$	$A + (1+\epsilon) I_n$
$DCNN$	0	1	0	0	-1	0	0	$D^{-1} A$

Theorem: $\gamma(A, S)$ has real eigenvalues and a set of real eigenvectors.

Limits of Graph aggregation methods

Oversmoothing: Over iterations (successive layers), no feature h^l tends to become independent of h^0 and hence only captures rough structural information.

Oversquashing: Node feature of a node i: is almost insentitive to the one of a distant node j.

Oversmoothing

- Dirichlet : One measure of oversmoothing:

$$E(f) = f^T L f = \frac{1}{2} \sum_{i,j} w_{ij} (f_i - f_j)^2 \text{ with } L = D - A, D = A1$$

where $A = (w_{ij})$ and f encodes a scalar value for each node.

$$\tilde{E}(f) = f^T \tilde{L} f = \frac{1}{2} \sum_{i,j} w_{ij} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2 \text{ with } \tilde{L} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

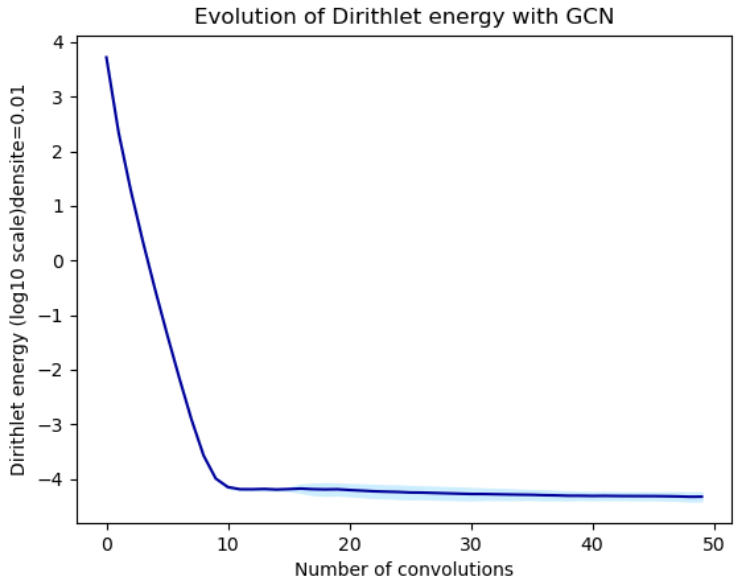
Extended to arbitrary dimension by:

$$\tilde{E}(X) = tr(X^T \tilde{L} X) = \frac{1}{2} \sum_{i,j} w_{ij} \| \frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}} \|^2$$

E (or $\tilde{E}$) tends to decrease up to a certain point.

Oversmoothing

- Example with $Conv = D^{-\frac{1}{2}}(A+I)AD^{-\frac{1}{2}}$, ($D$ degree matrix of $A+I$). Statistics computed over 50 graphs of 2000 nodes.



Oversmoothing: What is happening ?

Let $C = \gamma(A, \mathcal{S})$ denote a GSO. If we neglect non linear functions, GCN iterates :

$$h^{l+1} = Ch^l$$

is similar to the power iteration method:

$$h^{l+1} = \frac{Ch^l}{\|Ch^l\|}$$

and converges (up to a normalization) to the eigenvector associated to the largest eigenvalue of C (and is thus independent of h^0).

- Conclusion: A GNN do not converge toward a same value for all node, but toward a value for each node which is independent of the initial node values.

Oversmoothing: explicit convergence

- Let us consider a normalized GSO: $\tilde{C} = D^{-\frac{1}{2}} C D^{-\frac{1}{2}}$ with $C = \gamma(A, \mathcal{S})$ and
- D the degree matrix of C.
- Let us additionally consider the Laplacian matrix $\tilde{L} = I - \tilde{C}$. $\tilde{L}$ being normalized its eigenvalues belong to $[0, 2]$.

Let v denote an eigenvector of $\tilde{L}$ we have:

$$\tilde{C}v = (I - \tilde{L})v = v - \lambda v = (1 - \lambda)v$$

Since $\lambda \in [0, 2]$(0 reached) the highest eigenvalue of $\tilde{C}$ is equal to 1 and its associated eigenvector is the eigenvector associated to 0 in $\tilde{L}$.

Oversmoothing: explicit convergence

- Let us additionally suppose that the graph is connected according to C. Then the eigenvalue 0 of $\tilde{L}$ has multiplicity 1.
- Let us consider $L = D - C$ whose eigenvector associated to 0 is 1 and $v = D^{\frac{1}{2}}1$,

we have:

$$\tilde{L}v = \left(D^{-\frac{1}{2}}LD^{-\frac{1}{2}}\right)D^{\frac{1}{2}}1 = D^{-\frac{1}{2}}L1 = 0$$

The eigenvector of $\tilde{L}$ associated to 0 and hence the eigenvector of $\tilde{C}$ associated to 1 is equal to $v = D^{\frac{1}{2}}1$.

- Conclusion: All GCN without non linear operations converge to $D^{\frac{1}{2}}1$ independently of the input nodes' feature.
- Example: if $C = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}$, $v_i = \sqrt{d_i + 1}$, d_i: degree of node i.

Oversquashing

The influence of one node over an other node decay exponentially with their distance. More precisely, if :

$$h^{l+1} = \sigma(W^t C h)$$

where σ is a non linear function with a c_σ Lipschitz coefficient, $C = c_r I + c_a A = \gamma(A, \mathcal{S})$ with $\mathcal{S} = (0, c_a, c_r, 0, 0, 0, 0)$ is a GSO and $W^t \in \mathbb{R}^{p \times p}$ is a learnable weight matrix with a maximal value w. Then we have [Giovanni et al., 2023]:

$$\left\| \frac{\partial h_v^{(m)}}{\partial h_u^{(0)}} \right\|_{L_1} \leq (c_\sigma w p)^m (C^m)_{u,v}$$

- $(C^m)_{u,v}$: number of walks in C of length m between u and v.

$$\frac{\partial h_v^{(m)}}{\partial h_u^{(0)}} = 0 \text{ If } m \leq d_C(u, v)$$

- Increasing $c_s \sigma, w, p$ may decrease the oversquashing effect but it acts globally.

oversquashing: What happen when information is merged ?

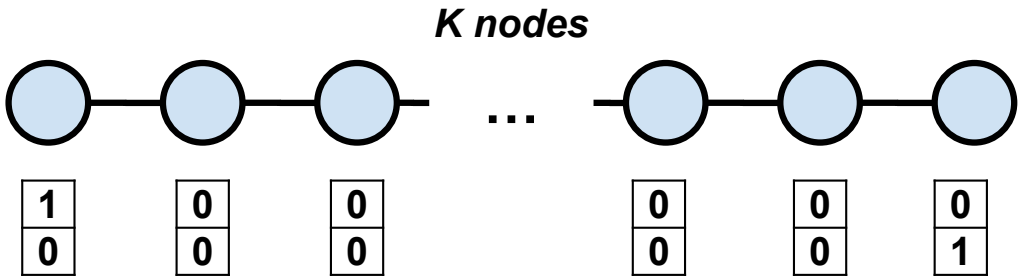
If $c_a \leq 1$ and $d(u,v) = r$ let $\gamma_l(u,v)$ denotes the number of walks of maximal length l between u and v. For any $0 \leq k < r$, it exists $C_k > 0$ independent of r and the graph, such that:

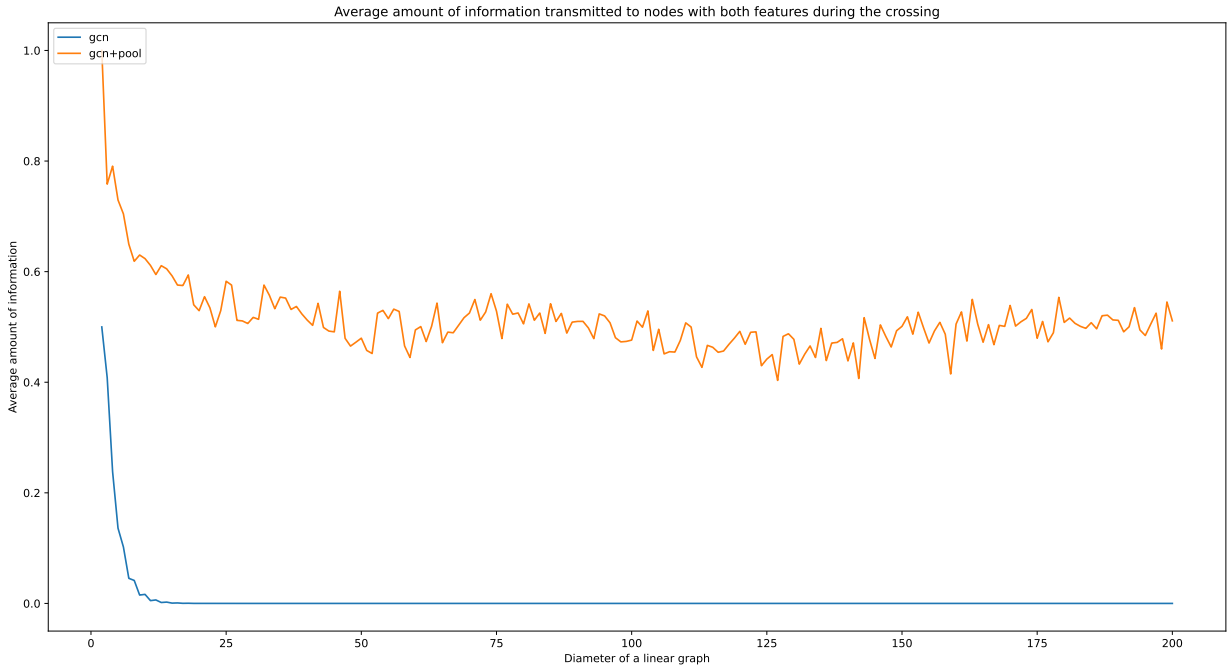
$$\left\| \frac{\partial h_v^{(m)}}{\partial h_u^{(0)}} \right\|_{L_1} \leq C_k \gamma_{r+k}(u,v) \left(\frac{2 c_\sigma w p}{d_{min}} \right)^r$$

where d_{min} is the minimal degree of G.

- If $2 c_\sigma w p < d_{min}$ we get an exponential decrease.

oversquashing: Experiments





Graph clustering

- Decimation is often considered as a graph clustering problem [Dhillon et al., 2007].
- Several criteria with:

$$link(A,B) = \sum_{u \in A, v \in B} w_{u,v} \; deg(A) = link(A,V)$$

Ratio association: maximize within links.

$$Rassoc(G) = \max_{\mathcal{V}_1,\ldots,\mathcal{V}_k} \sum_{c=1}^{k} \frac{links(\mathcal{V}_c,\mathcal{V}_c)}{|\mathcal{V}_c|}$$

Ratio cut: minimize links between clusters

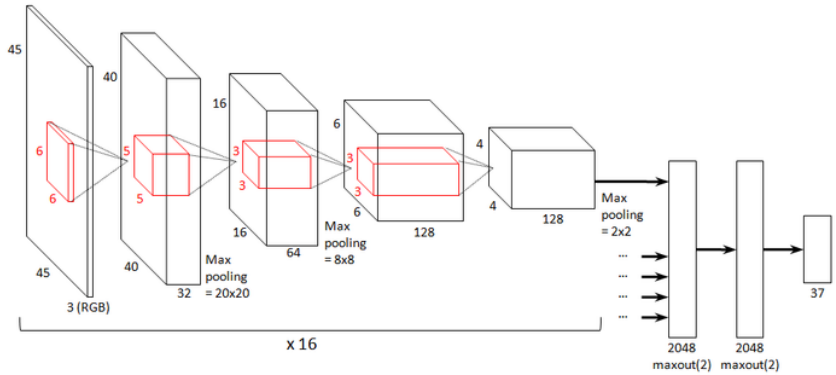
$$RCut(G) = \min_{\mathcal{V}_1,\ldots,\mathcal{V}_k} \sum_{c=1}^{k} \frac{links(\mathcal{V}_c,V-\mathcal{V}_c)}{|\mathcal{V}_c|}$$

Kernighan-Lin approx. same than Ration cut with clusters of identical size.

$$KLObj(G) = \min_{\mathcal{V}_1,\ldots,\mathcal{V}_k} \sum_{c=1}^{k} \frac{links(\mathcal{V}_c,V-\mathcal{V}_c)}{|\mathcal{V}_c|} \text{ with } |\mathcal{V}_c| \approx \frac{V}{k}$$

Graph Decimation vs Graph clustering

- Is it a clustering or a sub sampling problem ?



- Decimation is related to sub-sampling.
- Sub-sampling and clustering are two different problems.

On the need of sub sampling

- Given a $1D$ signal $x(t)$ with a maximal frequency w. Then x can be sampled at the sampling rate f_s if $f_s > 2w$ (Nyquist-Shanon theorem).
- Let us consider a low pass filter g applied n times and the resulting signal $x_n(t)$ of maximal frequency w_n.
  - w_n is decreasing,
  - if x_n conserve the same sampling rate f_s, we have $f_s >> 2w_n$ and the signal contain redundant information.

Few words from Graph Signal Processing(GSP)

Let G be a simple, connected, undirected graph whose normalized Laplacian's decomposition is given by $L = U\Lambda U^T$ where

$$\Lambda = diag(\lambda_1, \ldots, \lambda_N), 0 = \lambda_1 \leq \lambda_2, \ldots, \leq \lambda_n \leq 2.$$

The Graph Fourrier Transform (GFT) of a signal f on G is given by $\tilde{f} = U^T f$ (page 16).
A w-band limited signal on G is defined to have 0 GFT coefficients above w, i.e. its spectral support is limited to $[0, w]$.

Definition (Paley-Wiener space)

The space of all w band limited signals known as the Paley-Wiener *space is denoted $PW_w(G)$. If $\{u_1, \ldots, u_N\}$ denotes the columns of U, we trivially have*

$$PW_w(G) = span(\{u_1, \ldots, u_i\}) \text{ with } \lambda_i \leq w.$$

If $w < \lambda_2$ we have $PW_w(G) = \mathbb{R}u_1$ which corresponds to the set of constant signals on the graph.

Definition (Uniqueness set)

A subset of nodes $S \subset \mathcal{V}$ is a uniqueness set for the space $PW_w(G)$, if for any two signals from $PW_w(G)$, the fact that they coincide on S implies that they coincide on $\mathcal{V}$, i.e.:

$$\forall f, g \in PW_w(G), \quad f(S) = g(S) \Rightarrow f = g$$

For $w < \lambda_2$, any subset of $\mathcal{V}$ is a uniqueness subset of $PW_w(G)$.

Lemma

S is a uniqueness set for a signals in $PW_w(G)$ iff $PW_w(G) \cap L_2(S^c) = \{0\}$, where :

$$L_2(S^c) = \{\phi \in \mathbb{R}^N, \phi(S) = 0\}$$

is the space of all graph signals that are zero everywhere except possibly on the subset of nodes S^c.

Proof.

See [Anis et al., 2014].

Definition (Graph downsampling)

S is an allowed down-sampling for a signal f on G, if it exists w such that $f \in PW_w(G)$ and S is a uniqueness set for $PW_w(G)$.

Let us consider the low pass filter g_n which cuts off (set to zeros) the n highest eigenvalues of $\mathcal{L}$. The filtered version of f, denoted f_n is equal to $f_n = U g_n(\Lambda) U^T f$. We have:

Proposition

For any signal f and for any uniqueness set S of a set $PW_w(G)$, it exists a filtered version f_n of f such that S is an allowed down-sampling for f_n.

Proof.

Let us consider $n = N - dim(PW_w(G))$. We have by definition of g_n, $f_n \in span(u_1, \ldots, u_{dim(PW_w(G))})$, hence $f_n \in PW_w(G)$. □

Graph downsampling: Discussion

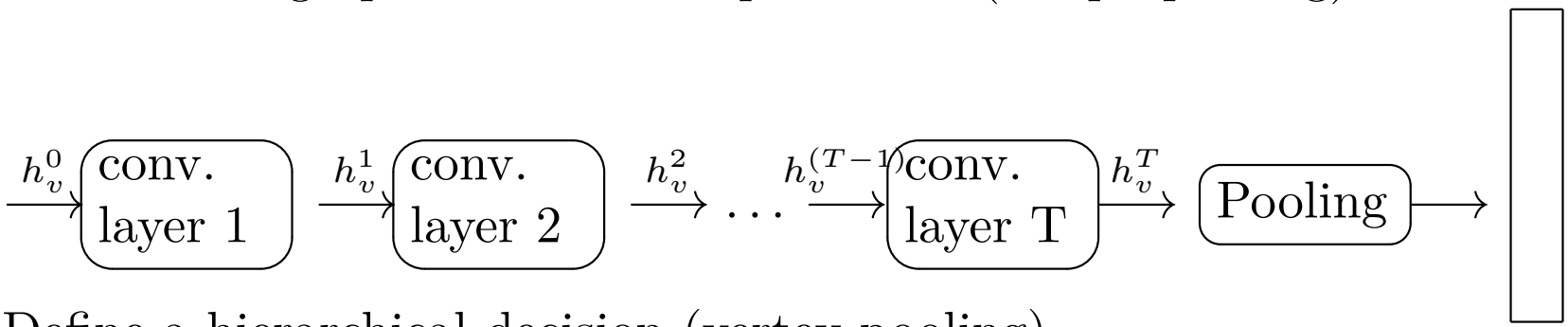
For any $S \subset V$ and any $f \in \mathbb{R}^N$, a sufficient low pass filtering of f "forces" the set S to become an allowed down-sampling for a filtered version f_n of f.
Intuitively, f_n contains a sufficient amount of redundant information to be described by the reduced set of vertices S.
At the extreme, using $n = N - 1$, any subset $S \subset \mathcal{V}$ is an allowed down-sampling. It corresponds to the case where f has been filtered up to a constant signal.

Let us consider a dataset $\mathcal{G}$ of graphs with a constant value (e.g. a set of molecular graphs encoding alkanes).

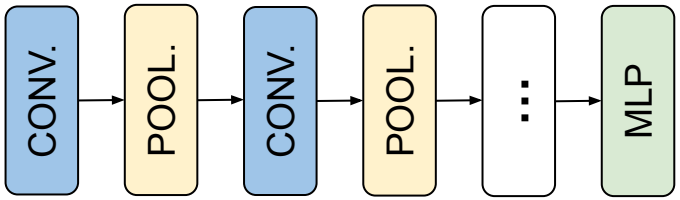
- The above results states that **the signal** on each graph can be encoded by a single vertex.
- **It is wrong** to conclude that each graph can be reduced to a single vertex. The information is in this case in the structure and not in the content.

Graph pooling

- Finalize the graph classification procedure (Graph pooling):



- Define a hierarchical decision (vertex pooling)



Graph pooling

- Attach a value to a surviving vertex based on its reduction window
- Classical methods:

$$h_v^t = mean/max/sum_{u \in RW(v)} h_u^{(t-1)}$$

  May also be used for final decision.
- [Zhang et al., 2018]:
  - concatenate all convolution results: $h_v^{1:K} = ||_{t=1}^{K} h_v^t$,
  - Identifies identical vertex feature's vector, sort them from right (K) to left (1).
  - Cut the number of vertices to a pre defined number q (or span with 0 vertices if $|V| < q$).
  - Transform the resulting $q \times F$ matrix into a 1D vector finish the work with 1D convolutional layers.

Vertex Pooling

- Usually defined through a matrix $S^l \in \mathbb{R}^{n_l \times n_{l+1}}$ which encodes the attachment of the vertices of G_l onto the one of G_{l+1}.

$$\left\{ \begin{array}{lcll} h^{l+1} & = & (S^l)^T h^l & \in \mathbb{R}^{n_{l+1} \times d} \\ A^{l+1} & = & (S^l)^T A^l S^l & \in \mathbb{R}^{n_{l+1} \times n_{l+1}} \end{array} \right.$$

- For example:

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

- We have:

$$(S^{(l)\top} A^{(l)} S^{(l)})_{i,j} = \sum_{k,m}^{n_l} A^{(l)}_{k,m} S^{(l)}_{k,i} S^{(l)}_{m,j}$$

i is adjacent to j in G_{l+1} iff: ?

DIFFPOOL

- Method proposed by [Ying et al., 2018]
- S^l is learned at each layer (through a GNN):

$$S^l = softmax(GNN_{l,pool}(A^l, h^l))$$

  the softmax being applied row-wize.
- A^{l+1} defines a complete graph and S^{l+1} has no constraints to define clear assignments.
- The authors proposed to add a penalization cost:

$$||A^l - S^l(S^l)^T||_F + \frac{1}{n_{l+1}} \sum_{i=1}^{n_{l+1}} H(S_i^l)$$

  where H is a measure of entropy and S_i^l is the i^{th} row of S^{l+1}.
  $||A^l - S^l(S^l)^T||_F$ pushes toward highly connected clusters.
  $\frac{1}{n_{l+1}} \sum_{i=1}^{n_{l+1}} H(S_i^l)$ pushes S^l toward a binary matrix.

DIFFPOOL

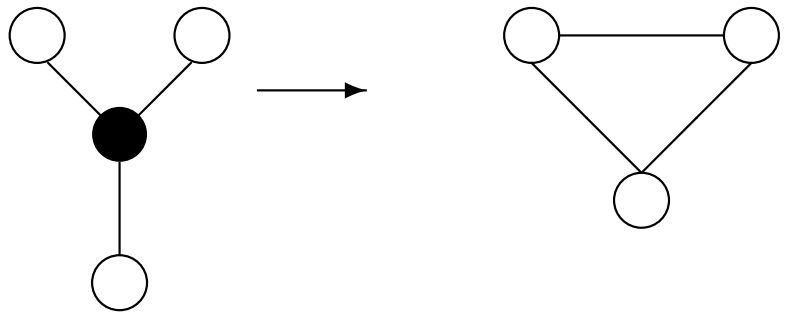
- Learning the decimation S^l is a good idea.
- S^l is not sparse and thus $A^{(l+1)}$ is almost complete.
- Learning S^l imposes to use graphs of fixed size.

Top K methods

- Select a given amount (k) of vertices:

$$S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \text{ selection of 1,3,4}$$

- The formula $A^{(l+1)} = (S^l)^T A^l S^l$ may provide disconnected graphs.
- We may use the Kron reduction [Bianchi et al., 2022]. Connects all pairs of surviving vertices adjacent to a same removed vertex.



Top K methods

- A score is assigned to each vertex of the graph.
- The vertices with the Top-k highest score are selected.

Projection:

$$score(v) = \frac{< p, h_v^t >}{\|p\|}$$

GNNPool:

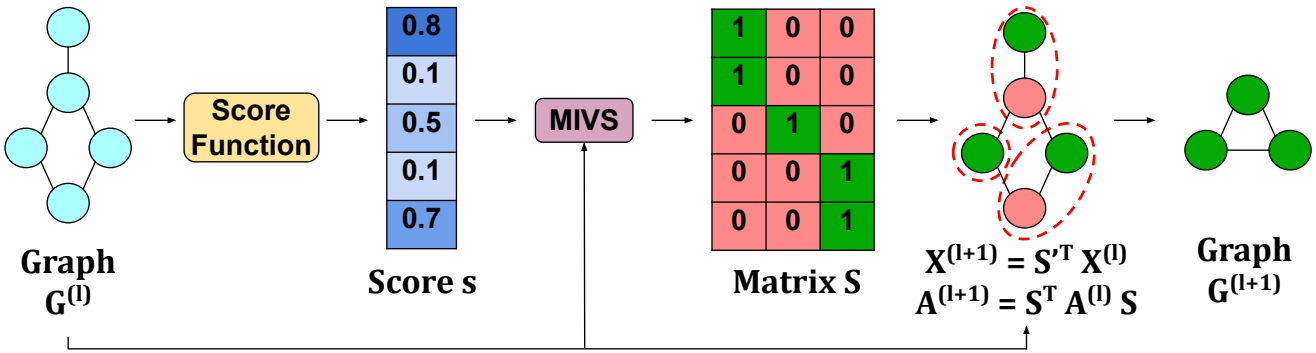
$$score = GNN_{l,pool}(A^l, h^l)$$

Combinaison GNNpool + relevance structurelle

Top-k/DiffPool: Pros/cons

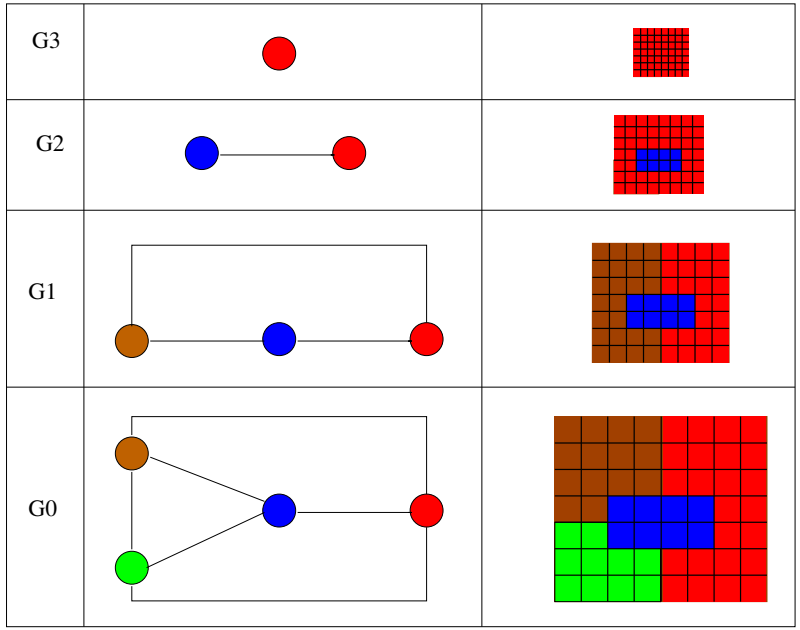
- DiffPool learns the attachment of non surviving vertices to surviving ones.
  - Dense matrices,
  - Graph of fixed size (the max one).
- Top-k learns the selection of surviving vertices but discard non survivning ones
  - May create disconnected graphs,
  - May ignore large parts of the graph,
  - removes a large part of the information related to the vertices.
- Why not defining a Top-k with a learned attachment of non surviving vertices ?

Introduction to Irregular Pyramids



Irregular Pyramids

- Stack of graphs $(G_0, G_1, \ldots, G_n)$ successively reduced.
- G_0 : encodes the initial grid or an initial segmentation.



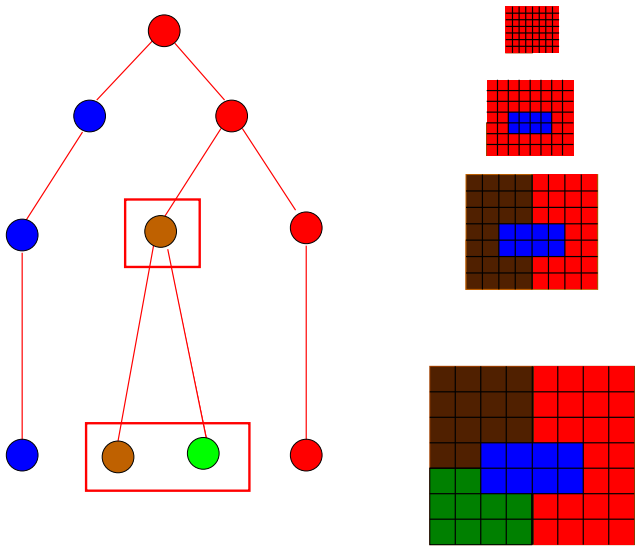
- Final results: sequence of reduced graphs $G_0, \ldots, G_n$

Reduction window

- $v \in V^i$ comes from the merge of a connected set of vertice in G_{i-1}.

$$RW_i(v) = \{v, v_1, \ldots, v_{n-1}\} \subset V^{i-1}$$

  - $v_j \in RW_i(v)$ is a son of v,
  - v is the father of all $v_j \in RW_j(v)$.



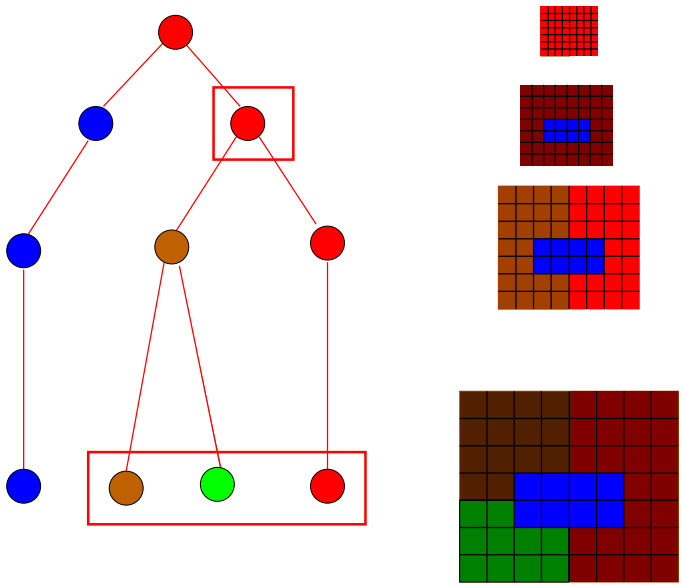
Receptive field

- Receptive field: transitive closure of the father/child relationship.

$$\forall i \geq 1, \forall v \in V^i, \quad RF_i(v) = \bigcup_{v' \in RW_i(v)} RF_{i-1}(v') \subset V^0$$

with $RF_0(v) = \{v\}, \forall v \in V^0$.

- $w \in RF_i(v)$ is a descendant of v,
- v is an ancestor of w.



Size of receptive fields

Let us suppose that at any layer l and for any vertex $w \in V^l$:

$$\left\{ \begin{array}{lcl} RW^l(w) & = & \{w\} \text{ or} \\ RW^l(w) & = & \{w, v_1, \ldots, v_n\} \text{ with } \forall i \in \{1, \ldots, n\}\ d_{G_{l-1}}(w, v_i) = 1 \end{array} \right. \tag{1}$$

where $d_{G_{l-1}}(.,.)$ is the distance within the graph G_{l-1} defined at layer $l-1$.

Proposition

Using a decimation scheme satisfying equation 1 we have for any vertex w surviving at level l in the hierarchy:

$$\forall (u, v) \in RF^l(w)^2 \quad d_{G_0}(u, v) \leq 2 * 3^l - 1$$

A only pooling scenario

L et us consider the simple following pooling function:

$$h^l = pool^l(h_{l-1}) = S^l h_{l-1} \tag{2}$$

Iterating the previous equation up to level 0 lead to:

$$h^q = \left(\prod_{l=1}^{q} S^l \right) h^0 =_{not} \Sigma^q h^0$$

with $\Sigma^q = \prod_{l=1}^{q} S^l \in \mathbb{R}^{n_q \times n}$, where n_q is the number of vertices of G_q and n the number of vertices of G_0.
Let $\sigma^q_{i,j}$ denotes the coefficient (i,j) of Σ^q. *If at any level a vertex either survives or is attached to an unique surviving neighbor* we have:

$$\sigma^q_{i,j} = \begin{cases} \prod_{k=1}^{q} s^k_{p^k(j),p^{k-1}(j)} & \text{If } j \in RF^q(i) \\ 0 & \text{otherwize} \end{cases}$$

where $p^k(j)$ denote the ancestor of j at level k.

We have thus:

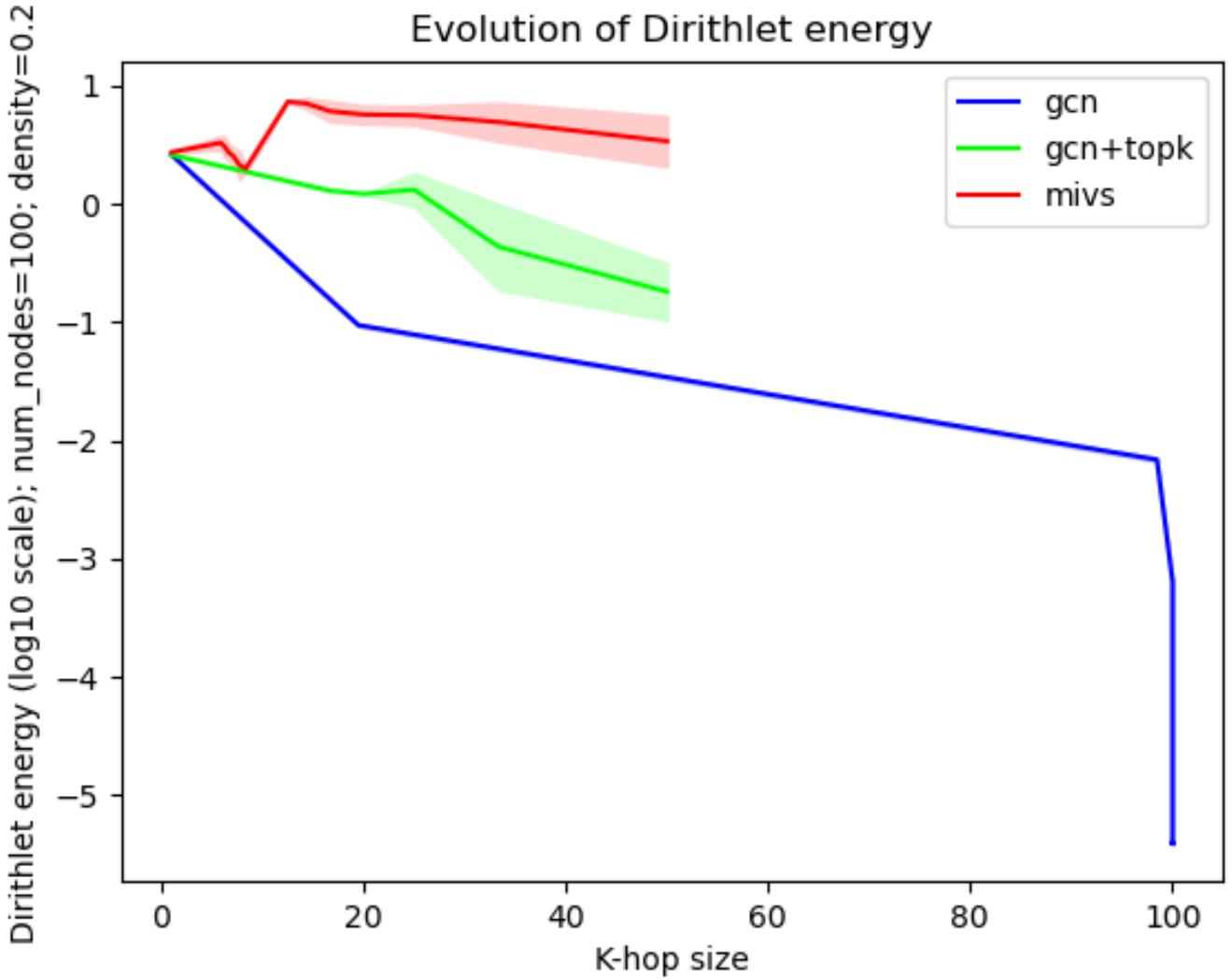
$$h_i^q = \sum_{j=1}^{n_q} \sigma_{i,j}^q h_j^0 = \sum_{j \in RF^q(i)} \sigma_{i,j}^q h_j^0$$

If all S^i are line stochastic we have:

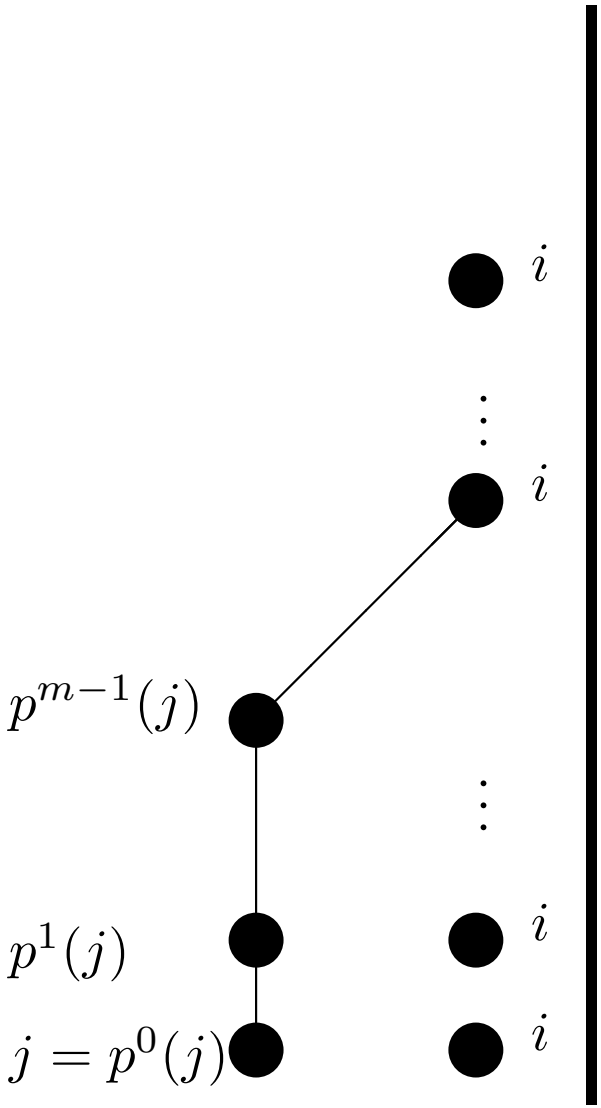
$$\sum_{j=1}^{N} \sigma_{i,j}^q = \sum_{j \in RF^q(i)} \sigma_{i,j}^q = 1$$

So h_i^q is a weighted sum of the features in $RF^q(i)$. Moreover, in this case, at any level q, $\{RF^q(i)\}_{i \in V^q}$ forms a partition of V^q.
This pooling scheme can not produce an over-smoothing unless all vertices in the base level have similar values.

Pooling and over-smoothing : Illustration



Pooling and over-squashing



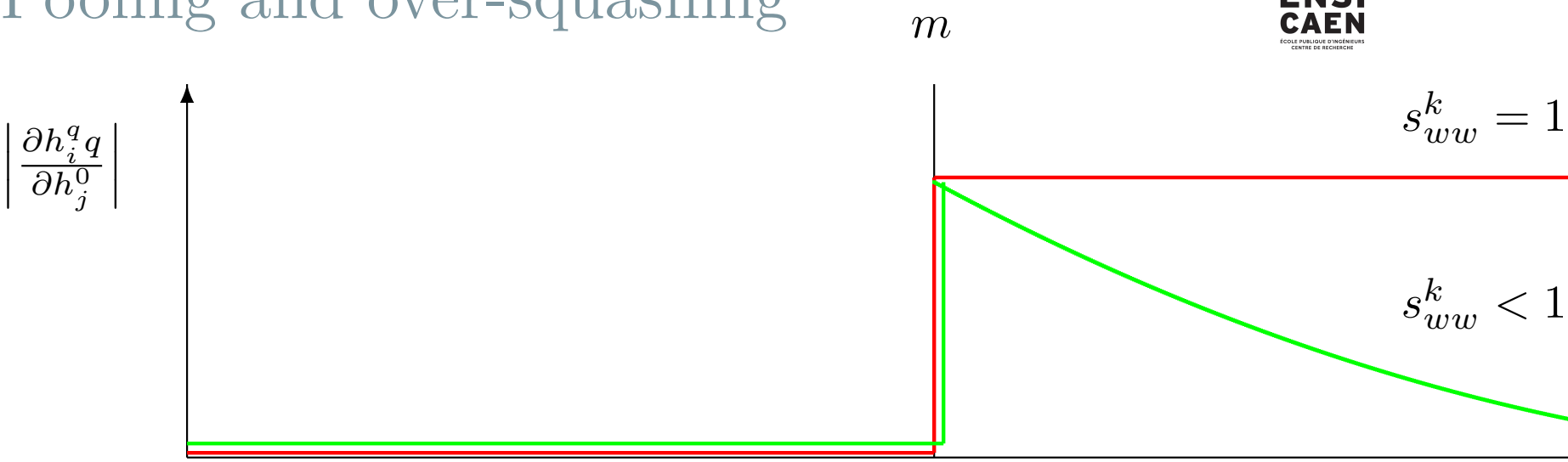
If Σ^q is independent of h^0 we have:

$$\frac{\partial h_i^q}{\partial h_j^0} = \sigma_{i,j}^q = \left(\prod_{k=m+1}^{q} s_{i,i}^k \right) \left(\prod_{k=1}^{m} s_{p^k(j),p^{k-1}(j)}^k \right)$$

if $j \in RF^q(i)$, 0 otherwise. If $s_{w,w}^k = 1, \forall k, w$:

$$\frac{\partial h_i^q}{\partial h_j^0} = \left(\prod_{k=1}^{m} s_{p^k(j),p^{k-1}(j)}^k \right) \text{ is constant}$$

Pooling and over-squashing



With Proposition 2 (slide 63) we have:

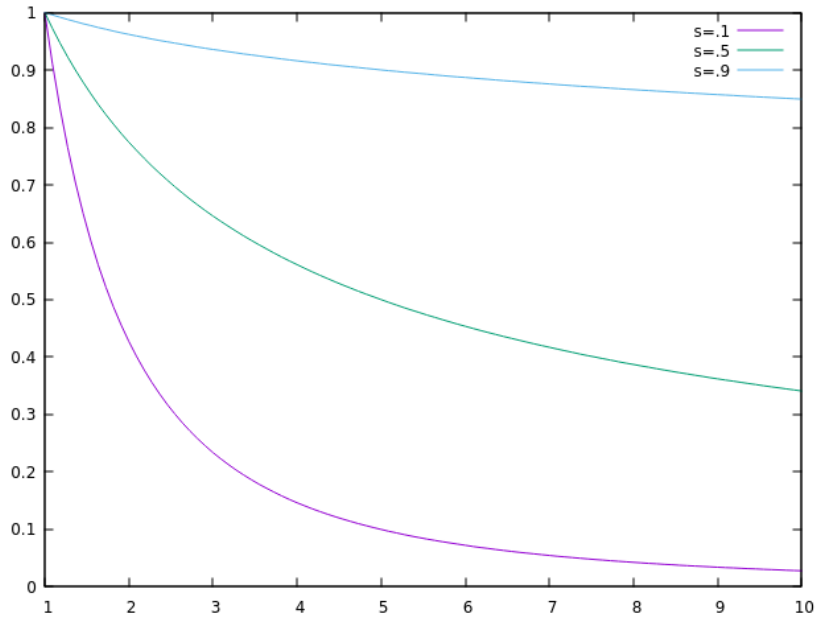
$$d_{G_0}(i,j) \leq 2 * 3^m - 1 \Rightarrow m \geq \log_3\left(\frac{d_{G_0}(i,j)+1}{2}\right)$$

Let us suppose that $m \approx \log_3(\frac{d_{G_0}(i,j)+1}{2})$ and let us look at the value of the pic. To this end, let $s_{k,l}^k = s < 1, \forall k \neq l$. We get:

$$\frac{\partial h_i^m}{\partial h_j^0} = \left(\prod_{k=1}^{m} s_{p^k(j),p^{k-1}(j)}^k\right) = s^m \approx s^{\log_3(\frac{d_{G_0}(i,j)+1}{2})} = \left(\frac{d_{G_0}(i,j)+1}{2}\right)^{\log_3(s)}$$

Since $log_3(s) < 0$ we get:

$$\frac{\partial h_i^m}{\partial h_j^0} \approx \left(\frac{2}{d_{G_0}(i,j)+1} \right)^{\alpha} \text{ with } \alpha = -\log_3(s) > 0$$



The higher s is, the higher $\frac{\partial h_i^m}{\partial h_j^0}$ will be. Note that it may not be an advantage, since it means that the network will have difficulties to differentiate near from far neighbors.

Construction schemes of the pyramid

- sequential methods:
  - sort the edges of the graphs
  - Union-find
- parallel method:
  - Define parallel merge operations
  - each step builds a new graph G_{i+1} from G_i.
  - $|G_{i+1}|$ is a fixed ratio of $|G_i|$.

$$|G_{i+1}| \approx q|G_i| \text{ with } q < 1\text{: reduction factor}$$

  -  the parallelism is a constraint for segmentation algorithms
    -  : "forces" a fixed amount of fusions at each step

$$|G_{i+1}| \approx q|G_i|$$

    -  bounds the number of graphs we have to build/store

$$\mathcal{P} = (G_0 \ldots, G_n) \text{ with } n = log_r(|G_0|)$$

Parallel construction schemes

- A set of independant processes merge vertices in parallel
- Problem : How to insure that: $\frac{V_i}{V_{i-1}} \lessapprox \frac{1}{2}$
  - computational time
  - storage memory.

Maximal Independent Set

Let us consider a set of abstract elements X and a symmetric neighborhood relationships $\mathcal{N}$ on X.

- $Y \subset X$ is an independent set of X iff it statisfies the **Internal stability** constraint:

$$\forall (y, y') \in Y^2,\ y \notin \mathcal{N}(y')$$

  Two neighbors cannot both survive

- Y is a maximal independant set iff adding any other element to it breaks independance. It satisfies in this case the **External stability** constraint:

$$\forall x \in X - Y,\ \exists y \in Y : x \in \mathcal{N}(y)$$

  Each element in $X - Y$ has a neighbor in Y.

Stochastic Pyramids

- Introduced by [Meer, 1989].
- V_{i+1} : maximal independent set of V_i.

  External stability:

  $$\forall v \in V_i - V_{i+1}\ \exists v' \in V_{i+1} : (v, v') \in E_i$$

  Each non surviving vertex is adjacent to at least a surviving one

  Internal stability:

  $$\forall (v, v') \in V_{i+1}^2\ (v, v') \notin E_i$$

  Two adjacent vertice cannot both survive

MIVS construction scheme

- Three variables :

  $p_i = true$ if v_i survives

  $q_i = true$ if v_i may become a surviving vertex (he is candidate).

  x_i value of the vertex (function or random variable)

MIVS construction scheme

- Three variables :

  $p_i = true$ if v_i survives

  $q_i = true$ if v_i may become a surviving vertex (he is candidate).

  x_i value of the vertex (function or random variable)

$$p_i^{(1)} = x_i = max_{j \in V(v_i)}\{x_j\}$$
$$q_i^{(1)} = \bigwedge_{j \in V(v_i)} \overline{p_j}^{(1)}$$

MIVS construction scheme

- Three variables :

  $p_i = true$ if v_i survives

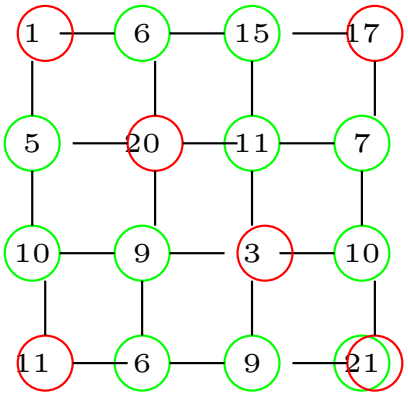
  $q_i = true$ if v_i may become a surviving vertex (he is candidate).

  x_i value of the vertex (function or random variable)

$$p_i^{(1)} = x_i = max_{j \in V(v_i)}\{x_j\}$$
$$q_i^{(1)} = \bigwedge_{j \in V(v_i)} \overline{p_j}^{(1)}$$

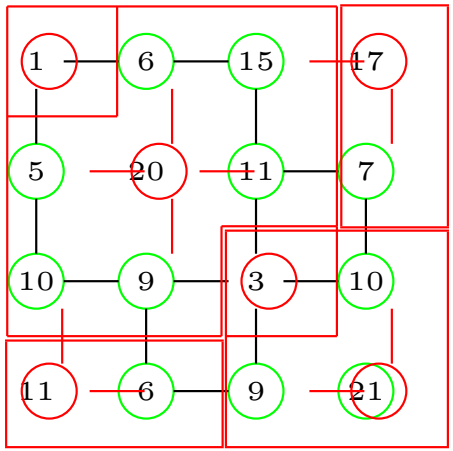
$$p_i^{(k+1)} = p_i^{(k)} \vee (q_i^{(k)} \wedge x_i = max_{j \in V(v_i)}\{q_j^{(k)} x_j\})$$
$$q_i^{(k+1)} = \bigwedge_{j \in V(v_i)} \overline{p_j}^{(k+1)}$$

MIVS : Father/child relationships



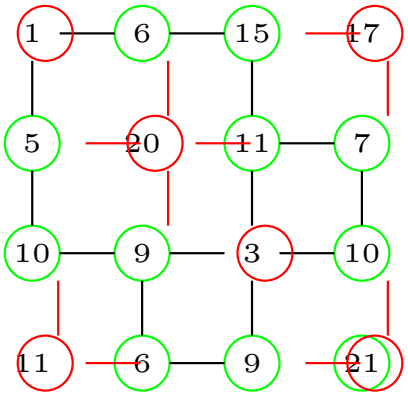
- link each non surviving vertex to one of its surviving neighbour $\Rightarrow$ definition of the edges
- merge non surviving vertice to surviving ones along the selected edges(merge in simple graphs).

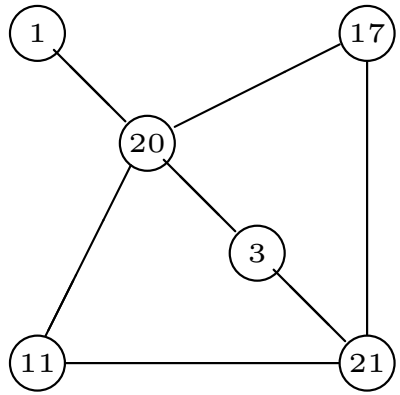
MIVS : Father/child relationships



- link each non surviving vertex to one of its surviving neighbour $\Rightarrow$ definition of the edges
- merge non surviving vertice to surviving ones along the selected edges(merge in simple graphs).

MIVS : Father/child relationships





- link each non surviving vertex to one of its surviving neighbour $\Rightarrow$ definition of the edges
- merge non surviving vertice to surviving ones along the selected edges(merge in simple graphs).

Data driven decimation

Data driven decimation

- perform one iteration of the kernel computation,

Data driven decimation

- perform one iteration of the kernel computation,
- attach each non surviving vertex to a surviving one

Data driven decimation

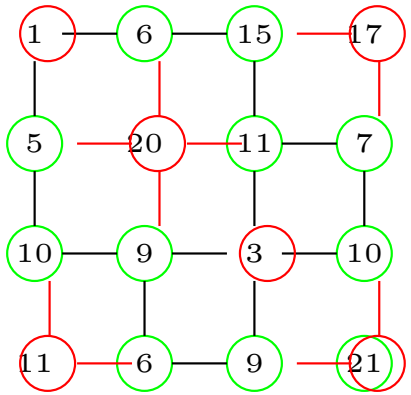
- perform one iteration of the kernel computation,
- attach each non surviving vertex to a surviving one
- merge vertices

Data driven decimation

- perform one iteration of the kernel computation,
- attach each non surviving vertex to a surviving one
- merge vertices
- continue on the reduced graph
- Method introduced by [Jolion, 2001].

Data driven decimation: conclusion

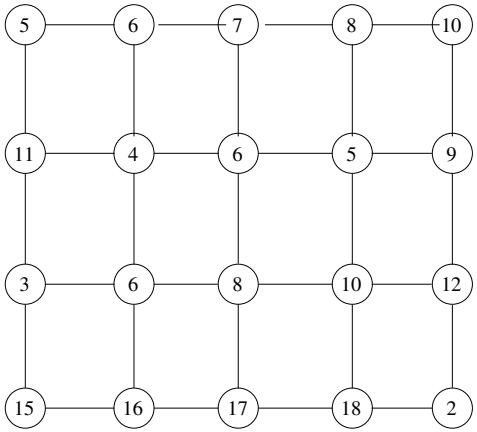
- only one step of the kernel computation is performed
  - "Corresponds" to a model of the behavior of our brain,
  - allows to avoid (in some cases) wrong merge operations.



Exercice : MIVS and D^3

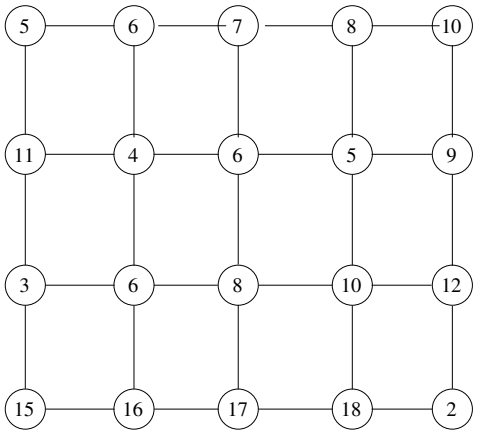
Define:

- p_i, q_i (Meer's algorithm, 2 steps),
- Reduction windows (legitimate father: max of r.v),
- Reduced graph.



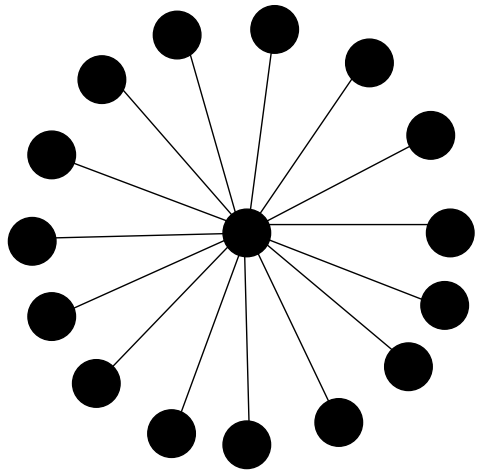
Exercice : MIVS and D^3

- Apply D^3 twice
- Legitimate father: max of r.v,



MIES : Motivations

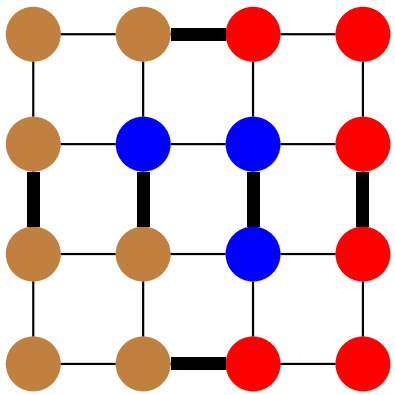
- Method introduced by Haximusa & Kropatsch [Kropatsch et al., 2005]
- within the kernel construction scheme the probability that a vertex survives decreases with its degree.
- The mean degree of vertices increases within the pyramid.



MIES : Motivations

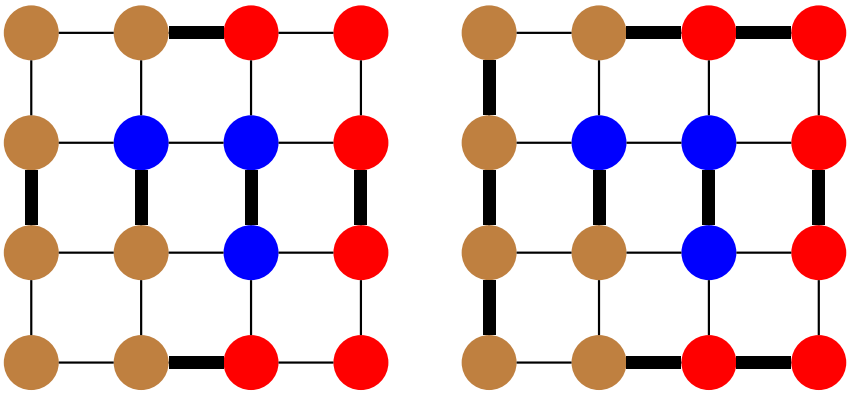
- Method introduced by Haximusa & Kropatsch [Kropatsch et al., 2005]
- within the kernel construction scheme the probability that a vertex survives decreases with its degree.
- The mean degree of vertices increases within the pyramid.
- $\Rightarrow$ The ratio $\frac{V_i}{V_{i-1}}$ computed by the kernel method decreases according to the level
  - Increases the computational time, even on parallel processors.
  - Useless graph storage.

- Define a maximal matching $C(\text{kernel of } G' = (E, E'))$
  - $(e, e') \in E'$ iff e and e' are incident to a same vertex.

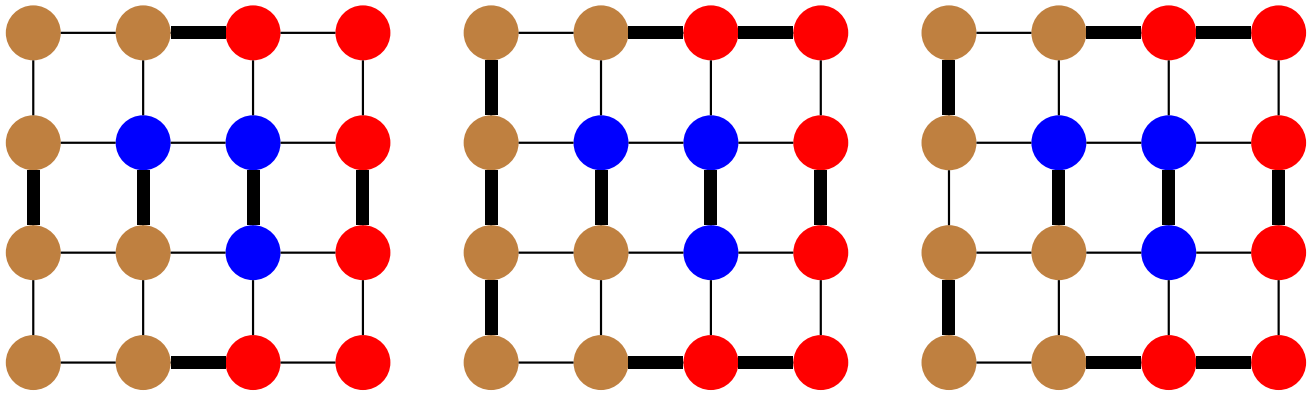


- A set $C \subset E$ is said to be a matching of $G = (V, E)$ if none of the edges of C are adjacent to a same vertex.
- A matching is said to be maximal if the addition of any edge breaks the matching property.
- A matching is said to be maximum if no larger matching may be found.

- Complete the matching C to C^+

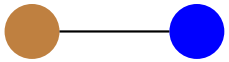
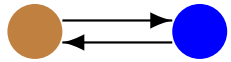


- Complete the matching C to C^+
- Remove edges from C^+ in order to obtain trees of depth 1.
- Merge vertice adjacent along selected edges.



Maximal Independent directed Edge Sets (MIDES)

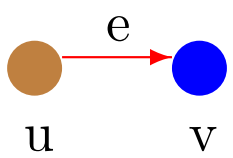
- How to design trees of depth 1 in one step ?
- How to take into account orientation of edges ?

- Solution: Orient edges.  $\rightarrow$ 
- Combine MIS method with an appropriate edge's neighborhood

Maximal Independent directed Edge Sets (MIDES)

- Let $G = (V, E)$.

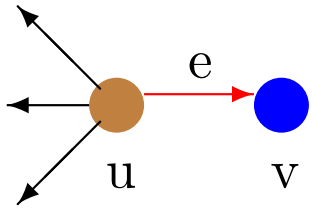
$$\mathcal{N}(e) = \mathcal{N}((u, v)) \quad =$$



Maximal Independent directed Edge Sets (MIDES)

- Let $G = (V, E)$.

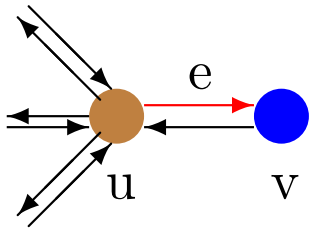
$$\mathcal{N}(e) = \mathcal{N}((u, v)) \quad = \quad \{(u, v') \in E\} \cup$$



Maximal Independent directed Edge Sets (MIDES)

- Let $G = (V, E)$.

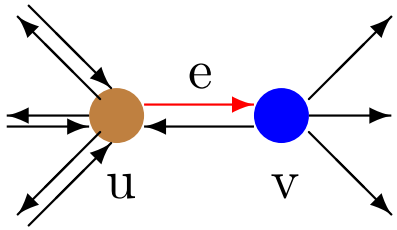
$$\begin{aligned} \mathcal{N}(e) = \mathcal{N}((u,v)) \quad &= \quad \{(u,v') \in E\} \cup \\ &\quad\ \{(u',u) \in E\} \cup \end{aligned}$$



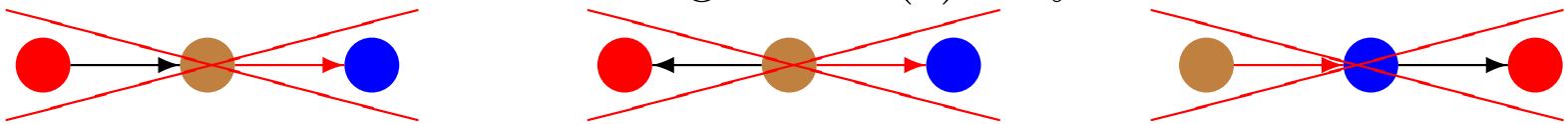
Maximal Independent directed Edge Sets (MIDES)

- Let $G = (V, E)$.

$$\begin{aligned} \mathcal{N}(e) = \mathcal{N}((u,v)) \quad &= \quad \{(u, v') \in E\} \cup \\ &\quad\ \{(u', u) \in E\} \cup \\ &\quad\ \{(v, u') \in E\} \end{aligned}$$

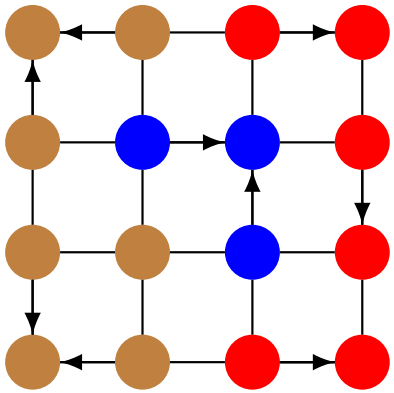


- If e is selected none of the edges of $N(e)$ may be selected.



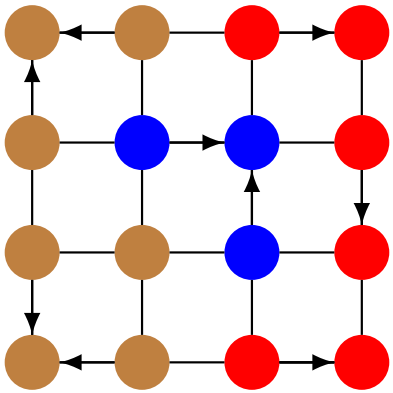
Maximal Independent directed Edge Sets (MIDES)

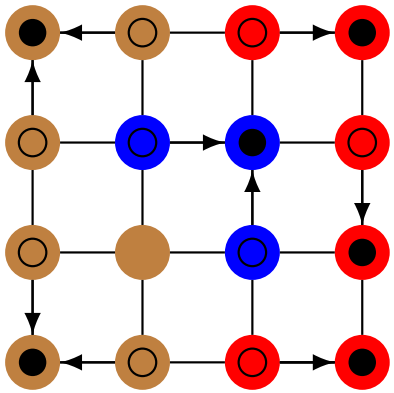
1. Apply a MIS on the edge graph $G = (E, E')$ with E' defined from $\mathcal{N}(E)$,



Maximal Independent directed Edge Sets (MIDES)

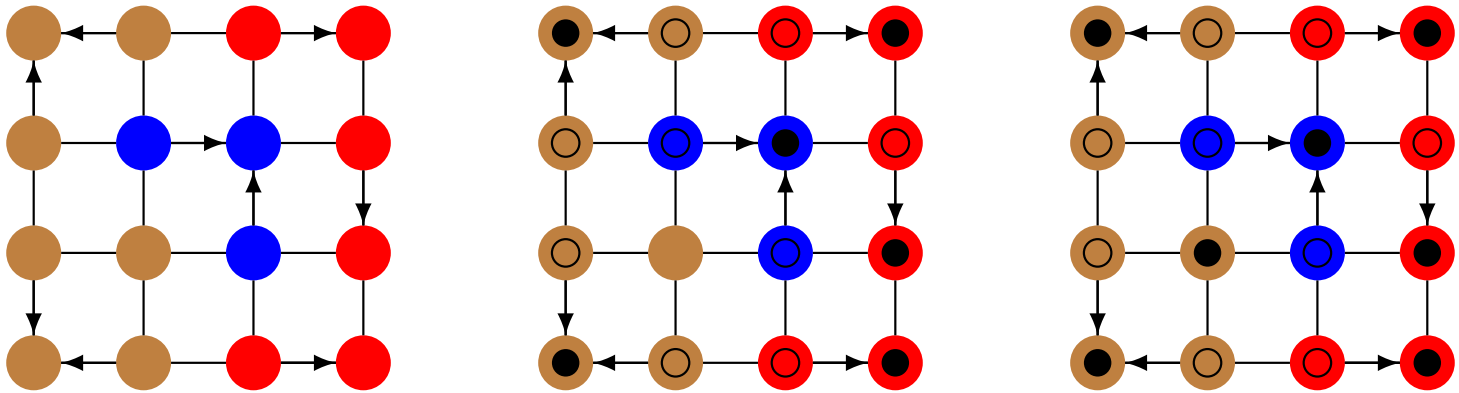
1. Apply a MIS on the edge graph $G = (E, E')$ with E' defined from $\mathcal{N}(E)$,
2. For each selected edge (u, v), mark v as survivor, u as non survivor,





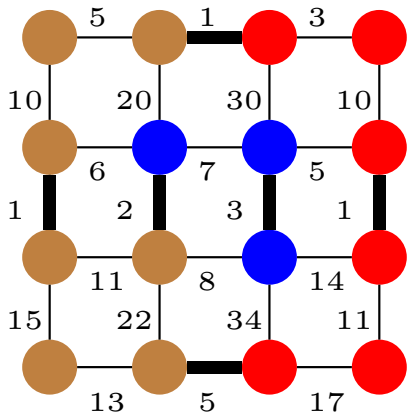
Maximal Independent directed Edge Sets (MIDES)

1. Apply a MIS on the edge graph $G = (E, E')$ with E' defined from $\mathcal{N}(E)$,
2. For each selected edge (u, v), mark v as survivor, u as non survivor,
3. Mark remaining vertices as survivors.



Data driven decimation and maximal matching

- Method introduced by Pruvot & Brun
- The maximal matching is defined as a MIS on the graph $G = (E, E')$.
  1. Value each edge as a merging cost,
  2. Perform only one iteration of the maximal matching algorithm
  3. One edge is selected if it is locally minimal (the two regions like each other more than any of their neighbour).



Conclusion on pooling

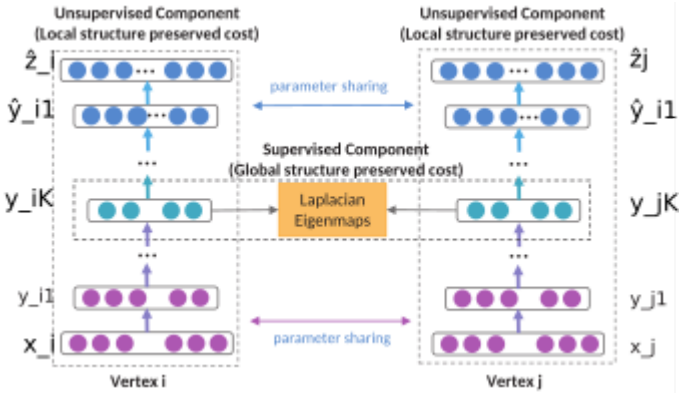
We can do:

- No pooling (pure gcn),
- pure pooling (MIVS,MIES,MIDES,...),
- gcn+pooling,
  - gcn+pooling with aggregation of non survivors to survivors $\Rightarrow$ two aggregation operations.
  - gcn+pooling restricted to a selection (Top k, MIVS without aggregation)

Network embedding

- Aims: Find a vectorial embedding of vertices which encodes their distance/proximity.
- Exemple of applications:
  - Recommendation systems (link prediction),
  - targeted advertizing (clustering (of users)),
- Problem: vertex embedding should capture both the vertex's features and the local structure of the graph.
- More formally: Given $G = (V, E)$ find a function $f : u \in V \rightarrow y \in \mathbb{R}^d$ with $d << |V|$ such that the proximity between y_u and y_v allows to determine the existence of $e_{u,v}$ (or richer property of the graph).

Exemple : SDNE



$$\mathcal{L} = \mathcal{L}_{2nd} + \alpha\mathcal{L}_{1st} + \nu\mathcal{L}_{reg}$$

- $\mathcal{L}_{2nd} = \sum_{i=1}^{n} ||(\hat{z}_i - x_i) \odot b_i||_2^2$
  - x_i is the i^{th} row of A, $\hat{z}_i$ is its reconstruction.
  - $b_i = (b_{i,j})_{j \in \{1,\dots,n\}}, b_{i,j} = 1$ if $A_{i,j} = 0$ and $\beta > 1$ otherwize.
- $\mathcal{L}_{1st} = \sum_{i,j} A_{i,j} ||y_i^K - y_j^K||_2^2 = (y^K)^T L y^K$
- $\mathcal{L}_{reg} = \frac{1}{2} \sum_{k=1}^{K} (||W^{(k)}||_F^2 + ||\hat{W}^{(k)}||_F^2)$
  - $W^{(k)}$ and $\hat{W}^{(k)}$ denote respectively the weighs of the the MLP encoder and decoder

The reconstruction is purely structural (no node feature)

Variational Graph Auto-Encoders

- Proposed by [Kipf and Welling, 2016]

  Encoder:

  $$Z = GCN(X, A) = \tilde{A}ReLU(\tilde{A}XW_0)W_1$$

  with $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$

  Decoder:

  $$\hat{A} = \sigma(ZZ^T)(A_{i,j} = \sigma(z_i^T z_j)).$$

  Loss: Minimize

  $$log(p(A|Z)) = \sum_{i,j|A_{i,j}=1} \log(\sigma(z_i^T z_j)) + \sum_{i,j|A_{i,j}=0} \log(1-\sigma(z_i^T$$

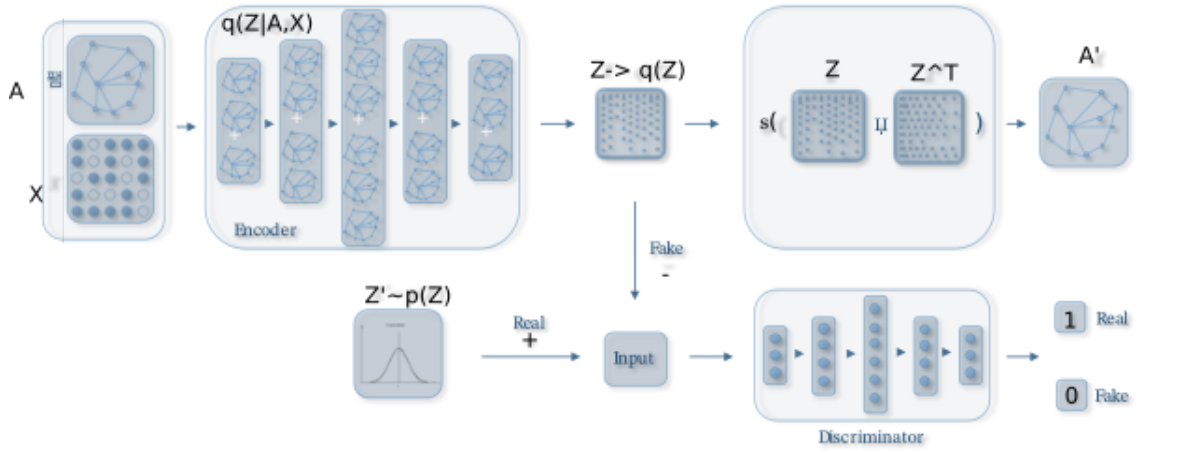
- A probabilistic version (with two GCN) estimating μ_i, σ_i such that z_i follows $\mathcal{N}(\mu_i, \sigma_i)$ is also proposed.

  $$\mathcal{L} = E_{q(Z|X,A)}\left[log(p(A|Z)\right] - KL\left[q(Z|X,A)||p(Z)\right],$$

  with $q(Z|X,A) = \Pi_{i=1}^{n} q(z_i|X,A)$;
  $q(z_i|X,A) = \mathcal{N}(z_i|\mu_i, diag(z_i))$. $P(Z) = \Pi_{i=1}^{n}\mathcal{N}(z_i|0,I)$.

ARGA/ARVGA

[Pan et al., 2018]



- Adversarial model $\mathcal{D}(Z)$ training:

$$-\frac{1}{2}\mathbb{E}_{z\sim p_z}\log(\mathcal{D}(Z)) - \frac{1}{2}\mathbb{E}_X log(1-\mathcal{D}(G(X,A)))$$

- Complete model:

$$\min_G \max_{\mathcal{D}} \mathbb{E}_{z\sim p_z}[\log(\mathcal{D}(Z)] + \mathbb{E}_{x\sim p(x)}[\log(1-\mathcal{D}(G(X,A)))]$$

- For each entry $G = X, A$ we compute Z using the encoder and train the discreminator with Z and an equal number of generated entries.

GraphSage

- GraphSAGE [Hamilton et al., 2017] embedding generation algorithm

  INPUT: G(V,E); input features $\{x_v, \forall v \in V\}$; depth K; weight matrices W^k; non-linearity σ; aggregator functions $AGGREGATEk, \forall k \in \{1, ..., K\}$;

  OUTPUT: Vector representations z_v for all $v \in V$.

  $h_v^0 \leftarrow x_v, \forall v \in V$;
  for k= 1...K **do**
    for $v \in V$ **do**
      $h_{\mathcal{N}(v)}^k \leftarrow AGGREGATE_k(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
      $h_v^k \leftarrow \sigma(W^k.CONCAT(h_v^{k-1}, h_{\mathcal{N}(v)}^k))$;
    end for
    $h_v^k \leftarrow \frac{h_v^k}{||h_v^k||^2}, \forall v \in V$;
  end for
  $z_v \leftarrow h_v^K, \forall v \in V$

- Loss:

$$J_G(z_u) = -\log(\sigma(z_u^T z_v)) - QE_{v_n \sim P_n(v)} \log(\sigma(-z_u^T z_{v_n})),$$

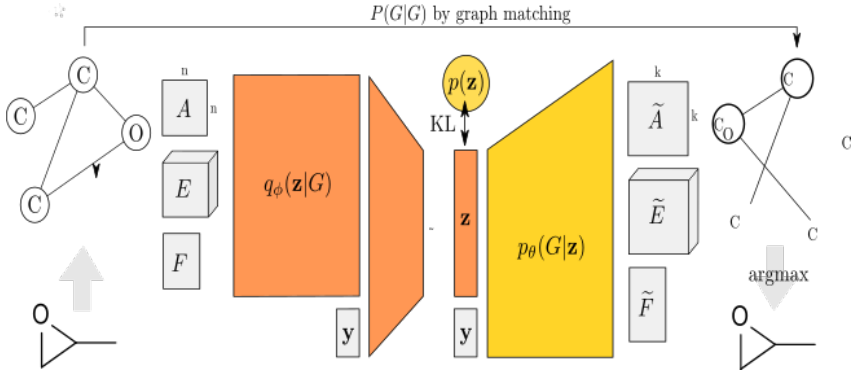
v close from u (on a fixed length random walk), v_n far from $u($

Graph embedding

- Determines a hidden representation of a graph z so as to be able to reconstruct both the features and the vertices from z.
- Exemple of applications:
  - Generate promizing new molecules (drug design),
  - perform graph clustering through graph embedding

GraphVAE

[Simonovsky and Komodakis, 2018]



Encoder: A feed forward network with edge-conditioned graph convolutions (ECC) [Simonovsky and Komodakis, 2017]

Decoder: a MLP with three outputs in its last layer producing $\tilde{A} \in \mathbb{R}^{k \times k}, \tilde{E} \in \mathbb{R}^{k \times k \times d_e}, \tilde{F} \in \mathbb{R}^{k \times k \times d_n}$. k: fixed size (around 10)

Loss:

$$\mathcal{L}(\phi, \theta; G) = \mathbb{E}_{q_\phi(z|G)}[-\log p_\theta(G|z)] + KL[q_\phi(z|G)||p(z)]$$

with $p(z)$ a Gaussian distribution and :

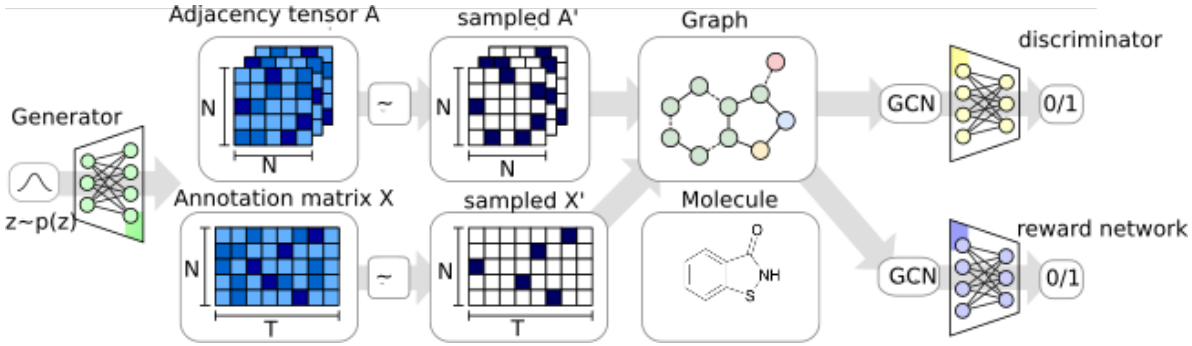
$$-\log p_\theta(G|z) = -\lambda_A \log p(A'|z) - \lambda_F \log p(F|z) - \lambda_E \log p(E|z)$$

where $A', p(A'|z), p(F|z), p(E|z)$ are deduced from a

GraphVAE: Extensions

GraphVAE do not provide any garantee on the (application based) validity of the obtained graphs. For e.g. number of bounds of a given atom.

- [Ma et al., 2018] formulate penalty terms that regularize the output distribution of the decoder (add penality terms to the loss corresponding to the different constraints).
- [Cao and Kipf, 2018] use an adversarial network to integrate the constraints.
  - The use of adversarial network avoid the use of a matching.
  - A reward component enforce the generation of graphs with specific properties.



Definition & Examples

A spatial-temporal graph is a graph whose structure and/or node/edge features vary over time.

- Brain Graphs:
  - Each node corresponds to a region of the brain. Each node is characterized by a feature vector encoding blood pressure (region activity).
  - Over time and according to neural activities different zones may act synchronously: They are connected by an edge.

  Both node features and graph structure vary over time.
- Trafic network:
  - Each node correspond to a speed sensor,
  - Each edge to the distance between sensors.

  The structure is invariant but the evolution of the speed at one sensor depends on the ones of the nearby sensors.
- Other applications: human activity recognition ; segmentation from videos, context-rich human-object interactions, modeling human motion, etc.

- Classical RNN:

$$H^t = \sigma(WX^t + UH^{(t-1)} + b)$$

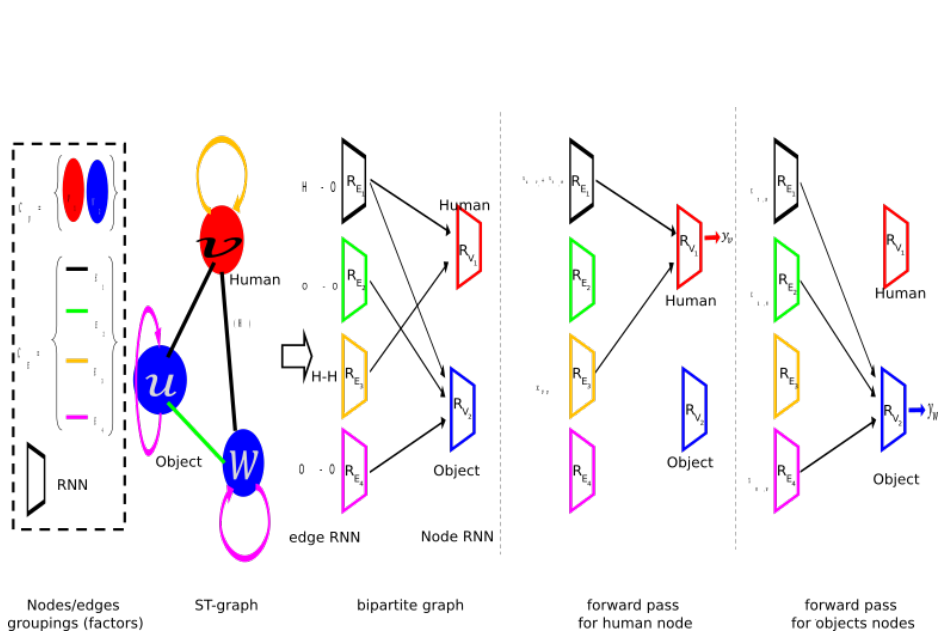
- RNN with spatial convolution:

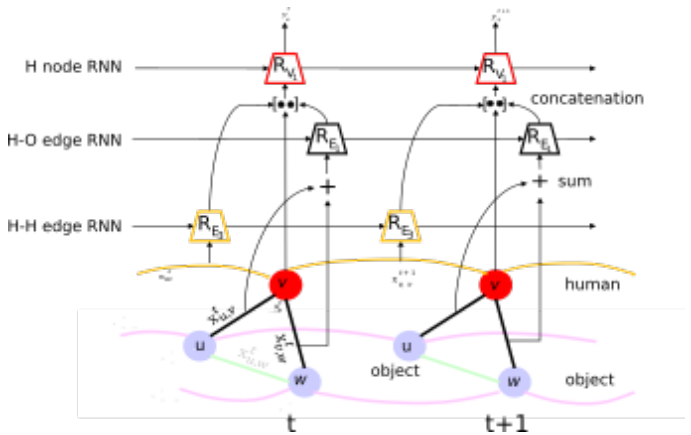
$$H^t = \sigma\left(Gconv(X^t, A^t, W) + Gconv(H^{(t-1)}, A^t, U) + b\right)$$

Structural RNN [Jain et al., 2015]

Let us consider a spatial-temporal graph $G = (V, E_S, E_T)$ where E_T connects a same node at different time steps.

- Group the nodes and edges into groups (factors) $V_1, \ldots, V_p$, $E_1, \ldots, E_m$. A group cannot mix spatial and temporal edges.
- Associate a RNN to each factor, connect a node factor V_i to an edge E_j iff $\exists u \in V_i, v \in V s.t (u, v) \in E_j$. The resulting graph is bipartite.





Interesting mainly if we can distinguish meaningfull groups.

ST-GCN [Yu et al., 2017]

Combine 1D and Graph convolution operations in order to predict the speed in a traffic network.

- Iterate two ST-convolution operations combined with a last temporal convolution and a FC layer.
- Each ST-convolution is composed of two temporal convolutions taking a graph convolution in sandwich:

  temporal convolution: combine a convolution of size K_t with a Gated Linear Unit (GLU) operation:

  $$\Gamma *_\tau Y = P \odot \sigma(Q)$$

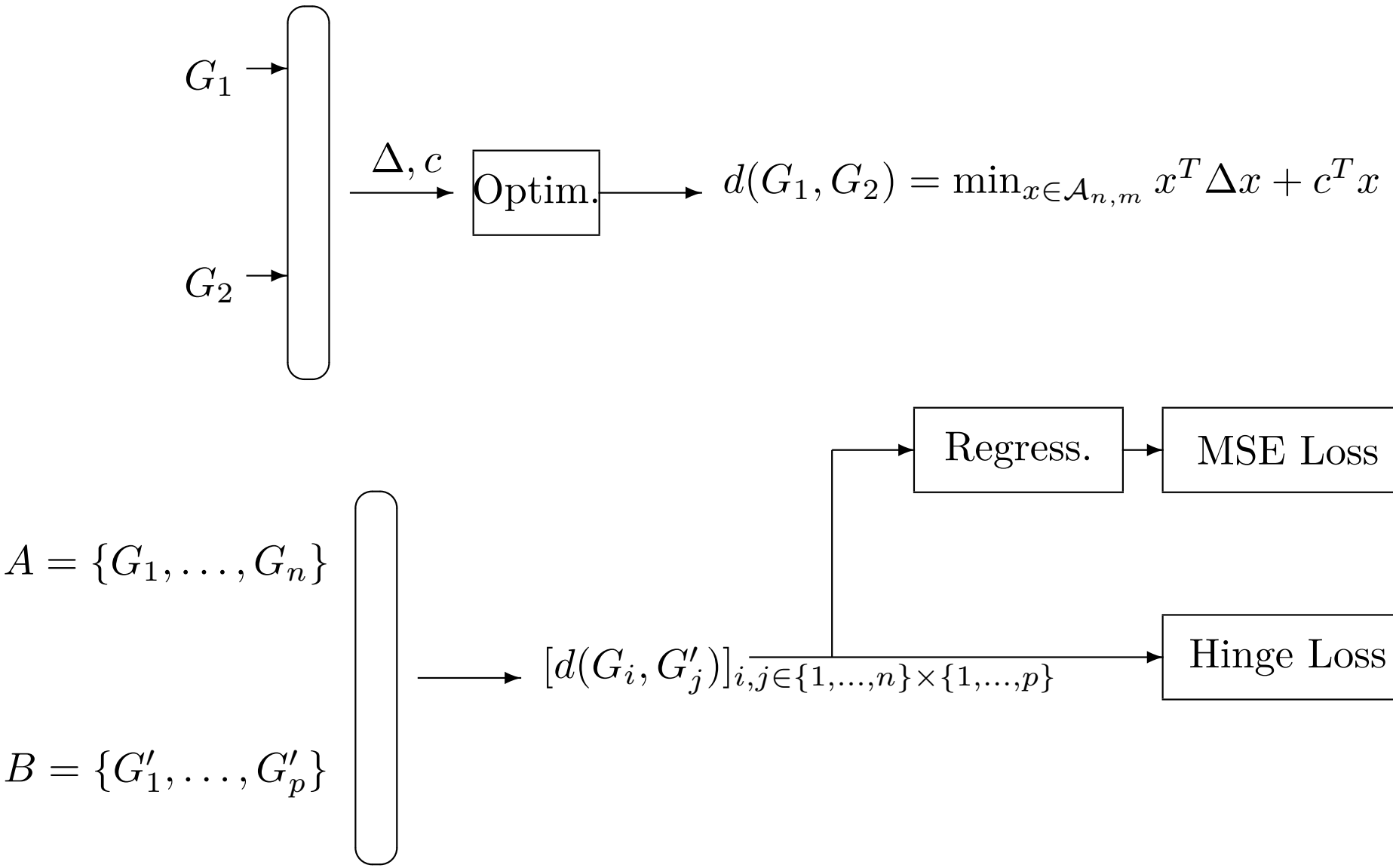
  P and Q being obtained by the convolution kernel.

  Graph convolution: See slide 20.
- Overall operation:

$$v^{l+1} = \Gamma_1^l *_\tau ReLU(\theta^l *_\mathcal{G} (\Gamma_0^l *_\tau v^l)) \in \mathbb{R}^{(M-2(K_t-1))\times n\times C^{l+1}}$$

with $v^l \in \mathbb{R}^{M\times n\times C^l}$, M: length of the time serie.

Metric Learning



Metric Learning

- Learned params correspond to the parameters of the metric (e.g. costs of substitution/insertion/removal for the GED).
- The metric may be applied to regression or classification problem.

  Regression: Kernel ridge or kNN (cheaper) regression. MSE Loss.

  Classification: Hinge Loss. A single set (e.e. A) may be used.

$$HL = \sum_{i,j \in \{1,\dots,n\} \times \{1,\dots,p\}} y_{i,j} d(G_i, G'_j)$$

with $y_{i,j} = 1$ if G_i and G'_j belong to a same class, -1 otherwize.

Let us note that we have two nested optimizations: The computation of the metric and the backward. Both may interfere. For example, too many iteration steps for computing the metric may induce vanishing gradients.

Bibliography

Anis, A., Gadde, A., and Ortega, A. (2014). Towards a sampling theorem for signals on arbitrary graphs. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014, Florence, Italy, May 4-9, 2014*, pages 3864–3868. IEEE.

Atwood, J. and Towsley, D. (2016). Diffusion-convolutional neural networks. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1993–2001.

Bacciu, D., Errica, F., and Micheli, A. (2018). Contextual graph markov model: A deep and generative approach to graph processing. *CoRR*, abs/1805.10636.

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D. J., and Kavukcuoglu, K. (2016). Interaction networks for learning about objects, relations and physics. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4502–4510.

Bianchi, F. M., Grattarola, D., Livi, L., and Alippi, C. (2022). Hierarchical representation learning in graph neural networks with node decimation pooling. *IEEE Trans. Neural Networks Learn. Syst.*, 33(5):2195–2207.

Brun, L. (2019). Graph classification (invited talk). In Vento, M. and Percannella, G., editors, *Proceedings of CAIP 2019*, LNCS, Salerno (IT). IAPR distinguished speaker.

Cao, N. D. and Kipf, T. (2018). Molgan: An implicit generative model for small molecular graphs.

Dasoulas, G., Lutzeyer, J. F., and Vazirgiannis, M. (2021). Learning parametrised graph shift operators. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Dhillon, I. S., Guan, Y., and Kulis, B. (2007). Weighted graph cuts without eigenvectors A multilevel approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(11):1944–1957.

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2224–2232.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1263–1272.

Giovanni, F. D., Giusti, L., Barbero, F., Luise, G., Lio, P., and Bronstein, M. M. (2023). On over-squashing in message passing neural networks: The impact of width, depth, and topology. In Krause, A., Brunskill, E., Cho, K., Engelhardt,

B., Sabato, S., and Scarlett, J., editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 7865–7885. PMLR.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. *CoRR*, abs/1706.02216.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Jain, A., Zamir, A. R., Savarese, S., and Saxena, A. (2015). Structural-rnn: Deep learning on spatio-temporal graphs. *CoRR*, abs/1511.05298.

Jolion, J.-M. (2001). Data driven decimation of graphs. In Jolion, J.-M., Kropatsch, W., and Vento, M., editors, *Proceedings of* 3^{rd} *IAPR-TC15 Workshop on Graph based Representation in Pattern Recognition*, pages 105–114, Ischia-Italy.

Kearnes, S. M., McCloskey, K., Berndl, M., Pande, V. S., and Riley, P. (2016). Molecular graph convolutions: moving beyond fingerprints. *J. Comput. Aided Mol. Des.*, 30(8):595–608.

Kipf, T. N. and Welling, M. (2016). Variational graph auto-encoders. *CoRR*, abs/1611.07308.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning*

Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.

Kropatsch, W. G., Haxhimusa, Y., Pizlo, Z., and Langs, G. (2005). Vision pyramids that do not grow too high. *Pattern Recognit. Lett.*, 26(3):319–337.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. S. (2016). Gated graph sequence neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings.*

Ma, T., Chen, J., and Xiao, C. (2018). Constrained generation of semantically valid graphs via regularizing variational autoencoders. *CoRR*, abs/1809.02630.

Massa, V. D., Monfardini, G., Sarti, L., Scarselli, F., Maggini, M., and Gori, M. (2006). A comparison between recursive neural networks and graph neural networks. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2006, part of the IEEE World Congress on Computational Intelligence, WCCI 2006, Vancouver, BC, Canada, 16-21 July 2006*, pages 778–785.

Meer, P. (1989). Stochastic image pyramids. *Computer Vision Graphics Image Processing*, 45:269–294.

Micheli, A. (2009). Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511.

Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., and Zhang, C. (2018). Adversarially regularized graph autoencoder for graph embedding. In Lang, J., editor,

Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, pages 2609–2615. ijcai.org.

Peng, N., Poon, H., Quirk, C., Toutanova, K., and Yih, W. (2017). Cross-sentence n-ary relation extraction with graph lstms. *TACL*, 5:101–115.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80.

Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K. R., and Tkatchenko, A. (2017). Quantum-chemical insights from deep tensor neural networks. *Nature Communications*, 8(1).

Simonovsky, M. and Komodakis, N. (2017). Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 29–38.

Simonovsky, M. and Komodakis, N. (2017). Dynamic edge-conditioned filters in convolutional neural networks on graphs. *CoRR*, abs/1704.02901.

Simonovsky, M. and Komodakis, N. (2018). Graphvae: Towards generation of small graphs using variational autoencoders. *CoRR*, abs/1802.03480.

Tran, D. V., Navarin, N., and Sperduti, A. (2018). On filter size in graph convolutional networks. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1534–1541.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Verma, N., Boyer, E., and Verbeek, J. (2017). Dynamic filters in graph convolutional networks. *CoRR*, abs/1706.05206.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2019). A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596.

Yan, S., Xiong, Y., and Lin, D. (2018). Spatial temporal graph convolutional networks for skeleton-based action recognition. *CoRR*, abs/1801.07455.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 4805–4815.

Yu, B., Yin, H., and Zhu, Z. (2017). Spatio-temporal graph convolutional neural network: A deep learning framework for traffic forecasting. *CoRR*, abs/1709.04875.

Zayats, V. and Ostendorf, M. (2018). Conversation modeling on reddit using a graph-structured LSTM. *TACL*, 6:121–132.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In McIlraith, S. A. and Weinberger, K. Q., editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4438–4445. AAAI Press.