Basics of Machine learning
Classic methods
From molecules to prediction
Neural Networks
Graph Neural Networks
Bibliography

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Basics of IA For Chemistry

March 27, 2026

GREYC

cnrs
www.cnrs.fr

Informations

- Intervenants:
  - Luc Brun
- Supports :
  - Support de cours:
    `https://lucbrun.ensicaen.fr/ENSEIGNEMENT/enseignement.html`
  - Support de TP: `https://foad.ensicaen.fr/`

1. Basics of Machine learning
   - Definition
2. Classic methods
   - How to measure ML performances
   - knn
   - SVM
   - Kernel Methods
3. From molecules to prediction
   - Fingerprints
   - Graph kernels
   - Kernels based on infinite Bags
   - Kernels based on finite Bags
   - Application to chemoinformatics
4. Neural Networks
   - Principles
5. Graph Neural Networks
   - Graph Aggregation
   - Recurrent Graph Neural Networks
   - Convolutional Graph neural networks
     - Spectral approaches
     - Spatial Approaches

- Graph Pooling
- Irregular Pyramids
- Stochastic Pyramids
- XAI

Basics of Machine learning
Classic methods
From molecules to prediction
Neural Networks
Graph Neural Networks
Bibliography

6 Bibliography

Sommaire de la section

1 Basics of Machine learning
- Definition

2 Classic methods
- How to measure ML performances
- knn
- SVM
- Kernel Methods

3 From molecules to prediction
- Fingerprints
- Graph kernels
- Kernels based on infinite Bags
- Kernels based on finite Bags
- Application to chemoinformatics

4 Neural Networks
- Principles

5 Graph Neural Networks
- Graph Aggregation
- Recurrent Graph Neural Networks
- Convolutional Graph neural networks
  - Spectral approaches

What do we mean by learning ?

Learning is an active process

- The learning process involves the deduction of some invariant and some kind of generalization, e.g.
  - learn to do bicycle/tennis,
  - learn to drive a car,...
- It involves an ability to deal with unknown situations/configurations.

Learning ≠ "by heart"

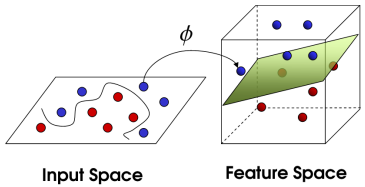
- A database stores a huge amount of data but learns nothing.

Different learning epochs

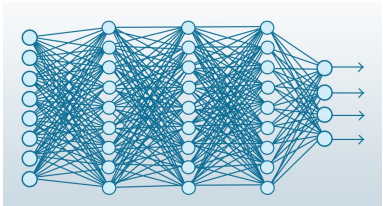
Level 0 (1965-1980): Hand made classification (Expert Systems)

If $nbcarbons > 10$ and $diameter < 15$ then $ebulitionPoint > 10°$

Level 1 (1990-2000):] Design of feature vectors/(di)similarity measures.
Automatic Classification



Level 2 (2000-): Automatic design of pertinent features / metric from huge amount of examples.



We will study Levels 1 and 2 systems within the lecture.

Machine learning and chemistry: Why ?

Motivations

1. Avoid/reduce expensive synthesis/test of candidate molecules.
2. Get a deeper understanding on the key molecular properties associated to some effect.
3. Analyses of huge amount of molecules,
4. Avoid huge computations (molecule conformations),
5. Generate molecular candidate for a given effect.

Objectives

1. Explore / structure large datasets
2. Predict molecular properties, conformation,...
3. Explain a prediction.

Chemoinformatics

Prediction of individual molecule properties. The prediction of the ebulition point belongs to chemoinformatics. The prediction of the freezing point is not.

Terminology

QSAR (Quantitative structure-activity relationship) and QSPR (Quantitative structure-property relationship) are the two main sub-fields of chemo-informatics.

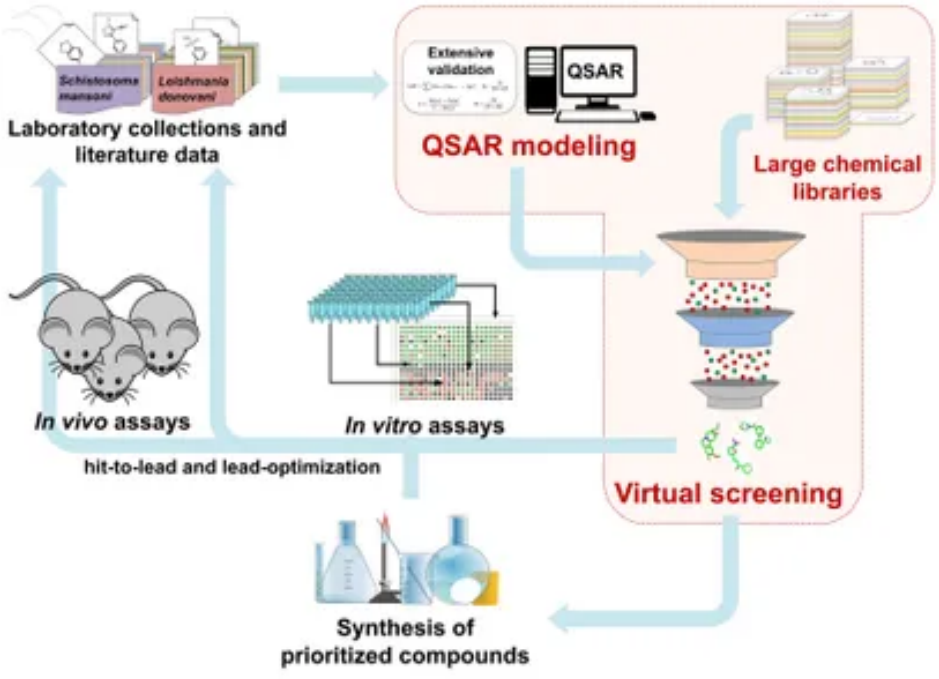
Criterium	QSAR	QSPR
Objective: predict	molecular activity	physico-chemical property
Type of property	functional or biological	physical or chemical
Example	Inhibition of an enzyme, toxicity	ebulition point, LogP, solubility,...)

Example : Let us take a set of molecules for which we want to estimate the toxicity:

- QSAR may predict the toxicity of each molecule,
- QSPR may predict the solubility and the logP.

Vitual Screening

- One molecule is selected from every 2.3 million compounds that target the research project in this process,
- The average cost for effective drug research and development ranges from $900 million to $2 billion,
- The time from the discovery of a drug produced for the treatment of a disease to its release on the market takes an average of 12-15 years,
- The success rate of launching a drug from a Phase I clinical trial is daunting, less than 10%.



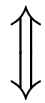
Deeper understanding of key properties

Level 1 ML methods are based on the following aphorism:

Similar molecules have similar properties.

Which hides many properties and many ways of being similar....

- Design a similarity/distance measure between molecules in order to predict a molecular property



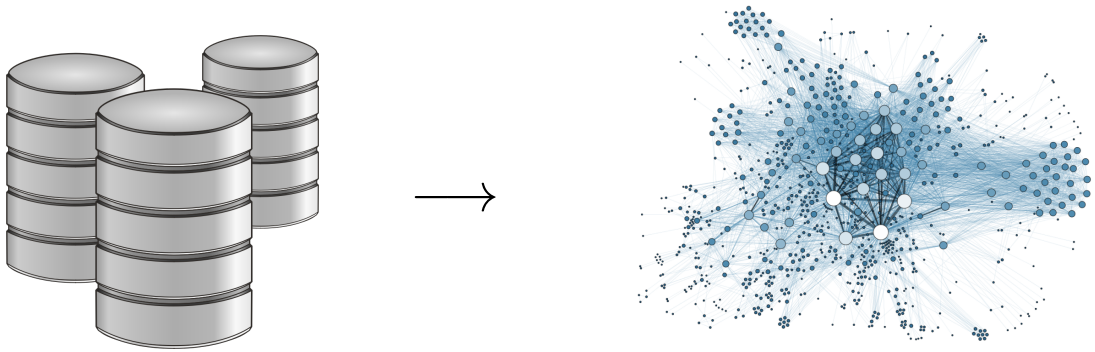
- search for the key properties of molecules related to this property.

This search being guided by the dataset and the ground true values.

Analyses huge amounts of molecules

Chemoinformatics allow to pass from raw chemical data to a structured dataset. Queries may be as follow:

- Find all molecules which have one/a set of compounds,
- Get all the molecules similar to a given one,
- group molecules by similarities, check cluster properties. . . .



- the analysis of heterogeneous graphs with molecules, proteins and effect nodes and 'similar', 'docking', 'induces' relationships is an active research field.

Avoid huge computations

- The $3D$ shape of a molecule is important for many properties.
- Computing all conformations of minimal energy or the evolution of a molecule given an initial conformation requires huge calculus on supercomputers.
- $\Rightarrow$ Restricts these computations to small molecules/small datasets.

Machine learning approach :

- Given a conformation, learn the molecular and atomic energies , induce the force fields applied to each atom, modify the conformation.
- Much more efficient.

Ask yourself: does all champions in freestyle skiing or in parallel bars have a master in solid mechanics ?

Sommaire de la section

1 Basics of Machine learning
- Definition

2 Classic methods
- How to measure ML performances
- knn
- SVM
- Kernel Methods

3 From molecules to prediction
- Fingerprints
- Graph kernels
- Kernels based on infinite Bags
- Kernels based on finite Bags
- Application to chemoinformatics

4 Neural Networks
- Principles

5 Graph Neural Networks
- Graph Aggregation
- Recurrent Graph Neural Networks
- Convolutional Graph neural networks
  - Spectral approaches

In classification

		predicted	
		0	1
ground true	0	TN	FP
	1	FN	TP

Let us suppose that you receive 10 emails, 7 are classified as spam and 3 as 'normal'. We suppose 2 errors among the 7 emails detected as spam, and 1 error among the emails classified as normal: TP=5, FN=1, TN=2, FP=2.

- Accuracy:

$$Accuracy = \frac{\text{correct classification}}{\text{total classification}} = \frac{TP + TN}{TP + TN + FP + FN}$$

  e.g. Accuracy=(5+2)/10=70%.

- Recall:

$$Recall(TPR) = \frac{\text{correctly classified positive}}{\text{total of positive}} = \frac{TP}{TP + FN}$$

  e.g. Recall=5/(5+1)≈ 83%

In classification

		predicted	
		0	1
ground true	0	TN	FP
	1	FN	TP

TP=5, FN=1, TN=2, FP=2.

- False positive rate (FPR)

$$FPR = \frac{\text{incorrectly classified negative}}{\text{total of negative}} = \frac{FP}{FP + TN}$$

  e.g. FPR=1/(1+2)=33%.

- Precision:

$$Precision = \frac{\text{correct positive classification}}{\text{total of positives classification}} = \frac{TP}{TP + FP}$$

  e.g. proportion of emails in spam folder which effectively correspond to a spam. Precision=5/(5+2)≈ 71%.

In Classification

		predicted	
		0	1
ground true	0	TN	FP
	1	FN	TP

Accuracy

		predicted	
		0	1
ground true	0	TN	FP
	1	FN	TP

Recall

		predicted	
		0	1
ground true	0	TN	FP
	1	FN	TP

FPR

		predicted	
		0	1
ground true	0	TN	FP
	1	FN	TP

Precision

$$\frac{\blacksquare}{\blacksquare + \blacksquare}$$

In classification

		predicted	
		0	1
ground true	0	TN	FP
	1	FN	TP

TP=5, FN=1, TN=2, FP=2.

- F1 score: harmonic mean between precision and recall:

$$F1 = 2 * \frac{\text{precision*recall}}{\text{precision+recall}} = \frac{2TP}{2TP + FP + FN}$$

  e.g. F1=2*5/(2*5+2+1)≈77%.

- mean F1 (mF1):

$$mF1 = \text{Arithmetic mean of F1 for all classes.}$$

  In our example:

$$mF_1 = \frac{1}{2}(F1_0 + F1_1) = \frac{1}{2}(2*2/(2*2+2+1) + 2*5/(2*5+2+1)) \approx 67\%.$$

  Mainly used for multi class problems with asymmetric classes.

In Regression

- Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n} \|y_i - \hat{y}_i\|^2$$

  y_i: prediction, $\hat{y}_i$: true value.

- Mean absolute error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

  - Less smooth than MSE :-(.
  - Avoid to focus the optimizer on large errors :-).

K-Nearest Neighbors algorithms

Introduced by [Fix and Hodges, 1951]
Let us consider a training set D and an input query x, $k \in \mathbb{N}^*$, and a distance between molecules.

1. Find the k closest neighbor from x in D,
2. Determine the class or the value of x from the k closest neighbors.

For a classification problem with N class we have:

$$R^* \leq R_{kNN} \leq R^* \left(2 - \frac{N}{N-1} R^*\right)$$

where R^* is the Bayes error rate (minimum achievable error rate).
Note that for $N = 2$ we have :

$$R^* \leq R_{kNN} \leq R^* \left(2 - 2R^*\right) \approx 2R^*$$

Class/value computation step

Basics of Machine learning
Classic methods
From molecules to prediction
Neural Networks
Graph Neural Networks
Bibliography

Basic kNN

Classification: Majority vote among the k neighbors,

Regression: mean or median value.

Weighted kNN

Associate a weigh to each neighbor, e.g.:

$$w_i = \left(\frac{1}{d(x, x_i) + 1}\right)^p ; \tilde{w}_i = \frac{w_i}{\sum_{j=1}^{k} w_j}$$

So that $\sum_{i=1}^{k} \tilde{w}_i = 1$.

Classification: Ponderate the vote of each neighbor by its weigh.

Regression: compute a weighted mean.

Problems to solve

1. Choose k.
2. Choose a distance: Graph based distance, vector based distance (Euclidean, Manhatan, Minkowski),
3. Take care to execution times and to the curse of dimensionality if vectors en employed.

Choice of k

k is an hyper parameter of the method which may be chosen by trial and errors or by a cross-validation.

- A small value of k (e.g. $k = 1$) makes the algorithm sensible to outliers,
- a large value induce intensive computations and reduce the relevance of the answer (smooth the borders between classes).

Choice of a distance

d is a distance on X iff:

Separation: $d(x, y) = 0$ if and only if $x = y$ (otherwise it is a pseudo-distance),

Symmetry: $d(x, y) = d(y, x)$

Triangular inequality: $d(x, y) \leq d(x, z) + d(z, y)$ for any $z \in X$.

Graph based distances

- Graph edit distance
- Distance based on the size of the maximal common subgraph.

Vector based distances

- Euclidean distance (real fingerprints),
- Hamming distance (binary fingerprints). If $a = (a_i)_{i \in I}$, $b = (b_i)_{i \in I}$:

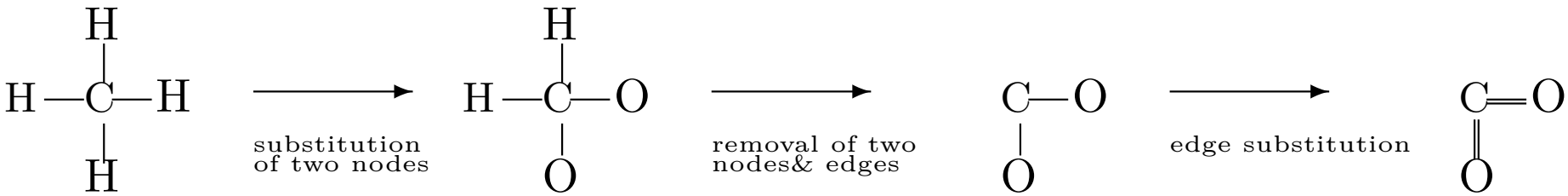
$$d(a, b) = |\{i \mid a_i \neq b_i\}|$$

Graph edit distance

$d(G_1, G_2)$ is the minimal cost of a set of operations transforming G_1 into G_2 using vertex/edge insertion/deletion/substitutions.

Definition (Edit path)

*Given two graphs G_1 and G_2 an **edit path** between G_1 and G_2 is a sequence of node or edge removal, insertion or substitution which transforms G_1 into G_2.*

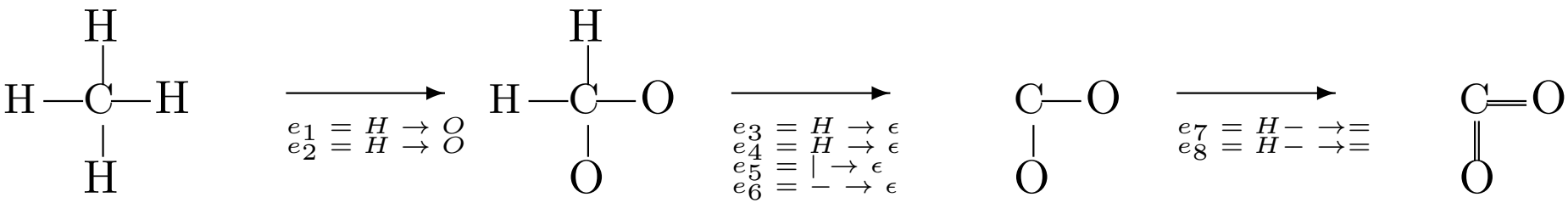


A substitution is denoted $u \to v$, an insertion $\epsilon \to v$ and a removal $u \to \epsilon$.

Costs

All paths go to Roma... However we are usually only interested by the shortest one.
Let $c(.)$ denote the cost of any elementary operation. The cost of an edit path is defined as the sum of the costs of its elementary operations.

- All cost are positive: $c() \geq 0$,
- A node or edge substitution which does not modify a label has a 0 cost: $c(l \rightarrow l) = 0$.



If all costs are equal to 1, the cost of this edit path is equal to 8.

Graph edit distance

Definition (Graph edit distance)

The graph edit distance between G_1 and G_2 is defined as the cost of the less costly path within $\Gamma(G_1, G_2)$. Where $\Gamma(G_1, G_2)$ denotes the set of edit paths between G_1 and G_2.

$$d(G_1, G_2) = \min_{\gamma \in \Gamma(G_1, G_2)} \sum_{e \in \gamma} c(e)$$

- The exact computation is NP-hard. Usually computed thanks to A^* algorithms with exponential complexity.
- Heuristics exist allowing to compute approximate GED in $\mathcal{O}(min(n, m).max(n, m)^2)$ where n and m are the size of both graphs [Blumenthal et al., 2020].

Distances based on maximum common subgraphs

A Maximum common induced subgraph (mcs), is a graph that is an induced subgraph of two given graphs and has as many vertices as possible.
A distance may be defined as follow from it:

$$d(G_1, G_2) = 1 - \frac{|mcs(G_1, G_2)|}{max\left(|G_1|, |G_2|\right)}$$

Using bytes instead of graphs, it would correspond to the number of common '1' of two bytes divided by the maximal number of '1' of both bytes.

Connection to GED

If costs are constants and $c(u \to v) \geq c(u \to \epsilon) + c(v \to \epsilon)$ then :

$$GED(G_1, G_2) = cte - |\hat{V}_1|(c_{vd} + c_{vi}) + |\hat{E}_1|(c_{ed} + c_{ei})$$

where $(\hat{V}_1, \hat{E}_1) = mcs(G_1, G_2)$.

Execution times

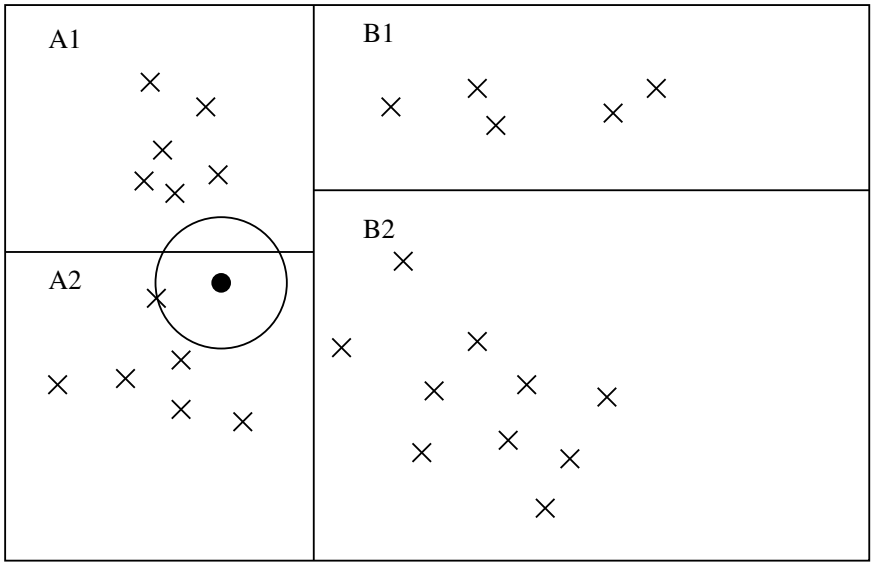
The brute force computation of the k closest neighbors require $\mathcal{O}(n)$ distance calculus (where n is the size of the training set).

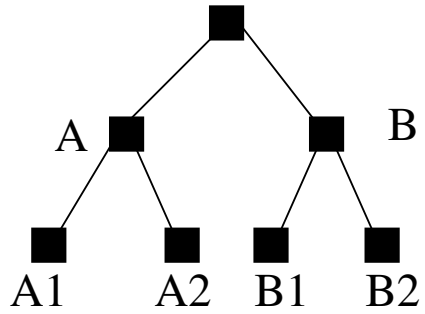
Use the distance to structurate the space and avoid as many unnecessary distance computations as possible.

Different structuration have been proposed for vector based representations:

- k-d tree [Sproull, 1991], based on a recursive split of the space into homogeneous clusters by cutting each cluster by an hyper plane until each cluster contains k molecules.
- R-tree/R^*-trees, Hilbert R-tree/PR-trees [Arge et al., 2008] insert one by one data into the leafs of a tree where each tree encodes a rectangle. The insertion in the chosen leaf induce a minimal modification of its rectangle. When a leaf exceed a given capacity it is split into two new leafs (for e.g. of equal size).

kd tree: The search phase





- Mean search time: $\mathcal{O}(\log(|D|))$
- Worse search time: $\mathcal{O}(|D|)$.

Trees for graph based representations

Metric trees [Chávez et al., 2001] such as monotonic bisector trees (MBST) [Kalantari and McDonald, 1983] split recursively the space using cluster's representatives (e.g. median).

- A cluster $\mathcal{N} = (o, \mathcal{C}, r, \mathcal{N}_1, \mathcal{N}_2)$ satisfies $d(o, c) \leq r$ for all $c \in \mathcal{C}$. The sons $\mathcal{N}_1$ and $\mathcal{N}_2$ are defined as:
  - $\mathcal{N}_1 = (o, r_1, \mathcal{N'}_1, \mathcal{N'}_2)$ with $r_1 \leq r$ and
  - $\mathcal{N}_2 = (o_2, r_2, \mathcal{N''}_1, \mathcal{N''}_2)$ with $r_2 \leq r$.

Search

Let us suppose that we search for all molecules m such that $d(q, m) \leq \tau$, τ fixed. Given a cluster $\mathcal{N} = (o, \mathcal{C}, r, \mathcal{N}_1, \mathcal{N}_2)$, let us distinguish 4 cases:

MBST-1: $d(q, o) \geq \tau + r$,

MBST-2: $d(q, o) \leq \tau - r$,

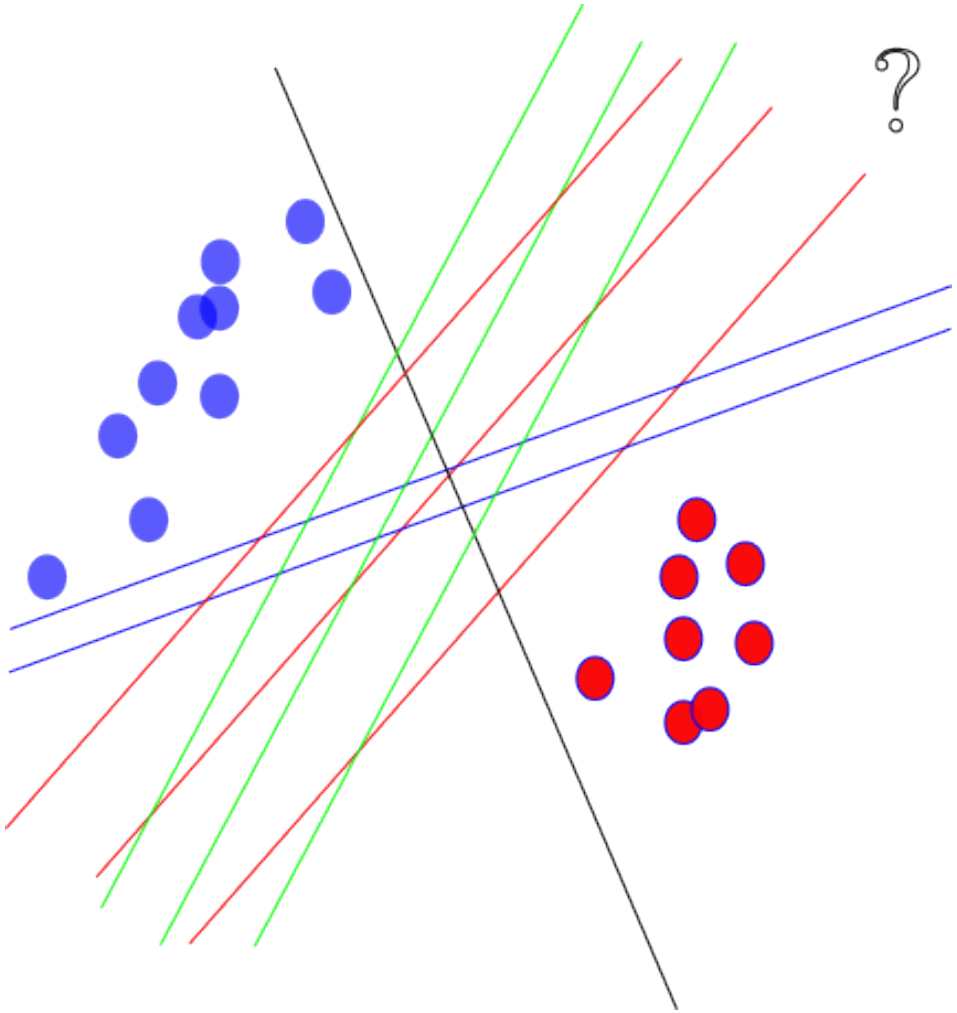
MBST-3: $\neg(1) \wedge \neg(2) \wedge \neg(\mathcal{N}_1 = \mathcal{N}_2 = null)$

MBST-3: $\neg(1) \wedge \neg(2) \wedge (\mathcal{N}_1 = \mathcal{N}_2 = null)$

In (1) $\mathcal{N}$ can be discarded, In (2) all elements of $\mathcal{N}$ can be accepted, In (3) search should be conducted on the sons of $\mathcal{N}$. Only in case (4) all elements of $\mathcal{N}$ must be checked.

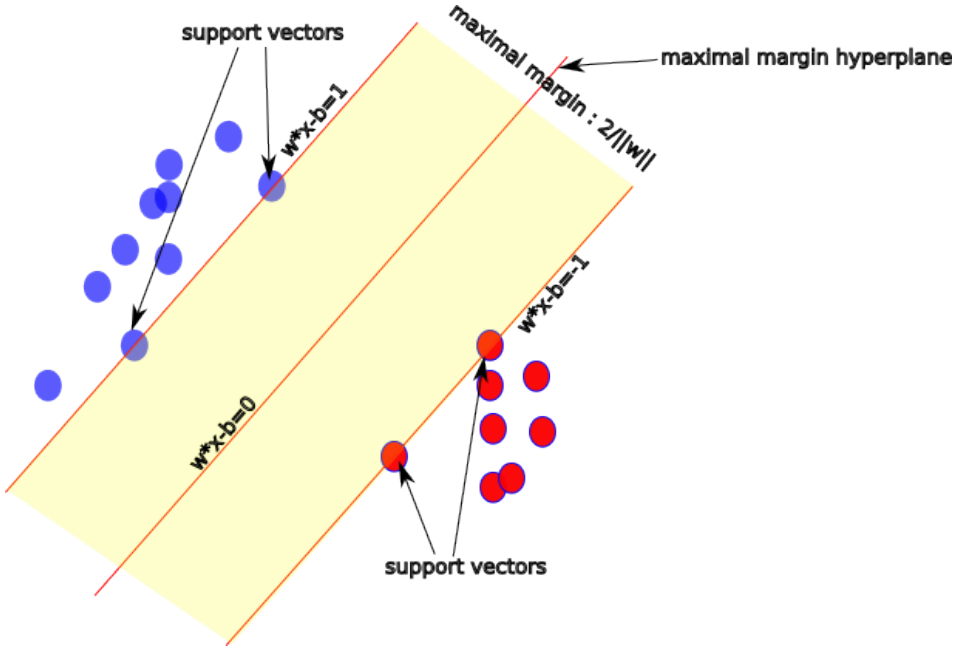
Classification problem

Classification is usually performed by dividing the feature space by an hyperplane.



What is a good hyper-plane ?

Support Vector Machine (SVM)



- Find the hyper plane with the maximal margin (solution unique).
- The only data which really matter are the support vectors.

SVM: a touch of theory

Let $\{(x_1, l_1), \ldots, (x_p, l_p)\}$ denote the training set with $l_i \in \{-1, 1\}$. Let $h(x) = w^T x + w_0$ denote the oracle. The oracle is perfect on the training set iff:

$$l_k h(x_k) \geq 0,\ 1 \leq k \leq p \text{ so iff } l_k(w^T x_k + w_0) \geq 0,\ 1 \leq k \leq p$$

- The margin of x_k is given by:

$$\frac{l_k(w^T x_k + w_0)}{\|w\|}$$

- We additionally suppose that the vectors of max (x^+_{marg})/resp. min (x^-_{marg}) satisfy:

$$\begin{cases} w^T x^+_{marg} + w_0 &= 1 \\ w^T x^-_{marg} + w_0 &= -1 \end{cases}$$

  The value of $l_k(w^T x_k + w_0) \geq 1$ for all k, 1 included.

SVM: a touch of theory

- We have thus:

$$min_k \left(\frac{l_k(w^T x_k + w_0)}{\|w\|} \right) = \frac{1}{\|w\|} min_k \left(l_k(w^T x_k + w_0) \right) = \frac{1}{\|w\|}$$

- Hence maximizing the margin is equivalent to:

$$Min_w \frac{1}{2}\|w\|^2 \text{ under the constraints } l_k(w^T x_k + w_0) \geq 1$$

- Using Lagrange multipliers we obtain the primal form of the SVM:

$$L(w, w_0, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{k=1}^{p} \alpha_k \left(l_k(w^T x_k + w_0) - 1 \right)$$

- Using partial derivatives of $L()$ according to w and w_0 we obtain:

$$\left\{ \begin{array}{lcl} \sum_{k=1}^{p} \alpha_k l_k x_k & = & w^* \\ \sum_{k=1}^{p} \alpha_k l_k & = & 0 \end{array} \right.$$

- Which gives the dual form of the SVM formulation:

$$Max_\alpha \tilde{L}(\alpha) = \sum_{k=1}^{p} \alpha_k - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j l_i l_j x_i^T x_j \text{ under } \alpha_k \geq 0 \text{ and } \sum_{k=1}^{p} l_k \alpha_k = 0$$

SVM: Few remarks

- One condition (KKT condition) hidden in Lagrangian from is that:

$$\forall k,\ \alpha_k \left(l_k((w^*)^T x_k + w_0) - 1 \right) = \alpha_k (l_k h(x_k) - 1) = 0$$

  Which means that at the optimum(w^*), either $l_k h(x_k) = 1$ (and x_k is a support vector) or $\alpha_k = 0$ and x_k plays no role in the solution.
- Both primal and dual forms only imply scalar products.

SVM: soft margin

If the data in the training set are not linearly separable no hyperplane can be found:

Replace the hard constraints by soft ones using slack variables.

$$l_k(w^T x_k + w_0) \geq 1 \rightsquigarrow l_k(w^T x_k + w_0) \geq 1 - \eta_k \text{ with } \eta_k \geq 0$$

Which leads to the minimization of the primal problem:

$$Min_w \frac{1}{2}\|w\|^2 + C\sum_{k=1}^{p} \eta_k$$

- C is an hyper-parameter of the model. Large C induce a better tolerance to outliers (restricting the margin), too large C induce an under exploitation of the training data.
- C is usually determined by cross validation with one or several validation sets.

Kernels : Definition

- A kernel k is a **symmetric** similarity measure on a set χ

$$\forall (x, y) \in \chi^2,\ k(x, y) = k(y, x)$$

- k is said to be **definite positive** (d.p.) iff k is symmetric and iff:

$$\left.\begin{array}{rcl} \forall (x_1, \ldots, x_n) & \in & \chi^n \\ \forall (c_1, \ldots, c_n) & \in & \mathbb{R}^n \end{array}\right\} \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0$$

- $K = (k(x_i, x_j))_{(i,j)\in\{1,\ldots,n\}}$ is the Gramm matrix of k. k is d.p. iff:

$$\forall c \in \mathbb{R}^n - \{0\},\ c^t K c \geq 0$$

Examples

If $\chi = \mathbb{R}^n$, classical kernels include:

- Linear kernel:

$$K(x, y) = x^t y$$

- Polynomial kernel

$$K(x, y) = (x^t y)^d + c,\ c \in \mathbb{R}, d \in \mathbb{N}$$

- Cosinus kernel:

$$K(x, y) = \frac{x^t y}{\|x\|\|y\|}$$

- Rational kernel:

$$K(x, y) = 1 - \frac{\|x - y\|^2}{\|x - y\|^2 + b}, b \in \mathbb{R} - \{0\}$$

- Gaussian Kernel

$$K(x, y) = exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right), \sigma \in \mathbb{R} - \{0\}$$

Kernels and scalar products

Aronszajn 1950 :

A kernel k is d.p. on a space χ
if and only if
it exists

- one Hilbert space $\mathcal{H}$ and
- a function $\varphi : \chi \to \mathcal{H}$

such that:

$$k(x, y) =< \varphi(x), \varphi(y) >$$

Reproducing kernel Hilbert Space

- To each d.p. kernel is associated a functional Hilbert space $\mathcal{H}$ called the *Reproducing kernel Hilbert Space*.
- $\mathcal{H}$ is composed of functions of the form:

$$f(.) = \sum_{i=1}^{n} \alpha_i k(x_i, .)$$

  $\mathcal{H}$ is composed of functions mapping real values to objects $x \in \chi$.
- For any $f = \sum_{i=1}^{n} \alpha_i k(x_i, .)$ and $g = \sum_{i=1}^{m} \beta_i k(x_i, .)$:

$$< f, g >= \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k(x_i, x_j)$$

- The norm induced by the scalar product on $\mathcal{H}$ is defined as:

$$\|f\|_K^2 = \sum_{i=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j k(x_i, x_j)$$

Support Vector Regression/Classification

- Given a set of observations $(x_i, y_i) \in \chi \times \mathbb{R}$, support vector regression/classification methods aim to find $f^* \in \mathcal{H}$ such that:

$$f^* = \underset{f \in \mathcal{H}}{\operatorname{argmin}}\, C.Loss(x_i, y_i, f(x_i)) + \|f\|_K$$

- Where:
  - $Loss()$: is the loss function (attach to data) and
  - $\|f\|_K$: is a regularization term. Encodes the smoothness of f.
  - C is the tradeoff between both terms.
- Kernel design determines how classification/regression algorithms generalize from the training set.

A basic example

- Let $\chi = \mathbb{R}^2$ and $k(x, y) = (x^t y)^2 + 1$
- For any $x = (x_1, x_2)$ and $y = (y_1, y_2)$ we have:

$$\begin{aligned} k(x,y) &= (x_1 y_1 + x_2 y_2)^2 + 1 \\ &= 1 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 x_2 y_1 y_2 \end{aligned}$$

- The function φ from $\mathbb{R}^2$ to $\mathbb{R}^4$ defined by:

- Satisfies

$$k(x, y) =< \varphi(x), \varphi(y) >$$

A basic example

- Let $\chi = \mathbb{R}^2$ and $k(x, y) = (x^t y)^2 + 1$
- For any $x = (x_1, x_2)$ and $y = (y_1, y_2)$ we have:

$$\begin{aligned} k(x,y) &= (x_1 y_1 + x_2 y_2)^2 + 1 \\ &= 1 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 x_2 y_1 y_2 \end{aligned}$$

- The function φ from $\mathbb{R}^2$ to $\mathbb{R}^4$ defined by:

$$\varphi(x) = \begin{pmatrix} 1 \\ x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{pmatrix}$$

- Satisfies

$$k(x, y) = < \varphi(x), \varphi(y) >$$

A basic example

- Let $\chi = \mathbb{R}^2$ and $k(x, y) = (x^t y)^2 + 1$
- For any $x = (x_1, x_2)$ and $y = (y_1, y_2)$ we have:

$$\begin{aligned} k(x,y) &= (x_1 y_1 + x_2 y_2)^2 + 1 \\ &= 1 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 x_2 y_1 y_2 \end{aligned}$$

- The function φ from $\mathbb{R}^2$ to $\mathbb{R}^4$ defined by:

$$\varphi(x) = \begin{pmatrix} 1 \\ x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{pmatrix}$$

- Satisfies

$$k(x, y) = < \varphi(x), \varphi(y) >$$

- Remark: An hyperplane in $\mathcal{H} = \mathbb{R}^4$ corresponds to a quadric of $\mathbb{R}^2$.

$$< \varphi(x), n >= K \Rightarrow n_1 + n_2 x_1^2 + n_3 x_2^2 + n_4 \sqrt{2} x_1 x_2 = K$$

The Kernel Trick

- Linear classifier become non-linear using kernels

- Problem: φ is usually unknown.
- Many methods only need scalar product between data (not explicit coordinates) $\Rightarrow$ replace scalar product by kernel.
- E.g. k-NN:

$$\begin{aligned} d_K^2(x_1,x_2) &= \|\varphi(x_1)-\varphi(x_2)\|^2 \\ &= <\varphi(x_1)-\varphi(x_2),\varphi(x_1)-\varphi(x_2)> \\ &= <\varphi(x_1),\varphi(x_1)>+<\varphi(x_2),\varphi(x_2)>-2<\varphi(x_1),\varphi(x_2)> \\ d_K(x_1,x_2) &= k(x_1,x_1)+k(x_2,x_2)-2k(x_1,x_2) \end{aligned}$$

- Kernel trick
  - Algorithm defined in $\mathcal{H} \Rightarrow$ (linear methods,non linear separation),
  - Data stored in χ.

Interesting but so what...

Sommaire de la section

1 Basics of Machine learning
- Definition

2 Classic methods
- How to measure ML performances
- knn
- SVM
- Kernel Methods

3 From molecules to prediction
- Fingerprints
- Graph kernels
- Kernels based on infinite Bags
- Kernels based on finite Bags
- Application to chemoinformatics

4 Neural Networks
- Principles

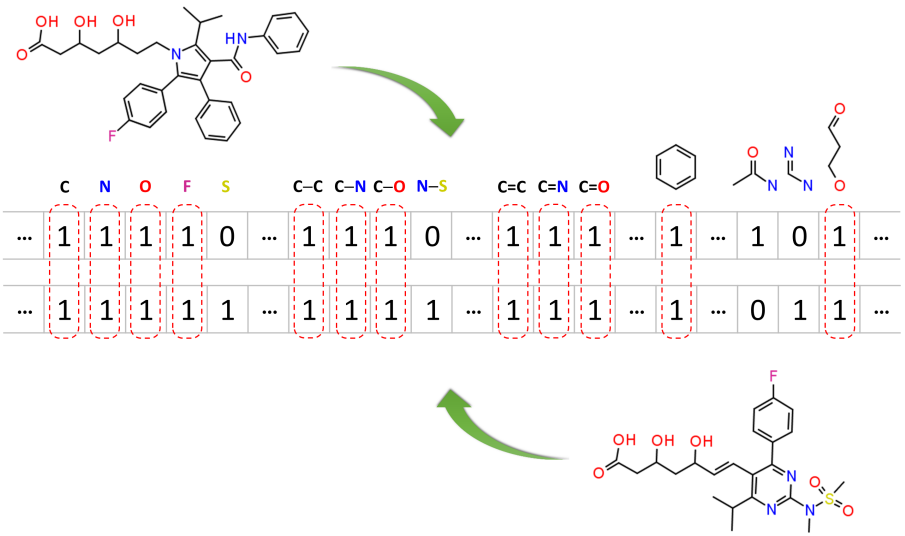
5 Graph Neural Networks
- Graph Aggregation
- Recurrent Graph Neural Networks
- Convolutional Graph neural networks
  - Spectral approaches

Fingerprints

The principle of a fingerprint consists to describe a molecule by a real or binary vector of fixed size.

- The mapping should be independent of the molecular representation (e.g. of the numbering of atoms),
- molecules with similar patterns should have close fingerprints.

Structural keys



Count or indicate the presence/absence of specified patterns.
Exemples:

- MACSS, use 166 keys, see `https://github.com/rdkit/rdkit-orig/blob/master/rdkit/Chem/MACCSkeys.py`,
- PubChem fingerprints 881-bit long structural keys, see `ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.pdf`

Topological Fingerprints

Basics of Machine learning
Classic methods
From molecules to prediction
Neural Networks
Graph Neural Networks
Bibliography

Atom pair fingerprint [Carhart et al., 1985]

- Atom type: tuples of atomic number, number of heavy atom neighbours, aromaticity and chirality.
- Atom pair substructure: (Atom type 1, Atom type 2, shortest 2D distance)
- Remove duplicates and use hash function to create a sparse count or bit vector (the fingerprint).

Topological torsion fingerprints [Nilakantan et al., 1987]

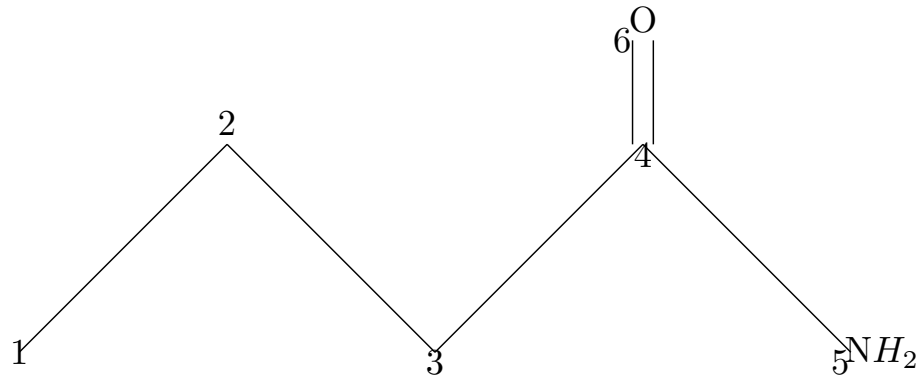
- Use the same atom types, but only consider sequences of 4 consecutive non hydrogen atoms: (Atom Type1,Atom Type 2, Atom Type 3, Atom Type 4).

Atom pair and Topological torsion get complementary information (short and long range).

Extended-Connectivity Fingerprints(ECFPs)

1. Compute for each atom a descriptor encoding atom's and atom's bonds properties. Use a hash function in order to derive an identifier from this list.
2. Each atom collect into an array its identifier and the ones of its neighbors. This list is again hash into an identifier. The process is iterated according to the size of the neighborhood we one to collect.
3. Remove duplicate identifiers (corresponding to a same substructure),
4. Create a binary fingerprint from the list of identifiers collected at step 2.

ECFP - step 1

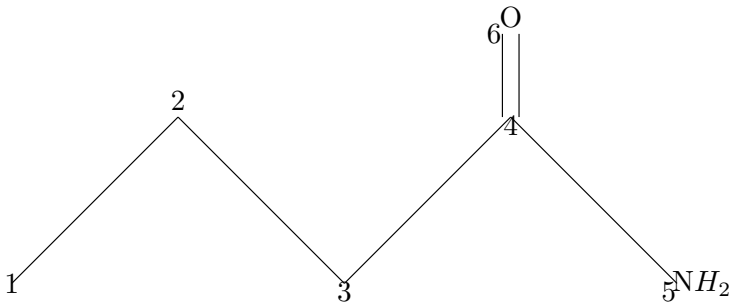


1. Compute features for e.g. using daylight atomic invariant:
   1. Number of non-hydrogen immediate neighbors
   2. Valency minus the number of connected hydrogens (in other words, total bond order ignoring bonds to hydrogens),
   3. Atomic number
   4. Atomic mass
   5. Atomic charge
   6. Number of attached hydrogens (both implicit and explicit)
   7. Whether the atom is a part of at least one ring(1, if yes, 0 otherwise)

$$f_4 = (3, 4, 6, 12, 0, 0, 0)$$

2. Apply a hash function $\rightsquigarrow -2155244659601281804$ using python hash function.

ECFP - step 2



1. -4080868480043360372
2. 831109852901413306 7
3. 831109852901413306 7
4. -2155244659601281804
5. -3602994677767288312
6. 8573586092015465947

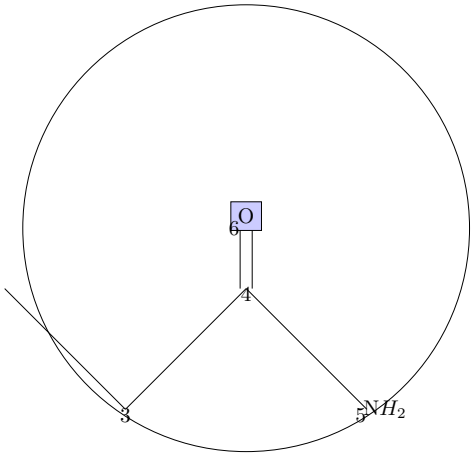
1. Initialize a list of neighbors for each atom. E.g. for atom 4, [(1, -2155244659601281804)].
2. For each non hydrogen neighbor, add a couple composed of the type of the bound and the id of the neighbor. For atom 4, it gives:

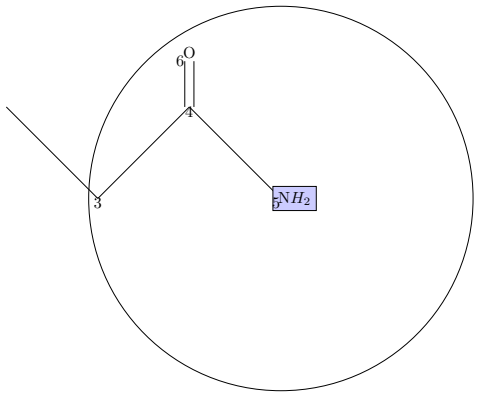
$$[(1, -215\ldots), (1, -360\ldots), (1, 831\ldots), (2, 857\ldots)]$$

   1, 2, 3: simple, double, triple bounds, 4 aromatics.
3. Convert each list of couple to a simple list and compute a hash.
4. Iterates for a given number of iterations. After each iteration concatenate the list of obtained identifier.

ECFP - step 3

1. Remove identical identifiers (for e.g. for nodes 2 and 3 in the previous example),
2. Determine if two different identifiers may correspond to a same substructure (oxygen and nitrogen at iteration 2).





We determine this by attaching to each identifier the list of edges of the corresponding structures. Two identifiers corresponding to a same list are considered as duplicates.

ECFP - step 4

The list of identifiers covers all substructure from the atom level to the given size of sub-patterns.

1. Set a size of fingerprint (e.g. size=1024),
2. Initialize a vector filled with 0 of the given size,

```
import numpy as np
fp = np.zeros(size)
```

3. For each identifier id: set a 1 at id%size in the fingerprint.

```
for id in identifiers:
        fp[id%size]=1
```

Note that collisions (different identifiers, same modulo) may occur.
The resulting vector register the existence of local substructures up to a given size. The number of such substructures is not registered.

Outline

Basics of Machine learning
Classic methods
From molecules to prediction
Neural Networks
Graph Neural Networks
Bibliography

Kernel and strutured data

The kernel trick provides an implicit embedding whose metric is defined from our similarity criterion (the kernel).

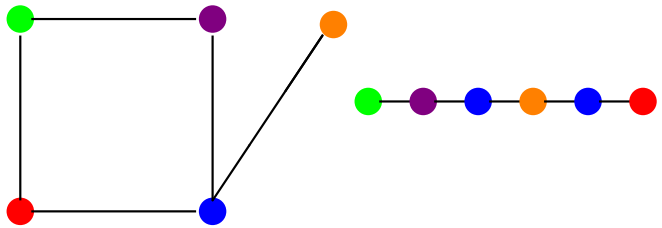
Do we absolutely need definite positiveness

- **No.**([Haasdonk, 2005]).
- results of SVM may be interpreted even with a non definite positive kernel (distance to convex hull).
- Several methods have been specifically designed to deal with non definite positive kernels.
- **But** ,
  - Results are usually more difficult to interpret (Krein space),
  - Mathematical properties are usually weaker,

Walk kernels

Basement paper: [Haussler, 1999].

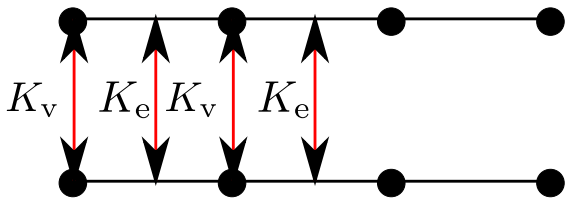
- Walks: Let $G = (V, E)$. $W = (v_1, \ldots, v_n)$ is a walk iff $(v_i, v_{i+1}) \in E, \forall i \in \{1, \ldots, n-1\}$.



is a walk

- Kernel between walks

$$K(h, h') = \begin{cases} 0 & \text{if } |h| \neq [h'| \text{ and} \\ K_v(v_1, v'_1).\Pi_{i=1}^{|h|} K_e(e_i, e'_i) K_v(v_{i+1}, v'_{i+1}) & \text{otherwize} \end{cases}$$



Walks kernels

- Walk kernels [Kashima et al., 2003] :

$$K(G_1, G_2) = \sum_{h \in \mathcal{W}(G_1)} \sum_{h' \in \mathcal{W}(G_2)} K(h, h') \lambda_{G_1}(h) \lambda_{G_2}(h')$$

- Covers different Graph kernels [Vishwanathan et al., 2010]:

$$If\ \lambda_G(h) = \begin{cases} 1\ iff\ |h| = n & K \text{is a nth order walk kernel} \\ P_G(h)(Markov\ RW) & K \text{is a random walk/marginalized kern} \\ \beta^{|h|} & K \text{is a geometric kernel} \end{cases}$$

$$P_G(h) = p_s(h_1)\Pi_{i=1}^{n} p_t(h_i|h_{i-1}) p_q(h_n) \text{ with } |h| = n$$

- Walks may induce totering problems: Walks with arbitrary length on the same set of edges and vertices.
- Framework extended to tree-pattern [Mahé and Vert, 2009].

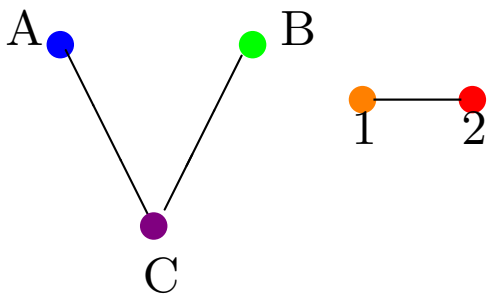
Walk kernel

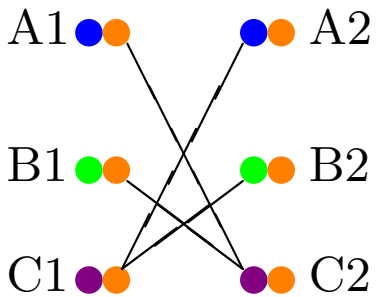
- Kronecker product
  Given two real matrices $A^{n\times m}$ and $B^{p\times q}$, the Kronecker product of A and B $(A \otimes B)$ belongs to $\mathbb{R}^{np\times mq}$ and is defined as:

$$A \otimes B = \begin{bmatrix} a_{1,1}B & a_{1,2}B & \dots & a_{1,m}B \\ \vdots & \vdots & & \vdots \\ a_{n,1}B & a_{n,2}B & \dots & a_{n,m}B \end{bmatrix}$$

- Tensor product graph Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the tensor product graph $G = G_1 \otimes G_2 = (V, E)$ is defined by:

$$\begin{aligned} V &= \{(v_1, v_2) \in V_1 \times V_2\} \\ E &= \{((v_1, v_2), (v_1', v_2')) \mid (v_1, v_1') \in E_1 \text{ and } (v_2, v_2') \in E_2\} \end{aligned}$$





Walk kernel

- Any walk in $G_1 \otimes G_2$ corresponds to a common walk in G_1 and G_2 and vice versa.
- If M_1 is the adjacency matrix of G_1 and M_2 the one of G_2, $M_1 \otimes M_2$ is the adjacency matrix of $G_1 \otimes G_2$.
- $\left((M_1 \otimes M_2)^k\right)_{i,j}$ encodes the number of common walks of length k between nodes $i = (v_1, v_2)$ and $j = (v_1', v_2)$.
- Matrices:

$$(I - \lambda M_1 \otimes M_2)^{-1} = \sum_{k=0}^{+\infty} \lambda^k (M_1 \otimes M_2)^k \text{ or } exp(\lambda M_1 \otimes M_2) = \sum_{k=0}^{+\infty} \frac{\lambda^k}{k!} (M_1 \otimes M_2)^k$$

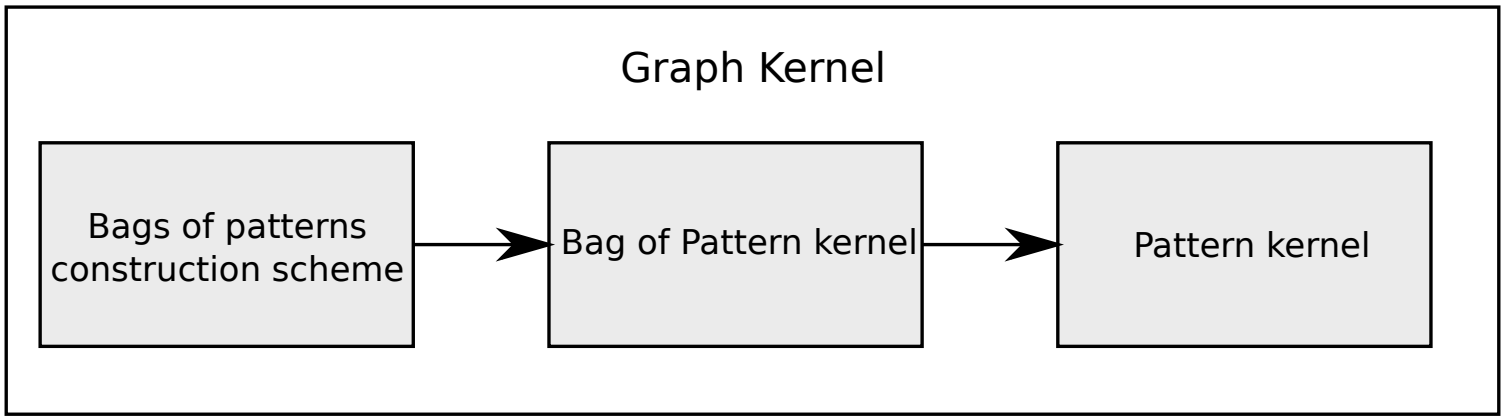
  encode the numbers of all common walks of G_1 and G_2 weighted by a factor depending of their length (λ^k or $\frac{\lambda^k}{k!}$).
- Kernel (e.g.):

$$K(G_1, G_2) = \sum_{i=1}^{np} \sum_{j=1}^{mq} exp(\lambda M_1 \otimes M_2)_{i,j}$$

Finite Bag kernels

$$\left.\begin{array}{rcl} G & \rightarrow & B(G) \\ G' & \rightarrow & B(G') \end{array}\right\} K(G,G') = K(B(G),B(G'))$$

Three independent step to design a graph kernel.

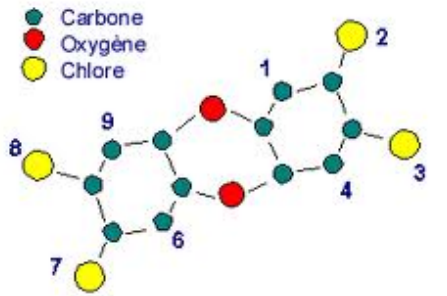


Graph kernels and chemoinformatics

- Aims : Predict the physical or biological properties of a molécule using the similarity principle :

  Two structurally similar molecules should have similar properties.

- Graph definition : $G = (V, E, \mu, \nu)$
  - μ encodes atom's type,
  - ν encodes types of bonds.

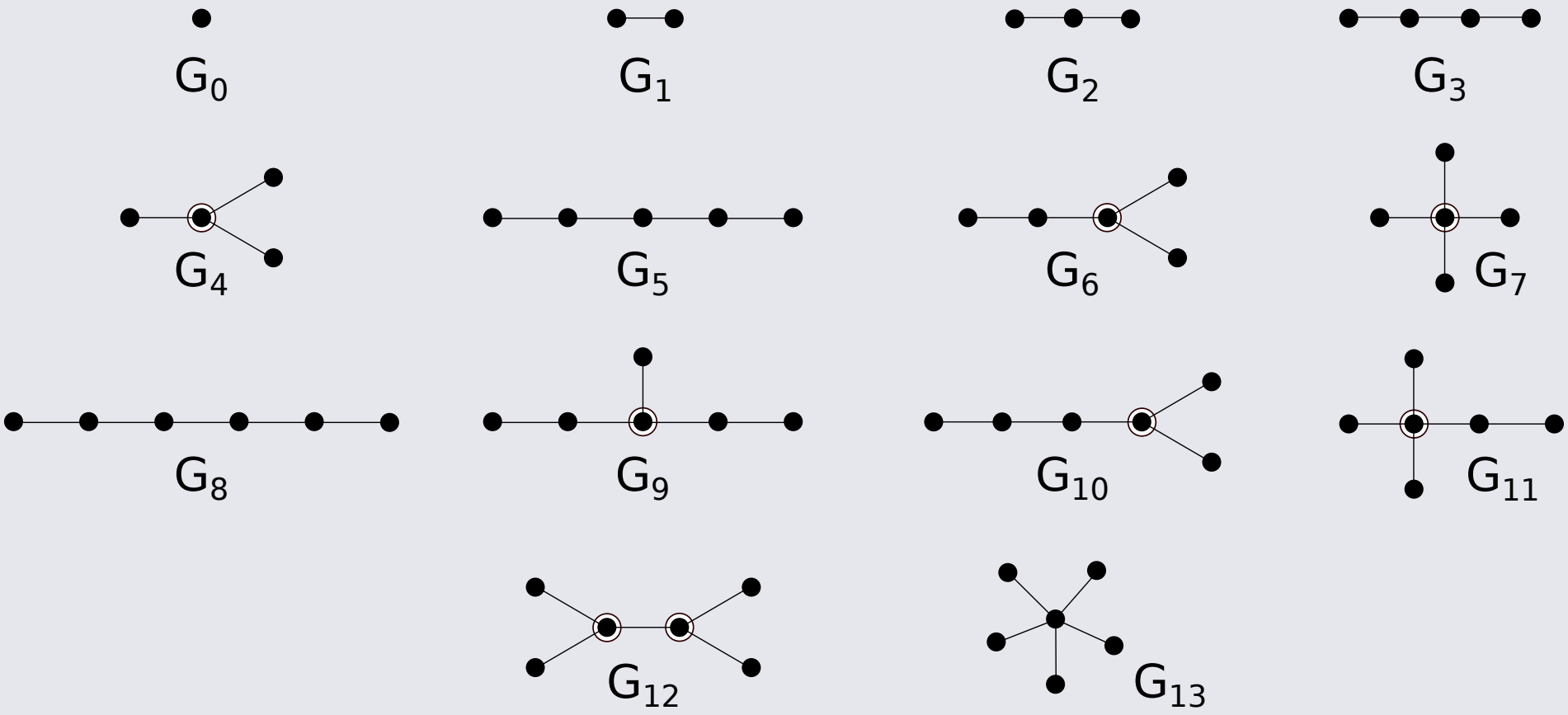


2,3,7,8-tétrachlorodibenzo-*p*-dioxine

Bag of patterns : unlabeled treelets

- One step beyong bag of paths : bags of treelets (trees of depth at most 6).

Treelets



Bag of labeled treelets

Treelet code

- Two parts :
  - Structural part : Treelet type $(G_0, G_1, \dots)$
  - Label part : Canonical traversal of the treelet

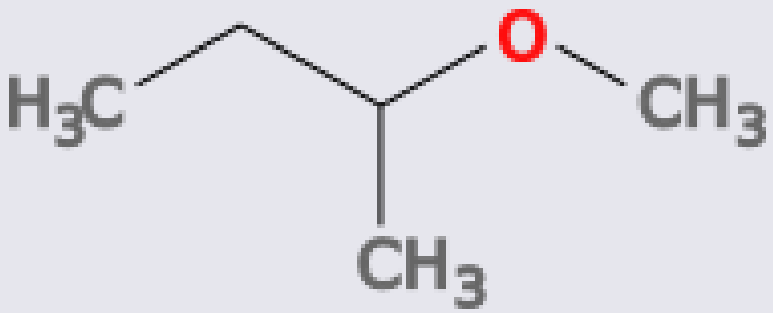


Figure: Code : G09-C1C1C1O1C1C

- $Code(G) = Code(G') \Leftrightarrow G \simeq G'$

Bag of labeled treelets

Treelet code

- Two parts :
  - Structural part : Treelet type $(G_0, G_1, \dots)$
  - Label part : Canonical traversal of the treelet

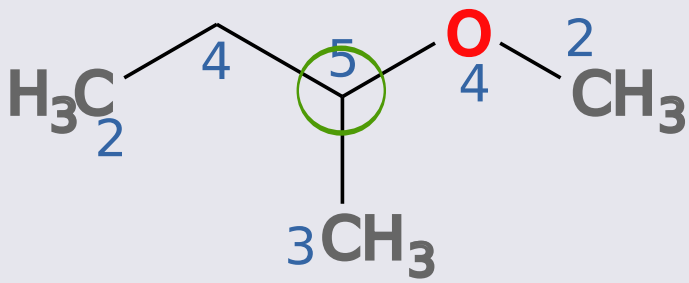


Figure: Code : G09-C1C1C1O1C1C

- $Code(G) = Code(G') \Leftrightarrow G \simeq G'$

Bag of labeled treelets

Treelet code

- Two parts :
  - Structural part : Treelet type $(G_0, G_1, \dots)$
  - Label part : Canonical traversal of the treelet

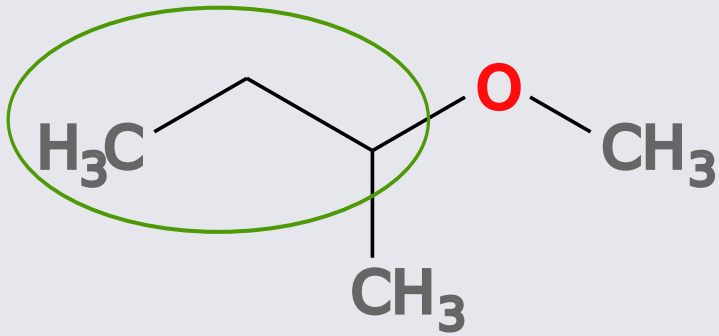


Figure: Code : G09-C1C1C1O1C1C

- $Code(G) = Code(G') \Leftrightarrow G \simeq G'$

Bag of labeled treelets

Treelet code

- Two parts :
  - Structural part : Treelet type $(G_0, G_1, \dots)$
  - Label part : Canonical traversal of the treelet

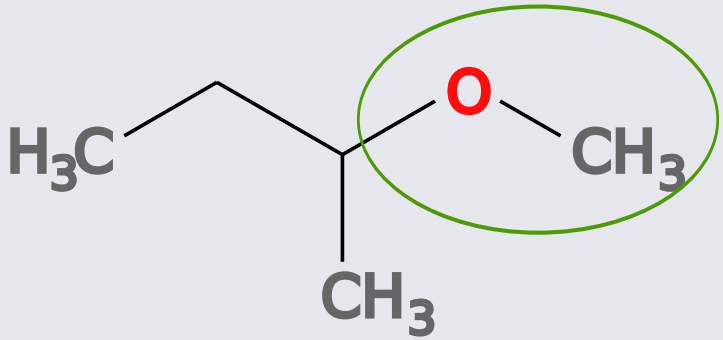


Figure: Code : G09-C1C1C1O1C1C

- $Code(G) = Code(G') \Leftrightarrow G \simeq G'$

Bag of labeled treelets

Treelet code

- Two parts :
  - Structural part : Treelet type $(G_0, G_1, \dots)$
  - Label part : Canonical traversal of the treelet

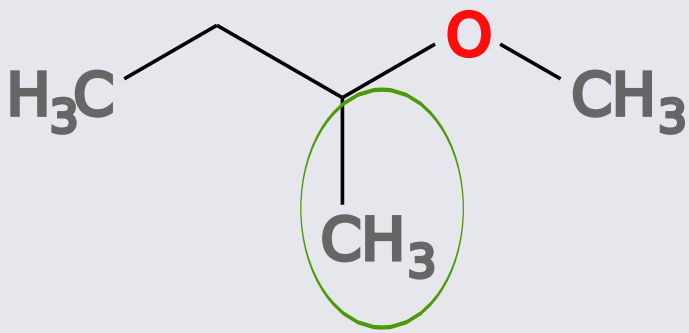
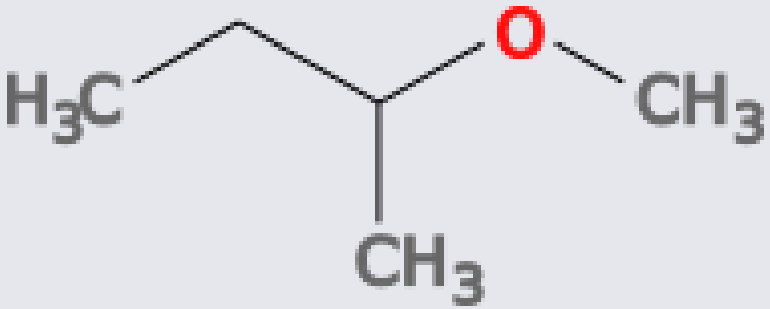


Figure: Code : C1C1C1O1C1C

- $Code(G) = Code(G') \Leftrightarrow G \simeq G'$

Kernel definition

From graph to histogram



(a) Molecule

$\tau(f_\tau(G))$		
C(5)	O(1)	
C-C(3)	C-O(2)	
C-C-C(2)	C-C-O(2)	C-O-C(1)
C-C-O(1) C		
⋮		

(b) Histogram

Kernel definition

$$K(G_1, G_2) = \sum_{\tau \in \mathcal{B}(G_1) \cap \mathcal{B}(G_2)} K(f_\tau(G_1), f_\tau(G_2))$$

where $K(.,.)$ is any kernel between real numbers (e.g. $K(x,y) = e^{-\frac{(x-y)^2}{\sigma^2}}$).

Lets go back on bag definition

- Some Facts :
  - Our first solution considers bags containing all patterns,
  - We do not have as in shape recognition a measure a priori of the relevance of a pattern,
  - The relevance of a given treelet should be fixed a posteriori given a dataset.
- MKL (Multiple Kernel Learning) method :

$$\text{If : } \quad K(x,y) = \sum_{i=1}^{n} w_i K_i(x,y)$$

  MKL allows to fix $(w_i)_{i\in\{1,\dots,n\}}$ optimally.
- Our kernel :

$$K(G_1,G_2) = \sum_{\tau\in\mathcal{B}(G_1)\cap\mathcal{B}(G_2)} K(f_\tau(G_1), f_\tau(G_2)) \overset{not.}{=} \sum_{\tau\in\mathcal{B}(G_1)\cap\mathcal{B}(G_2)} K_\tau(G_1,G_2)$$

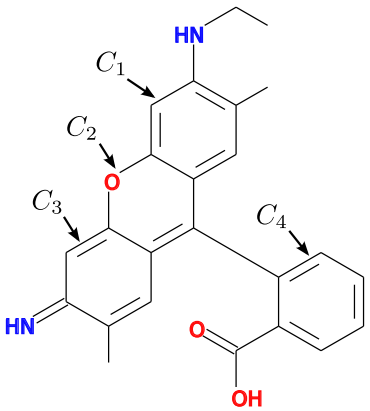
- A direct application of the MKL method provides the new kernel:

$$K(G_1,G_2) = \sum_{\tau\in\mathcal{B}(G_1)\cap\mathcal{B}(G_2)} w_\tau K(f_\tau(G_1), f_\tau(G_2))$$

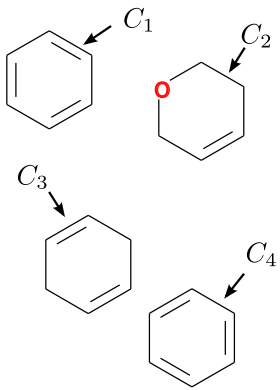
  where w_τ is defined using MKL.

Cycle information

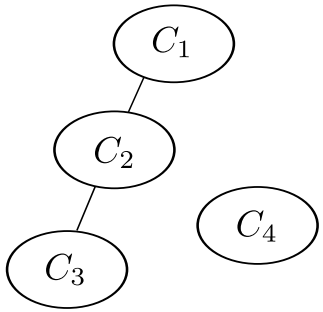
- Given $G = (V, E)$, define a graph $C = (V_C, E_C)$ such that:
  - $c \in V_C$ encodes a cycle of G.
  - $e \in E_C$ encodes an adjacency relationship between two cycles.
- Apply our treelet kernel on C and combine it with the one on G.



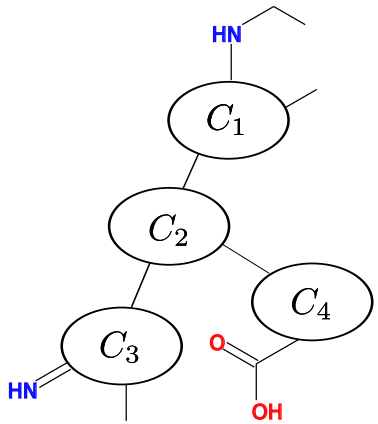
Molecular graph.



Relevant cycles(RC).



RC graph.



RC hypergraph.

Some results

Method	RMSE		Familly
Gaussian edit distance	10.27		Graph edit distance
Graph Embedding (Bunke)	10.19		Graph edit distance
Graph Embedding (EDM)	12.2		Graph edit distance
Kmean	12.24		Finite bag of paths
Random Walks	18.72		Infinite bag of walks
Tree Pattern Kernel	11.02		Infinite bag of tree pattern
Treelet Kernel (TK)	8.10		Finite bag of tree patterns
TK + Forward Selection	7.05		
TK + Backward Elimination	6.75		
Inter treelet kernel	5.89		
TK + MKL	**5.24**		

Table: Boiling point prediction on acyclic molecule dataset using 90% of the dataset as train set and remaining 10% as test set.

Some results

Method	RMSE	Times(s)	Familly
Gaussian edit distance	10.27	1.35	Graph edit distance
Graph Embedding (Bunke)	10.19		Graph edit distance
Graph Embedding (EDM)	12.2	7.21	Graph edit distance
Kmean	12.24		Finite bag of paths
Random Walks	18.72	19.10	Infinite bag of walks
Tree Pattern Kernel	11.02	4.98	Infinite bag of tree pattern
Treelet Kernel (TK)	8.10	**0.07**	Finite bag of tree patterns
TK + Forward Selection	7.05		
TK + Backward Elimination	6.75		
Inter treelet kernel	5.89		
TK + MKL	**5.24**		

Table: Boiling point prediction on acyclic molecule dataset using 90% of the dataset as train set and remaining 10% as test set.

Sommaire de la section

1 Basics of Machine learning
- Definition

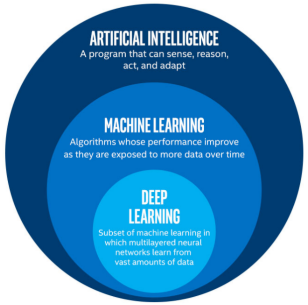
2 Classic methods
- How to measure ML performances
- knn
- SVM
- Kernel Methods

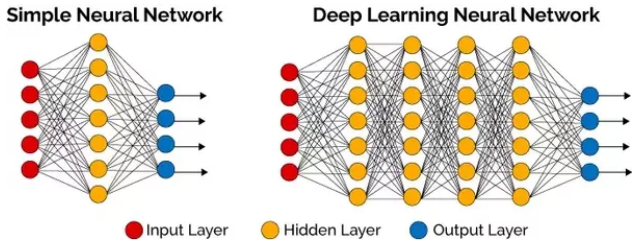
3 From molecules to prediction
- Fingerprints
- Graph kernels
- Kernels based on infinite Bags
- Kernels based on finite Bags
- Application to chemoinformatics

4 Neural Networks
- Principles

5 Graph Neural Networks
- Graph Aggregation
- Recurrent Graph Neural Networks
- Convolutional Graph neural networks
  - Spectral approaches

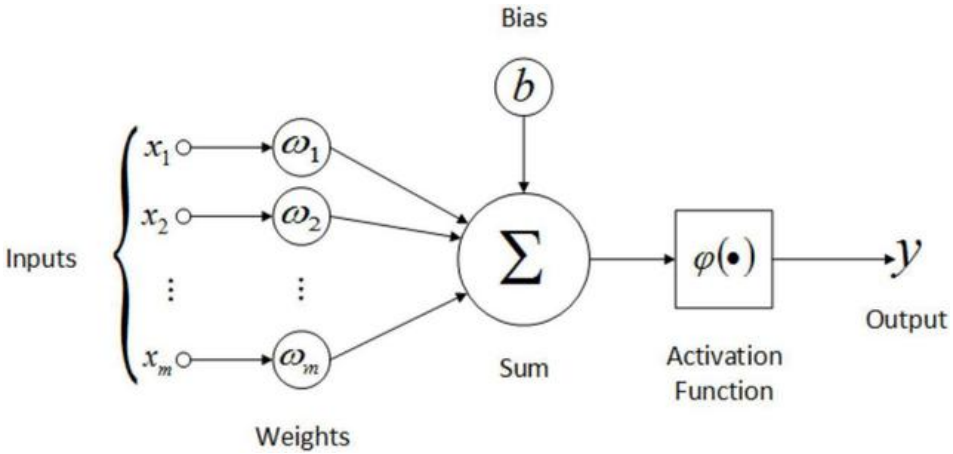
Deep learning





- Deep-learning is a part of machine learning,
- It avoids the necessity to design "good" features/distances/similarity functions,
- It usually requires more amount of data then "classical" ML models.

Lets have a look to a single neuron



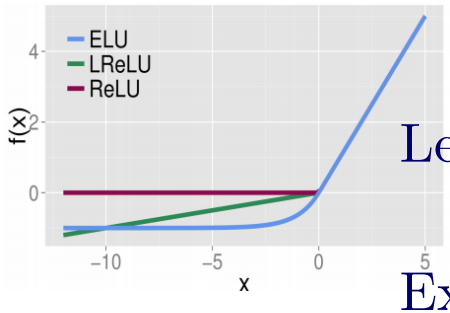
$$out = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

- w_i correspond to the weigh,
- b to the biais.
- ϕ is a non linear function.

Both (w_i) and b are learned.

Main non linear functions

Rectified Linear Unit:



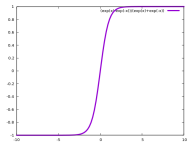
ReLU $F(x) = \max(0, x)$. Very popular due to its simplicity and efficiency.

Leaky ReLU: $F(x) = \max(\alpha x, x)$, with α usually equal to 0.01.

Exponential Linear Unit (ELU): $G(x) = x$ if $x \geq 0$, $e^x - 1$ otherwize.

tanh (Hyperbolic Tangent) :

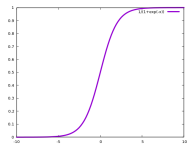
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Output between -1 and 1. Useful to focus the output around 0.

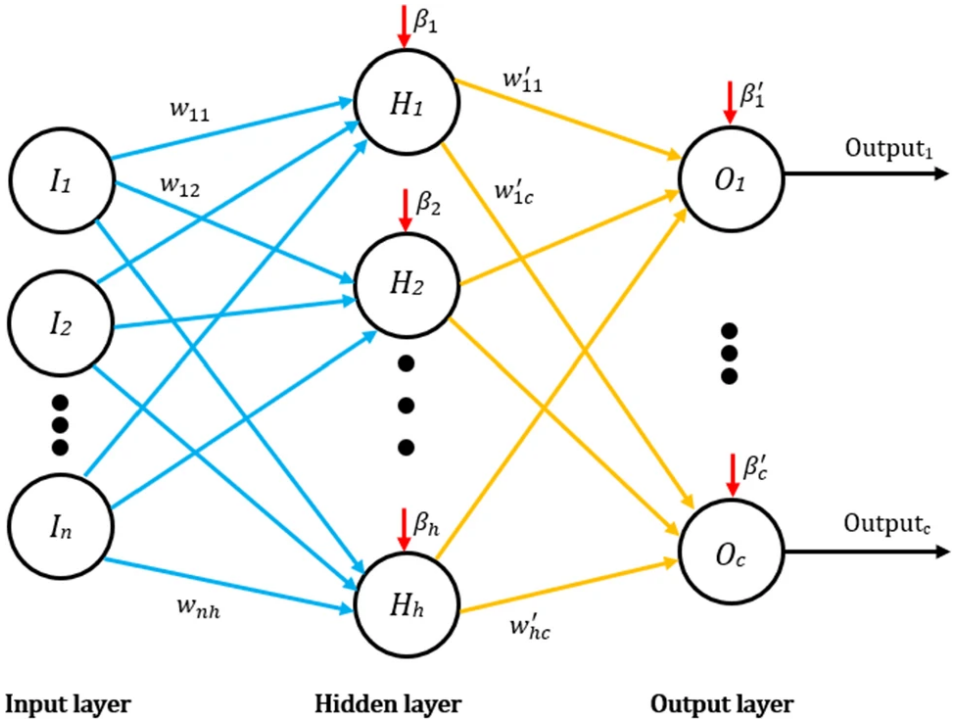
Sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Maps the input into $[0, 1]$. Often used for the output of linear binary classification models.

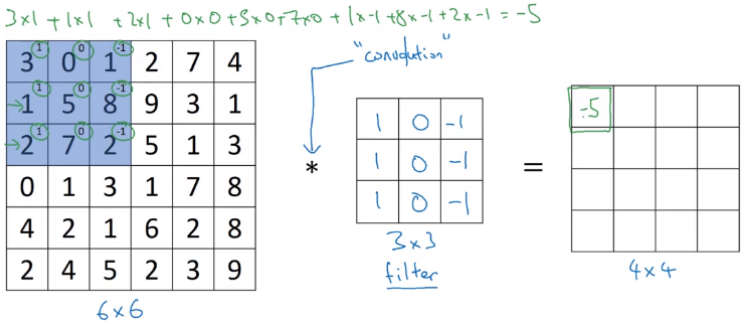
The MLP layer



$$H = \phi(WI + \beta) \text{ with } W = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ \vdots & \vdots & & \vdots \\ w_{h1} & w_{h2} & \dots & w_{hn} \end{pmatrix}$$

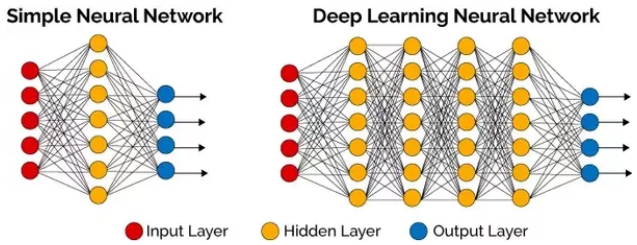
- Mainly used in the final classification stage.
- The number of output neurons depends on the classification task.

Convolution Neural Networks



- A convolution corresponds a filter with parametric size (e.g. $3 \times 3, 5 \times 5$) whose size is lower than the image's size,
- This filter is applied all along the image. This application is parameterized by:
  - Padding: do we allow convolution to be applied on pixels at theborders of the image by fixing missing values to a constant (usually 0),
  - stride: Defines the shift of the filter between two applications.

Main Loss functions

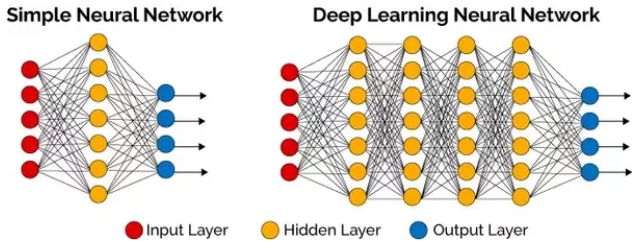


Loss

- Regression Loss:

  Mean Squared Error: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - gt_i)^2$
  Mean Absolute Error: $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - gt_i|$
  Hubert Loss: $HL = \frac{1}{n}\sum_{i=1}^{n} L(y_i, gt_i)$ where:
  - $L(y, gt) = \frac{1}{2}(y - gt)^2$ if $|y - gt| \leq \delta$,
  - $L(y, gt) = \delta \cdot (|y - t| - \frac{1}{2}\delta)$ otherwise.

Main Loss functions



Loss

- Classification Loss: y_i represents the ground true, p_i the prediction.
  Binary Cross-Entropy:

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right]$$

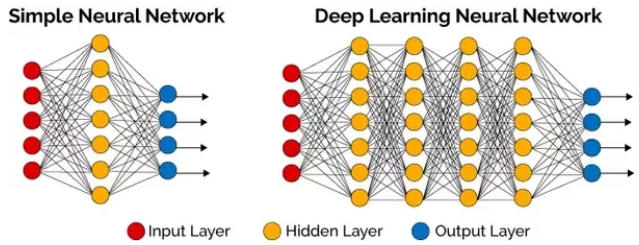
Usually used for binary classification.

Categorical Cross-Entropy: $CCE = \frac{1}{n}\sum_{i=1}^{n} L(y_i, p_i)$ where:

$$L(y_i, p_i) = -\sum_{j=1}^{C} y_{ij}\log(p_{ij})$$

for example with 3 classes: $y_i = [0, 1, 0], p_i = [0.2, 0.8, 0.0]$

Main Loss functions



Loss

- Metric Learning Loss functions:

  Contrastive Loss:

$$CL(x_i, x_j) = \delta_{ij} d^2(x_i, x_j) + (1 - \delta_{ij}) \max\left(0, (\epsilon - d(x_i, x_j))\right)^2$$

  where $\delta_{ij} = (y_i == y_j)$. ϵ minimal distance between elements of different classes.

  Triplet Loss: For each x let x^+ belong to the same class while x^- belongs to a different one:

$$L_{triplet}(x, x^+, x^-) = \sum_x \max(0, d(x, x^+) - d(x, x^-) + \epsilon)$$

Sommaire de la section

1. Basics of Machine learning
   - Definition
2. Classic methods
   - How to measure ML performances
   - knn
   - SVM
   - Kernel Methods
3. From molecules to prediction
   - Fingerprints
   - Graph kernels
   - Kernels based on infinite Bags
   - Kernels based on finite Bags
   - Application to chemoinformatics
4. Neural Networks
   - Principles
5. Graph Neural Networks
   - Graph Aggregation
   - Recurrent Graph Neural Networks
   - Convolutional Graph neural networks
     - Spectral approaches

Applications

The same as structural pattern recognition:

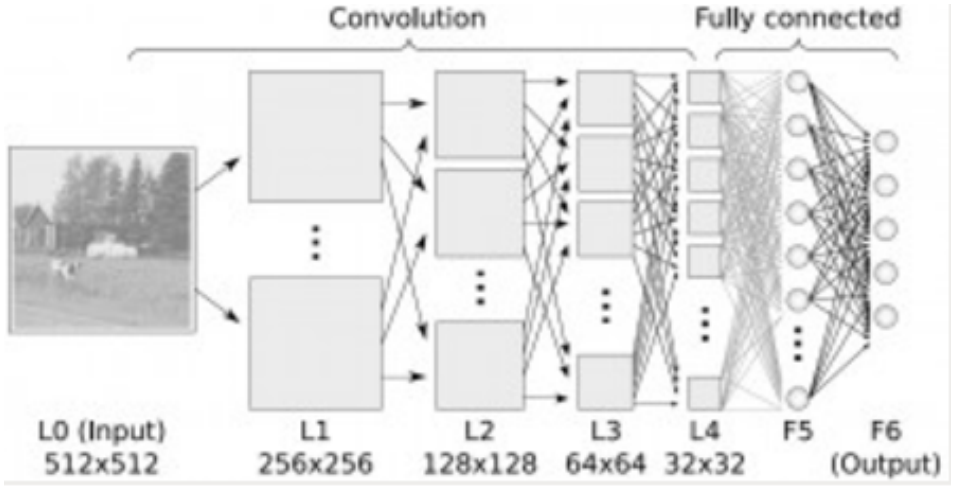
e-commerce : Find links between consumers and products (e recommandation),

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)

Graph autoencoders (GAEs): Encode nodes/graphs into a latent vector so as to be able to reconstruct the information (graph) from its latent representation.

Spatial temporal graph neural networks (STGNNs): 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,...)

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 Agregation

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

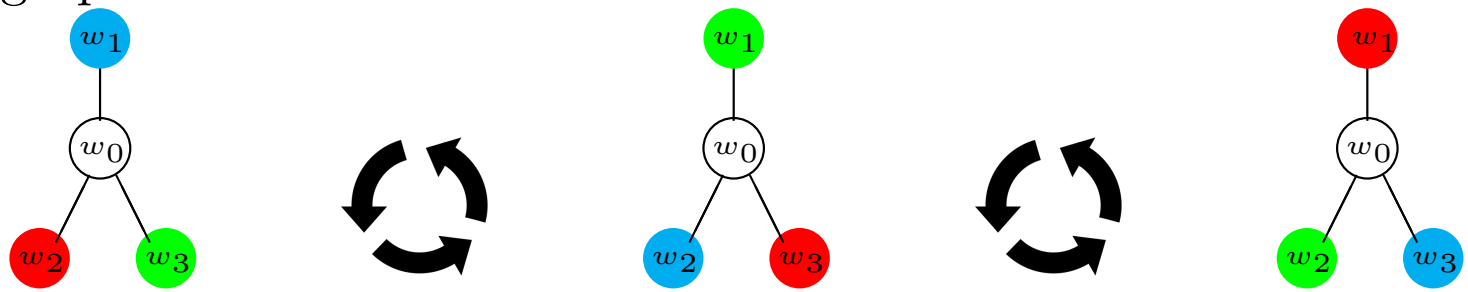
$$h_v^{(t)} = F(x_v, \{(x_u, x_{vu}^e, h_u^{(t-1)})\}_{u \in \mathcal{N}_v})$$

- 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

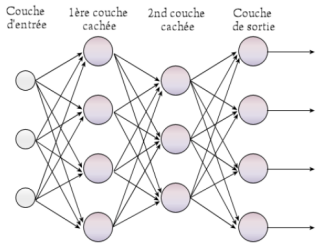
- Initially proposed for acyclic graphs:
- Basic idea: iterate an update of hidden values up to convergence:

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

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

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

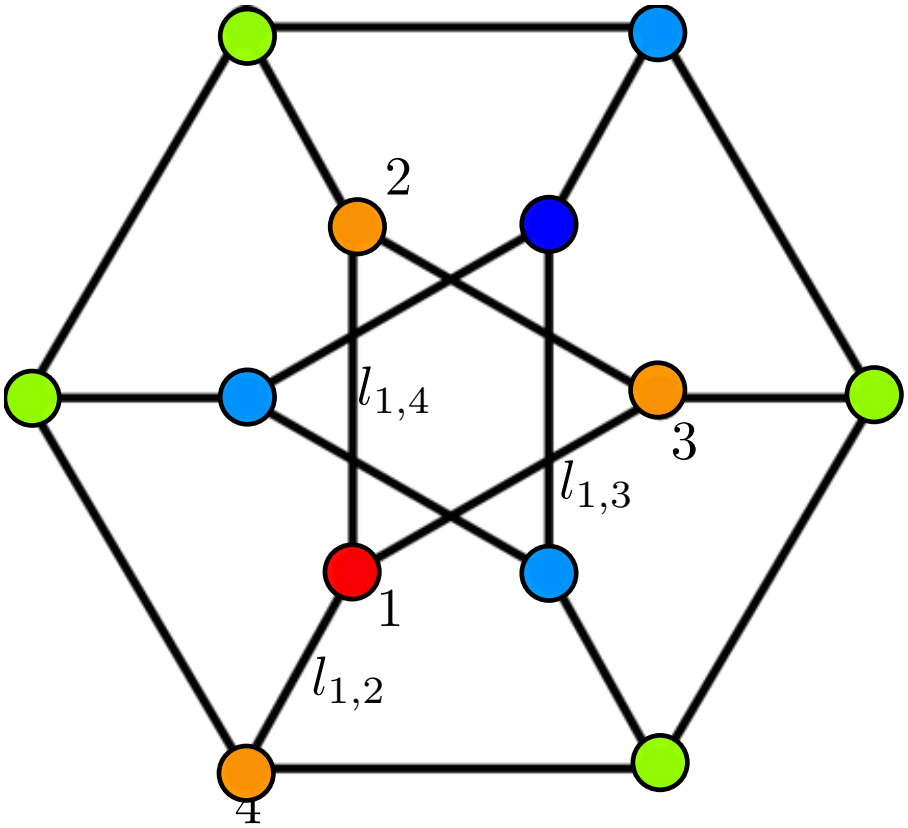
  - A MLP [Massa et al., 2006]



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 convergence.

Example



$t = 0$

$t = 1$

$t = 2$

$t = 3$

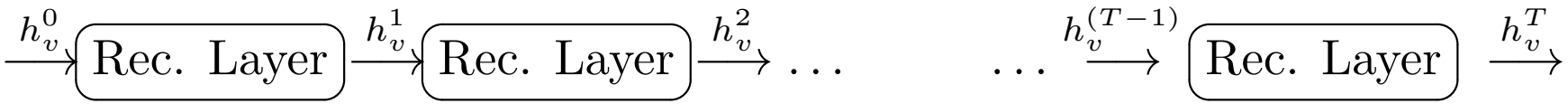
$t = 4$

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

Rec. and Conv. Graph neural networks

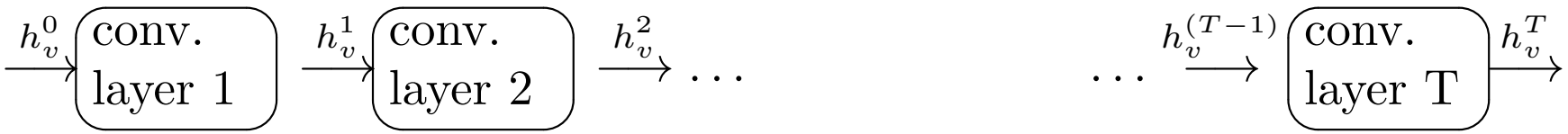
- Rec. Graph NN are iterative algorithms:

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



- 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:

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

- 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

[Defferrard et al., 2016]

- Problem: Computing L^i for $i \in \{0, \dots, K-1\}$ 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

[Levie et al., 2019]

- 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

[Kipf and Welling, 2017]

- 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

[Kipf and Welling, 2017]

- 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]

First methods

- 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] $M = A_{e_{v,w}} h_w^{(t-1)}$, $U = GRU(h_v^{(t-1)}, m_v^t)$ 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.
  $U(h_v^{(t-1}, m_v^t) = h_v^{(t-1)} + m_v^t, R = \sum_v NN(h_v^T)$.

Message Passing Networks

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

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

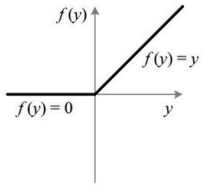
with $\sum_{m=1}^{M} q_m(u, v) = 1$

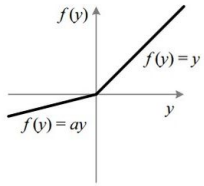
Graph attention Networks

[Velickovic et al., 2018]

- 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

[Zhang et al., 2018a]

- 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.

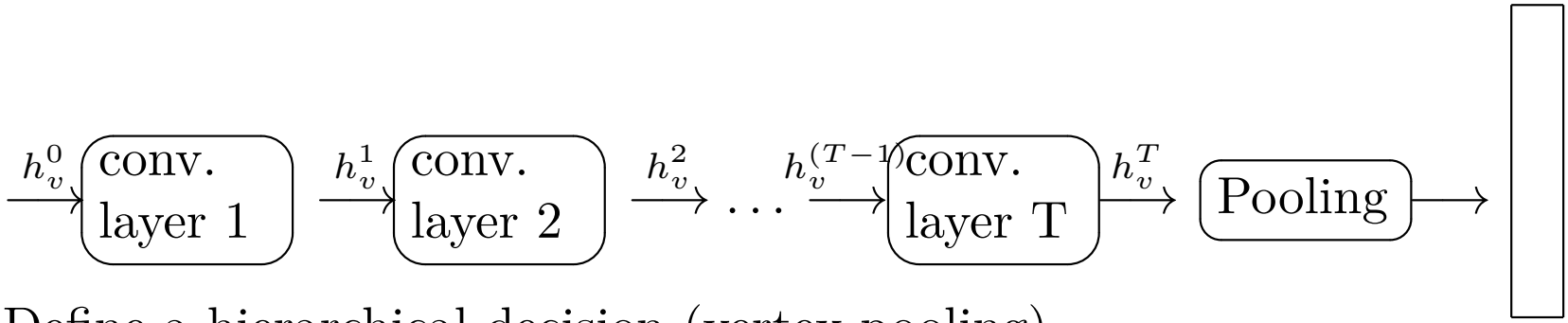
Spectral/Spatial approaches

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		✓

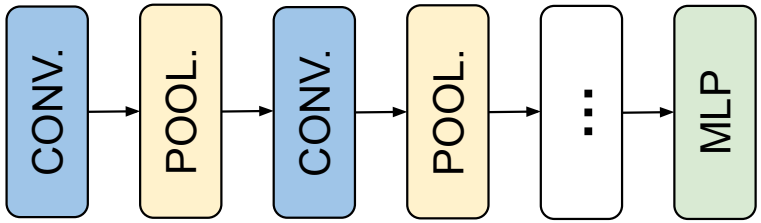
Graph pooling

A same name two meanings

- 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., 2018b]:
  - 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}.

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

- 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

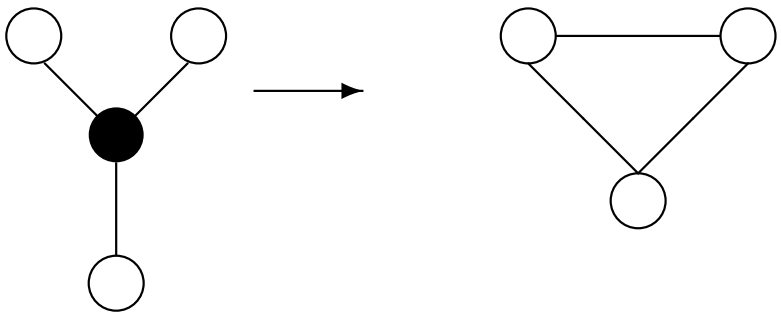
- Learning the decimation S^l is a good idea.
- S^l is not sparse and thus $A^{(l+1)}$ is almost complete.
- Learning S^l imposes to use graphs of fixed size.

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:

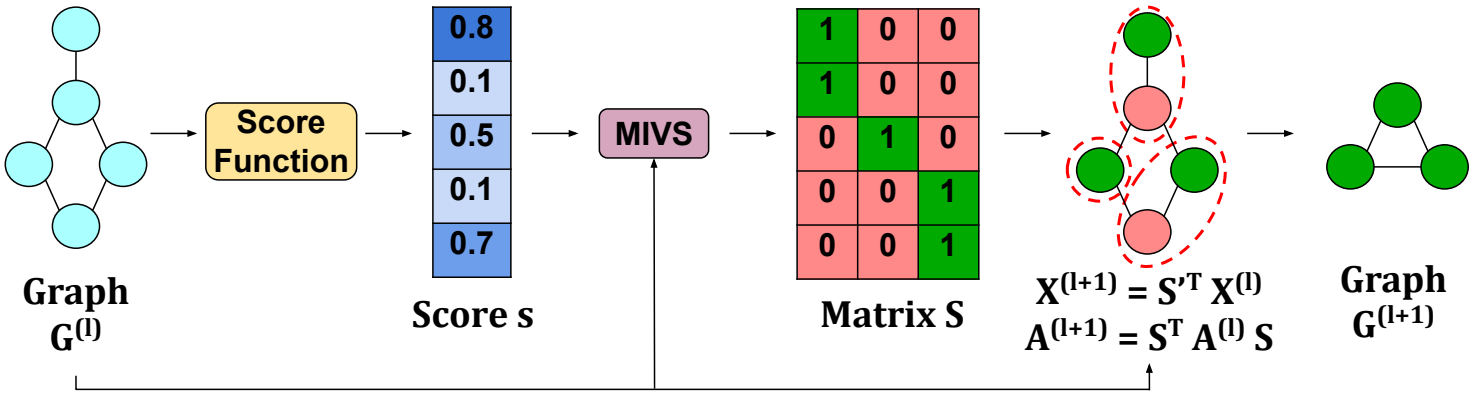
$$score = GNN_{l,pool}(A^l, h^l)$$

Combinaison GNNpool + relevance structurelle

Top-k/DiffPool: Pros/cons

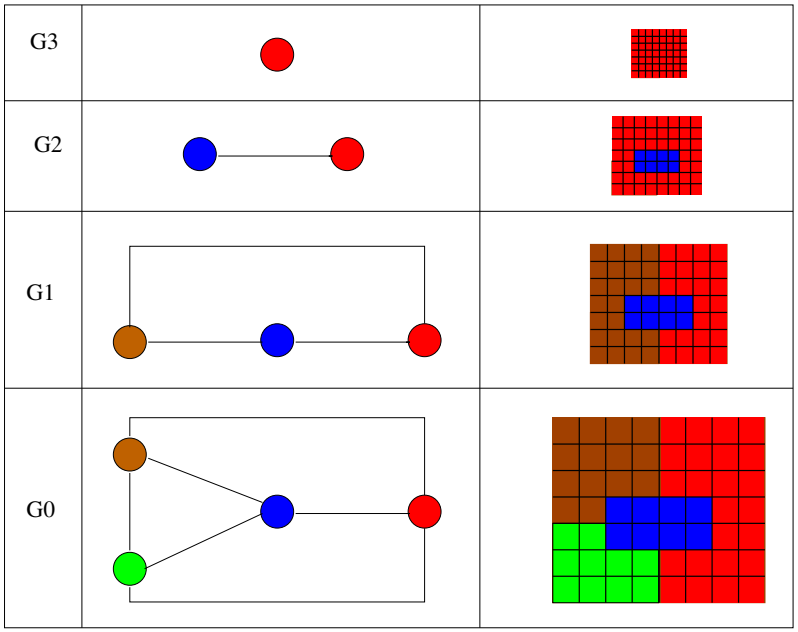
- DiffPool learns the attachment of non surviving vertices to surviving ones.
  - Dense matrices,
  - Graph of fixed size (the max one).
- Top-k learns the selection of surviving vertices but discard non survivning ones
  - May create disconnected graphs,
  - May ignore large parts of the graph,
  - 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.



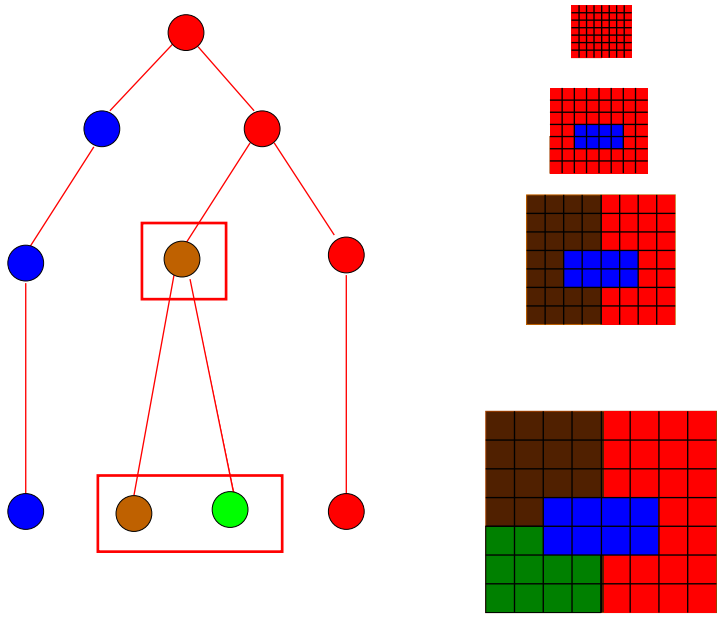
- Final results: sequence of reduced graphs $G_0, \ldots, G_n$

Reduction window

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

$$RW_i(v) = \{v_1, \ldots, v_n\} \subset V_{i-1}$$

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

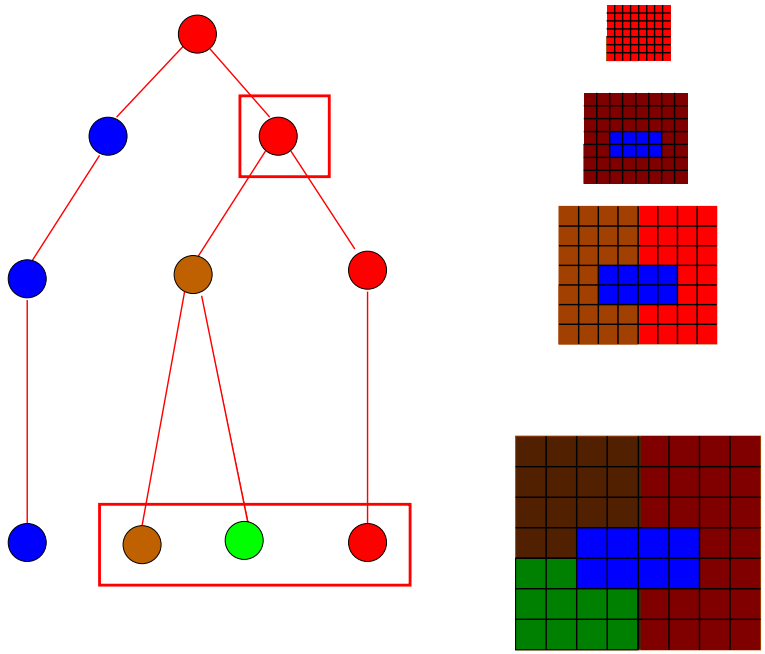


Receptive field

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

$$RF_i(v) = \bigcup_{v' \in RW_i(v)} RW_{i-1}(v') \subset V_0$$

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



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

Maximal Independent Set(MIVS)

- 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)

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

MIVS construction scheme

- Three variables :

  $p_i = true$ if v_i survives

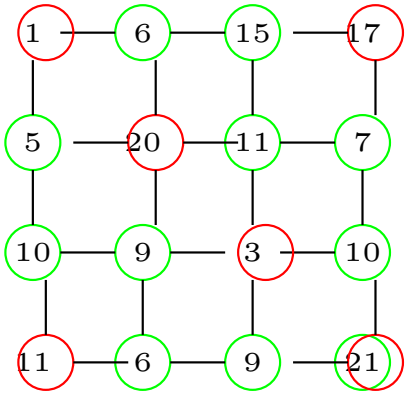
  $q_i = true$ if v_i may become a surviving vertex (he is candidate).

  x_i value of the vertex (function or random variable)

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

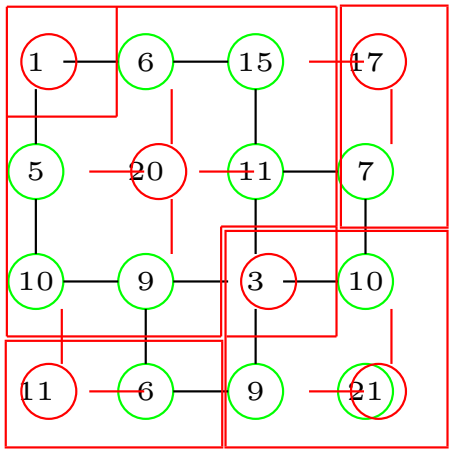
$$\begin{aligned} p_i^{(k+1)} &= p_i^{(k)} \vee (q_i^{(k)} \wedge x_i = max_{j \in V(v_i)}\{q_j^{(k)} x_j\}) \\ q_i^{(k+1)} &= \bigwedge_{j \in V(v_i)} \overline{p_j}^{(k+1)} \end{aligned}$$

MIVS : Father/child relationships



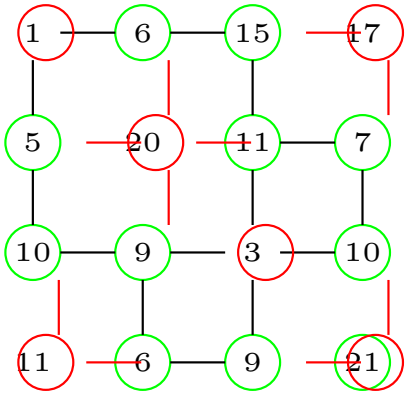
- link each non surviving vertex to one of its surviving neighbour ⇒ definition of the edges
- merge non surviving vertice to surviving ones along the selected edges(merge in simple graphs).

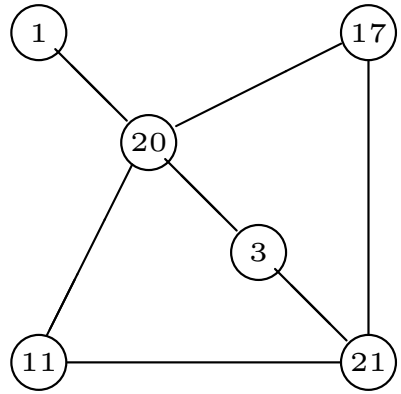
MIVS : Father/child relationships



- link each non surviving vertex to one of its surviving neighbour ⇒ 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

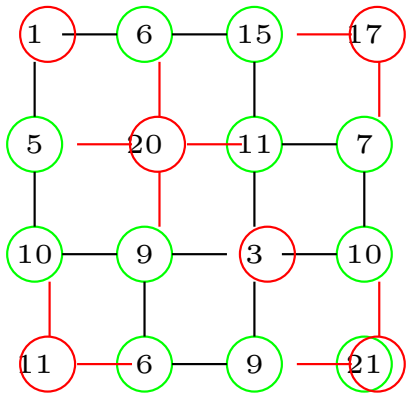
- perform one iteration of the kernel computation,
- attach each non surviving vertex to a surviving one
- merge vertices

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

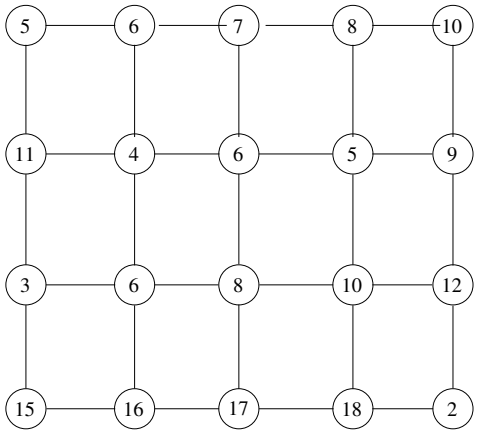
- only one step of the kernel computation is performed
  - "Corresponds" to a model of the behavior of our brain,
  - allows to avoid (in some cases) wrong merge operations.



Exercice : MIVS and D^3

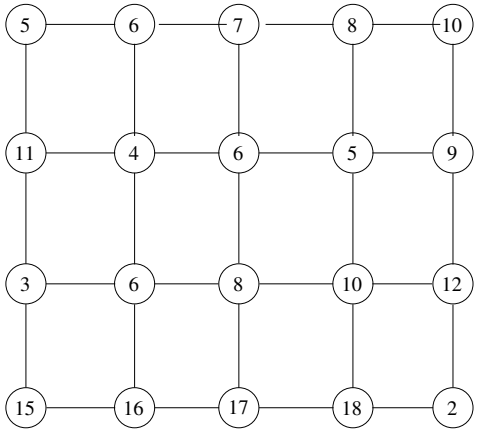
Define:

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



Exercice : MIVS and D^3

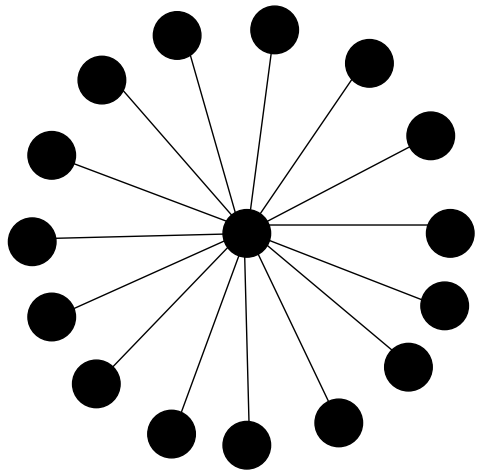
- Apply D^3 twice
- Legitimate father: max of r.v,



MIES : Motivations

Maximal Independent Edge Set

- 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.



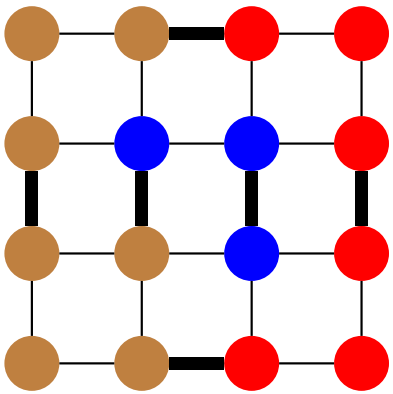
MIES : Motivations

Maximal Independent Edge Set

- 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.

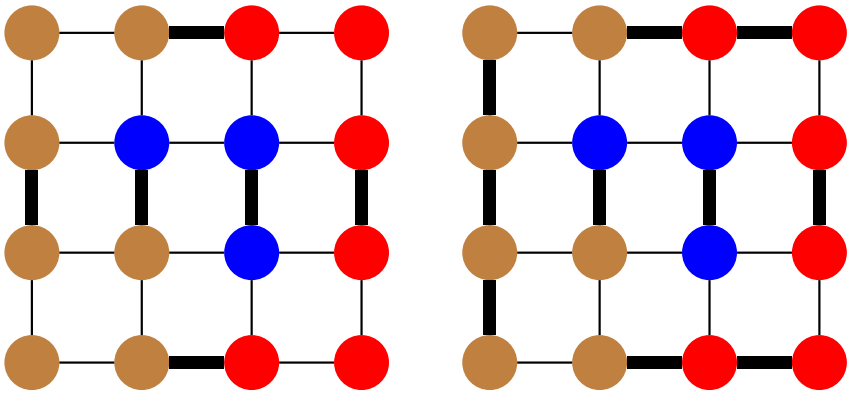
Maximal Matching

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

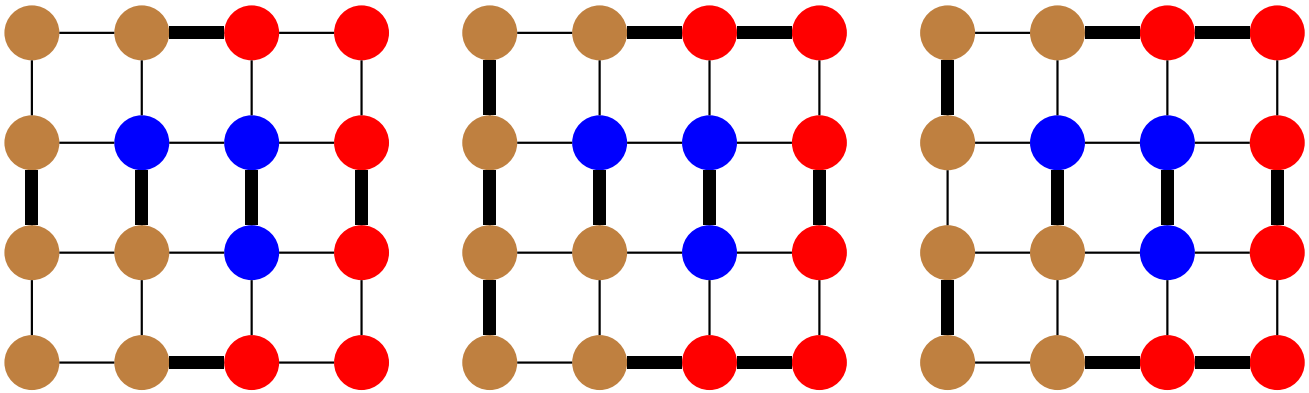


- A set $C \subset E$ is said to be a matching of $G = (V, E)$ if none of the edges of C are adjacent to a same vertex.
- A matching is said to be maximal if the addition of any edge breaks the matching property.
- A matching is said to be maximum if no larger matching may be found.

- Complete the matching C to C^+

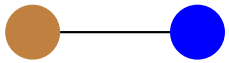
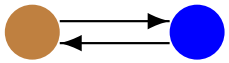


- Complete the matching C to C^+
- Remove edges from C^+ in order to obtain trees of depth 1.
- Merge vertice adjacent along selected edges.



Maximal Independent directed Edge Sets (MIDES)

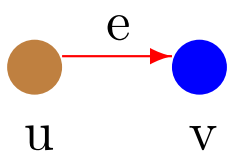
- How to design trees of depth 1 in one step ?
- How to take into account orientation of edges ?

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

Maximal Independent directed Edge Sets (MIDES)

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

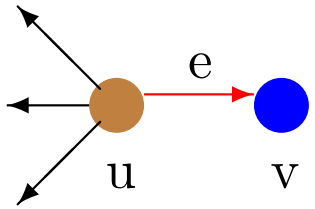
$$\mathcal{N}(e) = \mathcal{N}((u, v)) \quad =$$



Maximal Independent directed Edge Sets (MIDES)

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

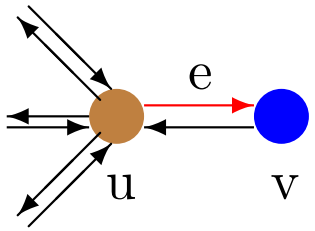
$$\mathcal{N}(e) = \mathcal{N}((u, v)) \quad = \quad \{(u, v') \in E\} \cup$$



Maximal Independent directed Edge Sets (MIDES)

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

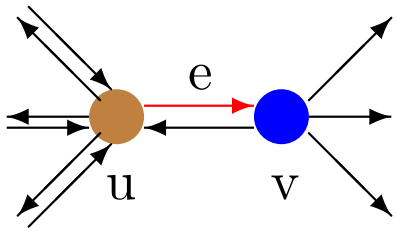
$$\begin{aligned} \mathcal{N}(e) = \mathcal{N}((u,v)) &= \{(u,v') \in E\} \cup \\ &\quad \{(u',u) \in E\} \cup \end{aligned}$$



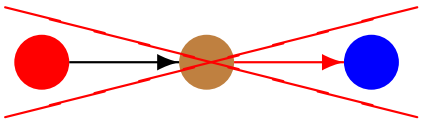
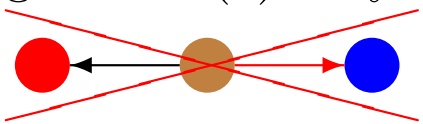
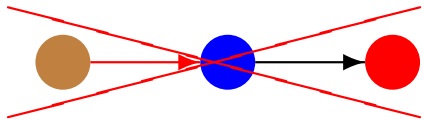
Maximal Independent directed Edge Sets (MIDES)

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

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

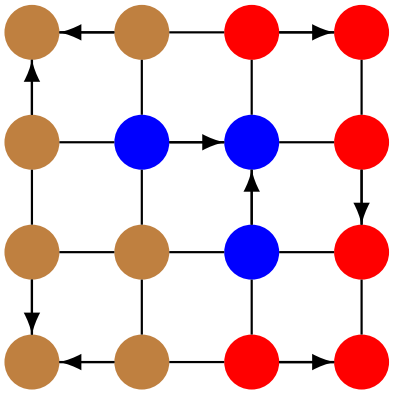


- If e is selected none of the edges of $N(e)$ may be selected.

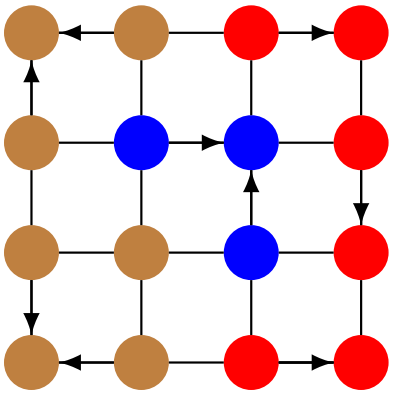
Maximal Independent directed Edge Sets (MIDES)

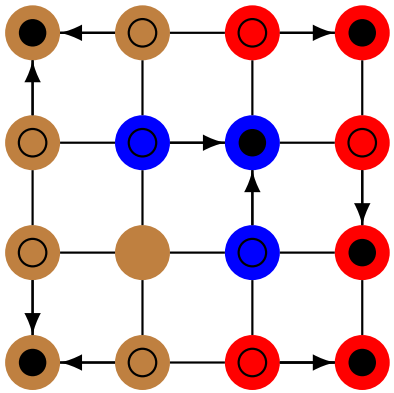
1. Apply a MIS on the edge graph $G = (E, E')$ with E' defined from $\mathcal{N}(E)$,



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,

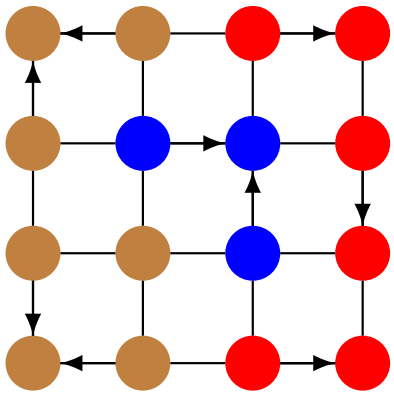
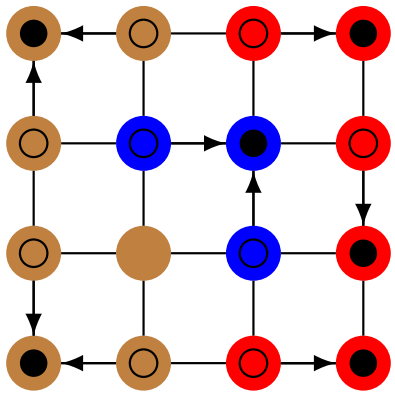
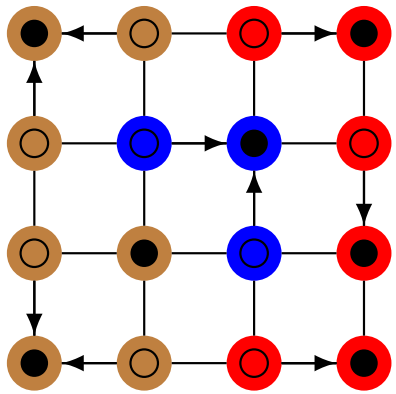




Maximal Independent directed Edge Sets (MIDES)

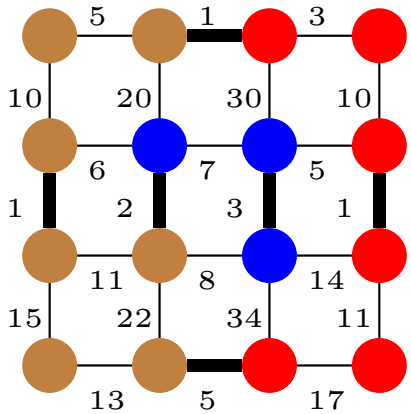
Algorithm

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.

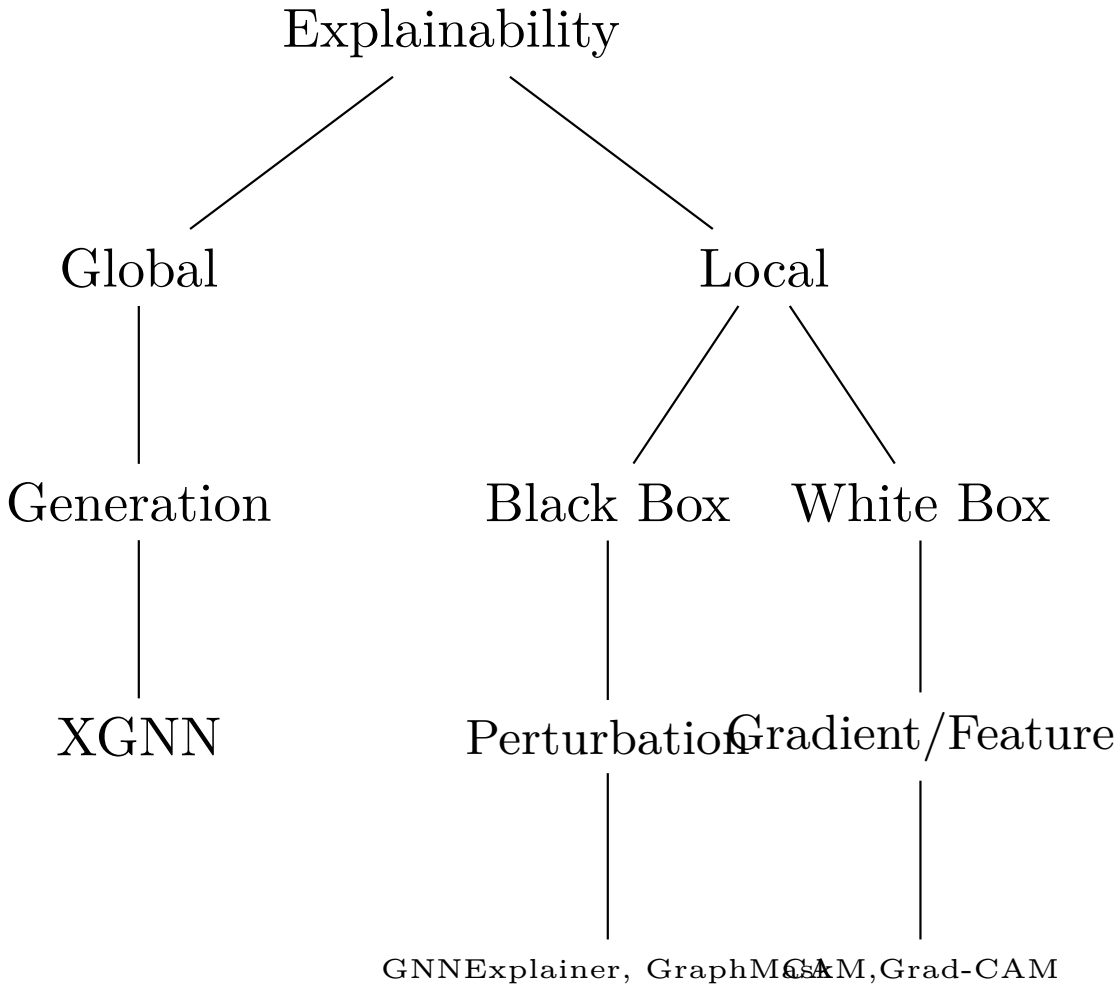




Data driven decimation and maximal matching

- 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).

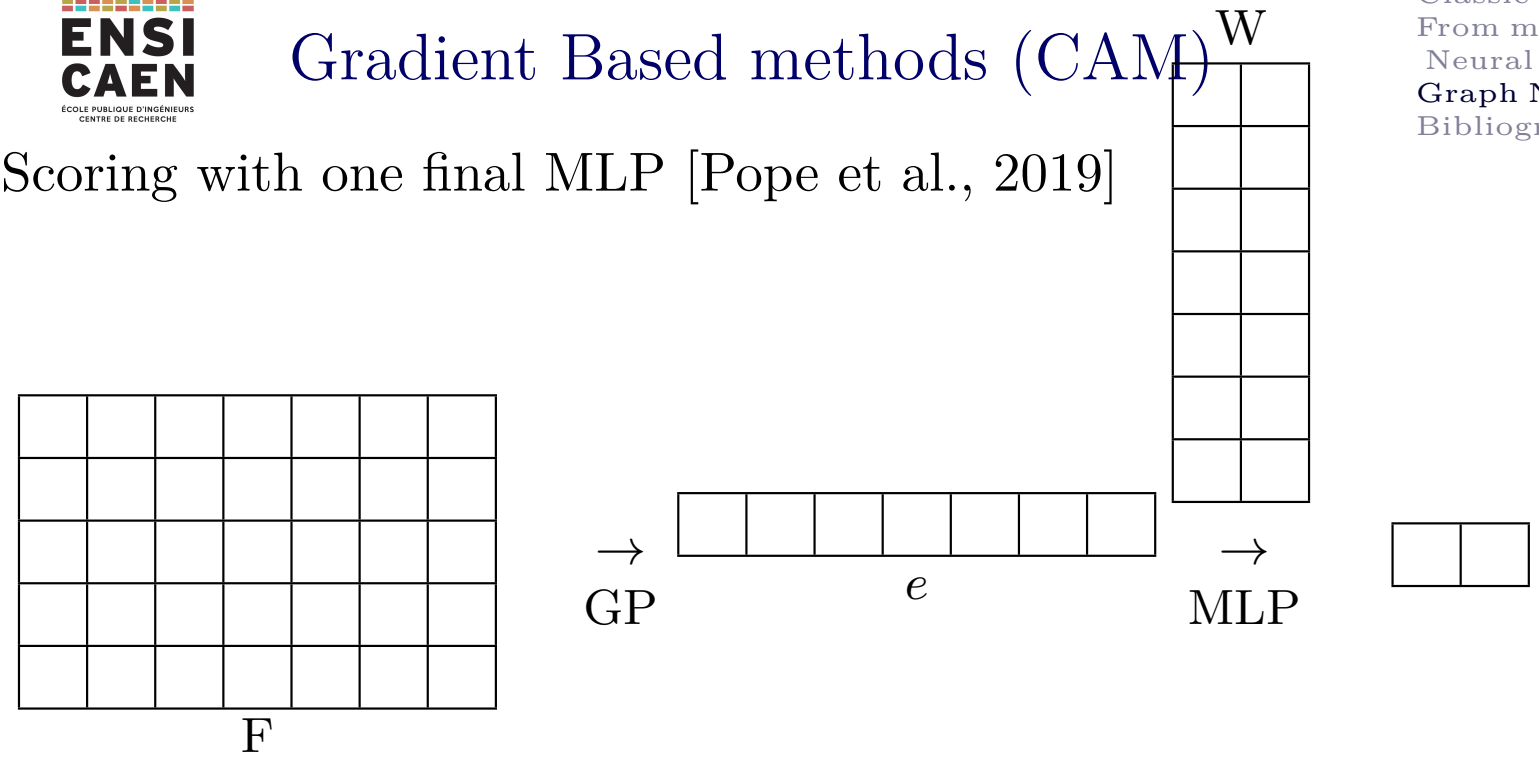


Brief visit of the zoo



Gradient Based methods (CAM)

Scoring with one final MLP [Pope et al., 2019]



$$\sum_k W_{k,c} e_k = \sum_k W_{k,c} \sum_{n=1}^{N} F_{k,n}^L = \sum_{n=1}^{N} \sum_k W_{k,c} F_{k,n}^L = \sum_{n=1}^{N} < W_c, F_n^L >$$

Hence $< W_c, F_n^L >$ encodes the relevance of vertex n for the class c. We can thus design a score defined as:

$$score^L(n) =< F_n^L, W^c > \text{ or } score^L(n) =< F_n^L, W^c - W^{c'} >$$

Which encodes the relevance of n for c (compare to c').

Gradient Based methods (CAM)

Basics of Machine learning
Classic methods
From molecules to prediction
Neural Networks
Graph Neural Networks
Bibliography

The feature of each vertex is computed from its neighborhood:

$$< F_n^L, W^c - W^{c'} >= \sum_{v \in \mathcal{N}(n)} c_{n,v}^L < F_v^{L-1}, W^c - W^{c'} >$$

The relevance of a vertex v may be defined as the weighted sum of the vertices to which it contributes:

$$score^{L-1}(v) = \sum_n \frac{c_{n,v}^L}{Z} score^L(n) \quad with \quad Z = \sum_n c_{n,v}$$

We may(should) also discard all vertices for which $< F_v^{L-1}, W^c - W^{c'} >< \epsilon$. This idea is fearther developped using Excitation-BackPropagation (see bellow).

Gradient Based methods (Grad-CAM)

Basics of Machine learning
Classic methods
From molecules to prediction
Neural Networks
Graph Neural Networks
Bibliography

The above method does not work if we have several MLP or more complex processing of the final vector. A generalization of CAM consists to consider:

$$\alpha_k^{l,c} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial y^c}{\partial F_{k,n}^k}$$

with $\alpha_k^{L,c} = w_k^c$ if we have a single MLP. Then:

$$L_{Grad-CAM}^c(l,n) = Relu(\sum_k \alpha_k^{l,c} F_{k,n}^l(X,A)).$$

We may also average (or apply any pooling function) across the layers to get a global relevance of a vertex:

$$L_{Grad-CAMAvg}^c(n) = \frac{1}{L} \sum_l L_{Grad-CAM}^c(l,n)$$

Excitation-BackPropagation (Principle)

Still based on CNN on Images.
Let us consider the contributions of neurons at layer $l-1$ (a_j^{l-1}) to neurons i at level l (a_i^l) as a probability function. The contribution of each a_j^{l-1} may be computed as:

$$P(a_j^{l-1}) = \sum_i P(a_j^{l-1}|a_i^l)P(a_i^l)$$

Given $P(a_j^{l-1}|a_i^l)$ the idea of excitation-Backpropagation methods is to set $P(y^c) = 1$ for the class c of interest at the final layer and to compute recursively the relevance of each neuron up to the initial layer.

Excitation-BackPropagation (GNN)

We consider a GCN composed of several convolutions, a GAP and one MLP/softmax. We have:

$$\begin{array}{lcll} p(e_k) & = & \sum_c \frac{1}{Z} e_k Relu(w_k^c) p(c) \quad with Z = \sum_k e_k Relu(w_k^c) & MLP/softmax \\ p(F_{k,n}^L) & = & \frac{F_{k,n}^L}{N e_k} p(e_k) & GAP \end{array}$$

$p(c) = 1$ for the class of interest 0 otherwise.
Let us decompose the convolution in 2 steps:

$$\left\{ \begin{array}{lcll} \hat{F}_{k,n}^l & = & \sum_m A_{n,m} F_{k,m}^l & averaging \\ F_{k',n}^{l+1} & = & \sigma(\sum_{k'} \hat{F}_{k,n}^l W_{k,k'}^l) & MLP \end{array} \right.$$

Where A is the convolution operator. We have:

$$\left\{ \begin{array}{lcll} p(F_{k,n}^l) & = & \sum_m \frac{1}{Z_n} A_{n,m} F_{k,n}^k p(\hat{F}_{k,m}^l) & with \quad Z_n = \sum_n A_{n,m} F_{k,m}^l \\ p(\hat{F}_{k,n}^l) & = & \sum_{k'} \frac{1}{Z_k} \hat{F}_{k,n}^l Relu(W_{k,k'}^l) p(F_{k',n}^{l+1}) & with \quad Z_k = \sum_k \hat{F}_{k,n}^l Relu(W_{k,k'}^l \end{array} \right.$$

The final relevance of each vertex is defined as the average relevance of its dimensions:

$$L_{EB}^c(n) = \frac{1}{d_{in}} \sum_{k=1}^{d_{in}} p(F_{k,n}^0).$$

Evaluation metrics

We got an explanation for a decision of the GNN. Is it accurate, necessary, sufficient, sparse ?

$\uparrow$ $Fidelity^+$: Do we have all necessary elements in the explanation ?

$$Fidelity^+ = f(G)_c - f(G^{1-m_c})_c$$

where all the elements of G belonging to the explanation have been removed from G^{1-m_c}.

$\downarrow$ $Fidelity^-$: Does the selected elements are sufficient ?

$$Fidelity^- = f(G)_c - f(G^{m_c})_c$$

where G^{m_c} represent the explanation of G for the class c.

Large explanations (G^{m_c}) induce low $Fidelity^-$ but also small (G^{1-m_c}) and hence large $Fidelity^+$.

Evaluation metric

↑ Characterization score: We want to have an explanation which is both necessary and sufficient→ A basic harmonic mean between $Fidelity^+$ and $1 - fidelity^-$.

$$Caracterization_{score} = \frac{2Fidelity^+(1 - Fidelity^-)}{Fidelity^+ + 1 - Fidelity^-}$$

The explanation corresponding to the complete graph has a characterization score of 1.

↑ Sparsity:

$$sparsity = 1 - \frac{|G^{m_c}|}{|G|}$$

↑ Explanation Accuracy:

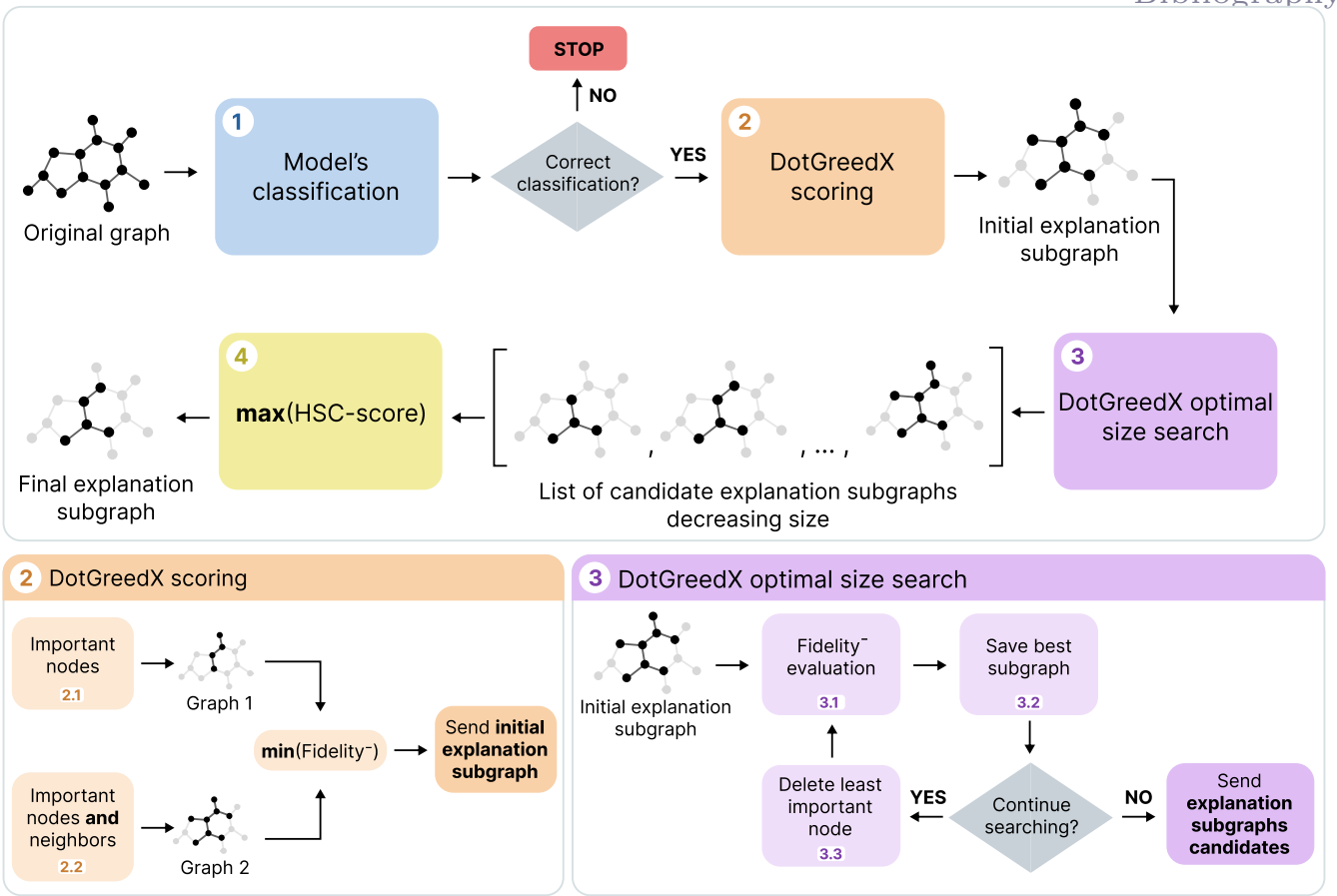
$$Exp_{acc} = \frac{1}{|T|} \sum_{G \in T} |\hat{y}(G) == \hat{y}(G^{m_c})|$$

A heat map is not an explanation

All previous methods associate a weight to each node/edge of a graph based on its relevance according to a class c. It does not identifies the subgraphs responsible for a given property.

- Most method applies a hard threshold on the weights $\rightarrow$ inefficient.
- EigSearch [Lu et al., 2024] sorts the edges according to their relevance and selects the subset which maximises $Fidelity^{+} - Fidelity^{-}$.

DotGreedX



DotGreedX select all nodes with a positive weigh and their neighbors. Then, it iteratively removes the nodes whose removal induce the lower $fidelity^{-}$.

$$HSC_{score} = \frac{2 \cdot Characterization_{score} \cdot Sparsity}{Characterization_{score} + Sparsity}$$

Sommaire de la section

1. Basics of Machine learning
   - Definition
2. Classic methods
   - How to measure ML performances
   - knn
   - SVM
   - Kernel Methods
3. From molecules to prediction
   - Fingerprints
   - Graph kernels
   - Kernels based on infinite Bags
   - Kernels based on finite Bags
   - Application to chemoinformatics
4. Neural Networks
   - Principles
5. Graph Neural Networks
   - Graph Aggregation
   - Recurrent Graph Neural Networks
   - Convolutional Graph neural networks
     - Spectral approaches

Bibliography

Arge, L., Berg, M. D., Haverkort, H., and Yi, K. (2008).
The priority r-tree: A practically efficient and worst-case optimal r-tree.
ACM Trans. Algorithms, 4(1).

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.

Blumenthal, D. B., Boria, N., Gamper, J., Bougleux, S., and Brun, L. (2020).
Comparing heuristics for graph edit distance computation.
The VLDB Journal, 29:419–458.

Carhart, R. E., Smith, D. H., and Venkataraghavan, R. (1985).
Atom pairs as molecular features in structure-activity studies: definition and applications.
J. Chem. Inf. Comput. Sci., 25(2):64–73.

Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. L. (2001).
Searching in metric spaces.
ACM Comput. Surv., 33(3):273–321.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016).
Convolutional neural networks on graphs with fast localized spectral filtering.
In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 3844–3852. Curran Associates, Inc.

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.

Fix, E. and Hodges, J. (1951).
Discriminatory analysis. nonparametric discrimination: Consistency properties.
Technical report, USAF School of Aviation Medicine, Randolph Field, Texas.

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.

Haasdonk, B. (2005).
Feature space interpretation of svms with indefinite kernels.
IEEE Trans. Pattern Anal. Mach. Intell., 27(4):482–492.

Haussler, D. (1999).
Convolution kernels on discrete structures ucsc crl.
Technical Report UCSC-CRL-99-10, Department of Computer Science, University of California at Santa Cruz, Santa Cruz, CA 95064.

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.

Kalantari, I. and McDonald, G. (1983).

A data structure and an algorithm for the nearest point problem.
IEEE Transactions on Software Engineering, SE-9(5):631–634.


Kashima, H., Tsuda, K., and Inokuchi, A. (2003).
Marginalized kernels between labeled graphs.
In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 321–328.


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. (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.


Levie, R., Monti, F., Bresson, X., and Bronstein, M. M. (2019).
Cayleynets: Graph convolutional neural networks with complex rational spectral filters.

IEEE Transactions on Signal Processing, 67(1):97–109.

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*.

Lu, S., Liu, B., Mills, K. G., He, J., and Niu, D. (2024).
Eig-search: Generating edge-induced subgraphs for gnn explanation in linear time.
In *ICML*. OpenReview.net.

Mahé, P. and Vert, J.-P. (2009).
Graph kernels based on tree patterns for molecules.
Machine Learning, 75(1):3–35.

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.

Nilakantan, R., Bauman, N., Dixon, J. S., and Venkataraghavan, R. (1987).
Topological torsion: a new molecular descriptor for SAR applications. comparison with other descriptors.
J. Chem. Inf. Comput. Sci., 27(2):82–85.

Pope, P. E., Kolouri, S., Rostami, M., Martin, C. E., and Hoffmann, H. (2019).
Explainability methods for graph convolutional neural networks.
In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10764–10773.

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.

Sproull, R. F. (1991).
Refinements to nearest-neighbor searching in k-dimensional trees.
Algorithmica, 6(4):579–589.

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.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. (2010).
Graph kernels.
Journal of Machine Learning Research, 11:1201–1242.

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.

Zhang, J., Shi, X., Xie, J., Ma, H., King, I., and Yeung, D. (2018a).

Gaan: Gated attention networks for learning on large and spatiotemporal graphs.

In Globerson, A. and Silva, R., editors, *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 339–349. AUAI Press.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018b).

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.