# Computational Intelligence

Année universitaire 2022/2023

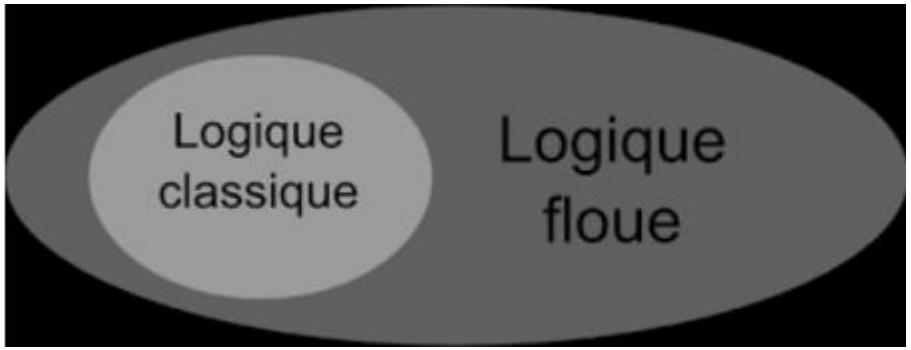
Pr. Jaouad Boumhidi

# Plan du cours

- ***Chap 1:*** Systèmes flous ou logique floue (l'intelligence assuré à base de règles)
- ***Chap 2:*** Réseau de neurones (l'intelligence assurée à base d'apprentissage)
- ***Chap 3:*** *Calcul* évolutionnaire (Algorithmes évolutionnaires):
  - ***Algorithmes génétiques***
  - ***Essaimes de particules***

**Computational intelligence** is a set of nature-inspired computational methodologies and approaches to address complex real-world problems to which mathematical or traditional modelling can be useless

# Chap 1: Logique floue




- La logique floue propose des modes de raisonnement approximatifs plutôt qu'exacts
- C'est principalement le mode de raisonnement utilisé dans la plupart des cas par **les humains**.

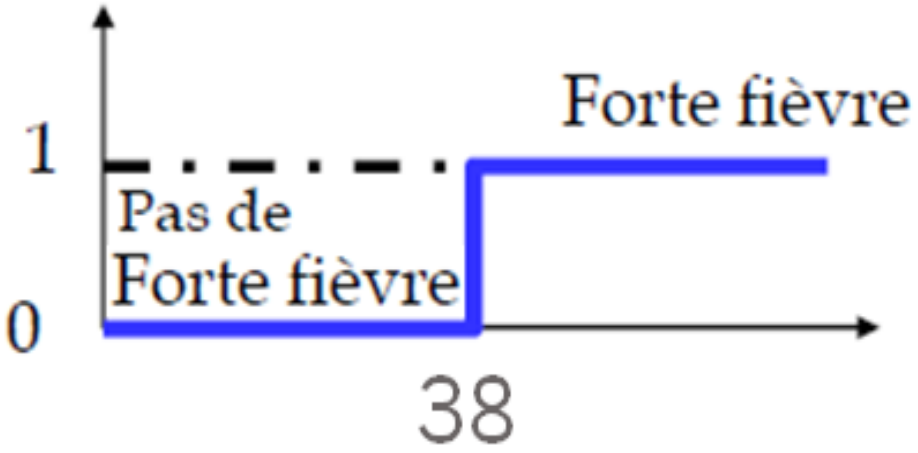
- En 1965, Lotfi Zadeh, introduit la notion d'ensembles flous (Fuzzy sets)
- La logique floue a été introduite en 1973,
- Elle est longtemps restée marginale.

**L'idée de Zadeh consiste à utiliser le modèle de l'esprit humain**

# Approche Classique

Logique Propositionnelle

Les propositions ont des valeurs dans l'intervalle {Vrai, Faux} ou {0,1}



# Ensemble Classique

- La relation d'appartenance à un ensemble classique A est représentée par une fonction μ qui prend des valeurs de vérité dans la paire $\{0,1\}$. Ainsi,

**la fonction d'appartenance** d'un ensemble classique A est définie par:

$$\mu_A(x) = \begin{cases} 1 & si\ x \in A \\ 0 & si\ x \notin A \end{cases}$$

# Inconvinients

- Les variables décrivant les états sont booléennes,
- Ne peuvent prendre que deux valeurs
- Sont donc mal adaptées à représenter la plupart des phénomènes courants

En effet:

Si la température est de 37,98  ou si la température est de 38,02  ??

- L'incertain et l'imprécis

Je crois que la température est élevée (***donc pas sure***)

# Logique floue: Théorie des ensembles flous

Selon Zadeh, **la logique floue est la théorie des ensembles flous** (Zadeh, 1994).

La théorie des ensembles flous est une théorie mathématique dont l'objectif principal est **la manipulation des notions incertaines** du langage naturel.

Ainsi, elle évite les inadéquations de la théorie des ensembles classiques

# Exemple:

l'ensemble *A représentant les* ***PC qui sont trop chers*** pour une population d'étudiants. Après une enquête menée au sein de cette population, un PC ayant un prix supérieur ou égal à 8000 DH sera déclaré ***trop cher***, *quand un prix inférieur ou égal à 5000* ***n'est pas trop cher***.

- Il existe un nombre important de PCs ayant un prix entre ces deux limites. Dans cet intervalle, on peut utiliser des valeurs, comprises strictement entre 0 et 1, pour classifier ces prix comme étant ***partiellement trop cher***.

- Cette classification permettra de définir:

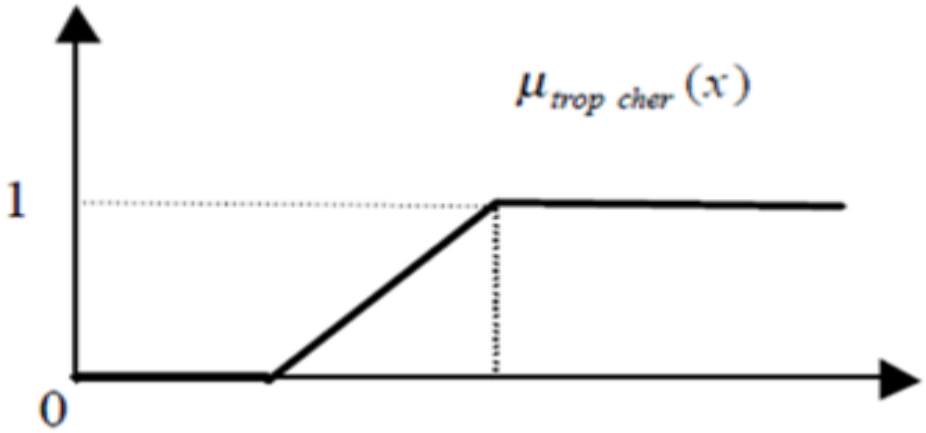
**une nouvelle fonction d'appartenance, $\mu A(x)$,**

*associée à l'ensemble A représentant les PC trop chers.*

- $\mu A(x)$ *est strictement* ***entre 0 et 1*** *implique que x appartient à A* avec un degré de vérité égal à $\mu A(x)$.

***A est donc le sous ensemble flou*** *associé à la* ***valeur* linguistique *trop cher***

*Chaque ensemble flou est caractérisé par sa fonction d'appartenance flou*



# Concepts fondamentaux

Si par exemple , $\mu A(x)=0.1$, *x appartient à l'ensemble flou A avec un degré d'appartenance de 10%*

⇔ Faible appartenance ⇔ Traduction de la valeur linguistique « Faible »

Si par exemple , $\mu A(x)=0.9$, *x appartient à l'ensemble flou A avec un degré d'appartenance de 90%*

⇔ Forte appartenance ⇔ Traduction de la valeur linguistique « Fort»

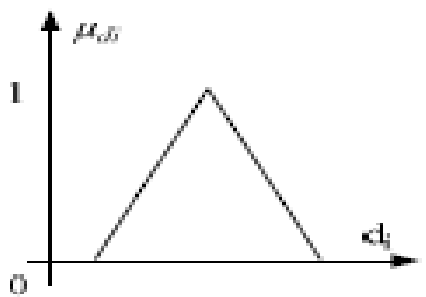
**Un degré d'appartenance constitue une mesure d'incertitude**

***Un ensemble flou est totalement déterminé par sa fonction d'appartenance***

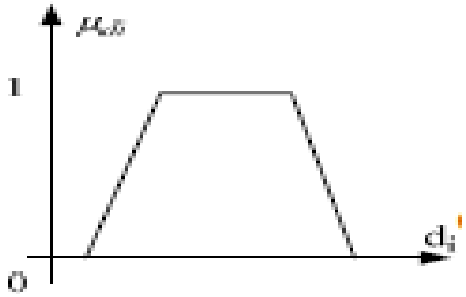
# Caractéristiques de la fonction d'appartenance

La fonction d'appartenance décrivant un ensemble flou est caractérisée par 4 propriétés:

**Type**: forme qui peut être triangulaire, trapézoïdale, gaussienne,…



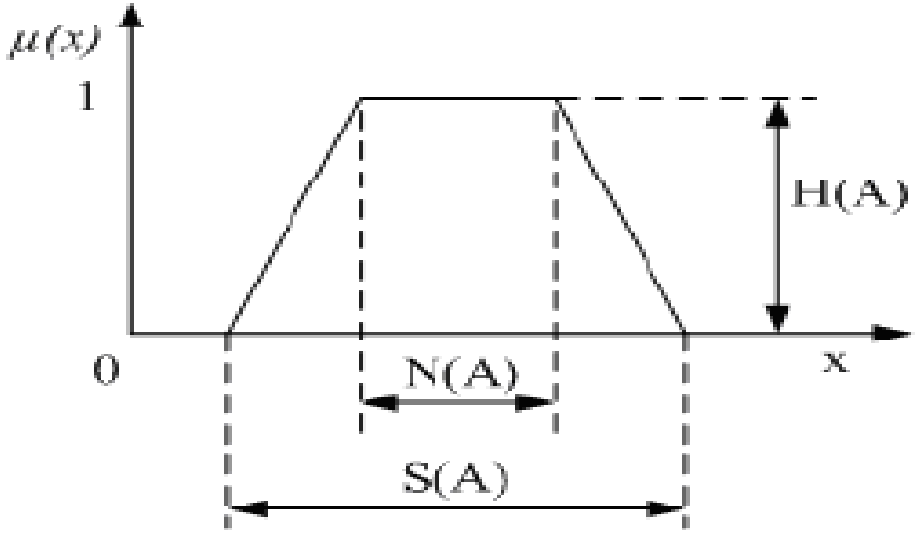

ou



La hauteur : $H(A) = Sup_{x \in X}(\mu_A(x))$ de la fonction d'appartenance. Un sous-ensemble flou est dit normalisé s'il est de hauteur 1.

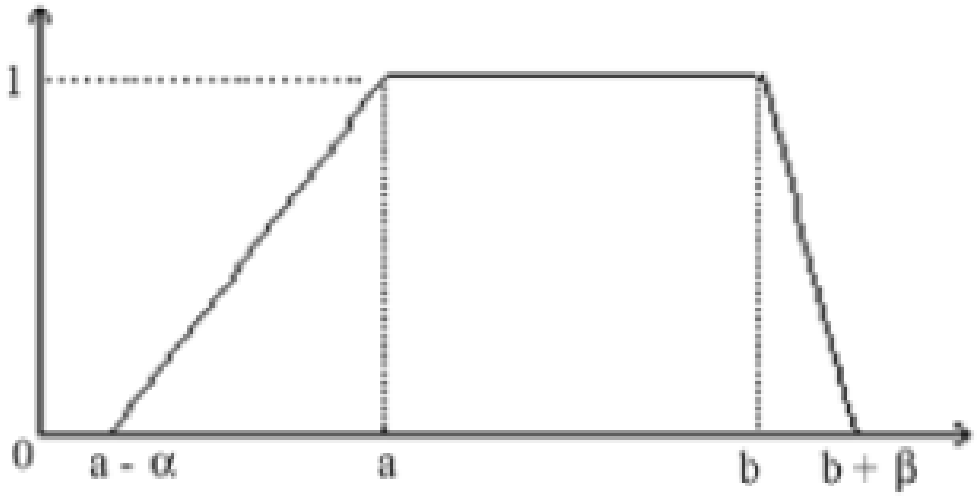
Le noyau : $N(A) = \{x / \mu_A(x) = 1\}$ est l'ensemble des éléments qui appartiennent totalement à A. Pour les fonctions de type triangulaire, le noyau est un singleton qui est appelé aussi valeur modale.

Le support : $S(A) = \{x / \mu A(x) \neq 0\}$ ; cet ensemble décrit l'ensemble des éléments qui sont partiellement dans A.



Un nombre flou trapézoïdale est notée généralement par: $(a, b, \alpha, \beta)$ :

$$\mu_A(x) = \begin{cases} 0 \text{ si } x < a - \alpha \text{ ou } b + \beta < x, \quad (x \text{ hors du support de } A) \\ 1 \text{ si } a < x < b, \quad (x \text{ dans le noyau de } A) \\ 1 + (x - a) / \alpha \text{ si } a - \alpha < x < a, \\ 1 - (b - x) / \beta \text{ si } b < x < b + \beta \end{cases}$$



# Triangulaire (Très utilisée)

- Un nombre flou triangulaire est un cas particulier de trapézoïdale est notée par: $(a, \alpha, \beta)$

- Déterminer l'expression de la fonction d'appartenance dans ce cas?

# VARIABLES LINGUISTIQUES ET VALEURS LINGUISTIQUES

- Une variable linguistique est une variable dont les valeurs associées sont **linguistiques plutôt que numériques** (Zadeh, 1997).

Par exemple, la **variable linguistique *âge*** *peut être évaluée* par les **trois valeurs linguistiques** suivantes:

*jeune, moyen et vieux*

- La notion de variable linguistique suppose donc l'existence d'un **univers de discours *X*** *et d'un ensemble de valeurs* linguistiques D associé à la variable linguistique étudiée

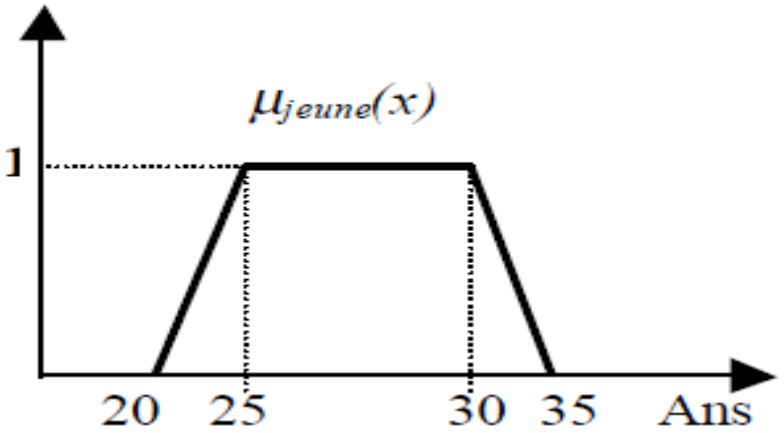
- Pour la variable linguistique *âge*, *X* est l'ensemble des nombres réels positifs IR+

Exemple: **[0 120]**

et D = {*jeune, moyen, vieux*}.

- En logique floue, **les valeurs linguistiques sont représentées par des sous ensembles flous**

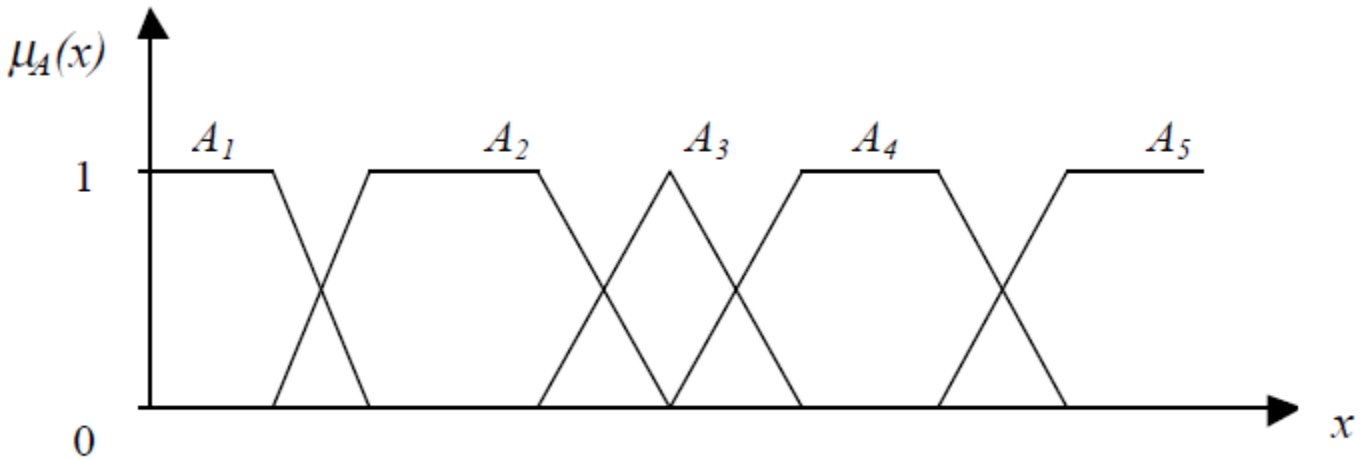
- Considérons, par exemple, le cas de la valeur linguistique ***jeune***; *elle* peut être représentée de la façon suivante:



# Partition floue

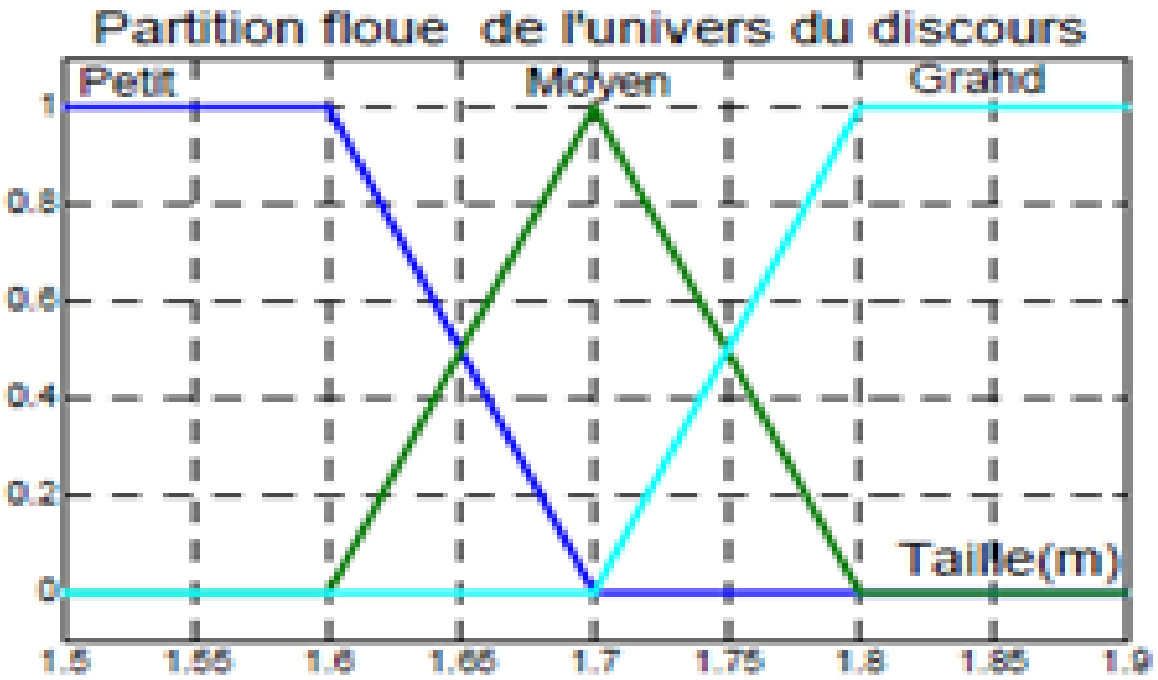
- Partition floue: Soient N ensembles flous *Aj du référentiel X*. *(A1, A2,.., Aj,., AN) est* dite une *partition floue si:*

$$\forall x \in X \quad \sum_{j=1}^{N} \mu_{A_j}(x) = 1 \quad avec \quad A_j \neq \phi \quad et \quad A_j \neq X \quad \forall \quad 1 \leq j \leq N$$




Exemple d'une partition floue formée de cinq ensembles flous.

Exemple, la variable linguistique *taille peut être évaluée* par les trois valeurs linguistiques suivantes: petit, moyen et grand (3 ensembles flous)



Compléter la partition floue, de l'âge  sur un univers de discours de votre choix et avec 5 valeurs linguistiques de votre choix

# Les opérateurs flous

Extention des Opérateurs de la théorie des ensembles classiques :$=, \cup, \cap, \subset,$ et Complément.

Soit A et B deux sous ensembles flous de X définis par les fonctions d'apprtenance: $\mu_A$ et $\mu_B$ :

Égalité

$$A = B \text{ ssi } \forall x \in X, \mu_A(x) = \mu_B(x)$$

*Inclusion*

$$A \subset B \text{ ssi } \forall x \in X, \mu_A(x) < \mu_B(x)$$

*Intersection*

$$\forall x \in X, \mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$$

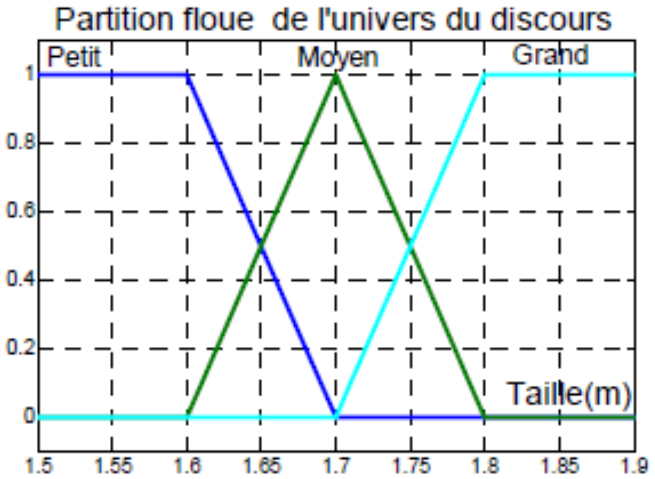
*Union*

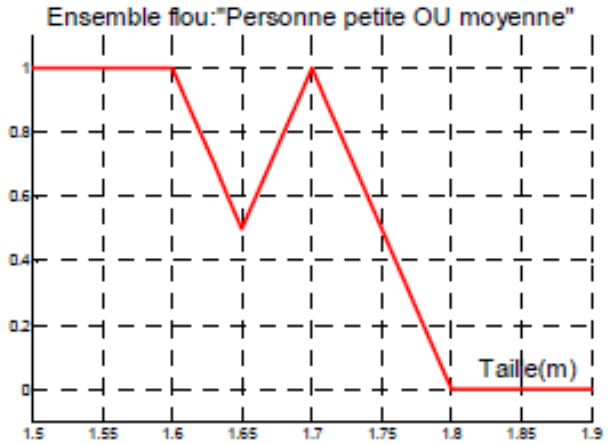
$$\forall x \in X, \mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$$

# Opérateur flou Union

L'ensemble des personnes petites OU moyennes est un ensemble flou de fonction d'appartenance :

$$\mu_{A\cup B}(x) = max\left(\mu_A(x), \mu_B(x)\right) \quad \forall x \in U$$

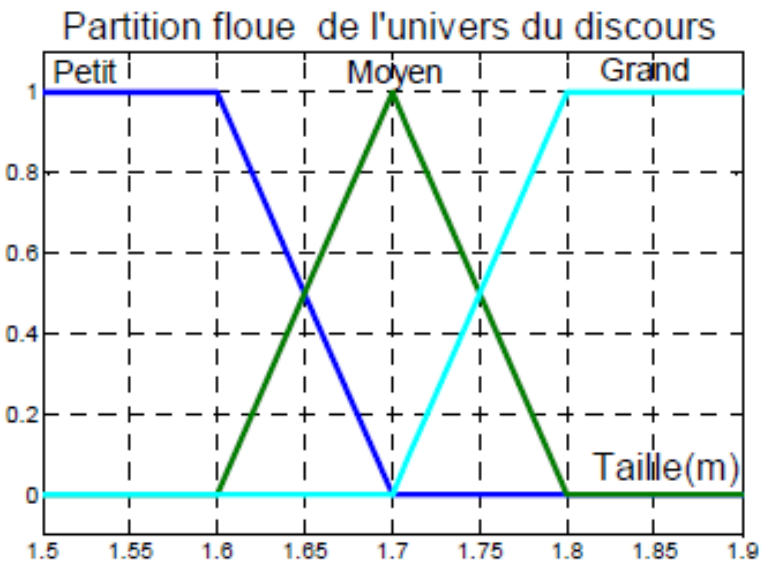


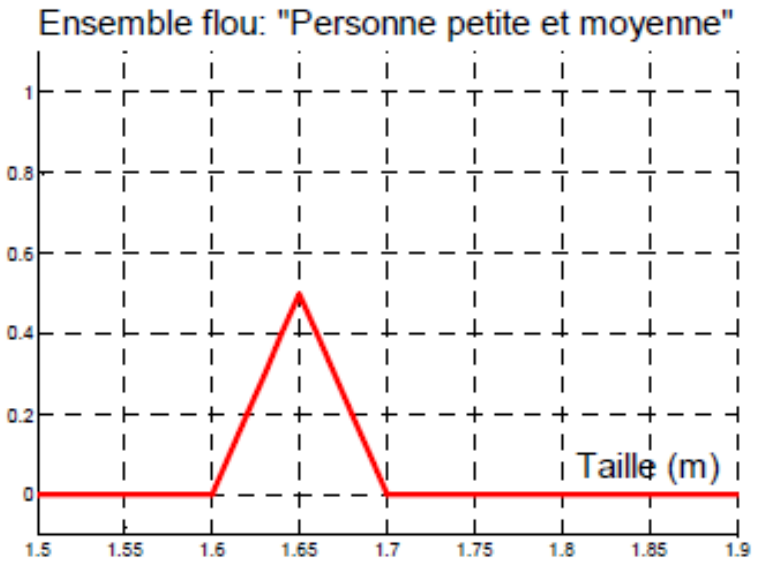



# Opérateur flou Intersection

L'ensemble des personnes petites ET moyennes est un ensemble flou de fonction d'appartenance :

$$\mu_{A\cap B}(x) = min\left(\mu_A(x), \mu_B(x)\right) \quad \forall x \in U$$

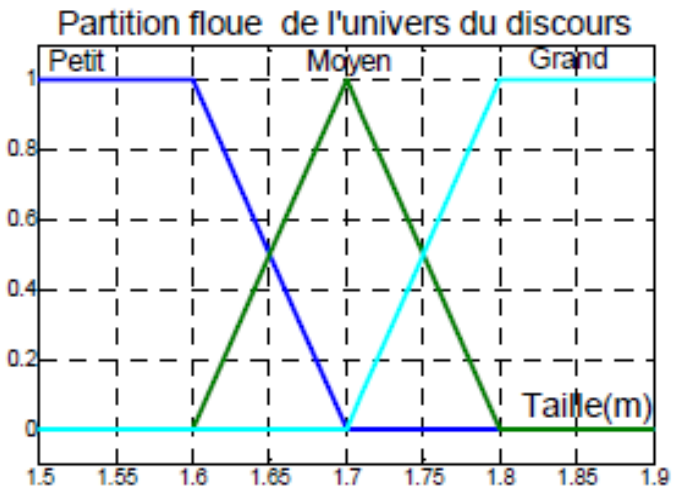


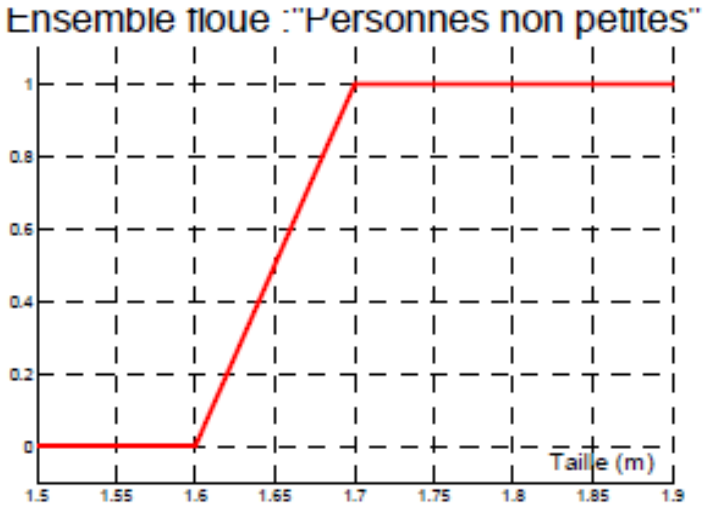



# Complément

L'ensemble des personnes NON petites est un ensemble flou de fonction d'appartenance :

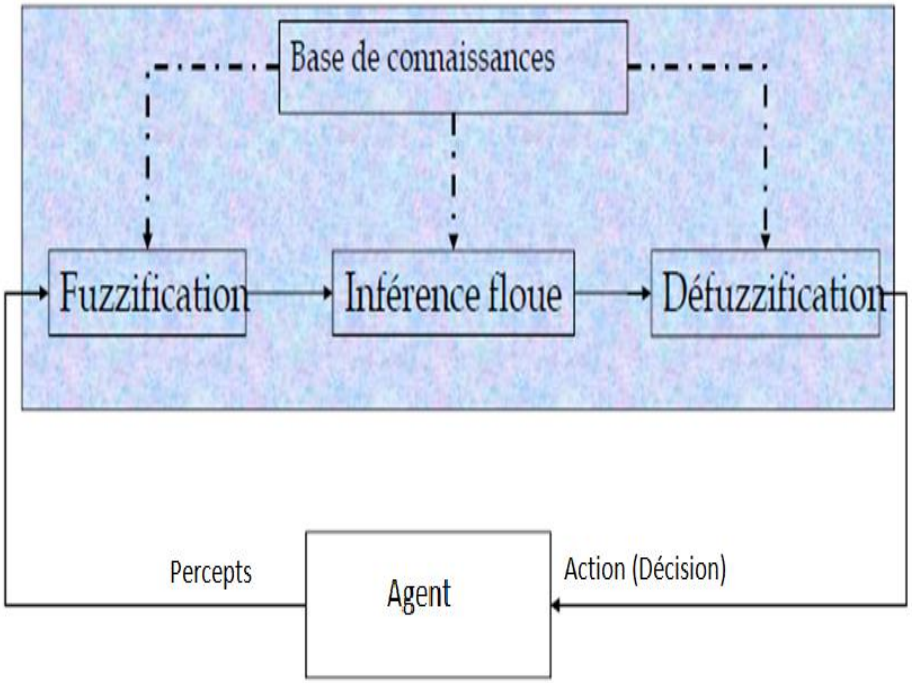
$$\mu_{\overline{A}}(x) = 1 - \mu_A(x) \quad \forall x \in U$$

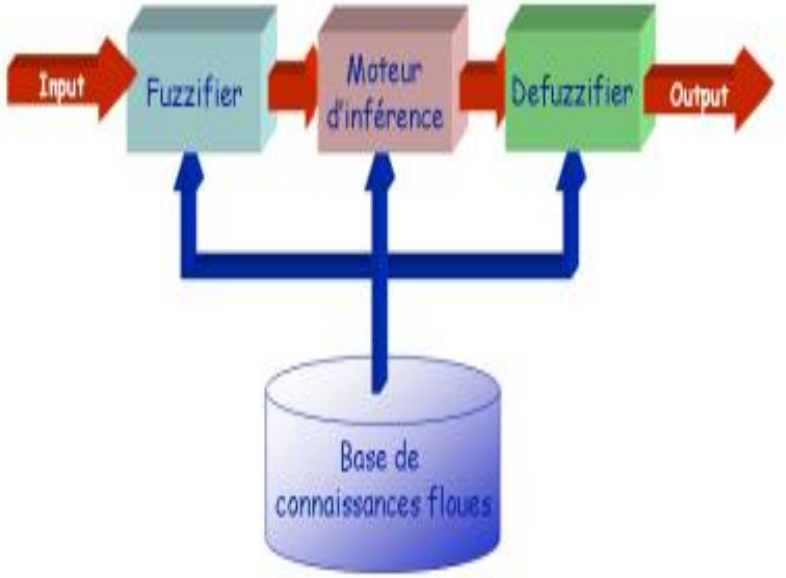


# Prise de décision par logique floue
## Fuzzy decision making

Il ya 4 étapes nécessaires lors de conception d'une **décision flou**:



# Fuzzification

C'est le processus qui consiste à transformer une ***grandeur numérique*** en un ***ensemble flou***.

Revient à qualifier une ***variable numérique*** par une ***variable linguistique***

# Comment fuzzifier?

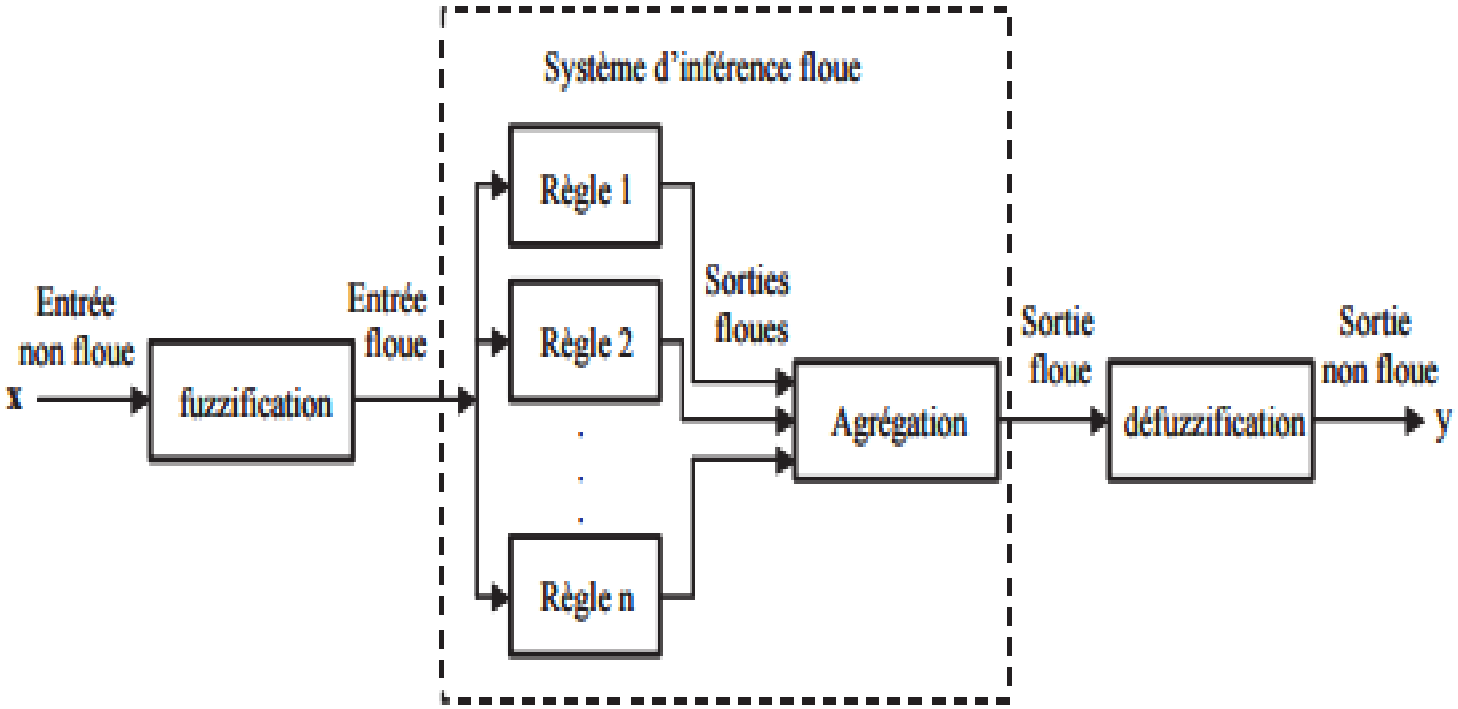
- Donner **l'univers de discours**: plage de variations possibles de l'entrée considérée(des percepts).
- **Une partition** en classe floue de cet univers

**Exemple:**

Une entrée *taille* (petit; moyenne;grand)

Une entrée *taille* (troppetit;petit; moyenne;grand; trog grand)

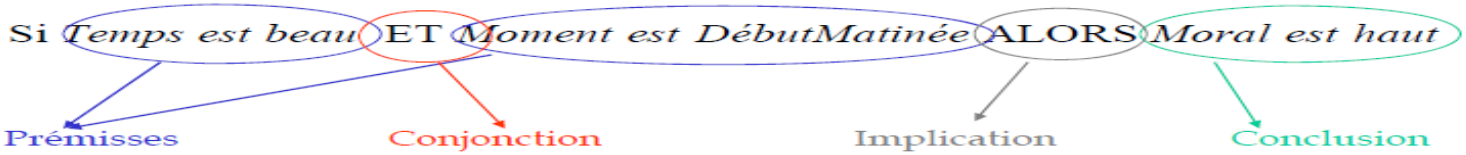
# Système d'Inférence Floue(SIF) ou (FIS en Anglais)



# Règle floue et proposition floue

Si x est A ET y est B Alors z est C

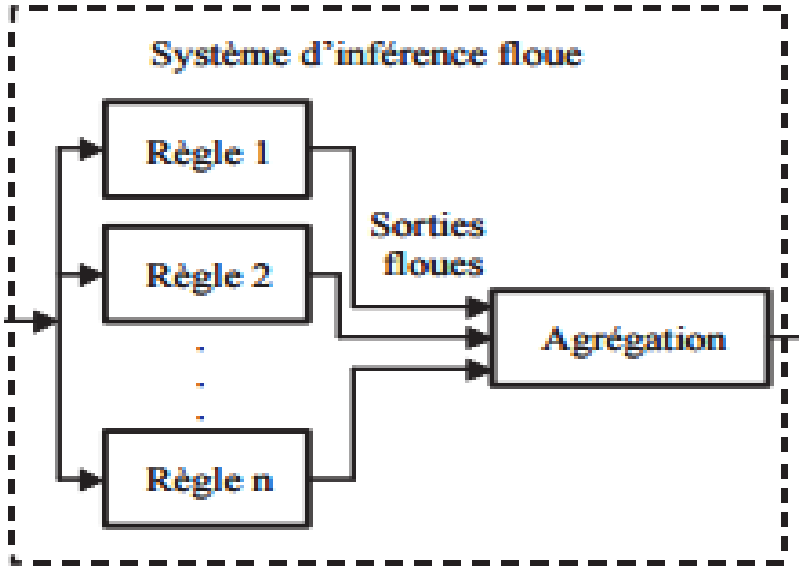
***Exemple 1***




***Exemple 2***

SI ***distance*** entre vehicles est ***petite*** et ***vitèsse*** est ***fort*** alors ***freiner*** ***tres*** ***fort***;

# Étapes d'établissement de la sortie floue: SIF



# Étape1 du SIF:
# Calcul du degré d'activation de chaque règle

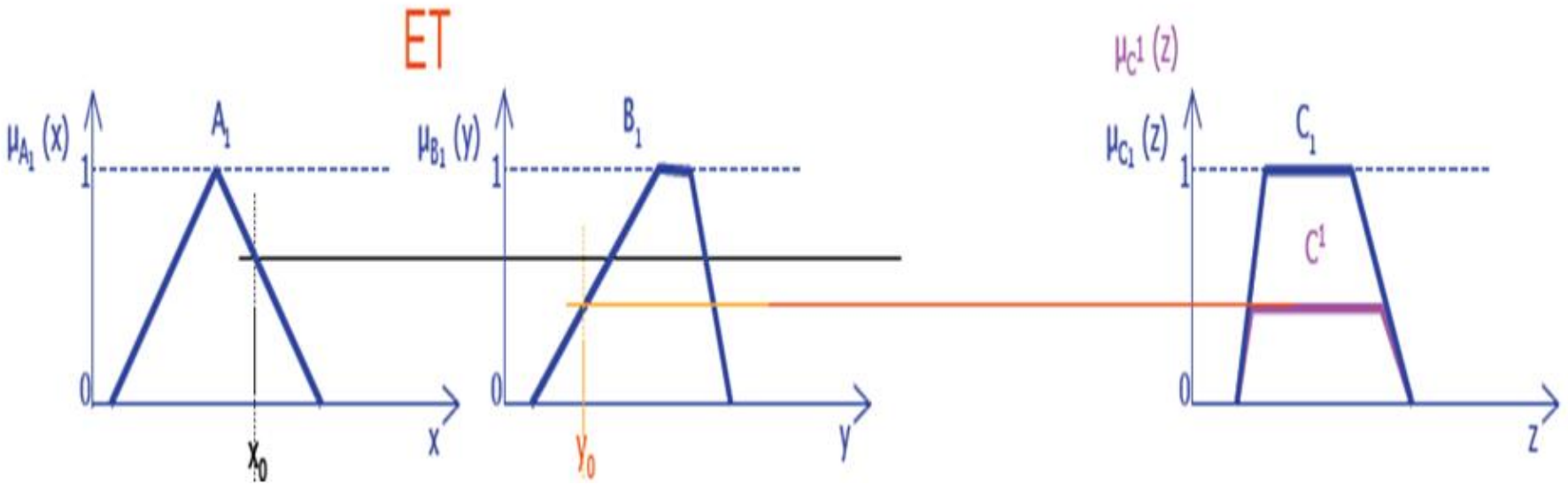
- L'activation des règles consiste à calculer le degré d'activation de chacune. C'est une valeur comprise entre 0 et 1
- **min(u,v)** permet de représenter la conjonction **ET**

Règle1 : ***Si*** *X1 est A1* ***ET*** *X2 est A2 alors y est B*

- ***Degré des deux permises (Si X1 est A1 ET X2 est A2) est le min des degrés d'appartenances des deux)***
- ***Dans ce cas:***

$\mu B(y)$ = poids de la règle 1 = $\min(\mu A1\ (x1), \mu A2\ (x2))$

$\Re_1$ : si x est $A_1$ et y est $B_1$ alors z est $C_1$



***Exemple d'illustration du cas d'une inférence avec une seule règle***,

Nous supposons que nous voulons connaître la force de freinage d'une voiture qui roule dans une ville si le feu est rouge.

Pour cela nous avons deux caractéristiques associées : **Vitesse** et **Distance**. Ces caractéristiques sont alors nos variables d'entrée et la variable de sortie est **freinage**. Si nous considérons une voiture qui roule à une vitesse de *65 Km/h* et le feu se trouve à *20 mètres*,

Soit la règle suivante :

R1 : Si Distance est Courte et Vitesse est Maximale Alors Freinage est Fortement

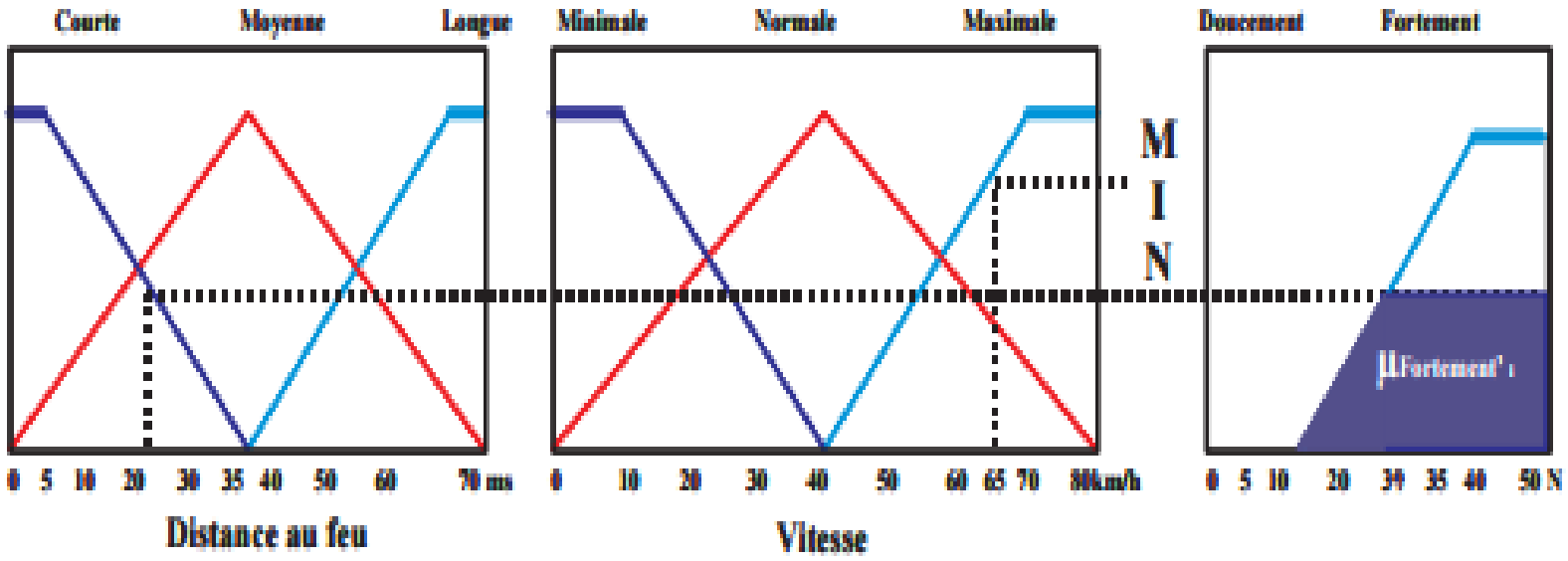
Le degré d'accomplissement ou degré d'appartenance ($\mu$) pour la variable Distance ( 20) et Vitesse ($x_2 = 65$) est comme suit :

$$x_1 = 20 \rightarrow \mu_{Courte}(20) = 0.42 \ , \ \mu_{Moyenne}(20) = 0.57 \ , \ \mu_{Longue}(20) = 0.0$$

$$x_2 = 65 \rightarrow \mu_{Minimale}(65) = 0.0, \ \mu_{Normale}(65) = 0.375, \ \mu_{Maximale}(65) = 0.833$$

$$R_1 : Freiner_{\mu Fortement_1} = [min(\mu_{Courte}(20), \mu_{Maximale}(65))] = min(0.42, 0.833) = 0.42$$

**Règle 1 : Si distance est Courte et Vitesse est Maximale Alors freiner Fortement**



# Activation des règles(calcul des degrés de toutes les règles

R1: $Si\ (X_1\ est\ A_{11})\ et\ (X_2\ est\ A_{12})$ $alors\ Y\ est\ B_1$

R2: $Si\ (X_1\ est\ A_{21})\ ou\ (X_2\ est\ A_{22})$ $alors\ Y\ est\ B_2$

R3: $Si\ (X_1\ est\ A_{31})\ et\ (X_2\ est\ A_{32})\ et\ (X_3\ est\ A_{33})$ $alors\ Y\ est\ B_3$

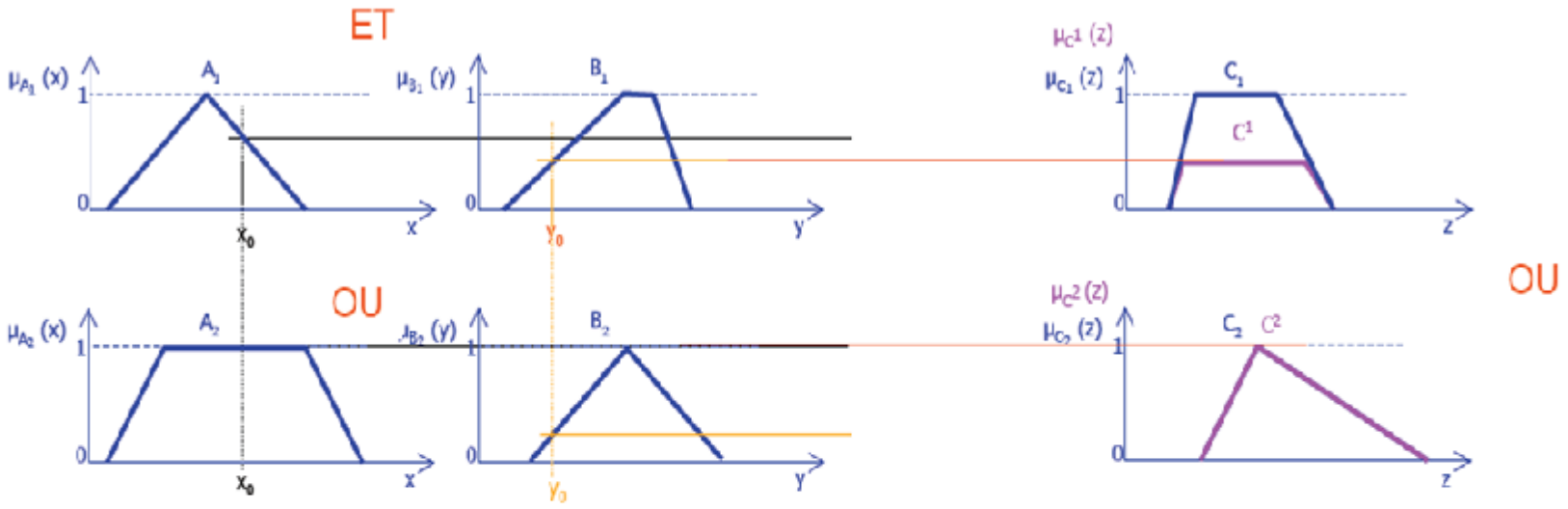
.....

# Agrégation

Les règles sont liées par un opérateur OU
c.à.d si R1 ou R2 ou ….ou Ri ou ….Rq

$$\mu_B(y) = MAX\left[\mu_{B_i}(y)\right] \qquad i \in \{indices\ des\ règles\ activées\}$$

y: sortie floue (décision floue)

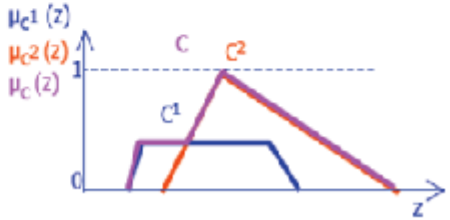



$\Re_1$ : si x est $A_1$ et y est $B_1$ alors z est $C_1$

$\Re_2$ : si x est $A_2$ ou y est $B_2$ alors z est $C_2$

Fait : x est $\overline{x}_0$ et y est $\overline{y}_0$

Conséquence : z est C



# *Exemple d'illustration du cas d'une inférence avec deux règles,*

- Nous continuons notre exemple de la voiture avec les mêmes valeurs, c'est-à-dire *20 mètres pour la distance au feu* et *65 km/h pour la vitesse de la voiture .*
- Nous considérons les deux règles suivantes :
- *R1 : Si Distance est Courte et Vitesse est Maximale Alors Freinage est Fortement*
- *R2 : Si Distance est Moyenne et Vitesse est Normale Alors Freinage est Doucement*

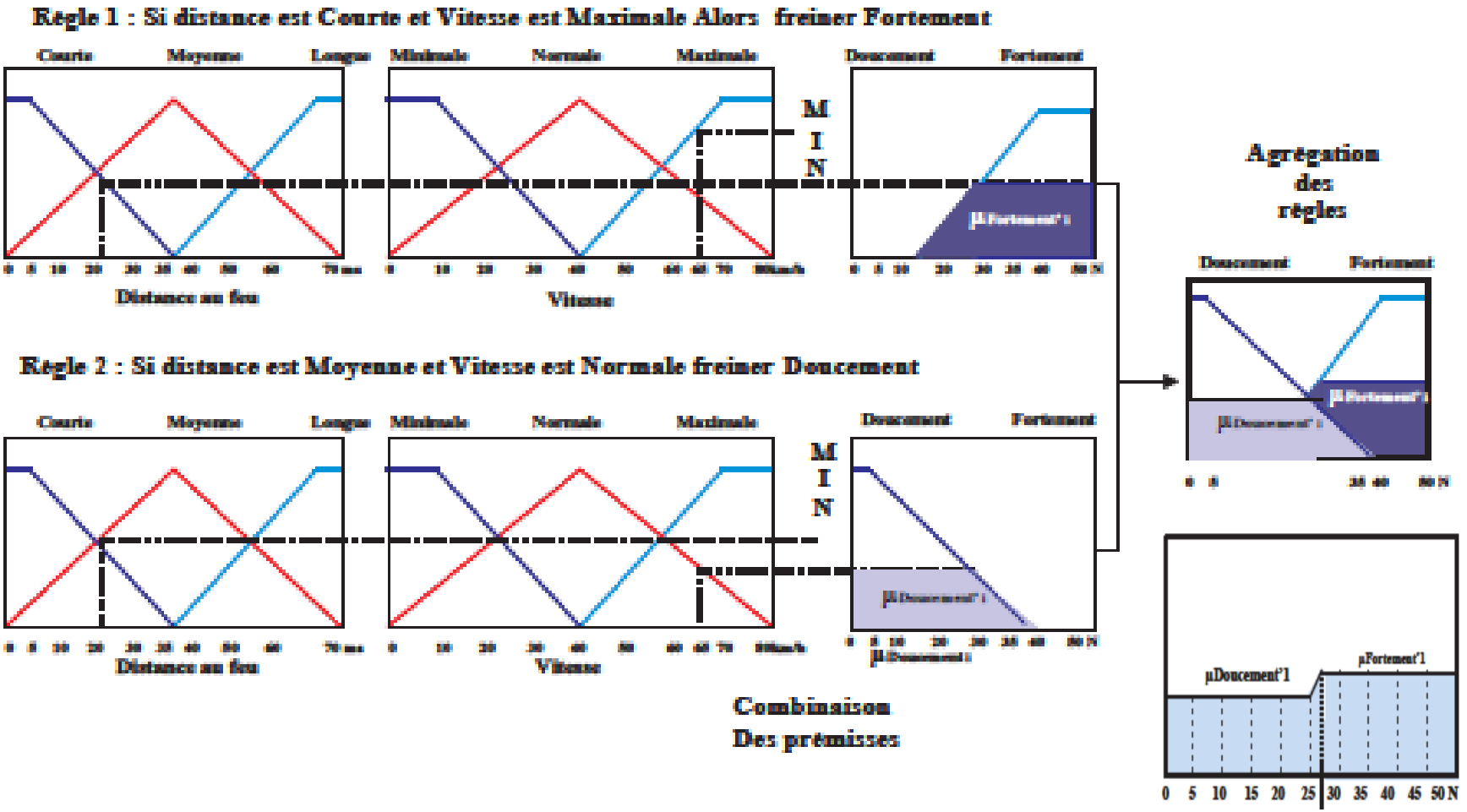
Comme on a vu que:

$$x_1 = 20 \rightarrow \mu_{Courte}(20) = 0.42\ ,\ \mu_{Moyenne}(20) = 0.57\ ,\ \mu_{Longue}(20) = 0.0$$

$$x_2 = 65 \rightarrow \mu_{Minimale}(65) = 0.0,\ \mu_{Normale}(65) = 0.375,\ \mu_{Maximale}(65) = 0.833$$

$$R_1 : \mu_{Fortement_1} = [min(\mu_{Courte}(0.42), \mu_{Maximale}(0.833))] = min(0.42, 0.833) = 0.42$$

$$R_2 : \mu_{Doucement_1} = [min(\mu_{Moyenne}(0.57), \mu_{Normale}(0.375))] = min(0.57, 0.375) = 0.375$$



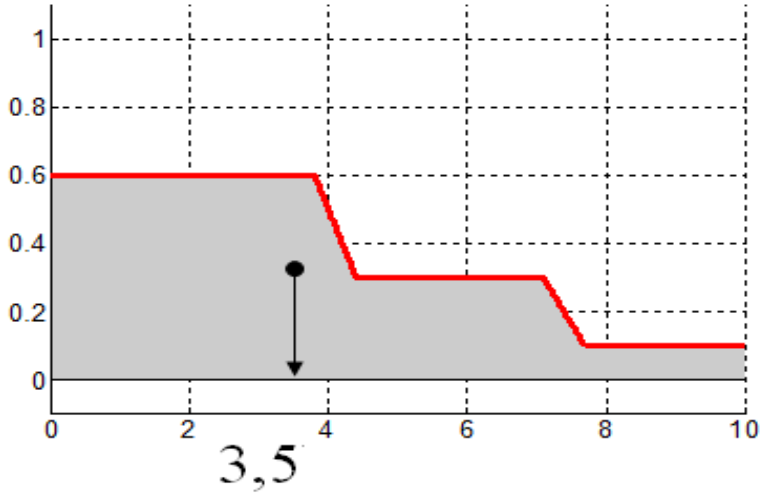
# Méthode de défuzzification

*Méthode du centre de gravité (COG)* C'est l'abscisse du centre de gravité de la surface du sous ensemble flou obtenu après agrégation des règles.

$$sortie = \frac{\int_U y \cdot \mu(y) \cdot dy}{\int_U \mu(y) \cdot dy}$$

$U$ = *Univers du discours*

= *Toutes les valeurs de sorties considérées*



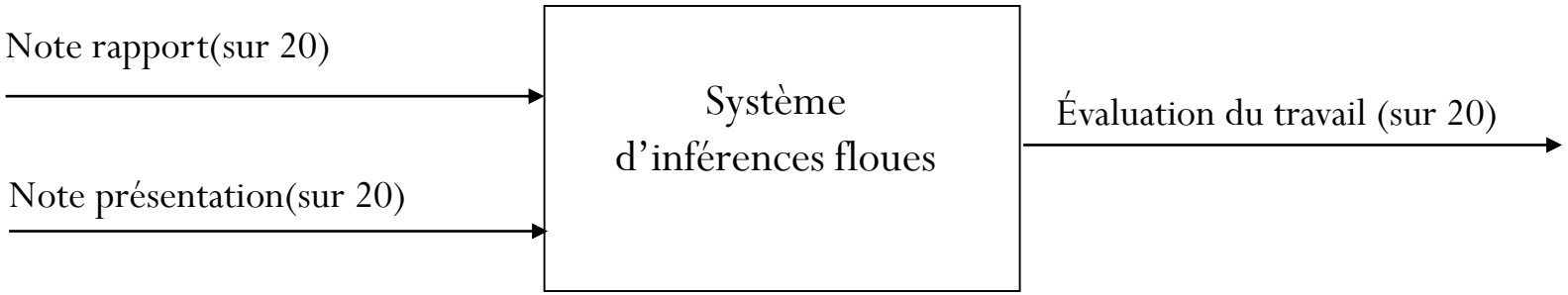
# Remarque:

Il existe d'autres méthodes de défuzzification [Ross, 2005] :

méthodes des maximums,

- ✓ somme−prod,
- ✓ moyenne−pondérée,
- ✓ moyenne des maximums.
- ✓ Etc…

# Exercice : Système d'évaluation floue des PFE Master



# *Mise en place du système d'inférences floues (1)*

1. Choix des entrées/sorties

2 entrées: Rapport; Présentation.
1 sortie: Évaluation

2. Univers des discours

[0..20] pour chacune des E/S

3. Classes d'appartenances:

Notes Rapport et Présentation $\{$Médiocre; Moyen; Excellent$\}$

Evaluation $\in \{$Médiocre; Mauvais; Moyen; Bon; Excellent$\}$

# Bases de règles (maximum 9 règles)

1. If (Rapport is excellent) and (Présentation is excellent) then (Evaluation is excellent)
2. . If (Rapport is moyen) and (Présentation is excellent) then (Evaluation is bon)
3. . If (Rapport is médiocre) and (Présentation is excellent) then (Evaluation is moyen)
4. . If (Rapport is médiocre) and (Présentation is moyen) then (Evaluation is mauvais)
5. . If (Rapport is médiocre) and (Présentation is médiocre) then (Evaluation is médiocre)
6. …

# Travaux Pratique Décision floue

Utilisation de la laibrairie **JFuzzyLogic en java**

Ou **Scikit Fuzzy en python**

```
FUNCTION_BLOCK tipper// Block definition
VAR_INPUT// Define input variables
    service : REAL;
    food : REAL;
END_VAR
VAR_OUTPUT// Define output variable
    tip : REAL;
END_VAR
FUZZIFY service// Fuzzify input variable 'service': {'poor', 'good' , 'excellent'}
    TERM poor := (0, 1) (4, 0) ;
    TERM good := (1, 0) (4,1) (6,1) (9,0);
    TERM excellent := (6, 0) (9, 1);
END_FUZZIFY
// voir explication des valeurs
FUZZIFY food// Fuzzify input variable 'food': { 'rancid', 'delicious' }
    TERM rancid := (0, 1) (1, 1) (3,0) ;
    TERM delicious := (7,0) (9,1);
END_FUZZIFY
```

```
DEFUZZIFY tip// output variable 'tip' : {'cheap', 'average', 'generous' }
TERM cheap := (0,0) (5,1) (10,0);
TERM average := (10,0) (15,1) (20,0);
TERM generous := (20,0) (25,1) (30,0);
METHOD : COG;// Use 'Center Of Gravity' defuzzification method
DEFAULT := 0;// Default value is 0 (if no rule activates defuzzifier)
END_DEFUZZIFY

RULEBLOCK No1
AND : MIN;// Use 'min' for 'and' (also implicit use 'max' for 'or' to fulfill
    DeMorgan's Law)
ACT : MIN;// Use 'min' activation method
ACCU : MAX;// Use 'max' accumulation method

RULE 1 : IF service IS poor OR food IS rancid THEN tip IS cheap;
RULE 2 : IF service IS good THEN tip IS average;
RULE 3 : IF service IS excellent AND food IS delicious THEN tip IS generous;
END_RULEBLOCK
END_FUNCTION_BLOCK
```

## *Quelques bibliothèques necessaires:*

- **import net.sourceforge.jFuzzyLogic.FIS;**
- **import net.sourceforge.jFuzzyLogic.rule.*; (ou** .rule.FuzzyRuleSet)

```
public static void main(String[] args) throws Exception {
String fileName = "c:/tipper.fcl";
    FIS fis = FIS.load(fileName,true);
if( fis == null ) {
        System.err.println("Can't load file:‘ »          + fileName + """);
        return;
    }
```

```
fis.chart(); // Visualiser les partitions floues de fuzzification
//Test:  // Set inputs
fis.setVariable("service", 4);
fis.setVariable("food", 7);
// Evaluate calcul de la décision pour la valeur de test
fis.evaluate();
// affichage de la décision:
System.out.println(fis.getVariable("tip").defuzzify());
fis.getVariable("tip").chartDefuzzifier(true);
```

# Exercice

- Réaliser sous Java (JFuzzyLogic) une application intelligente à base de logique floue pour prédire le montant d'un crédit que peut obtenir un client bancaire, en se basant sur son âge et son salaire. On suppose que le montant maximal de crédit accordé est de 100 000 DH.
- On suppose que l'âge des clients est compris entre [20 et 65] , et le salaire est entre 2500 et 35 000 DH.

Et On considère que :

$$Age \in \{Petit, Moyen, Grand\}$$

$$Salaire \in \{Petit, Moyen, Grand\}$$

$$Mon\tan t\_cr\acute{e}dit \in \{TrèsPetit, Petit, Moyen, Grand, TrèsGrand\}$$

## Exercice2:

Réaliser, sous JFuzzyLogic, une application d'aide à la conduite d'un véhicule, (système de freinage intelligent):

- On considère que la vitesse et la distance entre deux véhicules seront représentées par 5 valeurs linguistiques et le freinage par 7 valeurs linguistiques.
- On suppose que la vitesse maximale du véhicule est de 200 Km/h , la distance maximale qui nous intéresse entre les deux véhicules est de 100 m et le freinage varie entre 0 et 100.

| | |
|---|---|
| ET flou | : MIN |
| OU flou | : MAX |
| Implication floue | : MIN |
| Agrégation des règles | : MAX |
| Défuzzification | : COG |

# Installez Scikit-fuzzy

- **Lancer anaconda prompt(anaconda3)**
- **conda install -c conda-forge scikit-fuzzy**

# Introduction à l'Apprentissage Artificiel

**Agent intélligent**
**À base de règles:**
**(systèmes expert)**
**Logique floue**

**Agent intélligent**
**À base d'Apprentissage:**
**(Machine learning)**
- **Réseaux de Neurones**
- **Support vecteur machines**

**…..**

## Apprentissage Naturel (Humain)

Dès sa naissance l'enfant apprend:

- La voix de sa mère
- Apprend à marcher
- Apprend à lire
- …

## Apprentissage Artificiel (ou Automatique)

Les machines (agents) ont besoin d'apprendre

L'apprentissage est essentiel pour des environnements inconnus, où le concepteur ne peut pas tout savoir à l'avance.

Ce qui permet d'améliorer la capacité de l'agent à agir dans le futur

## *Les données pour l'apprentissage:*
## *Ensemble d'apprentissage S* et un *ensemble test T*

- On partitionne l'échantillon en un ***ensemble d'apprentissage S* et un *ensemble test T*.**

- **Exemple:**

**reconnaissance de chiffres:** Les données?

# Types d'apprentissage

- **Apprentissage supervisé (Existence de superviseur)**

(**sortie désirée(Target): connue**)

- **Apprentissage non-supervisé(Clustering)**

**sortie désirée inconnue**

- **Apprentissage par renforcement(par experiences)**

# Apprentissage supervisé

- $S = \{(X1, y1), (X2, y2), \ldots, (Xi, yi), \ldots, (Xm, ym)\}$
- $T = \{Xt1, Xt2,, \ldots, Xti, \ldots, Xtp\}$

**Objectif est de chercher Une fonction de décision (classifieur)**

# Deux types de problèmes

Régression(prédire des valeurs numériques continues)

et

Classification(les valeurs à prédire sont discrètes)

Apprentissage supervisée:

# Formalisme de la de la classification supervisée

- Un ensemble $X$ = X1 x X2x…Xn ou chaque Xi est le domaine d'un attribut ***Xi symbolique*** ou ***numérique***.
- Un ensemble fini de classes ***Y(classes désirées)***
- Les exemples sont des couples (x; y)$\in$***X***x ***Y***
- Un échantillon S est un ensemble fini d'exemples (Dataset)
- ***Classifieur*** : $\mathbf{f} : X \rightarrow Y$

***Le problème général de la classification supervisée :***
***étant donné un échantillon S = {(x1; y1); …. ; (xn ; yn)}, trouver un classifieur f qui minimise une fonction objectif (risque) R(f ).***

# Fonction risque (ou fonction erreur )

***Fonction de perte*** : $L(y, f(x)) = \begin{cases} 0 \text{ si } y = f(x) \\ 1 \text{ sinon.} \end{cases}$

Où y sortie désiré donné par le superviseur et f(x) sortie du classifieur

L'espérance mathématique de cette mesure permet de définir la ***fonction risque*** suivante(***dans le cas continu***):

$$R(f) = \int L(y, f(x)) dP(x, y)$$

***Le problème est de trouver un classifieur f qui minimise le risque R(f ).***

# Pour le problème régression

Dans un problème de régression, *y* prend des valeurs continues et l'on cherche également à exprimer par une fonction la dépendance entre x et y. La fonction de perte qu'on considère principalement est l'écart quadratique défini par:

$$L(y, f(x)) = (y - f(x))^2.$$

$$y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

L'erreur d'une fonction f est alors l''ecart quadratique moyen défini par :

$$R(f) = \int_{\mathcal{X} \times \mathcal{Y}} (y - f(x))^2 dP(x, y).$$

# Pratiquement: utiliser Risque empirique

- Le risque empirique Remp(f) d'une fonction f sur l'échantillon S = {(x1, y1), . . . (xl, yl)} est la moyenne de la fonction de perte calculée sur S:

$$R_{emp}(f) = \frac{1}{l} \sum_{i=1}^{l} L(y_i, f(x_i)).$$

Remp(f) est la moyenne du nombre d'erreurs de prédiction de f sur les éléments de S :

$$R_{emp}(f) = \frac{Card\{i | f(x_i) \neq y_i\}}{l}.$$

Lorsque f est une fonction de **régression** et L la fonction de perte quadratique, Remp(f) est la moyenne des carrés des écarts à la moyenne de f sur S :

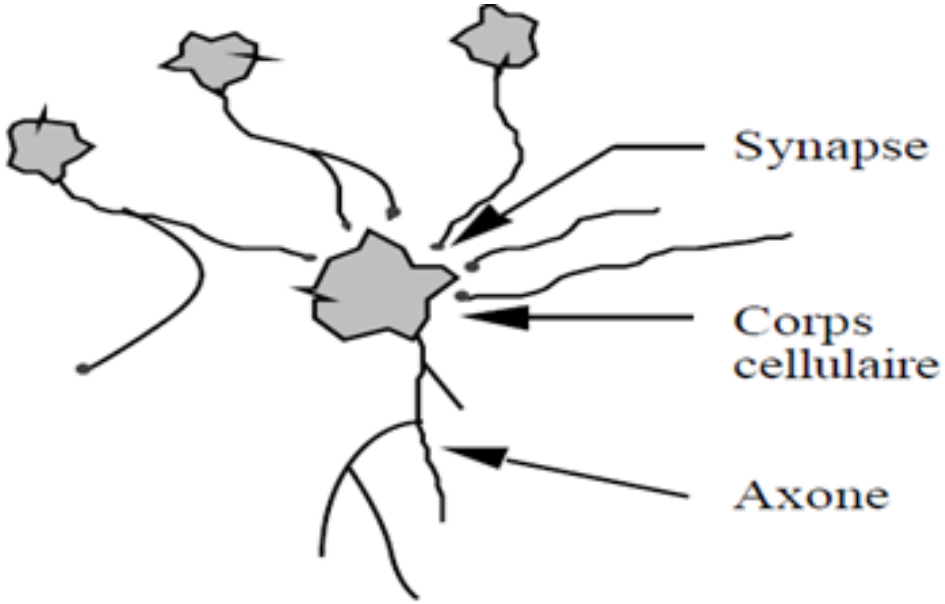
$$R_{emp}(f) = \frac{1}{l} \sum_{i=1}^{l} (y_i - f(x_i))^2.$$

# Réseaux de neurones:
# Une origine biologique (Système nerveux )

- Comment l'homme fait-il pour apprendre et calculer à travers son cerveau …?
- Comment s'y prendre pour créer une intelligence artificielle ?
- La physiologie du cerveau montre que celui-ci est constitué de cellules (les neurones) interconnectées:

- Un neurone = cellule cérébrale dont la fonction principale consiste à **collecter**, **traiter** et **transmettre** des informations

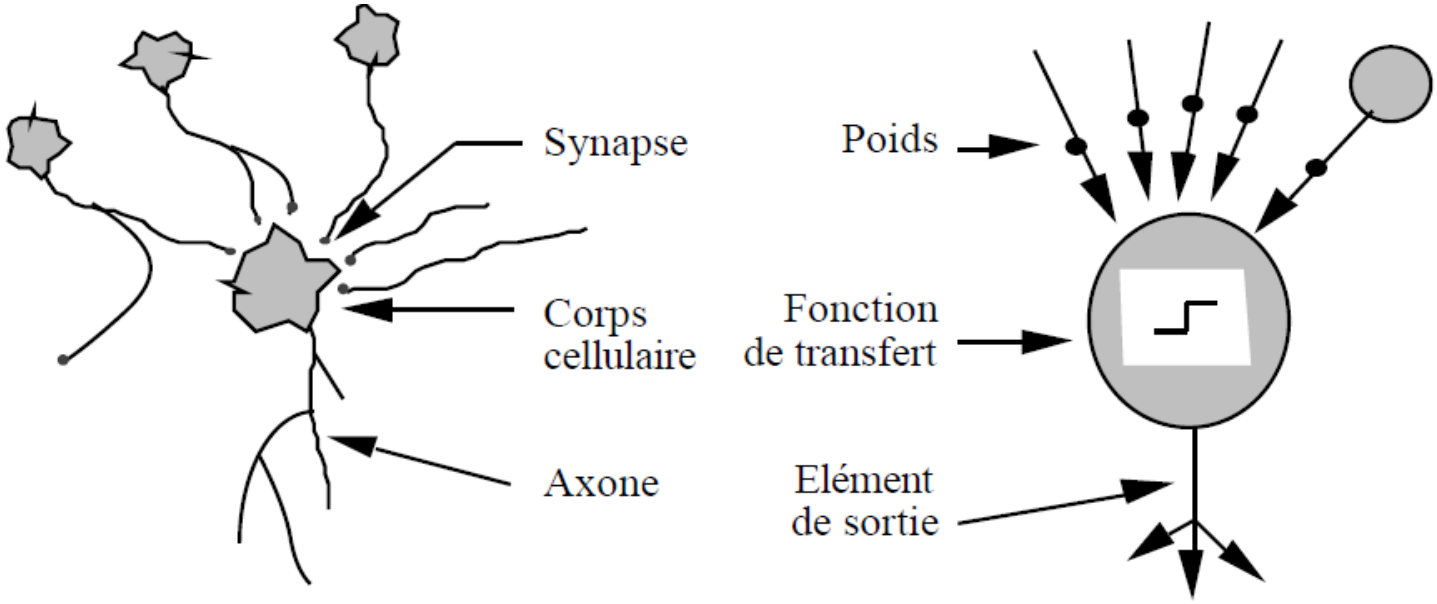
- le cerveau contient environ **100 milliards** de neurones
- nombre total de connexions est estimé à environ **10^15** .
- le nombre de neurones actifs décroit,




***L'apprentissage est  indispensable à son développement***

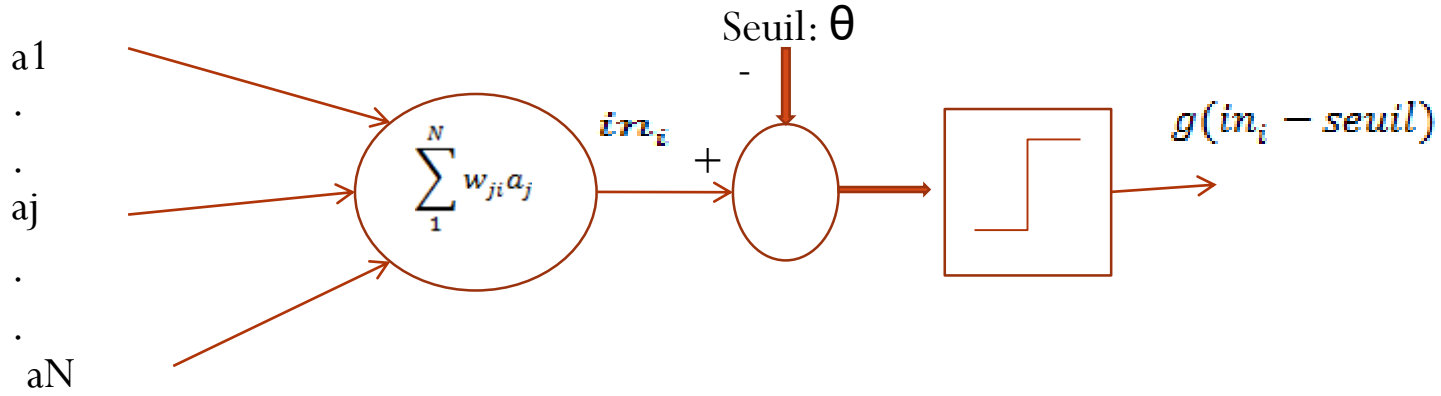
# Du neurone biologique au neurone artificiel

Les *neurones* ***reçoivent*** *des informations par les dendrites* (par l'intermédiaire des ***synapses***) ***traitent*** *et* ***envoient*** *l'information par les axones.*




- Dendrites : Signaux d'entrée
- Axone : Signal de sortie

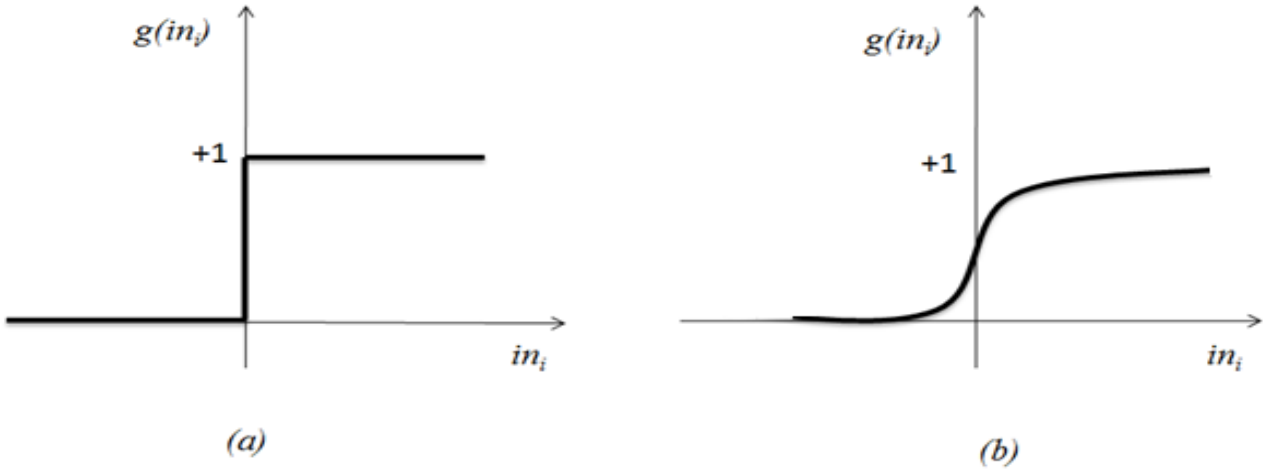
- Le neurone reçoit les entrées a1, …, ai, …, an.
- Le potentiel d'activation du neurone est défini comme la somme pondérée (les poids sont les coefficients synaptiques wi) des entrées.
- La sortie y est alors calculée en fonction du seuil θ.




$$s_i = g((\sum_{j=1}^{n} W_{j,i} a_j) - seuil)$$

***Modèle d'un neurone artificiel***

# Fonctions d'activations(seuillage)



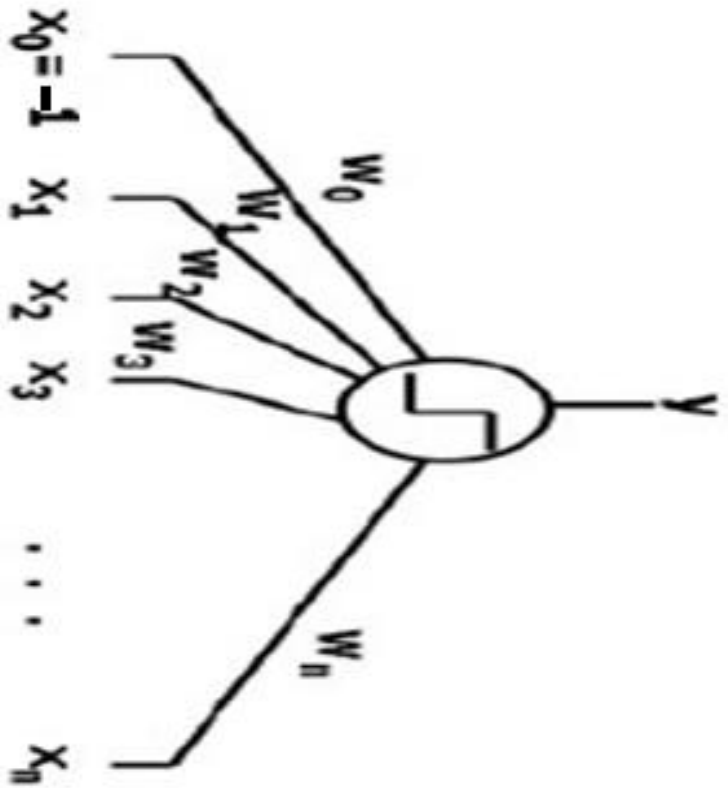

- (a) : seuil (fonction de Heavyside)
- (b) : sigmoïde $g(x) = (1 + e^{-\beta x})^{-1}$

*La fonction d'activation g est conçue pour que l'unité soit « active » (proche de +1) quand les « bonnes » entrées sont données, et « inactive » (proche de 0) quand elles sont « mauvaises ».*

# Forme Standard Utilisé (Neurone à Biais)

On ajoute une entrée supplémentaire x0 (le biais), avec le coefficient synaptique suivant : $w0 = \theta$

Coefficient de biais w0 connecté à une entrée x0=-1



# Définitions

• Déterminer un réseau de neurones = Trouver les coefficients synaptiques. On parle de phase d'***apprentissage : les caractéristiques du réseau*** sont modifiées jusqu'à ce que le comportement désiré soit obtenu.

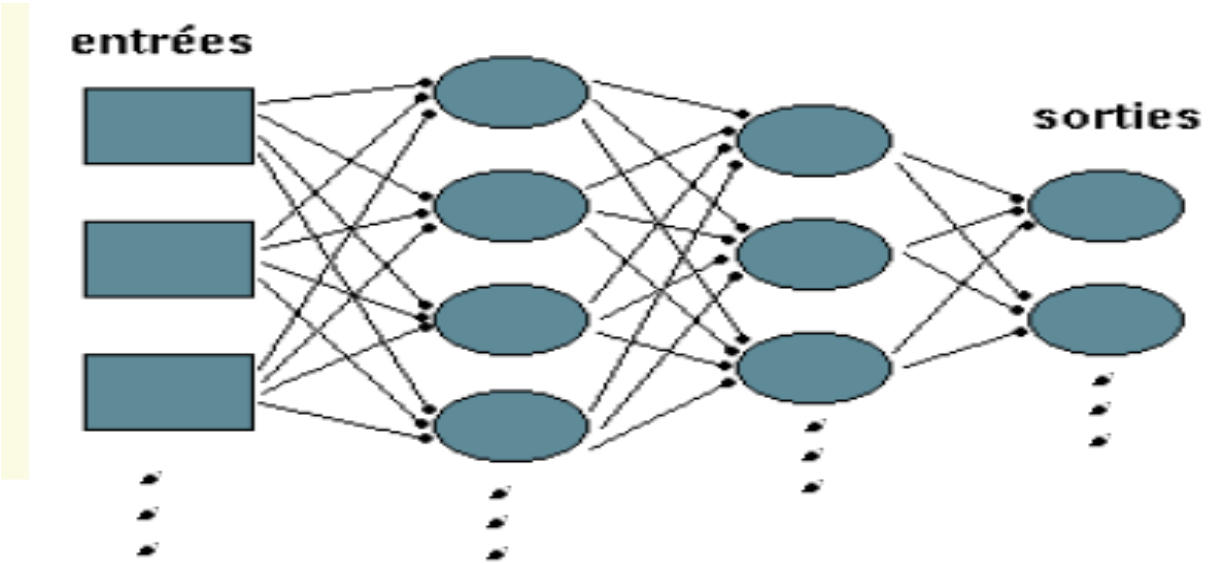
• ***Données d'apprentissage : exemples représentatifs du comportement ou***

de la fonction à modéliser. Ces exemples sont sous la forme de couples (entrée ; sortie) connus.

• ***Données de test : pour une entrée quelconque (bruitée ou incomplète),***

calculer la sortie. On peut alors évaluer la performance du réseau.

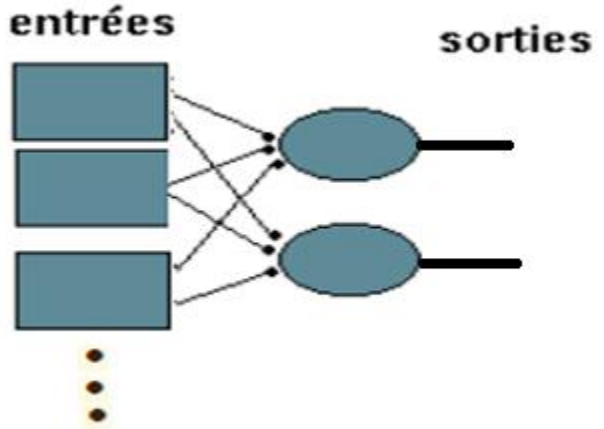
# Structures des réseaux de neurones



# Réseaux de neurones monocouche
# --Le perceptron--

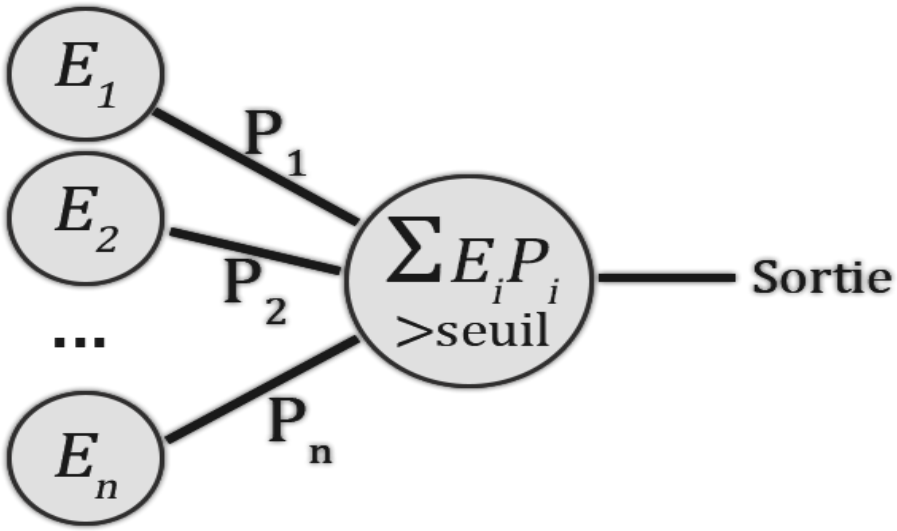
Toutes les entrées sont directement connectées aux sorties

Schéma:




Comme chaque unité de sorties est indépendante des autres on limite notre étude à une sortie.

On distingue généralement deux types:
Perceptron à seuil et Perceptron basé sur la descente du gradient



# Perceptron à seuil

**Algorithm Perceptron à seuil:**

1/ Initialisation des poids (y compris le bias) à des valeurs (petites) choisies au hasard.

2/ Présentation d'une entrée X = (x0,x1, … xn) de la base d'apprentissage.

3/ Calcul de la sortie obtenue y pour cette entrée :

$a = \sum(wi \cdot ei)$

y = signe (a) ( si a > 0 alors y = +1 sinon alors y = -1 )

4/ Si la sortie y du Perceptron est différente de la sortie désirée yd pour cet exemple d'entrée

alors modification des poids ($\mu$ le pas de modification choisi entre 0 et 1) :

$wi(t+1) = wi(t) + \mu.((Err).xi)$ où l'erreur : err= yd – y.

Ainsi de suite jusqu'à terminer l'ensemble d'exemples d'apprentissage avec une erreur nulle pour ***tous les exemples***, on retient alors les derniers poids ;

Considérant une fonction d'activation à ***seuil*** (fonction majorité) qui retourne 1 ssi

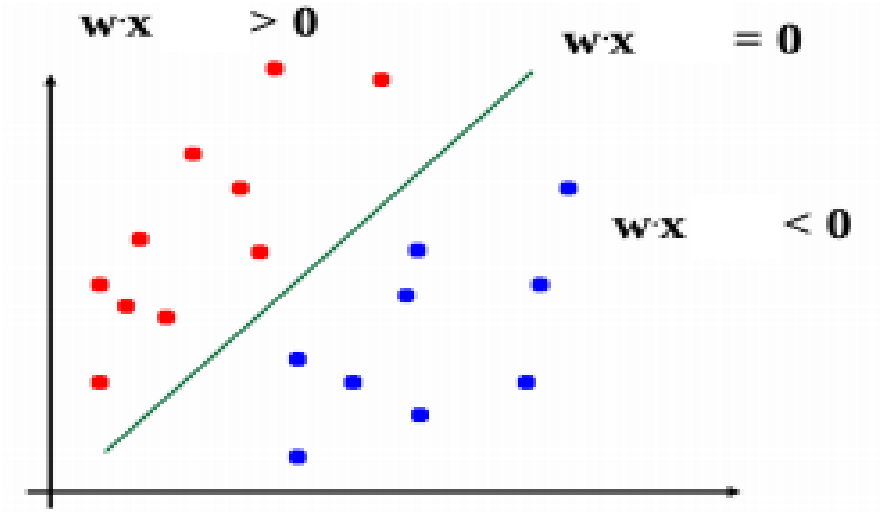
$$\sum_{j=0}^{n} wj\ xj > 0 \quad \text{ou} \quad W.X>0$$

Or: l'équation W.X=0 définit un hyperplan dans l'espace des entrées le perceptron ne retourne 1 que ssi l'entrée se trouve sur un côté de cet hyperplan

**Déf:** Si E est un espace vectoriel de dim n, alors tous sous espace H de dim n-1 s'appelle hyperplan de E

Càd: le perceptron ne retourne 1 que ssi l'entrée se trouve sur un coté de cet hyperplan

Perceptron à seuil = qualifié de séparateur linéaire



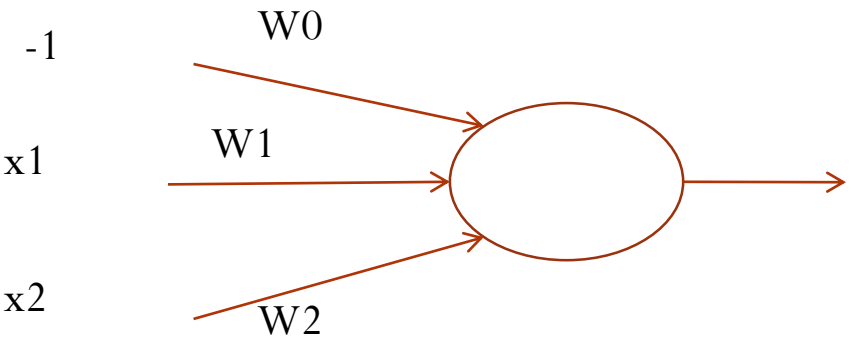

**On dit que les données sont linéairement séparables s'il existe un hyperplan qui sépare les deux catégories**

# Exemple de fonctionnement de l'algorithme d'apprentissage du Perceptron à seuil:

Application aux portes logiques (AND, OR,…)

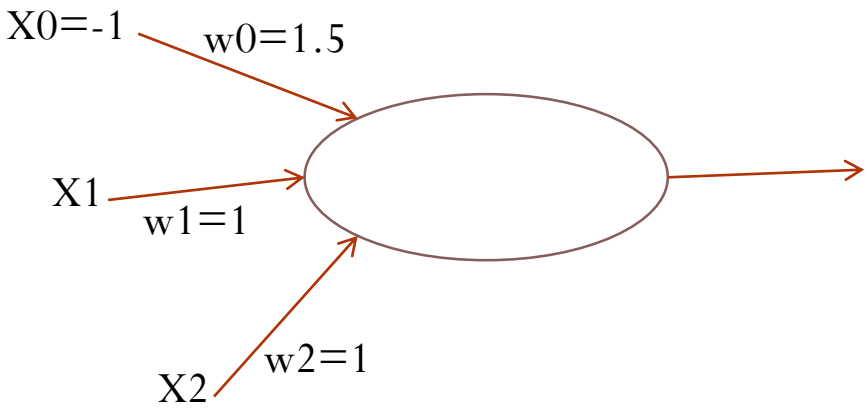
Base d'exemples d'apprentissage :X=(x1,x2), e=(X,yd)

| x1 | x2 | yd | |
|---|---|---|---|
| 1 | 1 | 1 | e1 |
| 0 | 1 | 0 | e2 |
| 1 | 0 | 0 | e3 |
| 0 | 0 | 0 | e4 |

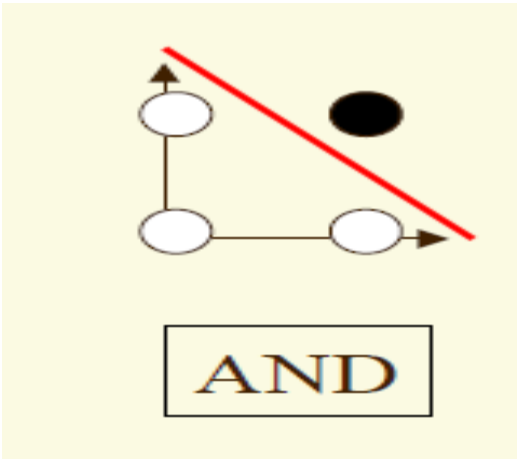



**Exécuter l'algorithme à seuil sur ces exemples d'apprentissage avec:**
**un taux d'apprentissage =0.5**
**Initialisation des poids: (1.5, 0.5, 0.5)**

# Exemple: perceptron à seuil de AND



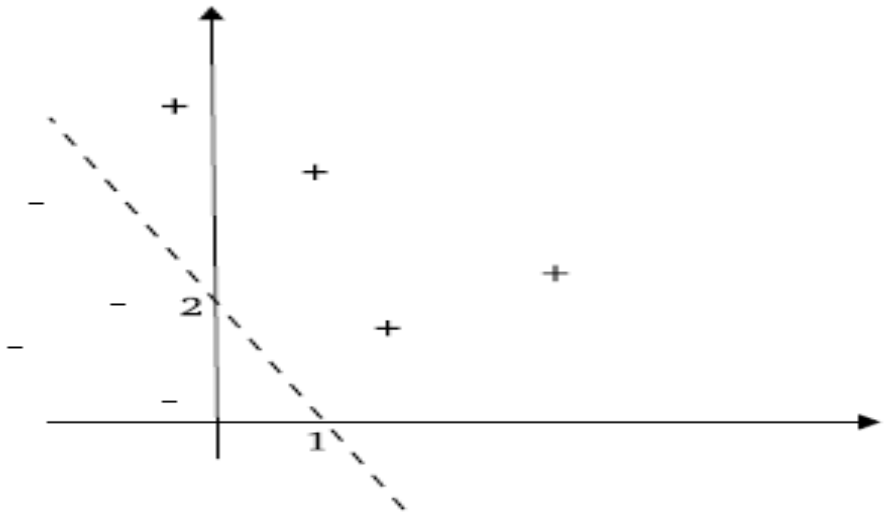

| x1 | x2 |
|---|---|
| 1 | 1 |
| 0 | 1 |
| 1 | 0 |
| 0 | 0 |



## Exercice:

Soit un perceptron dont le vecteur de pondération $(w_0, w_1, w_2)^T = (2,1,1)^T$

1. Tracer sur un diagramme le séparateur linéaire obtenu par ce perceptron et hachurer la surface correspondante à la partie du plan où le perceptron retourne la valeur 1.
2. Quelles sont les valeurs des poids w0; w1 et w2 du perceptron dont la frontière de décision est illustrée ci-dessous ? Y a t-il plusieurs choix possibles pour ces valeurs de poids ? Si les étiquettes de classification (+ ou -) sont inversées, les poids resteront-ils les mêmes ?



**Remarque:** l'algorithme à seuil fonctionne dans le cas où Les données sont linéairement séparables

Dans le cas contraire exemple de xor, cet algorithme ne fonctionne pas.

# Problème: ce n'est pas toutes les fonctions sont séparables

**Exemple:** la fonction XOR

---

**Algorithme 1. Perceptron à sigmoïde**

---

**Algorithme d'apprentissage par rétropropagation du gradient pour les perceptrons, supposant une fonction d'activation différentiable :**

**entrées** : *exemples*, un ensemble d'exemples, chacun avec l'entrée

$\mathbf{x} = x_1 \ldots x_n$ et la sortie $y$

réseau, un perceptron avec les poids $W_j$, $j = 0 \ldots n$,

et une fonction d'activation $g$

**répéter**

**pour chaque** $e$ **dans** *exemples* **faire**

$$in \leftarrow \sum_{j=0}^{n} W_j x_j[e]$$

$$Err \leftarrow y[e] - g(in)$$

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$$

**jusqu'à ce qu**'un critère d'arrêt quelconque soit satisfait

# *Démonstration*: Descente du gradient

$$g(x) = \frac{1}{(1+e^{-x})} \qquad g'(x) = g(x)(1-g(x))$$

**Descente du gradient**

$$W_j \leftarrow W_j - \alpha * \frac{\delta E}{\delta W_j}$$

***Démonstration algorithme1:***

*Voir démonstration en classe*

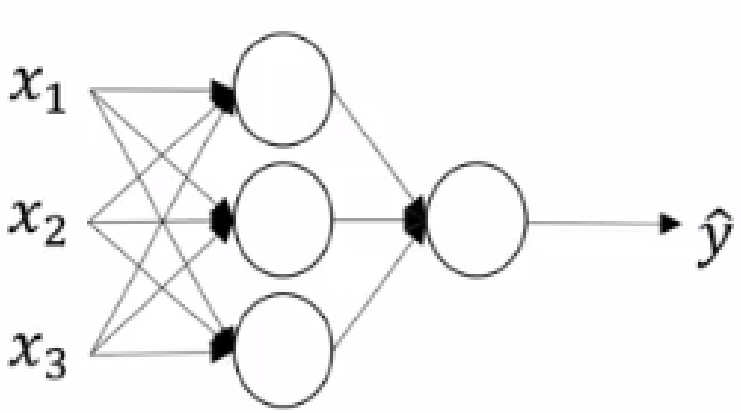
# *Etapes de mise en œuvre d'un réseau de neurones pour la prédiction ou la classification*

1. *Identification des données en entrée et en sortie*
2. *Constitution de la structure du réseau à utiliser*
3. *Apprentissage du réseau*
4. ***Test du réseau***

# *Perceptron Multi-couches*

1. **Définir la structure du MLP( taille de l'entrée, nombre de de neurones de la couche cachée, etc).**
2. **Initialiser les paramètres du modèle**
3. **Boucle:**
   - **Implémenter « forward propagation »**
   - **calculer la fonction du coût**
   - **Implémenter « backward propagation » pour trouver les gradients**
   - **Adapter les paramètres (gradient descent)**

Fonctions d’activations



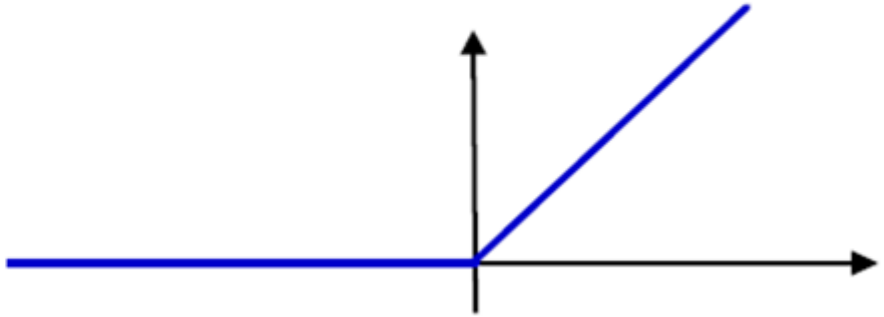

Sigmoid

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

Relu

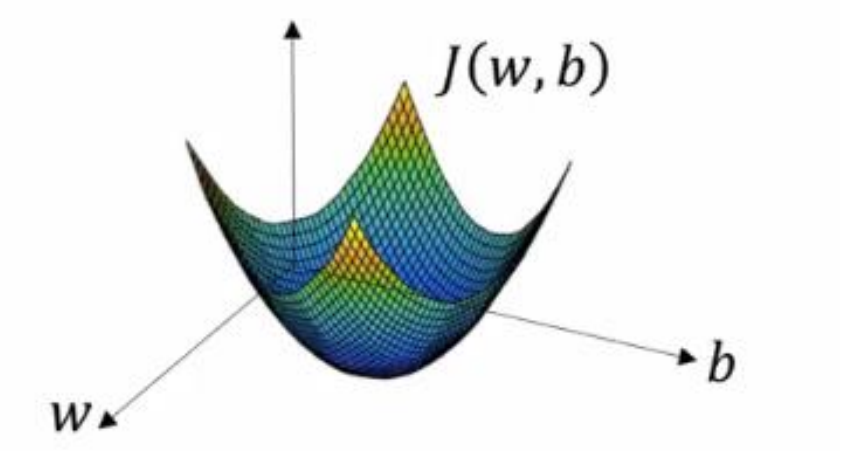
- ReLU : Rectifier Linear Unit
- Introduite en 2010 par Nair et Hinton

$$f(x) = \begin{cases} x & \text{si } x \geq 0 \\ 0 & \text{sinon} \end{cases}$$



## Calcul du coût (cost):

$$J = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right)$$



Entrée [13]:

```
import numpy as np
import time
```

Entrée [14]:

```
a=np.random.rand(1000000)
b=np.random.rand(1000000)
debut=time.time()
c=np.dot(a,b)
fin=time.time()
print(c)
print("vectorial   "+str(1000*(fin-debut))+"ms")
c=0
debut=time.time()
for i in range(1000000):
    c+=a[i]*b[i]
fin=time.time()
print(c)
print("Boucle   "+str(1000*(fin-debut))+"ms")
```

```
249825.699712114
vectorial   1.0051727294921875ms
249825.69971210771
Boucle   551.4650344848633ms
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets
iris = datasets.load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target)
mlp =
MLPClassifier(hidden_layer_sizes=(13,13,3),max_iter=500)
mlp.fit(X_train,y_train)
print(mlp.score(X_test,y_test))
print(mlp.predict(X_test))
```

# Jeux de données Iris

- Iris est un ensemble de données qui est disponible dans scikit-learn dans le package ***datasets***.
- Comme tout datasets, il est constitué de deux dictionnaires
- .data qui stocke un tableau de dimensions nxm ou n est le nombre d'instances, et m le nombre d'attributs
- .target qui stocke les classes, cibles (étiquettes) de chaque instance (dans le cas supervisé).

| Sepal length | Sepal width | Petal length | Petal width | Species |
|---|---|---|---|---|
| 5.1 | 3.5 | 1.4 | 0.2 | *I. setosa* |
| 4.9 | 3.0 | 1.4 | 0.2 | *I. setosa* |
| 4.7 | 3.2 | 1.3 | 0.2 | *I. setosa* |
| 4.6 | 3.1 | 1.5 | 0.2 | *I. setosa* |
| 5.0 | 3.6 | 1.4 | 0.2 | *I. setosa* |
| 5.4 | 3.9 | 1.7 | 0.4 | *I. setosa* |
| 4.6 | 3.4 | 1.4 | 0.3 | *I. setosa* |
| 5.0 | 3.4 | 1.5 | 0.2 | *I. setosa* |
| 4.4 | 2.9 | 1.4 | 0.2 | *I. setosa* |
| 4.9 | 3.1 | 1.5 | 0.1 | *I. setosa* |
| 5.4 | 3.7 | 1.5 | 0.2 | *I. setosa* |
| 4.8 | 3.4 | 1.6 | 0.2 | *I. setosa* |

*Iris setosa*

| | | | | |
|---|---|---|---|---|
| 7.0 | 3.2 | 4.7 | 1.4 | *I. versicolor* |
| 6.4 | 3.2 | 4.5 | 1.5 | *I. versicolor* |
| 6.9 | 3.1 | 4.9 | 1.5 | *I. versicolor* |
| 5.5 | 2.3 | 4.0 | 1.3 | *I. versicolor* |
| 6.5 | 2.8 | 4.6 | 1.5 | *I. versicolor* |
| 5.7 | 2.8 | 4.5 | 1.3 | *I. versicolor* |
| 6.3 | 3.3 | 4.7 | 1.6 | *I. versicolor* |
| 4.9 | 2.4 | 3.3 | 1.0 | *I. versicolor* |
| 6.6 | 2.9 | 4.6 | 1.3 | *I. versicolor* |
| 5.2 | 2.7 | 3.9 | 1.4 | *I. versicolor* |
| 5.0 | 2.0 | 3.5 | 1.0 | *I. versicolor* |
| 5.9 | 3.0 | 4.2 | 1.5 | *I. versicolor* |
| 6.0 | 2.2 | 4.0 | 1.0 | *I. versicolor* |
| 6.1 | 2.9 | 4.7 | 1.4 | *I. versicolor* |

*Iris versicolor*

| | | | | |
|---|---|---|---|---|
| 6.3 | 3.3 | 6.0 | 2.5 | *I. virginica* |
| 5.8 | 2.7 | 5.1 | 1.9 | *I. virginica* |
| 7.1 | 3.0 | 5.9 | 2.1 | *I. virginica* |
| 6.3 | 2.9 | 5.6 | 1.8 | *I. virginica* |
| 6.5 | 3.0 | 5.8 | 2.2 | *I. virginica* |
| 7.6 | 3.0 | 6.6 | 2.1 | *I. virginica* |
| 4.9 | 2.5 | 4.5 | 1.7 | *I. virginica* |
| 7.3 | 2.9 | 6.3 | 1.8 | *I. virginica* |
| 6.7 | 2.5 | 5.8 | 1.8 | *I. virginica* |
| 7.2 | 3.6 | 6.1 | 2.5 | *I. virginica* |
| 6.5 | 3.2 | 5.1 | 2.0 | *I. virginica* |
| 6.4 | 2.7 | 5.3 | 1.9 | *I. virginica* |
| 6.8 | 3.0 | 5.5 | 2.1 | *I. virginica* |
| 5.7 | 2.5 | 5.0 | 2.0 | *I. virginica* |
| 5.8 | 2.8 | 5.1 | 2.4 | *I. virginica* |

*Iris virginica*

*class* sklearn.neural_network.**MLPClassifier**(*hidden_layer_sizes=(100, )*, *activation='relu'*, *solver='adam'*, *alpha=0.0001*, *batch_size='auto'*, *learning_rate='constant'*, *learning_rate_init=0.001*, *power_t=0.5*, *max_iter=200*, *shuffle=True*, *random_state=None*, *tol=0.0001*, *verbose=False*, *warm_start=False*, *momentum=0.9*, *nesterovs_momentum=True*, *early_stopping=False*, *validation_fraction=0.1*, *beta_1=0.9*, *beta_2=0.999*, *epsilon=1e-08*, *n_iter_no_change=10*, *max_fun=15000*)

**batch_size*int, default='auto'***
Size of minibatches for stochastic optimizers. If the solver is 'lbfgs',
the classifier will not use minibatch. When set to "auto", batch_size=min(200, n_samples)

# Keras

## Construction rapide de vos programmes réseaux de neurones

Créer votre environnement de travail virtuel

- **Keras sequential models**
- **Keras Functional Models**

# Keras sequential models

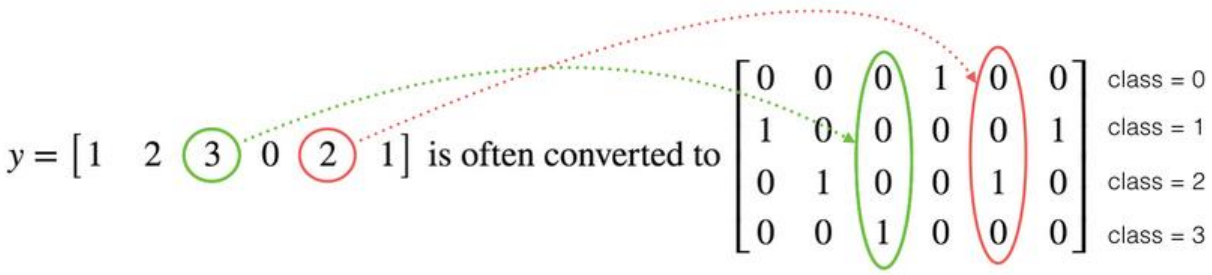
***Multi Layers Perceptron***

```
from keras.models import Sequential
from keras.layers import Dense,Activation

model = Sequential()
model.add(Dense(1024, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

# Chargement du Dataset MNIST

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

```
from keras.utils import np_utils
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
```



```
model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])

model.fit(X_train,
Y_train,batch_size=100,epochs=10,verbose=1)


 # Evaluer le modèle
scores = model.evaluate(x_test , y_test)
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

# Keras Functional Models

## ***Définir la couche d'entrée (input)***

- Il faut créer et définir une couche d'entrée qui spécifie les dimensions (shapes) des données d'entrée, cette couche d'entrée prend en argument un **tuple** qui indique la dimension de l'entrée,
- Si les données d'entrées sont de dimension 1, shape=(N,)

Exemple: shape=(3,)

```
From keras.layers import Input
X=Input(shape=(3,))
```

# Connextion des couches:

```
from keras.layers import Input
from keras.layers import Dense
X=Input(shape=(3,))
h1=Dense(3)(X)
```

# Création du modèle

- Après la création des couches et leurs connextion il faut créer le modèle:
- ***From keras.models import Model***
- ***model=Model(inputs=X,outputs=h1)***

# Multi Layers Perceptron

- *From keras.utils import plot_model*
- *From keras.models import Model*
- *From keras.layers import Input*
- *From keras.layers import Dense*

```
X=Input(shape=(10,))
hidden1=Dense(10,activation='relu')(X)
hidden2=Dense(20,activation='relu')(hidden1)
hidden3=Dense(10,activation='relu')(hidden2)
output=Dense(1,activation='sigmoid')(hidden3)
model=Model(inputs=X,outputs=output)
Print(model.summary())
```

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print(X_train.shape[1], 'train samples')
print(X_test.shape[0], 'test samples')
```

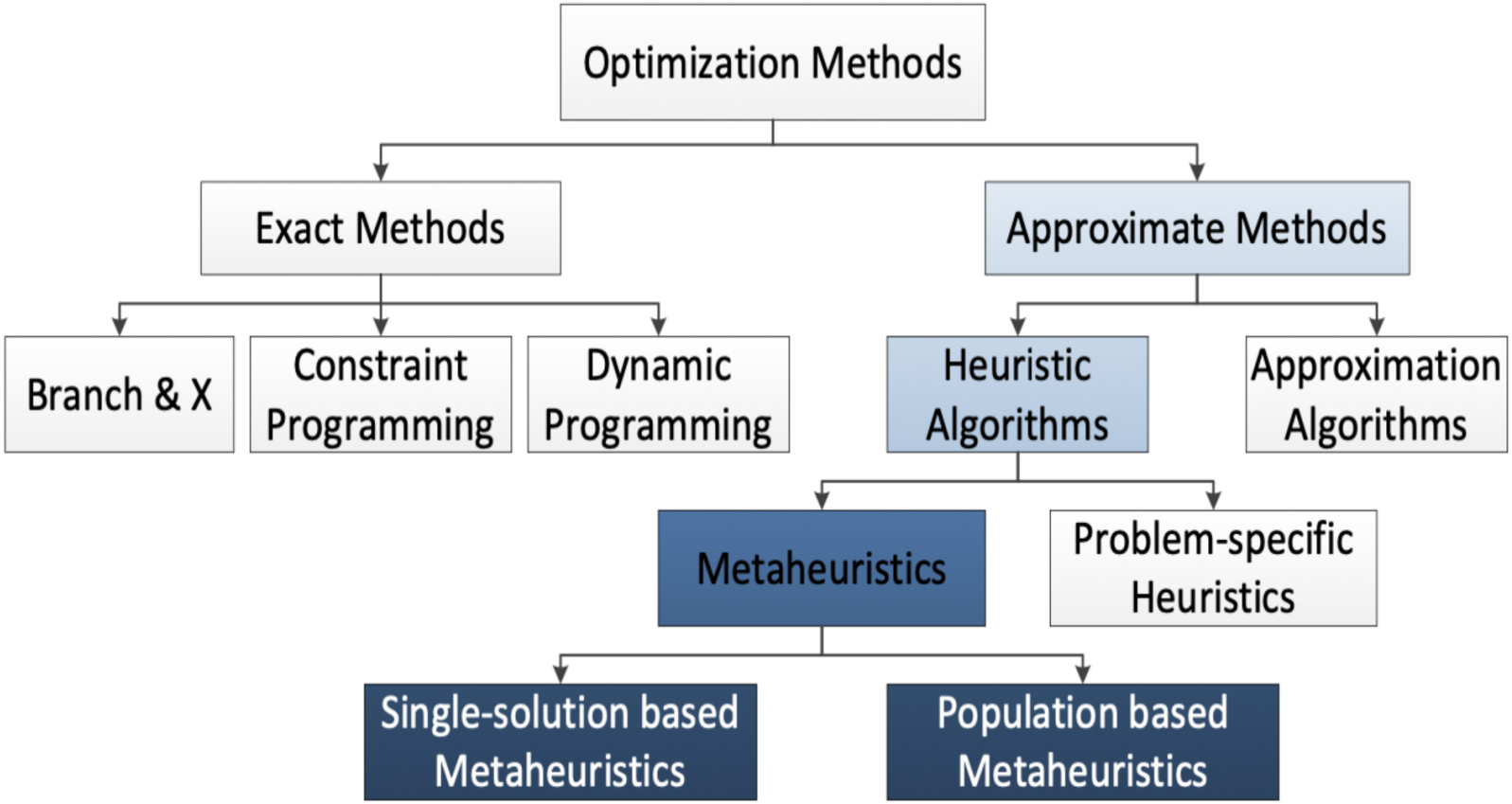
```
from keras.utils import np_utils
batch_size = 100
nb_epoch = 10

Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
print(X_train.shape)
print(Y_train.shape)
```

```
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
X=Input(shape=(784,))
hidden1=Dense(120,activation='relu')(X)
hidden2=Dense(84,activation='relu')(hidden1)
output=Dense(10, activation='softmax')(hidden2)
model=Model(inputs=X,outputs=output)
print(model.summary())
```

```
model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])

model.fit(X_train,Y_train,batch_size=batch_size,
epochs=nb_epoch,verbose=1)

# Evaluate the model
scores = model.evaluate(X_test ,Y_test)
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
import random
from keras.models import Model
from keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from keras.layers import Dense,Activation
irisData = datasets.load_iris()
x_train,x_test,y_train,y_test=train_test_split(irisData.data,irisData.target,test_size=0.2,r
andom_state=random.seed())

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

model = Sequential()
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=3, activation = 'softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train , y_train,batch_size=16,epochs=100,validation_data=(x_test , y_test))
scores = model.evaluate(x_test , y_test)
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

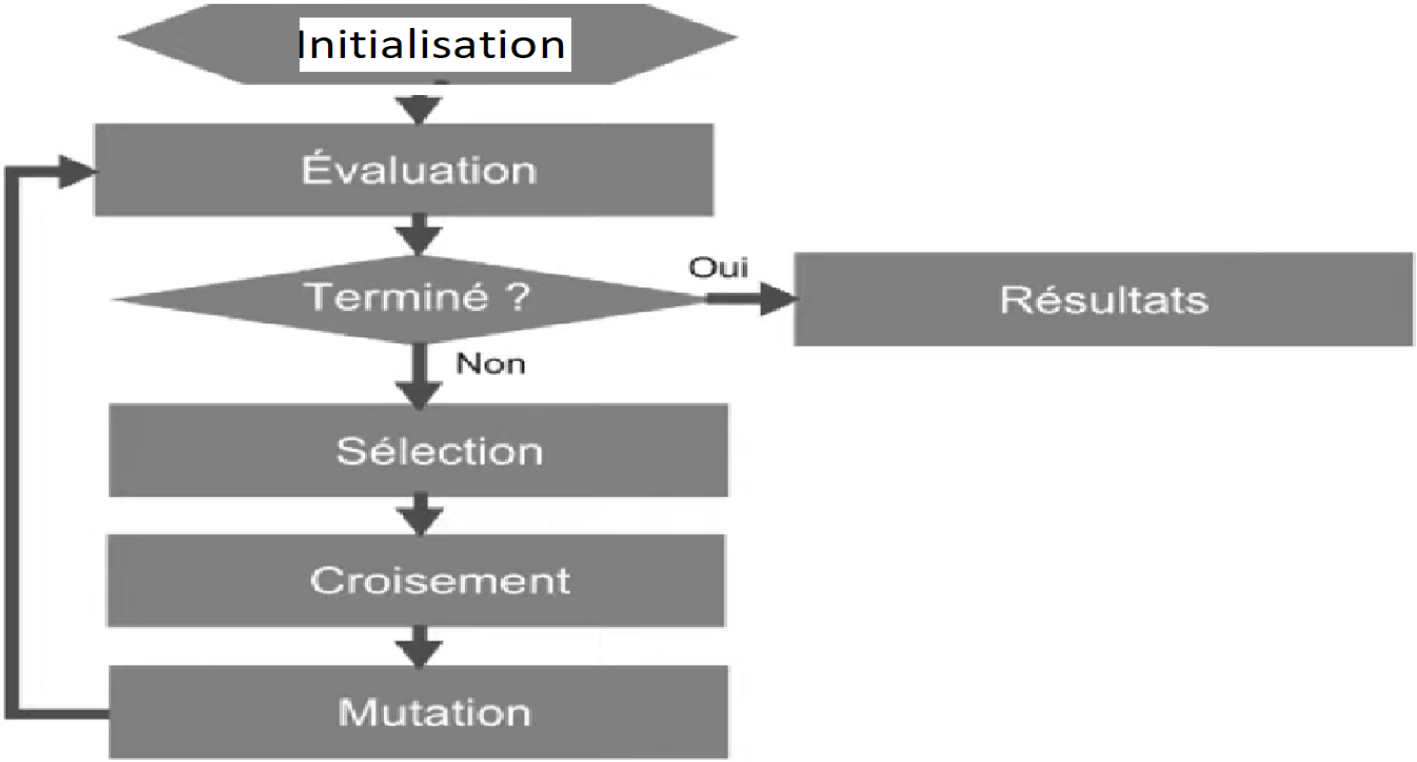
# Méthodes d'optimisation: Algo évolutionnaires



**Métaeuristiques à population (Inspiration naturel):**

- **AG**
- **Colonie de Fourmies**
- **essaimes de particules**

# Algorithme génétique:




Partir d'une population d'individus (solutions), et les faire évoluer afin d'obtenir l'individu optimal (la solution optimale)

Population= ensemble d'individus

Individu= chromosone=ensemble de gènes (0 ou 1) exemple 100101

**Fitness ou fonction d’évaluation:**
**Dans un AG on essaie toujours à maximiser la fitness**
**Problème de minimisation Fitness=1/F(X)**

**Exemple Minimiser une fonction:**

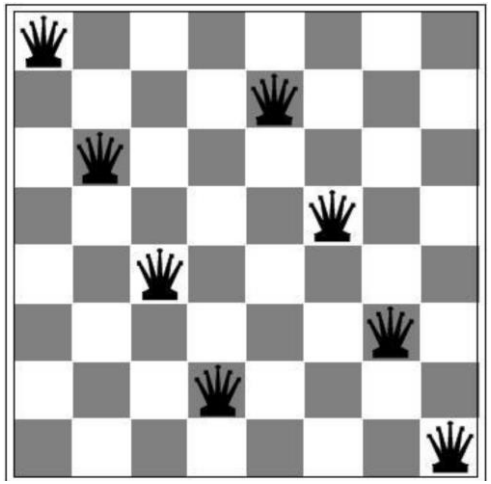
# Initialisation

On génère **aléatoirement** k individus(états) qui représente la population initiale

Exemples: Le problème des 8-reines

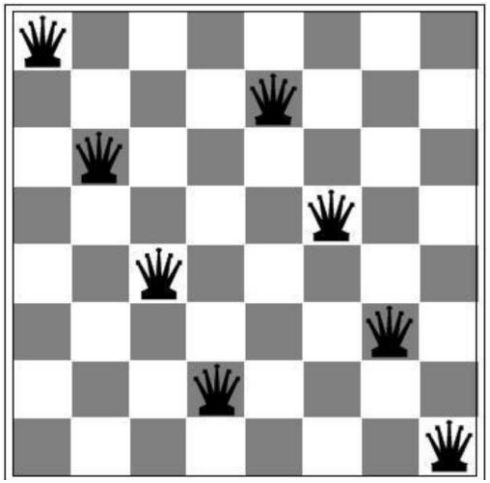
**Comment peut-on placer les reines sans qu'elles ne s'attaquent entre elles**

***deux dames ne devraient jamais partager la même rangée, colonne, ou diagonale***



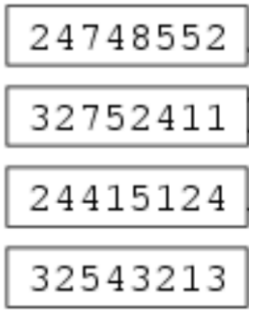
Comment modéliser une solution dans ce cas

Indice de la renne dans la colonne

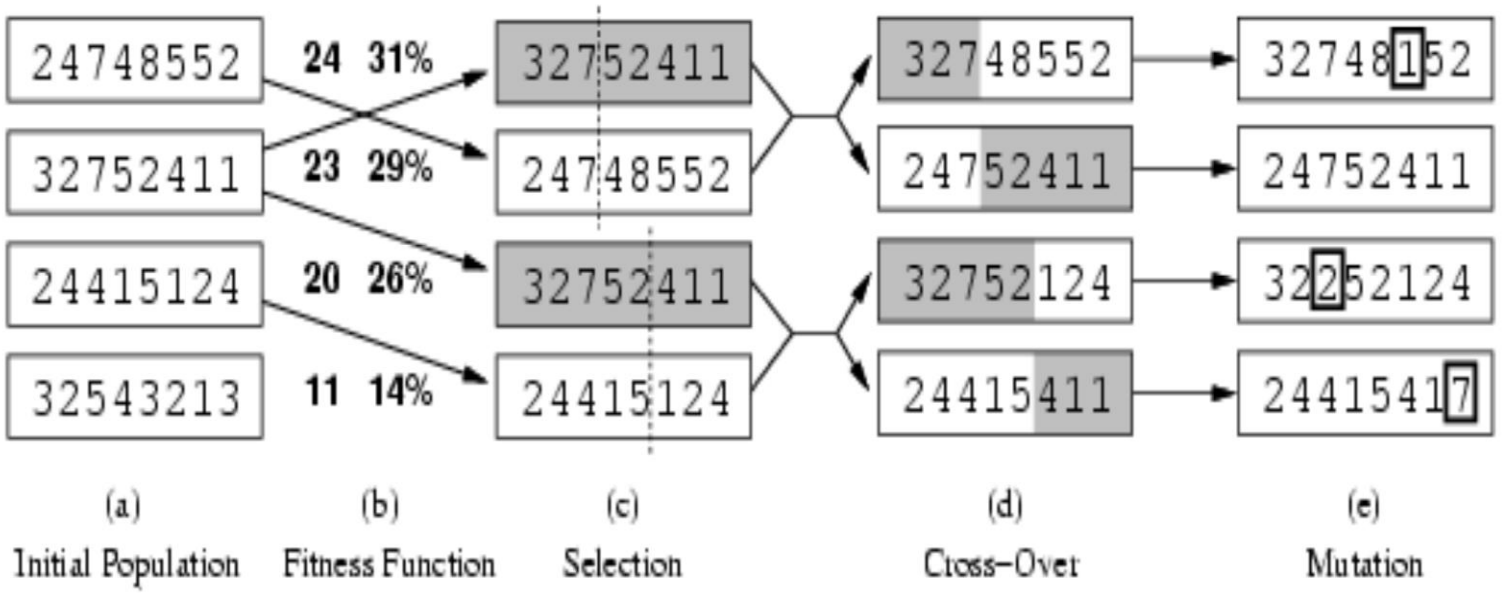


**En code décimal: 13572468**

Initialisation:




Comment évaluer une solution (fonction de fitness):
**Fitness function**: le nombre de couples de reines qui ne sont pas en conflit (min = 0, max = 8 × 7/2 = 28)


24748552
32752411
24415124
32543213
24 31%
23 29%
20 26%
11 14%
32752411
24748552
32752411
24415124
32748552
24752411
32752124
24415411
32748152
24752411
32252124
24415417
(a)
Initial Population
(b)
Fitness Function
(c)
Selection
(d)
Cross-Over
(e)
Mutation

# Algorithme génétique:

**function** GENETIC-ALGORITHM( *population*, FITNESS-FN) **returns** an individual
  **inputs**: *population*, a set of individuals
    FITNESS-FN, a function that measures the fitness of an individual

  **repeat**
    *new_population* ← empty set
    **for** $i$ = 1 **to** SIZE( *population*) **do**
      $x$ ← RANDOM-SELECTION( *population*, FITNESS-FN)
      $y$ ← RANDOM-SELECTION( *population*, FITNESS-FN)
      *child* ← REPRODUCE($x$, $y$)
      **if** (small random probability) **then** *child* ← MUTATE( *child*)
      add *child* to *new_population*
    *population* ← *new_population*
  **until** some individual is fit enough, or enough time has elapsed
  **return** the best individual in *population*, according to FITNESS-FN

---

**function** REPRODUCE($x$, $y$) **returns** an individual
  **inputs**: $x$, $y$, parent individuals

  $n$ ← LENGTH($x$); $c$ ← random number from 1 to $n$
  **return** APPEND(SUBSTRING($x$, 1, $c$), SUBSTRING($y$, $c + 1$, $n$))

# **DEAP**: Distributed Evolutionary Algorithms in Python

Lancer Anaconda Prompt:

```
pip  install  deap
```

Exemple d’application: **One Max Problem**

```
from deap import base
from deap import creator
from deap import tools
```

**Creator** pour créer les individus (type des solutions)
**Tools**: Initialisation des individus
**Base**: pour registration des différents opérateurs( mutation selection,…)

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("FitnessMax", base.Fitness, weights=(+1.0,))
```

Tutorial: DEAP documentation — DEAP 1.3.3 documentation

# Essaim de particules (PSO): Swarm Intelligence

**Swarm Intelligence** (SI) ou **intelligence en essaim de particules**: Intelligence collective

**Coopération pour la résolution de problèmes complexes**

Dans un système PSO, un essaim d'individus parcourt **l'espace de recherche**

Chaque **particule** représente une **solution candidate** au problème d'optimisation

La position d'une particule est influencée par:

- la meilleure position visitée par elle-même (c'est-à-dire sa propre expérience)
- la position de la meilleure particule de son voisinage (c'est-à-dire l'expérience des particules voisines).

Lorsque le voisinage d'une particule est l'essaim en entier, la meilleure position dans le voisinage est considérée comme la meilleure particule, et l'algorithme est appelé le gbest PSO.

Du point de vue de l'évolution, une particule se déplace avec une **vitesse adaptable au sein de l'espace de recherche**

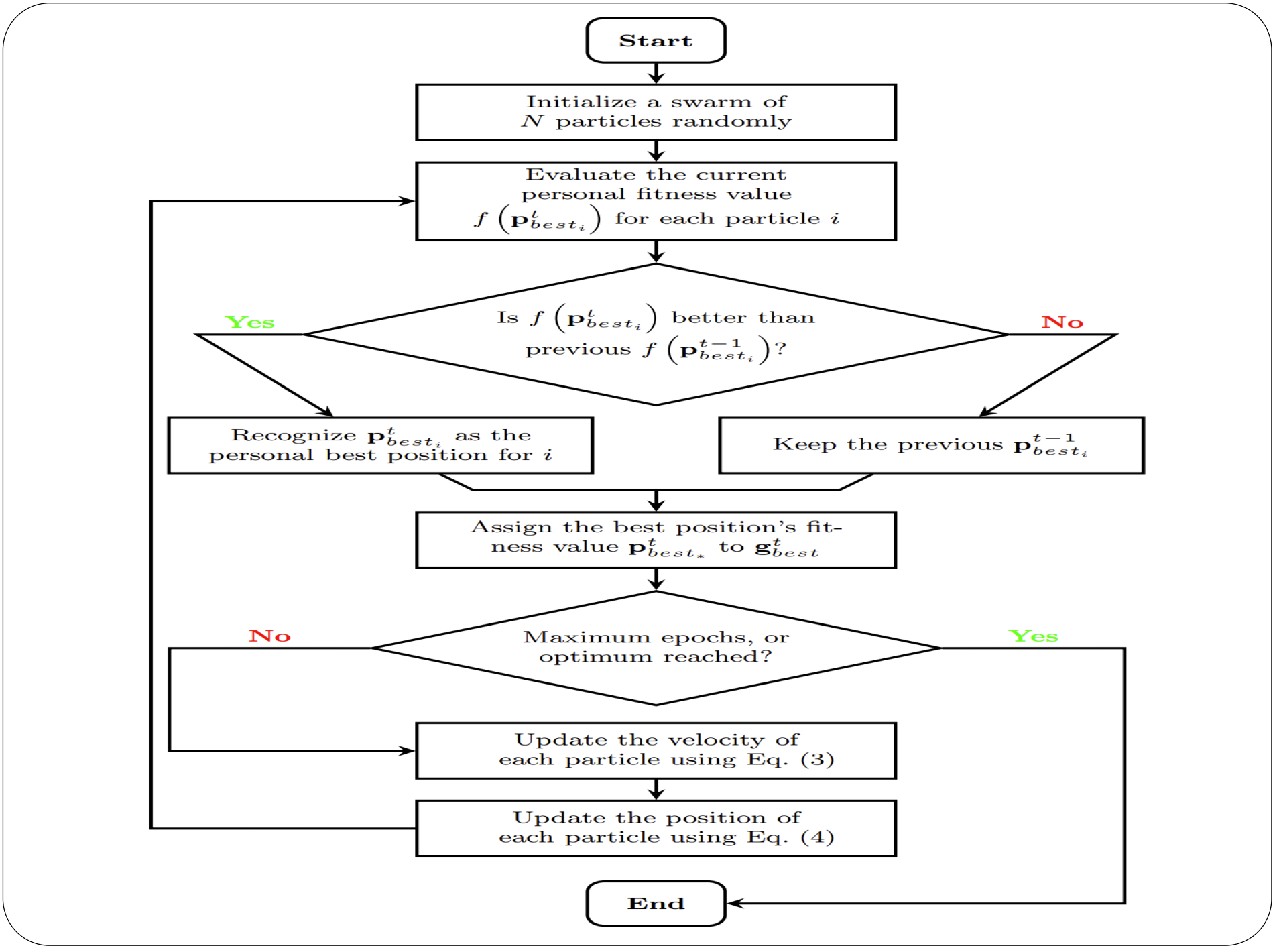
$$\mathbf{p}_{best_i}^{t} = \mathbf{x}_i^* \mid f\left(\mathbf{x}_i^*\right) = \min_{k=1,2,\ldots,t} \left(\left\{f\left(\mathbf{x}_i^k\right)\right\}\right), \tag{1}$$

$$\mathbf{g}_{best}^{t} = \mathbf{x}_*^t \mid f\left(\mathbf{x}_*^t\right) = \min_{\substack{i=1,2,\ldots,N \\ k=1,2,\ldots,t}} \left(\left\{f\left(\mathbf{x}_i^k\right)\right\}\right), \tag{2}$$

$$\mathbf{v}_i^{t+1} = \omega \mathbf{v}_i^t + c_1 \mathbf{r_1}\left(\mathbf{p}_{best_i}^t - \mathbf{x}_i^t\right) + c_2 \mathbf{r_2}\left(\mathbf{g}_{best}^t - \mathbf{x}_i^t\right), \tag{3}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}, \tag{4}$$

i: indice de la particule
t: itération courante
*f*: fonction objectif à optimiser
X: position (solution)
N: Nombre totale de particules
V: vitesse de la particule
W: paramètre poid (balance entre exploration et exploitation)
r1 et r2 vect aléatoires entre 0 et 1
C1 et c2 coefficients d'accélération (constantes positives)


Start
Initialize a swarm of $N$ particles randomly
Evaluate the current personal fitness value $f\left(\mathbf{p}_{best_i}^t\right)$ for each particle $i$
Is $f\left(\mathbf{p}_{best_i}^t\right)$ better than previous $f\left(\mathbf{p}_{best_i}^{t-1}\right)$?
Yes
No
Recognize $\mathbf{p}_{best_i}^t$ as the personal best position for $i$
Keep the previous $\mathbf{p}_{best_i}^{t-1}$
Assign the best position's fitness value $\mathbf{p}_{best_*}^t$ to $\mathbf{g}_{best}^t$
Maximum epochs, or optimum reached?
No
Yes
Update the velocity of each particle using Eq. (3)
Update the position of each particle using Eq. (4)
End

**Algorithm** PSO pseudocode

**Input:**

$N$ – Swarm size
$D$ – Problem dimensionality
$T$ – Maximum number of iterations
$LB$ – Lower bound of the search space
$UB$ – Upper bound of the search space

**Output:**

$\mathbf{g}^t_{best}$ – the best position (solution) found so far

**Start**
  Initialize the swarm randomly;
  **for** $i = 1$ to $N$ **do** ▷ Iterate through the swarm
    $\mathbf{v}^0_i \leftarrow$ a random vector within $[LB, UB]^D$; ▷ Initialize particles' velocity using a uniform distribution
    $\mathbf{x}^0_i \leftarrow$ a random vector within $[LB, UB]^D$; ▷ Initialize particles' positions using a uniform distribution
    $\mathbf{p}^0_{best_i} \leftarrow \mathbf{x}^0_i$; ▷ Initialize $\mathbf{p}_{best}$ to its initial position
  **end for**
  Apply Eq. (2) to find $\mathbf{g}^0_{best}$; ▷ Initialize $\mathbf{g}_{best}$ to position with the minimum fitness value
  $t \leftarrow 1$; ▷ Initialize first iteration number
  **while** $t \leq T$ **do**
    **for** $i = 1$ to $N$ **do** ▷ Iterate through the swarm
      $\boldsymbol{r_1}, \boldsymbol{r_2} \leftarrow$ two independent vectors randomly generated from $[0, 1]^D$;
      Apply Eq. (3); ▷ Update particle's velocity
      Apply Eq. (4); ▷ Update particle's position
      **if** $f\left(\mathbf{x}^t_i\right) < f\left(\mathbf{p}^{t-1}_{best_i}\right)$ **then** ▷ If new solution is better than current personal best
        $f\left(\mathbf{p}^t_{best_i}\right) \leftarrow f\left(\mathbf{x}^t_i\right)$; ▷ Update the best known position of the particle
      **end if**
    **end for**
    Apply Eq. (2) to find $\mathbf{g}^t_{best}$; ▷ Update the swarm's overall best known position
    $t \leftarrow t + 1$;
  **end while** ▷ Maximum iteration number is reached or termination criterion is satisfied
**End**

Particle Swarm Optimization Basics — DEAP 1.3.3 documentation

https://deap.readthedocs.io/en/master/examples/pso_basic.html

```
import operator
import random
import numpy
import math
from deap import base
from deap import benchmarks
from deap import creator
```

L’objectif de la particule est de maximizer la function objectif

```
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Particle",, fitness=creator.FitnessMax, speed=
, smin=None, smax=None, best=None)
```

**Operators:**

Trois opérateur sont utilisées: ***initializer, updater and evaluator***

```
def generate(size, pmin, pmax, smin, smax):
    part = creator.Particle(random.uniform(pmin, pmax) for _ in (size))
    part.speed = [random.uniform(smin, smax) for _ in(size)]
    part.smin = smin part.smax = smax
    return part
```

**La fonction: updateParticle() calcule d'abord la vitesse et la limite entre smin et smax et finalement calcule la nouvelle position de la particule**

```
def updateParticle(part, best, phi1, phi2):
u1 = (random.uniform(0, phi1) for _ in ((part)))
u2 = (random.uniform(0, phi2) for _ in((part)))
v_u1 =(operator.mul, u1,(operator.sub, part.best, part))
v_u2 =(operator.mul, u2(operator.sub, best, part))
 part.speed =((operator.add, part.speed,(operator.add, v_u1, v_u2)))
 for i, speed in part.speed):
   if(speed) < part.smin:
      part.speed[i] = math.copysign(part.smin, speed)
   elif (speed) > part.smax:
         part.speed[i] = math.copysign(part.smax, speed)
  part[:] =((operator.add, part, part.speed))
```

```
toolbox = base.Toolbox()
toolbox.register("particle", generate, size=2, pmin=-6, pmax=6, smin=-3, smax=3)
toolbox.register("population", tools.initRepeat, list, toolbox.particle)
toolbox.register("update", updateParticle, phi1=2.0, phi2=2.0)
toolbox.register("evaluate", benchmarks.h1)
```

```
def main():
    pop = toolbox.population(n=5)
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", numpy.mean)
    stats.register("std", numpy.std)
    stats.register("min", numpy.min)
    stats.register("max", numpy.max)
    logbook = tools.Logbook()
    logbook.header = ["gen", "evals"] + stats.fields
    GEN = 1000
    best = None
    for g in range(GEN):
        for part in pop:
            part.fitness.values = toolbox.evaluate(part)
            if not part.best or part.best.fitness < part.fitness:
                part.best = creator.Particle(part)
                part.best.fitness.values = part.fitness.values
            if not best or best.fitness < part.fitness:
                best = creator.Particle(part)
                best.fitness.values = part.fitness.values
        for part in pop:
            toolbox.update(part, best)

        # Gather all the fitnesses in one list and print the stats
        logbook.record(gen=g, evals=len(pop), **stats.compile(pop))
        print(logbook.stream)
 return pop, logbook, best
```